**UNIVERSITE DE NICE-SOPHIA ANTIPOLIS - UFR SCIENCES**

Ecole Doctorale de Sciences et Technologies de l'Information et de la Communication

T H E S E

pour obtenir le titre de

Docteur en Sciences de l'Universite de Nice-Sophia Antipolis

Discipline : Informatique

présentée et soutenue par

**Matti SIEKKINEN**

TITRE

# Root Cause Analysis of TCP Throughput: Methodology, Techniques, and Applications

Thèse dirigée par Ernst BIERSACK

Soutenue publiquement le 30.10. 2006 devant le jury composé de

| | | |
|---|---|---|
| Dr. | Walid DABBOUS | président |
| Professeur Dr. | Georg CARLE | rapporteur |
| Dr. | Tijani CHAHED | rapporteur |
| Professeur Dr. | Ernst W. BIERSACK | directeur de thèse |
| Dr. | Guillaume URVOY-KELLER | co-directeur de thèse |

# Acknowledgments

First of all, I would like to express my deepest gratitude to my thesis advisor Prof. Dr. Ernst W. Biersack. He was the one who suggested me to pursue a Ph.D. thesis before I even imagined that I would have the capabilities to do that. Looking back now, the decision to accept this proposal is among the best I have made in my life. You really taught me a lot and your door was always open, thank you Ernst.

I am also grateful to my thesis co-advisor Dr. Guillaume Urvoy-Keller. Guillaume contributed enormously to this thesis. I could always go to him when I had a problem to crack, was unsure of my own reasoning, or lacked faith in what I was doing. Thank you Guillaume for all the long and fruitful discussions. I am also grateful to Prof. Dr. Vera Goebel who taught me among other things to better understand database and operating systems. Vera's contribution to the thesis was essential and such that I could not have acquired from the "networking people".

I would like to thank Dr. Taoufik En-Najjary for his major contribution by the development of the PPrate tool and for all the interesting discussions we had. If I had a problem involving mathematics, I knew Taoufik was the person to turn to. I also would like to thank Prof. Dr. Thomas Plagemann whom I had the opportunity to work with in the early phases of the thesis.

I also want to thank colleagues at Eurecom, especially my good friends Walid Bagga, Luca Brayda, and Federico Matta with whom we go back a long way, and office mates Fabien Pouget, Suna Melek Önen, Corrado Leita, and Van Hau Pham, and Jérôme Härri (who I consider as an office mate even though he sat in another office) for their friendship and good times. I also owe thanks to all the other people in Corporate Communications Department and Institut Eurécom that contributed to the success of this thesis.

Last but not least, I wish to dedicate a very special thanks to my family for the endless support and help they provided me with, and close friends in Finland who were there for me. Kiitos.

# Abstract

The interest for the research community to measure the Internet has grown tremendously during the last couple of years. This increase of interest is largely due to the growth and expansion of the Internet that has been overwhelming. We have experienced exponential growth in terms of traffic volumes and number of devices connected to the Internet. In addition, the heterogeneity of the Internet is constantly increasing: we observe more and more different devices with different communication needs residing in or moving between different types of networks. This evolution has brought up many needs – commercial, social, and technical needs – to know more about the users, traffic, and devices connected to the Internet. Unfortunately, little such knowledge is available today and more is required every day. That is why Internet measurements has grown to become a substantial research domain today.

This thesis is concerned with TCP traffic. TCP is estimated to carry over 90% of the Internet's traffic, which is why it plays a crucial role in the functioning of the entire Internet. The most important performance metrics for applications is typically throughput, i.e. the amount of data transmitted over a period of time. Our definition of the root cause analysis of TCP throughput is the analysis and inference of the reasons that prevent a given TCP connection from achieving a higher throughput. These reasons can be many: application, network, or even the TCP protocol itself.

This thesis comprises three parts: methodology, techniques, and applications. The first part introduces our database management system-based methodology for passive traffic analysis. In that part we explain our approach, the InTraBase, which is based on an object-relational database management system. We also describe our prototype of this approach, which is implemented on PostgreSQL, and evaluate and optimize its performance.

In the second part, we present the primary contributions of this thesis: the techniques for root cause analysis of TCP throughput. We introduce the different potential causes that can prevent a given TCP connection to achieve a higher throughput and explain in detail the algorithms we developed and used to detect such causes. Given the large heterogeneity and potentially large impact of applications that operate on top of TCP, we emphasize their analysis.

The core of the third part of this thesis is a case study of traffic originating from clients of a commercial ADSL access network. The study focuses on performance analysis of data transfers from a point of view of the client. We discover some surprising results, such as poor overall performance of P2P applications for file distribution due to upload rate limits enforced by client applications. The third part essentially binds the two first ones together: we give an idea of the capabilities of a system combining the methodology of the first part with the techniques of the second part to produce meaningful results in a real world case study.

# Résumé

L'intérêt pour la métrologie de l'Internet s'est beaucoup accru ces dernières années. Ceci est en grande partie dû à la croissance de l'Internet en termes de volumes de trafic et de nombre de machines reliés à l'Internet. Cette évolution a sucité beaucoup d'envies - du point de vue commercial, social, et technique - d'en savoir plus au sujet des utilisateurs et du trafic Internet en général. Malheureusement, il y a peu de connaissances de ce type disponibles aujourd'hui. C'est pourquoi la métrologie de l'Internet est devenue un domaine substantiel de recherches.

Cette thèse porte sur l'analyse du trafic TCP. On estime que TCP transporte 90% du trafic Internet, ce qui implique que TCP est une pièce essentielle dans le fonctionnement de l'Internet. La métrique de performance la plus importante pour les applications est, dans la plupart des cas le débit de transmission ; c'est-à-dire la quantité des données transmises par périodes de temps. Notre objectif est l'analyse du débit de transmission de TCP et l'identification des raisons qui empêchent une connexion TCP d'obtenir un débit plus élevé. Ces raisons peuvent être multiples : l'application, le réseau, ou même le protocole TCP lui-même.

Cette thèse comporte trois parties. Une première partie sur la méthodologie, une seconde sur techniques d'analyse de TCP, et une dernière qui est une application de ces technique. Dans la première partie, nous présentons notre méthodologie basée sur un système de gestion de base de données (DBMS) pour l'analyse passive de trafic. Nous expliquons notre approche, nommée InTraBase, qui est basée sur un système de gestion de base de données objet-relationelle. Nous décrivons également notre prototype de cette approche, qui est implémenté au dessus de PostgreSQL, et nous évaluons et optimisons ses performances.

Dans la deuxième partie, nous présentons les contributions principales de cette thèse : les techniques d'analyse des causes du débit de transmission TCP observé. Nous présentons les différentes causes potentielles qui peuvent empêcher une connexion TCP d'obtenir un débit plus élevé et nous expliquons en détail les algorithmes que nous avons développé pour détecter ces causes. Etant donné leur hétérogénéité et leur impact sur le débit TCP, nous accordons une grande importance aux applications au dessus de TCP.

La troisième partie de cette thèse est une étude de cas du trafic des clients d'un réseau d'accès commercial d'ADSL. L'étude se concentre sur l'analyse des performances des transferts de données d'un point de vue client. Nous démontrons quelques résultats étonnants, tel le fait que les performances globalement faibles des applications pair-à-pair sont dues aux limitations du débit de transmission imposées par ces applications (et non à la congestion dans le réseau).

# CONTENTS

**1. Introduction** **1**
1.1. The Internet: Measurement Target in Constant Motion . . . . . 1
1.2. Root Cause Analysis of TCP Traffic: What and Why? . . . . . 2
1.3. Thesis Claims and Structure . . . . . 3

**Part I Methodology: Manageable Approach for Passive Traffic Analysis** **5**

**Overview of Part I** **7**

**2. Measuring the Internet** **9**
2.1. Setting the Measurement Context . . . . . 9
2.1.1. Passive and Active Measurements . . . . . 9
2.1.2. Reducing Passive Measurement Data . . . . . 10
2.2. Analysis of Passive Measurements . . . . . 11
2.2.1. Challenges . . . . . 11
2.2.1.1. Management . . . . . 12
2.2.1.2. Analysis Cycle . . . . . 12
2.2.1.3. Scalability . . . . . 13
2.2.2. Database Systems to the Rescue? . . . . . 14
2.2.3. Existing Approaches . . . . . 14
2.3. Conclusions . . . . . 16

**3. InTraBase: Integrated Traffic Analysis Based on Object-Relational DBMS** **17**
3.1. Approach . . . . . 17
3.1.1. IntraBase and Other Approaches . . . . . 17
3.1.2. Fully Integrated Solution Based on Object-Relational DBMS . . . . . 18
3.1.3. Benefits From Our Approach . . . . . 19
3.1.3.1. DBMS Is All About Management . . . . . 19
3.1.3.2. Shorter Analysis Process Cycle . . . . . 20

3.1.3.3. Improved Scalability . . . 20
3.2. PgInTraBase: Prototype Implementation of InTraBase . . . 21
3.2.1. Database Schema . . . 21
3.2.2. Processing a Trace: Populating Tables . . . 22
3.2.3. Analyzing Processed Data . . . 23
3.2.4. Properties of PgInTraBase . . . 23
3.3. Conclusions . . . 24

**4. Evaluation and Optimization of the InTraBase 25**
4.1. Evaluation of the Prototype . . . 25
4.1.1. Feasibility of PgInTraBase . . . 25
4.1.1.1. Processing Time of the Initial Steps . . . 26
4.1.1.2. Disk Space Consumption . . . 27
4.1.2. Comparison of InTraBase and Tcptrace . . . 28
4.2. Optimizing the DBS for Efficient Analysis . . . 29
4.2.1. Tuning the DBMS . . . 31
4.2.2. Identifying and Decomposing the Typical Analysis Task . . . 32
4.2.3. Cost Minimization of the Typical Analysis Task . . . 33
4.2.3.1. Indexes for Fast Lookup . . . 33
4.2.3.2. Clustering to Minimize Cost of I/O Reads . . . 34
4.2.3.3. Parallel I/O . . . 36
4.2.3.4. Caching . . . 37
4.3. Evaluation of the Impact of Optimization . . . 37
4.4. Conclusions . . . 38

**Conclusions for Part I 39**

**Part II Root Cause Analysis of TCP Traffic 41**

**Overview of Part II 43**

**5. Origins of TCP Transfer Rates 45**
5.1. TCP . . . 45
5.1.1. Connection Establishment and Tear Down . . . 46
5.1.2. Error Control: Cumulative Acknowledgments and Timeouts . . . 46
5.1.3. Flow Control: Sliding Window Technique . . . 47
5.1.4. Congestion Control: Resizing the Sliding Window . . . 48
5.1.4.1. Slow Start and Congestion Avoidance . . . 48
5.1.4.2. TCP Tahoe: Fast Retransmit . . . 48
5.1.4.3. TCP Reno: Fast Retransmit & Fast Recovery . . . 48
5.1.4.4. TCP NewReno: Improved Handling of Multiple Losses During Fast Recovery . . . 50
5.1.4.5. Other TCP Versions . . . 50
5.2. What Limits the Transmission Rate of TCP? . . . 50
5.2.1. Application . . . 50

5.2.2. TCP Layer . . . 53
5.2.2.1. TCP End-Point Buffers . . . 53
5.2.2.2. Congestion Avoidance Mechanism: Transport Limitation . . . 55
5.2.2.3. Short Transfers: Slow Start Mechanism . . . 56
5.2.3. Network . . . 56
5.2.4. Middleboxes . . . 60
5.3. Related Work . . . 62
5.3.1. Analytical Work: Modeling TCP . . . 63
5.3.2. Measurement-Based Analysis . . . 63
5.3.2.1. TCP Performance & Deployment Status . . . 63
5.3.2.2. TCP and the Network . . . 64
5.3.2.3. TCP and Applications . . . 65
5.3.3. TCP Extensions and Improvements . . . 66
5.3.4. Root Cause Analysis . . . 66
5.4. Scope of Our Work . . . 67

**6. Applications and Their Interaction with TCP 69**
6.1. Isolate & Merge (IM) Algorithm . . . 70
6.1.1. Context . . . 70
6.1.2. Procedures . . . 71
6.2. Validation . . . 71
6.3. Data Sets . . . 73
6.4. Distortion Due to ALPs on End-to-end Path Studies . . . 74
6.4.1. Studying Characteristics of Rates . . . 74
6.4.2. Case Study on RTT Estimation . . . 78
6.5. What Can We Learn From The Different Periods? . . . 80
6.5.1. Properties of the BTPs Identified . . . 80
6.5.2. Discovering the Nature of an Application . . . 81
6.6. Conclusions . . . 83

**7. Analysis of TCP Bulk Transfers 85**
7.1. Quantitative Analysis: The Limitation Scores . . . 85
7.1.1. Determining Position of Measurement Point . . . 86
7.1.2. Metrics Inferred from Packet Headers . . . 87
7.1.3. Limitation Scores . . . 88
7.1.4. Validations . . . 92
7.1.5. Sources of Errors and Inaccuracy . . . 94
7.2. Interpreting the Limitation Scores: the Classification Scheme . . . 95
7.2.1. Scores and Thresholds . . . 95
7.2.2. Accounting for Middleboxes . . . 97
7.3. Inferring the Threshold Values . . . 98
7.3.1. Experimentation Setup . . . 98
7.3.2. Threshold for Retransmission Score . . . 100
7.3.3. Threshold for Receiver Window Limitation Score . . . 100
7.3.4. Threshold for Dispersion Score . . . 100
7.3.5. Threshold for B-Score . . . 100

7.3.6. Root Cause Classification Results for the Experiments . . . 105
7.3.7. Critical Discussion of Our Approach . . . 108
7.4. T-RAT . . . 109
7.4.1. On the Flight Nature of TCP . . . 109
7.4.2. Comparison With Our Methods . . . 112
7.4.2.1. Unshared Bottleneck Link/Bandwidth Limitation . . . 113
7.4.2.2. Shared Bottleneck Link/Congestion Limitation . . . 114
7.4.2.3. Receiver Limitation . . . 114
7.4.2.4. Application Limitation . . . 116

**Conclusions for Part II** **119**

**Part III Real World Case Study on TCP Root Cause Analysis Using InTraBase** **121**

**Overview of Part III** **123**

**8. Adapting InTraBase for TCP Root Cause Analysis** **125**
8.1. Extended Design of InTraBase for Root Cause Analysis . . . 125
8.1.1. Table Layout . . . 125
8.1.2. Indexes . . . 126
8.2. Root Cause Analysis Functions . . . 127
8.2.1. Populating Root Cause Analysis Tables . . . 127
8.2.2. Going Further With Triggers . . . 128

**9. Case Study on Performance Analysis of ADSL Clients** **129**
9.1. Monitoring the ADSL Access Network of France Telecom . . . 130
9.1.1. Architecture and Setup . . . 130
9.1.2. Main Constraints and Challenges . . . 130
9.2. Traffic Characteristics: Applications, Connections, and Clients . . . 131
9.2.1. General Characteristics of the Traffic . . . 131
9.2.1.1. Traffic per Application . . . 131
9.2.1.2. Traffic per Connection . . . 132
9.2.2. Client Behavior . . . 133
9.2.2.1. Volumes and Applications . . . 133
9.2.2.2. Access Link Utilization . . . 135
9.3. Performance Analysis of Clients . . . 137
9.3.1. Taxonomy of Factors Limiting the Performance of Clients . . . 137
9.3.2. Observed Limiting Factors for Clients . . . 140
9.3.2.1. Main Limitation . . . 141
9.3.2.2. Limitations Experienced . . . 141
9.3.3. Throughput limitation causes experienced by major applications . . . 142
9.3.4. Impact of Limiting Factors On Performance . . . 143
9.3.5. Comparison With Other Related Analysis Work . . . 147
9.4. Closer Look at a Few Example Clients . . . 147

9.5. Conclusions . . . 152

**Conclusions for Part III** **153**

---

**10. Thesis Conclusions** **155**
10.1. Evaluation of the Thesis Work . . . 155
10.1.1. Claims and Contributions . . . 155
10.1.2. Critical Viewpoint . . . 157
10.2. Future Work . . . 157
10.2.1. InTraBase . . . 157
10.2.2. Root Cause Analysis of TCP Throughput . . . 158

**Bibliography** **161**

**Appendix** **169**

**A. List of Abbreviations, Acronyms, and Parameters** **171**

**B. Detailed Analysis of PgInTraBase Performance Measurements** **173**
B.1. Impact of Indexing and Clustering . . . 173
B.2. Measuring the Effectiveness of Caching . . . 175
B.3. The Impact of Parallel I/O: RAID Striping . . . 176
B.4. DBMS as the Final Bottleneck . . . 176

**C. Descriptions of Isolate & Merge Algorithms** **179**
C.1. Isolate . . . 179
C.2. Merge . . . 180

**D. Formal Definitions for Computed Metrics** **183**
D.1. Inter-arrival Times of Acknowledgments . . . 183
D.2. Round-trip Time . . . 183
D.3. Receiver Advertised Window . . . 184
D.4. Outstanding Bytes . . . 184
D.5. Retransmissions . . . 185

**E. Résumé de la Thèse** **187**
E.1. Introduction . . . 187
E.1.1. Internet : une évolution continuelle . . . 187
E.1.2. Analyse des causes du débit de transmission de TCP . . . 188
E.1.3. Contributions de la thèse . . . 189
E.2. Résumé des Trois Parties de la Thèse . . . 190
E.2.1. Première Partie : Méthodologie . . . 190
E.2.1.1. Métrologie de l'Internet . . . 190
E.2.1.2. InTraBase : Analyse de trafic intégrée et basée sur un système de gestion de base de données relationnelle orientée objet . . . 190

E.2.1.3. Évaluation et optimisation de l'InTraBase . . . . . 191
E.2.2. Partie 2 : Analyse des Causes du Débit de Transmission TCP . . . . . 193
E.2.2.1. Causes de limitation des transferts de TCP . . . . . 193
E.2.2.2. Identification des causes de limitation . . . . . 194
E.2.3. Partie 3 : Etude de Cas sur l'Analyse du Trafic d'un Réseau d'Accès d'ADSL 197
E.2.3.1. Adaptation de l'InTraBase pour l'analyse des causes de débit de transmission de TCP . . . . . 198
E.2.3.2. Étude de cas sur des limitations de performance des clients d'ADSL 198
E.3. Conclusions . . . . . 201
E.4. Contributions . . . . . 201
E.5. Perspectives . . . . . 202

# LIST OF FIGURES


1.1. The Internet protocol suite. . . . . 2

2.1. Typical cycle of tasks for the iterative process for off-line traffic analysis. . . . . 13

3.1. High-level Architecture of the DBS. . . . . 18
3.2. Integrated data and tool management. . . . . 19
3.3. Cycles of tasks for the iterative process for off-line traffic analysis. . . . . 20
3.4. The layouts of core tables in PgInTraBase after the 5 processing steps. Underlined attributes form a key that is unique for each row. . . . . 22

4.1. Total processing time of the three steps vs. `tcpdump` file size. . . . . 27
4.2. Processing times of different steps with respect to trace file size. . . . . 27
4.3. Disk space usage for different `tcpdump` file sizes containing bittorrent traffic. . . . 28
4.4. Comparison of Per-Connection Statistics from `tcptrace` and InTraBase. . . . . 30
4.5. Executing a typical analysis task. . . . . 32
4.6. The layouts of core tables with indexes. Numbers in parenthesis indicate the different indexes. . . . . 34
4.7. The effect of clustering a single connection within two different types of traffic traces of the same size. Black stripes are packets belonging to the connection that is being clustered and their horizontal distance from each other reflects the physical distance on the disk. . . . . 35
4.8. Elapsed time to index different sizes and types of traces. . . . . 36
4.9. Elapsed time to cluster different sizes and types of traces. . . . . 36
4.10. Raid striping over three disks, i.e. RAID level 0. . . . . 36

5.1. Establishing a connection using the three-way handshake. . . . . 46
5.2. Sender's window slides. . . . . 47
5.3. Evolution of the `cwnd` size during slow start (SS) and congestion avoidance (CA). 49
5.4. Evolution of the congestion window if Fast Recovery is used. . . . . 49
5.5. Data flow from the sender to the receiver application through a single TCP connection. . . . . 51
5.6. A short piece of Skype connection. . . . . 52

5.7. 20 minutes of a BitTorrent connection. . . . . 53
5.8. Entire FTP download connection. . . . . 53
5.9. A piece of a receiver window limited connection. . . . . 55
5.10. A transport limited bulk transfer period within a long BitTorrent connection. . . 56
5.11. Link utilization along a TCP/IP path where the narrow link is the same as the tight link. . . . . 58
5.12. Link utilization along a TCP/IP path where the narrow link is not the same as the tight link. . . . . 58
5.13. A piece of a bandwidth limited transfer where packets are regularly spaced due to the bottleneck link. . . . . 58
5.14. A piece of transfer limited by a shared bottleneck link. . . . . 59
5.15. Effect of consecutive losses within a BTP of a long BitTorrent connection. . . . . 59
5.16. Transfer to a wireless laptop on board of an airplane. . . . . 61
5.17. Transfer passing through a Packeteer packet shaper. . . . . 62
5.18. Round-trip time estimation in the middle of the path. . . . . 65

6.1. Successful merger. . . . . 71
6.2. Failed merger. . . . . 71
6.3. CDF of $diff$, the fraction of matching samples, for the periods. . . . . 72
6.4. Q-Q plot of throughput for the BitTorrent BTPs having $drop = 0.9$. . . . . 75
6.5. Q-Q plot of throughput for the BitTorrent connections transferring at least 100KB. 75
6.6. Throughput of the common connections in the sets of 50 fastest connections vs. 50 fastest BTPs ($drop = 0.9$) for BitTorrent. . . . . 77
6.7. Problem with RTT estimation during an ALP. . . . . 78
6.8. CDFs of ratio of the mean RTTs: $\frac{\overline{RTT}_{ALP}}{\overline{RTT}_{BTP}}$. . . . . 79
6.9. Piece of an HTTP connection. Dashed and dotted vertical lines start an ALP and BTP ($drop = 0.95$), respectively. . . . . 79
6.10. Number of identified BTPs vs. $drop$. . . . . 81
6.11. Total number of identified BTPs+STPs vs. $drop$. . . . . 81
6.12. Fraction of all bytes in BTPs vs. $drop$. . . . . 81
6.13. Number of identified BTPs per connection, $drop = 0.9$. . . . . 81
6.14. Rate limited eDonkey connection. . . . . 82
6.15. Rate limited BitTorrent connection. . . . . 82
6.16. CDF plots of duration, volume, and throughput ratios. . . . . 83

7.1. Determining the measurement position from the three-way handshake of TCP. . 86
7.2. Time series of outstanding bytes and receiver advertised window for a BitTorrent connection. Values are computed using 10 second time windows. . . . . 89
7.3. CDF plot of receiver window limitation score with threshold $lb \in \{1, 2, 3\}$. . . . . 89
7.4. Time sequence diagram of a receiver window limited transfer. Note the clear bursty IAT pattern. . . . . 90
7.5. Time sequence diagram of a shared bottleneck limited transfer with a high receiver window limitation score. Note the smoothed out IAT pattern. . . . . 90
7.6. Inter-arrival times of receiver window limited transfer. Black rectangles are sent packets and time runs from right to left. . . . . 91

7.7. CDF plots of the two receiver window limitation scores when measuring at the sender side. . . . 93
7.8. CDF plots of the two receiver window limitation scores when measurement point is away from sender. . . . 93
7.9. CDF of the absolute difference between the Web100 and InTraBase's scores for receiver window limitation. . . . 94
7.10. Root cause classification scheme. . . . 96
7.11. Root cause classification scheme with middleboxes taken into consideration. . . . 97
7.12. CDF plots of the dispersion score when downloading a single file at a time. . . . 101
7.13. CDF plots of the burstiness score when downloading multiple files simultaneously through a shared bottleneck link or with added delay. . . . 101
7.14. Difference of CDF plots between experiments with an artificial bottleneck and added delay, results with 100ms are excluded. The best matching threshold is found at 0.25 (vertical line). . . . 101
7.15. B-scores per server and transfer for experiments with 5Mbit/s bottleneck or 500ms added delay. Each marker corresponds to a single transfer: x is with delay, o is with a bottleneck. Y values are b-scores, x values are servers. . . . 102
7.16. B-scores per server and transfer for experiments with 3Mbit/s bottleneck or 500ms added delay. . . . 103
7.17. B-scores per server and transfer for experiments with 10Mbit/s bottleneck or 200ms added delay. . . . 104
7.18. B-scores per server and transfer for experiments with 1Mbit/s bottleneck or 400ms added delay. . . . 105
7.19. Classification of BTPs into clear root causes. . . . 106
7.20. Root cause classification of the three data sets with only a single download at a time. . . . 107
7.21. Root cause classification of the three data sets with ten parallel downloads. . . . 107
7.22. Root cause classification of the three data sets with three parallel downloads and added delay. . . . 108
7.23. Simulation Configurations. . . . 110
7.24. Histograms of inter-arrival times of packets. . . . 111
7.25. Evolution of the PDF of the inter-arrival times of packets from a receiver window limited connection without and with cross traffic. . . . 111
7.26. T-RAT's classification by limitation cause for traffic from unshared bottleneck link experiments. . . . 113
7.27. T-RAT's classification by limitation cause for traffic from shared bottleneck link experiments. . . . 115
7.28. RTT evolution of an example transfer over an ADSL access link with a particularly deep buffer. . . . 115
7.29. T-RAT's classification by limitation cause for traffic from receiver limited experiments. . . . 116
7.30. T-RAT's classification by limitation cause for eMule traffic limited by the application. . . . 117
7.31. Piece of an application limited eMule transfer. . . . 118

8.1. Table layouts of intrabase adapted for TCP root cause analysis. Underlined attributes form a key that is unique for each row. . . . . 126

9.1. Architecture of the monitored ADSL platform. . . . . 130
9.2. Amount of bytes transferred by different applications during the day. . . . . 132
9.3. CCDF plot of size of connections. Note the logarithmic scale of both axes. . . . . 133
9.4. Cumulative fraction of all bytes as a function of the connection size. . . . . 133
9.5. CCDF plot of bytes transferred by clients. . . . . 134
9.6. Cumulative fraction of all bytes transferred as a function of bytes transferred by a given client. . . . . 134
9.7. Amount of bytes transferred by client vs. time that client is active. . . . . 134
9.8. CDF plot of upper bound for link utilization per client for a 30min period: mean throughput divided by maximum instantaneous throughput. For each client, we selected the period during which that client achieved maximum throughput. . . . 136
9.9. Amount of transferred application limited bytes during the day for most common applications. . . . . 142
9.10. Amount of transmitted bytes through saturated access link by different applications. 143
9.11. Amount of transmitted bytes whose rate is limited by a distant link by different applications. . . . . 144
9.12. CDF plot of access link utilization during ALPs (application) and BTPs limited by different causes. For each client, we considered only traffic of the 30 min period during which that client achieved the highest instantaneous throughput of the day. 145
9.13. CDF plot of maximum aggregate per-host download throughput computed over five second intervals. . . . . 146
9.14. Client A's link utilization per half hour period during the day. . . . . 148
9.15. Three-hour piece of an activity plot for client A that transfers a lot of bytes and is active all day. . . . . 149
9.16. Close up of client A's connections originating likely from Web browsing that cause the peak download rates. . . . . 149
9.17. Activity plot for client B that is active only during an hour and most likely browses the Web. . . . . 149
9.18. Activity plot for client C that transfers a lot of bytes but is active only approximately five hours. . . . . 150
9.19. Client C's link utilization per half hour period during the day. . . . . 151

B.1. Total execution time of the c-query for the Gigabit trace. . . . . 174
B.2. Total execution time of the c-query for the BitTorrent trace. . . . . 174
B.3. CPU iowait time of the c-query for the Gigabit trace. . . . . 175
B.4. CPU iowait time of the c-query for the BitTorrent trace. . . . . 175
B.5. Number of sectors read when executing the c-query for the Gigabit trace. . . . . 175
B.6. Number of sectors read when executing the c-query for the BitTorrent trace. . . 175
B.7. Average execution time of the c-query for the Gigabit trace. . . . . 176
B.8. Average CPU iowait time of the c-query for the Gigabit trace. . . . . 176
B.9. Average execution time of the c-query with and without the `ORDER BY` clause for the BitTorrent trace. . . . . 177

B.10. Average execution times of the original c-query and a modified c-query that only counts packets for the BitTorrent trace. . . . . 177

C.1. Round-trip time estimation from a two-way data transfer. . . . . 180

D.1. Determining the measurement position from the three-way handshake of TCP. This figure appears with detailed explanations in Section 7.1.1 . . . . . 185

E.1. The Internet protocol suite. . . . . 188
E.2. Architecture du système de base de données (DBS). . . . . 191
E.3. Les tables de base utilisées dans PgInTraBase. Les paramètres sous-lignés forment un clef unique pour chaque ligne de la table. . . . . 192
E.4. Le temps total pour traiter une trace de `tcpdump` d'une taille variée. . . . . 193
E.5. Flux de données de l'application émettrice à l'application qui réceptionne par une simple connexion TCP. . . . . 194
E.6. Fusion réussie. . . . . 195
E.7. Fusion échouée. . . . . 195
E.8. Le numéro de séquence en fonction du temps pour un transfert limite par la fenêtre de récepteur avec les b-points hauts. . . . . 196
E.9. Le numéro de séquence en fonction du temps pour un transfert limite par la fenêtre de récepteur avec les b-points bas. . . . . 196
E.10. Le schéma de classification. . . . . 197
E.11. La conception du prototype PgInTraBase avec les tables nécessaires pour l'analyse des causes de débit. . . . . 198
E.12. Les volumes de données transmis par différents applications pendant la journée. . 199

# LIST OF TABLES

2.1. Different measurement approaches to achieve data reduction. Data reduction values are only indicative. . . . . 10
2.2. Characteristics of different approaches for traffic analysis. Traffic volumes are in the order of magnitude. . . . . 14

5.1. Summary of different application types. . . . . 54

6.1. Trace characteristics. . . . . 73
6.2. Mean values of the throughput ratio. . . . . 75
6.3. Coefficients of correlation between log of throughput and log of number of bytes transferred. Only connections transferring at least 100KB were included and $drop = 0.9$ was used when determining the BTPs. . . . . 76

7.1. Selected mirror sites. . . . . 99

9.1. Percentages of clients that transmit most bytes using a specific application. . . . 135
9.2. Percentage of active clients that sustain utilization of their access link above specific thresholds for a given fraction of a 30-minute period. . . . . 137
9.3. Number of active clients limited by different causes. . . . . 141

B.1. Average values of the measurements. . . . . 173

CHAPTER 1

# Introduction

## 1.1 The Internet: Measurement Target in Constant Motion

The Internet, started up as a research project of ARPA (Advanced Research Projects Agency) in the USA back in 1969, has evolved into an immense network connecting hundreds of millions of devices today. Its size is matched only by its diversity: On one hand, the devices connected to the Internet comprise PCs, servers, mobile phones, satellites, PDAs, sensors etc. On the other hand, there is a vast amount of services available today, including radio, television, telephone, videoconferencing, instant messaging, and peer-to-peer (P2P) content distribution in addition to the traditional email and World Wide Web (WWW).

Nevertheless, it is a fact that many questions about the behavior and characteristics of the Internet are open. While we are well aware of the characteristics of the individual building blocks of the Internet, it is the whole system in operation that is in many ways perceived as a "black box". For example, we would like to know what is precisely the size of the Internet, or just a part of it, in terms of connected nodes. As another example, it is non-trivial to find out the structure, i.e. the topology, of a given part of the Internet. We simply do not have many of the required quantitative metrics to answer these questions. There are many reasons for this unfortunate situation. As the authors of [40] point out, the evolution of the Internet has not been a centralized effort. Several parties have contributed to it, many with different objectives. In addition, the Internet is dynamic: devices come and go and new networks emerge.

The purpose of the research domain of Internet measurements is to provide answers to these open, yet important, questions. There are a multitude of reasons to do this. In [40], the authors distinguish three categories of reasons: commercial, social, and technical. From the commercial point of view, measurements are crucial for, e.g. Internet Service Providers (ISP) in order to evaluate and troubleshoot the performance of their clients. An example of a social reason is the need to know client behavior in the emergence of new popular applications. Technical reasons are related to evolution of devices and protocols operated in the Internet. As an example the authors of [40] name router design which depends strongly on the characteristics of the traffic it needs to process, e.g. the flow size distributions.

The Internet is an immense moving target to measure. That is why it is a great challenge to

measure and characterize it. Its dynamicity appears in many flavors: First of all, the Internet is in constant evolution. The set of services available evolve and change all the time, the amounts of users and traffic volumes grow at exponential rates. On one hand, this rapid evolution increases the need for measurements. For instance, in [36] the authors provide evidence of the dramatic impact of the emergence of new popular applications on traffic characteristics and its implications on network capacity engineering. On the other hand, the evolution brings up new issues in measurements: The volumes of measurement data are ever growing, which complicates the analysis process and poses significant storage problems. Second, there is no such thing as a representative snapshot of the Internet, which means that good local metrics today are not necessarily good local metrics tomorrow. Similarly, good local metrics are perhaps never good metrics elsewhere. For example, application traffic mix and user behavior can differ a lot depending on the day for a given network, and they can be completely different between enterprise networks and ADSL (Asymmetric Digital Subscriber Line) access networks.

## 1.2 Root Cause Analysis of TCP Traffic: What and Why?

In order to understand what we mean by TCP root cause analysis and why it is important, we need to review some facts about the way Internet functions. Devices connected to the Internet communicate with each other using a common Internet protocol suite. Figure E.1 shows the stack structure of this suite. Each application hands data to be transferred to the lower layer, the transport layer, which is responsible for end-to-end transportation of the data. Two transport layer protocols form the core of the layer in the current Internet: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The layer below, network layer, consists only of the Internet Protocol (IP) that is used by TCP to package and transmit pieces of data from source to destination.

| | |
|---|---|
| Application | DNS, FTP, HTTP, IMAP, IRC, NNTP, POP3, SMTP, SNMP, SSH, TELNET, BitTorrent, RTP, rlogin, ... |
| Transport | TCP, UDP |
| Network | IP |
| Link | Ethernet, Wi−Fi, Token ring, PPP, SLIP, FDDI, ATM, Frame Relay, SMDS, ... |

Figure 1.1: The Internet protocol suite.

On the highest layer in Figure E.1, the set of applications contributing most to the traffic in the Internet has changed over the last couple of years from WWW and file transfer (FTP) to P2P applications, and new Internet applications such as RSS feeds or PodCast are emerging constantly. In addition, application mix varies significantly between different environments (e.g. enterprise vs. access networks). However, TCP still transports the majority of bytes, typically over 90%. This fact together with the rapid growth of traffic volumes highlight the versatility of

TCP but also raise the question of how TCP and these new applications perform in these new environments. As a consequence, the analysis of TCP as a protocol and of TCP traffic is even more vital than before.

Throughput, defined as the amount of bytes transmitted within a specified interval of time, is typically the most important performance metric of an Internet application. Consider, for instance, a file download using FTP. The faster the download finishes, the better. *Our definition of root cause analysis of TCP traffic is the analysis and inference of the reasons that prevent a given TCP connection from achieving a higher throughput.* We often refer to these reasons as (rate) limitation causes in this thesis. While such an analysis may seem trivial at first sight, it will become clear for the reader that it is far from it due to the variety of ways these different limitation causes may manifest themselves within the TCP traffic and the constraints imposed by the measurement context. Indeed, it is often the case that many metrics that are required for this analysis cannot be directly measured but instead need to be estimated, which complicates significantly the analysis.

Knowledge about the root causes of TCP traffic implies knowledge about the root causes of the vast majority of the Internet's traffic. That is why this knowledge is very powerful and usable in diverse ways. For example, it could be used by ISPs to troubleshoot their access network or the clients' performance within that network, or it could enable evaluation of the operational performance of a deployed Internet application.

## 1.3 Thesis Claims and Structure

We make the following four claims in this thesis:

I. *We can overcome many of the problems in management and suboptimal analysis process cycle in passive packet-level traffic analysis by adopting a Database Management System (DBMS) -based approach.*

   (a) *An implementation of such an approach performs feasibly.*

   (b) *We can significantly improve the performance of such a system with Input/Output (I/O) optimizations based on characteristics of packet-level traffic data and popularity assessment of queries.*

II. *It is possible to infer root causes for TCP throughput from bidirectional packet traces recorded passively in a single measurement point located anywhere on the TCP/IP path (end-point or in the middle). Furthermore, unidirectional traffic traces are insufficient.*

III. *Different Internet applications interact in complex and different ways with TCP. That is why the effects of applications need to be first filtered out whenever studying the characteristics of the underlying TCP/IP end-to-end path.*

IV. *Our TCP root cause analysis methods implemented with our DBMS-based approach for traffic analysis enable:*

   - *performance evaluation of Internet application protocols,*
   - *evaluation of network utilization, and*
   - *identification of certain TCP configuration problems.*

The structure of the thesis follows largely the claims. In addition to this introduction and final conclusions, we divided the contents of this thesis in three parts.

In the first part, we introduce the world of Internet measurements to the reader in more detail, set the scope of our work within this context, and present our DBMS-based methodology for traffic analysis. We address claim I in this part. Chapter 4 is particularly concerned with the subclaims I.a and I.b.

The second part focuses on root cause analysis of TCP throughput. We first explain the details of the TCP protocol, related research work on that domain, and the origins of TCP transmission rates. We then describe our approach and algorithms to infer root causes of TCP traffic. The second part addresses claim II (Chapters 5 to 7) and claim III (Chapter 6).

The third part ties together the first and the second part. We first explain in detail how we use the DBMS-based approach presented in Part I to implement the root cause analysis techniques described in Part II. We then go through a use case on ADSL client performance analysis to address claim IV (Chapter 9).

Finally, in the last concluding chapter (Chapter 10) we revisit the thesis claims and evaluate how well we fulfilled them. We also assess the thesis work in general, i.e. in which parts we succeeded well and which parts could have been done better, and identify several directions of future work related to this thesis.

# Part I

# Methodology: Manageable Approach for Passive Traffic Analysis

# Overview of Part I

In Part I we motivate, describe, and justify our methodology for analyzing traffic measurements, specifically for the root cause analysis of TCP traffic. In Chapter 2, we present the diverse ways in which the Internet can be measured and describe the method we have chosen: passively captured packet headers. We then explain why the analysis of the measurements poses several challenges due to the vast amounts of unstructured measurement data and the multitude of ways it can be processed.

In Chapter 3, we present our DBMS-based approach for analyzing this passive measurement data called the InTraBase (Integrated Traffic Analysis Based on Object Relational DBMS). We explain how it can help to overcome the issues we presented in Chapter 2. We also describe the running prototype of the InTraBase for TCP traffic analysis that we have built.

We demonstrate that the approach is feasible through performance evaluation of the prototype in Chapter 4. Furthermore, we perform a study on optimizing the performance of the prototype and show through measurements that for specific tasks, the optimization phase is vital in order to have good performance.

In summary, this part essentially describes the methodology we used to perform the analysis and obtain the results on root cause analysis of TCP traffic presented in Part II.

# CHAPTER 2

# Measuring the Internet

In this chapter, we briefly describe the different ways of measuring the Internet and explain how our work is positioned in this domain of research. We then enumerate and elaborate on the challenges and issues, and existing approaches and solutions in our chosen measurement context: off-line analysis of passive measurements. Some of the contents of this chapter has been published in [107].

## 2.1 Setting the Measurement Context

### 2.1.1 Passive and Active Measurements

The domain of the Internet measurements is rich in the number of different measurement techniques to choose from. We can identify two different categories of measurement techniques: *active* and *passive* measurements.

Active techniques measure network characteristics by sending probe packets to infer characteristics of the path that the packets follow. Therefore, they are especially suitable for inferring end-to-end properties of a given network path. Active measurements are used for estimating link capacities or available bandwidth [69] [102], computing network coordinates [112], or discovering topology [42], for instance. Simple well-known examples of active measurement tools are `ping` and `traceroute`. A major limitation of active measurement techniques is that they generally require several (at least two) reference locations, measurement points that can coordinate between each other during the measurements. For example, in the context of available bandwidth or capacity estimation, a host sends probe packets and another host receives them and analyzes their inter-spacing to infer the capacity of the network path the probes followed [102].

Passive techniques are used to gather data for analysis of network and traffic characteristics by measuring observed traffic on a given host or a router. Passive measurements can be divided into three categories: SNMP/RMON based measurement, packet monitoring, and flow measurement [61]. Measurements using SNMP/RMON require access to the measured routers' MIBs, which is usually prohibited from the outside of the measured network. In addition, MIBs can provide only status information (e.g. the operational status of the interfaces on a router) or highly aggregated metrics (per-interface counters of bytes and packets inbound and outbound).

Packet monitoring is recording a copy of or only some information about packets passing by the measurement point. Flow measurement is recording aggregate statistics about groups of packets. These packets usually belong to the same TCP connection or sequence of UDP/ICMP packets between same host and port pairs and, in addition, appear close to each other in time. Some of the variety of applications for passive measurements are described in [61]. Examples include diagnosing performance problems and intra-domain route tuning. Note that we do not address here the ways traffic can be generated for passive measurements, which is always specific to the analysis. For instance, one could set up honeypots to attract malicious traffic or simply monitor a university edge aggregate link in order to learn what kind of traffic a large group of students generate. Despite the different analysis objectives, the measurement techniques remain the same for both cases.

This thesis focuses on inferring root causes from traffic observed on a single measurement point. Our objective is to be able to infer these root causes on potentially large set of real traffic in order to learn about and explain the possible existing root causes in the traffic of the current Internet and the way they manifest themselves in the traffic. Therefore, we focus in this thesis on passive measurements and do not address the active measurement techniques further.

### 2.1.2 Reducing Passive Measurement Data

In the context of passive measurements, it is necessary to consider issues related to storing the data and processing it, i.e. performing analysis tasks on the data. The data consists not only of the measurements but also of results of analysis tasks, that we call derived data, and further data derived from already derived data and measurements. The issues in handling the data in the context of passive measurements arise from the potentially huge amount of primarily unstructured measurement data due to the immense volumes of traffic flowing in the Internet today. Storage requirements and processing time are the first to limit the amounts of traffic that can be analyzed in practice. In order to limit the amount of measurement data, several alternatives to full packet measurements exist. Table 2.1 summarizes the different options. As usual, each choice has advantages and drawbacks, and the choice depends on the tradeoff between the level of detail and the amount of data.

Table 2.1: Different measurement approaches to achieve data reduction. Data reduction values are only indicative.

| **measured data** | **data reduction** | **advantages** | **drawbacks** |
|---|---|---|---|
| full packets | none | have it all | a lot of data, privacy concerns |
| packet headers | around 1/20 (70 B hdr vs. 1.5 KB pkt) | have most of knowledge in summarized format | still a lot of data |
| flows | 1/avg(flow size in pkts) x 1/20 | data reduction, feasible on-line (Cisco's Netflow) | loose packet details, connections need to be reconstructed |
| sampled headers/flows | depends completely on the scheme | improved data reduction | not usable for all types of analysis e.g. loss estimation |

In many situations, the payloads of packets, i.e. the actual data transmitted by the application and, hence, the main and largest component of traffic, are not necessary for the analysis. Moreover, packet payloads may contain privacy sensitive information about the user. Because of this, publicly available packet traces generally either do not contain the payloads or have scrambled payloads. For these reasons, many analysis approaches focus only on the packet headers.

Flow-level measurements produce an order of magnitude less data than packet-level measurements but have the drawback of loosing the packet-level details. Measured aggregates are usually flows defined as group of packets sharing the same five-tuple (source and destination IP addresses and port numbers and transport protocol number) with specific timeouts, e.g. Cisco's Netflow, a specific flow record type supported by Cisco's routers, has 15 second inactive and 30 minute active timeouts. Memory limitations in the routers is the reason why the timeouts exist and aggregates may not be complete TCP connections.

Some research has been done on sampling passive network measurements [44]. The idea is to apply classical sampling methods from mathematics on traffic measurements and, thus, record only a subset of all observed data. Sampling can be applied to packet monitoring or flow measurements. The amount of data recorded depends on the utilized scheme. For example, in [45] the authors propose a method for flow measurements called *threshold sampling* that dynamically controls sample volumes. Moreover, as stated in [44], the best choice of sample design, and, consequently, the amount of data reduction, depends on the traffic characteristics and statistics needed by applications. Unfortunately, not all types of analysis can be applied to sampled traffic measurements. Consider, for example, end-to-end path diagnostics through identification of retransmitted, reordered, and duplicated packets using the method described in [32]. The method inspects the ordering of TCP sequence numbers and IP identification numbers of packets passing by. A sampled packet stream no longer contains the necessary information for this type of analysis.

In this thesis, we concentrate on the analysis of traffic traces consisting of non-sampled TCP/IP packet headers. Flow-level measurements and sampled packet-level measurements do not convey enough information for the techniques we use. We need to be able to perform detailed packet-level analysis tasks, such as in [32], for instance. On the other hand, packet payloads are considered as unnecessary burden for our analysis.

## 2.2 Analysis of Passive Measurements

### 2.2.1 Challenges

The analysis of passively collected measurements is non-trivial as the amount of data is potentially very large. In addition, this data is typically stored during the measurement process into files in an unstructured format making it difficult to process afterwards. This type of approach is often called *off-line* traffic analysis because the analysis is not done while measuring. In contrast, *on-line* analysis can reduce the amount of data that needs to be stored. However, as on-line analysis means performing the analysis tasks on a continuous stream of traffic, such a system needs to be able to process the input data at a rate equal to its arrival rate, which can severely limit the analysis tasks that can be performed. That is why in certain cases it would be desirable to perform a part of the analysis on-line in order to reduce the amount of data, and then perform the heaviest analysis tasks off-line. The raw measurement data, such as TCP/IP

packet headers, is generally processed and analyzed in many ways. Each analysis task generates new data that needs to be stored and possibly processed again later. In other words, traffic analysis is often an iterative process: A first analysis is performed and based on the results obtained, new analysis goals are defined for the next iteration step. Today, handcrafted scripts and a large number of software tools specialized for a single task are commonly used as the tools for traffic analysis. Putting all these facts together, we identify three major challenges in the analysis of passive Internet measurements: management of data, optimization of the analysis process cycle, and scalability.

#### 2.2.1.1 Management

We identify the problem of management as a result of three facts: 1) many tasks are solved in an ad-hoc way using scripts that are developed from scratch, instead of developing tools that are easy to reuse and understand, 2) traffic analysis involves large amounts of data, and 3) the data is typically stored in plain files in a file system.

By data we mean not only the traffic traces containing unprocessed packet data, but also all derived data generated by each analysis task. In [60], the authors describe this type of research work as data intensive science. They describe the data hierarchy in NASA terminology: "The raw *Level 0* data is calibrated and rectified to *Level 1* datasets that are combined with other data to make derived *Level 2* datasets." The authors continue: "Most analysis happens on these *Level 2* datasets with drill down to *Level 1* data." We can see the analogy with the analysis of passive traffic measurements: For example, Level 0 data is the unstructured raw packet traces, Level 1 data is structured packet data, and Level 2 data is aggregated data such as connection-level statistics. However, the tools used generally do not provide any support for managing this large amount of data in this way. Instead, the data is typically archived in plain files in a file system. The problem is that data stored in files has no structure and files contain no metadata beyond a hierarchical directory structure and file names. In fact, we can see the ad-hoc analysis approach as a result of the plain file data storage, because such unstructured and unannotated data encourages ad-hoc techniques to parse and access the data. The situation gets worse when time passes: Depending on the number of files and the skills of the researcher to properly organize them, the later retrieval of a particular data item may be a non-trivial problem. As Paxson [98] has pointed out, the researchers themselves often cannot reproduce their own results. The issue of preserving research data in a larger scale is also discussed in [66].

#### 2.2.1.2 Analysis Cycle

A common workflow to analyze network traffic proceeds in cycles (see Figure 2.1). Due to the lack of structure and metadata, the semantics of the data are not stored during the analysis process. As a result, reusing intermediate results becomes cumbersome and usually the process needs to restart again from the raw data after modifying or changing parameters of the analysis scripts or tools.

Let us take as an example the analysis of BitTorrent [65], a peer-to-peer system for file distribution. When following the analysis steps, one can identify three iterations of the analysis. In a first iteration, we studied the global performance of BitTorrent in terms of how many peers succeeded downloading the complete file. From the results, we noticed a large flash-crowd of peers arriving at the beginning. In a second iteration, the performance of individual sessions was studied. First, the raw data was analyzed on the basis of individual sessions that either had

successfully completed the file download or aborted. In the next step, the performance of the individual sessions in both categories was computed. The information from the previous cycle was combined to obtain average performance of a session during the flash-crowd period and after. In a last iteration, we considered the geographical location of the clients that successfully completed their download to study download performance per geographic region. Since the semantics of the data were not stored during the analysis process, reusing intermediate results (e.g. to integrate geographic information) turned out to be cumbersome and most of the time the data extraction had to be done again starting from the raw data after modifying the scripts.

The issue originates from the fact that the tools do not "understand" the data. Structuring and annotating the data can tell the tool where in a sequence of bytes to find the data values and what they mean, i.e. the semantics. In this way, building on intermediate results becomes much easier, since each tool does not need to separately parse each piece of data, and the tool and its user know the type and semantics of the data they are handling. For example, subtracting a timestamp from another one automatically produces an interval. Subsequently, one comparison operation can determine whether a third timestamp belongs to this produced interval.

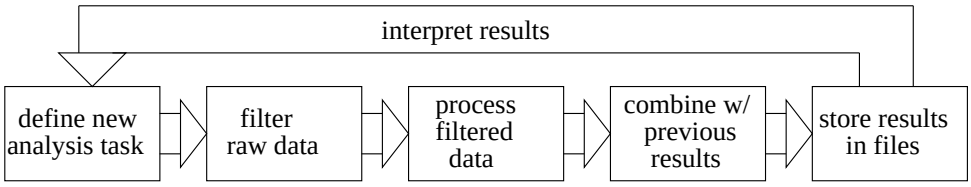



Figure 2.1: Typical cycle of tasks for the iterative process for off-line traffic analysis.

#### 2.2.1.3 Scalability

Scalability is an important issue in traffic analysis as the amount of data is typically large. Often analysis tools are first applied to process small amounts of data. If then applied on larger data sets, it often turns out that the run-time or memory requirement of the tool grows *more than linearly* with the amount of data, in which case modifications and heuristics are introduced that often sacrifice quality of the analysis for performance. Already a measurement data set larger than some Gigabytes may pose serious problems for certain tools because of too large memory or too long run-time requirements. An example is the well-known tool `tcptrace` [13] that can be used to produce summary connection-level data or to extract an individual flow given as an input packet data measured using `tcpdump`. `tcptrace` uses a heuristic to determine the end of a TCP connection. While this heuristic is nowhere clearly documented, the major part of it is an inactivity timeout, as generally in flow record definitions. Despite this heuristic, we were unable to process file sizes larger than 6GB, as we will detail in Section 4.1.

Scalability can be thought of also in another dimension: the depth of the analysis. If data management is an issue, the depth, i.e. the complexity, of the analysis that can be performed in practice may be greatly reduced.

Table 2.2: Characteristics of different approaches for traffic analysis. Traffic volumes are in the order of magnitude.

| Approach | | Aggregation level | Traffic volumes | Data mgt | MD mgt | SW mgt | PA | IA | Online |
|---|---|---|---|---|---|---|---|---|---|
| Ad-hoc scripts | | varies | varies | | | | | | |
| Specialized tools (tcptrace [13]) | | varies | varies | | | | X | | (X) |
| Toolkits (CoralReef [79]) | | varies | varies | | X | | X | | (X) |
| DBMS-based | NetLogger/NetMiner [21] | flow | 10Gbps | | X | | | | X |
| | LOBSTER [2] | flow | 10Gbps | | X | | | | X |
| | Gigascope [39] | packet stream | Gbps | | | X | | | X |
| | Internet Traffic Warehouse [34] | packet | $10^2$ MB/day | X | X | X | | | |
| | IPMon [56] | packet | TB | | X | | | | |
| | InTraBase | packet | $10^1$ GB/trace | X | X | X | X | X | |

X = feature is supported in the approach
blank = feature is not at all supported or is implemented in an ad-hoc manner
(X) = feature is supported in some members of the category
MD = Metadata
SW = Software
PA = Publicly Available
IA = Integrated Approach

### 2.2.2 Database Systems to the Rescue?

The challenges and issues discussed in the previous section raise the question whether database management systems (DBMSs) might ease the process of analyzing passively collected traffic measurements. Traditional database systems (DBSs) have been used for more than 40 years for applications requiring persistent storage, complex querying, and transaction processing of data. Furthermore, DBMSs are designed to separate data and their management from application semantics to facilitate independent application development. Internet protocols have a standardized behavior that leads to well-structured traffic data in the form of packets, and, potentially, could therefore easily be handled with a DBMS. Both, DBSs and plain file systems provide persistent data storage. Thus, in both of them data is physically stored on disk and handling of that data is similar in both approaches. Consequently, one may state that there is only a thin line between file systems and DBSs. The authors of [60] state: "Most file systems can manage millions of files, but by the time a file system can deal with billions of files, it has become a database system.". We present our DBMS-based approach called the InTraBase in Chapter 3 where we also elaborate more on the benefits and gains from using a DBS for analyzing passive packet-level measurements.

### 2.2.3 Existing Approaches

Table 2.2 summarizes and compares the different existing approaches for passive traffic analysis. For comparison, we have also included the InTraBase, but discuss it in detail in Chapter 3.

Data, metadata, and software management are related to the issues in management and analysis process cycle (see Section 2.2.1). Because publicly available solutions are generally more interesting for the research community, we include public availability as a metric. Integrated ap-

proach means that in addition to data, metadata and software are also managed in an integrated way. It is a feature of our approach only, which will be described in Section 3.1. Finally, the capability to analyze traffic on-line is the last feature that we consider. By on-line analysis we mean the ability to perform the analysis tasks on a continuous stream of traffic, i.e. to process the input data at a rate equal to its arrival rate. Naturally, off-line analysis is the other option.

The first approach listed is ad-hoc scripts. However, this approach does not have any of the characteristics we look for. The next step forward are the specialized tools such as `tcptrace` [13] which allows to analyze a `tcpdump` trace and produce statistics or graphs to be visualized using the xplot software. Still none of the important management issues are considered by these tools.

There have been some efforts toward complete analysis toolkits that are flexible enough to be used in customized ways. One example is the Coralreef software suite [79] developed by CAIDA, which is a package of device drivers, libraries, classes and applications. The programming library provides a flexible way to develop analysis software. The package already contains many ready-made solutions. The drivers support all the major traffic capturing formats. This approach concentrates on the software management aspect but addresses neither the problem of handling nor managing the data, i.e. source data and results, nor managing related metadata. Also scalability is an issue.

The next level of approaches is DBMS-based systems. They usually involve large amounts of measurement data, and therefore, require a lot of attention to the organization and handling of the data, i.e. raw traffic data, associated metadata, and derived analysis data. Several of these systems were given birth by industrial projects done by or aimed at Internet Service Providers (ISP): Gigascope, Internet Traffic Warehouse, and IPMON. Consequently, these systems are tailored mostly to fit the needs of large ISPs. Unfortunately, none of them is publicly available.

Sprint labs initiated a project called IP Monitoring Project (IPMON) [57] [56] to develop an infrastructure for collecting time-synchronized packet traces and routing information, and to analyze terabytes of data. In their architecture, a DBS is used for metadata management only and metadata is stored about raw input data sets, analysis programs, result data sets, and analysis operations. Details about metadata management can be found in [93]. IPMON has adopted CVS for managing software.

Gigascope [39] is a fast packet monitoring platform developed at AT&T Labs-Research. In fact, it is not a traditional DBMS but a Data Stream Management System (DSMS) that allows on-line analysis of traffic arriving at high rates. As a powerful DSMS, Gigascope can handle a high rate traffic stream in real-time. However, the real-time requirements imply that the input data are processed in one pass, what evidently imposes limits on the operations that can be performed. We refer the reader to [99] for a detailed assessment of the suitability of DSMSs for traffic analysis. Gigascope is specialized for network monitoring applications such as health and status analysis of a network or intrusion detection. Gigascope does not manage data nor metadata, which requires another solution. At AT&T, this solution is typically their proprietary data warehouse Daytona [1]. Gigascope has a registry for query processes that are providing output streams according to the associated query. The user can also define his own functions and data types for the queries. In this way, Gigascope addresses the software management problem.

The Internet Traffic Warehouse [34] is a data warehouse for managing network traffic data built at Telcordia. Analysis results on application level are provided by storing application information about traffic in addition to IP packet headers. Using a suite of programs, input traffic traces are filtered and parsed into temporary files, which are subsequently loaded into the data warehouse. This system is especially suitable for accounting at ISPs.

## 2.3 Conclusions

In this chapter, we have explained how the Internet can be measured and how our work fits into this context. Off-line analysis of passively collected traffic measurements is challenging because of issues in management of tools and data, suboptimal analysis process cycle, and scalability of tools in terms of the amount of data that they can process and the depth of analysis that can be reached. We have suggested that a DBS could help overcome these issues. In the next chapter, we show that this is indeed the case when we describe our DBMS-based solution.

CHAPTER 3

# InTraBase: Integrated Traffic Analysis Based on Object-Relational DBMS

As discussed in the previous chapter, off-line analysis of passive traffic measurements is challenging from several points of view. Specifically, one needs to consider issues in management, analysis process cycle, and scalability. We also stated that a DBS could overcome many of these issues. In this chapter, we show that it is indeed the case. We introduce our DBMS-based approach for off-line analysis of passive packet-level measurements called InTraBase. We also describe the prototype of the InTraBase built on top of PostgreSQL, an open-source object-relational DBMS. Most of the work described in this chapter has been published in [107].

## 3.1 Approach

### 3.1.1 IntraBase and Other Approaches

The Table 2.2 in Chapter 2 summarizes the differences between the InTraBase and the various other existing approaches for analyzing passive traffic measurements. The following are the main characteristics that differentiate the InTraBase from the other approaches:

- InTraBase is aimed only for *off-line* analysis and does not address the packet capturing or on-line monitoring related issues at all
- InTraBase is designed for intensive *packet-level* analysis of
- *moderate size* ($<$ 50GB) traffic traces.

The InTraBase is not designed, for instance, to monitor the health of a large ISP's network in real-time due to the immense amounts of data that would need to be treated constantly. It is rather an exploratory tool for fine-grained analysis of Internet traffic.

We wish to perform complex traffic analysis tasks that cannot be performed with a single pass over the input data. For this, we need to be able to make multiple iterations over the analysis process cycle, which is generally impossible with systems capable to do on-line analysis. In addition, we perform filtering and parsing operations *within* the DBS, as opposed to processing

the measurement data before loading it into the DBS, and preserve the raw measurements stored in the DBS as unchanged as possible.

We present later the first prototype of the InTraBase, an implementation of our DBMS-based approach. The goal is to devise a platform for traffic analysis that would facilitate the researchers' task. Our solution:

I. conserves the semantics of data during the analysis process;

II. enables the user to manage his own set of analysis tools and methods;

III. enables the user to share his tools and methods with colleagues;

IV. allows the user to quickly retrieve pieces of information from analysis data and simultaneously develop tools for more advanced processing;

V. includes a portable graphical user interface for facilitating exploratory analysis;

### 3.1.2 Fully Integrated Solution Based on Object-Relational DBMS

We advocate a DBMS-based approach for traffic analysis. First of all, we **completely** manage the collected data within the DBS. In other words, we process the "raw" measurement data as little as possible prior to loading it into the DBS. A high-level architectural view of our solution is shown in Figure E.2. We store data from different sources into the DBS. The data that is uploaded into the DBS is referred to as *base data*. Examples of base data are packet traces collected using `tcpdump`, but also logs or other data obtained from application layer, or time series created with the help of Web100 [85] that allows to track precisely the state of a TCP connection at the sender client.

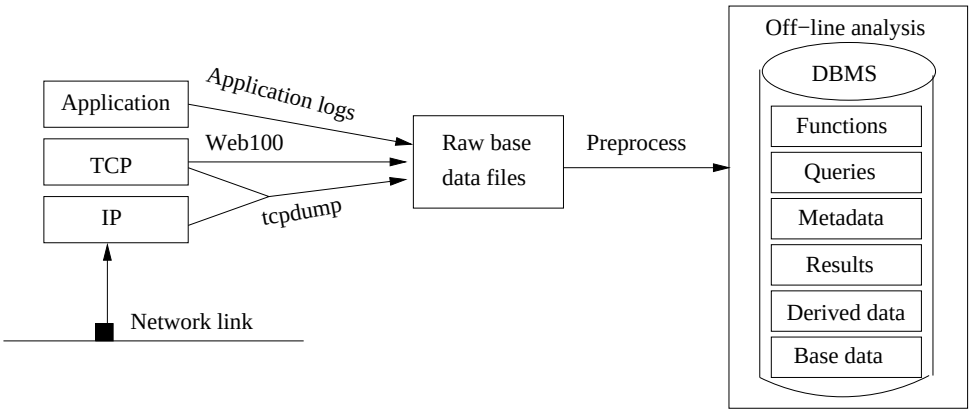



Figure 3.1: High-level Architecture of the DBS.

Once the base data is uploaded into the DBS, we process it to derive new data that is also stored in the DBS. This processing includes, for example, computations of aggregate metrics for each identified connection from the packet-level `tcpdump` base data. All the processing is done **within** the DBS using the functions and queries of the DBS (see Figure E.2). The DBS contains not only all the data but also contains reusable *elementary functions* and more complex tools built on top of the elementary functions, as illustrated in Figure 3.2. The boxes on the lowest layer represent the base data uploaded into the DBS. The middle layer contains the elementary

functions that primarily process the base data and create new data. Finally, the highest layer represents tools for more complex analysis tasks. A given component commonly utilizes some of the components at the lower layers. These relationships are described with arrows. For example, connection level information together with the statistics on packet dispersions allows to compute the capacity of the path of each observed connection [47].

We require an *object-relational* DBMS because of the elementary functions that we need in order to achieve an integrated approach. There must be a way to extend the functionality of the DBS, which is not possible with a standard relational DBMS. A pure object-oriented DBMS (OODBMS) would be another option. The inconveniences include the fact that an OODBMS has no longer tabular data structures which suit well the packetized structure of traffic. OODBMSs lack also a standard query language and all theoretical foundation to support query optimization. OODBMSs advantage is the ability inherited from object-oriented programming languages to describe real world objects. They are also claimed to be superior in performance in specific scenarios when compared to relational DBMS. However, in straightforward queries dealing with data that fits well the tabular structure (not real world objects, for instance) relational DBMS are generally considered more efficient. Finally, one would be bound to use the object-oriented programming language which is used by the DBMS. We see this as a drawback: for example, we explain in Section 3.2 how an object-relational DBMS enables us to use one programming language specifically for statistical computations and to produce graphics, and other programming languages for algorithmic computations.

### 3.1.3 Benefits From Our Approach

#### 3.1.3.1 DBMS Is All About Management

Considering the issues raised in Section 2.2.1, an obvious advantage of using a DBS is the support provided to organize and manage all data. A well defined database schema enables to separate different data (e.g. metadata from "raw" measurements or derived data) and at the same time link them together through common attributes so as not to lose track of which piece of data is related to what.

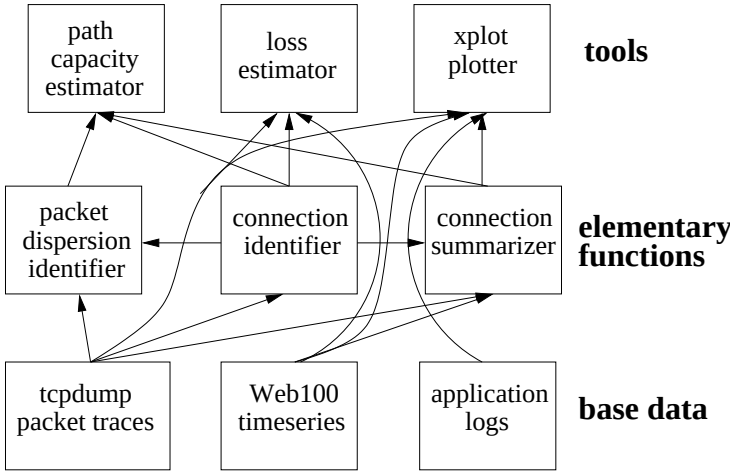
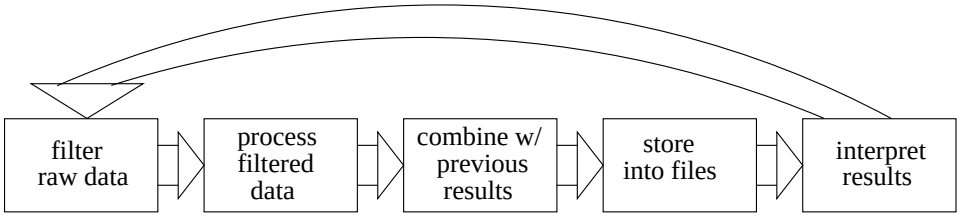



Figure 3.2: Integrated data and tool management.

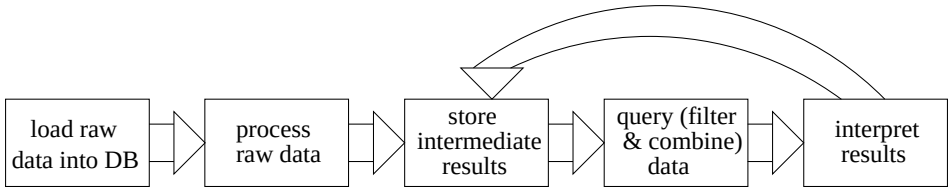
From Figure 3.2, one can identify two additional strong points of a DBMS-based approach. First, since the InTraBase consists of reusable components, implementing new tools will be less laborious and error prone. Also, the data is structured and its semantics are preserved, which facilitates programming. Second, it is possible to easily combine different data sources provided that time synchronization issues between different recorded base data are solved. For example, it is realistic to assume that application layer events explain some of the phenomena in the traffic observed at TCP layer. One can issue joint queries on the tables holding application layer events and TCP traffic to extract the necessary information.

#### 3.1.3.2 Shorter Analysis Process Cycle

Once the base data is in the DBS, it becomes structured data and, therefore, processing and updating becomes easier. Figure 3.3(b) describes the common analysis cycle with our DBMS-based approach. If we compare to Figure 3.3(a), we can see that the initial processing steps performed by the DBS to store data derived from the base data shorten the analysis cycle. Filtering, combining, and aggregating data are operations performed automatically by the query processor of a DBS, which makes this cycle shorter than the one in Figure 3.3(a).




(a) Typical case




(b) Using an integrated DBMS-based approach

Figure 3.3: Cycles of tasks for the iterative process for off-line traffic analysis.

#### 3.1.3.3 Improved Scalability

DBSs are designed to handle very large amounts of data and there exists a lot of literature about performance tuning, which is essential for good performance [106]. In Chapter 4, we look at optimizing the performance of a DBS like the InTraBase. We also evaluate and show how well the prototype scales. As mentioned in Section 2.2.1, scalability can be thought of from

another point of view as well: the depth of the analysis that can be reached. Structured and well manageable data makes searching and filtering easy and allows for complex queries.

## 3.2 PgInTraBase: Prototype Implementation of InTraBase

The prototype of the InTraBase, pgInTraBase, provides the basis for analysis of TCP packet traces captured with `tcpdump` or Endace's `DAG` cards. While this prototype is designed only for the analysis of TCP traffic due to the focus of this thesis, we do not want to imply that our approach is limited to this kind of study. For example, PgInTraBase could be easily extended to support UDP traffic. PgInTraBase is built on PostgreSQL which is an open source DBMS that has a widespread user community. The main reason for choosing PostgreSQL [10] is its object-relational nature that allows to extend the functionality by adding new programming language bindings, called loadable procedural languages (PL). After adding a binding, one can implement external functions in either PL/pgSQL or well-known programming languages, such as Perl or Python, which is impossible with standard relational DBS like MySQL [4]. It is also possible to implement functions in C/C++ and incorporate them into the database as modules. We are using extensively PL/pgSQL [9] and PL/R [8]. In addition, we have implemented some modules with C++. PgSQL is specifically designed for PostgreSQL and R [11] is a language and environment for statistical data analysis and visualization.

### 3.2.1 Database Schema

Packet-level traffic data is commonly recorded in plain files as packet *traces* which may contain hundreds of millions of packets and millions of connections. The focus of the analysis can often be flows of packets or individual connections, as is the case for PgInTraBase. However, when storing this data into the database, it is out of the question to store packets from each connection into a separate table given the potentially huge number of connections in a typical packet trace file. The reason is that handling millions of tables in a single database becomes very inefficient. In addition, querying more than a single connection at once becomes very cumbersome since each additional connection means joining an additional table into the query. Storing all packets from each trace into the one dedicated table would eventually lead to performance problems when the table size grows excessively. Hence, a logical approach is to store packets from each trace into a separate table. In this way, since the typical packet trace size in our case is smaller than 50 GB, the maximum table size stays reasonable.

The core tables used in PgInTraBase are described in Figure E.3. The table *traces* contains annotations about all the packet traces that are uploaded in the database. The *packets* table holds all packets for a single trace. The *connections* table holds connection level summary data for all traces. The *cnxid* attribute identifies a single connection in a packet trace, *reverse* differentiates between the two directions of traffic within a connection, and *tid* identifies a single trace. *Cid2tuple* is a table to store a mapping between unique *cnxid*s and *ap-4-tuples* formed by source and destination IP addresses and TCP ports. The attributes of the *packets* table are those TCP and IP header fields that we found most interesting for the purposes of our root cause analysis. The attributes of the *connections* table covers the commonly interesting connection-level information that can be directly computed as aggregate values from the packets. A similar set of information is given by `tcptrace`, for instance.

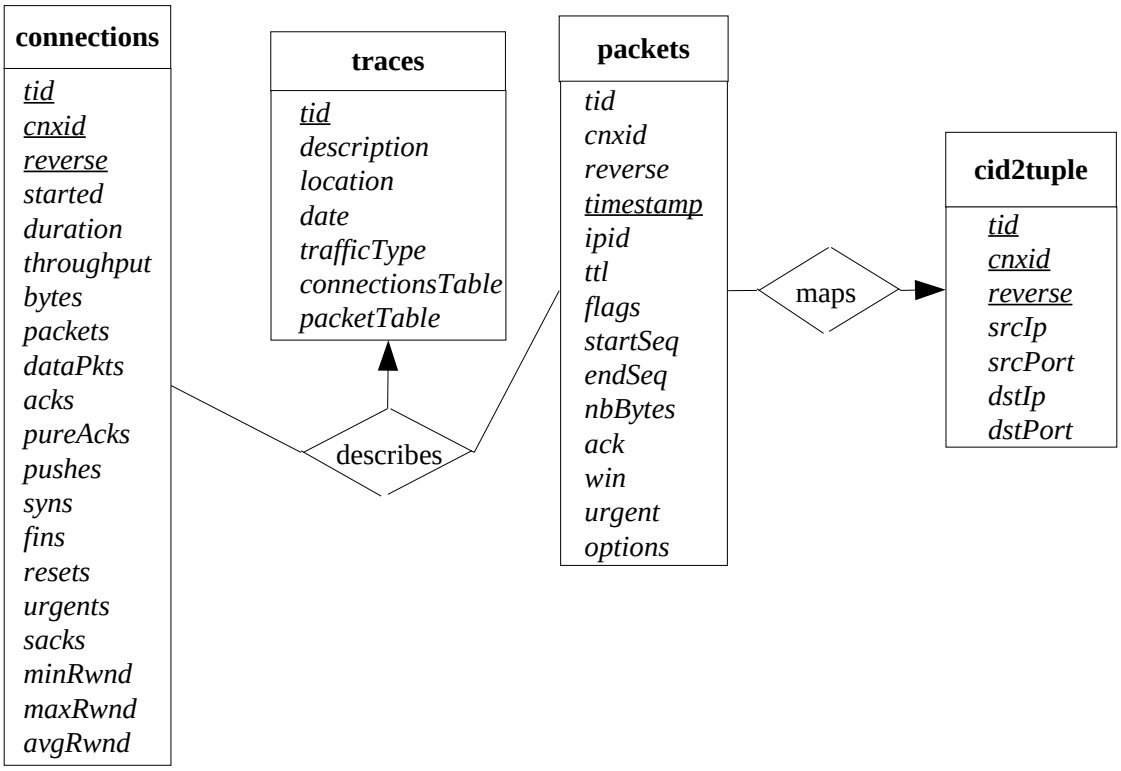



Figure 3.4: The layouts of core tables in PgInTraBase after the 5 processing steps. Underlined attributes form a key that is unique for each row.

### 3.2.2 Processing a Trace: Populating Tables

Processing a `tcpdump` packet trace with PgInTraBase consists of five steps:

I. Store annotation about the trace into the *traces* table.

II. Create the *packets* table.

III. Copy the packet header information into the *packets* table.

IV. Create connection level statistics from the *packets* table into the *connections* table.

V. Insert unique 4-tuple to *cnxid* mapping data from *packets* table into the *cid2tuple* table.

The annotation information of Step I is given by the user that initiates the process. Execution of Step I involves only a single SQL INSERT query. Similarly, Step II is a single SQL CREATE TABLE query. Step III, copying packets into the *packets* table is done as follows: a modified version of `tcpdump` or a program similar to `tcpdump` to read a DAG trace (called `dagdump`) is used to read the packet trace file. The output is piped directly to the database using SQL COPY command. The modifications made to `tcpdump` ensure that each line of text, i.e. the header

fields of each packet, is well-structured before uploading it into the database. More specifically, each line of text representing a TCP packet contains all the attributes defined in the packet table. If an attribute is missing, the filter program adds a special character signifying that its value is null. For example, pure acknowledgments do not have starting and ending sequence numbers and the filter program inserts null values for them. The modified `tcpdump` also adds a trace identifier *tid*, which for a new trace is simply the largest one in the *traces* table plus one (thus, it is the same for each packet in the same trace), a unique connection identifier *cnxid*, and a reverse attribute for each packet. A connection identifier is unique for each ap-4-tuple within a trace. More information about using the ap-4-tuple as unique identifiers for connections is given in Section 4.1.2. The remaining two processing steps are performed with two SQL queries. The SQL language allows to compute these aggregate metrics and store them into a table using a single query. In Figure E.3, the table *packets* does not contain the 4-tuple attributes and, in fact, the reason for performing the processing step IV is that we can drop the 4-tuple attributes data from the *packets* table, which saves significant amounts of disk space because we only store the 4-tuple twice per connection (both directions) instead of once for each packet.

### 3.2.3 Analyzing Processed Data

After the five processing steps, the tables in Figure E.3 are populated with data from the packet trace and the user can either issue standard SQL queries or use a set of functions provided for more advanced querying. Alternatively, the user may develop his own functions. The schema shown in Figure E.3 enables the user to limit the analysis on connection level but also to drill down to packet level.

We have implemented functions in procedural languages to perform operations that cannot be done with plain SQL queries. We used the PL/pgSQL language to write "algorithmic" functions as opposed to statistical computations. We wrote functions that plot graphs in xplot format and produce time series of throughput, packet inter-arrival times, jitter, retransmitted packets etc. We also developed functions that perform analysis tasks on a packet trace. These functions belong to the tools category in Figure 3.2 as they generally use the lower-level *elementary functions* to complete their task. For example, using specific time series functions, we can separate for all connections in a given packet trace all bulk transfer periods from inactive periods where the application operating on top of TCP is producing little data (detailed in Chapter 5). PL/R is used to write functions that produce graphs and do statistical calculations.

In order to ease the typical exploratory analysis process, we have built a Java-based graphical user interface (GUI). This GUI is especially useful for those users of PgInTraBase that are familiar neither with SQL nor with PL languages. The GUI is portable to different operating systems since it has been developed in Java. The user can navigate between the different tables (e.g. drill down to packet details of a selected connection) and visualize connections with several different plots. The GUI connects to the DBMS server (Postmaster) through the network and, therefore, it is possible to use it remotely.

### 3.2.4 Properties of PgInTraBase

In Section 3.1.1, we presented the following list of desired properties for the InTraBase prototype:

I. conserve the semantics of data during the analysis process;

II. enable the user to manage his own set of analysis tools and methods;

III. enable the user to share his tools and methods with colleagues;

IV. allow the user to quickly retrieve pieces of information from analysis data and simultaneously develop tools for more advanced processing;

V. include a portable graphical user interface for facilitating exploratory analysis;

We now check whether PgInTraBase has these properties: Property I is enabled by the structured and semantic data provided by the DBMS. Properties II and III are ensured by the PL language functions that can be ported as such to another installation of PostgreSQL. The user can either manually issue SQL queries or use the GUI for quick retrieval of a particular piece of information. On the other hand, a way to perform more advanced analysis task is to develop PL functions. These two facts together enable property IV. Finally, property V is ensured by the Java-based GUI.

## 3.3 Conclusions

We presented in this chapter our approach for off-line analysis of passively collected Internet traffic measurements. This approach that we call InTraBase is based on using an object-relational DBMS. It is able to overcome many of the issues encoutered with ad-hoc methods such as scripting flat files. We have built a running prototype of the InTraBase that we call PgInTraBase based on the PostgreSQL DBMS and have shown that it fulfills our five desired properties.

In the next chapter, we focus on the performance evaluation and optimization of the prototype to give an idea on how feasible such a DBMS-based approach is. This is of great interest because DBMS are often considered as too slow for such scientific data processing due to, for instance, the transaction management overhead.

CHAPTER 4

# Evaluation and Optimization of the InTraBase

In [60], the authors asked scientists why they do not use DBS to manage their data. One of the answers was: "We tried them but they were too slow". Indeed, any system, no matter how sophisticated and attractive on paper, is useful only if it can be implemented and offers good enough performance to be usable in practice. For example, suppose that an easy-to-use system is able to compute a result of a complex analysis task that is cumbersome to perform with other systems. However, if it takes days to obtain a result for every analysis task, such a system is useless in practice, given that another system may compute a result in minutes but require more efforts from the user. That is why this chapter is devoted to performance evaluation and optimization of the prototype of the InTraBase. We first look at the feasibility in terms of disk space consumption and performance with a focus on the initial processing steps common to all traffic traces.

We then concentrate on typical analysis tasks performed on the data that is already processed with the initial steps and, thus, available as populated tables in the database. Good performance of these tasks is essential for efficient root cause analysis. We do an in-depth analysis and optimization of the prototype for the typical analysis tasks and quantify the impact of the optimizations through performance measurements. While we present the optimizations and analyze their impact from the point of view of PgInTraBase, most of them can be applied as such also to other DBS that process packet-level traffic measurements. The majority of the contents of this chapter has been published in [107] and [108].

## 4.1 Evaluation of the Prototype

We evaluate the prototype of the InTraBase in two ways: first, we assess the feasibility in terms of processing time and disk space consumption and, second, we perform qualitative and quantitative comparisons of our approach with the `tcptrace` tool.

### 4.1.1 Feasibility of PgInTraBase

We conducted measurements in order to evaluate whether a DBMS-based solution scales in an acceptable way, i.e. has a reasonable processing time for large files and acceptable disk

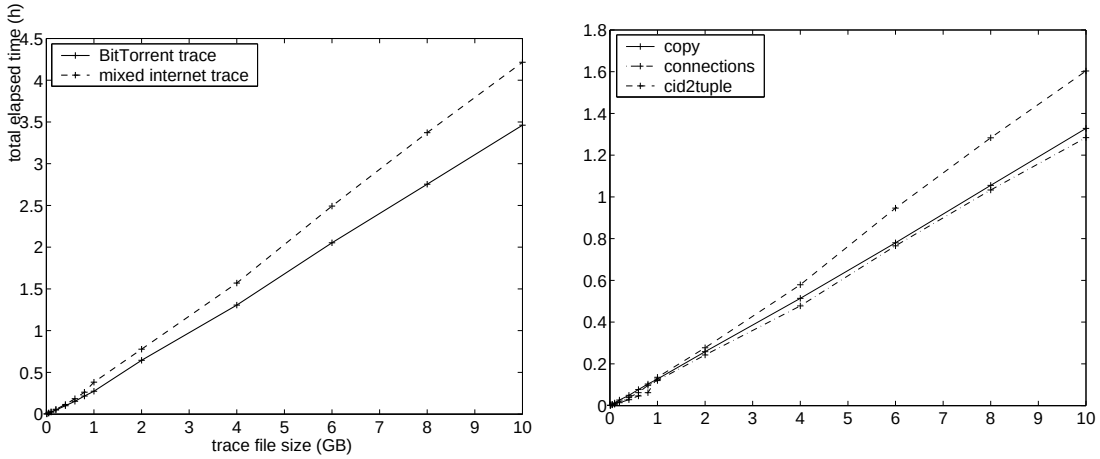
space overhead. We executed the five different processing steps described in Section 3.2.2 using different size `tcpdump` packet trace files of two different types of traffic: BitTorrent traffic and mixed Internet traffic. BitTorrent, as a peer-to-peer file distribution system, tends to produce long-lived and large connections in terms of transferred bytes. A typical mixture of Internet traffic, on the other hand, contains many short connections, the so-called mice, and few long-lived connections, the so-called elephants [62]. Consequently, a BitTorrent trace file contains fewer connections than a mixed Internet trace file of the same size. For example, the 10 GB files of BitTorrent and mixed Internet traffic contain 53,366 and 1,709,993 connections, respectively. As most of the processing is done on a connection basis, we expect the number of connections to have an impact on the performance. Our BitTorrent traffic trace file contains also fewer packets than the mixed Internet traffic trace file of the same size. The reason is that the BitTorrent traffic was captured on a Sun machine where the minimum capture length is 96 B whereas the mixed Internet traffic was captured using the default length of 68 B. We did our tests using Linux 2.6.3 running on an Intel Xeon Biprocessor 2.2GHz with SCSI RAID disks and 6GB RAM[1].

#### 4.1.1.1 Processing Time of the Initial Steps

In order to build confidence on the measurement results concerning the processing time of the initial steps, all the measurements were repeated ten times consecutively. We present in each case the mean values. The variations between consecutive measurements were in all cases negligible. Out of the five steps, only steps 3, 4, and 5 (copying the packets, creating connection level information and the 4-tuple to *cnxid* mapping) contribute significantly to the total processing time. Thus, steps 1 and 2 are not included in the analysis.

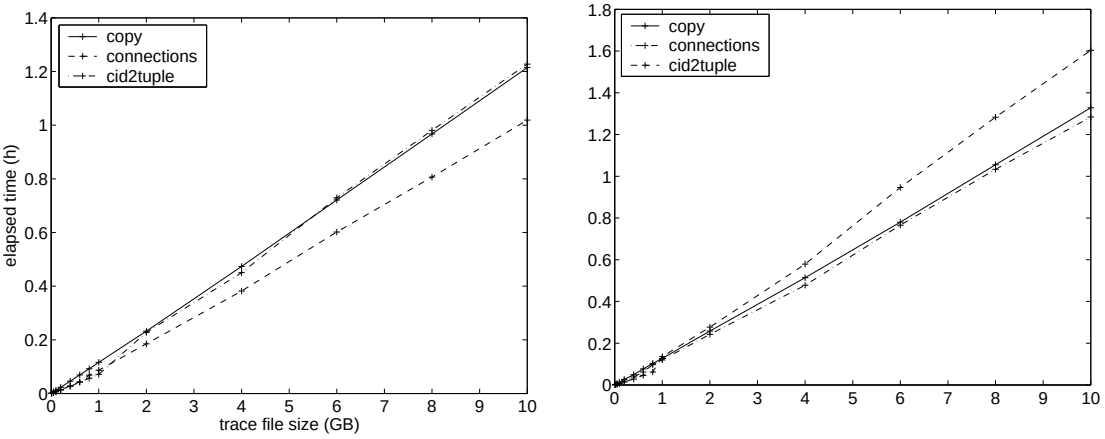
Figure 4.1(a) shows the evolution of the processing time as a function of the trace file size for BitTorrent traffic and mixed Internet traffic. We observe that for both of the traces the growth is linear. In fact, since eventually each packet in the packet trace needs to be processed, it is impossible to find a solution that scales better than $O(n)$, where $n$ is the number of packets. We can only try to minimize the processing time per packet. Processing BitTorrent traffic takes approximately 20 minutes per GB and mixed Internet traffic approximately 25 minutes per GB. The difference between the two traces is mostly explained by the different number of packets when comparing traces of the same size in bytes. Figure 4.1(b), that plots the processing times against the number of packets, confirms this reasoning. The processing time per packet is quite similar for the two traces: 2.3 and 2.1 minutes per thousand packets for mixed Internet and BitTorrent trace, respectively. It is somewhat surprising, but encouraging, that the processing time scales similarly regardless of the number of connections within the trace. One would expect at least the operations performed on connection basis (steps 2 and 3) to become more and more time consuming when the number of connections increases. Instead, the DBMS seems to be able to optimize the execution in a way that avoids such issues. We also analyzed how much each step contributes to the total processing time. The results are presented in Figures 4.2(b) and 4.2(a). There is little differences between the execution times of each processing step. The overall performance results are good enough as the prototype is rarely used for processing traffic traces larger than 10 GB and it is not used for real-time analysis.

[1]Unfortunately, PostgreSQL is unable to take advantage of more than 2GB of this memory in certain critical operations such as sorting.




(a) time vs. number of bytes (b) time vs. number of packets

Figure 4.1: Total processing time of the three steps vs. `tcpdump` file size.




(a) BitTorrent traffic traces (b) Mixed Internet traffic traces

Figure 4.2: Processing times of different steps with respect to trace file size.

#### 4.1.1.2 Disk Space Consumption

Modern DBMSs use indexes to speed up access to information stored on disk. We use indexes in the InTraBase prototype and discuss the way we use them and their benefits in Section 4.2.3. However, indexes introduce an overhead in disk space consumption due to the necessary indexing data that needs to be stored. In other words, speed-up given by indexes is somewhat traded off

in disk space usage. Also, data stored in a database takes up more disk space than in a flat file because of different data structures used in the DBSs, e.g. a value stored as a four-byte integer in a database might not require all the four bytes when stored in a flat file. Moreover, we need to store two-byte and four-byte fields from the TCP/IP packet headers using 4 B and 8 B data types because PostgreSQL does not support unsigned data types. Finally, we add some data such as connection and table identifiers for each packet but also remove some redundant data by storing only single instances of IP addresses and TCP port numbers. Figure 4.3 visualizes the disk space consumption for the BitTorrent trace. It is the *packets* table that consumes most of the disk space. The sizes of other tables are negligible and are therefore excluded from the figure. The total disk space consumption of the data in the database is 1.4 to 1.5 times larger than for a flat file. The disk space overhead due to indexes is around 15% from the plain data stored in the database. The results are similar for the mixed Internet trace. A total disk space overhead of 50% is acceptable since our objective is not to process terabytes of data at a time. Moreover, nowadays disk space can be considered cheap and, thus, is rarely an issue. Note that this overhead in disk space consumption is the price to pay for having structured data.

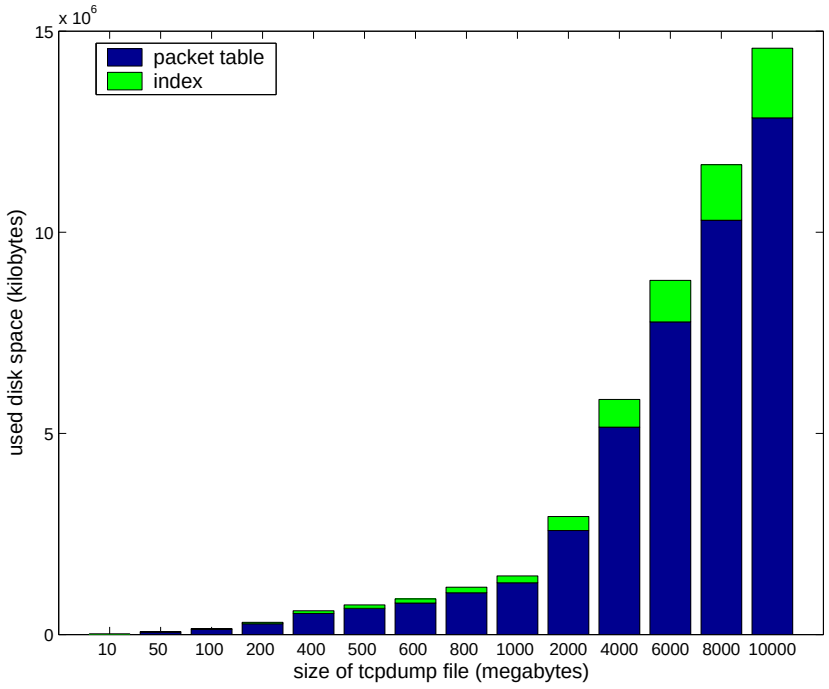



Figure 4.3: Disk space usage for different `tcpdump` file sizes containing bittorrent traffic.

### 4.1.2 Comparison of InTraBase and Tcptrace

Let us emphasize here that comparing the processing times of `tcptrace` and InTraBase as such is meaningless. As stated in Section 3.1.3, the benefits of InTraBase are in the depth of the analysis that can be done and its scalability. As for the processing times with `tcptrace`, with file sizes up to 4 GB the processing times were in the order of minutes, as expected. However, while

we were able to analyze BitTorrent traces up to 10 GB in size with `tcptrace`, we encountered severe problems when we tried to analyze mixed Internet traces. `Tcptrace` was unable to finish the analysis with file sizes of 6 GB and larger, because after using 3 GB of memory, it could no longer allocate more. The problem is due to the large number of connections whose state `tcptrace` tries to maintain in the memory throughout the processing of a trace.

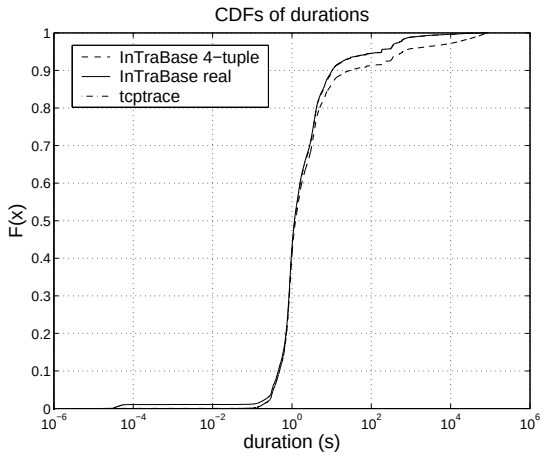
Because `tcptrace` is among those tools that often need to trade-off accuracy for performance, as we already discussed in Section 2.2.1, it is interesting to compare the results obtained from InTraBase and `tcptrace`. We compare the per connection statistics produced by `tcptrace` and InTraBase. In the case of InTraBase, we consider two different definitions of a connection: the *ap-4-tuple* formed by source and destination IP addresses and TCP ports, from now on called ap-4-tuple connection, and an accurate one, referred to as true connection, where we further separate connections within a particular ap-4-tuple due to TCP port number reuse. We accomplish this by searching multiple TCP three-way hand shakes, i.e. SYN packet exchanges, among packets sharing a common ap-4-tuple. The definition of a true connection agrees perfectly with the definition from the specification of TCP [101] and, therefore, can be considered as the "correct" one. These two different cases for InTraBase are included to get an idea of how often they differ and in this way to assess the need for separating the connections within a distinct ap-4-tuple[2].

`Tcptrace` reported statistics from a 10 GB BitTorrent trace on 55482 connections while InTraBase found 53366 ap-4-tuple connections and 56162 true connections. This shows that `tcptrace` and InTraBase's ap-4-tuple definition indeed miss some true connections. Cumulative distribution function (CDF) plots of connection durations are shown for each case in Figure 4.4(a). The cdf plots for `tcptrace` and InTraBase's true connections agree with each other almost perfectly and cannot be distinguished from each other. Between 10 and $10^5$ seconds the cdf of durations of the ap-4-tuple connections deviates from the two other curves. This suggests that the ap-4-tuple definition captures fewer short connections, which is expected since some of the ap-4-tuple connections are in reality several connections due to the reuse of TCP port numbers. However, a look at Figures 4.4(b) and 4.4(c), which show the cdf plots of connection sizes in bytes and packets, respectively, reveals that the connections missed by the ap-4-tuple definition carry negligible amounts of bytes and packets. All in all, the differences between these three sets of connection statistics for the 10 GB BitTorrent trace seem to be marginal. Therefore, it is justified to simply use the ap-4-tuple as connection identifier in most of the cases and avoid heavy operations to further improve the accuracy. We did not conduct the same comparison with the mixed Internet traffic trace since `tcptrace` was unable to process the 10 GB trace due to memory limitations.

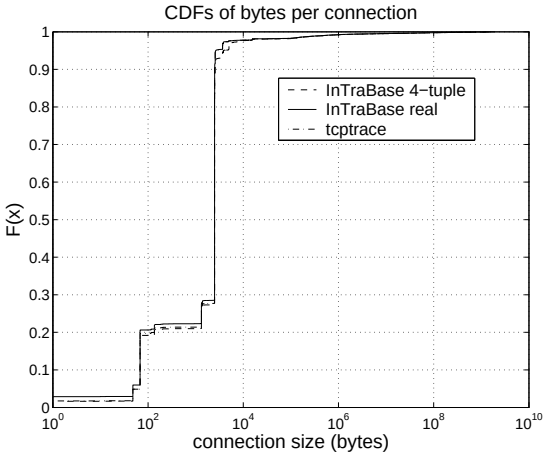
## 4.2 Optimizing the DBS for Efficient Analysis

We now turn the focus from evaluation to optimization of the performance of our prototype system. A typical off-the-shelf DBMS is optimized for a usage pattern that is very different from the one we have in the InTraBase. For example, the InTraBase generally does not need to process many concurrent queries, and thus, we can relax the parameters affecting the concurrent query processing as much as possible in order to reduce performance overhead. Therefore, tuning

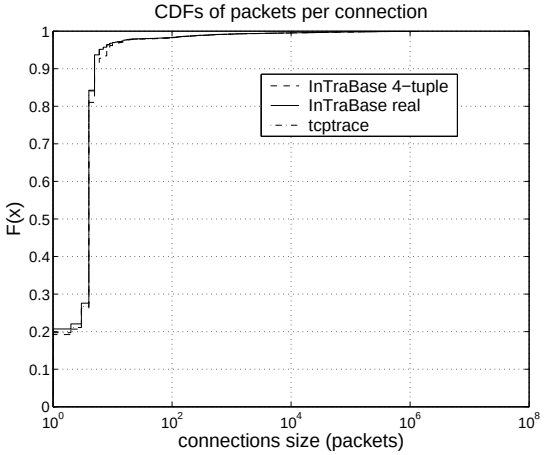
[2]Generally, we consider only the ap-4-tuple as the identifier of a connection. As it is not entirely accurate and improving the accuracy would come with a price to pay in performance, we wish to assess the need for it.




(a) Durations of connections




(b) Sizes of connections in bytes




(c) Sizes of connections in data packets.

Figure 4.4: Comparison of Per-Connection Statistics from `tcptrace` and InTraBase.

the DBMS to fit this specific usage is very important. Furthermore, the characteristics of the traffic data and the focus of the analysis generally lead to certain specific queries being extremely popular. For example, in our case, we identify one very frequent query: "give me all the packets from this connection in chronological order". Hence, to reach maximum performance, the performance of the DBS should be optimized for these popular queries.

We describe in this section the key issues to take into consideration when designing a DBS to support efficient analysis of packet-level network traffic measurements. The way the DBS should be optimized generally depends on the type of analysis that is commonly performed, i.e. the queries issued. For this reason, this section presents a case study on our prototype PgInTraBase. Nevertheless, many of the concepts are also applicable in a general context. For

example, a system where the analysis focuses on UDP flows does not differ significantly from another system focusing on analysis of TCP connections. In Section 4.3, we demonstrate the importance of the optimization through performance measurements on PgInTraBase.

### 4.2.1 Tuning the DBMS

A standard off-the-shelf DBMS is optimized for processing a large number of concurrent queries. It takes care of issues related to parallel access to data and enforces atomicity of transactions, etc. However, our experiences with the InTraBase suggest that the typical usage of a packet-level traffic analysis DBS for research purposes generates a very different workload: few users seldom issuing queries that commonly are very I/O intensive touching large amounts of data. In addition, several queries are rarely executed simultaneously. In this case the DBMS must be tuned to conform to the special usage.

The fact that concurrent query processing is rare allows to set most of the buffer sizes high since they are normally defined on per process, i.e. query, basis. For the majority of DBMSs, it is possible to tune parameters such as memory available for a sorting or index creation. Since a queried connection can potentially contain millions of packets and often they need to be sorted, for instance, chronologically, it is important to set the amount of memory available for sorting as high as possible.

Write-Ahead Logging (WAL), a.k.a. redo logging [58], is a standard approach to transaction logging in DBMSs. Briefly, WAL's central concept is that changes to data files (where tables and indexes reside) must be written only after those changes have been logged, that is, when log records have been flushed to persistent storage. Since we can assume little parallel access to data, the WAL parameters can be set as lazy as possible, by setting the commit delays to the maximum, in order to let the underlying operating system (OS) optimize the I/O operations.

Caching plays a very important role in the performance of the DBS. The amount of memory available for caching is also a modifiable parameter in most of the DBMSs. Caching can greatly improve the performance of per-connection or per-flow querying of packets in cases where the same groups of packets, i.e. specific connections, are analyzed over and over again in different analysis tasks. In this case the data should be read from the disk once and cached for the following queries. Clearly, the execution order of the queries needs to be carefully chosen. In addition, as we explain in the next section, the caching techniques used need also to be well chosen.

PgInTraBase runs on PostgreSQL on top of Linux 2.6.3. The hardware consists of an Intel Xeon Biprocessor 2.2 GHz with a SCSI RAID system and 6 GB RAM. PostgreSQL allows to set the parameter `work_mem` that controls the amount of memory available for internal sort operations and hash tables. We set this to no higher than 1.5 GB in order to avoid out-of-memory problems: On a 32-bit system the maximum process size is around 3 GB and in certain cases several sort operations (typically not more than two) may be run in parallel each of which is allowed to consume the amount of memory specified by `work_mem`. Also, 1.5 GB should be sufficient in most cases to sort in main memory all the packets of a single connection, which is important since we do generally per-connection analysis. WAL commit delays were set to 100 ms (the maximum).

### 4.2.2 Identifying and Decomposing the Typical Analysis Task

The analysis tasks are generally performed for predefined groups of packets within a trace. In the case of TCP traffic, this group is typically a connection or a flow. The latter can be used also for traffic generated by a connectionless protocol such as UDP. In the case of PgInTraBase, the most common query that we have used is:

```
SELECT *  FROM trace1_packets
          WHERE connection_id=x
          ORDER BY timestamp
```

The above query, hereafter called *c-query* (c for connection and common), returns all the attributes (e.g. for TCP traffic all the IP and TCP header fields and a timestamp) of all the packets that belong to connection `x` from table `trace1_packets` in chronological order. As indicated in Figure 4.5, this query is executed in the beginning of each typical analysis task and the analysis results, e.g. the number of reordered packets, are computed by inspecting the query results row by row. Finally, the result is stored into another table or alternatively printed on the screen. Note that even if the c-query is made more complex (e.g. by adding WHERE rules or by joining in another table), the query processor of the DBMS would strip it down into the original c-query and perform additional filtering and querying separately and merge the results in the end. In other words, fetching packets for a specified connection from the database boils every time down to executing the c-query.

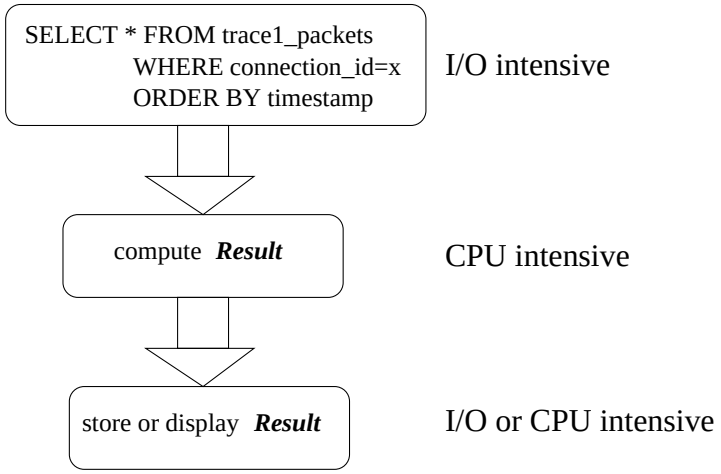



Figure 4.5: Executing a typical analysis task.

In this discussion, we ignore those frequently issued queries that do not involve querying packet-level data because their contribution to the performance is negligible due to orders of magnitudes smaller data sets. For example, before executing analysis tasks in the way described in Figure 4.5, one may wish to select a set of connections having certain characteristics (e.g. more than 100 packets) for which these analysis tasks are performed. The table storing per-connection statistics contains for each trace only a very small fraction of the amount of rows stored in the table holding the packets.

### 4.2.3 Cost Minimization of the Typical Analysis Task

Our goal is to optimize the execution of the analysis task displayed in Figure 4.5. Optimizing the middle step in Figure 4.5 is always specific to the analysis task and, therefore, generic solutions do not exist. Based on our experience, regardless of the analysis task, the number of results stored or displayed per task is typically very small compared to the number of packets queried in the first step. Thus, the last step in Figure 4.5 is rarely the bottleneck and it is the first step that we focus on. This step reads the tuples that represent the packets from disk and it is therefore generally I/O bound. We derive from the above the main optimization goal:

- *Minimize the I/O time for the c-query.*

#### 4.2.3.1 Indexes for Fast Lookup

The two most important concepts in DBMSs for I/O optimization are *indexing* and *clustering* [106]. Indexes allow fast lookup of specific rows from tables. They can be thought of as hash lookups of logical records. In the case of our c-query, since each queried packet belongs to the same group, we can increase the performance by adding an index on the connection or flow identifier to each table containing packets. This enables the DBMS to do an *index scan* with the help of the created index, touching only the required disk blocks, instead of doing a *sequential scan* on the contents of the entire table, touching all the disk blocks associated with the table, each time the query is executed. Since the c-query includes ordering by timestamp, one might think that it would be beneficial to create another combined index of the connection identifier and the timestamp. In this way, the DBMS can perform simply an index scan on this combined index and does not need to sort by the timestamp in addition. There is a caveat, though: Index scanning using this combined index is computationally much more expensive than with the simple connection identifier index. We compared the cost of executing an index scan using the combined index of connection identifier and timestamp versus the plain connection identifier index. With PgInTraBase, the query planner of the DBMS reported almost hundred times larger cost estimates[3] for the combined index than for the plain connection identifier index. Thus, in the case we have created both types of indexes, each time the query planner optimizes the execution of a query, it always decides to use the plain connection identifier index and sort afterwards if the number of packets to be fetched is higher than only one percent of total packets in the connection. The reason is that it is simply faster because the overhead of performing an index scan using the combined index is greater than the additional cost of sorting after using the plain index. After all, sorting data that is in main memory is very fast compared to the index lookups. Consequently, the combined index is useful only when a small fraction of packets of the connection is queried. We have found it useful when studying the option negotiation (e.g. MSS and window scaling) during the TCP connection establishment, for instance.

As for details about indexes of PgInTraBase, each table containing packets and the two tables *cid2tuple* and *connections* are indexed. In Figure 4.6, the indexes of the tables are added to the table layout as numbers in parenthesis following the attribute. Each packet table is indexed by connection identifier and a second index on the combination of connection identifier and timestamp. The two other tables are indexed on the trace identifier and connection identifier to enable fast lookup of information of a given connection within a given trace.

[3] A cost estimate is the query planner's guess at how long it will take to run the statement. It is measured in units of disk page fetches.

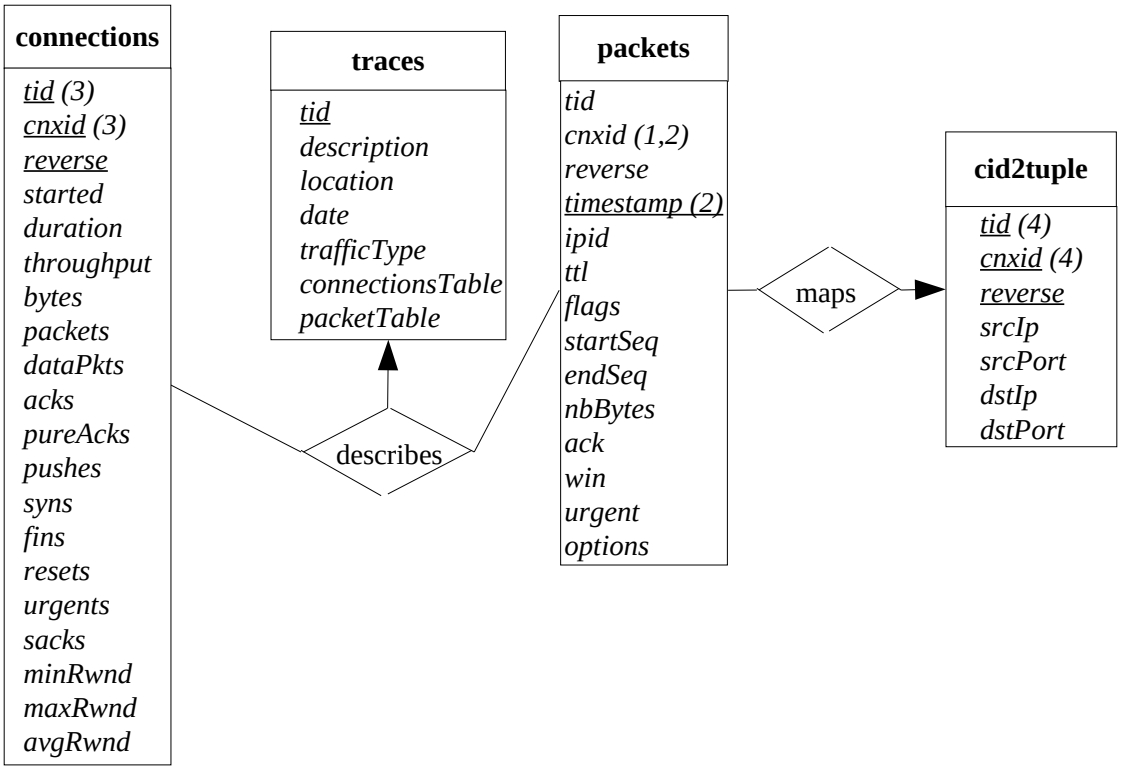


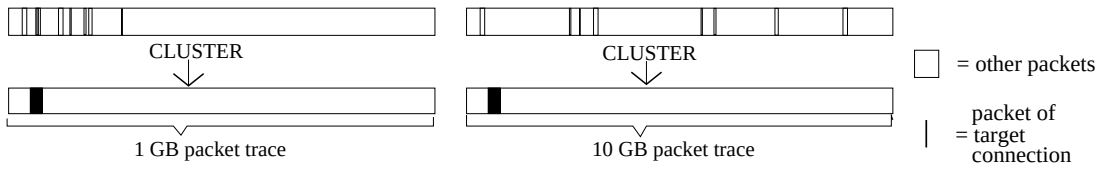

Figure 4.6: The layouts of core tables with indexes. Numbers in parenthesis indicate the different indexes.

#### 4.2.3.2 Clustering to Minimize Cost of I/O Reads

While indexes assure that during the execution of a read query only necessary disk blocks are read, clustering makes sure that these read operations are efficient. Clustering means grouping together data by physically reordering it on the hard disk. The advantage is improved performance for operations that access the grouped data, i.e. a query with a given attribute value that is indexed. The reason is that in this way the disk read head accesses only adjacent disk blocks instead of moving throughout the entire disk in the worst case. In addition, fewer disk blocks are required to be read if the data is not scattered throughout the disk.

The speedup from clustering can be tremendous but depends highly on the characteristics of the data and the clustering parameters. For the case of packet-level traffic data, consider Figure 4.7 that illustrates the effect of clustering for one example connection in two different packet traces. The two traffic traces are of the same size in terms of number of packets but the one in Figure 4.7(b) has many more connections in parallel. This trace could represent a short capture of a high speed aggregate edge link. The trace in Figure 4.7(a) has only a few connections in parallel and could be captured at an end host or lightly loaded server, for instance. In order to speed up the c-query, we cluster the packet data with respect to connections or flows. The

packets of each traffic trace are originally stored on the disk in their arrival order. That is why the impact of clustering on a packet trace with numerous parallel connections, Figure 4.7(b), is much bigger than on a trace with only a few parallel connections, Figure 4.7(a). It is clear that the penalty of the random seeks necessary for reading all the packets of the connection in the unclustered case compared to the sequential reads in the clustered case is much higher in Figure 4.7(b) than in Figure 4.7(a).




(a) Trace with few connection in parallel.

(b) Trace with many connections in parallel.

Figure 4.7: The effect of clustering a single connection within two different types of traffic traces of the same size. Black stripes are packets belonging to the connection that is being clustered and their horizontal distance from each other reflects the physical distance on the disk.

In PgInTraBase, the `packets` table is clustered based on the index on `cnxid`. We would not gain much by clustering the data in the other tables since in the other tables each index value returns only a few rows, i.e. two rows per connection. Also, these tables would need a periodical reclustering since the contents are changing when ever a new packet trace is inserted into the system.

Indexing and clustering cause some additional overhead: indexes consume disk space and clustering is a rather expensive operation. We observed in Section 4.1 that the indexes cause approximately 15% overhead in disk space consumption. Figures 4.8 and 4.9 show for different sizes of packet traces the time it took us to create an index for the `cnxid` attribute of the packet table and to then cluster this table with PgInTraBase[4], respectively. We observe that the execution time of both operations scales linearly except that there is a "step" in the processing time between the file sizes of 2 GB and 4 GB. This behavior is most likely due to the memory allocation in Linux. The maximum allocatable amount of memory for a single process in Linux 2.4 on a 32-bit machine is between these two values, a bit less than 3 GB[5]. Crossing this boundary in trace file size probably causes a step in the processing time, after which it still scales linearly but with a higher factor. Both execution times are acceptable given that they are *one time* operations and that the potential gain is large, as we will demonstrate in Section 4.3.

[4]PostgreSQL provides a `CLUSTER` operator that is very slow. A faster way to cluster a table with PostgreSQL is to create a new one from query results, i.e. by executing `CREATE TABLE newtable AS SELECT * FROM oldtable ORDER BY cnxid` and then recreate the indexes for the `newtable` table (see the manual [10]).

[5]In theory the maximum is 3 GB but in practice there are always some tiny chunks of available wasted memory that are smaller than what is attempted to be allocated.

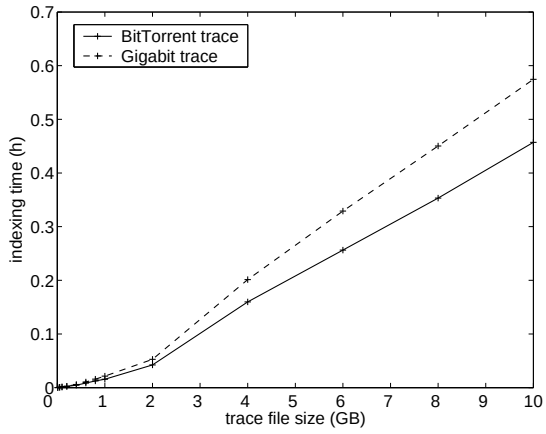



Figure 4.8: Elapsed time to index different sizes and types of traces.

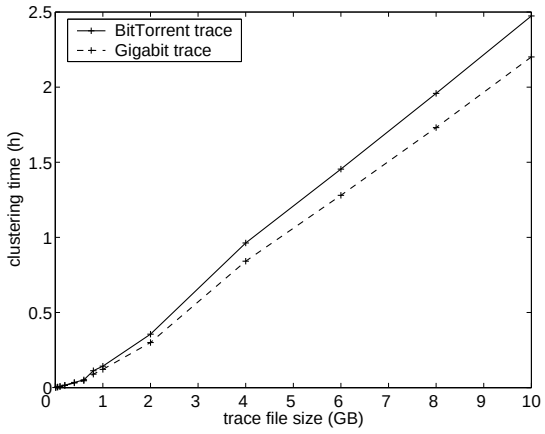



Figure 4.9: Elapsed time to cluster different sizes and types of traces.

#### 4.2.3.3 Parallel I/O

The execution time of I/O operations can be further reduced by means of parallel I/O. There are several ways to go. It is possible to implement it on application level with parallel DBSs. This approach gives a lot of control over the parallelism through load balancing, for instance, but may suffer from severe overhead due to complexity (e.g. distributed transaction handling). A simpler approach is to implement it on the lowest layer possible, that is, using RAID (redundant array of independent disks) on striping mode, i.e. RAID level 0. RAID striping is typically implemented so that adjacent blocks are written on different disks in order to maximize the parallelism in the case of sequential disk access. Figure 4.10 illustrates how adjacent disk blocks would be distributed among three disks. For example, when striping over $n$ disks, any $n$ adjacent disk blocks would be stored on different disks. Since we have already clustered the data in connections, and thus, access adjacent disk blocks whenever executing the c-query, we obtain maximal parallelism with I/O operations. The server used for PgInTraBase is operating a RAID consisting of eight SCSI disks capable of running in striping mode.

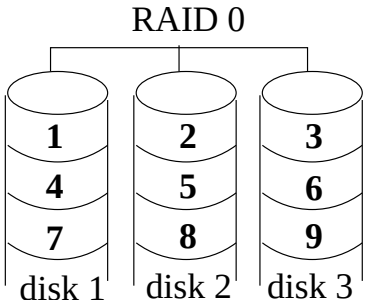



Figure 4.10: Raid striping over three disks, i.e. RAID level 0.

#### 4.2.3.4 Caching

In addition to minimizing the I/O time of the c-query, it is equally important to avoid repeating it unnecessarily. Therefore, caching is important. If the same c-query is executed several times in a row, the query results should remain cached in the main memory after the first query in order to avoid repeating the expensive I/O operations. It is possible to cache the data on different layers: on the OS layer (file system cache) or on the application layer. All modern OSs implement some sort of file caching techniques. The same applies for most modern DBMSs. However, they may all use different techniques and, therefore, implementing this principle is specific to each DBS.

As for PgInTraBase, there are several options as it runs on PostgreSQL on top of Linux. PostgreSQL has two separate caching methods: it uses its own buffer cache, implemented as ARC (Adaptive Replacement Cache [89]) starting from version 8.0, but also relies on the file system cache of the underlying OS. ARC tries to avoid cache flushing (caused by a one-time very large sequential scan, for instance) by keeping track of how frequently and how recently pages have been used and keeps in the cache pages that have the best balance of the two. Since we would like to cache every time the query results of the c-query, cache flushing would be in fact desirable each time we execute a new c-query. Thus, we minimize the utilization of PostgreSQL's own cache and attempt to force the DBMS to use the Linux file system caching as much as possible.

## 4.3 Evaluation of the Impact of Optimization

In order to demonstrate the importance of the I/O optimizations described in Section 4.2.3, we measured several metrics in the case of the InTraBase. To quantify the impact of individual optimizations on the performance, we did measurements while executing the c-query with and without indexing, clustering, and RAID striping. The query was executed on thousands of connections with different sizes in terms of number of packets. The query results were directed into /dev/null in order to measure the retrieval process of the packets only (first step in Figure 4.5). We measured also the effect of caching when executing several times the same most common query.

As in the evaluation in Section 4.1, we used two different packet traces: one recorded on a Gigabit link on a university edge containing a mixture of Internet traffic, and another one recorded on a much slower link with a total throughput all the time below 10 Mbit/s containing only BitTorrent traffic. In this way we could observe the difference in clustering visualized in Figure 4.7. We selected from the Gigabit trace the 3060 connections having more than 100 packets and a different number of packets. From the BitTorrent trace we selected all the 583 connections having at least 100 packets. In the Gigabit trace, we had only few connections with very large numbers ($> 10^6$) of packets and a lot of small connections, which is typical for Internet traffic given the heavy-tailed distribution of flow sizes [62]. We explain here only the main findings from the measurements. Detailed discussions with plots of measurements results are presented in Appendix B.

The analysis of the measurements revealed that while it is always good to use an index when querying a small number of packets, it is not necessarily the case when querying a large number of packets unless the data is clustered. Indeed, unclustered packet data may be so dispersed throughout the disk that sequential scan proves to be more efficient than index scan. If the

queried data is indexed and clustered, the total execution time of the query scales linearly with the number of packets queried.

Without indexing and clustering the time to execute the c-query of a single analysis task for all the 3060 selected connections of the Gigabit trace would take approximately eight days and one and a half days for the 583 BitTorrent connections. When introducing an index, the total execution times drop to 5 h and 1.3 h, respectively. Finally, when the data is also clustered we obtain total execution times of 27 min and 36 min, respectively.

The effects of caching and parallel I/O with RAID striping proved to be minor for the total execution time. The reason is that after indexing and clustering our system was no longer I/O bound and became CPU bound instead. Thus, further I/O optimizations had little effect. Nevertheless, these particular results cannot be generalized: With a faster CPU, multiple CPUs with parallel processing support from the DBMS[6], or slower disks the situation might not be similar and further I/O optimizations could improve significantly the performance. Finally, we discovered that once the system was CPU bound, most of the time went to internal DBMS operations related to handling the result set tuples, e.g. transforming the physical storage format of the data into the logical format of the tuples, which we cannot further optimize. This computational overhead is the price to pay for having the structured data and advanced querying facilities provided by the DBMS.

## 4.4 Conclusions

In this chapter, we evaluated and optimized PgInTraBase, our prototype of the InTraBase. We have shown through measurements that the InTraBase is a feasible approach for off-line analysis of passive packet-level traffic measurements if the DBS is correctly tuned and optimized. Our evaluation of the PgInTraBase demonstrated that the difference in performance is substantial between an optimized and default DBS solution.

[6]The version of PostgreSQL that we use does not support parallel processing and, thus, a single query that is executed as a single process does not benefit from both of the processors in our system.

# Conclusions for Part I

In Part I of this thesis, we first explained different ways to measure the Internet. Our work on root cause analysis of TCP traffic focuses on the analysis of passively collected traffic measurements. Using ad-hoc methods for such analysis work is common but often leads to severe problems. Based on our initial experience with ad-hoc methods, we identified the need for a new approach.

Our methodology for the analysis of passive measurements is based on an object-relational DBMS. We showed that this approach is able to overcome many of the issues encountered with ad-hoc approaches. Furthermore, we demonstrated through measurements of our prototype implementation that such a system has good enough performance. Nevertheless, our measurements also showed that in order to obtain good performance, the system needs to be correctly tuned and optimized based on the type of analysis tasks performed.

We have used the prototype implementation presented in this part to derive all the results on TCP root cause analysis that we present in this thesis. In Part III, where we tie the first two parts of this thesis together, we explain in detail how the InTraBase can be used in this type of analysis work. Specifically, we describe the final layout of the database and design of the functions used in the prototype for TCP root cause analysis.

# Part II

# Root Cause Analysis of TCP Traffic

# Overview of Part II

Part II is concerned with the techniques and algorithms for root cause analysis of TCP throughput, the main contributions of this thesis.

In Chapter 5, we first introduce briefly the basic concepts of the TCP protocol. We then describe the different potential throughput limitation causes and the ways they present themselves in TCP traffic observations.

After this background knowledge the reader should be able to understand the main content of this part: the root cause analysis techniques and algorithms that we present in the Chapters 6 and 7. Chapter 6 focuses on the interaction of applications with TCP. We develop an algorithm that enables to partition a TCP connection into Bulk Transfer Periods (BTP) and Application Limited Periods (ALP), refer to Section 5.2.1 for the definitions. We study their properties using example traffic traces originating from different applications. In Chapter 7, we concentrate on the analysis of BTPs and develop several algorithms to extract different metrics for them from the traffic traces. These metrics can then be used to classify BTPs according to the dominating throughput limitation causes that they experienced.

CHAPTER 5

# Origins of TCP Transfer Rates

The behavior of TCP, specifically its performance in terms of delay and throughput, has been studied and modeled since its emergence. The mechanisms of the protocol itself are intuitive. However, understanding and modeling the behavior and traffic of TCP becomes difficult when it is interacting with the application layer above and network layer below. In addition, the interactions with other cross-traffic within shared links complicate the behavior of an individual TCP connection. In this chapter, we first briefly describe how the TCP protocol works. We then discuss the different causes that can prevent TCP from achieving a higher throughput. Finally, we present earlier research work related to this part of the thesis.

## 5.1 TCP

The *Internet Protocol (IP)* is the network-layer protocol used in the Internet to communicate between two devices. Data is carried in packets. IP packets may be delivered out of order, they may be dropped or even duplicated before they reach the destination. That is why the *Transmission Control Protocol (TCP)* is needed. TCP is the reliable transport-layer protocol of the Internet that operates on top of IP. It is reliable in the sense that it tries to guarantee the correct delivery of data between applications that are communicating through a TCP connection. Correct delivery means the in-sequence delivery of non-corrupted data. TCP is estimated to carry over 90% of the traffic in the current Internet.

The original specification of TCP dates back to 1981 and was first introduced as the RFC 793 [101]. The specification describes the format of data and acknowledgments, the mechanisms for providing reliability and flow control, and schemes for connection establishment and termination. Congestion control was not introduced until 1988.

We describe in this section the basics of TCP before diving into the root cause analysis of TCP performance in the following section. We first go through separately the main characteristics of TCP, namely, connection management, error control, flow control, and congestion control. There have been numerous versions of TCP and improvements to the basic mechanisms introduced since 1981. However, only few are widely adopted. We focus here only on these versions and improvements.

### 5.1.1 Connection Establishment and Tear Down

A host that is identified with an IP address may contain several active TCP connections at the same time. A *TCP end point*, also known as a *TCP socket*, is identified with a port number associated with the host, i.e. the IP address and TCP port number tuple. A *TCP connection* is established between two TCP end points. Thus, a TCP connection is identified by a four tuple containing the pairs of IP addresses and TCP port numbers.

TCP uses a *three-way handshake* procedure to establish a connection. Figure 5.1 illustrates how the handshake proceeds. The first packet is sent with a SYN flag. The receiver replies by transmitting an acknowledgment (ACK) packet with SYN flag set. Finally, the initiator replies by sending an ACK. The initiating TCP is allowed already to transmit data in the third packet.

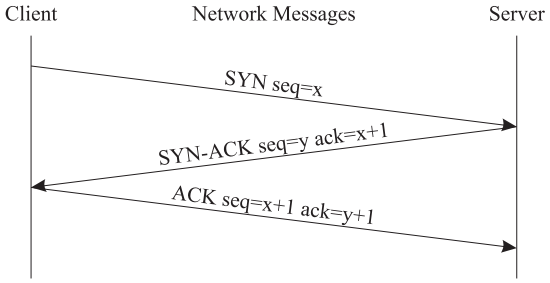



Figure 5.1: Establishing a connection using the three-way handshake.

The primary purpose of the three-way handshake is to prevent old duplicate connection initiations from causing confusion. Also the initial sequence numbers are randomly chosen to avoid collisions with an old incarnation of the connection. In addition, some parameters of the connection. such as the initial sequence number, maximum segment size (MSS) [101], receiver window scaling [68], and TCP timestamps [68], are exchanged during the handshake.

Closing a connection happens similarly through an exchange of packets with FIN control flag set. We refer the reader to [101] for details on the connection establishment and tear down. These procedures become more complicated when some packets get lost during the handshake, for instance.

### 5.1.2 Error Control: Cumulative Acknowledgments and Timeouts

Error control is needed in order to cope with corrupted and lost packets. The principle is that each received non-corrupt data packet needs to be acknowledged by the receiver with an ACK packet before new data packets can be transmitted. Checksums are used to detect corrupted packets and sequence numbers to detect lost packets. If a packet is not acknowledged, because it was corrupt or lost, within a predefined time from the sending, it will be retransmitted. Such an event is called a timeout. The delay between the time instants when a packet is sent by a host and received by another host varies constantly in the Internet. For this reason, the retransmission timeout used by TCP is continuously updated and computed as a function of the current sample value and the variation of the round-trip time (RTT), i.e. the time interval between sending a packet and receiving an ACK for it.

TCP uses a *cumulative acknowledgment scheme* meaning that, upon receiving a packet, the receiver always acknowledges the last correctly in sequence arrived packet. For example, consider

a scenario where packets 1 and 2 have been correctly received and packet 3 has been lost. When packets 4 and 5 arrive, the receiver will continue to send ACKs for packet 2. Advantages of this scheme are its simplicity and the fact that a lost ACK does not force retransmission if the ACK of the subsequent data packet is received in time. The drawback is that if a data packet is lost, the sender does not know if subsequent data packets were correctly delivered. That is why the sender typically uses a retransmission mechanism called *go-back-n*, i.e. the sender retransmits all sent packets in sequence starting from the lost one.

RFC 2018 [84] proposed *Selective Acknowledgments (SACK)* as a TCP option which enable the receiver to inform the sender about successfully transmitted out-of-sequence segments while still maintaining the cumulative acknowledgment scheme. In this way the sender can retransmit selectively only those packets that were not correctly delivered. SACK is nowadays widely supported. The authors of [88] stated that in 2004 64.7% of the web servers in the Internet they examined supported SACK correctly. The current versions of TCP use typically a delayed acknowledgment strategy, which means the receiver acknowledges only every second packet. This mechanism incorporates also a timer that forces, upon expiry, the receiver to send an ACK even if only one packet has been received after the last ACK sent. ACKs can also be *piggybacked* into a data packet in the case of two-way data transfer within a single connection.

### 5.1.3 Flow Control: Sliding Window Technique

TCP uses a so called sliding window technique to control the flow of data. The goal of flow control is that the sender does not transmit data faster than the receiver is able to handle it, i.e. sender does not overrun the receiver's buffer. The sender is allowed to have transmitted multiple unacknowledged packets at a time. The amount of allowed unacknowledged bytes, also called the amount of *outstanding bytes*, is controlled via the size of the sender's sliding window. The size of this window controls the transmission rate: rate= $\frac{\text{window size}}{RTT}$. The TCP receiver advertises the size of its available receive buffer as *receiver advertised window (rwnd)* (this value is added in each sent ACK packet) which sets the size for the sender's sliding window. The sender's transmission rate is then adjusted by the `rwnd` value so that the maximum number of allowed outstanding packets is equal to the size of the receiver advertised window at any given time instant, i.e. rate= $\frac{\text{rwnd}}{RTT}$.

Figure 5.2 demonstrates how the window slides. Each time the leftmost sent packet (the one with the lowest sequence number) in the window is acknowledged, the window slides to the right and a new packet can be sent.

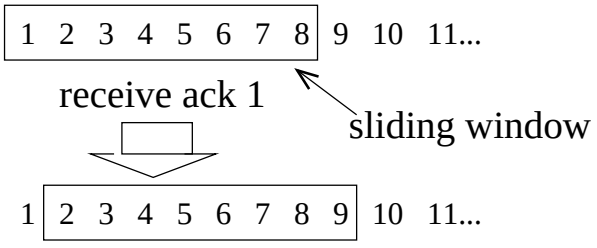



Figure 5.2: Sender's window slides.

### 5.1.4 Congestion Control: Resizing the Sliding Window

Lost packets in the Internet are generally due to network congestion. In this context, congestion refers to a state of the network where one or more routers receive packets faster than they can forward them. After the queues of one of those routers fill up, it starts to drop packets. TCP retransmits lost packets, which introduces an overhead in bandwidth utilization. The purpose of congestion control is to try to minimize congestion and, consequently, the need for retransmissions by adjusting the transmission rate of TCP. The main concept of congestion control is the *congestion window (cwnd)*, which controls, with the receiver advertised window, the size of the sender's sliding window and, thus, the transmission rate. At any given time instant, the maximum amount of outstanding bytes is equal to $min(cwnd, rwnd)$. Different flavors of TCP have different strategies to react to a loss event, i.e. they resize the `cwnd` differently. After introducing the basic acknowledgment and retransmission mechanisms of TCP, we describe these variations between the versions of TCP used today.

RFC 793 did not include any congestion control. However, after the first congestion collapse in the history of the Internet in 1986, it was considered necessary that TCP reacts to network congestion by reducing its current transmission. Jacobson was the first to introduce such mechanisms in [67]. These mechanisms are part of each implementation of TCP today. The proposal contains the basic rules used for resizing the `cwnd`. According to these rules, TCP functions in two modes: slow start and congestion avoidance.

#### 5.1.4.1 Slow Start and Congestion Avoidance

Slow start and congestion avoidance are essentially different strategies to grow the `cwnd`. In the beginning of a connection, TCP sender is in slow start mode. The size of the `cwnd` is initialized to one and is increased by one each time a new ACK arrives. Thus, the size of the `cwnd` grows exponentially. Figure 5.3 plots the evolution. Each time a loss is detected, i.e. a timeout occurs, the TCP backs off by resetting the `cwnd` again to one. At the same time *slow start threshold (ssthresh)* parameter is set to half of the `cwnd` size before the loss detection.

When the size of `cwnd` reaches the current value of *ssthresh*, TCP enters congestion avoidance mode. In this mode the size of `cwnd` is increased by one each time a `cwnd` worth of packets have been acknowledged.

#### 5.1.4.2 TCP Tahoe: Fast Retransmit

Because of the cumulative acknowledgment scheme, a TCP receiver can only acknowledge the last packet received in sequence. Thus, if packets arrive out of sequence (e.g. one packet was lost but packets sent later arrive correctly) the receiver sends the same ACK more than once. The Tahoe version of TCP uses duplicate ACKs as an indication of packet loss. Specifically, after the arrival of the third duplicate ACK, the sender retransmits the expected segment immediately without waiting for the retransmission timer to expire. This mechanism is called *fast retransmit*. In TCP Tahoe, the sender enters slow start after retransmission and resets `cwnd` to one packet.

#### 5.1.4.3 TCP Reno: Fast Retransmit & Fast Recovery

TCP Reno introduced a new mechanism called *fast recovery* [111] that changes the congestion control behavior after fast retransmit: When three duplicate ACKs are received, TCP sets

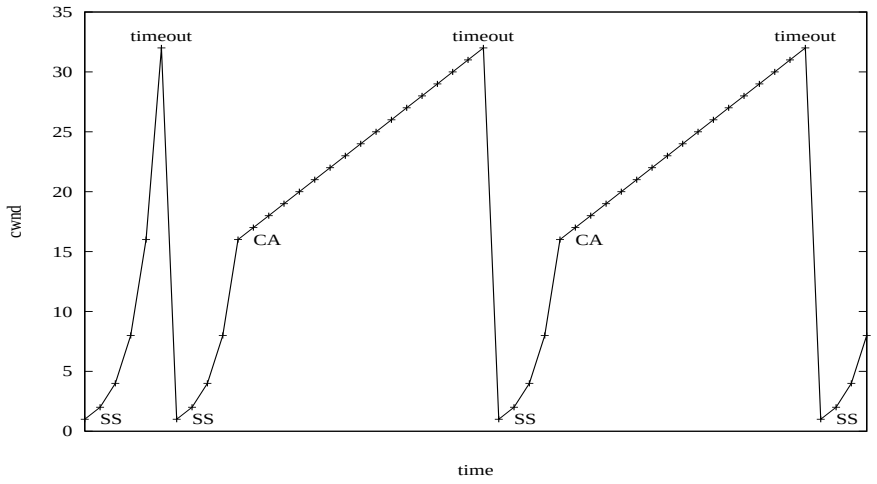



Figure 5.3: Evolution of the `cwnd` size during slow start (SS) and congestion avoidance (CA).

the `ssthresh` to half of `cwnd` and `cwnd` to `ssthresh` plus three packets. For each subsequent duplicate ACK `cwnd` is incremented by one. The next time new data is acknowledged, `cwnd` is set to `ssthresh` and TCP leaves fast recovery mode. This behavior is illustrated in Figure 5.4.

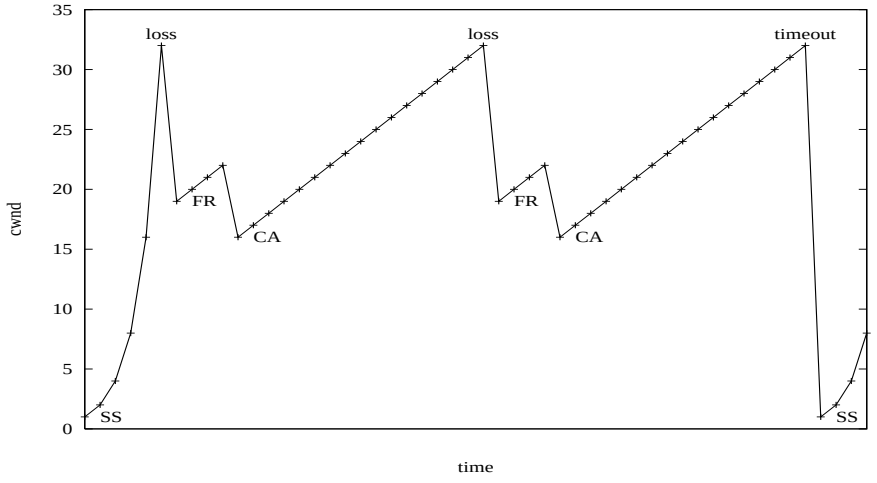



Figure 5.4: Evolution of the congestion window if Fast Recovery is used.

The idea behind this modification is that the reception of a duplicate ACK does not only mean that a segment has been lost, but also that a subsequent segment has arrived at the receiver. Thus, the path is not completely congested and TCP does not need to back off as much as suggested by the earlier versions. Note that the fast recovery algorithm does not apply after a timeout. In that case slow start is entered as described earlier.

#### 5.1.4.4 TCP NewReno: Improved Handling of Multiple Losses During Fast Recovery

TCP Reno works well when one packet is lost within a window. However, it runs into trouble when multiple packets within a window are lost. Even with only three packets lost within a window, Reno may end up waiting for a timeout [48]. The NewReno version of TCP [52] includes a small change to the Reno's fast recovery algorithm in the sending TCP's behavior that fixes this problem.

NewReno utilizes the idea of *partial acks*: when there are multiple packet drops, the acks for the retransmitted packet will acknowledge some, but not all the segments sent before the fast retransmit. In TCP Reno, the first partial ACK will bring the sender out of the fast recovery phase. This will result in timeouts when there are multiple losses in a window. In New Reno, a partial ack is taken as an indication of another lost packet. Unlike Reno, partial acks do not take NewReno out of fast recovery. Instead, it retransmits one packet per RTT until all the lost packets are retransmitted and avoids multiple fast retransmits from a single window of data.

#### 5.1.4.5 Other TCP Versions

There have been other improvements and proposals for versions of TCP. TCP Vegas [31], for instance, is a proposal for a TCP version that attempts to avoid congestion before losses actually occur. In [54] the authors discuss using Explicit Congestion Notification (ECN) mechanisms with TCP. ECN mechanisms are proactive methods to avoid severe congestion through notifications. The Forward Acknowledgment (FACK) congestion control algorithm [86] suggests some extensions for SACK TCP. None of the above proposals have really been widely adopted. For instance, in [88], the authors report that over 90% of the web servers they evaluated were not ECN-capable. Their study also shows that most of the TCP versions they encountered and were able to classify turned out to be NewReno.

## 5.2 What Limits the Transmission Rate of TCP?

The common view of a TCP transfer is that its transmission rate is limited by the network, i.e. a link with a small capacity or a congested bottleneck link. We demonstrate in this section through examples that this view is too restrictive. Instead, the limitation causes may lie in different layers of the network stack and either in the end points or in the middle of the TCP/IP data path. Zhang et al. [118] performed pioneering research work into the origins of Internet TCP throughput limitation causes. They defined a taxonomy of rate limitations (application, congestion, bandwidth, sender/receiver window, opportunity and transport limitations). While our classification is greatly inspired by their work, we extend the scope of this work and discuss the difficulties of identifying certain causes through examples. We present the causes in a top down manner, starting from the application level down to the network level.

### 5.2.1 Application

Figure E.5 describes the way data flows from sender to receiver application through a single TCP connection. The interaction happens through buffers: at the sender side the application stores data to be transmitted by TCP in buffer `b1`, while at the receiver side TCP stores correctly received and ordered data in buffer `b2` that is consequently read by the receiving application.

Data that is received out of order is stored in buffer `b4` until it can be delivered in order and stored into buffer `b2`. The behavior of the sending application reflects the behavior of the application protocol while the receiving application should always read the buffer `b2` whenever it contains data. In case the receiver application is unable to read the buffer `b2` as fast as TCP delivers data into it, the receiving TCP will notify the sending TCP by lowering the receiver advertised window. We discuss this case as a TCP-layer phenomenon in Section 5.2.2.

We define two types of periods within a given TCP connection that we use throughout this part of the thesis. When the application sends data constantly, buffer `b1` in Figure E.5 always contains data waiting to be transferred. We refer to such a period as *Bulk Transfer Period (BTP)*. In other cases, when the application limits the throughput achieved, TCP is unable to fully utilize the network resources due to lack of data to send. We call such a period *Application Limited Period (ALP)*. Note that a TCP connection consists entirely of these two types of periods.

The interaction between the sending application and TCP manifests itself in the traffic in diverse ways depending on the type of application. To illustrate our point, we present time vs. sequence number plots of TCP traces that carry different types of application traffic. Throughout this section, we use similar plots to visualize the other limitation causes as well. These plots are created using the `xplot` (www.xplot.org) plotter. The bottom line tracks the received acknowledgments and vertical arrows indicate sent data packets. A diamond on top of a vertical arrow means that the data in this packet was "pushed", i.e. sent with a Push flag. RFC-793 [101] says: "The sending user indicates in each SEND call whether the data in that call (and any preceding calls) should be immediately pushed through to the receiving user by the setting of the PUSH flag.". Thus, "pushing" is a way for the application to notify TCP that this piece of data should be sent right away even if it is not a full size (MSS) packet, because it does not have more data to provide to TCP at the moment.

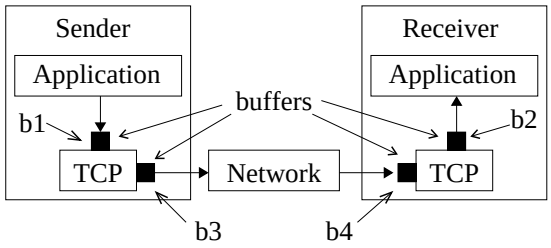



Figure 5.5: Data flow from the sender to the receiver application through a single TCP connection.

A type 1 application continuously produces small amounts of data. At the TCP layer, this results in small bursts of data, typically packets of size less than the allowed maximum segment size (MSS) of the connection. Typical examples are (i) live streaming applications, such as Skype [25], an IP telephony application, that transfers data over TCP at a constant rate of 32 Kbit/s (if it cannot operate over UDP), and (ii) applications that impose transmission rate limits. Figure 5.6 shows a time vs. sequence diagram of Skype traffic. All packets sent have the push flag set and carry only 42 bytes. Note that even if Nagle's algorithm was enabled, pushed data is sent immediately. Therefore, in a case such as our example of Skype, Nagle's algorithm has no effect.

A type 2 application comprises applications such as telnet and instant messaging applications (IRC, MSN Messenger), for instance. These applications do not generally produce constant rate traffic. Instead, the traffic pattern, specifically the instantaneous transmission rate, depends on the user behavior. However, the transmission rates reach very rarely the limits set by transport and network layers and, thus, the traffic remains entirely limited by the application, or the user to be more exact.

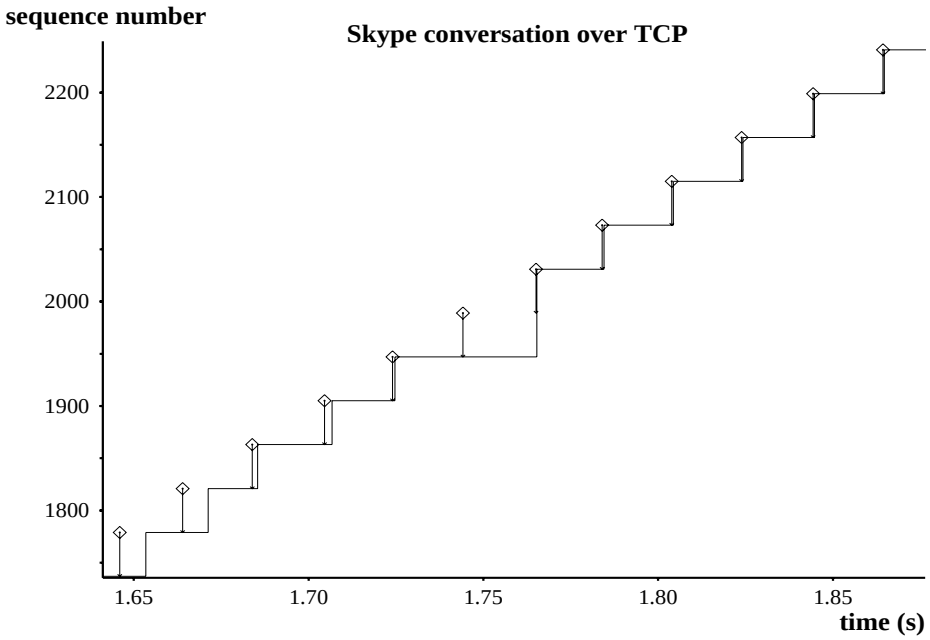



Figure 5.6: A short piece of Skype connection.

A type 3 application produces data in bursts separated from each other by idle periods. An example of such behavior is Web browsing with persistent HTTP connections. The user clicks on a link to load a web page, causing a transfer period, reads the page, causing an idle period, and clicks on another link that points to the same web site, which causes another transfer period. Another example is BitTorrent that uses permanent TCP connections to send blocks of data during transfer periods and keep-alive packets during choked periods [38]. Figure 5.7 shows an example of a typical BitTorrent connection that alternates between transfer periods ("vertical" lines) and choked periods ("horizontal" lines). The upper line tracks the receiver advertised window. Keep alive messages, visible as plain diamonds, are sent regularly during the choked periods. Note that the transfer periods are visible as almost straight lines because the time scale is much larger than the one in Figure 5.6, for example.

The type 4 application traffic is produced by FTP like applications that typically transmit everything at once (see Figure 5.8).

Some TCP connections may even include combinations of these four types of behavior. For example, a BitTorrent connection can exhibit a behavior that is a combination of type 1 and type 3 applications. Such connections alternate between choked and transfer periods, and, in addition, the client application enforces a transmission rate limit during the transfer periods. Table 5.1 summarizes these different application types.

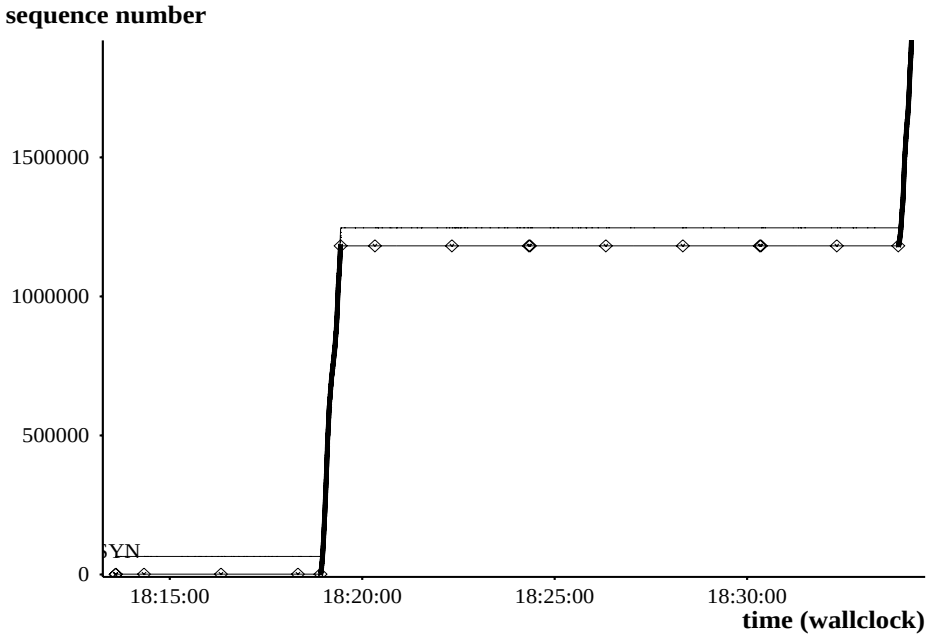



Figure 5.7: 20 minutes of a BitTorrent connection.

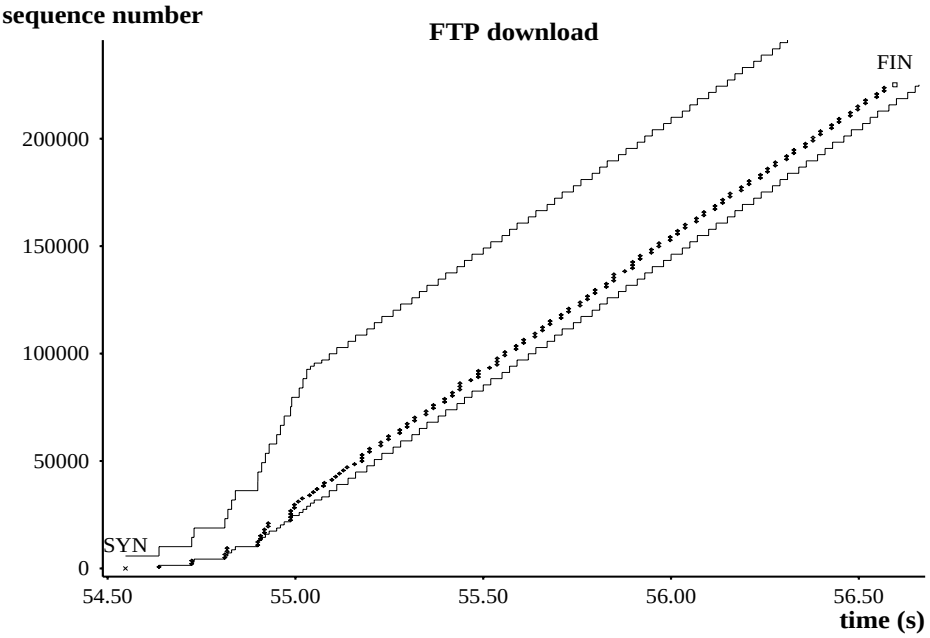



Figure 5.8: Entire FTP download connection.

### 5.2.2 TCP Layer

#### 5.2.2.1 TCP End-Point Buffers

The achieved throughput of TCP can be limited by the size of the buffers allocated at the two end-points of a connection, i.e. buffers `b3` and `b4`, but also `b2` as discussed in previous section.

Table 5.1: Summary of different application types.

| type | main characteristics | example applications |
|---|---|---|
| 1 | constant application limited transmission rate, consists of a single ALP | Skype and other live streaming, and client rate limited eDonkey |
| 2 | user dependent transmission rate, consists typically of a single ALP | telnet and instant messaging applications |
| 3 | transmission bursts separated by idle periods, applications using persistent connections, consists of BTPs interspersed with ALPs | Web w/ persistent HTTP connections, BitTorrent |
| 4 | transmit all data at once | FTP |
| 5 | mixture of 1 and 3 | BitTorrent with rate limit imposed by client application |

The receiver side buffer `b2` (between the TCP layer and the application layer) constrains the maximum number of outstanding bytes the other end is allowed at any given time instant. In theory also buffer `b4` can limit the maximum outstanding bytes if a lot of bytes are received out of order and the buffer is small. However, this behavior should be rare since in this case it is the sending TCP's congestion window that should get exhausted first. On the other hand, the sender buffer (between the TCP layer and the MAC layer) constrains the maximum number of bytes in the retransmit queue. Consequently, the size of the sender buffer also constrains the amount of unacknowledged data that can be outstanding at any time. We call the first limitation *receiver limitation* and the second one *sender limitation*. If the transmission rate of a connection is limited by a window size (either sender or receiver window limitation), the sliding window of TCP will be consistently smaller than the bandwidth delay product of the path. Figure 5.9 shows a time vs. sequence diagram of a receiver window limited connection. The staircase-like lines indicate the left (lower) and right (upper) limit of the sliding window and the vertical arrows represent data segments that were sent. Since the lines for the data segments transmitted coincide with line tracking the upper limit of the sliding window, the sender is receiver window limited.

Sender and receiver window limitations result in the same observable behavior. We expect that for most transfers in the Internet, the sender buffer to be at least the size of the receiver window. For instance, in most Unix implementations of TCP, the minimum size for the sender buffer is 64 Kbytes, which is equal to the maximum receiver window size when the window scale option (RFC 1303 [68]) is not used. When the window scale option is used, a correct implementation of a TCP stack should resize the sender buffer when receiving the window scale factor of the other side. For example, all current Linux 2.4 and 2.6 versions include sender side buffer size autotuning, so the actual sending socket buffer size (wmem value in the /proc filesystem: /proc/sys/net/ipv4/tcp_wmem) will be dynamically updated for each connection. Moreover, a recent study [88] has observed that 97% of the hosts that support the window scale option used a window scale factor of 0, meaning that the maximum receiver window was at most 64 Kbytes. For these reasons, we focus in this thesis only on the receiver limitation.

Receiver limitations can occur in two flavors: as an intentional or unintentional cause. Intentional limitation is imposed by the receiver application because it is unable to process data

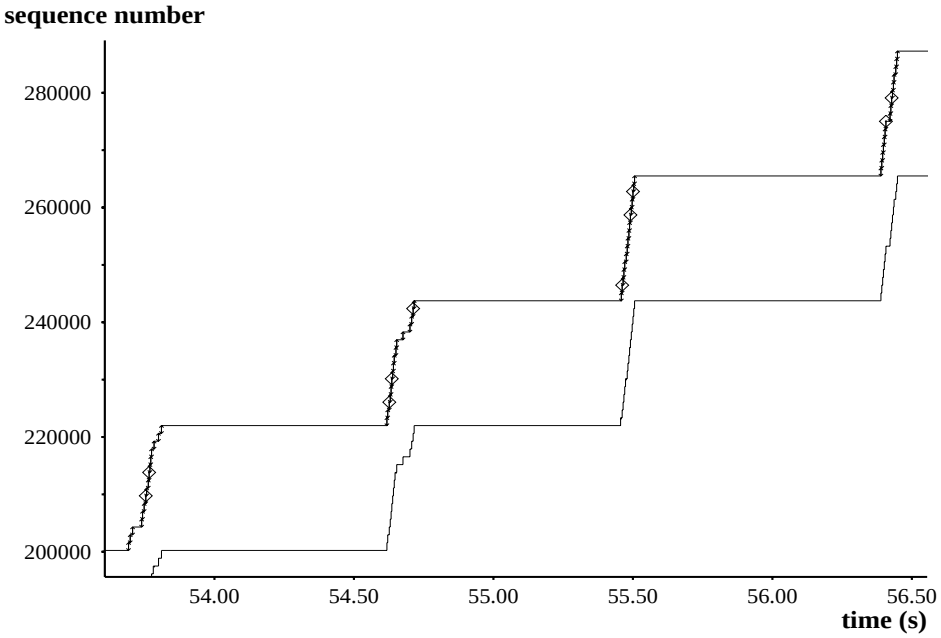



Figure 5.9: A piece of a receiver window limited connection.

as fast as TCP delivers it (recall Section 5.2.1). Unintentional limitation can happen if the receiving TCP advertises an unnecessarily small window by default. For example, consider a situation where two applications communicate with each other through an end-to-end path with high bandwidth and relatively long delay. If the TCPs do not use receiver window scaling[1], the maximum advertised window size will be 64 KB. In this case it suffices to have more than 3.5 Mbit/s of available bandwidth with a RTT of 150 ms (typical for a trans-atlantic path) to be receiver limited with the maximum possible receiver advertised window. We observed another peculiar example related to Windows OS. A computer was connected to the Internet through a wired ADSL access link. When we downloaded a file using this computer, our TCP advertised a window of 65KB. The ADSL access link was in fact a router capable of serving wireless local clients as well. We then connected our computer through the wireless path (WLAN) to the ADSL router and observed that the OS reduced the default advertised window to approximately 32 KB. Some versions of the Windows OS seem to reduce the TCP's default receiver advertised window value in the case of a wireless connection. The reason is unclear. Nevertheless, in this case, the download of the same file became receiver limited unintentionally and some available bandwidth at the access link was unnecessarily left unused.

#### 5.2.2.2 Congestion Avoidance Mechanism: Transport Limitation

Figure 5.10 demonstrates another type of limitation captured from a BitTorrent connection. During this BTP the sending TCP is in congestion avoidance and experiences no losses, thus no limitation by the network (see Section 5.2.3). In addition, the sending TCP does not reach the limit set by the receiver advertised window before the end of the transfer. Hence, the remaining

[1]Some TCP implementations do not use window scaling by default. In addition, certain other implementations can fail to negotiate the scaling factors properly preventing the use of window scaling.

limitation cause is the congestion avoidance mechanism of the TCP protocol that slowly grows the size of the congestion window. This phenomenon may occur when the initial slow start threshold is set unnecessarily low. In this case the sending TCP enters congestion avoidance before experiencing any losses. Consider, for instance, an initial slow start threshold of 32 KB, receiver advertised window of 64 KB, MSS of 1.5 KB, and a path with a lot of available bandwidth (network does not limit the transfer rate). The sender is able to transfer more than a megabyte through this path before reaching the receiver advertised limit. Another example is a relatively short transfer through a path with a lot of available bandwidth and a large scaled receiver advertised window.

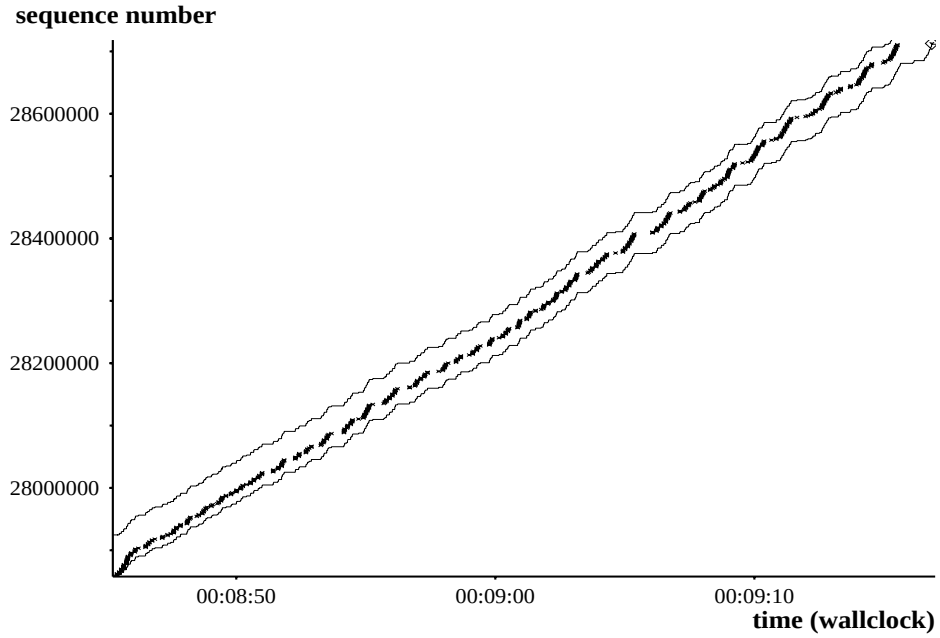



Figure 5.10: A transport limited bulk transfer period within a long BitTorrent connection.

#### 5.2.2.3 Short Transfers: Slow Start Mechanism

There is an additional type of limitation that can be considered as a limitation at the transport layer. This limitation occurs for short connections carrying so few bytes that the connection never leaves the slow start phase. Since it is the slow start behavior of TCP that limits the rate of the TCP transfer we do not classify these connections as application limited.

### 5.2.3 Network

A third category of limitation causes for the throughput seen by a TCP connection are due to the network. We focus on the case where one or more bottlenecks on the path limit the throughput of the connection (see [64] for a study on the location and lifetime of bottlenecks in the Internet). While other network factors, such as link failures or routing loops [113], might impact a TCP connection, we do not consider them in the thesis work as we can reasonably expect their frequency to be negligible as compared to the occurrence of bottlenecks.

For the following, we borrow a few definitions from [102]. We define first general metrics independent of the transport protocol:

**Capacity $C_i$ of link $i$:**
the maximum possible IP layer transfer rate at that link

**End-to-end capacity $C$ in a path:**
$C = min_{1,...,H} C_i$, where $H$ is the number of hops in the path

**Average available bandwidth $A_i$ of a link $i$:**
$A_i = (1 - u_i)C_i$, where $u_i$ is the average utilization of that link in the given time interval

**Average available end-to-end bandwidth $A$ in a path:**
$A = min_{1,...,H} A_i$, where $H$ is the number of hops in the path

We also define two TCP specific metrics:

**bulk transfer capacity ($BTC_i$) of link $i$:**
the maximum capacity obtainable by a TCP connection at that link

**bulk transfer capacity ($BTC$) of a path:**
$BTC = min_{1,...,H} BTC_i$, where $H$ is the number of hops in the path

Note that while available bandwidth is transport protocol independent, $BTC$ is TCP specific. $BTC$ depends on how the TCP connection throughput is affected by other flows. To understand how the available bandwidth differs from the $BTC$, consider the following classical example: A link of capacity $C$ used at 100% by a single TCP connection. When a new TCP connection arrives, the available bandwidth is zero on the link while the bulk transfer capacity should be $\frac{C}{2}$ because TCP backs off and shares the available bandwidth. As in [102], we call the link $i$ with the capacity $C_i = C$ the *narrow link* of the path and the link $j$ with the average available bandwidth $A_j = A$ the *tight link* of the path. These definitions are illustrated in Figure 5.11. Furthermore, we define link $k$ as the *bottleneck link* if it has a bulk transfer capacity $BTC_k = BTC$. Note that while at a given time instant, there is a single bottleneck for a given connection, the location of the bottleneck as well as the bulk transfer capacity at the bottleneck can change over time. One should note also that the bottleneck link is not necessarily the same as the tight link, as demonstrated in Figure 5.12.

If the bottleneck link explains the throughput limitation observed for a given TCP bulk transfer, it can be considered as *network limited*.

We distinguish two different cases of network limitation depending on the type of bottleneck link on the path: *unshared and shared bottleneck limitations*. Intuitively, an unshared bottleneck limitation means that the considered TCP transfer utilizes alone all the capacity of the bottleneck link that is responsible for the throughput achieved. In a shared bottleneck case, there is cross traffic competing for the bandwidth of the bottleneck link. Figure 5.13 shows a time vs. sequence diagram of a connection whose throughput is limited by an unshared bottleneck link. The regular spacing of sent data segments, and similarly of the acknowledgments received, conforming to the bottleneck link capacity is easy to observe. Because of the self-clocking behavior of TCP (sent data packets pace the acks that pace the next data packets etc.), this regular spacing is observable at every point along the path. Figure 5.14 shows a typical xplot of a piece of transfer going through a shared bottleneck. A packet with R on top is a retransmission.

A bottleneck link that limits the throughput of a given TCP connection is commonly experienced as packet losses by the connection when the buffer at the bottleneck link is overrun.

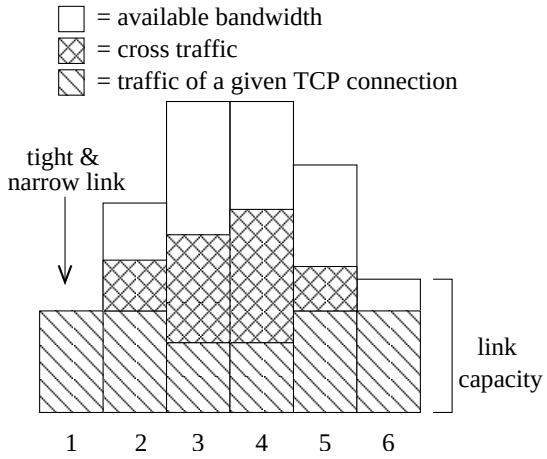



Figure 5.11: Link utilization along a TCP/IP path where the narrow link is the same as the tight link.

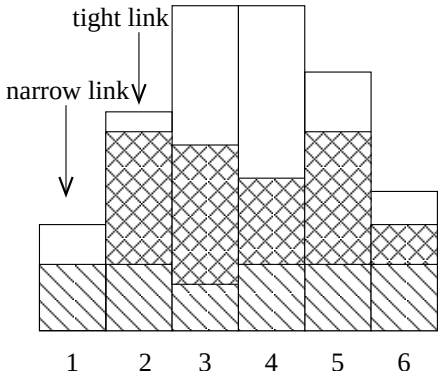



Figure 5.12: Link utilization along a TCP/IP path where the narrow link is not the same as the tight link.

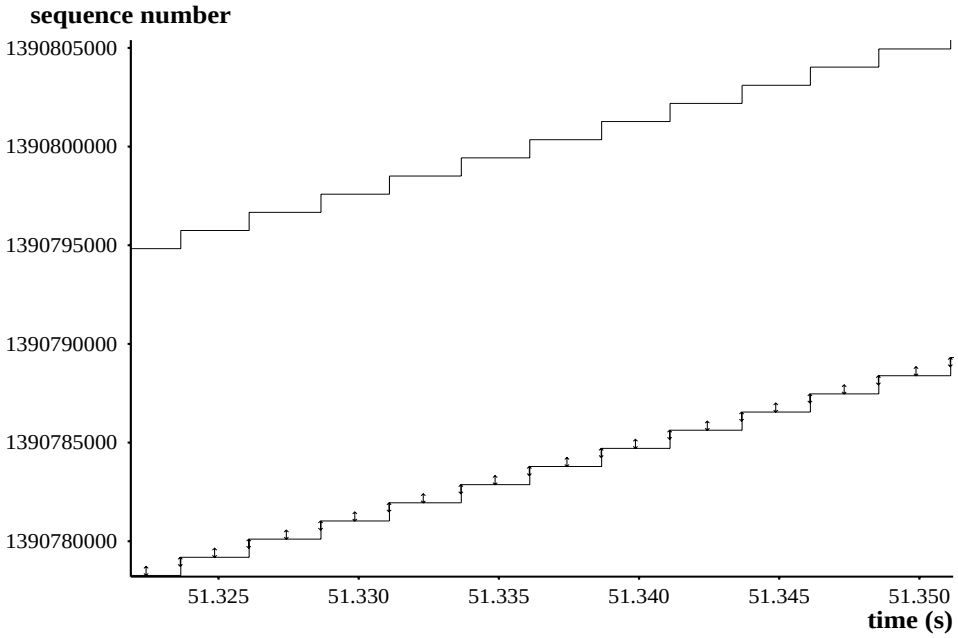



Figure 5.13: A piece of a bandwidth limited transfer where packets are regularly spaced due to the bottleneck link.

However, the packet loss rate by itself does not fully characterize the impact of congestion on the throughput of a given connection for several reasons. First of all, two connections with different RTT values will not see their throughput affected in the same way even if they experience a similar loss rate, as can be seen directly from the TCP throughput formula [87]: $Tput = \frac{MSS}{RTT}\frac{C}{\sqrt{p}}$, where $MSS$ is the maximum segment size of the connection, $C$ is a constant and $p$ is the loss event probability (which is related to the loss rate, while not similar, as it indicates the frequency of loss periods where one or more packets are lost). Second reason is that consequent

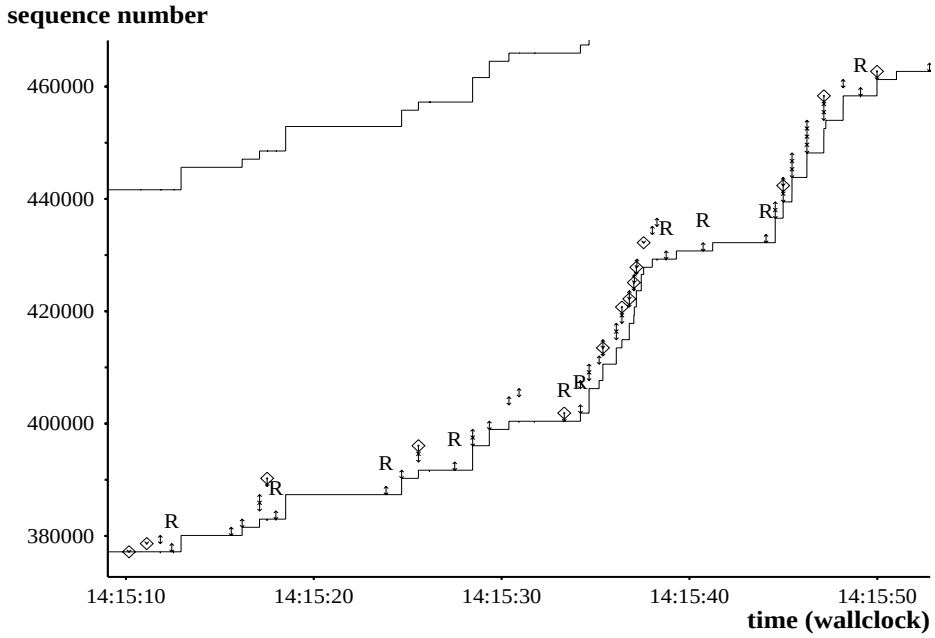



Figure 5.14: A piece of transfer limited by a shared bottleneck link.

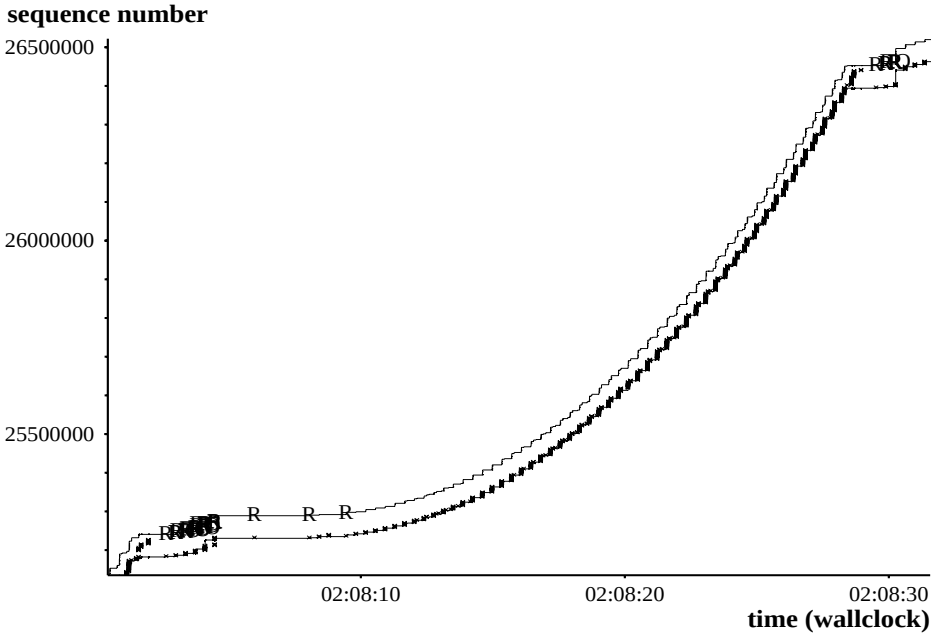



Figure 5.15: Effect of consecutive losses within a BTP of a long BitTorrent connection.

losses may reduce the average throughput clearly more than occasional losses even if the total loss rate is the same. Consider, for example, the piece of a real transfer plotted in Figure 5.15. A bunch of packets are lost consecutively including some already retransmitted packets, which causes TCP to set eventually the slow start threshold very low, namely down to two packets.

As a consequence, TCP enters congestion avoidance early and grows slowly its transmission rate for a long time until again a bunch of packets is lost. This behavior persists during the whole connection. Third reason is that a TCP transfer whose rate is limited by bottleneck link does not necessarily experience any losses. Whether losses occur depends primarily on the depth of the buffer in the bottleneck link and the size of the receiver advertised window. To understand why, let us consider a scenario where a TCP transfer is set up on a given path. The context is the following:

- $MSS$ is the packet size in bytes that the TCP sender uses on the path.
- $RTT$ is the round-trip time when queues on the path are empty.
- Narrow link $i$ on the path has capacity $C_i$ and available bandwidth also equal to $C_i$ (i.e. no cross traffic).
- Link $i$ is also the tight link.
- Receiver advertises a window of size $W_r$ packets (of size $MSS$).

Now, suppose that $W_r > \frac{RTT \cdot C_i}{MSS}$, i.e. the receiver advertised window is larger than the maximum number of packets that can be transmitted through the narrow link per $RTT$. Thus, the narrow link is the bottleneck link and the scenario corresponds to the unshared bottleneck limitation. The sending TCP will continue to grow its congestion window (cwnd) until one of the two things happens: the size of cwnd reaches $W_r$ or there is a loss on the bottleneck link due to buffer overrun. Clearly, it is the buffer size on the bottleneck link that determines which of the two happens. Specifically, if $B > W_r - \frac{RTT \cdot C_i}{MSS}$, where $B$ is the buffer size at the bottleneck link, then the cwnd of the TCP sender will reach the size $W_r$ and experience no losses at the bottleneck link. This scenario described here is the simplest one possible, but the same reasoning applies also to the case of the shared bottleneck limitation. If the buffer of the bottleneck link is not overrun before the TCP sender exhausts the receiver advertised window, there will be no losses.

### 5.2.4 Middleboxes

Middleboxes constitute yet another category of limitations that may occur in a multitude of ways not restricted to any layer. Middlebox in this context means any device that is located somewhere along the TCP/IP path altering intentionally the flow of traffic in such a way that the throughput achieved is modified by it. These "boxes" are typically rate limiters often deployed either at public Internet access points (e.g. WiFi hot spots) in order to let a given user use only a predefined slice of the available bandwidth, or at the edges of a private network to assure enforcement of SLA. Note that, as mentioned earlier, also the application on top of the sending TCP can enforce rate limitations. However, we categorize these cases as application limited transfers.

Figure 5.16 shows an xplot of a piece of an example transfer. This trace was captured at a server located at Institut Eurecom while transferring data from that server to a laptop with a wireless connection on board a Lufthansa aeroplane flying over the Atlantic. It is unsure how data is transmitted from the airplane to ground. We computed the evolution of the RTT which varied approximately between 100ms and 300ms. In [63], the authors say that the latency in

satellite links are in the range of 50-300 ms, which does not overrule the possibility that our data was transferred via a satellite link. We observe that the size of the receiver window oscillates regularly (this behavior persists throughout the entire transfer). It systematically decreases to zero in such a way that approximately half of the time the connection is idle. By estimating the capacity of the path and computing the throughput obtained, we could determine that our transfer obtained exactly 50% of the capacity. It is unclear whether it is the Cisco Aironet 350 series equipment used that is responsible for this shaping, or if there is another device taking care of this. Nevertheless, it is sure that there was a box in the middle of the path rewriting the receiver advertised window values of packets passing by since it is impossible that a web browser on an otherwise idle laptop cannot receive data from TCP layer faster than 23 KB/s.

Another example trace is plotted in Figure 5.17. This one was captured at Institut Eurecom behind a Packeteer [7] traffic shaper whose purpose is to make sure that the traffic conforms to the SLA. Specifically, this box ensures that neither the upstream nor the downstream traffic never exceeds an instantaneous rate of 10 Mbit/s on the 100 Mbit/s link that connects Eurecom to the ISP's backbone. The shaping includes, at least, constant modifications of the receiver advertised window values of packets passing by and delaying of the delivery of acknowledgments. We can observe the effect of receiver advertised window value modifications in the xplot: in the middle of the plot the receiver window is almost entirely closed (the lower and upper lines are very close to each other) and after approximately 21:04:50 the window starts to open up again.

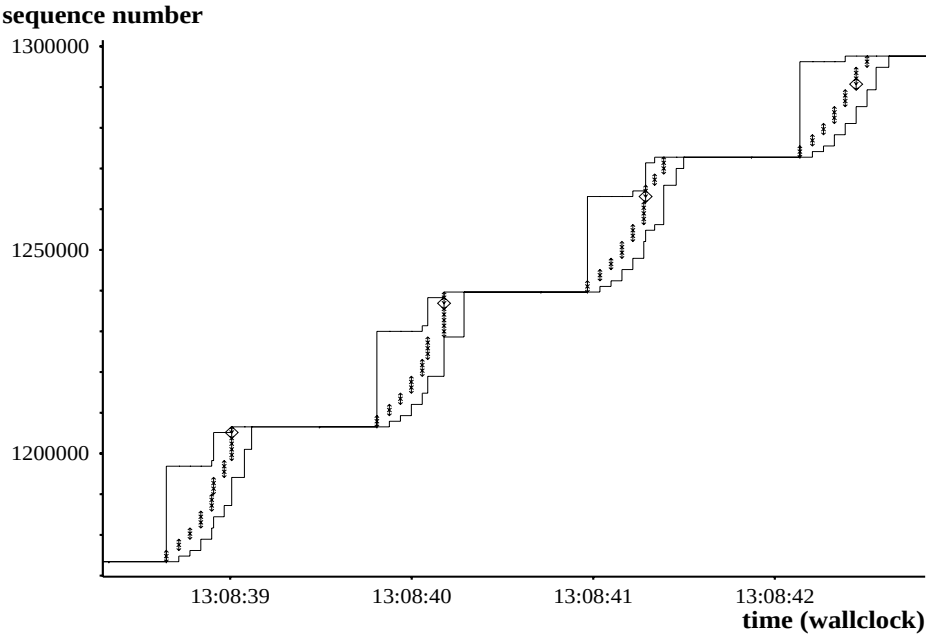



Figure 5.16: Transfer to a wireless laptop on board of an airplane.

These two examples demonstrate that the middleboxes commonly modify the receiver window values to regulate the transmission rates. This kind of behavior can be potentially detected as it deviates significantly from normal traffic behavior. It is, though, possible that the TCP receiver itself actively lowers the receiver advertised window value because the application on top is too slow, as we discussed in Section 5.2.2. However, there is one important difference between these two cases. The middleboxes generally tamper with the individual connections in

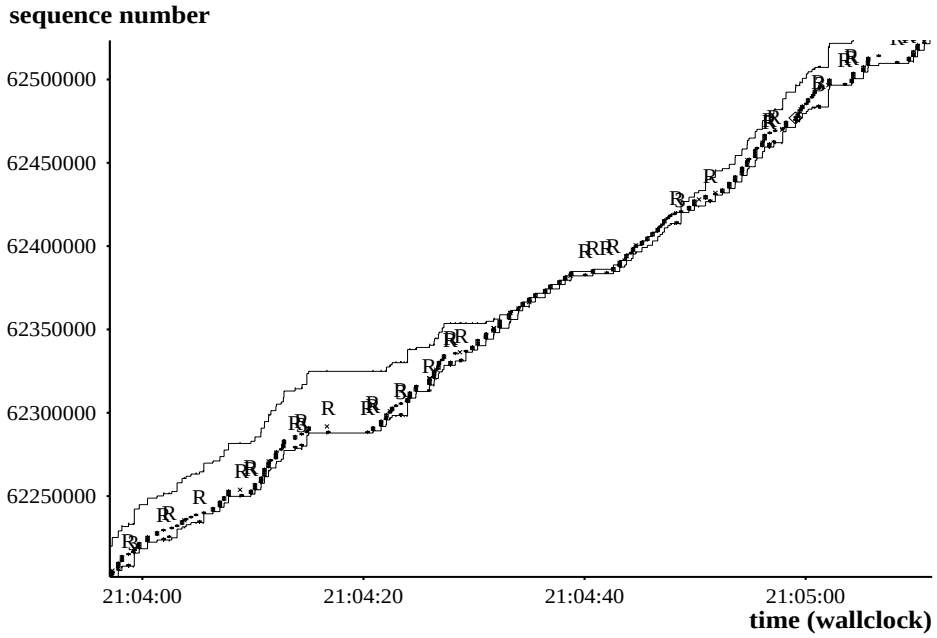



Figure 5.17: Transfer passing through a Packeteer packet shaper.

order to control the total aggregate transmission rate while a slow receiver is concerned only of the transmission rate of the single connection. Therefore, in the case of a slow receiver, it is reasonable to assume that the receiver advertised window value does not fluctuate constantly: once set to sufficiently low value, it stays that way. Instead, the middlebox may need to adjust the rates of individual connections constantly depending on the arrivals and departures of connections in order to maintain a constant aggregate rate.

Other approaches for middleboxes to limit the rates are possible as well. The boxes can potentially also delay or drop packets. However, this type of traffic shaping is very difficult to distinguish from the effects of the network itself. That is why it is very hard to detect the presence of such a middlebox.

## 5.3 Related Work

Given that TCP exists since 1981, it is understandable that the amount of research analyzing TCP is very large. On one hand, almost all research work done on Internet traffic analysis concerns TCP, as it transports over 90% of that traffic. On the other hand, the TCP protocol itself and its behavior has also been extensively studied. In this Section, we classify the most important contributions within this research work in order to position our work properly. We distinguish two main tracks of TCP analysis that we discuss separately: analytical work and measurement-based analysis. In addition, we can identify a large body of research that proposes extensions and improvements for TCP. Finally, we discuss separately the research work related to TCP root cause analysis and its relation to all the other work presented here.

### 5.3.1 Analytical Work: Modeling TCP

There is a vast amount of research done on modeling TCP behavior. Commonly, the analytical models concentrate on the throughput, typically the most important performance measure of TCP. The first simple models of TCP throughput were introduced in [53] [87]. The authors in [87] introduced the now famous square root formula that relates the throughput to the transmitted packet size, RTT, and loss event probability. This model was subsequently enhanced by several works to better take into account the packet nature of TCP transfer (as opposed to the fluid assumptions in [87]), timeouts, and receiver advertised window [96] [22] [109]. While these papers focus mainly on modeling the Tahoe and Reno versions of TCP, the authors of [104] develop models for TCP Vegas.

Other more advanced mathematical methods have also been used in the quest for more accurate models. The above works assumed periodic loss process, i.e. times between congestion events are constant. The work in [17] modeled loss events as a stochastic process. In [55] [92], the TCP congestion window is modeled using Markov chains.

The work in [59] describes a closed queuing network model concentrating on interacting long-lived TCP transfers. In contrast, the performance of short-lived transfers is modeled in [23].

### 5.3.2 Measurement-Based Analysis

In addition to the analytical work on TCP, there has been a growing interest in measurement based analysis on TCP traffic analysis, and Internet traffic in general. Being the dominant transport protocol of the Internet, TCP and its traffic are naturally interesting topics to study when we try to understand the behavior of the Internet. While analytical work can provide very useful input for simulations of certain specific scenarios, measurement-based studies bring knowledge on "what is really out there". It is important to know, for instance, which application traffic is currently dominant or what is the common flow size distribution. Many of the most important results, such as the self-similarity of Web traffic [41], have come from measurement studies. Several tracks can be identified within this research domain.

#### 5.3.2.1 TCP Performance & Deployment Status

Some of the measurement-based work focuses on evaluating the performance of TCP protocol itself or simply measuring evolution of the TCP protocol deployment.

In [83], the authors studied the delay-based congestion avoidance mechanisms of TCP, such as the one used by TCP Vegas, and came to the conclusion that the correlation between an increase in RTT and a packet loss event is not strong enough to allow TCP to reliably improve throughput. The authors of [29] studied also the main premise of such congestion avoidance techniques by looking at correlation between the number of outstanding bytes and the round-trip time. They concluded that the correlation is often weak.

The authors of [97] developed a tool, TBIT, to test web servers for number of properties about the specific TCP implementation run on that server. They used the tool to get statistics about the version of TCP deployed and conformance for several TCP options. The work was repeated after four years in order to see the evolution [88].

The work in [16] lists some guidelines for evaluating the performance of TCP. The author discusses pitfalls when simulating and emulating TCP, and doing live measurement studies.

The work in [85] describes Web100, an architecture and infrastructure that exposes otherwise hidden TCP protocol events. By using Web100 infrastructure on an end host, it is possible to query precisely the state of each TCP connection, which, the authors claim, provides helpful input for research, education, and network diagnosis. Indeed, we have used Web100 to validate some of the results in this thesis (Sections 7.1.4 and 6.2).

#### 5.3.2.2 TCP and the Network

Another part of the measurement-based research work around TCP concerns the interaction between the network and TCP traffic. This interaction is visible in the characteristics of the observed traffic. Several studies have focused on packet reordering and losses due to the network [26] [15] [72] [32]. TCP traffic is bursty in nature. The burstiness and its harmful impact on the performance of TCP transfers has been studied in [30] [73] [74]. A similar study on "flights" of packets in TCP connections is presented in [105]. The authors argue that large time scale flights are indicators of excess resources in the network due to large buffers or large available bandwidth. Stationarity of BitTorrent transfers over TCP has been evaluated in [115] where the author presented a tool to split a TCP transfer into stationary regimes. A large scale study on the constancy, also a notion referring to stationarity, of several Internet path properties including TCP throughput is presented in [119].

TCP traffic characteristics can also tell us something about the infrastructure of the Internet. The authors of [78] and [47] present algorithms to infer the capacity of a TCP/IP path from TCP traffic traces. [37] describes a method to infer queue sizes of access links by establishing a TCP connection and analyzing the RTT evolution during a transfer and comparing the results with ping.

The RTT experienced by a TCP connection is a combined effect of network infrastructure (link capacities and distance) and the current network state (degree of load). Thus, RTT is an important metrics for the diagnosis of network performance perceived by the end user. When traffic is observed in the middle of a TCP/IP path, a situation common in traffic monitoring [71], or at the receiving end of the path, computing the RTT samples becomes a non-trivial problem. In this situation, when estimating the RTT using techniques based on observing bidirectional traffic, the RTT needs to be computed in two parts (see Figure 5.18): (i) delay between observing a data packet and the corresponding ACK packet and (ii) delay between observing the ACK and the data packet the transmission of which the ACK triggered. The main challenge of such a technique is to be able to compute the second part (`d2` in Figure 5.18), i.e. to associate a data packet to an ACK that triggered its transmission. So far several different techniques to solve this problem have been proposed. These proposals include constructing a replica of the TCP senders state to track current congestion window and in that way make the association between data packets and ACKs [70]. Another method uses TCP timestamp options [116] to make the association. Other RTT estimation techniques based only on unidirectional traffic observations[2] have been proposed in [116] and in [118]. These techniques base the estimation on observing the self-clocking behavior of TCP in the traffic pattern. Finally, a study of RTT variability was presented in [14].

Network tomography is the art of inferring delay and loss characteristics on network-internal links using end-to-end active probes. The idea comes from the Computed Axial Tomography

[2]Unidirectional traffic flows are often observed in backbone measurements due to asymmetric upstream and downstream paths [72].

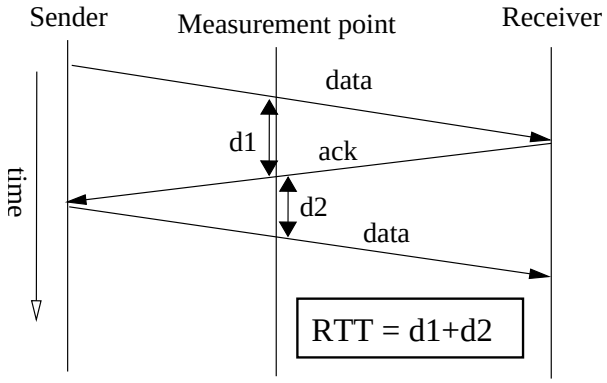



Figure 5.18: Round-trip time estimation in the middle of the path.

(CAT) used in the medical world generate a three-dimensional image of the internals of an object from a large series of two-dimensional X-ray images taken around a single axis of rotation. Packet level network tomography from RTT measurements has been proposed in [114]. The problem of how to efficiently position beacons has been addressed in [80]. In [18], the authors propose flow-level tomography.

#### 5.3.2.3 TCP and Applications

Application performance is typically what users care about. Nevertheless, as many of the applications today use TCP to transport data, the performance of the applications is reflected and also often due to the TCP layer. The study in [33] looks at performance of "elastic" applications, i.e. HTTP, SMTP, and POP3. This work analyzes the performance not from the point of view of TCP but rather by looking at the performance perceived by the application.

The work in [91] discusses the Nagle's algorithm invented by John Nagle in 1984 [94] and thereafter integrated in all TCP implementations. In the early days, large amounts of small packets carrying little data were troubling the Internet. Nagle's algorithm addressed this problem by allowing the TCP sender to transmit only at most one small (less than MSS) packet per RTT per connection. This algorithm has some undesired effects on some applications and the authors propose a modification that introduces a tradeoff between the large delays when Nagle's algorithm is enabled and large amounts of small packets when disabled.

The works in [95], [49] and [24] focus on Web performance mainly from TCP point of view. In [95], the performance improvement of HTTP/1.1 [50] [51] over HTTP/1.0 [28] is quantified. The two HTTP versions use TCP in a different way. HTTP/1.0 opens and closes a new TCP connection for each operation, which causes a great overhead when transferring small web objects. HTTP/1.1 introduced persistent connections and "pipelining". Persistent connections allow multiple objects to be transferred over a single connection and pipelining allows multiple requests to be sent without waiting for a response. These two improvements diminish significantly the communication overhead for TCP and the authors show that HTTP/1.1, when pipelining is enabled, outperforms HTTP/1.0 in all their tests. The authors of [49] analyzed the impact of different values of parameters of TCP and HTTP protocols for Web performance. They studied specifically the end-to-end latency experienced by Web clients and the impact on

the Web server in handling the requests. Their goal was to explain how performance of a Web server would change if certain parameters of TCP (e.g. using Tahoe or Reno version of TCP) or HTTP (e.g. persistent, parallel, pipelined connections or not) were tuned. In [24], the authors apply critical path analysis to HTTP/1.0 transactions over TCP Reno to pinpoint the origins of most of the delays experienced when we use Web.

### 5.3.3 TCP Extensions and Improvements

Fair amount of research has been done on extending TCP to better suit specific network conditions. The authors in [77] and [117] discuss the fact that the standard TCP congestion control is not well suitable for high bandwidth-delay product networks. Both of these works propose a new TCP congestion control mechanism that solves the problem. The work in [103] proposes a scheme for automatically sizing the TCP buffers that would be useful in high-performance networks such as grid computing.

### 5.3.4 Root Cause Analysis

We separate root cause analysis of TCP performance from the rest of the related research work. But in fact, it can also be seen as a research topic that cross-cuts most of the other TCP related analysis work. Indeed, as shown in Section 5.2, root cause analysis needs to incorporate also the analysis of the application layer as well as the network layer effects on TCP transfers. Consequently, as becomes clear in Chapter 7, we benefit from a lot of related work in our analysis, especially those described in Section 5.3.2.2, combined with our own techniques to accomplish our objectives. The type of root cause analysis we are interested in is more general than much of the research work presented earlier in this section. For example, we do not necessarily need to know which TCP and HTTP parameters cause a certain throughput limitation. In the first step, we only want to be able to identify this cause by looking at the traffic flow without knowing the version of TCP or HTTP used.

The analytical modeling work on TCP presented earlier in this section generally does not include the impact of the application layer in the analysis but focuses on modeling TCP bulk transfer performance. In addition, the models often need to make a fair amount of assumptions concerning TCP version, occurrence of loss events etc. While this type of work provides important input for protocol design, it is equally important to perform measurement-based studies. Let us take an example on the existing TCP versions to start with. Analytical models always assume a certain version of TCP. While there are well-defined and standardized set of TCP versions available, nobody has accurate information on what is actually deployed[3] at the moment. In addition, an implementation of a standard and the standard itself do not always perfectly correspond to each other.

In [118], the authors introduced the *TCP Rate Analysis Tool (T-RAT)*. Originally, very much inspired by their research work, we started our work on root cause analysis using T-RAT. We implemented our own version of this tool, as it was not publicly available, and experimented with it. Unfortunately, T-RAT turned out to suffer from a number of limitations, which is why we chose to develop our own algorithms. The limitations of T-RAT originate from the fact

[3]The studies in [97] and [88] provided important information but unfortunately these studies were based on interaction with web servers. Today, the popular media suggests that a great majority of the Internet's traffic is peer-to-peer, which questions the value of this study. Furthermore, their method was able to classify only 33% of the servers for the TCP version.

that it was designed to work only on unidirectional traffic traces. While this choice may clearly increase the usability of the tool, it also severely decreases the accuracy in certain cases. We discuss in detail the limitations of T-RAT and perform a comparison against our methods in Section 7.4.

## 5.4 Scope of Our Work

We do not treat all possible scenarios in our root cause analysis of TCP traffic. We focus only on the analysis of long-lived connections and neglect the short ones. Previous research work (e.g. in [120] and [23]) has studied the performance of short TCP transfers from measurement and analytical point of view. The slow start mechanism of the TCP protocol is typically considered as the main limitation cause for the short transfers. Our definition of a long-lived connection is such that the connection is not likely to be limited by the slow start mechanism of the TCP protocol. In practice we set a threshold around 100 to 150 KB. We give a more precise definition is given in Section 6.1.

We have also made one important assumption when we devised our root cause analysis techniques. We assume that the traffic analyzed passed through routers using FIFO (First In, First Out) scheduling. One of the crucial components of our root cause analysis techniques imposing this limitation is the capacity estimation tool, PPrate [47]. In Chapter 7 the importance of this metric becomes clear when we detail our techniques. PPrate utilizes packet dispersion techniques to estimate the capacity which assume FIFO scheduling. While this assumption is still true for most of the traffic flowing in the Internet, there are cases in which the assumption does not hold, such as cable modem and wireless 802.11 access networks. The work presented in [82] discusses these scenarios and the way they can affect the packet dispersion techniques.

CHAPTER 6

# Applications and Their Interaction with TCP

In Chapter 5 we looked at the way data is transferred between applications through the TCP/IP stack and network path. We identified potential root causes that limit the throughput of a TCP transfer. These causes are found in the application, TCP, and network layers. In this chapter, we focus on the application layer.

We saw in Section 5.2.1 that the various applications on top of TCP behave quite differently when looking at a single connection. Specifically, the xplot examples demonstrated how they differ in the way they provide data to the TCP socket of each connection for transmission (continuously or sporadically). The applications also differ in the number of connections they establish (one or many simultaneously). Consider, for instance, a P2P application such as BitTorrent that may establish several tens of connections, of which only a subset is actively used at any time for transmitting data. On the other hand, FTP establishes one control and one data connection. FTP transfers "all the data at once", whereas BitTorrent connections alternate between transfer (unchoked) and idle (choked) periods.

The related work we presented in Section 5.3 showed that much research has been done to characterize TCP traffic in the Internet. Much of this work focuses on the TCP and IP layers (see Section 5.3.2.2), but ignores the effects of the application on top. This is generally because of simplicity, especially in the case of analytical models. On the other hand, the interactions of specific applications with TCP are analyzed separately (see Section 5.3.2.3). However, when seeking to explain certain characteristics of TCP traffic, e.g. burstiness [30] [73], it is crucial to account also for the effects of the application. It is even more the case today, as the application set is such a heterogeneous one, that the effects can be unpredictable. Therefore, our work on the identification of root causes for TCP throughput limitation must also address separately the impact of the applications operating on top of TCP.

We present in this chapter, in Section 6.1, an algorithm that allows to isolate the two types of periods, introduced in Section 5.2.1, *bulk data transfer periods* (BTP) and *application limited periods* (ALP) within a TCP connection. This algorithm enables us to quarantine the impact of the application layer on the throughput of a TCP transfer. Hence, for the subsequent analysis of the BTPs (Chapter 7), we can focus only on the identification of the limitation causes due to the transport and network layers.

Our algorithm to identify BTPs within a TCP connection is *generic* in the sense that it works regardless of the type of application on top of TCP. Furthermore, this algorithm enables a quantitative evaluation of the impact of the application on the throughput achieved for a given BTP. The algorithm processes bidirectional TCP/IP headers passively collected at a single measurement point. It may not always be possible to capture the traffic in both directions, e.g. in the backbone where connections may have asymmetric upstream and downstream paths [70]. Nevertheless, we argue that unidirectional traces are often not sufficient for in-depth analysis. We validate the algorithm in Section 6.2.

We argued that it is important, not only for root cause analysis of TCP throughput, but also for other types of analysis on TCP traffic, to account for the effects due to the application. We show in Section 6.4 through a sequence of case studies/examples that unless the effects of the application are filtered out, studying the end-to-end path and traffic characteristics, namely throughput and delay, can produce biased results. In these case studies, we apply the algorithm to a variety of traces each of which contains traffic generated by a single application. Most of these traces are extracted from the same public set of traffic traces of an ADSL access network recorded in 2004 [3] (Location 4 traces).

In Section 6.5, we further study the BTPs and ALPs and demonstrate that different applications have a very different impact on the underlying flow of TCP packets, and that we can capture this difference through the properties of the BTPs and ALPs. Thus, the different periods could potentially serve as a kind of signature of the application nature.

## 6.1 Isolate & Merge (IM) Algorithm

The examples in Section 5.2.1 illustrate the diverse ways the application can influence the shape of the TCP traffic. Given such a diversity, it is quite challenging to design a generic algorithm that separates BTPs from ALPs since the application may interfere on very different time scales.

### 6.1.1 Context

We call our algorithm for identifying BTPs and ALPs the Isolate & Merge (IM) Algorithm due to the way it proceeds. The algorithm is generic in that it can be applied without any calibration to traffic from any application. Additionally, it does not depend on the version of TCP used. Instead, the algorithm relies only on observing generic behavior common to all TCP versions. We define a TCP *connection* as a sequence of packets having the same source-destination or destination-source pairs of IP addresses and TCP port numbers. The IM algorithm processes only connections consisting of at least 130 data packets: Connections with fewer than 130 packets are very likely to be dominated by the TCP slow start algorithm and therefore convey little information about the TCP/IP data path for future analysis. We have chosen the number of packets to be at least 130, since a TCP sender that starts in slow start needs to transmit approximately 130 data packets (assuming a MSS of 1460 bytes) in order to reach a congestion window size equal to 64 Kbytes, a typical size of a receiver advertised window [88]. Thus, we define the minimum required size of a BTP to be 130 data packets. We also define as *short transfer period* (STP) a sequence of packets that contains fewer than 130 data packets and whose rate of transfer is not application limited. We use the term *transfer period* (TP) to refer to either a BTP or STP. The IM algorithm identifies BTPs for a single direction of a connection at a time, since a TCP connection may have two-way data transfers.

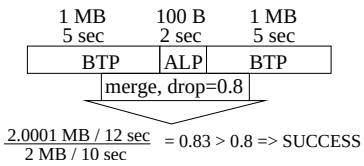



Figure 6.1: Successful merger.

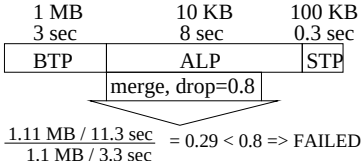



Figure 6.2: Failed merger.

### 6.1.2 Procedures

The IM algorithm consists of two phases: First, it partitions the connection into TPs separated by ALPs. In the second phase, the algorithm attempts to merge two consecutive TPs including the ALP that separates these two TPs in order to create a new BTP.

In order to understand the reasoning behind the merging phase, let us consider how the ALPs differ from the TPs. ALPs reach by definition a lower throughput than the TPs, because the application prevents TCP from fully using the network resources. Thus, the application interference is visible as a lowered throughput. The merging phase is needed because, after the isolate phase, a connection may be divided into many BTPs and STPs separated by very short ALPs. It would be often desirable to combine these periods into one long BTP for subsequent analysis if the effect of these short ALPs on the overall throughput achieved is small. For the above reasons, the procedure of merging periods is based on comparing the throughputs of the periods involved in the merger.

The mergers are controlled with the threshold parameter $drop \in [0, 1]$. Figures E.6 and E.7 demonstrate successful and failed mergers, respectively. Periods can be merged if and only if the throughput of the resulting merged BTP (total bytes divided by total duration) is higher than the *drop* value times the throughput of the TPs combined together excluding the ALP in the middle (sum of bytes of TPs divided by sum of durations of TPs). In this way, the *drop* parameter value limits the maximum amount of application interference, i.e. throughput decrease, within the resulting merged periods. Hence, by selecting a specific value of the *drop* parameter, the user can choose the desired maximum amount of application interference allowed to be present in the resulting BTPs that can then be used for further analysis. The algorithm for merging periods proceeds in an iterative manner, which ensures that eventually all possible mergers, and only those allowed, are performed.

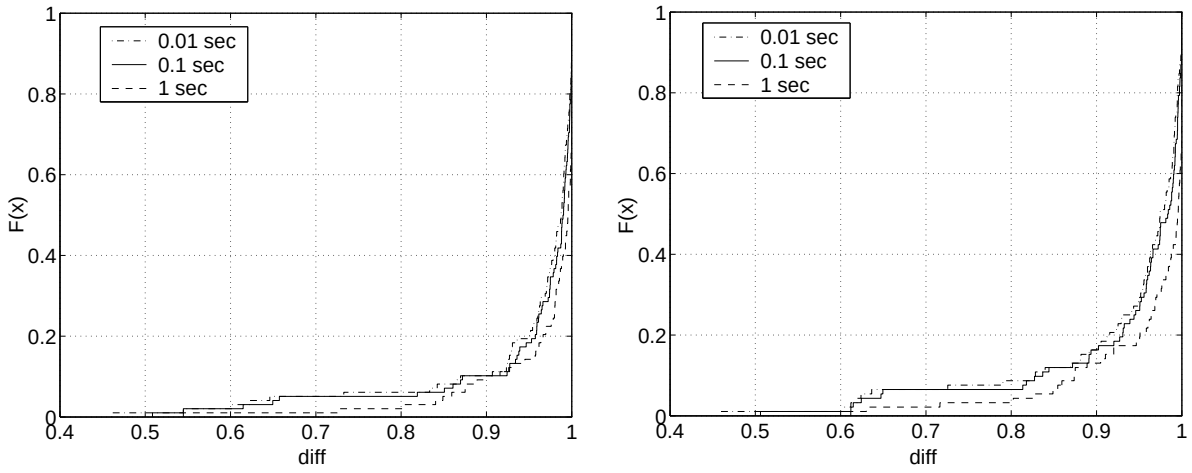
Experimenting with different values of the *drop* parameter allows for a quantitative analysis of the application impact on the throughput achieved, which can provide interesting input for characterizing the application behavior, as we show in Section 6.5. The procedures corresponding to these two phases are called *isolate* and *merge* and are presented in detail in Appendix C.

## 6.2 Validation

The IM algorithm was validated using the Web100 software [85] that allows querying of the state variables of active TCP connections locally on a Web100-enabled machine. Specifically, we investigated the correctness of the *isolate* procedure of the algorithm, which forms also the basis for the functioning of the *merge* procedure. If the BTPs, STPs, and ALPs are not correctly identified, merging them afterwards produces equally faulty results. We generated traffic with

a BitTorrent client by uploading to other clients of a large and popular torrent. We recorded packet headers with `tcpdump` and simultaneously sampled the current state of application write buffer for all active connections with Web100. Both measurements were done at the uploading client's machine. The measured Web100 variable gives the current number of bytes of application data buffered by TCP that is waiting for a first transmission (i.e. it corresponds to the buffer `b1` in Figure E.5. We ran our IM algorithm on the collected `tcpdump` data and compared the results to those from Web100.

We computed two binary time series for each observed connection that contained at least 130 data packets: one from the output of our algorithm and another from the Web100 data. A binary time series sample per predefined time window was generated. A zero value signifies that the sample belongs to an ALP and one to a BTP. In the case of the IM algorithm, we output a one whenever the time window was *mostly* within a transfer period identified by our algorithm. In the case of Web100 data, we output one whenever the average amount of bytes in the application write buffer was worth at least MSS during the time window. A zero was output otherwise. We compared these two time series sample by sample and computed the fraction $diff$ of matching samples.




(a) $drop = 1$ (b) $drop = 0.95$

Figure 6.3: CDF of $diff$, the fraction of matching samples, for the periods.

We analyzed 99 connections amounting to 11.7 GB of transfer in total. Each connection was broken on the average into 279 periods by our algorithm. Figure 6.3(a) shows a CDF of $diff$ for these connections. The periods identified by the *isolate* procedure of the IM algorithm, i.e. those with $drop = 1$ (no merging), were used. Overall the match is very good but not perfect. One reason for this can be the failure to capture small time-scale variations due to lack of accuracy: The ability to query the Web100 variables and to write the data on the disk is limited by the speed of CPU and disk I/O. We were able to achieve a maximum sampling rate of approximately 67 samples/s, i.e. one sample each 15 milliseconds, during the experiments. As the changing rate of the measured variable is in fact limited only by the arrival rate of packets

Table 6.1: Trace characteristics.

| **traffic type** | BitTorrent | eDonkey | FTP data | SSH | Gnutella | HTTP(S) | FastTrack | WinMX |
|---|---|---|---|---|---|---|---|---|
| **port numbers** | 6881-6889 | 4661,4662 | 20 | 22 | 6346,6347 | 80,443 | 1214 | 6699 |
| **duration** | 4d 22h | 4d 22h | 18d 22h | 7d | 18d 22h | 4d 22h | 18d 22h | 18d 22h |
| **packets** | 31M | 44M | 9M | 3.6M | 8M | 14M | 20M | 13M |
| **bytes** | 19GB | 20GB | 7GB | 2.9GB | 2GB | 9GB | 14GB | 5GB |
| **cnxs** | 150K | 1.6M | 5.9K | 48K | 410K | 590K | 360K | 6.3K |
| **cnxs carrying >10KB** | 30K | 23K | 1.1K | 670 | 8.0K | 53K | 11K | 3.3K |
| **cnxs with BTPs** | 10K | 5.5K | 390 | 442 | 940 | 3.2K | 5.6K | 480 |
| **bytes in BTPs (*drop*=1)** | 2.9GB | 690MB | 4.3GB | 2.7GB | 560MB | 4.5GB | 7.0GB | 150MB |
| **bytes in BTPs (*drop*=0.9)** | 7.4GB | 3.0GB | 5.2GB | 2.8GB | 1.0GB | 4.8GB | 11GB | 1.2GB |
| **avg BTP size (*drop*=1)** | 640KB | 550KB | 2.9MB | 4.8MB | 590KB | 1.6MB | 770KB | 290KB |
| **avg BTP size (*drop*=0.9)** | 850KB | 780KB | 13.4MB | 5.9MB | 1.2MB | 1.9MB | 1.9MB | 3.7MB |
| **avg BTP dur. (*drop*=1)** | 38s | 2m 23s | 55s | 14s | 51s | 35s | 1m 47s | 27s |
| **avg BTP dur. (*drop*=0.9)** | 1m 45s | 4m 14s | 4m 31s | 17s | 2m 26s | 51s | 5m 13s | 6m 7s |

which can be easily ten times higher than our sampling rate, we failed to capture all dynamics with Web100. On the other hand, the accuracy of `tcpdump` timestamping is not perfect either. Consequently, as Figure 6.3(a) shows, the coarser the granularity, the better the accordance because of smoothing of the smallest time-scale variance in both time series. Means of $diff$ are 0.9713, 0.9570, and 0.9517 for time windows 1000, 100, and 10 milliseconds, respectively. For comparison, Figure 6.3(b) shows the same results when using merged periods with $drop = 0.95$, i.e. 5% of maximum drop in throughput due to mergers allowed. The match is not as good as with $drop = 1$, which suggests that the *isolate* procedure correctly partitions a TCP connection into BTPs, STPs, and ALPs, even if the impact of the application to the overall throughput was small.

## 6.3 Data Sets

To obtain the results in Sections 6.4 and 6.5, we apply the IM algorithm to eight application-specific traffic traces because we want to perform per-application analysis on the resulting periods. Except for the SSH data set, all of the application specific traces are extracted from the same original public ADSL access network traces recorded in February 2004 (the first 19 days from Location 4 traces in [3], two traces per day, 15 minutes each). The description of these traces inform that there were a couple of hundred ADSL customers, mostly student dorms, connected to this access network. We filter on the well-known TCP port numbers of the applications of interest in order to produce the application specific traces. This method gives in most of the cases solely the traffic from the expected application except for some cases where well-known TCP ports, such as port 80, are used, for example, by P2P applications to bypass firewalls. The SSH traffic data set consists of `scp` downloads from various locations all over the world to a single destination.

Table 6.1 summarizes the characteristics of the traces. We used threshold $th = 3$ with the Isolate procedure of the IM algorithm (see Appendix C for details about this threshold). Regardless of the application type, BTPs are found only in a small fraction of the connections, which is mostly explained by the large number of small connections and the fact that BTPs are required to contain at least 130 packets. BTPs generally carry the majority of the bytes. Though,

BitTorrent and eDonkey traffic are exceptions with BTPs containing a smaller fraction of the bytes, which can be explained by the fact that these applications often throttle their transmission rates, hence, generating only ALPs. The average size of the connections including no BTPs was below 30KB for all applications except for FTP which had an average of 220KB. Oddly enough, the largest ones of these FTP connections, carrying up to 90MB, appeared clearly to be rate limited by the application sending constantly small packets. These unexpected examples clearly emphasize the need to identify the BTPs even for "bulk transfer applications" such as FTP.

## 6.4 Distortion Due to ALPs on End-to-end Path Studies

BTPs can capture the TCP/IP path properties in a very different way than do the entire connections. If TCP sends at full rate, the effects for the data path (e.g. congestion) and, thus, the behavior observed (e.g. retransmissions), are different from the situation when the application limits the transmission rate. In many cases (network health monitoring, network aware applications etc.), it would be desirable to capture only the effects of the TCP/IP data path excluding the application impact.

In this section, we attempt to quantify what we call distortion in the TCP/IP data path analysis results due to the presence of ALPs. We present two case studies. The first one focuses on the characteristics of the rates, i.e. the throughput achieved on a given data path. In the second one, we perform running estimates of the RTT of connections. Note that our goal is not to demonstrate that connection-level measurements yield necessarily wrong results (especially in the first case study). Instead, through the following examples, we want to underline the fact that one should carefully consider what is exactly being measured when drawing conclusions based on the measurements. There are cases when connection-level measurements are desirable, however, there are also other cases where they can be misleading.

### 6.4.1 Studying Characteristics of Rates

Throughput is one of the key measures of the performance of an Internet application. It can be seen as a manifestation of the underlying TCP/IP data path characteristics at a given time instant. However, if the application controls the transmission rate, such an interpretation is false. In order to demonstrate the difference in measuring the mean throughput on connection level vs. filtering out first the application impact, we compared the mean rates of BTPs within a connection to the mean rates of entire connections. Throughout this study, we used $drop = 0.9$ when identifying the BTPs used in the analysis, which means that we allowed a maximum of 10% of reduction in the throughput of the merged TPs. We computed the ratio $\frac{\left(\frac{connection\_bytes}{connection\_duration}\right)}{\left(\frac{\sum BTP\_bytes}{\sum BTP\_duration}\right)}$, which is the throughput computed for the entire connection divided by the throughput obtained when including only the bytes and durations of the BTPs of the connection. The mean values of the ratios for each application data set are in Table 6.2. These values show that the results can differ a lot depending on the application. The interpretation depends also on the application: For example, while the average download throughput of a BitTorrent BTP might express the average achievable throughput of that specific TCP/IP path, the average download throughput of an entire BitTorrent connection could be interpreted as the average rate a specific peer is providing to another peer. On the other hand, in the case of web browsing using persistent HTTP connections, a difference between these two throughput values could be interpreted as a

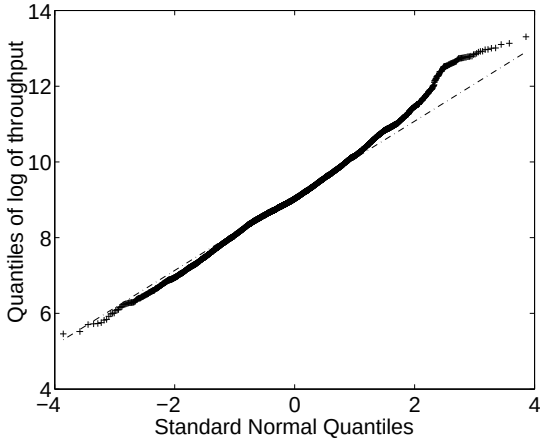



Figure 6.4: Q-Q plot of throughput for the BitTorrent BTPs having $drop = 0.9$.

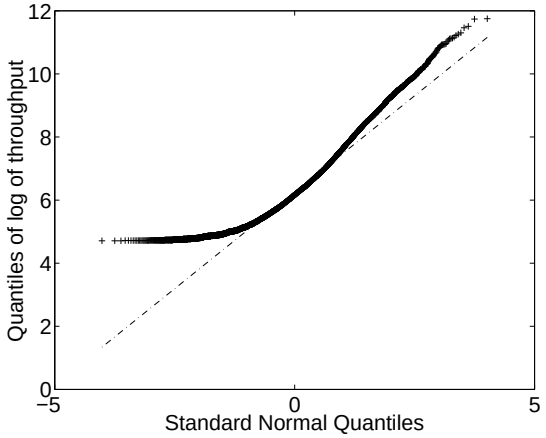



Figure 6.5: Q-Q plot of throughput for the BitTorrent connections transferring at least 100KB.

sign of particular user behavior, e.g. a large difference means that the user spends a long time reading the current page before clicking on a new link on that page.

Table 6.2: Mean values of the throughput ratio.

| **traffic type** | BitTorrent | eDonkey | FTP data | SSH | Gnutella | HTTP(S) | FastTrack | WinMX |
|---|---|---|---|---|---|---|---|---|
| **avg tput ratio** | 0.36 | 0.86 | 0.96 | 0.73 | 0.74 | 0.64 | 0.94 | 0.87 |

In [118] and [20], the authors looked at the rate distributions and observed that they were well described by a log-normal distribution. As in both of those papers, we also use Q-Q plots to visually check the validity[1] of this hypothesis in our data sets. Figure 6.4 shows a Q-Q plot that compares the distribution of log of throughput of the BitTorrent BTPs (using $drop = 0.9$) to the normal distribution. Figure 6.5 shows a similar Q-Q plot comparing the normal distribution to the log of throughput of the entire BitTorrent connections transferring at least 100KB. The log of BTP rates (Figure 6.4) seem visually to agree rather well with the normal distribution since the markers (+) showing quantile to quantile comparison do not deviate much from the dashed line, which passes through the 25th and 75th percentiles, except at the highest quantiles implying a longer tail than the one of normal distribution. The same cannot be stated about the distribution of the rates of the entire connections (Figure 6.5), which is skewed to the right since both ends of the quantiles of the rates are larger than the quantiles of the normal distribution.

The authors found in [118] that the rates and sizes of transfers were highly correlated (coefficients of correlation consistently over 0.8) which they considered as an indication of specific user behavior: the users choose what they download based on the available bandwidth. Table 6.3 contains the coefficients of correlation between the rates (throughput) and sizes (number

[1] In fact, Q-Q plot can only be used to reject the normality hypothesis as it does not constitute a normality test. However, at this point, we only want to do a rough comparison with the normal law.

of bytes transferred) computed for entire connections and BTPs of our data sets. In the case of BTPs, average throughput and sum of transferred bytes was computed for each connection. As in [118], we compared the logarithms of the rates and sizes because of the large range and uneven distribution.

Table 6.3: Coefficients of correlation between log of throughput and log of number of bytes transferred. Only connections transferring at least 100KB were included and $drop = 0.9$ was used when determining the BTPs.

| **traffic type** | BitTorrent | eDonkey | FTP data | SSH |
|---|---|---|---|---|
| **connections** | 0.92 | 0.66 | 0.41 | 0.83 |
| **BTPs** | 0.37 | 0.42 | 0.32 | 0.16 |
| **traffic type** | Gnutella | HTTP(S) | FastTrack | WinMX |
| **connections** | 0.63 | 0.19 | 0.56 | 0.91 |
| **BTPs** | 0.48 | 0.13 | 0.52 | 0.77 |

We also observe some correlation of rate and size throughout our data sets. However, the results vary a lot depending on the application. Furthermore, when comparing connection and BTP-level results for each application, the difference in the degree of correlation varies from negligible (HTTP and FTP) to very large (BitTorrent and SSH). The large difference for BitTorrent traffic is due to two characteristics specific to the application protocol. First, BitTorrent favors fast peers. Fast peers, i.e. the peers having large available bandwidth, are less likely to be choked and, hence, get to transfer more bytes than slower peers. Slow peers are more likely to be choked more often and, thus, transfer less bytes. While this effect is also visible in the correlations when looking at the BTPs, it is amplified when the throughput is computed for the entire connection. The reason is that the choked periods, during which the peer is idle (see Figure 5.7), are identified as ALPs, which, therefore, decrease the connection-level throughput but do not affect the throughput of the BTPs. The second reason are the BitTorrent download connections that are not used simultaneously to upload torrent data. In this case, the upstream data traffic of these connections consists only of periodically sent very small packets containing requests for new chunks and other control messages. This type of traffic generates very low-rate ($< 5Kbit/s$) small-size connections that are identified as ALPs and, therefore, excluded when studying the BTPs. The large difference in the degree of correlation in connection-level and BTP-level results for SSH traffic is explained by the parameter negotiation in the beginning of the connection. This negotiation takes a relatively long time (even up to a few seconds) during which few bytes are transferred. Since the very low throughput during this negotiation phase is controlled by the application, this period is identified as an ALP. Therefore, it only decreases the connection-level throughput. Moreover, the fewer are the bytes transmitted in total, the larger is the impact on the rate of the connection.

Overall, the degrees of correlation seem to be slightly higher for the P2P applications (BitTorrent, eDonkey, Gnutella, FastTrack, and WinMX). One explanation is their ability to download a given file in pieces from several sources simultaneously: the faster the peer, the more it contributes by transferring more pieces, i.e. a larger portion of the file. This behavior may be reflected in both connection-level and BTP-level results depending on whether a transfer of multiple pieces is identified as a single or multiple BTPs. In the case that the application waits

for a download of a piece to finish before requesting a new piece, which can cause an ALP, a transfer of a piece is likely to be identified as a separate BTP. If the application "pipelines" the requests, the transfer of multiple pieces is likely to be continuous and be identified as a single BTP.

The relatively low correlation for FTP and HTTP contradicts with the results in [118] and suggests that users download content regardless of the available bandwidth. In other words, the content is what matters most. The data sets used in [118] date back to 2001 and 2002, which could partially explain this change in behavior. Five years ago, many users were still accessing the Internet using standard modem and ISDN lines with relatively low access capacities and paying for each minute of connection. In such a case, a cost-aware user may not want to wait a long time for a large download to finish and, thus, aborts it or does not start it at all if the available bandwidth is low. An impatient user may do the same. Today, the standard is broadband access (e.g. DSL and cable modems) which is typically flat rate, as is the case for our data sets originating from an ADSL access network. In addition, the typical access link speeds have multiplied. Therefore, there are fewer reasons for choosing the content size based on the available bandwidth today.

We also ranked connections based on their rates. We identified the set of the 50 fastest connections for BitTorrent, eDonkey, and SSH traffic first by computing the throughput for the whole connection and then by computing the average of the BTP throughputs for a given connection when using $drop = 0.9$. When comparing the resulting two sets, we found only 13, 22, and 36 common connections for BitTorrent, eDonkey, and SSH traffic, respectively. These numbers indicate that the ranking is highly affected by the application behavior. As an example, Figure 6.6 shows the corresponding throughput for the common connections in the two sets for BitTorrent traffic. We observe that the ranking is completely different also within the set of the common connections.

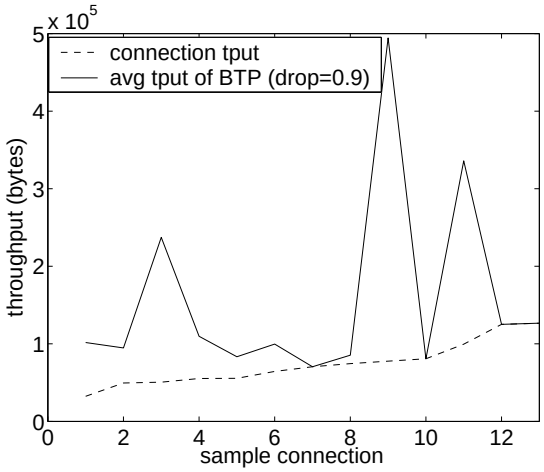



Figure 6.6: Throughput of the common connections in the sets of 50 fastest connections vs. 50 fastest BTPs ($drop = 0.9$) for BitTorrent.

### 6.4.2 Case Study on RTT Estimation

The case of RTT estimation is particularly interesting, since the ALPs may distort the results in yet another way when the measurement point is in the middle of the TCP/IP path. As we discussed in Section 5.3.2.2, in this situation, when using techniques based on observing bidirectional traffic, the RTT needs to be computed in two parts (see Figure 6.7): (i) delay between observing a data packet and the corresponding ACK packet, and (ii) delay between observing the ACK packet and the data packet the transmission of which the ACK packet triggered. The main challenge in such a technique is to be able to compute the second part (`d2` in Figure 6.7). However, as Figure 6.7 shows, there is an additional error `d3` added to the RTT estimate whenever the application delays giving more data to TCP for transmission. This error occurs in every RTT estimation technique that is based on bidirectional traffic observations, such as the TCP timestamp-based technique in [116] (the authors acknowledge the problem in Section 4.1) or the technique described in [70].

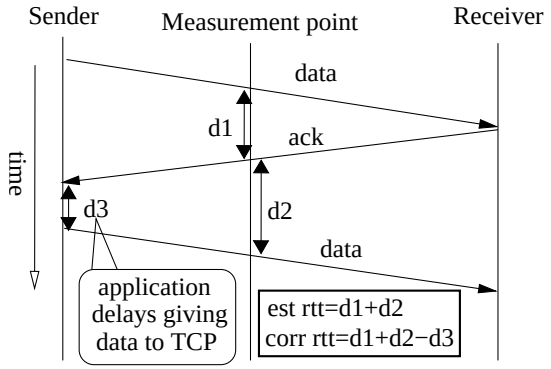



Figure 6.7: Problem with RTT estimation during an ALP.

Other RTT estimation techniques based only on unidirectional traffic observations have been proposed in [116] and in [118]. These techniques base the estimation on observing the self clocking behavior of TCP in the traffic pattern. However, when the application dominates the transfer rate and, hence, controls the pace at which new data packets are sent, this pattern will cease to exist and the estimations will be distorted.

To compute the running RTT samples we used the technique from [116] relying on TCP timestamps or the technique from [70] when TCP timestamps were not available. For each connection, we computed the average for two sets of estimated RTT samples: first, including only all the BTPs, and second, including only all the ALPs. We used $drop = 0.9$. The CDF plot in Figure 6.8 shows how the ratio of the average RTTs ($\frac{\overline{RTT}_{ALP}}{\overline{RTT}_{BTP}}$) is distributed.

We can first observe that the differences between RTTs during BTPs and ALPs are striking: For instance, approximately 18% of the eDonkey connections have ten times longer and approximately 10% ten times shorter average RTT during ALPs than during BTPs. Second, the results vary significantly from one application to another. Many of the inflated RTT values during the ALPs can be due to large values of `d3` (see Figure 6.7). Figure 6.9 shows a short piece of an example HTTP connection that incorporates several BTPs interleaved with ALPs. Similar sawtooth pattern of the RTTs persists throughout the lifetime of the connection. It could be a persistent HTTP connection transferring several objects of a web page. The first RTT sample

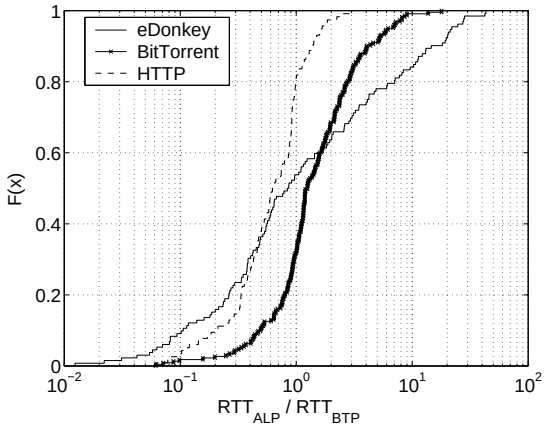



Figure 6.8: CDFs of ratio of the mean RTTs: $\frac{\overline{RTT}_{ALP}}{\overline{RTT}_{BTP}}$.

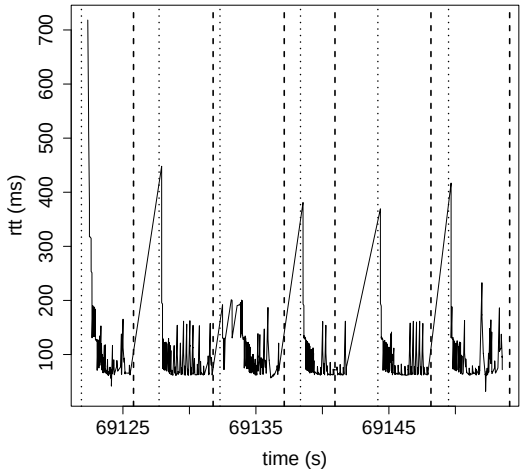



Figure 6.9: Piece of an HTTP connection. Dashed and dotted vertical lines start an ALP and BTP ($drop = 0.95$), respectively.

of each BTP after an ALP is an outlier, which corresponds to the situation depicted in Figure 6.7. The largest component of these inflated RTTs is the delay `d3` by the application in Figure 6.7, which on the average seems to be much larger than the actual RTT. In this example, `d3` is included in the first RTT sample of the BTP because no packets are transmitted and, thus, no samples obtained during the ALPs. As a consequence, the first sample of each BTP should be filtered out as well, if the effects of the application are to be minimized. This example may partially explain why in Figure 6.8 we also observe larger RTTs during the BTPs than during the ALPs of a connection. In order to fully explain all of these particular findings (e.g. why we observe many shorter RTTs during ALPs especially in eDonkey traffic) requires further research

including experimentations with these applications in a controlled environment. This work is beyond the scope of our work here whose goal is to demonstrate the potential impact of the application on the analysis results when studying the TCP/IP path.

## 6.5 What Can We Learn From The Different Periods?

While the primary purpose of the IM algorithm is to filter out the application as a root cause for TCP throughput, it can also be used to learn something about the analyzed traffic. Since we applied the algorithm to application specific data sets described in Section 6.3, we study in this section their characteristics through the properties of the identified TPs and ALPs. We demonstrate that we can capture well-known characteristics but also reveal more surprising facts about different applications. The goal is on one hand to show that not all well-known applications behave as expected and, on the other hand, to give an idea what one could learn about applications without existing pre-knowledge through this kind of study.

### 6.5.1 Properties of the BTPs Identified

Figure 6.10 shows the number of BTPs identified as a function of the *drop* parameter for some selected applications. We notice that often, when *drop* value is decreased, the total BTP count increases. In these cases, new BTPs are formed by merging together *only STPs*. This counter-intuitive effect of mergers is visible for BitTorrent and eDonkey in Figure 6.10 where the BTP count actually increases when the *drop* threshold value is decreased. Figure 6.11, which plots the total TP count as a function of *drop*, confirms the fact that while the BTP count may increase when lowering the *drop* value, the total TP count always decreases. The FTP traffic behaves quite differently from the other data sets. One would expect each FTP connection to consist of a single BTP. However, Figures 6.10 and 6.11 together show that while the number of BTPs in the FTP traffic stays stable for $drop \leq 0.9$, more STPs are merged into them until *drop* is below 0.6 after which mergers are no longer performed. In other words the application can cause a drop of up to 50% of the achievable throughput with this set of FTP traffic. Figure 6.12, which plots the fraction of bytes carried by the identified BTPs as a function of *drop*, confirms that mergers always bring more bytes into BTPs. This effect can be seen from the fractions of bytes in BTPs that are increasing for all data sets when the *drop* value is lowered.

eDonkey clients often limit their maximum upload rate and we observe that majority of the bytes are in ALPs. Also BitTorrent clients often throttle back their upload rate. However, as Figures 6.14 and 6.15 show, eDonkey and BitTorrent clients implement rate limitation differently. While eDonkey clients regulate the rate by varying constantly the amount of bytes passed to TCP (packet sizes in Figure 6.14 are systematically below MSS), BitTorrent clients give data in larger blocks and stay idle between the transfers of two blocks. The effect is also visible as different shapes of curves in Figure 6.12 and the larger amount of BitTorrent STPs in Figure 6.11 (note that the y-axis is logarithmic): The small chunks of data sent by a BitTorrent client generate STPs that are then merged together as BTPs when *drop* value is low enough while the small packets sent by eDonkey client generate connections consisting only of ALPs. In contrast, with FTP the vast majority of bytes are carried by BTPs and the byte ratio in Figure 6.12 is less sensitive to the *drop* value than the other applications indicating that most FTP connections are composed of only a single BTP. In other words, FTP exhibits the purest form of "bulk transfer application" among the studied applications, which is also confirmed by Figure 6.13,

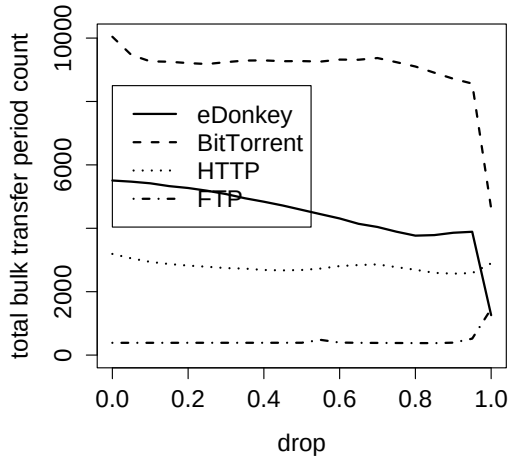



Figure 6.10: Number of identified BTPs vs. *drop*.

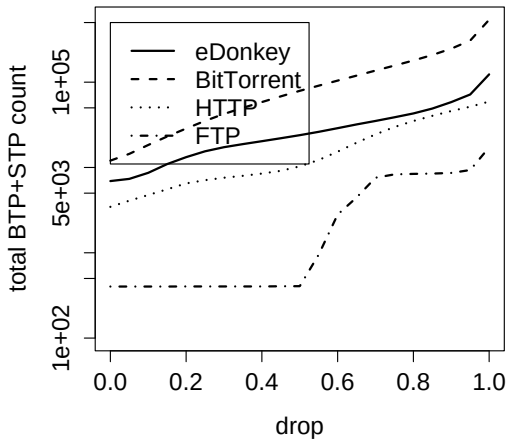



Figure 6.11: Total number of identified BTPs+STPs vs. *drop*.

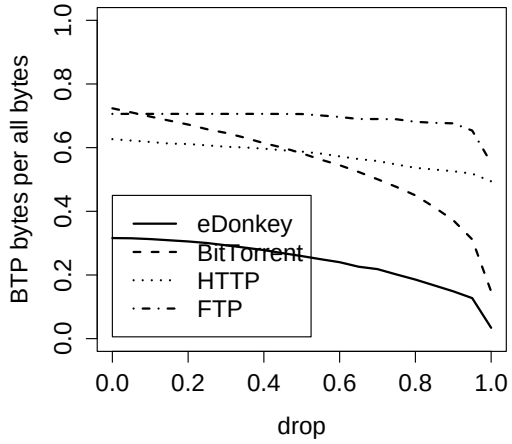



Figure 6.12: Fraction of all bytes in BTPs vs. *drop*.

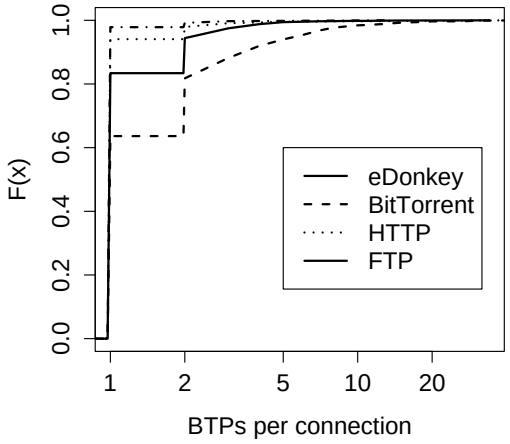



Figure 6.13: Number of identified BTPs per connection, $drop = 0.9$.

which shows the number of BTPs per connection. BitTorrent connections are commonly broken into many BTPs due to the oscillation between the choked and unchoked state of the protocol.

### 6.5.2 Discovering the Nature of an Application

If BTPs manifest the characteristics of the underlying end-to-end network path, ALPs can potentially tell us something about the behavior and nature of the application. In order to see the impact of ALPs on application specific traffic, we investigated the relation between the

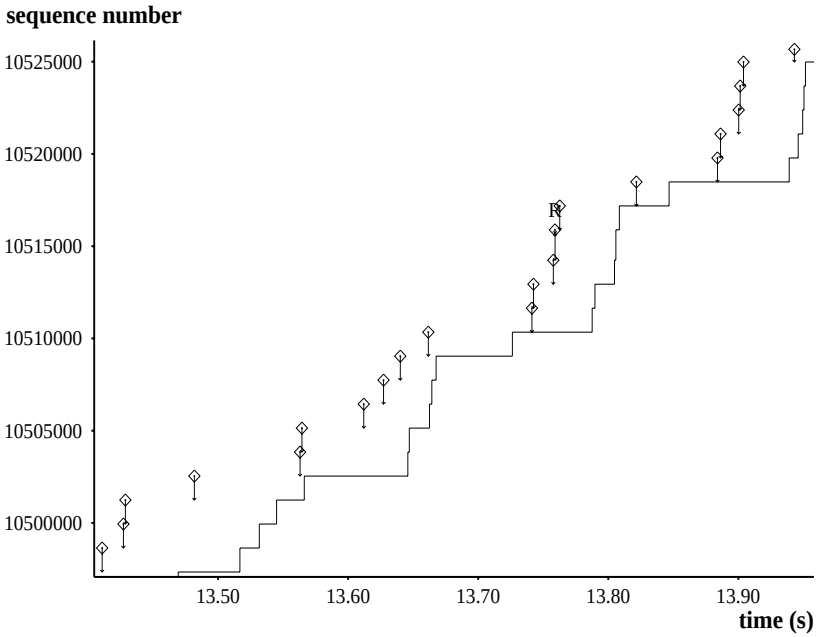



Figure 6.14: Rate limited eDonkey connection.

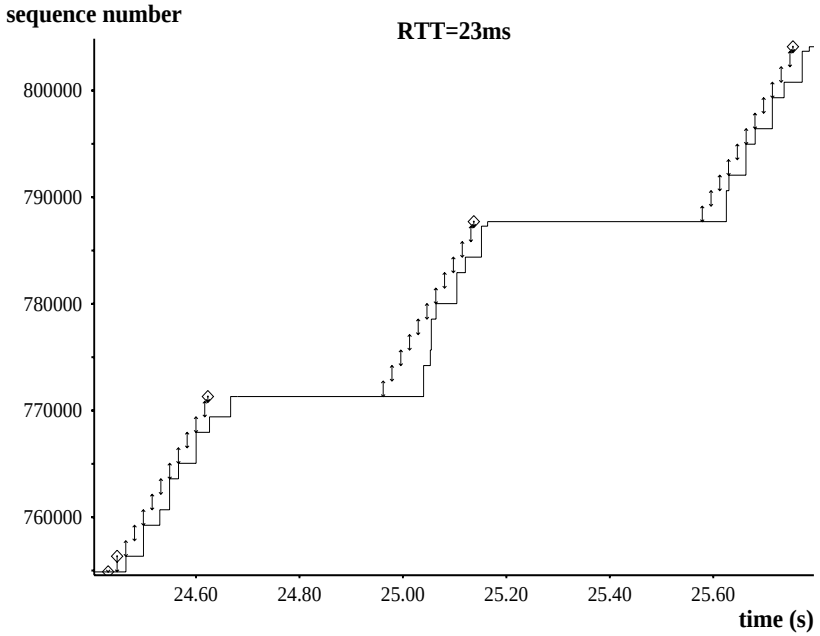
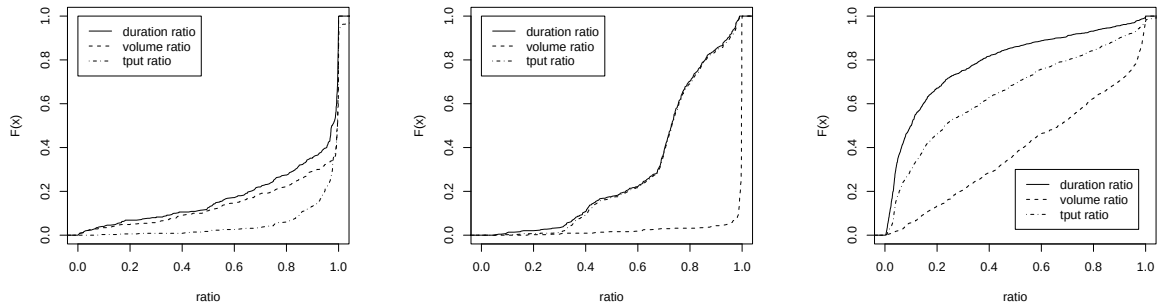



Figure 6.15: Rate limited BitTorrent connection.

durations and volume of BTPs of a given connection with respect to the duration and volume of that entire connection including the ALPs. It turned out that these metrics are able to capture interesting characteristics specific to a given application. For the connections with at least one BTP, we computed ratios of durations ($\frac{\sum BTP\_duration}{connection\_duration}$), volumes ($\frac{\sum BTP\_bytes}{connection\_bytes}$), and

throughputs ($\frac{\left(\frac{connection\_bytes}{connection\_duration}\right)}{\left(\frac{\sum BTP\_bytes}{\sum BTP\_duration}\right)}$) between BTPs (with $drop = 0.9$) and the entire connections for FTP, SSH, and BitTorrent shown (cf. Figures 6.16(a), 6.16(b), and 6.16(c)). Note that these figures do not include the connections with no BTPs at all.




(a) FTP (b) SSH (c) BitTorrent

Figure 6.16: CDF plots of duration, volume, and throughput ratios.

As expected, the overall effect of the application on FTP traffic is minor, hence the dominance of large ratios in the CDF plots of Figure 6.16(a). BitTorrent connections often experience long choke periods that are identified as ALPs, which explains the small duration ratio observed for most connections, i.e. a great majority of the lifetime of a BitTorrent connection consists of ALPs. On the other hand, most of the bytes are in BTPs, though, not as great majority as with FTP, for instance. The reason for this surprising result is that the many STPs (resulting from the way the BitTorrent clients often limit the rates) were not included when computing the ratios. As can be seen from Figures 6.11 and 6.12, much of the bytes carried by the numerous STPs are merged into the BTPs only when the *drop* value is low. The CDF of duration ratio and consequently the CDF of throughput ratio in Figure 6.16(b) reveal the impact of establishing a SSH connection, which requires authentication, key exchange, and negotiation of a set of parameters and, thus, commonly produces an ALP of up to a few seconds in the beginning of the connection. The CDF of volume ratio shows the FTP type file transfer nature (scp) of this application.

We have shown that the ALPs of different applications impact the duration and volume of connections in very different ways. By looking at these metrics, we can learn about the behavior and, thus, about the nature of the application. Potentially, they might even serve as a kind of signature of a specific application or a class of applications. However, we do not investigate this aspect further in this thesis but rather raise it as an interesting open issue for future studies.

## 6.6 Conclusions

We have demonstrated that the application can interfere in various ways with the flow of packets injected into the network. For TCP root cause analysis, this interference is essentially throughput

limitation by the application layer. Our approach is to isolate this limitation cause before analyzing the traffic further for other root causes. We presented in this chapter the IM algorithm that achieves this goal for TCP packet traces containing traffic of any kind of application. The algorithm partitions the traffic of a given TCP connection into BTPs and ALPs. We can then leave out the traffic within the ALPs and proceed to analyze only the BTPs for root causes on transport and network layers, which is the topic of the next chapter.

While isolation of the interference by the application can be considered as the first step in the process of TCP root cause analysis, we argued that it is an equally important procedure to do before studying TCP connection traces in general. We applied the IM algorithm to a variety of different application traffic extracted from public Internet traces and demonstrated that this is indeed the case. We showed how the impact of the application introduces bias in analysis results when studying the TCP/IP path properties such as rate characteristics and RTT estimation. We also studied TPs and ALPs of different applications and showed that such analysis can bring out surprising observations about the behavior of well-known applications and could help to characterize unknown application traffic.

CHAPTER 7

# Analysis of TCP Bulk Transfers

Chapter 6 focused on the root causes originating from the application layer, i.e. TCP rate limitations caused by the application. Our approach is to partition each TCP connection into application limited periods (ALP) and periods not limited by the application, called bulk transfer periods (BTP). In this way, we are able to filter out the impact of the application layer and concentrate on the transport and network layer limitation causes. In this chapter, we describe our approach and techniques to infer the root causes for the throughput achieved during BTPs.

In the root cause analysis of the traffic contained by the BTPs, we consider limitation causes due to TCP and network layers as the impact of application has been filtered out from the BTPs. Our approach consists of two phases. In the first phase, we compute for each BTP several *limitation scores*. These scores are quantitative metrics that assess the level of a given limitation cause experienced by the BTP. We compute all these scores from information found in the packet trace. We describe the techniques to compute the limitation scores in Section 7.1.

There is only a single root cause for a throughput achieved at every time instant during a TCP transfer. Unfortunately, it is not trivial to pinpoint which of the causes is mostly responsible for each BTP because a BTP can experience several causes, which is visible as several high limitation scores. Moreover, in many cases one needs to evaluate several of the limitation scores in order to reveal the single most dominant root cause. Nevertheless, this kind of qualitative information is of great interest because the mere quantitative limitation scores are difficult to interpret. For example, one would like to know what a dispersion score of 0.7 signifies and whether it is significantly different from 0.85. That is why the goal of the second phase of our approach is to map scores into a single most dominant root cause. We introduce in Section 7.2 a scheme to perform root cause classification based on the computed quantitative limitation scores.

## 7.1 Quantitative Analysis: The Limitation Scores

We compute the limitation scores by inspecting the packet headers. These packet header traces are captured at a single measurement point along the path from a source to a destination. In order to be as generic as possible, we do not impose restrictions on the location of the

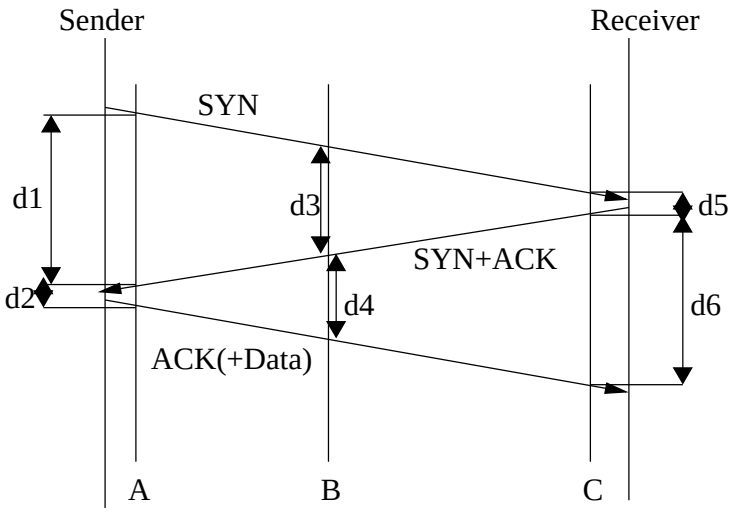



Figure 7.1: Determining the measurement position from the three-way handshake of TCP.

measurement point relative to the TCP end-points. Instead, we infer the measurement point from the traces (Section 7.1.1) and utilize this information to handle the different cases of measurement point location.

Limitation scores are computed with the help of different metrics inferred from the packet trace. These metrics enable us to obtain quantitative scores. We describe the way they are generated in Section 7.1.2 before explaining how the scores themselves are computed in Section 7.1.3. Finally, we validate some of the techniques that we use to compute the scores in Section 7.1.4.

### 7.1.1 Determining Position of Measurement Point

Most of the metrics that we use, are straightforward to compute if the packet trace is captured at the sender side. However, we do not want to limit ourselves to this specific case as, for instance, we may use publicly available traces collected by third parties and for which we do not have exact information about the measurement configuration.

The first problem then is to infer the location of the measurement point. Let us consider the case depicted in Figure 7.1, where the measurement point A is close to the sender while points B and C are not. We can determine the measurement point with respect to the connection initiator (also the sender in Figure 7.1) by measuring and comparing the delay between the SYN and SYN+ACK packets, and the delay between SYN+ACK and ACK packets, referred to as $d1$ and $d2$, respectively, for the measurement point A. We conclude that A is close to the connection initiator if $\frac{d2}{d1} < 0.01$. Note that the connection initiator is not always the sender. Therefore, as the last step, we associate the connection initiator either as the sender or the receiver by simply comparing the directions of the data flow and of the first SYN packet. Consecutively, we conclude whether the measurement point is close to the sender.

### 7.1.2 Metrics Inferred from Packet Headers

In this Section, we only describe the different metrics we compute. Detailed definitions for them are given in Appendix D.

#### Round-Trip Times

We need to compute running RTT estimates for each BTP throughout its lifetime. In the case that our measurement point is close to the sender, we compute the RTT for each acknowledged data packet as the time interval between the timestamps of the last transmission of a data packet and the first acknowledgment for this data packet[1]. In the case that our measurement point is not close to the sender and TCP timestamps are available, we implement either the TCP timestamp-based method introduced in [116], if TCP timestamps are available, or the method described in [72] otherwise. These techniques were described in Section 5.3.2.2.

#### Inter-arrival Times of Acknowledgments

We compute the inter-arrival times of acknowledgments separately for each direction of a connection. The ACKs included in the computation are either acknowledging one or two data packets of size MSS or duplicate acknowledgments. Furthermore, we cancel the effect of delayed ACKs by dividing by two the inter-arrival time of ACKs that acknowledge two data packets.

#### Retransmissions

We cannot assume to observe all retransmitted packets twice since the packets may be lost before the measurement point, especially if the measurement point is far from the sender. Therefore, simply counting bytes carried by packets seen more than once is not sufficient. A packet is considered to be a retransmission if (i) the packet carries an end-sequence number lower than or equal to any previously observed one; and (ii) the packet has an IPID[2] value higher than any previously observed values. With the help of the IPID we remove false positive retransmissions caused by reordering of packets by the network that can occur if the measurement point is far from the sender. Similar analysis of out-of-order packets was also done in [32].

#### Time Series of Receiver Advertised Window

We compute a time series for receiver advertised window, which consists of time-weighted averaged values over a given time interval: Each time a packet is received from the other end, the receiver window indication in the packets will be considered as the actual receiver window value until either the end of the time window occurs or the reception of a new packet. This technique is valid if the measurement point is located at the sender side. However, if the measurement point is away from the sender, we virtually shift in time the observed timestamp values by the time delay between the sender and the observation point. For example, in Figure 7.1, when the measurement point is at C, we would shift in time the timestamp values of packets sent by the

[1] We must take into account that packets may be sent multiple times, in the case of losses, and similarly acknowledged multiples times, in the case of lost or piggybacked acknowledgments.

[2] IPID = IP identification number is assumed unique for each IP packet originating from a given sender. However, this number is only 16 bits long and therefore wrap around of this number needs to be taken into account. There are also some implementation differences between OS. Refer to [35] for more details.

receiver by $+\frac{d6}{2}$, which is the estimated time at which this packet should arrive at the sender. $d6$ is obtained from the running RTT estimates and is, thus, continuously updated.

#### Time Series of Outstanding Bytes

Another metric of interest is the amount of data bytes sent and not yet acknowledged at a given time instant. Since the computation is done by inspecting both directions of the traffic, we need to take into account the location of the measurement point.

If the measurement point is close to the sender, we produce the time series by calculating the difference between the highest data packet sequence number and the highest acknowledgment sequence number seen for each packet and then averaging these values over a time window in the same way that we do for the receiver advertised window values.

If the measurement point is away from the sender, we do the computation by shifting in time the timestamp values of arriving packets. For example, in Figure 7.1, we would shift the timestamp values of data packets arriving from the sender at C by $-\frac{d6}{2}$ and of acknowledgments arriving from the receiver at C by $+\frac{d6}{2}$.

The computed value relates very closely to the outstanding bytes values of the TCP sender at a given time instance. However, it is not 100% accurate in some specific cases but close enough for our limitation tests. We discuss more about the accuracy in Section 7.1.5.

### 7.1.3 Limitation Scores

We compute four different limitation scores: receiver window limitation score, burstiness score, retransmission score, and dispersion score. The first two ones are used to identify receiver limitation through the advertised window on the transport layer. The two other scores are used to identify different cases of network limitation, i.e. limitations by unshared and shared bottleneck links. Here we present the semantics of each score and the way they are computed. In Section 7.2 we perform the mapping of the scores into the limitation categories described in Section 5.2.

#### Receiver Window Limitation Score

We use two time series to compute the receiver window limitation score: the outstanding bytes time series and the receiver advertised window time series. Both of these time series are computed using a time window equal to the minimum RTT of the connection observed[3]. In this way, we ensure that we capture also rapid dynamics in the receiver advertised window (e.g. see Figure 5.16). The difference of the values of these two time series indicates how close the TCP sender's congestion window is to the limit set by the receiver window. Figure 7.2 shows plots of these two time series from an example BTP extracted from a BitTorrent connection which is receiver window limited approximately the first 80 seconds.

Specifically, for each pair of values in the two time series, we compute their difference and generate a binary variable with value one if this difference is less than $lb * MSS$ and zero otherwise, where $lb$ is a small value (typically $lb \in \{1, 2, 3\}$). The receiver window limitation score is the average value of the resulting binary time series for the analyzed bulk transfer period.

[3]If the RTT is less than 10ms, we use 10ms as the time window in order to guarantee that the computation is not too time consuming.

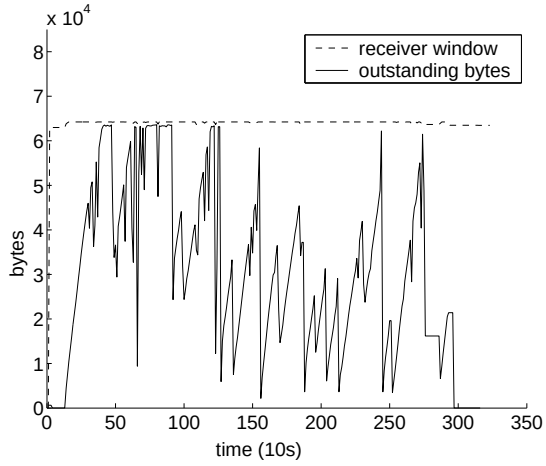



Figure 7.2: Time series of outstanding bytes and receiver advertised window for a BitTorrent connection. Values are computed using 10 second time windows.

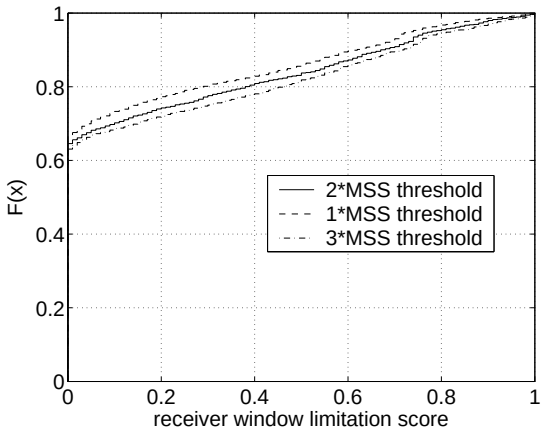



Figure 7.3: CDF plot of receiver window limitation score with threshold $lb \in \{1, 2, 3\}$.

Figure 7.3 shows a CDF of the receiver window limitation score for different values of the $lb$ threshold[4]. Clearly, the choice is not critical as the shape of the curve remains practically the same for $lb \in \{1, 2, 3\}$.

### Burstiness Score (b-score)

Recall from Sections 5.2.3 and 5.2.2 that the outstanding bytes of the TCP sender can reach the limit of the receiver advertised window in two different cases of rate limitation:

- When the buffer of the TCP receiver is too small and the transfer is, thus, receiver limited.
- Transfer is network limited and the link buffers on the path, and specifically on the bottleneck link, are large enough.

We introduce the burstiness score, a.k.a. b-score, in order to distinguish these two cases. The difference of the behavior between these two cases is reflected in Figures E.8 and E.9. If the receiver limits the throughput, the sending TCP needs to wait for acknowledgments for a burst of packets during a significant part of the RTT before it can send new packets (Figure E.8). In contrast, if there is a shared bottleneck link that limits the throughput, the inter-spacing pattern of packets is smoothed out by the cross traffic at the bottleneck link and a higher throughput would not be achieved even if the receiver would increase the advertised window (Figure E.9). Instead, growing the receiver window would at some point cause buffer overflow at the bottleneck link causing retransmissions of packets and reduction of the *cwnd* value, which would lower the overall throughput.

[4]We analyzed a 10 GB `tcpdump` packet trace of BitTorrent traffic captured at the University of Navarra, Spain. The machine at Navarra was involved in a single torrent and the traffic was recorded once the machine had obtained a full copy of the file and thus only acting as a server (seed in the BitTorrent terminology). Hence, all the traffic was captured at the sender side. The trace contains nearly $60,000$ connections with a total amount of 102 million packets.

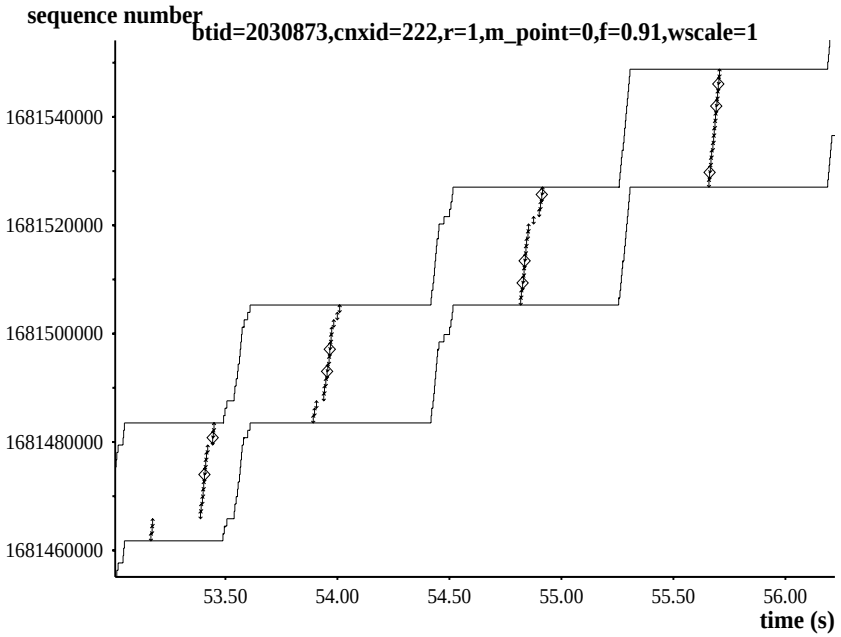



Figure 7.4: Time sequence diagram of a receiver window limited transfer. Note the clear bursty IAT pattern.

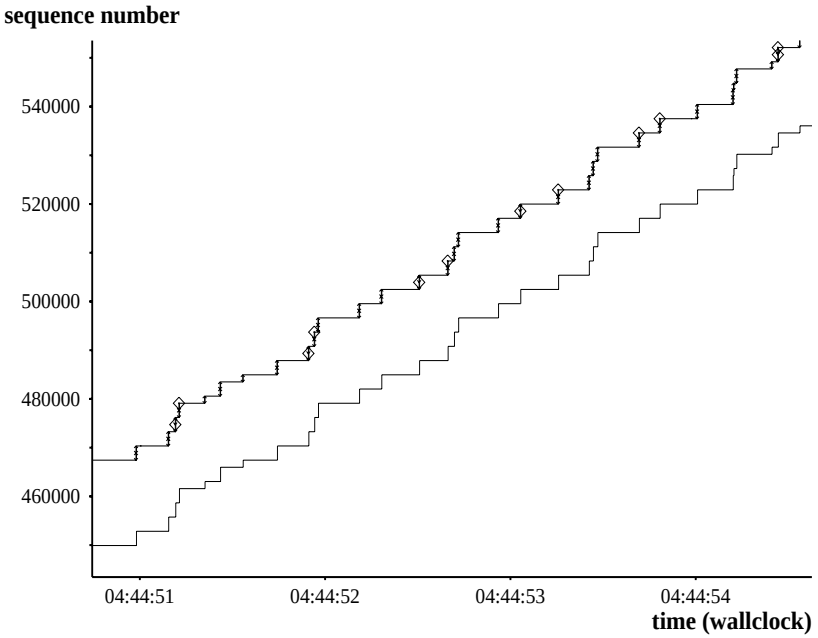



Figure 7.5: Time sequence diagram of a shared bottleneck limited transfer with a high receiver window limitation score. Note the smoothed out IAT pattern.

We first derive the definition for the b-score for the ideal case. We then describe the way we compute an approximation of this ideal case. We compute an approximation, because it does

not involve parameters that would need to be estimated.

Consider the receiver limited scenario depicted in Figure 7.6. Y is the time that the TCP sender waits because it has exhausted the receiver advertised limit for the outstanding bytes, and x is the time that it takes to send a receiver advertised window full of packets. Hence, the throughput is less than the available bandwidth on the path because of the waiting time y, i.e. the sender does not "fill the pipe" because of y. Thus, we define the b-score in the ideal case as($W_r$ is the average receiver advertised window size divided by MSS, $C$ is the path capacity, refer to Section 5.2.3):

$$\mathcal{B} = 1 - \frac{x}{x+y} = 1 - \frac{(W_r - 1)\frac{MSS}{C}}{RTT} \tag{7.1}$$

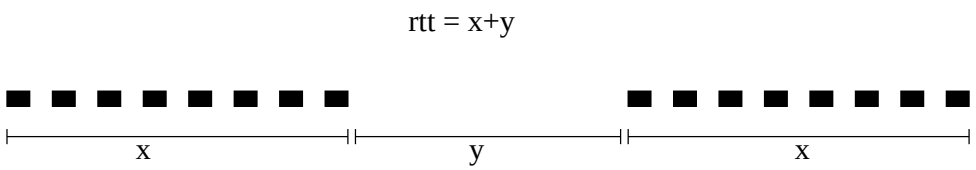



Figure 7.6: Inter-arrival times of receiver window limited transfer. Black rectangles are sent packets and time runs from right to left.

The above definition states that when the waiting time y approaches zero, the b-score also approaches zero[5].

Note that the above definition contains $RTT$ and $C$. It is not always robust to estimate these two values. That is why we propose an easier way to compute an approximation for $\mathcal{B}$. The idea is to observe a certain amount of clearly longer IATs than the average IAT. These long IATs correspond to $y$ in Figure 7.6. These can be observed periodically in the IAT pattern due to the self-clocking of TCP. Thus, we can compute a given percentile of the IAT in order to capture the IAT that corresponds to $y$.

However, the percentile that we should compute varies between different connections and depends on the TCP transfer parameters that affect the IATs, namely the RTT, the receiver advertised window size, and the MSS of the path. For example, suppose that the receiver is advertising a window of 64 KB and the MSS is set to 1.5 KB. In this case, assuming that the receiver is limiting the transmission rate through the advertised window, the long IAT occurs approximately every 43 IATs. Since $\frac{1}{43} = 0.02$, we should search for the 98th percentile of the IAT, i.e. $p = 98$. Using this reasoning we can compute $p$, the correct percentile to be used for each connection separately from the average receiver advertised window:

$$p = 100 \cdot (1 - \frac{1}{W_r})$$

Thus, consider first the following two definitions for the average IAT and $IAT_p$, the $p^{th}$ percentile of the IAT that approximates $y$ of Figure 7.6:

$$\overline{IAT} = \frac{RTT}{W_r}$$

[5]The case when y is zero corresponds to the situation where an unshared bottleneck is the limitation cause.

$$IAT_p \approx y = RTT - (W_r - 1)\frac{MSS}{C}$$

Combining the above definitions with equation 7.1, we get an approximation for the $\mathcal{B}$:

$$\text{b-score} = \mathcal{B} \approx \frac{IAT_p}{\overline{IAT} \cdot W_r} \tag{7.2}$$

The above score is what we compute and use in our root cause analysis. It is easier and more robust to simply measure the $p^{th}$ percentile and mean of the IAT instead of estimating $rtt$ and $C$.

#### Retransmission Score

The retransmission score for a bulk transfer period is computed as the ratio of the amount of data retransmitted divided by the total amount of data transmitted during this period. Note that since TCP may perform unnecessary retransmissions, this score does not exactly correspond to the loss rate. However, we can expect these quantities to be strongly correlated in general and especially if the version of TCP uses SACK.

#### Dispersion Score

The objective of the dispersion score is to assess the impact of the bottleneck link on the throughput of a connection. We define the *dispersion score* as follows ($C$ is the capacity of the path, and *tput* is the average throughput of the BTP computed as $\frac{\text{total bytes transferred}}{duration}$):

$$\text{dispersion score} = 1 - \frac{tput}{C} \tag{7.3}$$

Obviously, correct estimation of $C$, the capacity of the path, is very important for the computation of this score. We use the tool called PPrate which is described in [47].

Let us first consider the case where the network limitation cause is a non shared bottleneck link on the path. The bottleneck is evidently the narrow link of the path (refer to 5.2.3 for the definition). Since the BTP is network limited and the narrow link is not shared, the dispersion score should be close to zero. In all other cases, including the shared bottleneck limitation, the dispersion score is greater than zero.

If we consider the semantics of the dispersion score in the case of rate limitation caused by a shared bottleneck link, we can distinguish two cases. In the first case, the bottleneck link is still the narrow link but it is now shared. Thus, the dispersion score represents the share of the capacity that the cross traffic obtains at the narrow link during this bulk transfer period. Conversely, 1−dispersion score is the share the connection itself obtains. In the second case, the bottleneck is not the narrow link. Thus, the dispersion score does not represent any more the share obtained at the bottleneck.

### 7.1.4 Validations

We validate the computations of the receiver window limitation score in this section. The paper in [47] contains assessment of the accuracy of the capacity estimation of a path. As for the b-score, we perform detailed evaluation of the b-score value in different conditions in Section 7.2.

The retransmission score is computed in a straightforward way and we do not see any reason to validate it.

To perform the validation on the sender side, we ran BitTorrent as seed on a Web100 enabled machine – in the same way as for the IM algorithm described in Section 6.2. We queried the state of the *SndLimTimeRwin* Web100 variable that stores the cumulative time spent in the "Receiver Limited" state. In Web100, TCP is defined to be in this state when transmission is paused, because the sender has filled the advertised receiver window. Hence, for a given BTP, the value of this variable divided by the total duration corresponds exactly to our definition of receiver window limitation score.

We also performed similar validations of the more complex case of the computation of the receiver window limitation score where the measurement point is located in the middle of the path or at the receiver side. In this case, the algorithm needs to virtually "shift" the measurement point to the sender based on the RTT estimates. In order to perform a similar study using Web100 as for the sender side, we used NISTNet to delay incoming ACK packets to increase the RTT. The amount of added delay was changed every hour. We used the following delays: $400, 600, 1000, 1200, 1400, 1600, 1800, 2000$ ms. As the delay by NISTNet [5] was added for only the incoming ACK packets after they had already been captured with `tcpdump`, the measurement point lay in the middle of the path or, in case of long added delay, closer to the receiver.

Figure 7.7 shows CDF plots of the receiver window limitation score computed from Web100 measurements and by our algorithm for the case where the measurement point lies at the sender side. We observe that the difference between the distributions is negligible. The corresponding CDFs for the case where the measurement point is located away from the sender are plotted in Figure 7.8. As expected, the discrepancy is clearly larger but reasonable. Our algorithm seems to overestimate the scores in many cases. Figure 7.9 shows a CDF plot of the absolute difference between these two scores for both measurement point locations. We see that in approximately 50% of the cases the match is perfect for both scenarios. When the measurement point lies at the sender, our algorithm makes an error of less than 10% in 90% of the cases. When the measurement point is not at the sender, the error is less than 20% in 90% of the cases.

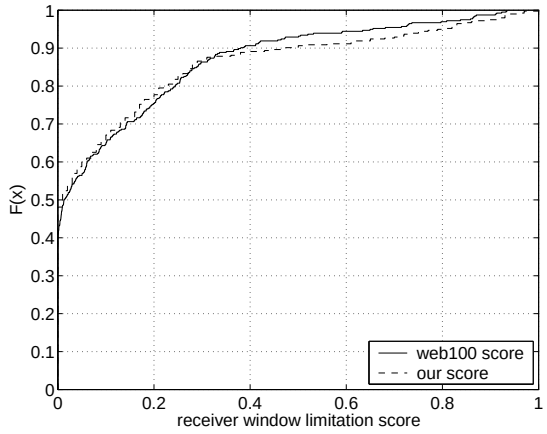



Figure 7.7: CDF plots of the two receiver window limitation scores when measuring at the sender side.

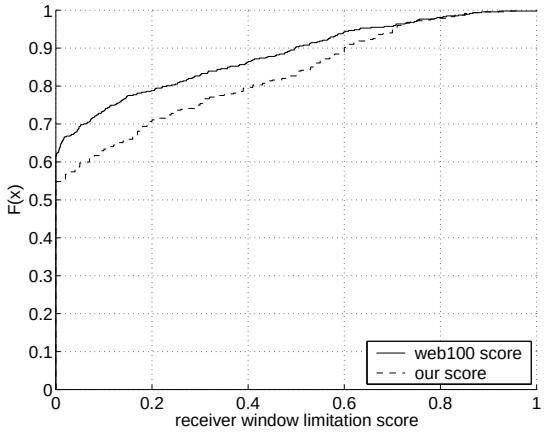



Figure 7.8: CDF plots of the two receiver window limitation scores when measurement point is away from sender.

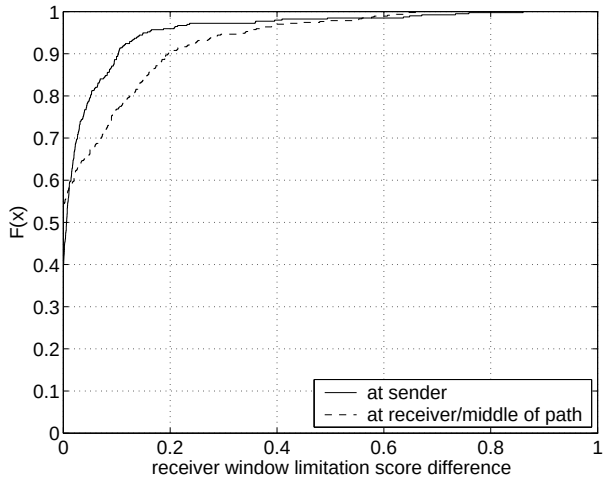



Figure 7.9: CDF of the absolute difference between the Web100 and InTraBase's scores for receiver window limitation.

From the above experiments, we conclude that, in 90% of the cases, we can expect to observe a maximum discrepancy between the actual and estimated values of the receiver window limitation score that remains below 20%.

### 7.1.5 Sources of Errors and Inaccuracy

Virtually shifting the measurement point to the sender side is the most challenging and, at the same time, the weakest component in the computation of the limitation scores. The issues arise from the fact that delays in the Internet do not build up symmetrically at each part of the path. For example, when shifting the measurement point in Figure 7.1 from C to Sender, we shift the timestamp values of data packets arriving from the sender at C by $-\frac{d6}{2}$ and the timestamp values of acknowledgments arriving from the receiver at C by $+\frac{d6}{2}$. $d6$ is obtained from the running RTT estimates and is, thus, continuously updated. However, the delay between the observation of an ACK at C and its arrival to Sender is not exactly the same as the delay between the transmission of a data packet from Sender and its observation at C. Consider, for instance, that there is queue build-up on the upstream path from Sender to Receiver. In this case, the delay caused by the upstream path from Sender to C is clearly larger than the downstream path from C to Sender. Unfortunately, given a single measurement point, it is impossible to take this effect into account. This limitation decreases the accuracy of the receiver window limitation score.

Another issue related to receiver window limitation score concerns the computation of the time series of the outstanding bytes. Consider a situation where TCP experiences losses. For example, packet 7 is lost from a set of packets 6 to 11. In this kind of a situation TCP that is not SACK capable uses go-back-n method after realizing that packet 7 was lost and reduces the congestion window to half and starts retransmitting packets 7 up to 11. Our algorithm maintains the highest data packet sequence number seen so far, i.e. packet 11, whereas the correct value would be 7. In this sense, our algorithm may overestimate the amount of outstanding bytes during the period when TCP is recovering from losses but gets back on track after the recovery. However, if losses are frequent, the size of the sending TCP's congestion window is likely to stay

well below the receiver advertised window and the transfer rate is likely to be limited rather by a bottleneck link. Therefore, we expect the impact of this defect to be small in overall results.

Estimating the capacity of a path from passive measurements is a challenging task. While the authors of [47] show that their tool, which we use also, achieves good accuracy, such a method is never accurate in all cases. Inaccuracy of the capacity estimates are directly reflected in the dispersion score.

## 7.2 Interpreting the Limitation Scores: the Classification Scheme

Quantitative scores for the level of each limitation cause are desirable because they can reveal information about presence of a single rate limitation cause as well as a mixture of causes. However, it is not intuitive how these scores should be interpreted in each case. For example, questions such as "Does a dispersion score of 0.3 mean that the throughput of the BTP is mostly limited by an unshared bottleneck link?" are raised. Typically, we would like to know what is the single most dominant root cause for each BTP, i.e. which cause contributes the most for the throughput achieved. Moreover, it is not necessarily one score that determines each cause, i.e. finding out the most dominant root cause is not a one-to-one mapping between a score and a cause in each case.

### 7.2.1 Scores and Thresholds

As we have computed a set of limitation scores, we propose a threshold based classification scheme for root cause classification of BTPs: Each score has a threshold attached to it. We consider the causes one after another by elimination. At each step, one score is evaluated against a threshold and, as a result, one or more causes are eliminated. The cause that remains in the end is the root cause we are looking for. Thus, in this type of scheme, combinations of limitation scores are evaluated before arriving to certain causes.

Our scheme can be represented with the flow chart depicted in Figure E.10. We introduce a set of thresholds which we calibrate later in Section 7.3. We first describe the different steps in the flow chart and then explain what is the importance of the order of these steps.

The first step is to determine whether the root cause is a bottleneck link that is unshared. For that, we compare the dispersion score against the corresponding threshold *th1*. If the dispersion score is low enough, it means that the BTP achieves a throughput that is close to the capacity of the path. Note that even if the throughput of a BTP is limited by an unshared bottleneck, the average throughput computed for the entire BTP (total bytes divided by total duration) can be smaller than the capacity of the path for two reasons. The first reason is that TCP needs to tune to reach the final transmission rate by growing the congestion window gradually, which lowers the average throughput of the BTP, supposing that the BTP starts from the beginning of the connection. The second reason is that TCP may periodically overrun the buffer at the bottleneck link unless the number of outstanding bytes of the TCP sender is limited by the receiver advertised window (recall the discussion in Section 5.2.3). In this situation, TCP may experience momentarily lower throughput due to loss recovery, depending on the loss recovery strategy. Some TCP versions recover fast (e.g. SACK enabled Newreno, recall Section 5.1.4) with no significant reduction in the throughput, while other versions may suffer much more. Naturally, the degree of reduction in the throughput depends also on the number of packets lost when the buffer is overrun.

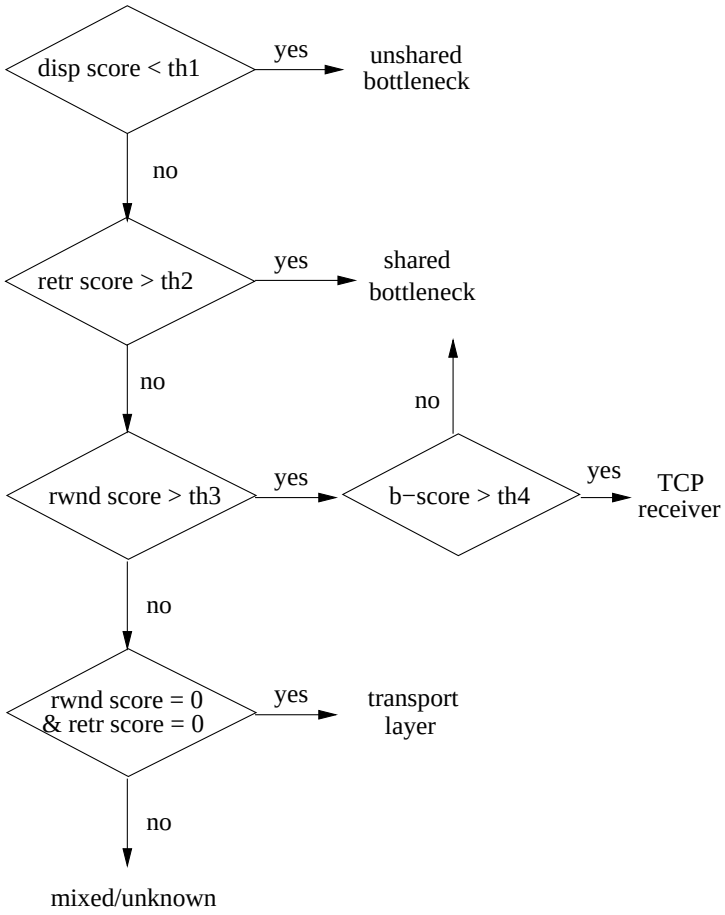



Figure 7.10: Root cause classification scheme.

The second step, comparing the retransmission score to the corresponding threshold *th2*, identifies some of those BTPs that are limited by a bottleneck link that is shared. Remember that retransmissions are clear indications of the presence of a bottleneck link limitation (see Section 5.2.3). Since the first step eliminates those BTPs whose throughput is limited by an unshared bottleneck link, a shared bottleneck link is the obvious cause for the BTPs with high retransmission score.

If no retransmissions are observed and there is no unshared bottleneck link limiting the throughput, our scheme inspects the receiver window limitation score. In the third step, this score is compared to the threshold *th3*. If the score is above the threshold, the transmission rate of the BTP is determined either as receiver limited or limited by a shared bottleneck link. At this point, the scheme considers those cases of shared bottleneck limitations where the amount of outstanding bytes is limited by the receiver advertised window, which, cannot be detected by looking only at the retransmission score. The final separation between the receiver limited BTPs and shared bottleneck link limited BTPs is done by comparing the b-score to the threshold *th4*. The intuition behind this step was discussed in Section 7.1.3.

The final step distinguishes the BTPs whose throughput is transport limited, i.e. limited

by the TCP protocol (either slow start or congestion avoidance mechanism), from those that cannot be classified. We consider those BTPs transport limited that do not experience any retransmissions and no limitation by the receiver advertised window. Otherwise the BTP is classified as limited by a mixture of causes or an unknown cause.

As far as the order of the steps is concerned, inspection of the dispersion score needs to be done before we determine anything about shared bottleneck limited transfers. Thus, steps 2 and 3 need to come after step 1 in Figure E.10. Otherwise a large part of the transfers limited by an unshared bottleneck link would be classified as shared bottleneck limited either because of high receiver window limitation score and low b-score (due to regular inter-spacing of packets) or by a high enough retransmission score. The ordering between steps 2 and 3 does not matter. The step to identify transport limited BTPs could be performed at any point.

### 7.2.2 Accounting for Middleboxes

As we discussed and illustrated in Section 5.2.4, middleboxes typically manipulate the receiver advertised window values of packets passing by in order to shape traffic rates of individual connections. Because of this, in order to detect a presence of such a middlebox, we propose the modification to the classification scheme shown in Figure 7.11.

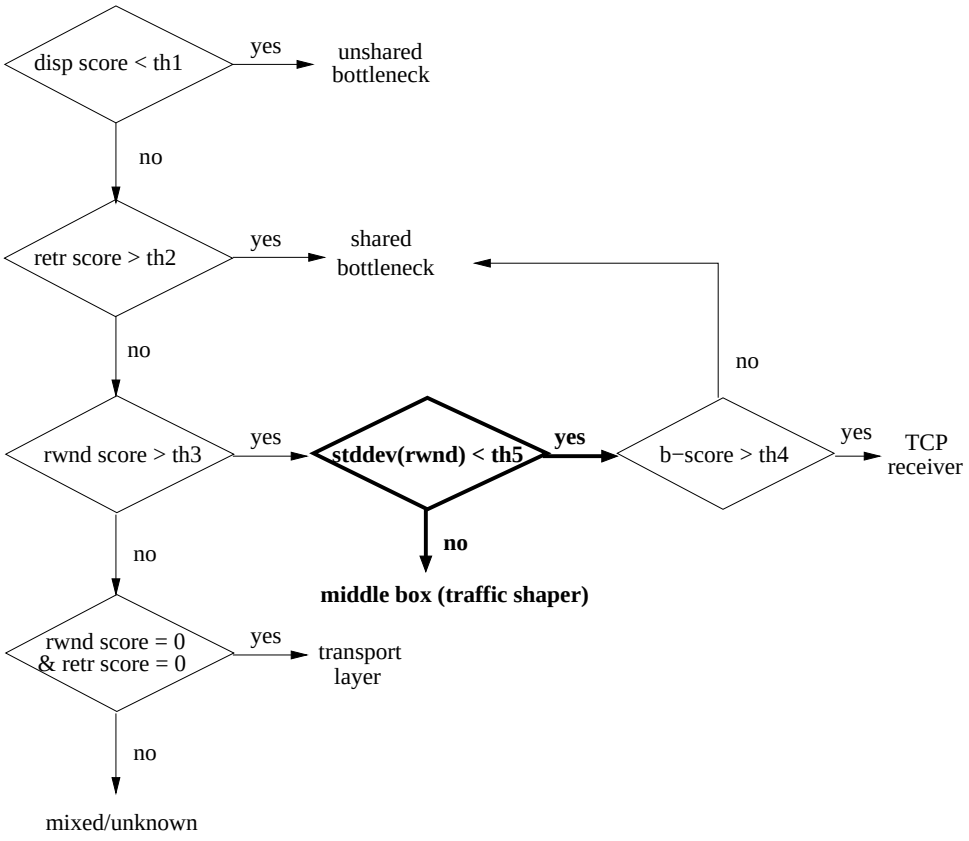



Figure 7.11: Root cause classification scheme with middleboxes taken into consideration.

These boxes cause a high receiver window limitation score. However, if the receiver advert-

ised window size varies wildly, we take this as an indication of a middlebox that shapes the transmission rate. We present here only a possible modification to the classification scheme. Otherwise, we do not further address this limitation cause in this thesis and leave it as a future subject of study.

## 7.3 Inferring the Threshold Values

The flow chart of our classification scheme presented in Figure E.10 contains four thresholds. They need to be adjusted to correct values. It is difficult to find absolute correct values for these thresholds. For example, as we discussed in Section 7.2.1, the right value for the dispersion score varies between transfers and depends on many things including the size of the transfer, the TCP version used, and path characteristics such as delay and buffer sizes at links. Nevertheless, we present in this section a method to infer these threshold values from measurements. We try to be as general as possible, but any such method is always unable to capture all the dynamics of the Internet. Our goal is to generate reference traffic of different categories each of which can be characterized by our limitation scores. The problem can then be stated as follows: How to be sure to generate traffic of a particular category if finding out the parameter values that characterize this traffic are the problem to be solved in the first place? Simulations are not the answer because they fail to capture the diversity of the Internet traffic. The best we could then do is to set a context where we generate the desired type of traffic with a high probability. We discuss more about the limitations of our scheme and the consequences of these limitations in Section 7.3.7.

### 7.3.1 Experimentation Setup

We set up several experiments to generate real traffic that is representative for one of the three different rate limitation causes: unshared bottleneck, shared bottleneck, and receiver limitations. This traffic was then analyzed from the point of view of the scores in order to infer appropriate thresholds for each of them.

The traffic itself was generated by FTP downloads initiated from a machine at Institut Eurecom called metrojeu. The same machine also recorded the traffic traces. We selected all the functional FTP mirror sites for the Fedora Core Linux distribution [6]. The number of selected servers by country are listed in Table 7.1. We selected randomly a new server from the list whenever the previous download was finished. We downloaded each time the same set of files of different sizes (SRPM directory) with a total amount between 40 to 60 MB[6]. The number of simultaneous downloads was controlled in order to produce unshared and shared bottleneck limited traffic. We used rshaper [12] to create an artificial bottleneck link at metrojeu when needed and similarly NISTNet [5] to delay packets in order to increase RTTs.

#### Unshared Bottleneck Limited Transfers

In order to generate traffic that should be limited with a high probability by an unshared bottleneck link, we created an artificial bottleneck and downloaded from one single server at a

[6]The total amount was not constant, because it turned out that the set of files on each mirror server was not exactly the same. For instance, sometimes files were missing.

Table 7.1: Selected mirror sites.

| continent | country | number of servers |
|---|---|---|
| North America | USA East | 28 |
| | USA West | 18 |
| | Canada | 8 |
| South America | Brazil | 2 |
| | Chile | 2 |
| Europe | Austria | 5 |
| | Belgium | 2 |
| | Bulgaria | 1 |
| | Czech Republic | 6 |
| | Denmark | 2 |
| | Estonia | 1 |
| | Finland | 2 |
| | France | 6 |
| | Germany | 8 |
| | Greece | 2 |
| | Hungary | 1 |
| | Iceland | 1 |
| | Ireland | 2 |
| | Italy | 1 |
| | Netherlands | 6 |
| | Norway | 2 |
| | Portugal | 1 |
| | Poland | 5 |
| | Romania | 5 |
| | Russia | 3 |
| | Serbia and Montenegro | 1 |
| | Slovakia | 1 |
| | Slovenia | 1 |
| | Spain | 4 |
| | Sweden | 3 |
| | Switzerland | 2 |
| | Turkey | 2 |
| | Ukraine | 2 |
| | United Kingdom | 5 |
| Africa | Namibia | 1 |
| | South Africa | 1 |
| Asia Asia/Pacific | Australia | 4 |
| | Hong Kong | 2 |
| | Japan | 5 |
| | Korea | 2 |
| | New Zealand | 1 |
| | Singapore | 1 |
| | Taiwan | 2 |

time. We performed the experiments with three different bottleneck link capacities: 0.5, 1, 2 Mbit/s.

### Shared Bottleneck Limited Transfers

To generate transfers likely to be limited by a shared bottleneck, we downloaded from several servers simultaneously. Our download script ensured that downloads from 10 servers were continuously ongoing. We used higher bottleneck link capacities during these experiments: 1, 3, 5, 10 Mbit/s. We checked that the bottleneck link was fully utilized during all the experiments.

**Receiver Limited Transfers**

Transfers typically are receiver limited in the case of a high bandwidth delay product where the sender fully exhausts the receiver advertised window before the ACKs arrive. That is why we used NISTNet to delay packets in order to increase the RTT, as in the validation experiments presented in Section 7.1.4. We experimented with different amounts of delay added to the RTT: 100, 200, 400, 500 ms. The delay by NISTNet was added for only the incoming packets, i.e. the arriving ACKs, after the capture with `tcpdump`. This caused the measurement point to lie in the middle of the path or, in case of long added delay, closer to the receiver. The link capacity at the edge of Eurecom is 100Mbit/s. We checked that during the experiments the aggregate throughput never exceeded this value. Thus, we can conclude that there is no shared bottleneck on our side.

### 7.3.2 Threshold for Retransmission Score

We set the threshold for the retransmission score (*th2* in Figure E.10) to 0.01. It is an empirically justified choice (see the classification results in Section 7.3.6).

### 7.3.3 Threshold for Receiver Window Limitation Score

As for the receiver window limitation score, we choose a threshold of 0.5 (*th3* in Figure E.10). We chose such a value in order to capture those BTPs that experience a majority of the time a throughput limitation by the receiver or by a shared bottleneck link. In other words, we did not want to capture those BTPs that experience mixed root causes during their lifetimes.

### 7.3.4 Threshold for Dispersion Score

The threshold for the dispersion score (*th1* in Figure E.10) distinguishes the BTPs whose transmission rate is limited by an unshared bottleneck link from the rest. That is why we looked at traces captured while downloading from a single server at a time. Figure 7.12 shows the CDF plots of the dispersion score for BTPs generated during these experiments. We notice that a threshold of 0.2 for the unshared bottleneck limitation seems to be a good choice. It is logical that when the capacity of the artificial bottleneck was set to the highest 2 Mbit/s, there were some servers that did not have that much available bandwidth and the cause transitioned to a shared bottleneck. We confirm this in Section 7.3.6 where we look at the classification results using our scheme for each of the experimentation data sets. We also observe up to 10% of BTPs, depending on the trace, having a dispersion score below zero. This means that the capacity is sometimes underestimated by PPrate.

### 7.3.5 Threshold for B-Score

The b-score threshold (*th4* in Figure E.10) separates the receiver limited BTPs from those that are limited by a shared bottleneck link. Therefore, to determine a suitable value for the threshold, we analyzed the traffic from the experiments where we generated traffic corresponding to these two rate limitations.

Figure 7.13 shows CDF plots of the b-score for BTPs from three different types of experiments with 10 simultaneous downloads: experiments where an artificial bottleneck was set up with different capacities, experiments where different amounts of delay were added . We selected

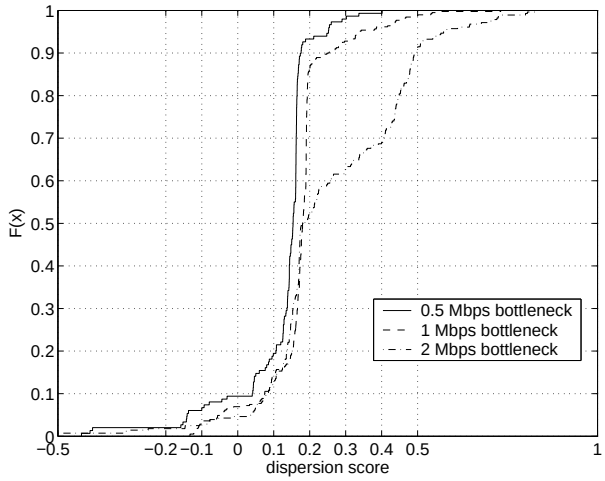



Figure 7.12: CDF plots of the dispersion score when downloading a single file at a time.

for this analysis only those BTPs that had a receiver window limitation score above 0.5, retransmission score below 0.01 and dispersion score above 0.2, that is, the ones that would require the use of the b-score in the classification process.

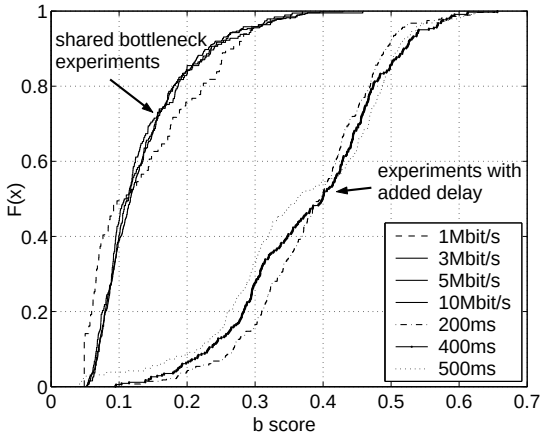



Figure 7.13: CDF plots of the burstiness score when downloading multiple files simultaneously through a shared bottleneck link or with added delay.

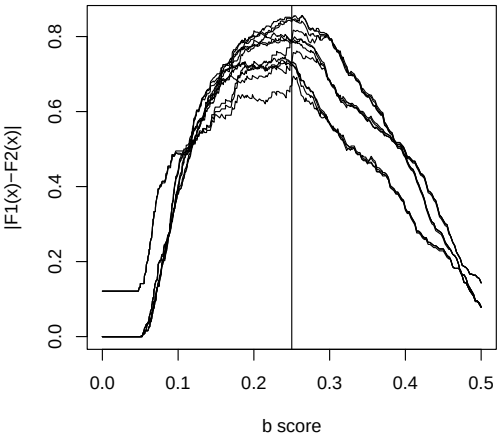



Figure 7.14: Difference of CDF plots between experiments with an artificial bottleneck and added delay, results with 100ms are excluded. The best matching threshold is found at 0.25 (vertical line).

In Figure 7.14, we computed the differences $F1(x) - F2(x)$ for each pair of CDFs plotted in Figure 7.13 where $F1(x)$ is a CDF from the shared bottleneck experiments and $F2(x)$ is a CDF from the experiments with added delay. By computing the mean of the maxima of all

combinations of $F1(x) - F2(x)$, we obtain the threshold 0.25 that provides the best separation between these two limitation causes. This threshold is plotted as a vertical line in Figure 7.14.

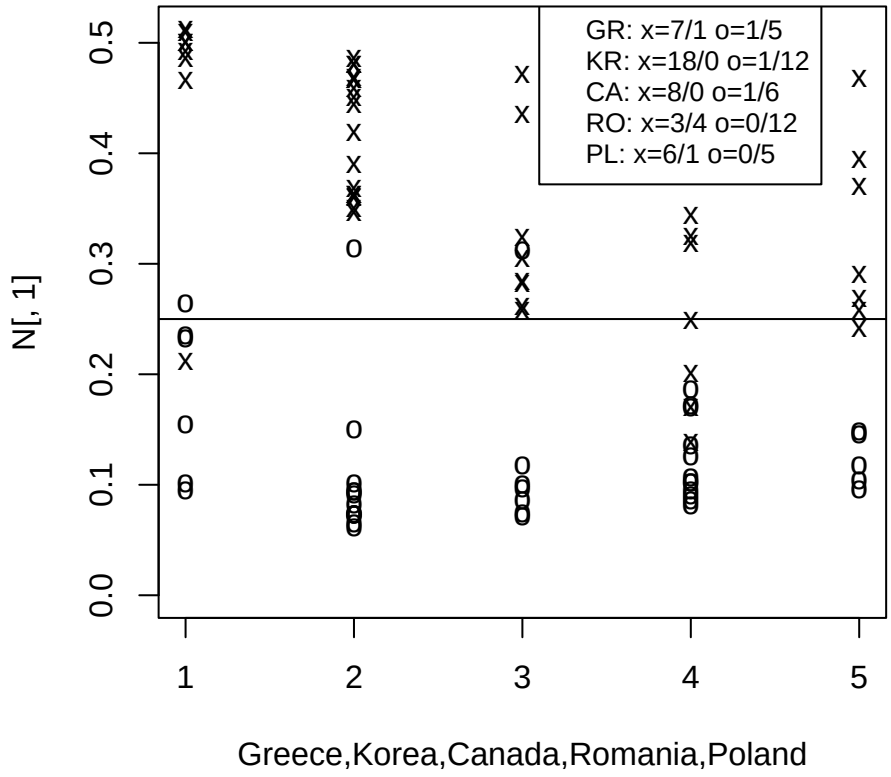



Figure 7.15: B-scores per server and transfer for experiments with 5Mbit/s bottleneck or 500ms added delay. Each marker corresponds to a single transfer: x is with delay, o is with a bottleneck. Y values are b-scores, x values are servers.

In order to see how the path characteristics (those independent of the ones introduced artificially by us) impact the results, we analyzed some FTP downloads on a per-server basis. We wanted to check whether the chosen threshold works correctly not only on average when looking at the distributions but also in the individual cases. Especially, we wanted to see how well the downloads from a single server are separated by the threshold in the cases of a shared bottleneck or receiver window limitation. We also wanted to check that there are no correlations between locations, i.e. to verify that we did not, for example, end up choosing mostly US servers in the experiments, which would lead to choosing a threshold value that only works for such a set of paths. Figures 7.15 to 7.18 show the b-score of downloads from a given server in two different scenarios: when a shared bottleneck was set up and when delay was added. In the figures the horizontal line is the chosen threshold 0.25. x and o markers represent a transfer with added

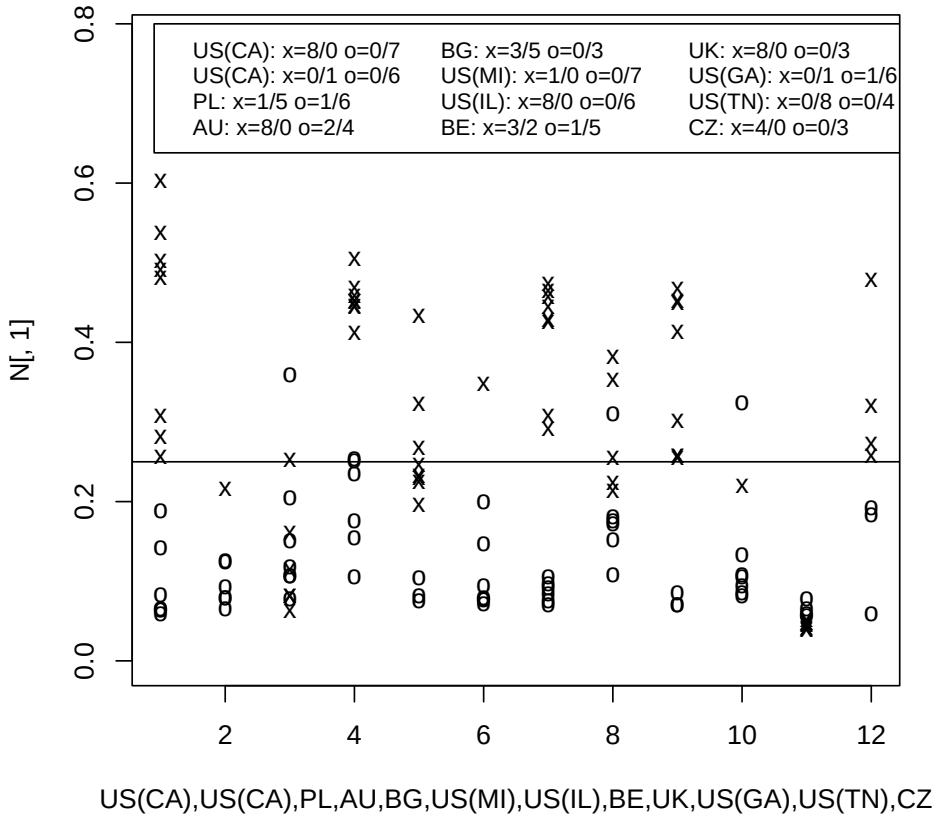



Figure 7.16: B-scores per server and transfer for experiments with 3Mbit/s bottleneck or 500ms added delay.

delay or through a shared bottleneck link, respectively. Thus, in the ideal case all the x markers reside above the horizontal line and the o markers below. The legend displays the counts of each markers below and above the threshold for each server. The geographical locations of the servers were resolved using the IP2location service (http://www.ip2location.com). These plots show that there are surely servers which did not produce ideal b-score separation. More importantly, they show that the geographical location of the server does not impact the correctness of the threshold for a particular case but it seems to depend more on the characteristics of the individual server: There are several cases of transfers from different parts of US with varying results. For example, in Figure 7.18 most of the downloads from the server located in Georgia, USA during the experiments with 400 ms of added delay achieved a b-score below the threshold. A closer look revealed that the capacity estimates for the path from this server varied between 1 Mbit/s and 3 Mbit/s during the experiments where the RTT was increased but no artificial bottleneck was introduced. Thus, it may have been the case that there was indeed a shared bottleneck on the path from the server in Georgia due to the low capacity of the path during

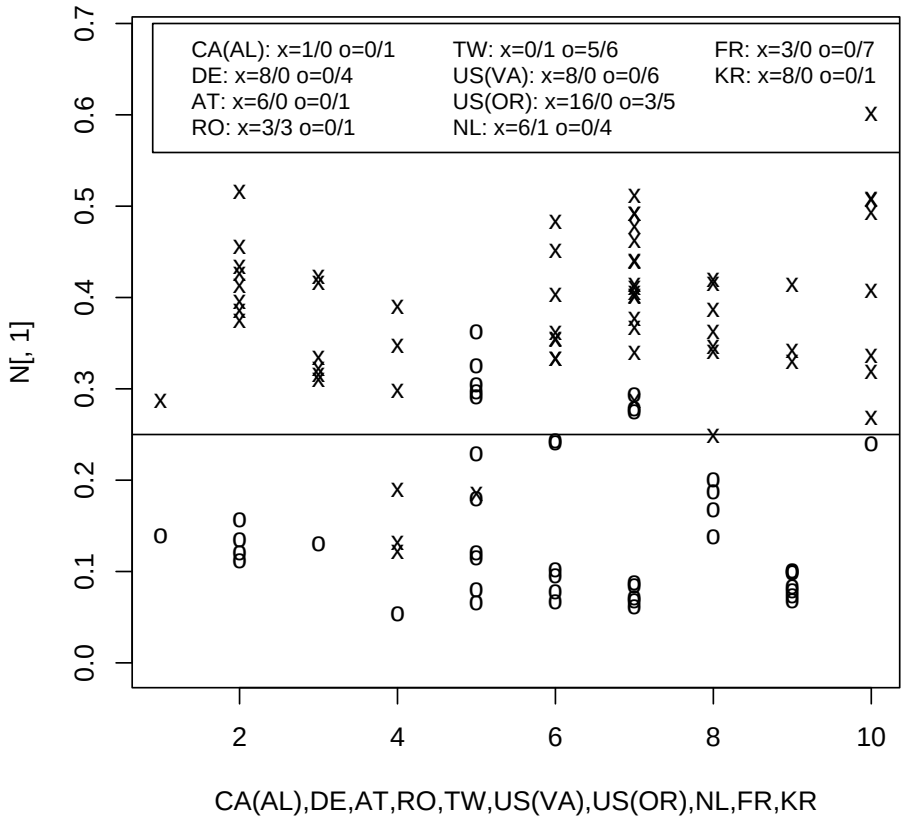



Figure 7.17: B-scores per server and transfer for experiments with 10Mbit/s bottleneck or 200ms added delay.

the experiments, and that the threshold yielded correct results. Furthermore, this bottleneck link was presumably on the USA side since the capacity estimates for the path from the server located in Oregon, USA, for instance, were constantly around 100 Mbit/s. Traceroutes to these two USA servers showed that the paths diverged just before the transatlantic link (one path used Opentransit's and another one Geant's transatlantic links). Similarly, in Figure 7.16 the capacity estimates are within 1-1.5 Mbit/s for the path from the server located in Poland in both cases, with or without the artificial bottleneck link. These estimates indicate that there was a bottleneck link during both experiments along the path which appears to be shared and the root cause for the throughput achieved.

By putting everything together, we obtain the classification procedure with the threshold values shown in Figure 7.19.

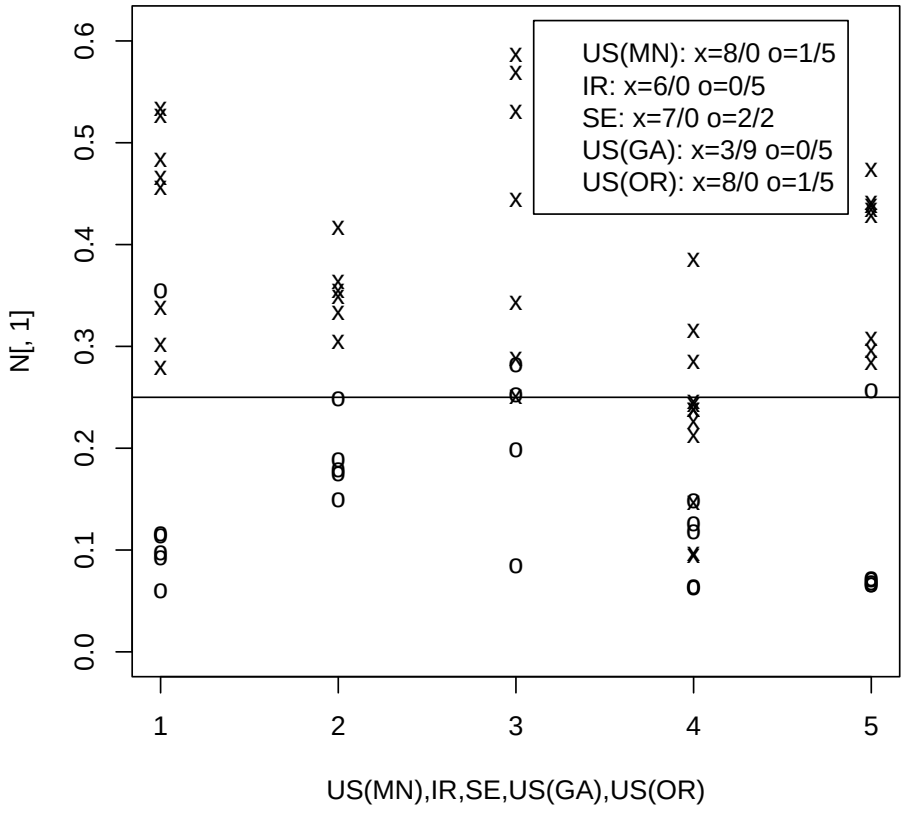



Figure 7.18: B-scores per server and transfer for experiments with 1Mbit/s bottleneck or 400ms added delay.

### 7.3.6 Root Cause Classification Results for the Experiments

In this section, we apply our threshold-based method to classify the traffic traces from the three different experiments which we used to derive some of the thresholds described earlier.

#### Unshared Bottleneck Limited Transfers: Single Download at a Time

During the first experiments, we downloaded from a single mirror site at a time. Three traces were captured with a different link capacity (inbound and outbound) set with rshaper each time. The classification results obtained using the method described in Figure 7.19 are presented in Figure 7.20. The fractions of BTPs in each class are plotted in Figure 7.20(a) and fractions of bytes in Figure 7.20(b). The dominance of the unshared bottleneck limitation cause fades when increasing the bottleneck link capacity. The transition is toward a shared bottleneck, especially in bytes. 81% of the BTPs that are classified as mixed/unknown for the case of 2Mbps bottleneck are fairly small transfers (980 KB and 269 KB as the mean and median of transferred bytes,

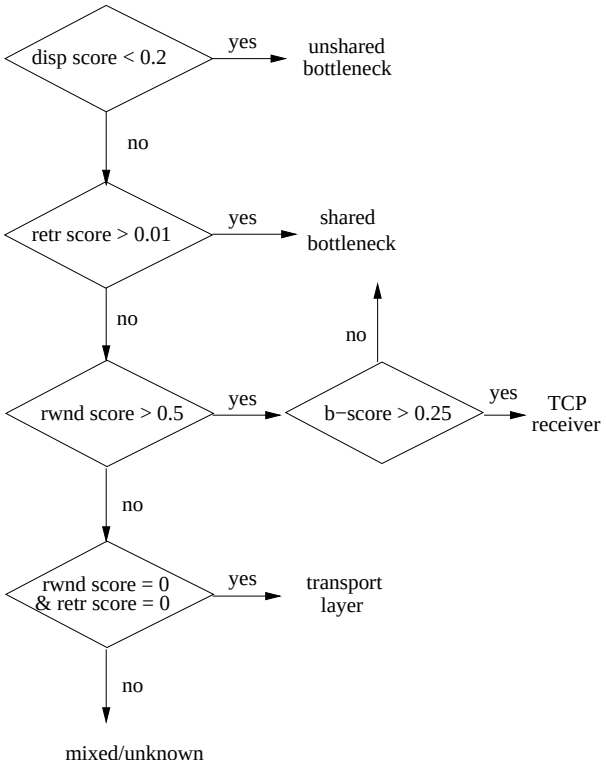
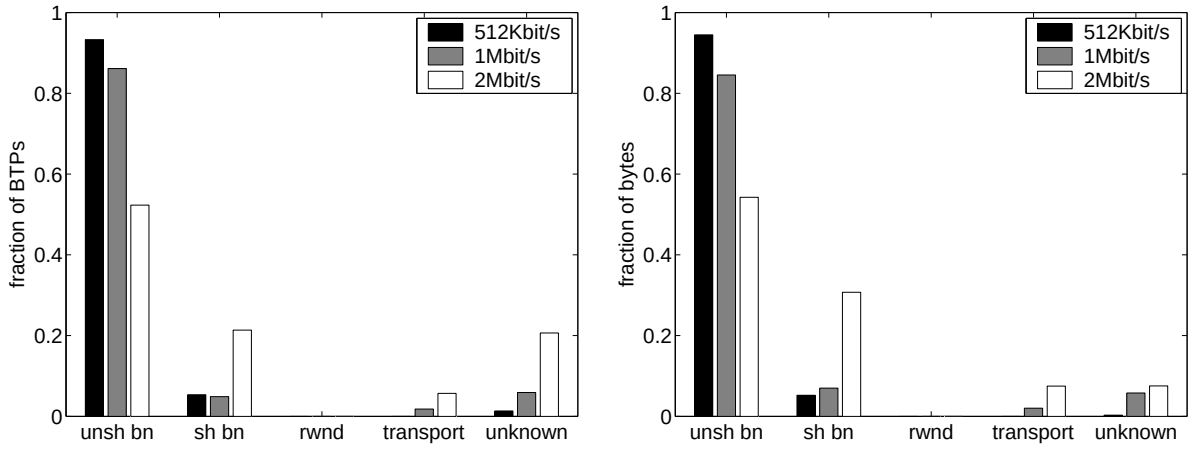



Figure 7.19: Classification of BTPs into clear root causes.

respectively) that retransmit no data and have a receiver window limitation score above 0 but below 0.5. A visual check agrees with the fact that these BTPs are in fact transport limited at first but manage to reach the receiver window limit before the end.

### Shared Bottleneck Limited Transfers: Multiple Downloads at a Time

Figure 7.21 shows the classification during the experiments with 10 parallel downloads and an artificial bottleneck with different capacities. We notice that when the capacity is increased, the amount of BTPs classified as shared bottleneck limited decreases and unshared bottleneck limited BTPs increases, which is logical as in these cases the narrow link (probably the server access link) capacity becomes smaller than the available bandwidth at the artificial shared bottleneck link. Another, non-intuitive, observation is that the amount of BTPs classified as shared bottleneck limited is less for the experiments with a 1 Mbit/s bottleneck link than for the experiments with a 2 Mbit/s and 3 Mbit/s bottlenecks. Instead, the amount of bytes transferred and classified as unknown are significantly higher. A closer look revealed that most of these BTPs were retransmitting between 0.1% and 1% of the bytes and experienced some receiver window limitation, however, less than half of the time (score less than 0.5). There are two paths to reach this class in the classification procedure in Figure 7.19: by retransmitting more than 1% of the bytes or by having a receiver window limitation score above 0.5 and b-score below the threshold. In this way these BTPs fell between the two classification paths. One way to cope with such a situation is that in case the amount of unknown bytes is significant, additional

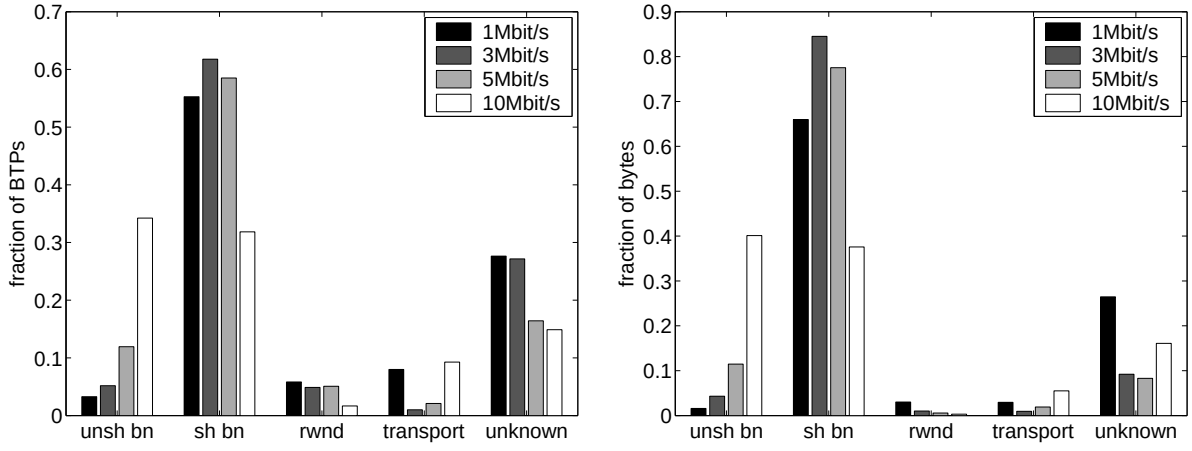



(a) number of BTPs in each class

(b) bytes carried by the BTPs in each class

Figure 7.20: Root cause classification of the three data sets with only a single download at a time.

analysis could be done for those BTPs to detect false positives and reclassify them as limited by a shared bottleneck link. The only issue would be to distinguish these transfers from those that experience really two different causes because of sudden congestion build up after a period of receiver window limited transfer, for instance.



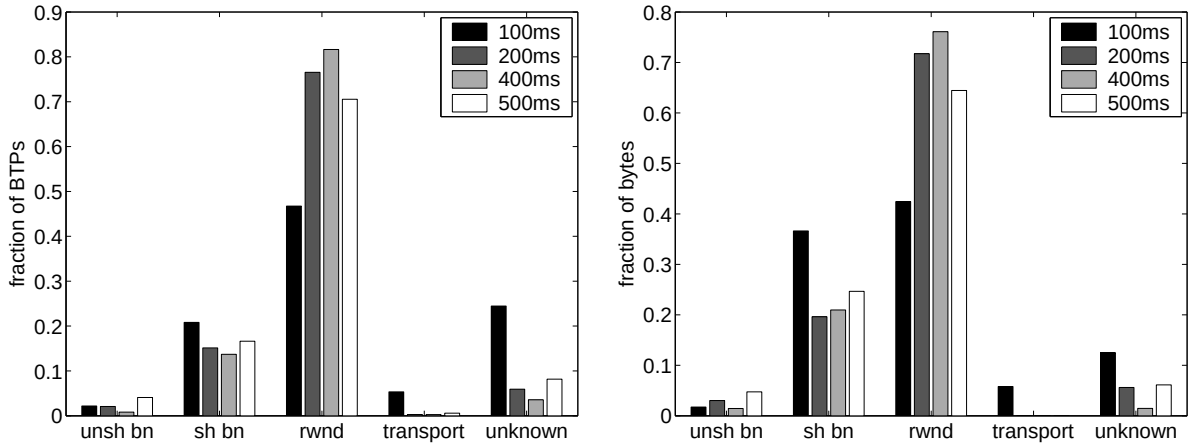

(a) number of BTPs in each class

(b) bytes carried by the BTPs in each class

Figure 7.21: Root cause classification of the three data sets with ten parallel downloads.

**Receiver Window Limited Transfers: Added Delay**

Figure 7.21 shows the classification during the experiments with 10 parallel downloads and different amounts of delay added on the path. We observe that receiver window limitation is dominant except for the experiments with only 100 ms, which can be explained by the fact that 100 ms was not enough in these cases to create a path with a delay bandwidth product higher than the receiver advertised window.




(a) number of BTPs in each class

(b) bytes carried by the BTPs in each class

Figure 7.22: Root cause classification of the three data sets with three parallel downloads and added delay.

### 7.3.7 Critical Discussion of Our Approach

Inference of the thresholds for the root cause classification scheme is something that one can endlessly argue against. Specifically, it is justified to ask questions such as: "How can you be sure that the threshold values you have inferred hold for all cases and are not only specific to the scenarios in the experiments?". The undeniable truth is that we cannot. It is simply impossible to create reference traffic from measurements that represents all possible situations in the Internet. We can only try to be as general as possible and to consider a set of scenarios as diverse as possible. That is why we used real Internet traffic measurements instead of simulations. That is also why we chose to download files from servers located in all continents in order to generate traffic.

More generally, the problem with measurement based root cause analysis is the difficulty to rigorously validate the methods. Especially, since the purpose of these methods is to detect these root causes and other satisfactory methods do not exist. Nevertheless, we are able to validate some individual parts of the algorithms, as we did in Section 7.1.4.

Therefore, rather than arguing against the inference of the thresholds or other validation details of the scheme, we turn the focus on the evaluation of the utility of our scheme in real world applications in Chapter 9. In many cases, it does not matter if a part of BTPs are incorrectly

classified if a clear majority is correctly classified. After all, the ultimate validation of our methods comes from the fact that they enable us to capture interesting and useful phenomena that we are also able to explain with common sense.

## 7.4 T-RAT

We briefly discussed the T-RAT tool [118] in Section 5.3.4. In this Section, we first investigate in detail the main limitations of T-RAT and then perform comparisons with our methods.

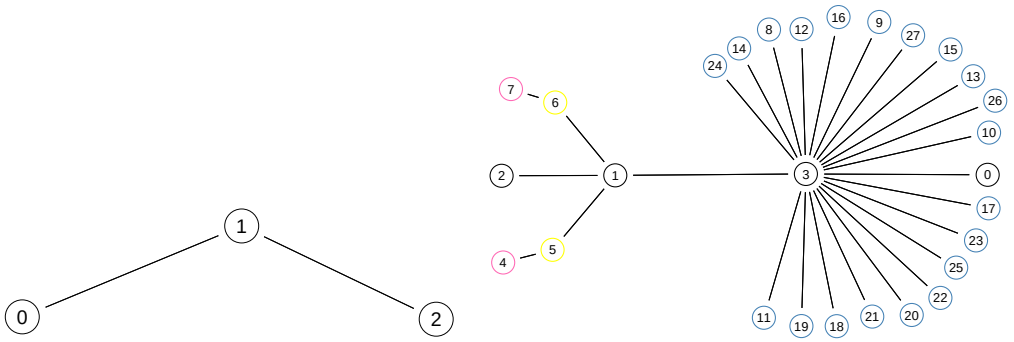
### 7.4.1 On the Flight Nature of TCP

We implemented our own version of T-RAT, as it was not publicly available, and experimented with it. One of the main problems of T-RAT is related to a notion of a "flight" of packets on which the approach of T-RAT is based. To identify a rate limitation cause, T-RAT needs to identify such flights of packets and relate them into the state of the sending TCP. This requirement is due to the fundamental design choice that T-RAT should operate on unidirectional traffic traces. However, we observed that the flights T-RAT looks for often cannot be identified, which undermines the main premise of T-RAT.

It is commonly assumed that TCP transfers packets in flights, i.e. in groups of packets that are sent back to back within a group. This is justified by the window based flow and congestion control mechanisms used in TCP. Flights are a very important notion for T-RAT as it needs to relate the flights to the different phases of TCP, namely slow-start and congestion avoidance. In order to do this, T-RAT needs to observe flights whose sizes correspond to the current size of the cwnd. A recent research work in [74] studied flights in the context of burstiness of TCP transfers. According to the authors, burstiness in some time scales can be thought of as the consequence of the self-clocking behavior of TCP that creates flights of packets. In [105], the authors search for flights in Internet traffic traces and arrive to the conclusion that they exist in two time scales: large and short. Short-time-scale flights exist regardless of the network environment and are due to the implementation of the delayed ACKs mechanism. In contrast, large-time-scale flights exist only in the presence of large buffers or large available bandwidth. Consequently, observable flight sizes can represent the current window size of the TCP only in specific conditions, which means that a tool such as T-RAT is most of the time unable to function properly. In the present work, we investigate the notion of flights through simulations, and come to a similar conclusion: while it is possible to observe groups of packets it is difficult to relate them to the well-known phases of TCP.

We simulated TCP connections limited by a specific cause using `ns-2` and varied different parameters (RTT, receiver advertised window, TCP version, etc.) affecting the behavior of the connection. Our objective was to study the similarity of the signatures of connections limited by the same cause but having different parameter values. By signature, we mean the distribution of packet inter-arrival times (IATs). For example, in the case of a receiver window limited connection one would expect to observe a bimodal distribution of the IATs, with the principal mode at $\Delta t_1 = \frac{S}{C}$ and the secondary mode at $\Delta t_2 = RTT - \frac{(W-1)*S}{C}$, where $S$ is the packet size (typically can be assumed to be equal to MSS), $C$ the capacity of the narrow link, and $W$ is the receiver advertised window. The principal mode corresponds to the time it takes to transmit a single packet on the narrow link on the path. As all the packets of a single window should be sent back to back in a single flight, their IATs correspond to this value. The position of the

second mode corresponds to the time interval between observing the last packet of the previous flight and the first packet of the next flight (see Figure 5.9). Moreover, the ratio of the heights of these peaks should be close to a factor of $W-1$ because for each window worth of packets one observes $W-1$ times an IAT of $\Delta t_1$ and one time an IAT of $\Delta t_2$. In the following, we demonstrate with a few examples that this type of simple reasoning rarely holds.




(a) Basic configuration

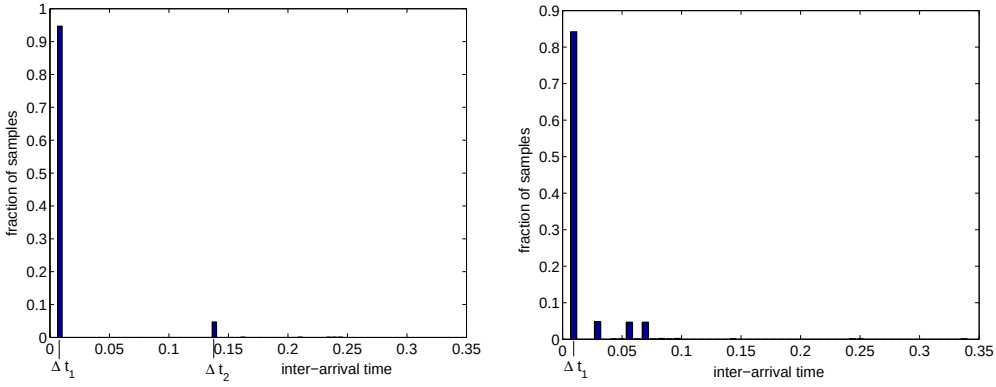
(b) Configuration with cross traffic

Figure 7.23: Simulation Configurations.

We start with a simple topology with one client (node 0), server (node 2), and one intermediate router (node 1) shown in Figure 7.23(a). A two-minute long FTP transfer was set up on top of a TCP connection established from node 2 to node 0. Figures 7.24(a) and 7.24(b) show histograms of IATs of packets where the connection is limited by the receiver advertised window of 20 packets. In Figure 7.24(a), delayed acknowledgments were not used by the TCP receiver while in Figure 7.24(b) delayed acknowledgments were used. As expected, in Figure 7.24(a) we observe the two modes at $\Delta t_1 = 5.1ms$ and $\Delta t_2 = 83.7ms$ and the ratio of their heights is approximately 20. However, if the TCP receiver is delaying acknowledgments the situation becomes more complex. We can still observe the principal mode $\Delta t_1$ in Figure 7.24(b) but instead of a single secondary mode we observe several additional modes. Due to the delayed acknowledgment timer at the receiver, the set of $W$ packets sent is divided into several smaller sets of packets sent back-to-back. The number of these groups of packets depends on the ratio of the $RTT$ to the delayed acknowledgment timer value but also on $W$. We can already conclude from this first experiment that relating flights to one of the phases of TCP is a difficult task.

We next consider a more realistic scenario (Figure 7.23(b)) with cross-traffic using the web client-server class of ns-2 at node 1. Tuning the parameters of the clients, we simulated different load values. Figure 7.25 shows an example evolution of the probability density function (PDF)[7] of the inter-arrival times of packets when increasing the offered load of the cross-traffic. In these simulations the delayed acknowledgments mechanism is used as this is the most common case. The loss rate for the FTP connection experiencing cross-traffic was zero for all cases of offered load.

[7]We compute the PDF estimates using a kernel density estimation technique [110] with Gaussian kernel function. The bandwidth parameter was chosen according to Silverman's rule of thumb for Gaussian kernel: $bw = 1.06\hat{\sigma}n^{-1/5}$, where $n$ is the number of samples and $\hat{\sigma}$ their empirical standard deviation.




(a) Without Delayed ACKs (b) With Delayed ACKs

Figure 7.24: Histograms of inter-arrival times of packets.

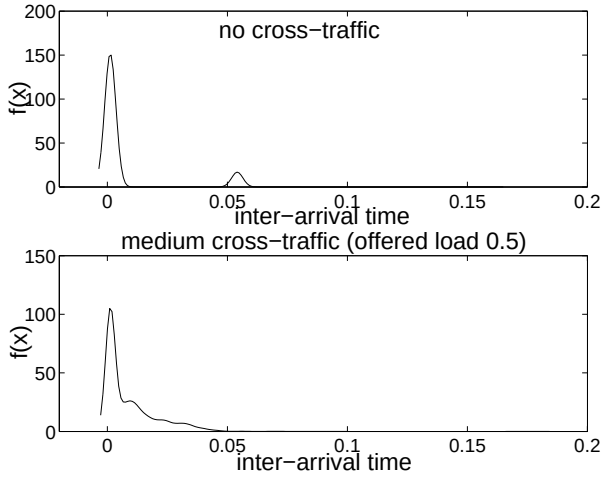



Figure 7.25: Evolution of the PDF of the inter-arrival times of packets from a receiver window limited connection without and with cross traffic.

The main observation from these plots is that even with small amounts of cross-traffic the structure of the PDF (and consequently of the groups of packets) is much more complex than in the first simple scenario. In general, cross-traffic adds to the queuing delay, which lowers the modes (i.e. creates much more different group sizes). This means that the flight sizes become more complex to identify, which makes it difficult to track the size of the congestion window. These simulations further confirm that it is often impossible to rely on the flight sizes to identify the state of the TCP connection.

### 7.4.2 Comparison With Our Methods

T-RAT breaks long connections into "flows" of at most 256 consecutive packets. This choice enables T-RAT to track changes in the limitations during a connections lifetime. However, as we focus on analyzing full connections, our implementation of T-RAT works on full connections as well. We do not consider this unfair, since majority of the traces we use for the comparison consist of connections having stable and same dominant limitation cause.

Our implementation of T-RAT classifies each flow, or in our case connection, into one or more of the following eight categories (from [118]):

I. Opportunity limited: the application has a limited amount of data to send and never leaves slow-start. This places an upper bound on how fast it can transmit data.

II. Congestion limited: the sender's congestion window is adjusted according to TCP's congestion control algorithm in response to detecting packet loss.

III. Transport limited: the sender is doing congestion avoidance, but does not experience any loss.

IV. Receiver window limited: the sending rate is limited by the receiver's maximum advertised window.

V. Sender window limited: the sending rate is constrained by buffer space at the sender, which limits the amount of unacknowledged data that can be outstanding at any time.

VI. Bandwidth limited: the sender fully utilizes, and is limited by, the bandwidth on the bottleneck link. The sender may experience loss in this case. However, it is different from congestion limited in that the sender is not competing with any other flows on the bottleneck link. An example would be a connection constrained by an access modem.

VII. Application limited: the application does not produce data fast enough to be limited by either the transport layer or by network bandwidth.

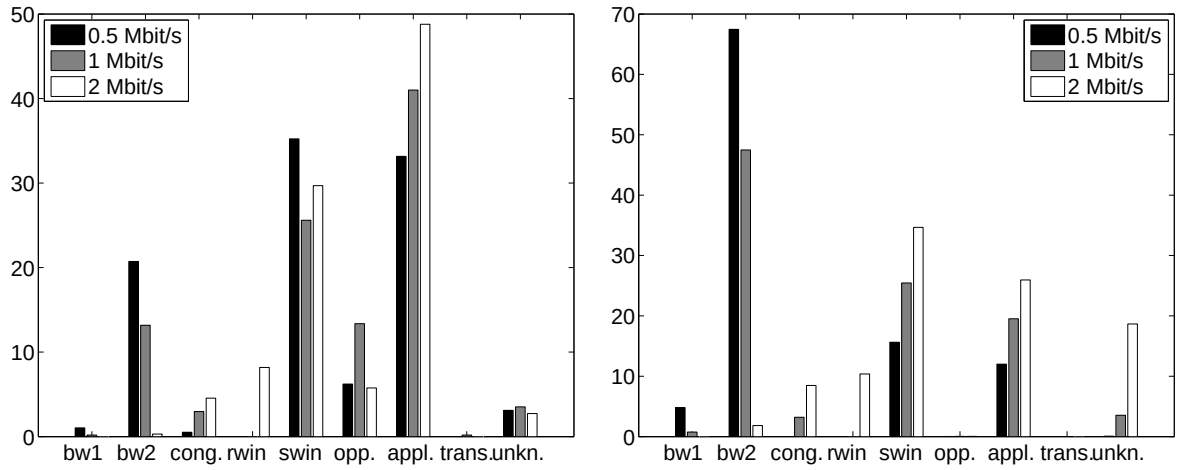
VIII. Unknown.

Opportunity limited transfers are mostly excluded from our analysis or are categorized as application limited (STPs). The definition of congestion of T-RAT corresponds to our definition of a shared bottleneck link limitation. Transport and application limitations are similar for T-RAT and our methods. Receiver window limitation is the same as receiver limitation. Our methods do not consider sender window limitation (refer to Section 5.2.2). Bandwidth limitation corresponds to our definition of unshared bottleneck limitation.

First, we used T-RAT to analyze the traces we used in Section 7.3 to calibrate and evaluate our methods. The advantage of these traces is that they contain real diverse Internet traffic but, at the same time, the traffic was generated by experiments on a controlled environment. We treat each of the three different categories of traces separately: traces limited by 1) unshared bottleneck link (bandwidth for T-RAT), 2) shared bottleneck link (congestion for T-RAT), and 3) receiver. Refer back to Section 7.3.1 for details about the experimentation setup. As the experiments involved only FTP transfers, the amount of application limited bytes in these traces should be negligible. Thus, to benchmark the application limitation tests (T-RAT and our IM algorithm), we analyzed also a trace containing only eMule traffic, which is almost purely application limited.

#### 7.4.2.1 Unshared Bottleneck Link/Bandwidth Limitation

T-RAT incorporates two tests for bandwidth limitation. If either one is passed, the flow is considered bandwidth limited. The first test is that the flow repeatedly achieves the same amount of data in flight prior to loss. The second test defines heuristics for the inter-spacing of packets in order to capture the regular spacing of packets due to the bottleneck link capacity (as in Figure 5.13). Specifically, the test is $T_{hi} < 2 \cdot T_{lo}$, where $T_{lo}$ is the $5^{th}$ percentile of the inter-arrival times of packets (IAT) and $T_{hi}$ is the $P^{th}$ percentile. They set $P = max(95, 100 \cdot (1 - .75 \cdot \frac{\#flights}{\#packets}))$. When operating on a stream of data packets, T-RAT computes the IAT sequence instead of the default $\Delta P_i$ as $\Delta P_i' = max(\Delta P_i, P_{i+1})$. In this way the effect of delayed ACKs, that cause otherwise evenly spaced packets to be transmitted in bursts of 2, is canceled[8]. We used the three traces where we set up an artificial bottleneck with a capacity of 0.5, 1, and 2 Mbit/s. The classification results of T-RAT per connections and bytes are in Figures 7.26(a) and 7.26(b), respectively.




(a) Connections

(b) Bytes

Figure 7.26: T-RAT's classification by limitation cause for traffic from unshared bottleneck link experiments.

We observe that most of the connections are limited by the application, which corresponds most likely to the control connections of FTP. However, most bytes are correctly classified as bandwidth limited by the second test for this limitation cause except with the 2 Mbit/s trace. The majority of the traffic of the 2 Mbit/s trace is classified as sender window limited. We observed with our classification results in Section 7.3.6 that in this particular trace, some 30% of the bytes were limited by a shared bottleneck link. It is logical that when we increase the capacity of our bottleneck link, some transfers get limited by a shared bottleneck link located elsewhere with a lower available bandwidth. This misclassification of shared bottleneck link limitation (i.e. congestion) as sender window limitation by T-RAT is systematic, as we will see

[8]Note that this effect is visible only if the observation point of the traffic is before the bottleneck link of the data path

from the analysis results of the shared bottleneck experiments. Also the amount of application limited bytes increases along with the amount of sender window limited bytes. Our methods found consistently below 10% of application limited bytes, which were mostly small transfers that they did not qualify as BTPs (< 130 packets). However, there are also a few larger transfers that our methods classify application limited. Those transfers consist of constant size packets that are smaller than the advertised MSS. That is why they are classified as application limited. The reason could be that some FTP servers are enforcing rate limits per connection. The origin of the overestimation of application limitation by T-RAT is linked to the misclassification of shared bottleneck limited transfers, as we will see in the next section.

#### 7.4.2.2 Shared Bottleneck Link/Congestion Limitation

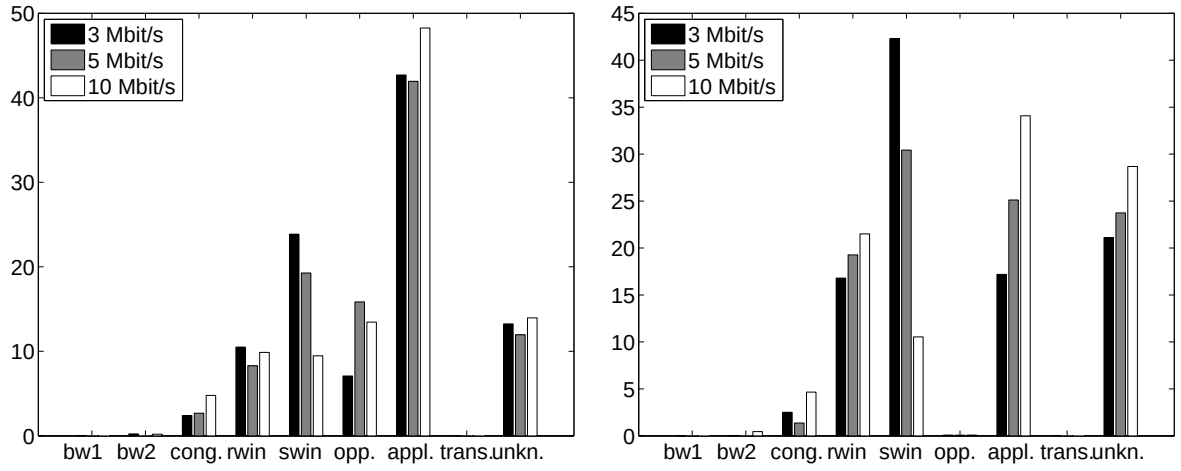
T-RAT considers a flow congestion limited if it experienced loss and it does not satisfy the first test for bandwidth limitation. We analyzed the traces having an artificial bottleneck link with a capacity of 3, 5, and 10 Mbit/s. The results are in Figures 7.27(a) and 7.27(b). Again, the majority of connections fall into the category of application limitation while the majority of bytes are classified as sender window limited for the traces with 3 Mbit/s and 5 Mbit/s bottleneck links. The reason for this misclassification is that in these experiments very few packets were lost. Thus, for majority of the transfers the number of outstanding bytes was limited by the receiver window while the rate was limited by the shared bottleneck link (refer to Section 5.2.3). Hence, our methods classified the transfers as shared bottleneck limited by inspecting the receiver window limitation score and b-score. Since the shared bottleneck broke the regular flight structure, T-RAT underestimated the RTT, and as a consequence, underestimated also the flight sizes. Thus, it found for most of the connections consecutive small flights of approximately same size clearly below the receiver advertised limit, and deemed these connections as sender window limited. Had it estimated the RTT and grouped the packets into flights correctly, the dominant limitation cause would have been the receiver window.

T-RAT classifies the majority of the bytes of the trace with a 10 Mbit/s bottleneck link as application limited. This is due to the fact that the RTT is clearly underestimated, as we explained above. T-RAT's test for application limitation searches for idle intervals longer than RTT preceded by a packet smaller than the MSS. If the RTT is largely underestimated, each packet smaller than the MSS will cause a positive result.

As discussed in Section 5.2.3, the main factor that makes the difference between shared bottleneck link limited transfers that do or do not experience losses is the buffer size at the bottleneck link (and possibly other links on the path). We set the buffer with rshaper to no longer than two seconds. Nowadays, it is not unrealistic to assume that such cases occur. We have observed ADSL access links that buffer up to eight seconds of packets. Figure 7.28 plots the RTT evolution of an example transfer captured on such an access link. Thus, such cases must be taken into account.

#### 7.4.2.3 Receiver Limitation

T-RAT determines a flow to be receiver window limited if it finds three consecutive flights which have a size $S \cdot MSS > awnd_{max} - 3 \cdot MSS$, where $awnd_{max}$ is the largest receiver advertised window size. We ran T-RAT on the traces where we added 200ms, 400ms, and 500ms of delay to the path. Most of the connections are application limited, as expected, while most of the bytes fall into the category of receiver window limitation. T-RAT's RTT estimation seems to




(a) Connections

(b) Bytes

Figure 7.27: T-RAT's classification by limitation cause for traffic from shared bottleneck link experiments.

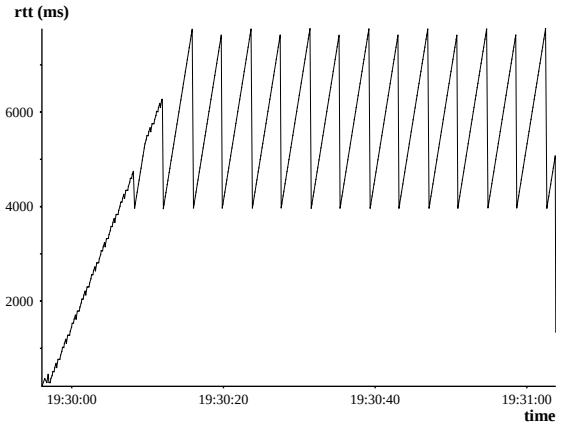
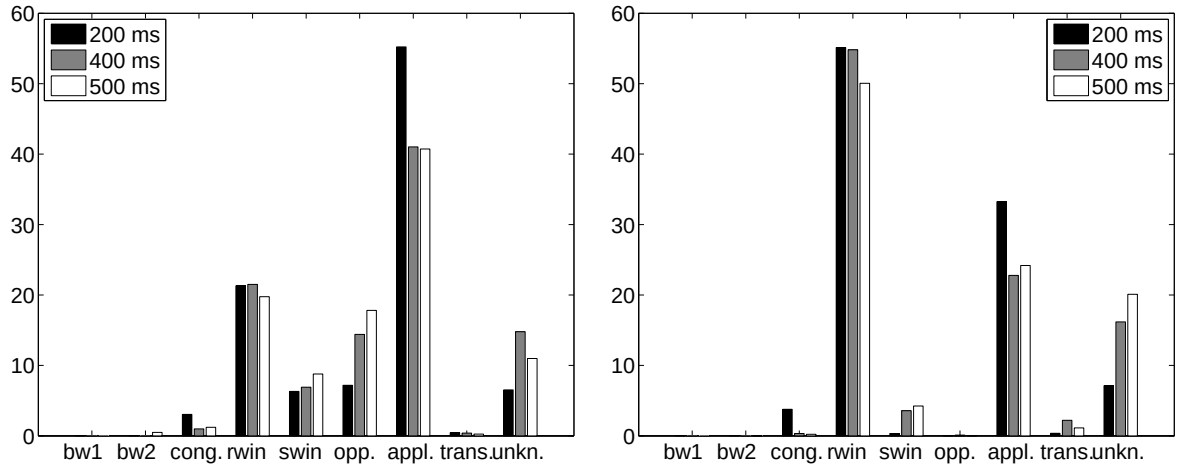



Figure 7.28: RTT evolution of an example transfer over an ADSL access link with a particularly deep buffer.

work rather well with these traces since the flight grouping induces correctly receiver window limited transfers. However, there are again a significant amount of application limited bytes, clearly more than our IM algorithm found (at most 13%). For the traces with 400ms and 500ms of added delay around 20% of bytes are unknown. Most likely many of these application limited and unknown bytes are transfers that suffer from the problem related to correctly identifying flights that we described with simulation examples in Section 7.4.1.




(a) Connections

(b) Bytes

Figure 7.29: T-RAT's classification by limitation cause for traffic from receiver limited experiments.

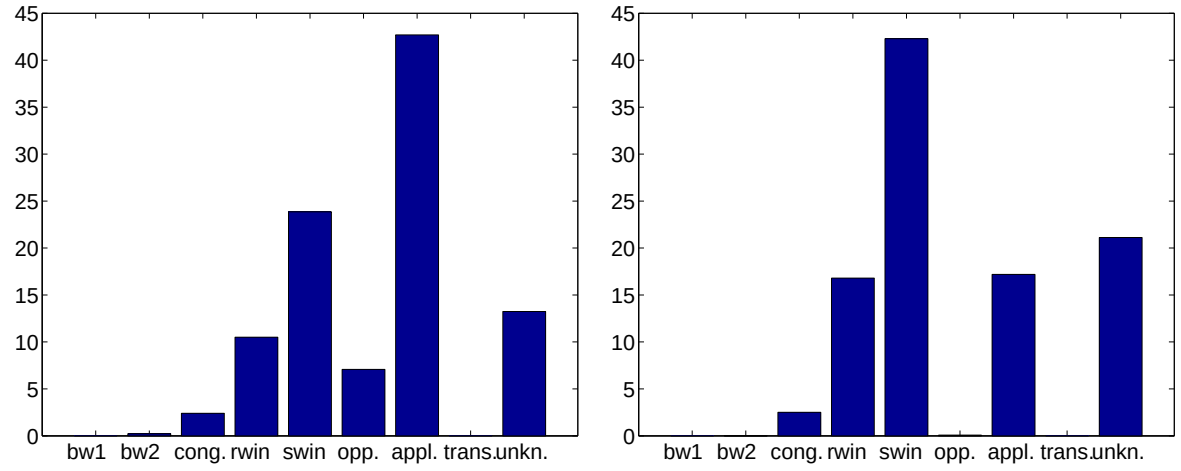
#### 7.4.2.4 Application Limitation

As we already mentioned, T-RAT searches for idle intervals longer than RTT preceded by a packet smaller than the MSS as indications of application limitation. The authors mention in [118] that T-RAT is unable to identify application limitation in all cases. They admit that, for example, an application that sends constant bit rate traffic in MSS-sized packets will likely be identified as bandwidth limited due to the regular inter-spacing of packets. Thus, they state that the exhaustive study of application limited traffic in the Internet remains a subject for future study.

Nevertheless, we wanted to confirm with experiments such a limitation of T-RAT. Hence, we analyzed a more than a day long eMule trace with T-RAT and our algorithm. The trace was captured on a single host downloading a large file. Total amount of data transferred was 1.9 GB. The host was enforcing a maximum upload rate limit, which is done by default by a typical eMule (or any eDonkey) client, as we already discussed in Section 6.5. That is why it is likely that also the clients that we downloaded from enforced an upload rate limit. Thus, we expect to see a large majority of application limited traffic. Furthermore, eMule client generally limits the transmission rate by sending constant bit rate traffic in smaller than MSS-sized packets (see Figure 6.14). As a consequence, this application limited constant bit rate traffic should be unrecognized by T-RAT. Indeed, while our IM algorithm reported the vast majority of all bytes being application limited[9], T-RAT's classification results, shown in Figures 7.30(a) and 7.30(b), are quite different.

Surprisingly, most of the bytes are classified as sender window limited and not as bandwidth limited, as we would expect. The reason for failing the bandwidth test is visualized with a time-sequence diagram of a typical example in Figure 7.31. In addition to pairs of packets, we

[9]We obtained 97% with $drop = 1$ and 89% with $drop = 0.9$.




(a) Connections

(b) Bytes

Figure 7.30: T-RAT's classification by limitation cause for eMule traffic limited by the application.

observe regularly groups of three packets. These occasions are highlighted with boxes in the figure (the third packet is very small). Hence, T-RAT fails to observe the regular inter-spacing of packets because the computed $5^{th}$ percentile of the IATs ($T_{lo}$ in the bandwidth test) is too short caused by these groups of three packets. Remember that the IAT sequence is computed as the maximum value of each IAT pair. Therefore, each group of three short IATs will cause a very short IAT value in the sequence. In Figure 7.31, these groups of three packets appear each time after three or four consecutive pairs of packets. Hence, they occur frequently enough to significantly affect the $5^{th}$ percentile of the IATs. Consequently, as with the shared bottleneck experiments, the flight structure does not exist since the application paces out the packets and T-RAT underestimates the RTT as well as the flight sizes and classifies the transfers as sender window limited.

T-RAT was designed to operate on unidirectional traces of TCP connections and even on traces where the beginning of the connection is missing. Because of these fundamental design choices, it is impossible to tweak T-RAT to function correctly in several cases of application limitation, which would require the ability to compute the amount of packets smaller than MSS. The reason is that for this T-RAT should, of course, know the MSS of each connection, which is not always the case. MSS information is exchanged during the connection establishment only, which is not necessarily seen by T-RAT. That is why T-RAT estimates the MSS from a data packet stream as the largest packet seen. An application sending a constant bit rate traffic (e.g. Skype) usually sends only fixed size packets using a fixed length interval. In this case, MSS-size packets are never seen by T-RAT and MSS is misestimated.

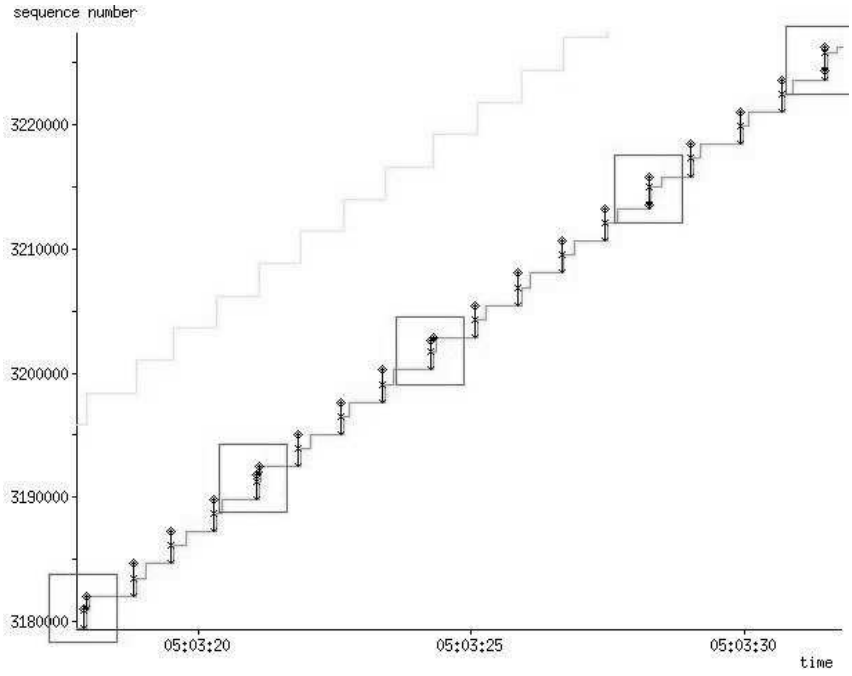



Figure 7.31: Piece of an application limited eMule transfer.

# Conclusions for Part II

In this part, we first discussed the potential limitation factors that prevent a given TCP connection from achieving a higher instantaneous throughput. We show, inspired by the work in [118] that these reasons can lie on different layers of the Internet protocol stack, namely on application, transport, and network layers, and that they can take many forms when observing the generated traffic patterns.

We then presented tools and techniques to infer these different limitation causes for TCP connections observed at a single point along their path. Our techniques are based on a set of metrics extracted and computed from the packet traces. Validation of the techniques is very challenging due to the diversity of the Internet traffic. Nevertheless, we validated and calibrated our tools and techniques as best we could by applying them to a set of real Internet traces from experiments in a partially controlled environment. The goal was to be as general as possible and at the same time control the experiments to produce desired type of traffic with a high probability.

As the impact of applications is potentially large and diverse, we investigated their interaction with the underlying TCP in more detail. We showed, for instance, that one must pay great attention on the effect of application when measuring throughput and delay from the point of view of the network.

The next part of this thesis binds together the methodology described in Part I and the techniques from this part. We guide the reader through a real world case study on analysis of traffic from a commercial ADSL access network. This part serves as a final validation of our techniques, it shows the potential of our methodology and techniques together, and gives an idea about the type of scenarios they could be useful in.

# Part III

# Real World Case Study on TCP Root Cause Analysis Using InTraBase

# Overview of Part III

The core of Part 3 is a real world case study on analysis of traffic from an ADSL access network of France Telecom. The goal is to demonstrate the capabilities and prove the utility of the InTraBase, presented in Part I equipped with the techniques for root cause analysis of TCP throughput that we presented in Part II. This case study gives also an idea of the way the InTraBase combined with the root cause analysis techniques could be used in future research work.

First, we describe in Chapter 8 the way we adapt our InTraBase approach for root cause analysis of TCP throughput. We show the customized database schema and explain how the tables are populated with the help of root cause analysis functions.

Then, in Chapter 9, we guide the reader through a study of traffic generated by ADSL customers of commercial ISPs using the ADSL platform of France Telecom. We had the unique opportunity to apply our techniques on real, and proprietary, traffic data collected during one full day at the edge of an ADSL access network. We discover a variety of interesting results on client and application behavior as well as the limitation causes in general.

CHAPTER 8

# Adapting InTraBase for TCP Root Cause Analysis

So far in this thesis, we have presented our methodology for traffic analysis in Part I and the methods and techniques to analyze root causes of the throughput of TCP connections in Part II. In this chapter, we tie these two parts together by explaining how we utilize the InTraBase approach in practice to perform the root cause analysis. We describe the design of our prototype implementation of the InTraBase adapted specifically to root cause analysis of TCP traffic. We explain the way we implement techniques to compute limitation scores and the classification scheme.

## 8.1 Extended Design of InTraBase for Root Cause Analysis

### 8.1.1 Table Layout

We use the same pgInTraBase prototype implemented on PostgreSQL and extend the design to incorporate necessary tables for TCP root cause analysis. In Section 3.2.1 we described the base schema for the PgInTraBase prototype. Figure E.11 shows the table layout that is extended for the purposes of our root cause analysis techniques.

The tables *bulk_transfer* and *app_period* store the BTPs and ALPs, respectively, identified by the IM algorithm (refer to Chapter 6). A single connection may contain several of these periods each of which is a single row in these tables. Furthermore, different values of the *drop* parameter used when executing the IM algorithm produce a different set of BTPs and ALPs. That is why this parameter is also included in both of these tables. Note that we give each BTP a unique identifier *btid*.

The remaining three tables *bnbw_test*, *retr_test*, and *rwnd_test* hold each data from a single quantitative limitation test, i.e. the limitation scores (refer to Section 7.1). *bnbw_test* table contains the capacity estimated by PPrate algorithm which, combined with the throughput of the BTP, enables the computation of the dispersion score. The table *retr_test* holds retransmission score but also the fraction of reordered and duplicated packets out of all packets[1]. Finally,

[1]These metrics are computed as a side product of computing the retransmission score but they are currently not used in the root cause classification scheme.

the receiver window limitation score and b-score are stored in the *rwnd_test* table. This table contains also the mean and standard deviation of the receiver advertised window that could be used for detecting middleboxes (refer to Section 7.2.2). Note that the identifier *btid* enables us to link each BTP in the *bulk_transfer* table to the corresponding limitation scores in these three tables. We refer to the set of these five tables as root cause analysis tables (RCA tables).

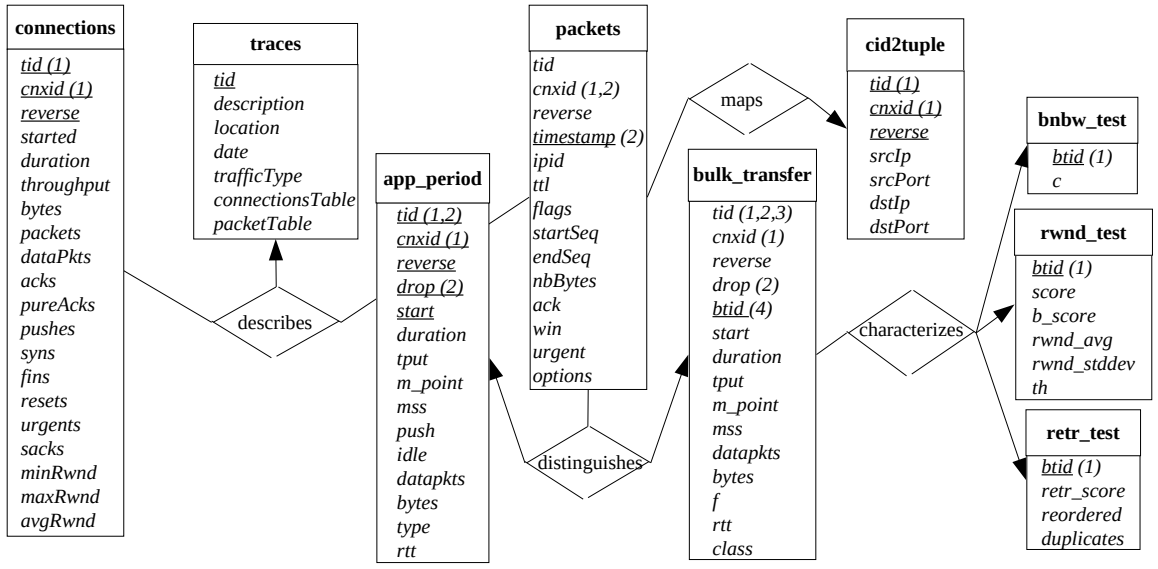



Figure 8.1: Table layouts of intrabase adapted for TCP root cause analysis. Underlined attributes form a key that is unique for each row.

### 8.1.2 Indexes

Figure E.11 shows also the various indexes that we create for the RCA tables. The *bulk_transfer* table contains four indexes: two single attribute indexes on *tid* and *btid* and two combined indexes on *(tid,cnxid)* and on *(tid,drop)*. For the *app_period* table, we create the same two combined indexes as for the *bulk_transfer* table on *(tid,cnxid)* and *(tid,drop)*. We create these indexes primarily to speed up the process of populating these tables, i.e. the execution of the IM algorithm. However, many of them also serve later querying.

The three other tables, *bnbw_test*, *retr_test*, and *rwnd_test*, are indexed only on the *btid* attribute. The queries issued on these tables typically fetch scores for a given BTP. Based on our experience, it is rarer to issue queries to find those BTPs that have a particular score among all the BTPs. Usually, at least the trace from which the BTPs originate is specified in which case the index on the *btid* attribute is used to fetch only the scores for those BTPs. Finding then given scores within the remaining set is generally fast enough without further indexes.

## 8.2 Root Cause Analysis Functions

### 8.2.1 Populating Root Cause Analysis Tables

All the RCA tables are populated using functions written in PL/pgSQL language. There are two functions that implement the IM algorithm. The first function implements the *Isolate* procedure and the second function the *Merge* procedure. These two functions populate the tables *app_period* and *bulk_transfer*. In order to populate the three other tables, there are four other functions. Thus, each of these functions implements one of the limitation tests in order to compute a given limitation score (remember that there are four scores and three tables). These main functions use other functions during their execution. For example, the function that computes the receiver window limitation score uses two other functions to compute the necessary time series of outstanding bytes and receiver advertised window (refer to Section 7.1.3 for details). The resulting time series are then combined within the main function in order to yield the resulting score.

Each of the functions takes as input the name of the table that holds the packets for a given trace and some other parameters related to the algorithm. The execution of the function is illustrated by the pseudo code in Algorithm 1. When populating the *bulk_transfer* and *app_period* tables, the function selects the identifiers of the connections belonging to this trace from the *connections* table and for each connection at a time queries the packet-level details from the packet table and stores the identified periods in corresponding tables. The procedure is similar when the other RCA tables are populated except that the functions query the identifiers of each BTP belonging to the trace from the *bulk_transfer* table and computes the limitation score from the packet details. The score is then stored into the corresponding table.

**Algorithm 1**: Populating root cause analysis tables with PL/pgSQL functions.

```
input argument tablename;
my_tid := SELECT tid FROM traces WHERE packetTable=tablename;
foreach SELECT cnxid/btid AS my_id FROM connections/bulk_transfer WHERE tid=my_tid do
    foreach SELECT * FROM tablename WHERE cnxid=my_id ORDER BY ts do
        apply IM algorithm or compute limitation score;
        store identified periods or limitation score;
```

Note that in this way we end up querying the packet details every time we compute a new limitation score, for instance. It is for sure overhead as opposed to an approach where the packet details would be queried only once and all analysis tasks performed at the same time. The latter approach would require that we query the packet-level details for a given connection or BTP once and then store them into the main memory. Once in memory, the algorithms would access this packet-level data rapidly instead of having to read it from the disk. The problem is that PL/pgSQL is rather limited as a programming language. It contains arrays for storing such data but the array handling is so cumbersome and inefficient that it is often more attractive to query the packets again from the disk. Another option to improve the performance would be to implement a higher level database function in C++. This function would query the packet details for each connection or BTP in turn and call each PL/pgSQL analysis function with the packet details. However, for the moment, we have not found it necessary to try such an approach.

### 8.2.2 Going Further With Triggers

Currently the execution of the functions to populate the RCA tables is launched manually with SQL queries. It would be possible to make the database more autonomous with triggers. A trigger, as the name suggests, triggers execution of an associated trigger function before or after an INSERT, UPDATE, or DELETE operation, as specified. The execution can happen either once per modified row, or once per SQL statement. Thus, we could add triggers for the *connections* and *bulk_transfer* tables. The trigger in the *connections* table would launch the execution of the function that populates the *bulk_transfer* and *app_period* tables, i.e. the IM algorithm, whenever a new row is inserted into the table. Similarly, the trigger in the *bulk_transfer* table would launch the execution of the functions to populate the other three RCA tables, i.e. to compute and store each limitation score. For these triggers, our PL/pgSQL functions only need to be modified to work on a single connection, which means that we would remove the first *foreach* loop from Algorithm 1 and give the identifier of the connection or the BTP as an input parameter.

CHAPTER 9

# Case Study on Performance Analysis of ADSL Clients

In this chapter, we present the case study on the analysis of traffic from France Telecom's ADSL access network. We focus on the performance limitations from the point of view of the client. By performance, we refer to the throughput. We describe first the monitoring set up and characteristics of the captured data. We then define a taxonomy of the different factors that can potentially limit the performance of an ADSL client and use this taxonomy to identify clients that experience a specific limitation factor and study the impact of a particular factor on the performance.

For an ISP, client satisfaction is very important. While low prices may attract many clients at first, clients leave and the word spreads fast if the service does not meet expectations. Therefore, if a client perceives lower download or upload transfer rates than what was specified in the subscription purchased by the client, it is important for the ISP to understand why. On one hand, it is desirable to be able to explain to the client the reasons for not being able achieve rates that correspond to the subscribed capacity, especially in the case that the reasons are not due to the ISP. On the other hand, if the reasons are due to the ISP, i.e. congestion in the internal network, the ISP needs to know that in order to be able to improve the service.

Unfortunately, the knowledge about the causes that limit the performance, i.e. the throughput, of a specific client at a given time instance is not readily available. The metrics to directly infer these causes are not provided by the network. Hence, these metrics need to be computed and partially estimated. In this case study, we use our TCP root cause analysis techniques, described in Part II, to analyze the performance limitations of clients of France Telecom's ADSL access network. These techniques provide us with the necessary means to compute the required metrics and study the performance limitations that clients experience.

It becomes clear during the study that we compute from the large amount of traffic data many metrics on several different level of detail or different time scale. Furthermore, this study was a typical example of the iterative nature of a traffic analysis process (refer to Chapter 2), and the analysis work spanned over several months for various reasons. Without the use of the InTraBase or a similar system, it would have been very difficult to keep track and manage all the data and analysis results.

We first describe in Section 9.1 the measurement setup and discuss the limitations and

challenges posed by the scenario. Then, in Section 9.2, we look at the characteristics of the traffic data set collected. We look at some general characteristics of the traffic and client behavior. After that, in Section 9.3, we focus on the performance analysis of the clients. Finally, in Section 9.4, we study in detail the behavior of three different example clients.

## 9.1 Monitoring the ADSL Access Network of France Telecom

### 9.1.1 Architecture and Setup

A classical ADSL architecture is organized as follows (see Figure 9.1): the BAS (Broadband Access Server) aggregates the traffic issued from many DSLAMs (Digital Subscriber Line Access Multiplexer) before forwarding it through the local routers to the France Telecom IP backbone. Each client is connected to one DSLAM using one ATM Virtual Circuit. The traffic of clients is controlled by the up and down capacities of this access link. A variety of subscription types of the ADSL access service is defined through different combinations of uplink and downlink capacities.

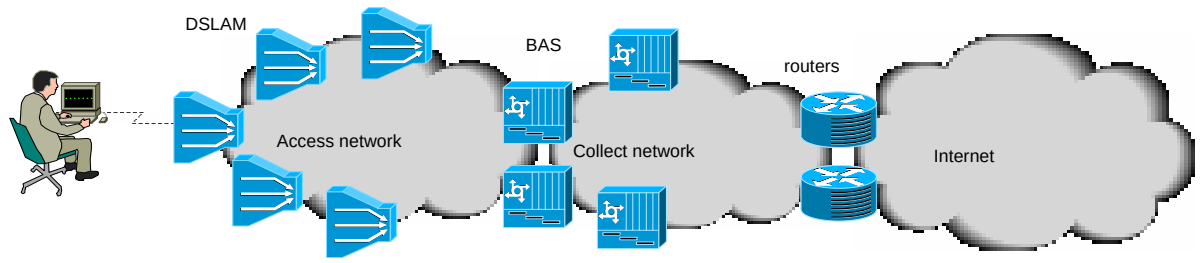



Figure 9.1: Architecture of the monitored ADSL platform.

Our two probes are located between a BAS and the first two routers – a single probe captures packets flowing through a single router. This BAS multiplexes the traffic of three DSLAMs. It connects around 3000 clients to the Internet. It is noteworthy that we capture 64 bytes of all IP, TCP and UDP headers of packets going through the BAS without any sampling or loss. We use for the capture an internal tool similar to tcpdump, and based on the *pcap* C-library.

Our analysis is based on a whole day of traffic measurements, March 10, 2006. The collected data for this day represents a significant amount of traffic: approximately 290 GB of transferred data in total. A single capture file is recorded by each of the two probes every thirty minutes by an individual tcpdump process in order to avoid unnecessarily large trace file sizes. There is a short overlapping between the start of the new capture process and the termination of the former one. This overlapping may induce some duplicated packets, which we identify through identical timestamps and TCP/IP header fields.

### 9.1.2 Main Constraints and Challenges

For all of our analysis, we use only two main pieces of data or information. The packet traces are the main input. In addition, we have a list of IP addresses that belong to local clients, which

allows us to distinguish the direction of the traffic – the probe itself is not aware of where a given packet came from or is going to.

The above described information is all we have to perform the analysis. For instance, we do not know clients' subscription rates, i.e. uplink and downlink capacities. Such knowledge would be very desirable and the lack of it complicates the analysis work, as becomes clear when we advance further in this study. Another valuable information that we do not possess is the precise knowledge of clients current IP address. Throughout this study, we identify clients through their IP addresses. However, the IP address of a client can change during the day. The address is changed periodically approximately once per day. It can change also if a client shuts down the ADSL modem/router and then restarts it later that day. Unfortunately, we did not have access to data that would map a client to its current IP address, which further complicates the analysis.

It is clear that the above constraints are limitations for the scope of the analysis. However, they can also be seen as challenges. The lack of precise knowledge about clients is a common situation is such a monitoring setup. We show how meaningful analysis can be performed by using only minimal amount of a-priori knowledge.

## 9.2 Traffic Characteristics: Applications, Connections, and Clients

### 9.2.1 General Characteristics of the Traffic

#### 9.2.1.1 Traffic per Application

Figure E.12 shows how the amount of transferred bytes evolves and distributes between the most common applications each half an hour. We account separately for only those applications that have significant fractions of traffic transferred ($> 5\%$ of the total bytes). These applications are only five (email comprises SMTP, POP3, and IMAP traffic). Other applications are included in the "other" category. Bytes transferred upstream are above the x-axis and bytes downstream below the x-axis. We associated the TCP port range 4660-4669 to eDonkey, the ports 6880-6889 and 6969 (tracker) to BitTorrent, and standard TCP port numbers for the rest of the applications.

The application responsible for most of the transferred bytes is eDonkey followed by traffic originating or destined to ports 80 and 8080. We do not want to declare this traffic as Web traffic since it is likely to include also P2P traffic, as we show later. However, the clearly largest category is the "other" traffic with more than 50% of all uploaded and downloaded bytes on the average. We have reason to suspect that much of this traffic is disguised P2P traffic that uses uncommon TCP ports. We show some examples of such P2P traffic in Section 9.4 where we look at a few clients' behavior in detail. Unfortunately, we were unable to classify applications using techniques that are based on string matching because the payloads of packets were discarded during the capture. Other methods have been proposed as well but, unfortunately, we could not find any publicly available tools. The methods described in [27] and [76], for instance, were not available as public tools. This work is thus left as future subject of study.

The amount of eDonkey traffic is almost constant throughout the 24 hours. Similarly, there is no clear structure in the traffic of the "other" applications. Rather, the amount of this traffic tends to stay constant as well, which is another hint of hidden P2P traffic within this traffic. We observe a diurnal pattern with port 80/8080 traffic that is almost negligible in the middle of the

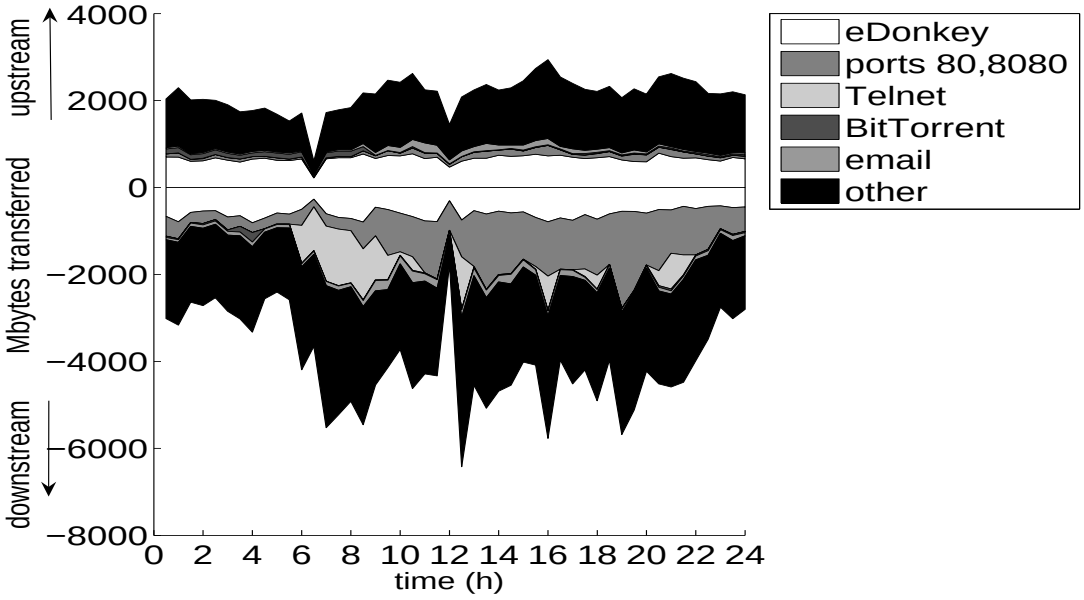



Figure 9.2: Amount of bytes transferred by different applications during the day.

night and peaks at 2pm and 7pm. This effect is much more pronounced in the downstream traffic. Interestingly, BitTorrent traffic appears only during the night and almost only upstream. This observation further suggests that there is some hidden P2P at least in the downstream traffic since it is very unlikely that BitTorrent clients would only upload data. As one could expect, email traffic is more present during the day than during the night. Telnet traffic emerges from time to time in an inexplicable way. We looked at this traffic in more detail and it turned out to consist of only few long and fast transfers originating from a couple of hosts.

#### 9.2.1.2 Traffic per Connection

As our root cause analysis techniques operate on individual TCP connections, let us also have a glance at their characteristics. Figure 9.3 shows a log-log complementary distribution (LLCD) plot of the sizes of connections. We can see that, considering all connections, while the distributions are not necessarily heavy tailed by definition, they tend towards it, since the LLCD plot has a somewhat linear shape. Heavy tailness is a typical property of especially Web traffic [41] that also among our data sets shows the strongest evidence (port 80,8080 data set). Thus, most of the connections are very small but, as Figure 9.4 shows, they do not contribute much to the total traffic. As a remark, our IM algorithm that analyzes the impact of the application as a root cause inspects only connections larger than 130 data packets, which is equivalent to about 190 KB (if we assume MSS to be 1450 Bytes). As a consequence, we exclude 99% of connections from the root cause analysis. However, the excluded connections carry only approximately 15% of total bytes.

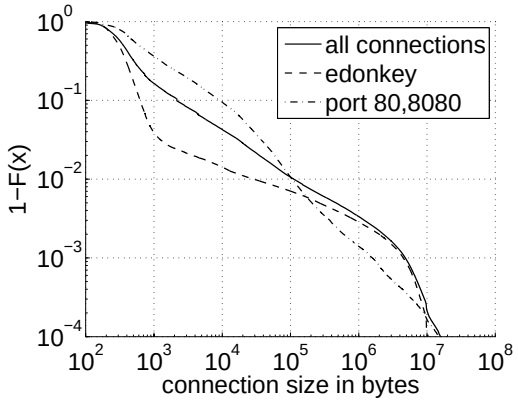



Figure 9.3: CCDF plot of size of connections. Note the logarithmic scale of both axes.

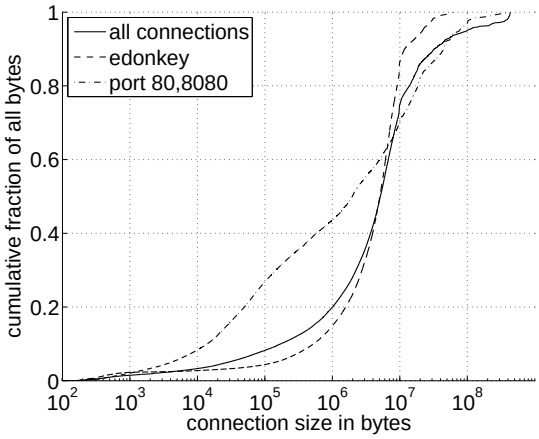



Figure 9.4: Cumulative fraction of all bytes as a function of the connection size.

### 9.2.2 Client Behavior

As mentioned before, we identify a client through the IP address. In the analysis, we often focus on *active* clients. We define a client active during a period of 30 minutes if it transferred at least 100 KB during that period. While we observed packets from approximately 3000 clients, our analysis focuses only on a subset of them: those 1335 clients that generated at least one long enough connection for root cause analysis.

#### 9.2.2.1 Volumes and Applications

To get a better idea about the set of clients our analysis captured, we study their behavior in terms of volumes of data transferred, applications used, and link utilization. Figure 9.5 shows the distribution of the bytes transferred by clients, and Figure 9.6 shows the cumulative fraction of bytes contributed by a client that transferred a given amount of bytes. These two figures tell us that the amount of traffic generated per client is heavily skewed: About 15% of the most active clients transfer roughly 85-90% of the total bytes, both upstream and downstream. These 15% account for 200 out of the total of 1335 analyzed clients and we refer to them as the **heavy-hitters**. A study that was recently performed on a much larger scale for Japan's residential user traffic [36] reported that 4% of heavy-hitter clients s account for 75% of the in-bound and 60% of the out-bound traffic.

Note that these two sets of heavy-hitter clients, upstream and downstream, are distinct sets that both comprise about 200 clients. However, the sets are heavily overlapping since among these 200 clients, 128 clients are in both sets, which indicates that the majority of the heavy-hitters both, upload and download a lot of data, which comes most likely from P2P applications. The average amount of bytes uploaded and downloaded by a heavy-hitter client is approximately 470 MB and 760 MB, respectively, while for the non-heavy-hitters these average values are 9 MB and 27 MB.

Figure 9.7 shows that a clear majority of clients are active less than an hour. Remember that we have identified the clients over the whole day using the IP addresses. However, as we

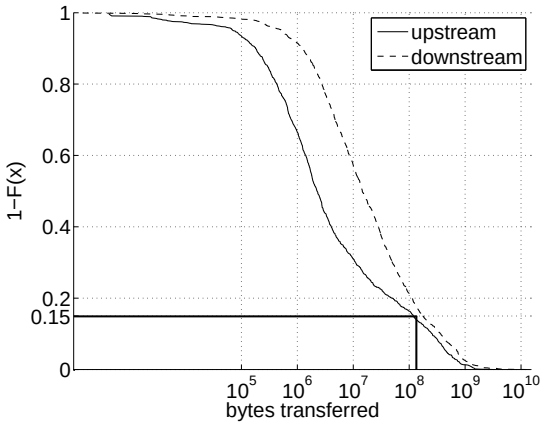



Figure 9.5: CCDF plot of bytes transferred by clients.

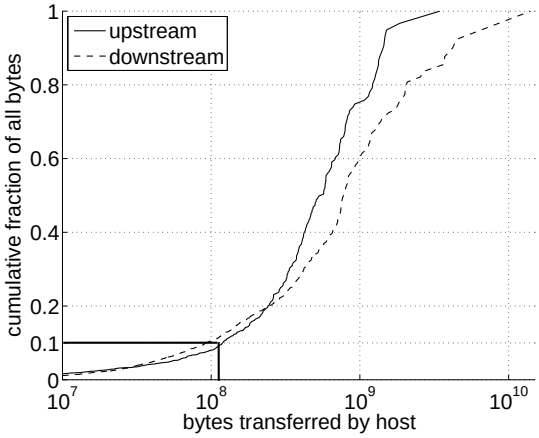
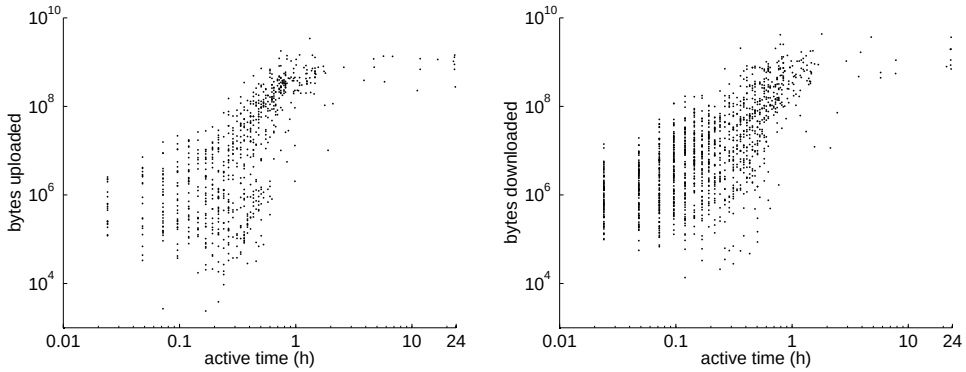



Figure 9.6: Cumulative fraction of all bytes transferred as a function of bytes transferred by a given client.

explained earlier, the IP address of a client can change during the day. It is unlikely that a given IP has been reused by another client because we observe only few IP addresses that are active a long time. On the other hand, as there are many clients active only a short time, it is possible that some clients changed their IP address and emerged as new clients. However, the figure points out an interesting characteristic of the heavy hitters: Those clients that transmit the most bytes can be active either almost the whole day or only a few hours. We look at an example of both of these types of clients in Section 9.4.




(a) uplink

(b) downlink

Figure 9.7: Amount of bytes transferred by client vs. time that client is active.

We next compare the profile of an average heavy-hitter and of an average non-heavy-hitter in terms of the applications they use. To do so, we compute for both groups how much certain applications contribute to the total amount of traffic. Then, we selected for each client the application that generated the most bytes. The results are shown in Table 9.1. The main

difference between an average client and a heavy hitter is that heavy hitters tend to use P2P applications (esp. eDonkey) more extensively.

Table 9.1: Percentages of clients that transmit most bytes using a specific application.

| **Upstream** | | | | | | |
|---|---|---|---|---|---|---|
| | **total** | **eDonkey** | **Ports 80,8080** | **BitTorrent** | **email** | **others** |
| non-heavy-hitters | 1135 | 6.1% | 4.1% | 0% | 0.53% | 89% |
| heavy hitters | 200 | 44% | 2.5% | 0.50% | 0.50% | 53% |
| **Downstream** | | | | | | |
| | **total** | **eDonkey** | **Ports 80,8080** | **BitTorrent** | **email** | **other** |
| non-heavy-hitters | 1135 | 8.4% | 6.9% | 0% | 0.26% | 84% |
| heavy hitters | 200 | 28% | 9.5% | 0% | 1.0% | 62% |

#### 9.2.2.2 Access Link Utilization

To compute the utilization of a link, one needs to know its capacity. As we do not have this knowledge, we need to make an estimation. Note that we cannot use tools such as Pathrate [43] that estimate capacity of an entire path, which is equal to the capacity of the link with the smallest capacity along the path, because the local access link is in many cases not the one with the smallest capacity on the path. For instance, in P2P download from another ADSL client, the downlink capacity of the local client is very likely to be higher than the uplink capacity of the distant peer. We use *the maximum observed instantaneous throughput as the lower bound of the access link capacity*. The instantaneous throughput samples were computed for each client as the throughput during non-overlapping five-second intervals. To compute the link utilization for a client during a period of 30 minutes, i.e. one period of capture, we divide the mean aggregate throughput (all bytes uploaded/downloaded by the client during the period divided by the duration of the period) by the maximum instantaneous throughput sample observed for that client. In this way, we obtain an *upper bound for the utilization* because we use a lower bound for the capacity. From here on, whenever we mention link utilization, we refer to this upper bound.

However, an important question arises concerning the above described method for computing the utilization: Should the access link capacity estimate be the maximum instantaneous throughput value observed for that IP address during the entire day or only during a given 30-minute period? Given that the IP address of a client changes over the course of the day, we cannot choose the maximum value over the whole day. On the other hand, if we take the maximum value for each 30-minute period for each active IP address, we are likely to greatly underestimate the capacity and, consequently, overestimate the utilization in many cases. For example, consider a client that transmits at a steady slow rate traffic during an entire 30-minute period in such a way that the rate is far from the real capacity of the access link, i.e. the throughput is constantly limited by another factor. In this case the maximum instantaneous throughput is much lower than the capacity and, as a result, computing the utilization as the mean throughput divided by this maximum throughput value yields incorrectly a value close to 100%.

Because of the above explained issues, we compute utilization for a given client only during

the 30-minute period during which we observe the maximum instantaneous throughput value over the whole day for that specific IP address. In other words, utilization for each client is computed only for a specific 30-minute period and this period is not necessarily the same for different clients. In this way, we increase the probability of obtaining an estimate of the capacity which is closer to the true value.

Figure 9.8 shows the CDF plot of the utilization. we see that overwhelming majority of clients are far from fully utilizing their access links. This is even more the case if we remember that our approximation of the utilization tends to *over estimate* the actual utilization. We see that 80% of the clients have an utilization of less than 20% for their downlink and less than 40% for their uplink.

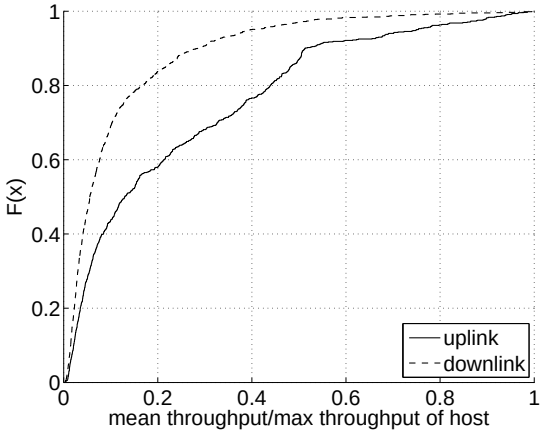



Figure 9.8: CDF plot of upper bound for link utilization per client for a 30min period: mean throughput divided by maximum instantaneous throughput. For each client, we selected the period during which that client achieved maximum throughput.

To get another point of view of the link utilization, we studied the duration of utilization above certain thresholds for each client. Namely, we computed the fraction of time a given client's utilization is above 0%, 50%, 70%, and 90%. We note these variables as $t_0$, $t_{50}$, $t_{70}$, and $t_{90}$. In other words, we counted those 5-second intervals when the instantaneous throughput is above a given threshold times the maximum observed rate. We then compute the following values:

$$t_{50}^0 = \frac{t_{50}}{t_0},\ t_{70}^0 = \frac{t_{70}}{t_0},\ t_{90}^0 = \frac{t_{90}}{t_0}$$

In other words, in addition to computing $t_0$, the time the client is using the link, we evaluate by computing $t_{50}^0$, $t_{70}^0$, and $t_{90}^0$ how high rates are sustained when the link is used. These fractions of time were computed for each client over the peak 30-minute period, i.e. similarly to the mean utilization in Figure 9.8. Table 9.2 summarizes the main observations. These results demonstrate that there is a fair amount of clients active during almost the whole period: 34% and 20% of clients use at least 90% of the 30-minute period their uplink and downlink, respectively. However, clients rarely sustain high transfer rates during the time they are active. For example, only 15% of the clients sustain upload rates above half of the maximum more than half of the time they are active. For downloads the percentage of clients is only 5%.

Table 9.2: Percentage of active clients that sustain utilization of their access link above specific thresholds for a given fraction of a 30-minute period.

| **Upstream** | | | | |
|---|---|---|---|---|
| | **1% of time** | **10% of time** | **50% of time** | **90% of time** |
| utilization >0% | 97% | 91% | 55% | 34% |
| utilization > 50%, when >0% | 91% | 50% | 16% | 6.4% |
| utilization > 70%, when >0% | 85% | 29% | 12% | 4.1% |
| utilization > 90%, when >0% | 65% | 21% | 6.4% | 1.3% |
| **Downstream** | | | | |
| | **1% of time** | **10% of time** | **50% of time** | **90% of time** |
| utilization > 0% | 100% | 83% | 36% | 20% |
| utilization > 50%, when >0% | 91% | 36% | 5.0% | 0.9% |
| utilization > 70%, when >0% | 82% | 21% | 3.2% | 0.5% |
| utilization > 90%, when >0% | 66% | 12% | 1.8% | 0.4% |

Having seen that most clients achieve very low link utilization, we will now set out to investigate the causes. For this purpose, we will use our the root cause analysis techniques.

## 9.3 Performance Analysis of Clients

In this section, we first identify the performance limitation causes that clients may experience and discuss how they could be identified. Then we identify the clients that experience these different limitation causes in our data set and, finally, discuss the impact of a particular limitation cause on the performance.

### 9.3.1 Taxonomy of Factors Limiting the Performance of Clients

We first present a taxonomy of the potential limitation causes for an ADSL client. We define the performance of a client as the aggregate throughput either upstream or downstream. Note that, as our focus is the performance from the client's perspective, we should not use directly the limitation causes we have defined and identified for TCP connections. Instead, we will build on those definitions and use our techniques to identify them in order to come up with a methodology for client-level root cause analysis. Specifically,

For each of the limiting factor, we define the following:

(a). How the limiting factor can be detected using the connection-level root cause knowledge.

(b). What is the expected performance.

(c). Which applications are mainly concerned.

(d). What are the possible solutions to improve the throughput.

We identify four categories of factors that potentially limit the aggregate throughput of a client:

I. Applications limit the throughput.

(a) **identification:** the majority of traffic contained in ALPs,

(b) **performance:** low aggregate throughput,

(c) **applications:** P2P applications,

(d) **solutions:** upgrade client applications to use dynamic upload rate limits that adapt according to the amount of other traffic from the client.

We categorize traffic into this limitation cause when it is identified as ALP by our IM algorithm (refer to Section 6.1). We expect mainly traffic from P2P applications to fall into this category. The reason is that P2P client applications often limit the upload throughput in order to prevent saturation of the uplink and performance problems of other simultaneous upstream and downstream traffic, e.g. Web browsing. For example, a typical eDonkey client sets an upper limit for the aggregate upload throughput. The bandwidth allocated by this limit is then split equally among the active upload connections. If the uplink is saturated, also download performance can suffer greatly because of "ACK compression" which means that ACK packets get stuck in the uplink queue behind data packets. The effect of asymmetric links on the performance of TCP has already been extensively studied, e.g. in [81], [19], [75], and [90]. Many currently used P2P client applications do not adapt to the utilization of the uplink. The user sets the uplink capacity in the configuration of the application after which the application maintains constant aggregate upload rate limits. Naturally, a better approach is to adapt to the current utilization. In other words, if there is no or little other traffic, the uplink could be mostly saturated.

II. Access link is saturated. We identify several different cases:

- Access uplink/downlink is saturated by a single TCP connection.
  (a) **identification:** high uplink/downlink utilization, the majority of upstream/downstream traffic classified as BTPs limited by an unshared bottleneck link,
  (b) **performance:** high aggregate throughput
  (c) **applications:** bulk transfer applications (e.g. FTP and Web),
  (d) **solutions:** upgrade to a higher capacity access link.
- Access uplink/downlink is saturated by many TCP connections.
  (a) **identification:** high uplink/downlink utilization, the majority of upstream/downstream traffic classified as BTPs limited by a shared bottleneck link,
  (b) **performance:** high aggregate throughput,
  (c) **applications:** various applications,
  (d) **solutions:** upgrade to a higher capacity access link.
- Downstream traffic suffers from saturated access uplink.
  (a) **identification:** same diagnosis as the above saturated uplink cases
  (b) **performance:** low downstream aggregate throughput
  (c) **applications:** various applications,

(d) **solutions:** e.g. prioritized scheduling for acknowledgments or size based scheduling of packets (solutions also in [81] [19] [75] [90]), or rate limits for upstream traffic.

We define the limitation cause of a given BTP to be a saturated access link, if during that period the access link utilization is at least 90%. The amount of such BTPs corresponding to downloads or uploads of a client are then evaluated in order to determine whether this particular client experiences this limitation cause. In further analysis presented in this section, we do not distinguish between limitation by shared and unshared saturated access link. The reason is obvious: the client only cares about the fact that the link is saturated.

The last of the above limitation cases, where the saturated uplink damages the performance of downstream traffic, corresponds to the case where the application does not implement rate limits for upstream traffic. The performance degradation can be significant, as we already explained above. If the access uplink utilization is 90% or higher during a download BTP, we label that BTP limited by a saturated uplink.

III. Network limitation due to a bottleneck link at the distant host or along the path

(a) **identification:** the majority of traffic classified as BTPs limited by a bottleneck link and aggregate throughput is not close the access link capacity,

(b) **performance:** low aggregate throughput,

(c) **applications:** various applications,

(d) **solutions:** nothing can be done if the bottleneck link is not local.

The traffic carried by the rest of the BTPs limited by the network and during which the access link utilization is smaller than 90% fall into this category. There are two potential cases of such limitation: either the bottleneck link resides in the internal ADSL network or it is located outside of the France Telecom network. In the first case, the performance can be naturally improved by means of traffic engineering (re-routing, capacity planning etc.). Otherwise, there is not much that can be done.

IV. TCP configuration problems limit the throughput: too small default receiver advertised window or slow start threshold.

(a) **identification:** the majority of downstream traffic classified as receiver limited or upstream traffic transport limited BTPs,

(b) **performance:** throughput depends on the delay,

(c) **applications:** various applications,

(d) **solutions:** adjust the TCP parameters.

This limitation case occurs if the default TCP parameter values (receiver advertised window size and slow start threshold) are set too low. Such a situation is visible as BTPs unintentionally limited by the receiver window or transport limited BTPs (refer to Section 5.2.2). In order to realize that the receiver limitation is unintentional for a particular client, we need to check whether the advertised receiver window size is relatively constant and corresponds to a standard default value (e.g. 8 KB, 16 KB, 24 KB, 32 KB, and 64 KB).

In each case that we manually checked where a client had majority of traffic falling into this category (very few cases as we will see in Section 9.3.2), it was the case. For obvious reasons, TCP configuration problems for a local client can only occur as unintentionally receiver limited downloads and transport limited uploads.

In summary, to evaluate the presense of these client-level root causes for a given client, we perform the following procedure. For each active client we consider all the bytes transferred by all the connections of the client within a given 30-minute period. We then associate these bytes into the three considered client-level limitations. To do this association, we use the connection-level RCA as follows: All the bytes carried by the ALPs of all the connections of the client are associated to application limitation. All the bytes carried by all the BTPs that are labeled network limited (unshared or shared bottleneck) by connection-level RCA and during which the utilization is above 90% of the maximum are associated to access link saturation. All the bytes carried by the rest of the network limited BTPs during which the utilization is below 90% of the maximum are associated to network limitation due to a distant bottleneck. All the rest of the bytes transferred by the client, and not covered by these three limitations, are associated to "other" (unknown) client limitation. The amount of bytes associated with a limitation serves as a quantitative metric of the degree of that limitation for a given client during a 30-minute period.

### 9.3.2 Observed Limiting Factors for Clients

We analyzed the traffic data set using the PgInTraBase prototype equipped with our root cause analysis techniques. We first applied the IM algorithm (refer to Section 6.1) to partition individual TCP connections into BTPs and ALPs (we used $drop = 0.9$ with the algorithm) after which we computed the limitation scores for each BTP, as explained in Section 7.1.3, and then classified the BTPs in the way described in Section 7.2. Finally, this TCP connection-level knowledge was mapped into client-level root causes according to the method described in the previous section. We first look at the number of clients that experience a particular limiting factor, and then investigate which applications are mainly used by the applications that experience a specific limitation. Our analysis showed that TCP configuration issues was a marginal cause in our data set. Hence, we exclude this limitation from further discussions.

We know from our previous work on RCA that for a single, possibly very long connection, the limitation cause may vary over time. Also, a single client may run one or more applications that will originate multiple connections. For this reason, we distinguish for each client between "main limitation" and "limitations experienced". As **main limitation**, we understand the limitation that effects the most number of bytes for this client. This classification is exclusive. i.e. each client belongs to a single limitation category.

On the other hand, under **limitations experienced** a single client will be considered in all the categories whose limitation causes it has experienced. Therefore, this classification is not exclusive. The results are presented in Table 9.3. We present the results for two 30-minute periods of the day: 4-4:30am and 3-3:30pm, which are representative for the different periods of the day. We see that during the night time heavy hitters dominate (70 out of 77 active uploading clients and 61 out of 83 active downloading clients), which is not surprising if one considers that heavy hitters heavily use P2P applications and P2P file transfer that can run for several hours [100]. If we look at the absolute number of clients, we see that only a small fraction of 1335 clients is active in either 30-minute period.

#### 9.3.2.1 Main Limitation

If we look at the main limitation cause experienced by the clients, we see that *almost all clients see their throughput performance mainly limited by the application*. This holds irrespectively of the direction of the stream (upstream or downstream), of the type of client, average client or heavy hitter, and of the period of the day.

The clients that are not application limited see their throughput either limited by the capacity of the access link or the capacity of another link along the end-to-end path. Capacity limitations occur more frequently during the daytime than at night. The very limited number of cases where we observe a saturation of the access link complies with the low utilization observed in Figure 9.8.

#### 9.3.2.2 Limitations Experienced

Besides the main limitation, we also consider *all the limitation causes* experienced by a single client. The most striking result is the difference between main limitation and limitations experienced for the "other link" limitation. As we have seen, this limitation is rarely the main limitation, while the percentage of clients that experience such limitation is between 40% and 60%, which means that while approximately half of the clients experience such network limitation, this limitation cause is not dominant. Moreover, we checked that for a given client, the amount of bytes transferred while limited by the network is generally clearly less than the amount of bytes transferred while limited by the dominant cause, i.e. the application in almost all of the cases.

Table 9.3: Number of active clients limited by different causes.

| **Upstream** | | | | | | | |
|---|---|---|---|---|---|---|---|
| limitation cause | | | *Total active #* | *application* | *access link* | *other link* | *other cause* |
| main limitation | all clients | 4am | 77 | 95% | 0% | 4% | 1% |
| | | 3pm | 205 | 86% | 6% | 4% | 4% |
| | heavy hitters | 4am | 70 | 94% | 0% | 4% | 2% |
| | | 3pm | 111 | 92% | 2% | 3% | 3% |
| limitations experienced | all clients | 4am | 77 | 100% | 0% | 60% | – |
| | | 3pm | 205 | 100% | 7% | 39% | – |
| | heavy hitters | 4am | 70 | 90% | 0% | 66% | – |
| | | 3pm | 111 | 92% | 5% | 64% | – |
| **Downstream** | | | | | | | |
| limitation cause | | | *Total #* | *application* | *access link* | *other link* | *other cause* |
| main limitation | all clients | 4am | 83 | 93% | 1% | 4% | 2% |
| | | 3pm | 286 | 76% | 4% | 18% | 2% |
| | heavy hitters | 4am | 61 | 97% | 0% | 2% | 1% |
| | | 3pm | 114 | 80% | 2% | 16% | 2% |
| limitations experienced | all clients | 4am | 83 | 100% | 1% | 53% | – |
| | | 3pm | 286 | 100% | 7% | 42% | – |
| | heavy hitters | 4am | 61 | 100% | 0% | 59% | – |
| | | 3pm | 114 | 100% | 4% | 61% | – |

### 9.3.3 Throughput limitation causes experienced by major applications

Having done the root cause analysis on a per-client basis, we now investigate what are the most important applications that experience the different limitation causes, namely (i) application limited, (ii) saturated access link, and (iii) bottleneck at distant link.

Figure 9.9 shows the main applications that generate traffic that is application limited. We compute the different amounts by simply summing all the bytes for all the ALPs for each 30 minute period. If we look at the evolution of the total volume of traffic that is application limited we see very little variation in time and an upload volume almost as big as the download volume, both being around 2 GBytes per 30 minutes. The largest single application that generates application limited traffic is, as expected, eDonkey. However, if we look by volume, the largest category is "other", i.e. the one where we were not able to identify the application generating the traffic. The overall symmetry of upload and download volumes for the "other" category as well as a manual analysis (refer to Section 9.4) of the traffic of some heavy hitters strongly suggest that the "other" category contains of a significant fraction of P2P traffic.

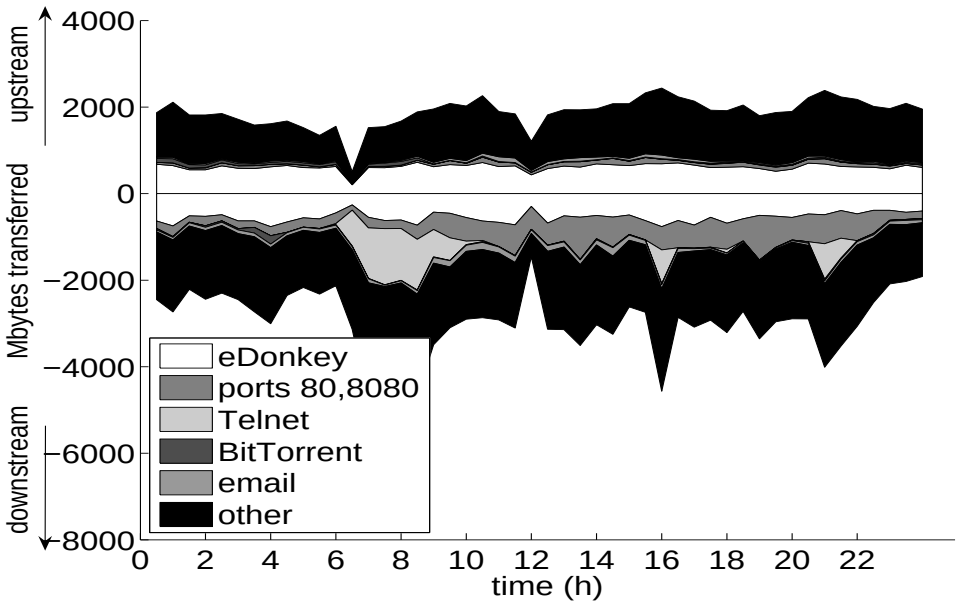



Figure 9.9: Amount of transferred application limited bytes during the day for most common applications.

Figure 9.10 shows the main applications that saturate the access link. For this cause, no traffic originating from recognized P2P applications was seen. Instead, a significant portion of traffic saturating the uplink is e-mail. For the downlink it is mainly traffic on ports 80 and 8080 and traffic for which the application could not be identified. The fact that the traffic using ports 80 and 8080 primarily saturates only downlink suggests that it could be real Web traffic that consists of small upstream requests and larger downstream replies from the server, as opposed to P2P traffic which is typically more symmetric. If we compare the absolute volume we see that most of the activity is concentrated to day time, with the peak being in the early afternoon and a total volume that is even at its peak almost negligible as compared to the traffic volume

that is application limited (see Figure 9.9).

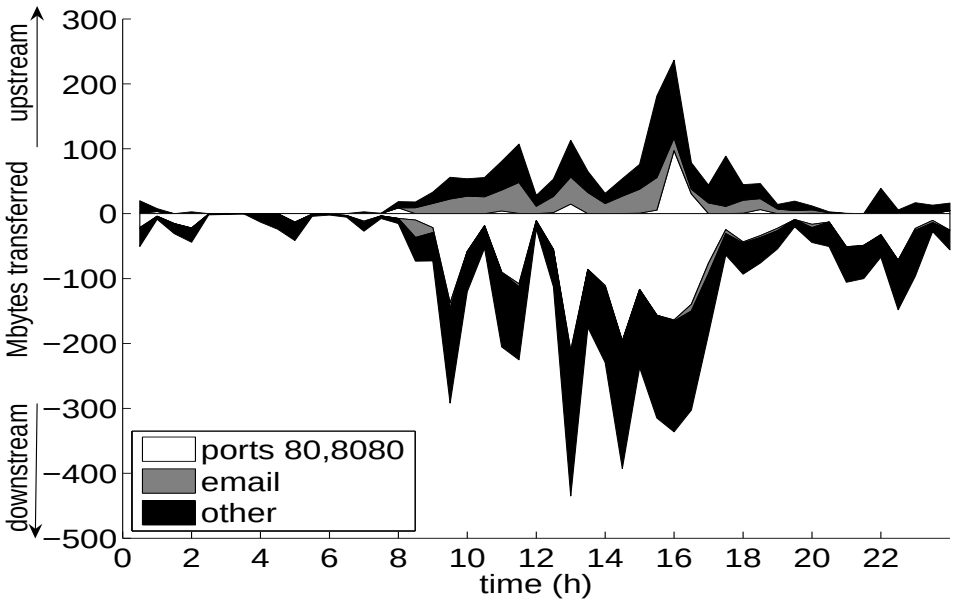



Figure 9.10: Amount of transmitted bytes through saturated access link by different applications.

Figure 9.11 shows the main applications that see their throughput limited by a link that is not the access link. Here, the category of other applications is clearly dominating in terms of volume. Otherwise, we observe a mixture of applications. It is expected that the set of applications is diverse since this type of network limitation can occur at any point of the network regardless of the application behavior at the client side.

In the download direction, the total traffic that is limited by a distant bottleneck reaches in the late afternoon a proportion that, in terms of volume, is almost as important as the download traffic that is application limited. The fact that this traffic peaks late afternoon[1] may be an indication of higher overall network utilization just after working hours, not only within the France Telecom network but in wider scale, that causes more cross traffic in aggregating links. Note that at the same time, the amount of traffic limited by the access link is very low (Figure 9.10).

The number of clients with TCP configuration problems was so small that the application set is not representative, and we decided to exclude that analysis. Finally, we would like to point out that a comparison of the absolute traffic volumes of Figures 9.9 – 9.11 reveal that the application limitation category represents almost 80% of the total number of transmitted bytes.

### 9.3.4 Impact of Limiting Factors On Performance

Given the distributions of clients into different limitation categories, the next logical subject of study is the impact of a given limitation cause. We would especially like to know the throughput obtained by a client limited by a particular cause. Originally, we wanted to study also the

[1] An analysis of the IP addresses using Maxmind (http://www.maxmind.com/) revealed that most of the local clients exchange data primarily with peers/servers located in France or surrounding countries.

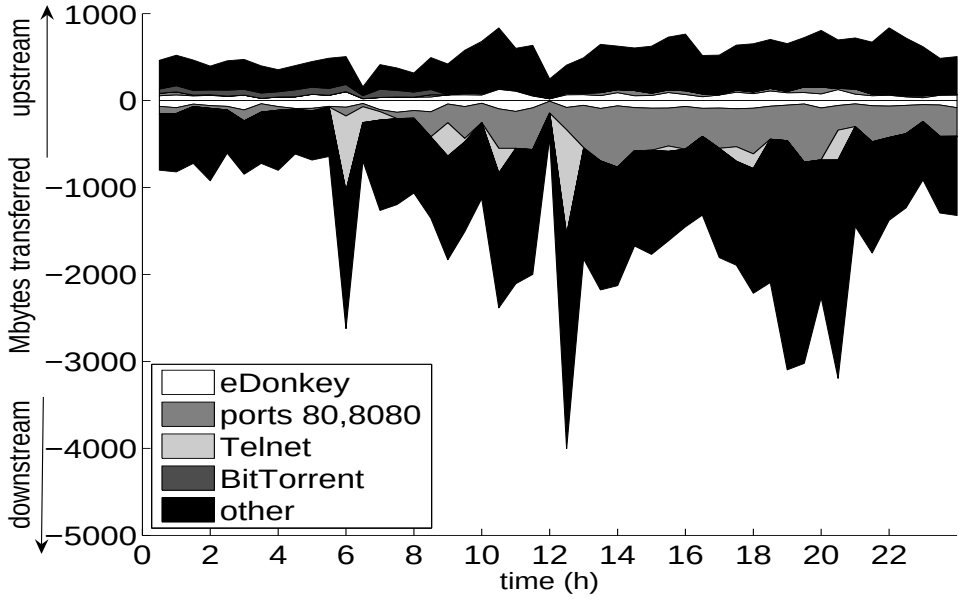



Figure 9.11: Amount of transmitted bytes whose rate is limited by a distant link by different applications.

evolution of local and distant RTT in the different limitation cases. By local RTT we mean the part of the total RTT from the host to our measurement point and back. Thus, distant RTT is the remaining part of the total RTT. These RTT values could potentially serve as identifier of certain limitation categories. For example, when the client saturates the access link, we expect the local RTT to be long with respect to the distant RTT. Unfortunately, we discovered a clock skew between the two probes. This clock skew means that we cannot rely on the RTT values, because packets of a connection may pass through one router downstream and through the other one upstream. Thus, we leave this analysis as a future subject of study.

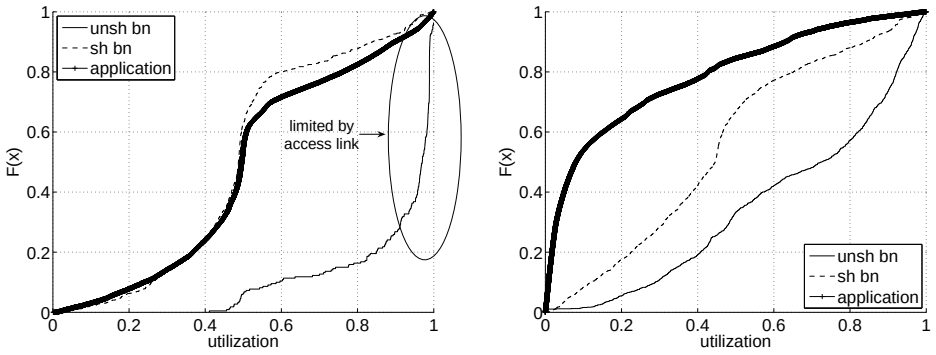
We decided to study link utilization for clients instead of absolute throughput, because clients have different link capacities and we want to understand how far from the optimal, i.e. link saturation, the performance is in different situations. As the optimal case is limitation by saturated access link, we would like to study the utilization for the other categories, namely, application limitation, limitation by the network through a bottleneck link other than the access link, limitation by TCP configuration, and download interference by saturated uplink.

As before, we included in the analysis for each client only the traffic of the 30-minute period for which that client achieved its highest instantaneous throughput. We computed client's link utilization during ALPs and BTPs limited by different causes. In this way, we can quantify the impact of different limitation causes on the performance. Figure 9.12 shows CDF plots of the results.

We focus first on uplink utilization: We see that for the case of an unshared bottleneck, the utilization is in approximately 70% of the cases very close to one, which means that in these cases the uplink of the client is the bottleneck. In the remaining 30% of cases where we observe an unshared bottleneck, we see a link utilization between 0.4 and 0.85 that can be due to a distant access *down*-link, e.g. a peer that has lower downlink capacity than the uplink capacity

of the local peer, or due to simply misclassification. For the two other root causes, application limitation and shared bottleneck, the clients achieve in about 60% of the cases a link utilization of less than half the uplink capacity.

We can also notice that the CDF plots of the utilization during uploads limited by the application and shared bottleneck link have a peculiar shape: the utilization approaches the 50% value asymptotically. In the case of application limited traffic, this phenomenon is likely due to the upload rate limit imposed by P2P client applications, specifically eDonkey. They would allocate half of the uplink capacity to the aggregate upload traffic. The reason for the same phenomenon in the case of shared bottleneck limited transfers may be the same: While the P2P client application tries to maintain an maximum uplink utilization of 50% by enforcing a specific rate limit for each active TCP transfer, some of those transfers may get limited by the network, for instance. Consider a following scenario: The eDonkey application allocates a given TCP connection a certain amount of bandwidth by enforcing a rate limit. The application tries to maintain the rate at all times by scheduling the transfers of pieces of data of specific size at specific time intervals. However, if the end-to-end network path has equal amount or less available bandwidth than the rate allocated by the application, the application will try to send constantly data in order to fulfill the rate quota and the transfer will not be limited by the application. Nevertheless, the link utilization may still be close to the allocated 50% if the other active transfers meet their rate quota.




(a) upstream (b) downstream

Figure 9.12: CDF plot of access link utilization during ALPs (application) and BTPs limited by different causes. For each client, we considered only traffic of the 30 min period during which that client achieved the highest instantaneous throughput of the day.

If we look at the utilization of the downlink, we see that application limited traffic results most of the time in a very poor downlink utilization. Given that most of the application limited traffic is eDonkey traffic (cf. Figure 9.9), one might be tempted to explain this low utilization by that fact that most likely the peer that sources the data has an asymmetric connection with the uplink capacity being much lower than the downlink capacity of the receiving peer[2]. However,

[2]Maxmind also reported that a clear majority of the distant IPs that the heavy-hitters communicated with were clients of ISPs providing residential services.

a downloading peer has usually multiple parallel download connections, which in aggregation should be able to fully utilize the downlink capacity. The fact that this is not the case seems to indicate that many users of eDonkey use the possibility to rate-limit their upload rate to a rate much lower than the capacity of their uplink. Figure 9.13, which plots the maximum instantaneous aggregate download rates achieved per-client for different applications, further underlines this effect. We see that the maximum aggregate download rates of P2P applications, eDonkey and BitTorrent, fall clearly behind the maximum download rates of FTP and port 80/8080 traffic.

A recent study of eDonkey transfers by ADSL clients [100] found that the average file download speed achieved was only a few KByte/sec. The traffic traces for their analysis was collected using the exact same set up at the same access network as for this analysis, only an approximately year earlier. However, that paper does not attempt to explain the origins of these low rates. Our findings seem to indicate that such a poor performance is not due to network or access link saturation but rather due to *eDonkey users drastically limiting the upload rate of their application.*

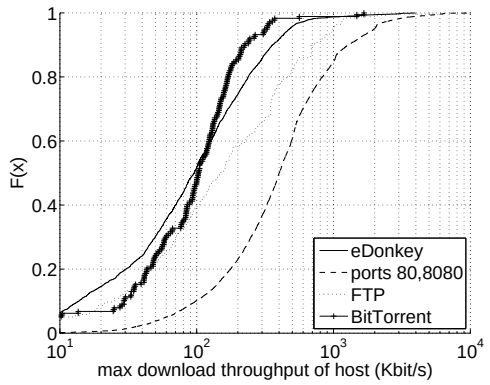



Figure 9.13: CDF plot of maximum aggregate per-host download throughput computed over five second intervals.

The previous research work in [81], [19], [75], and [90] have clearly made the case that a highly utilized upstream path can greatly damage the downstream performance. However, it is difficult to quantify the impact of a saturated uplink on download performance from our passive traffic measurements. We computed the ratios of the amount of bytes transferred downstream over the amount of bytes transferred upstream during a given download BTP. Two cases were distinguished: 1) the access uplink was saturated (utilization $\geq 90\%$) and 2) the access uplink was not saturated (utilization $< 90\%$) during the download BTP. We observed large differences for these two cases. Specifically, when the uplink is saturated, this ratio never exceeded 2, while the ratio reach values as high as 20 when the uplink was not saturated. However, it is not clear that these values reflect only the impact of the saturated uplink. The main reason is that we do not know whether during a particular download the client was even trying to upload data continuously at full rate when the uplink was not saturated. It may well be that the client applications did not need to upload data at that particular time.

### 9.3.5 Comparison With Other Related Analysis Work

In [118], Zhang et al. performed flow-level root cause analysis of TCP throughput with T-RAT (see Section 7.4). They analyzed packet traces collected at high speed access links connecting two sites to the Internet; a peering link between two Tier 1 providers; and two sites on a backbone network. As results, the authors reported that, in terms of traffic volumes, congestion (similar to network limitation in our vocabulary) was the most common limiting factor followed by host window limitation (TCP end-point in our vocabulary). It is important to notice that their studies were based on data collected in 2001-2002. At that time, the popularity of P2P applications such as eDonkey was far from what it is today.

In order to understand if our results are specific to this particular access network, we applied our RCA tool also to other publicly available packet traces collected at an ADSL access network in Netherlands (http://m2c-a.cs.utwente.nl/repository/). We looked at two 15-minute traces: one captured in 2002 and another one in 2004. A similar port based study than in Section 9.2.1 showed that in the 2002 trace, the applications generating most traffic where FTP and applications using ports 80 and 8080, while eDonkey and BitTorrent were dominating in the 2004 trace. We were unable to perform similar client-level study due to lack of knowledge about local client IP addresses and limited capture durations. However, simple connection-level RCA revealed that in the 2002 trace around 40% of bytes were throughput limited by the application. In the 2004 trace, this percentage was already roughly 65%, which demonstrates the impact of the increase in P2P application traffic.

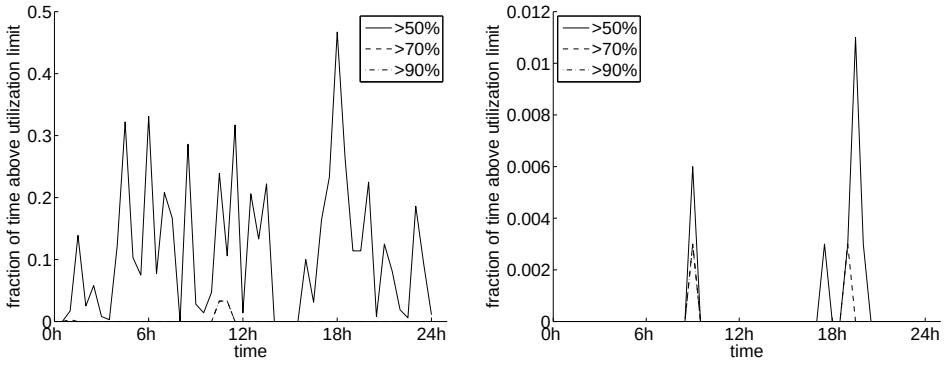
## 9.4 Closer Look at a Few Example Clients

Finally, we selected a few interesting examples of clients and studied them in more detail. In order to visualize the activity of the client, we show *activity diagrams* that plots for a given client all connections as rectangles on a coordinate system where the x-axis represents time and y-axis the throughput. The width of each rectangle represents the duration, height represents the average throughput, and, consequently, area represents the volume of the connection[3]. Thus, starting time of the connection is the left boundary and ending time the right boundary of the rectangle. Each rectangle is placed vertically as low as possible in such a way that no rectangle overlaps with another one. Upstream connections are plotted above the line $y = 0$ and downstream connections below that line. Downstream connections are thus mirrored with respect to the x-axis. The fill pattern of the rectangle represents the application that generated the connection. Different applications are detected from the TCP port number used. We selected only connections that transmit at least 10 KB to avoid too messy plots.

The first example, *client A*, is a client who is active the entire 24 hours. This client belongs to the set of top 15% clients in terms of transmitted bytes. Figure 9.14 shows the client's utilization for the day using the thresholds from Table 9.2 (note the different scales of Y-axis for uplink and downlink). We see that while being all the time active, client A transfers data with modest rates. However, there are occasional short periods of time when the client achieves high throughput. Figure 9.15 shows three hours of typical activity for this client. Same kind

[3]Note that we only show the scale for Y-axis but not cumulative values because the absolute vertical position of the upper boundary of a particular rectangle does not necessarily signify the instantaneous throughput. In other words, these plots are not similar to area plots, as Figure E.12, in that they do not represent accurately the cumulative throughput.

of activity persists throughout the day. We make two main observations: First, the client used mainly eDonkey, which produced the low throughput activity persisting throughout the day. We can visually identify from the plot a typical size of an eDonkey connection by looking at the areas of the rectangles. This size is approximately 9 MB which is the typical chunk size for eDonkey transfers, i.e. each file larger than that is divided into 9 MB chunks. If we compare the areas of the rectangles representing eDonkey connections to the rectangles of port 80/8080 and unknown traffic, we see that those connections are often of the same size. Moreover, many of these port 80/8080 and unknown transfers are upstream. Thus, as a second remark, there is probably a significant amount of eDonkey traffic disguised as Web traffic or that is not using the standard port numbers (6881-6889). Just before the activity represented in Figure 9.15, client A had different activities. The client produced occasional fast downloads originating from port 80/8080 that caused peak rates for this client. We zoomed in to one of these which took place at around 8h30, the activity diagram is in Figure 9.16. Note that the scale is more than ten times larger in this plot than in the plot of Figure 9.15. We see that it contains several simultaneous connections and when one ends there is another one that starts. The largest connections carry around 100 KB. Thus, it seems likely to be a download of a Web page using parallel and persistent (HTTP/1.1) connections. Clear majority of the bytes (most of the time more than 85%) uploaded and downloaded by this client are rate limited by the application on top.




(a) uplink (b) downlink

Figure 9.14: Client A's link utilization per half hour period during the day.

The second example, *client B*, is a client who is active only about an hour (within the period from 9pm to 10:30pm). As Figure 9.17 shows, there is practically no upstream activity (all upstream connections are smaller than 10 KB). All the downstream connections are port 80/8080 traffic. It is very likely that this client is merely browsing the Web. In that case all the upstream traffic would consist of HTTP requests that are very small.

The third example client, *client C*, transfers a lot of bytes but is active only for a few hours. Thus, also this client belongs to the set of top 15% clients in terms of transmitted bytes but differs from the first example client in that the transfers are faster and total activity time is shorter. Figure 9.18 shows the activity diagram for this client. Except for a few eDonkey

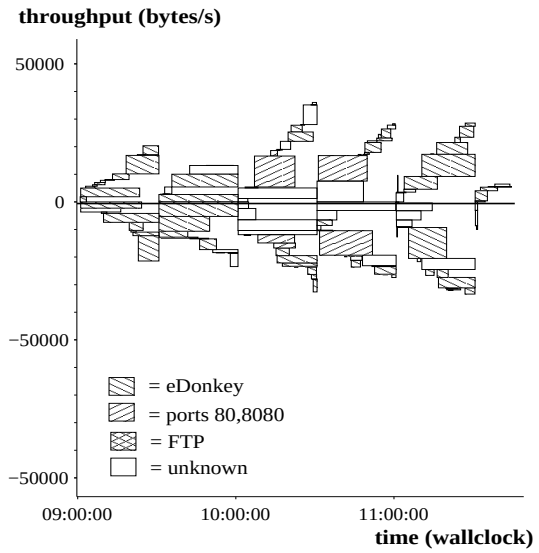



Figure 9.15: Three-hour piece of an activity plot for client A that transfers a lot of bytes and is active all day.

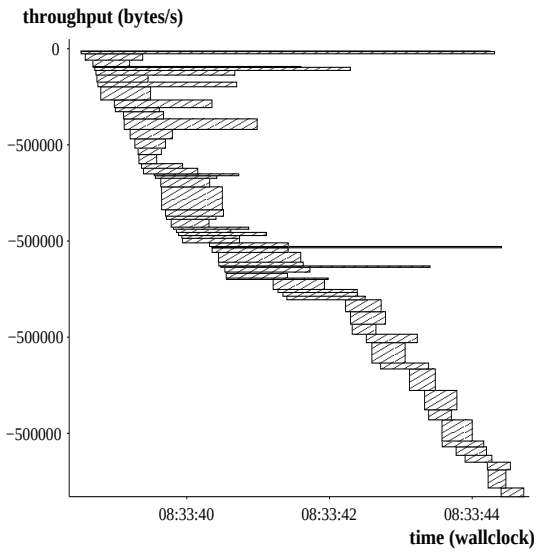



Figure 9.16: Close up of client A's connections originating likely from Web browsing that cause the peak download rates.

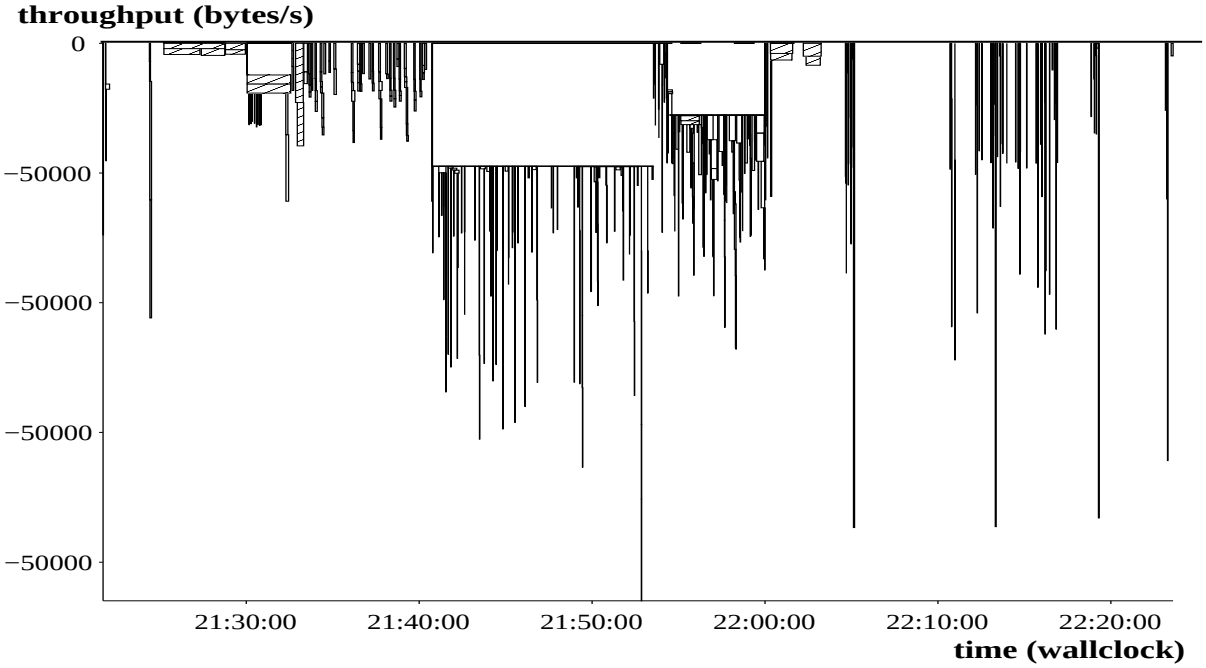



Figure 9.17: Activity plot for client B that is active only during an hour and most likely browses the Web.

connections all the connections for this client are using ports 80 and 8080. However, judging by the size of the connections and the fact that there are large amounts of uploaded bytes, it is

likely that most of the activity after 6pm is eDonkey as well, only hidden behind HTTP ports. The activity before 6pm is different from the rest and looks similar to that in Figure 9.17. Thus, this client probably browsed the Web for a while before starting to utilize eDonkey. We see the difference also from the above 0% link utilization plotted in Figure 9.19 which stays below one before the P2P activity starts and then stays constantly at one. The vast majority of the bytes transferred by this client (80-100%) are rate limited by the application. Thus, the only difference with the first example client we analyzed is that this client achieves clearly higher rates probably because of faster ADSL subscription.

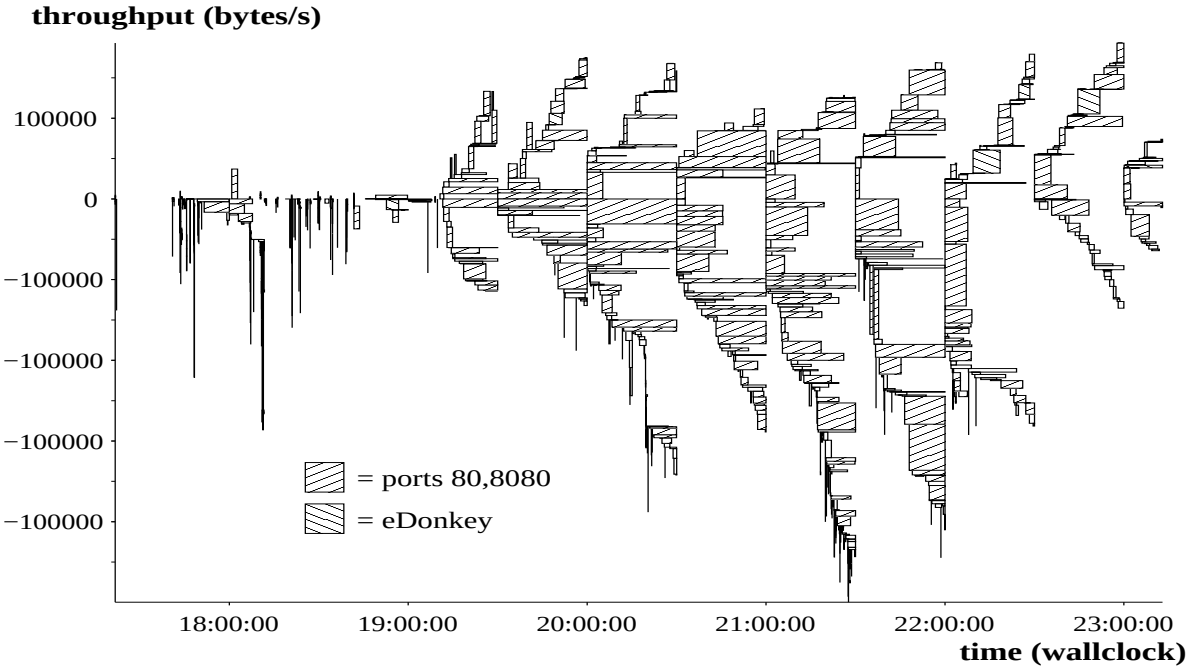
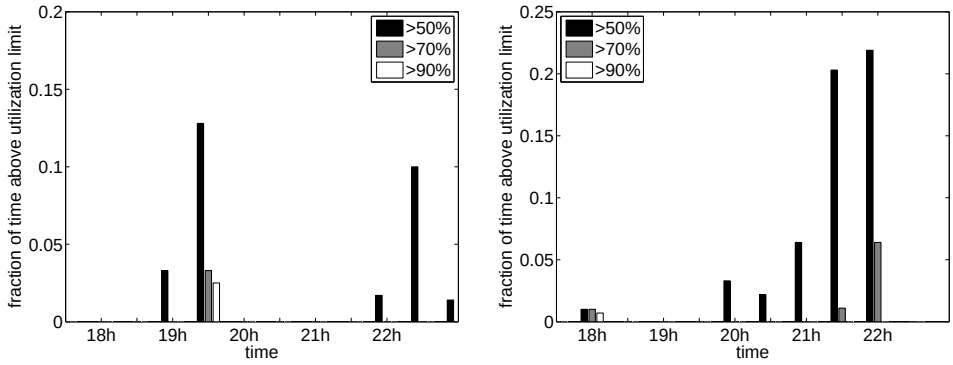



Figure 9.18: Activity plot for client C that transfers a lot of bytes but is active only approximately five hours.

The examples that we showed in this Section illustrate the power of simple visual analysis tools in the context of exploratory data analysis. Unfortunately, it is far from trivial and out of scope of this study to automatize this type of client behavioral analysis that we performed visually.




(a) uplink (b) downlink

Figure 9.19: Client C's link utilization per half hour period during the day.

## 9.5 Conclusions

We presented in this chapter a case study on the performance analysis of clients of an ADSL access network. We focused on throughput as the performance metric. We defined a taxonomy of the performance limiting factors and explained how they could be detected using our root cause analysis techniques. The analysis results revealed that large majority of the clients experience performance limitation by the application. This limitation is likely due to upload rate limits imposed by P2P client applications – P2P is the dominating application category in terms of traffic volumes. The consequence of such a limitation cause in conjunction with typically asymmetric connectivity of peers, where the uplink capacity is factors smaller than the downlink capacity, is poor download performance. As a consequence, clients rarely saturate their access links, especially downlinks.

The implication of such a low access link utilization is naturally low utilization of the entire access network, which is beneficial for the service provider. However, there is a caveat: the utilization and, consequently, the traffic volumes can change dramatically in case a new type of popular P2P application is deployed or an already used popular P2P application is upgraded to utilize the uplink in a different, more intelligent way.

In the future, it would be interesting to perform a similar study on traffic traces that spans over several days. Such analysis would help to understand how the client behavior and their performance limitations evolve. That kind of a study could also give further insights of the implications of the increase of number of clients or deployment of new popular applications to this particular access network (similarly to the study in [36]), for instance. While the study over a single day presented in this chapter brings many useful insights, it cannot alone be considered fully representative. The main goal of this study was to show how our techniques can be applied to such a scenario and to produce the first results to guide further research.

# Conclusions for Part III

Part III was the element that glued the two first parts, methodology and techniques, together through the case study. This case study on performance analysis of clients of an ADSL access network demonstrated the potential of the root cause analysis techniques in an interesting real world scenario.

In the last concluding chapter of the thesis, we reevaluate the thesis claims made in the beginning and evaluated how we fulfilled those claims. We also take a critical look back at the entire thesis work and discuss the possible future directions of research for which this thesis has laid the basis.

# CHAPTER 10

# Thesis Conclusions

In this chapter, we revisit the thesis claims we stated in the introduction and evaluate how well we succeeded to fulfill those claims. We also discuss the thesis work in general and identify what could have been done differently and better. Finally, we give our vision on how the research work done for this thesis could be extended in future.

## 10.1 Evaluation of the Thesis Work

### 10.1.1 Claims and Contributions

In this thesis, we have made four claims that constitute the contributions of this thesis. We revisit and evaluate how well we fulfilled each one of them. The thesis claims are:

I. *We can overcome many of the problems in management and suboptimal analysis process cycle in passive packet-level traffic analysis by adopting a DBMS-based approach.*

   (a) *An implementation of such an approach performs feasibly.*

   (b) *We can significantly improve the performance of such a system with I/O optimizations based on characteristics of packet-level traffic data and popularity assessment of queries.*

   In Chapter 2, we described the problems encountered in passive traffic analysis and suggested that a DBMS-based approach could potentially help to overcome them. Then we described in Chapter 3 our approach based on an object-relational DBMS and explained how this approach addresses the problems discussed in Chapter 2. Furthermore, we have implemented a prototype of such an approach that operates on PostgreSQL, which we describe also in that chapter. We demonstrated with performance measurements that such a system performs feasibly in Chapter 4. Furthermore, we identified the performance bottleneck for our prototype, and showed how we can greatly improve the performance through standard I/O optimizations.

II. *It is possible to infer root causes for TCP throughput from bidirectional packet traces recorded passively in a single measurement point located anywhere on the TCP/IP path (end-point or in the middle). Furthermore, unidirectional traffic traces are insufficient.*

In Chapter 5, we described the different potential limiting factors for the throughput of a TCP connection, and showed through examples how they manifest themselves in the traffic, e.g. as specific IAT patterns. In Chapters 6 and 7, our techniques to infer these limiting factors are presented. The techniques are based on extracting and analyzing a set of metrics from packet traces recorded at a single observation point. Our metrics are quantitative measures and require us to calibrate our root cause classification scheme in order to produce qualitative classification results (e.g. the dominating limitation cause for a given TCP transfer is an unshared bottleneck link on the network path). We validated our techniques as best we could and compared them to the only other currently existing tool, T-RAT [118], which demonstrated that our techniques are able to perform correctly also in several cases that are too complex for T-RAT.

III. *Different Internet applications interact in complex and different ways with TCP. That is why the effects of applications need to be first filtered out whenever studying the characteristics of the underlying TCP/IP end-to-end path.*

In Chapter 5, we enumerated the different types of applications based on their way of interacting with TCP. Chapter 6 focused specifically on analyzing this interaction of applications with TCP. We described our IM algorithm to partition a TCP connection into two types of periods: Bulk Transfer Periods (BTP) and Application Limited Periods (ALP). We stated that the BTPs only manifest the properties of the underlying end-to-end network path, while ALPs can manifest the properties of the application. We showed through several examples using real traffic traces originating from different applications how the presence of the application may bias the end-to-end throughput and delay measurements.

IV. *Our TCP root cause analysis methods implemented with our DBMS-based approach for traffic analysis enable:*

- *performance evaluation of Internet application protocols,*
- *detailed evaluation of network utilization,*
- *identification of certain TCP configuration problems.*

In Chapter 9, we presented a case study on performance analysis of clients of a commercial ADSL access network. We used our prototype of the InTraBase customized to support our root cause analysis techniques and applied this system on a full-day of traffic traces captured at the edge of France Telecom's ADSL access network. We defined a set of performance limiting factors on client level and applied our root cause analysis techniques on TCP connection level to infer these client limitation factors. We discovered surprisingly that the overall poor performance of peer-to-peer applications is mostly due to upload limits set by the client applications. In addition, we showed that clients rarely saturate their access links and provided strong evidence of the fact that a potential bottleneck link resides usually away from the client. We demonstrated also that some analyzed clients were very likely suffering from TCP configuration problems, namely too low default maximum receiver advertised window. This case study showed that TCP connection-level techniques

combined with our InTraBase methodology can be effectively used to study ADSL client-level performance, for instance.

### 10.1.2 Critical Viewpoint

Looking back at the entire process of producing results for the thesis, perhaps the partitioning of the time spent in each part could have been better. We spent a long time defining and implementing the methodology, i.e. producing the content for Part I. Similarly, a lot of time was spent on improving the root cause analysis techniques, content of Part II, to function as well as possible in all possible scenarios. What we realized at some point was that this work has no end. The diversity of the traffic encountered in the Internet today is overwhelming and continues to increase. Thus, designing methods and algorithms that are able to handle each possible case is not realistic. As a consequence, we had eventually less time for applying our techniques in specific cases, as in the study presented in Chapter 9. On the other hand, the amount of material in this thesis is large, even though we only present one such case study.

This thesis work included a relatively large amount of implementation work. We ended up with approximately 14000 lines of PL/pgSQL and PL/R code for the database functions. Naturally, time spent for implementation is always time away from other work. Therefore, while the scope of this thesis can be considered large, as performance of TCP connections touches the entire Internet and majority of its applications, it would have been desirable to spend more time to enlarge the scope, e.g. to include wireless traffic and short connections as particular scenarios and analyze the performance problems encountered with TCP in these scenarios.

TCP traffic analysis is a fascinating domain of research. However, it can also be somewhat unrewarding, because research around TCP is no longer "trendy". In other words, many researchers feel that all problems around TCP have already been solved, while in reality, new questions are arising constantly. In addition, it seems that the research community of Internet measurements is more interested in addressing particular small problems rather than doing more persistent and farsighted research work. That is why an incremental Ph.D. work, such as presented in this thesis, is very challenging to publish in the conferences of this community. Instead, a Ph.D. thesis that is knitted together from publications that each address a particular small problem remotely related to each other would have been easier to "sell" on the course. Nevertheless, such an approach would have lead to a thesis composed of bits and pieces while this thesis is a consistent piece of work, which is the outcome that we wanted to have.

## 10.2 Future Work

We identify future research tasks and directions in two categories: first, related to our InTraBase methodology and, second, directly on the root cause analysis of TCP throughput.

### 10.2.1 InTraBase

So far, we have implemented a prototype of the InTraBase only based on PostgreSQL (PgInTraBase). It would be interesting from the point of view of performance comparison to port this prototype to another DBMS, e.g. Oracle. In fact, this work is already ongoing, but it is still in a very preliminary stage. We envision a prototype running on Oracle in early 2007 after which we will compare the performance of the two prototypes.

In the current prototype, we implemented the root cause analysis techniques entirely as PL/pgSQL functions (refer to Section 8.2), as PL/pgSQL functions. The advantage of this approach is that the functions are perfectly portable as they are not compiled. Also, they are in many situations simpler to program since one can directly embed SQL queries in the syntax, for instance. The main disadvantage is performance and is due to the language itself: it has little support for different data structures. Because of this, each time a root cause analysis function is called in order to analyze a specific connection, the data is read again from the disk. In this way, the data is read from the disk as many times as the number of root cause analysis functions for each connection, which is an unnecessary burden even with the I/O optimizations presented in Section 4.2.3. A better approach would be to read the data once in the memory and then process it with each of the functions at a time, which is currently impossible with PL/pgSQL. Thus, one could implement a database module written in C/C++, for instance, that would read data of a given connection from the disk and then call each root cause analysis function written in PL/pgSQL.

### 10.2.2 Root Cause Analysis of TCP Throughput

We have limited the scope of our work into the analysis of long-lived TCP connections. However, previous research work has focused also specifically on the case of short TCP connections, as we pointed out in Section 5.4. Our techniques could potentially provide interesting insights when applied to those cases as well. In [120], the authors state that at that time the Internet was dominated by short HTTP connections. However, it is no longer the case since the emergence of HTTP/1.1 which primarily uses persistent connections. Therefore, it would be of interest to perform a comparative measurement study on older and recent traces, for instance.

As we explained in Section 5.4, we also restricted the scope to include only FIFO scheduled (with drop-tail policy) traffic. While this is generally not a very strong assumption to make considering the overall Internet traffic, there is a growing number of cases where FIFO scheduling is not used. These cases occur typically in broadband access networks, e.g. cable modem and wireless (e.g. WLAN 802.11) access networks. Since there is a growing interest in wireless networking, it would be of interest to evaluate to which extent our root cause analysis techniques could be applied in these scenarios. One of the challenges would be the estimation of the capacity of the TCP/IP path. We currently perform the estimation using the PPrate [47] tool. This tool assumes FIFO scheduling and as such is likely, depending on the particular situation, to produce incorrect capacity estimates for traffic originating from an 802.11 access network, for instance.

Our root cause analysis techniques involve many computations. Therefore, it is challenging to apply them in cases where on-line analysis is required, which was a conscious decision when we defined the scope for this thesis. However, it would be interesting to investigate whether it is possible to transform some of the root cause analysis techniques into on-line streaming style algorithms. While it may be too challenging when monitoring a high speed link due to the vast amount of data to be processed, it could be doable as a end host analysis tool, for instance. In that case, the host would run a root cause analysis software that analyzes all the TCP connections in real time. Every time the IM algorithm (refer to Section 6.1) identifies and stores a new BTP, the limitation scores for this BTP would be computed and it would be classified based on the dominant limitation cause using the techniques described in Chapter 7. We could also define an upper limit for the size of a BTP in number of packets, for example. In this way, the software would constantly give root cause information even in the case of very

long transfers. Such a software could serve as merely informing the user about the limitations or automatically react accordingly. For example, if transport limitation is observed frequently, the slow start threshold could be increased. Similarly, if receiver limitation is observed because of a too low default receiver advertised window during downloads, the default size of this window could be increased or receiver window scaling activated.

In case of large amounts of traffic data, it is possible to take another approach. After analyzing the root causes using the full-blown machinery, we could try to find simpler metrics to identify a given limitation cause that could be used in a particular scenario. For example, if we were to monitor a network at the edge, in a similar way that we did in the case study of Chapter 9, we could try to look at variations of local (between a local host and the observation point) and distant (between a distant host and the observation point) RTTs to detect the presence of local and distant bottleneck links (the RTT grows when queuing delay at a congested link increases), respectively. In this way, we would first do heavy computations in order to produce the "reference" root cause knowledge and use that knowledge to find the metrics that are easy to compute, and afterwards use only those metrics to perform faster future analysis.

As we pointed out in Chapter 9, it would be interesting to extend the case study on performance limitations of ADSL clients to include traffic traces that span over several days. The results from the analysis of a single day are a good starting point but by no means the finish line. There are also a number things that we could try to do better in the next study. For example, we could try to obtain the subscription rates of clients, attempt to find a way to identify clients by other means than dynamic IP addresses, and utilize more advanced techniques for application identification.

In Section 6.5.2, we raised a point on another potential direction of future work. We showed that a study on the properties of the ALPs and BTPs for a given TCP connection can reveal much information about the behavior and nature of the application. We gave examples of different applications using simple metrics: ratios of duration, volume, and throughput between BTPs and the entire connections (including the ALPs). This sort of metrics could potentially be used to categorize applications by their behavior. One could even imagine fingerprinting certain types of applications so that particular value ranges of these metrics serve as signatures.

# BIBLIOGRAPHY


[1] "Daytona: http://www.research.att.com/projects/daytona/".

[2] "Large-scale Monitoring of Broadband Internet Infrastructures (LOBSTER): http://www.ist-lobster.org/".

[3] "M2C Measurement Data Repository: http://m2c-a.cs.utwente.nl/repository/".

[4] "MySQL: http://www.mysql.com/".

[5] "NIST Net home page: http://www-x.antd.nist.gov/nistnet/".

[6] "Official mirror sites for the Fedora Core Linux distribution: http://fedora.redhat.com/download/mirrors.html".

[7] "Packeteer PacketShaper: http://www.packeteer.com/products/packetshaper/".

[8] "PL/R - R Procedural Language for PostgreSQL: http://www.joeconway.com/plr/".

[9] "PostgreSQL 7.4 Documentation (PL/pgSQL - SQL Procedural Language): http://www.postgresql.org/docs/current/static/plpgsql.html".

[10] "PostgreSQL: http://www.postgresql.org/".

[11] "The R Project for Statistical Computing: http://www.r-project.org/".

[12] "rshaper: http://freshmeat.net/projects/rshaper/".

[13] "Tcptrace: http://www.tcptrace.org/".

[14] J. Aikat et al., "Variability in TCP Round-trip Times", In *Internet Measurement Conference 2003*, October 2003.

[15] M. Allman, W. M. Eddy, and S. Ostermann, "Estimating loss rates with TCP", *SIGMETRICS Perform. Eval. Rev.*, 31(3):12–24, 2003.

[16] M. Allman and A. Falk, "On the effective evaluation of TCP", *Comput. Commun. Rev.*, 29(5):59–70, 1999.

[17] E. Altman, K. Avrachenkov, and C. Barakat, "A stochastic model of TCP/IP with stationary random losses", *IEEE/ACM Trans. Netw.*, 13(2):356–369, 2005.

[18] D. Arifler, G. de Veciana, and B. L. Evans, "Network Tomography Based on Flow Level Measurements", In *Proceedings of IEEE Int. Conf. on Acoustics, Speech, and Signal Proc. (ICASSP)*, 2004.

[19] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, "The effects of asymmetry on TCP performance", In *MobiCom '97: Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pp. 77–89, New York, NY, USA, 1997, ACM Press.

[20] H. Balakrishnan, M. Stemm, S. Seshan, and R. H. Katz, "Analyzing stability in wide-area network performance", In *SIGMETRICS '97: Proceedings of the ACM SIGMETRICS*, pp. 2–12, 1997.

[21] M. Baldi, E. Baralis, and F. Risso, "Data Mining Techniques for Effective and Scalable Traffic Analysis", In *9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, May 2005.

[22] C. Barakat, "TCP/IP Modeling and Validation", *IEEE Network*, 15(3):38–47, 2001.

[23] C. Barakat and E. Altman, "Performance of Short TCP Transfers", In *Proc. Networking 2000*, May 2000.

[24] P. Barford and M. Crovella, "Critical Path Analysis of TCP Transactions", In *Proc. of ACM SIGCOMM'00*, Stockholm, Sweden, August 2000.

[25] S. Baset and H. Schulzrinne, "An Analysis of the Skype P2P Internet Telephony Protocol", CUCS-039-04, Department of Computer Science, Columbia University, 2004.

[26] J. Bellardo and S. Savage, "Measuring packet reordering", In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 97–105, New York, NY, USA, 2002, ACM Press.

[27] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly", *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26, 2006.

[28] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0", RFC 1945 (Informational), May 1996.

[29] S. Biaz and N. Vaidya, "Is the Round-Trip-Time correlated with the number of packets in flight", In *ACM Sigcomm Internet Measurement Conference 2003*, October 2003.

[30] E. Blanton and M. Allman, "On the Impact of Bursting on TCP Performance", In *Proceedings of Passive and Active Measurements (PAM)*, 2005.

[31] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", In *Proceedings of ACM SIGCOMM '94*, London, England, 1994.

[32] E. Brosh, G. Lubetzky-Sharon, and Y. Shavitt, "Spatial-Temporal Analysis of Passive TCP Measurements", In *Proceedings of IEEE Infocom '05*, apr 2005.

[33] J. Charzinski, "Observed Performance of Elastic Internet Applications", *Computer Communication*, 26(8):914–925, 2003.

[34] C.-M. Chen, M. Cochinwala, C. Petrone, M. Pucci, S. Samtani, P. Santa, and M. Mesiti, "Internet Traffic Warehouse", In *Proceedings of ACM SIGMOD*, pp. 550–558, 2000.

[35] W. Chen, Y. Huang, B. F. Ribeiro, K. Suh, H. Zhang, E. de Souza e Silva, J. F. Kurose, and D. F. Towsley, "Exploiting the IPID Field to Infer Network Path and End-System Characteristics.", In *Proceedings of Passive and Active Network Measurement (PAM)*, pp. 108–120, 2005.

[36] K. Cho, K. Fukuda, H. Esaki, and A. Kato, "The Impact and Implications of the Growth in Residential User-to-User Traffic", In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 207–218, New York, NY, USA, September 2006, ACM Press.

[37] M. Claypool, R. Kinicki, M. Li, J. Nichols, and H. Wu, "Inferring Queue Sizes in Access Networks by Active Measurement", In *Proceedings of Passive and Active Measurements(PAM)*, April 2004.

[38] B. Cohen, "Incentives to Build Robustness in BitTorrent", Technical Report, http://bitconjurer.org/BitTorrent/bittorrentecon.pdf, May 2003.

[39] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk, "The Gigascope Stream Database", *IEEE Data Eng. Bull., 26(1)*, 2003.

[40] M. Crovella and B. Krishnamurthy, *Internet Measurement: Infrastructure, Traffic and Applications*, John Wiley and Sons, Inc, 2006.

[41] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: evidence and possible causes", *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[42] B. Donnet, P. Raoult, T. Friedman, and M. Crovella, "Efficient algorithms for large-scale topology discovery", In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 327–338, New York, NY, USA, 2005, ACM Press.

[43] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet-dispersion techniques and a capacity-estimation methodology", *IEEE/ACM Trans. Netw.*, 12(6):963–977, 2004.

[44] N. Duffield, "Sampling for Passive Internet Measurement: A Review", *Statistical Science*, 19(3):472–498, 2004.

[45] N. Duffield, C. Lund, and M. Thorup, "Learn more, sample less: control of volume and variance in network measurement", *IEEE Transactions in Information Theory*, 51(5):1756–1775, 2005.

[46] M. Dyrna, "Network Tomography Tools", M.S. Thesis, TU Muenchen/Eurecom, September 2005.

[47] T. En-Najjary and G. Urvoy-Keller, "PPrate: A Passive Capacity Estimation Tool", *In the Proceedings of IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*, April 2006.

[48] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *SIGCOMM Comput. Commun. Rev.*, 26(3):5–21, June 1996.

[49] K. Farkas, P. Huang, B. Krishnamurthy, Y.Zhang, and J. Padhye, "Impact of TCP Variants on HTTP Performance", In *Proceedings of High Speed Networking '02*, May 2002.

[50] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", RFC 2068 (Proposed Standard), January 1997, Obsoleted by RFC 2616.

[51] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616 (Draft Standard), June 1999, Updated by RFC 2817.

[52] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 2582 (Experimental), April 1999, Obsoleted by RFC 3782.

[53] S. Floyd, "Connections with multiple congested gateways in packet-switched networks part 1: one-way traffic", *SIGCOMM Comput. Commun. Rev.*, 21(5):30–47, 1991.

[54] S. Floyd, "TCP and Explicit Congestion Notification", *SIGCOMM Comput. Commun. Rev.*, 24(5):10–23, October 1994.

[55] S. Fortin-Parisi and B. Sericola, "A Markov model of TCP throughput, goodput and slow start", *Perform. Eval.*, 58(2+3):89–108, 2004.

[56] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi, "Design and Deployment of a Passive Monitoring Infrastructure", In *Proceedings of Passive and Active Measurements(PAM)*, April 2001.

[57] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level Traffic Measurement from the Sprint IP Backbone", *IEEE Network Magazine*, November 2003.

[58] H. Garcia-Molina, J. D. Ullman, and J. D. Widom, *Database Systems: The Complete Book*, Prentice Hall, 2001, ISBN 0130319953.

[59] M. Garetto, R. L. Cigno, M. Meo, and M. A. Marsan, "Closed queueing network models of interacting long-lived TCP flows", *IEEE/ACM Trans. Netw.*, 12(2):300–311, 2004.

[60] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Heber, "Scientific data management in the coming decade", *SIGMOD Rec.*, 34(4):34–41, 2005.

[61] M. Grossglauser and J. Rexford, "Passive traffic measurement for IP operations", In K.Park and W. Willinger, editors, *The Internet as a Large-Scale Complex System*, Oxford University Press, 2002.

[62] L. Guo and I. Matta, "The War between Mice and Elephants", In *Proc. 9th IEEE International Conference on Network Protocols (ICNP)*, Riverside, CA, November 2001.

[63] A. Gurtov and S. Floyd, "Modeling wireless links for transport protocols", *SIGCOMM Comput. Commun. Rev.*, 34(2):85–96, 2004.

[64] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating internet bottlenecks: algorithms, measurements, and implications", In *Proceedings of ACM SIGCOMM 2004 Conference*, pp. 41–54, New York, NY, USA, 2004, ACM Press.

[65] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Al Hamra, and L. Garcés-Erice, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime", In *Passive and Active Measurements (PAM '04)*, April 2004.

[66] J. Jacobs and C. Humphrey, "Preserving Research Data", *Communications of the ACM*, 47(9):27–29, September 2004.

[67] V. Jacobson, "Congestion Avoidance and Control", In *Proceedings of ACM SIGCOMM '88*, pp. 314–329, Stanford, CA, August 1988.

[68] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance", RFC 1323 (Proposed Standard), May 1992.

[69] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput", *IEEE/ACM Transactions on Networking*, 11(4):537–549, 2003.

[70] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP Connections Characteristics from Passive Measurements", In *Proceedings of Infocom '04*, March 2004.

[71] S. Jaiswal, *Measurements-in-the-middle: Inferring end-end path properties and characteristics of TCP connections through passive measurements*, Ph.D. Thesis, Univ. of Massachusetts, Amherst, 2005.

[72] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP backbone", In *Proceedings of IEEE Infocom '03*, April 2003.

[73] H. Jiang and C. Dovrolis, "Source-level IP packet bursts: causes and effects", In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pp. 301–306, New York, NY, USA, 2003, ACM Press.

[74] H. Jiang and C. Dovrolis, "Why is the Internet traffic bursty in short (sub-RTT) time scales?", In *Proceedings of ACM SIGMETRICS*, June 2005.

[75] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Improving TCP throughput over two-way asymmetric links: analysis and solutions", In *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pp. 78–89, New York, NY, USA, 1998, ACM Press.

[76] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy, "Transport layer identification of P2P traffic", In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 121–134, New York, NY, USA, 2004, ACM Press.

[77] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks", In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 89–102, New York, NY, USA, 2002, ACM Press.

[78] S. Katti, D. Katabi, C. Blake, E. Kohler, and J. Strauss, "MultiQ: Automated Detection of Multiple Bottleneck Capacities Along a Path", In *Proceedings of Internet Measurement Conference (IMC '04)*, pp. 245–250, October 2004.

[79] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and K. Claffy, "The architecture of the CoralReef Internet Traffic monitoring software suite", In *Proceedings of Passive and Active Measurements(PAM)*, 2001.

[80] R. Kumar and J. Kaur, "Efficient beacon placement for network tomography", In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 181–186, New York, NY, USA, 2004, ACM Press.

[81] T. V. Lakshman, U. Madhow, and B. Suter, "TCP/IP performance with random loss and bidirectional congestion", *IEEE/ACM Trans. Netw.*, 8(5):541–555, 2000.

[82] K. Lakshminarayanan, V. N. Padmanabhan, and J. Padhye, "Bandwidth estimation in broadband access networks", In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 314–321, New York, NY, USA, 2004, ACM Press.

[83] J. Martin, A. Nilsson, and I. Rhee, "Delay-based congestion avoidance for TCP", *IEEE/ACM Trans. Netw.*, 11(3):356–369, 2003.

[84] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options", RFC 2018 (Proposed Standard), October 1996.

[85] M. Mathis, J. Heffner, and R. Reddy, "Web100: extended TCP instrumentation for research, education and diagnosis", *Comput. Commun. Rev.*, 33(3):69–79, 2003.

[86] M. Mathis and J. Mahdavi, "Forward acknowledgement: refining TCP congestion control", In *SIGCOMM '96*, pp. 281–291, New York, NY, USA, 1996, ACM Press.

[87] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, July 1997.

[88] A. Medina, M. Allman, and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet", *Comput. Commun. Rev.*, 35(2):37–52, April 2005.

[89] N. Megiddo and D. S. Modha, "ARC: A Self-tuning, Low Overhead Replacement Cache", In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST 03), San Franciso, CA*, March 2003.

[90] I. T. Ming-Chit, D. Jinsong, and W. Wang, "Improving TCP performance over asymmetric networks", *SIGCOMM Comput. Commun. Rev.*, 30(3):45–54, 2000.

[91] G. Minshall, Y. Saito, J. C. Mogul, and B. Verghese, "Application performance pitfalls and TCP's Nagle algorithm", *SIGMETRICS Perform. Eval. Rev.*, 27(4):36–44, 2000.

[92] M. Mitzenmacher and R. Rajaraman, "Towards More Complete Models of TCP Latency and Throughput", *J. Supercomput.*, 20(2):137–160, 2001.

[93] S. B. Moon and T. Roscoe, "Metadata Management of Terabyte Datasets from an IP Backbone Network: Experience and Challenges", In *Proceedings of Workshop on Network-Related Data Management (NRDM)*, 2001.

[94] J. Nagle, "Congestion control in IP/TCP internetworks", RFC 896, January 1984.

[95] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of HTTP/1.1, CSS1, and PNG", In *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 155–166, New York, NY, USA, 1997, ACM Press.

[96] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Reno Performance: A simple Model and Its Empricial Validation", *IEEE/ACM Transactions on Networking*, 8(2), April 2000.

[97] J. Padhye and S. Floyd, "On Inferring TCP Behavior", In *Proceedings of ACM SIGCOMM '01*, August 2001.

[98] V. Paxson, "Experiences with Internet Traffic Measurement and Analysis", Lecture at NTT Research, February 2004.

[99] T. Plagemann, V. Goebel, A. Bergamini, G. Tolu, G. Urvoy-Keller, and E. W. Biersack, "Using Data Stream Management Systems for Traffic Analysis - A Case Study", In *Passive and Active Measurements (PAM '04)*, April 2004.

[100] L. Plissonneau, J.-L. Costeux, and P. Brown, "Detailed Analysis of eDonkey Transfers on ADSL", In *Proceedings of EuroNGI*, 2006.

[101] J. Postel, "Transmission Control Protocol", RFC 793 (Standard), September 1981, Updated by RFC 3168.

[102] R. S. Prasad, M. Murray, C. Dovrolis, and K. Claffy, "Bandwidth Estimation: Metrics, Measurement Techniques, and Tools", *IEEE Network*, 17(6):27–35, November 2003.

[103] R. S. Prasad, M. Jain, and C. Dovrolis, "Socket Buffer Auto-Sizing for High-Performance Data Transfers", *Journal of Grid Computing*, 1(4):361–376, December 2003.

[104] C. B. Samios and M. K. Vernon, "Modeling the throughput of TCP Vegas", In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 71–81, New York, NY, USA, 2003, ACM Press.

[105] S. Shakkottai, N. Brownlee, and kc claffy, "A Study of Burstinesss in TCP Flows", In *Proc. Passive & Active Measurement: PAM-2005*, April 2005.

[106] D. Shasha and P. Bonnet, *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*, Morgan Kaufmann Publishers, 2003, ISBN 1-55860-753-6.

[107] M. Siekkinen, E. W. Biersack, V. Goebel, T. Plagemann, and G. Urvoy-Keller, "InTraBase: Integrated Traffic Analysis Based on a Database Management System", In *Proceedings of IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*, May 2005.

[108] M. Siekkinen, V. Goebel, and E. W. Biersack, "Object-Relational DBMS for Packet-Level Traffic Analysis: Case Study on Performance Optimization", In *Proceedings of IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*, April 2006.

[109] B. Sikdar, S. Kalyanaraman, and K. S. Vastola, "Analytic models for the latency and steady-state throughput of TCP tahoe, Reno, and SACK", *IEEE/ACM Trans. Netw.*, 11(6):959–971, 2003.

[110] B. Silverman, *Density Estimation for Statistics and Data Analysis*, CRC Press, 1986, ISBN 0412246201.

[111] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001 (Proposed Standard), January 1997, Obsoleted by RFC 2581.

[112] L. Tang and M. Crovella, "Virtual Landmarks for the Internet", In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, Miami Beach, Florida, USA, October 2003, ACM.

[113] R. Teixeira and J. Rexford, "A measurement framework for pin-pointing routing changes", In *NetT '04: Proceedings of the ACM SIGCOMM workshop on Network troubleshooting*, pp. 313–318, New York, NY, USA, 2004, ACM Press.

[114] Y. Tsang, M. Yildiz, P. Barford, and R. Nowak, "Network radar: tomography from round trip time measurements", In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 175–180, New York, NY, USA, 2004, ACM Press.

[115] G. Urvoy-Keller, "On the Stationarity of TCP Bulk Data Transfers", In *Passive and Active Measurements (PAM '05)*, March 2005.

[116] B. Veal, K. Li, and D. Lowenthal, "New Methods for Passive Estimation of TCP Round-Trip Times", In *Proceedings of Passive and Active Measurements(PAM)*, 2005.

[117] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One more bit is enough", In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 37–48, New York, NY, USA, 2005, ACM Press.

[118] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the Characteristics and Origins of Internet Flow Rates", In *Proceedings of ACM SIGCOMM '02*, Pittsburgh, PA, USA, August 2002.

[119] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the constancy of internet path properties", In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet measurment*, Nov 2001.

[120] Y. Zhang and L. Qiu, "Speeding Up Short Data Transfers: Theory, Architectural Support, and Simulation Results", In *The 10th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2000)*, 2000.

APPENDIX A

# List of Abbreviations, Acronyms, and Parameters

| | |
|---|---|
| ALP | Application Limited Period (Section 5.2.1) |
| B | byte |
| b | bit |
| Bps | bytes per second |
| bps | bits per second |
| BTP | Bulk Transfer Period (Section 5.2.1) |
| CDF | Cumulative Density Function |
| FTP | File Transfer Protocol |
| IP | Internet Protocol (Section 1.2) |
| MSS | Maximum Segment Size 5.1.1) |
| P2P | Peer-to-Peer |
| PDF | Probability Density Function |
| STP | Short Transfer Period (Section 6.1) |
| TCP | Transmission Control Protocol (Section 5.1) |
| TP | Transfer Period (Section 6.1) |
| T-RAT | Tcp Rate Analysis Tool (Section 7.4) |

APPENDIX B

# Detailed Analysis of PgInTraBase Performance Measurements

## B.1 Impact of Indexing and Clustering

Table B.1 contains the means of the measured values: exec time is the total execution time measured in wall clock time, CPU iowait is the time that the CPU was idle during which the system had an outstanding disk I/O request, CPU system is the CPU utilization time spent executing tasks at the kernel level, and number of sectors are those read from the hard disk (the size of a sector is 512 bytes). If the packet table is not indexed by the connections, the DBMS is forced to read through all the packets of the table when executing the most common query. Thus, the execution time should be more or less constant regardless of the number of packets queried[1]. When we indexed the data by connections, the means dropped dramatically. Clustering the indexed data had again a similar effect.

Table B.1: Average values of the measurements.

| **Gigabit Trace** | | | | |
|---|---|---|---|---|
| **test case** | **exec time (s)** | **CPU iowait (s)** | **CPU system (s)** | **sectors read** |
| unindexed & unclustered | 222 | 160 | 20.4 | - |
| indexed & unclustered | 6.07 | 5.22 | 0.338 | 30600 |
| indexed & clustered | 0.524 | 0.050 | 0.031 | 2160 |
| **BitTorrent Trace** | | | | |
| **test case** | **exec time (s)** | **CPU iowait (s)** | **CPU system (s)** | **sectors read** |
| unindexed & unclustered | 223 | 168 | 19.1 | 7280000 |
| indexed & unclustered | 7.81 | 3.96 | 0.529 | 42200 |
| indexed & clustered | 3.71 | 0.486 | 0.255 | 20400 |

[1] In the case of the Gigabit trace, we executed on a few randomly chosen connections since it would have taken too long with all the 3060 connections without indexing.

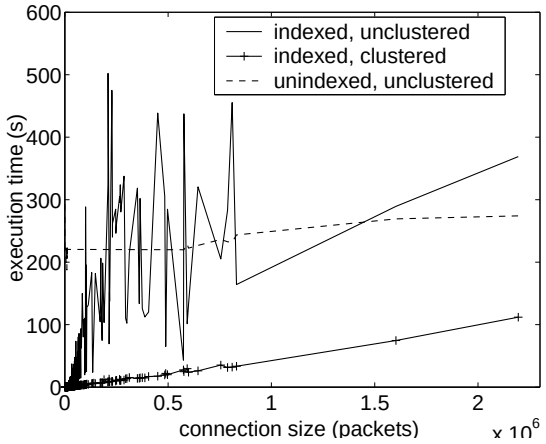



Figure B.1: Total execution time of the c-query for the Gigabit trace.

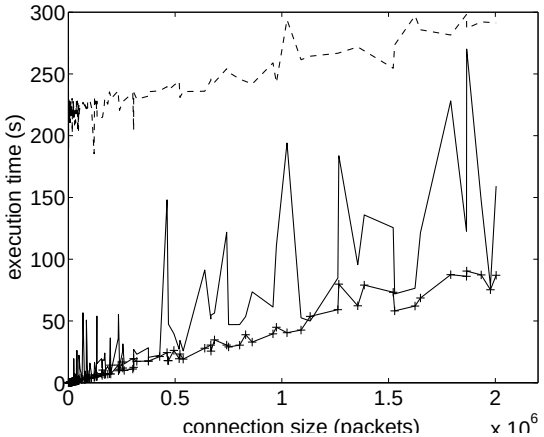



Figure B.2: Total execution time of the c-query for the BitTorrent trace.

Figures B.1 and B.2 reveal that while it is always good to use an index when querying a small number of packets, it is not necessarily the case when querying a large number of packets unless the data is clustered. In the case of the Gigabit trace, an index with unclustered data becomes virtually useless after the number of queried packets reaches a few hundred thousand. In the case of the BitTorrent trace, using an index proves to be always beneficial. The difference with respect to the number of parallel connections between the two types of the traces used, explained in Section 4.2.3.2 (Figure 4.7), is clearly visible in Figures B.1 and B.2: clustering has a much bigger impact on the Gigabit trace than on the BitTorrent trace captured on a low-speed link. We notice large variation in the execution times for the indexed but unclustered case. The variation is due to the fact that the packets of a given connection may be more or less dispersed within the trace depending on its thoughput related to the throughput of the other connections. In the plots for the Gigabit trace, we expect similar variation in the measured values with the larger connections as well, there simply were not many samples of those connections for it to be visible in these plots. In the case that data is clustered according to the index, the total execution time of the query scales more or less linearly, as expected.

The above observations are even more clearly visible in Figures B.3 and B.4 that show the CPU's iowait part of the total execution time. Figures B.5 and B.6 confirm that, in addition to the fact that the read head needs to move a lot more when seeking for unclustered data, many more sectors are generally required to be read from the disk.

Without indexing and clustering the time to execute the c-query of a single analysis task for each analyzed connection of the Gigabit trace would take approximately eight days. As for the BitTorrent trace, it would take one and a half days. Without I/O optimizations the total execution time is linearly dependent on the number of connections. When introducing an index, the total execution times drop to 5 h and 1.3 h, respectively. Finally, when the data is additionally clustered we obtain total execution times of 27 min and 36 min, respectively. We observe that after indexing and clustering the execution time is no longer dependent on the number of connections but rather depends on the number of packets.

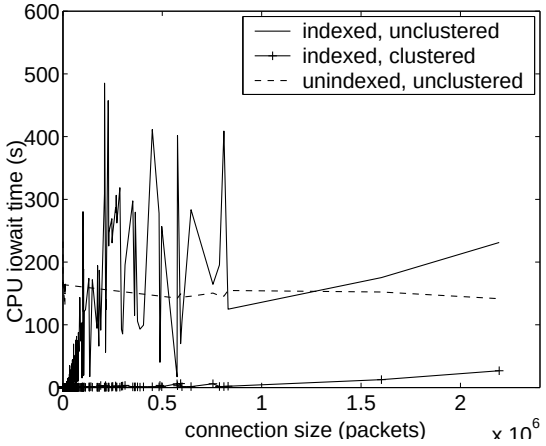



Figure B.3: CPU iowait time of the c-query for the Gigabit trace.

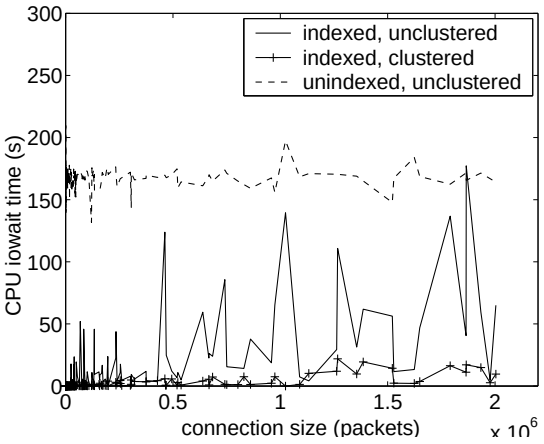



Figure B.4: CPU iowait time of the c-query for the BitTorrent trace.

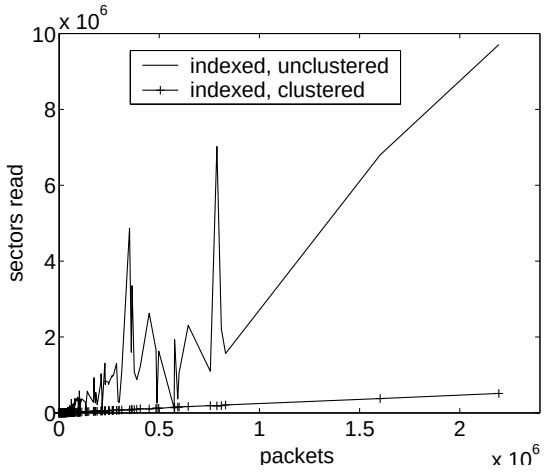



Figure B.5: Number of sectors read when executing the c-query for the Gigabit trace.

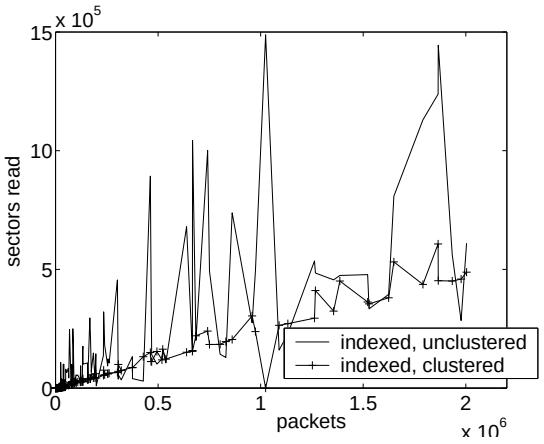



Figure B.6: Number of sectors read when executing the c-query for the BitTorrent trace.

## B.2 Measuring the Effectiveness of Caching

In order to measure the effect of caching, we executed several times the same c-query. First, the query was repeated 5 times in sequence for each connection in order to take advantage of available caching. During the second measurements, the query was executed once for all the connections and then the second time and so on until each query had been executed 5 times. In this way the caches got flushed between the subsequent executions of the same query. In order to focus on the effect of caching we excluded the first execution of each query from the measurements since only the subsequent ones benefit from the potential caching. As the results for both traces were similar, we only show those for the Gigabit trace.

Figure B.7 shows the evolution of the average execution time of the c-query in both cases. The difference between the cases with and without caching is minor: when caching, we gain

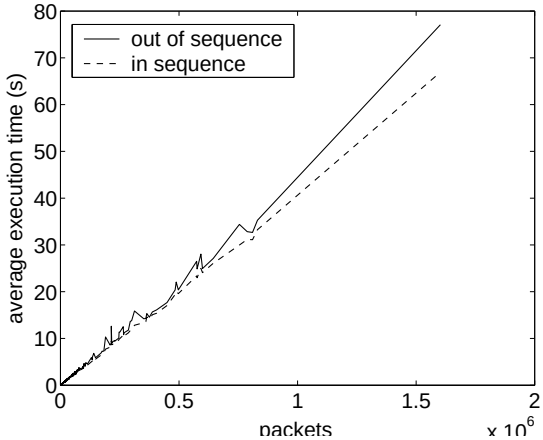



Figure B.7: Average execution time of the c-query for the Gigabit trace.

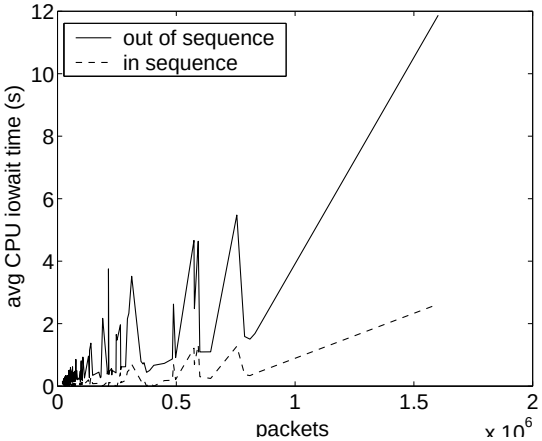



Figure B.8: Average CPU iowait time of the c-query for the Gigabit trace.

at most 13% in the execution time. However, Figure B.8 shows that the difference between the CPU iowait time is much more pronounced, which, together with the observations from Figure B.7, means that the processes executing the c-query were most of the time CPU bound and not I/O bound. Indeed, a closer look revealed that, instead of waiting for pending I/O operations, the CPU spent most of the time doing user level computations (as opposed to kernel level computations), which suggests that the DBMS itself kept the CPU busy.

## B.3 The Impact of Parallel I/O: RAID Striping

We had the option to further minimize the I/O time through parallelizing those operations using a RAID system in striping mode. However, a glance at Figures B.3 and B.4 already shows that we could not expect a very significant speedup since the iowait times are already very low after indexing and clustering. Rough measurements confirmed that indeed the gain is marginal and, therefore, we decided not to use striping. Nevertheless, the fact that the performance of the c-query with the InTraBase is mostly CPU bound after indexing and clustering can not be generalized. With a faster CPU, multiple CPUs with parallel processing support from the DBMS[2], or slower disks the situation might not be similar and parallel I/O could improve significantly the performance.

## B.4 DBMS as the Final Bottleneck

We have shown that after the I/O optimizations proposed, the performance of the c-query with the InTraBase is mostly CPU bound. On the average, the CPU spent approximately 84% of the time processing user level tasks with both of the traffic traces. We further investigated the origins of the main CPU activity by trying to exclude potential options.

[2]The version of PostgreSQL that we use does not support parallel processing and, thus, a single query that is executed as a single process does not benefit from both of the processors in our system.

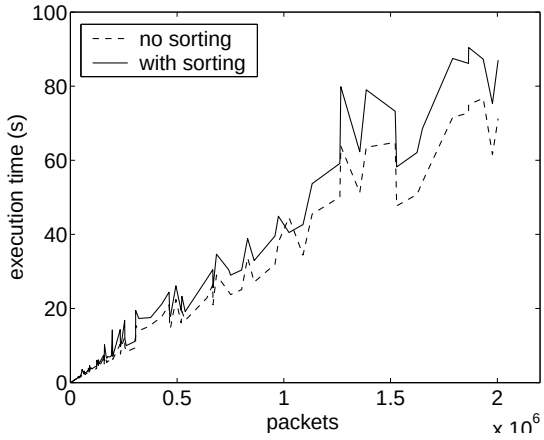



Figure B.9: Average execution time of the c-query with and without the `ORDER BY` clause for the BitTorrent trace.

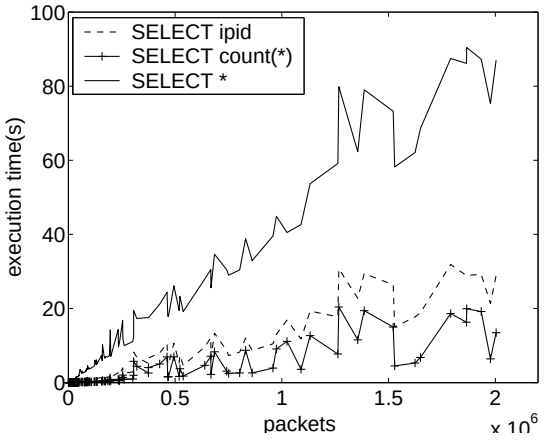



Figure B.10: Average execution times of the original c-query and a modified c-query that only counts packets for the BitTorrent trace.

First, we evaluated the effect of sorting (the c-query specified a chronological ordering of packets). We executed the c-query with and without the `ORDER BY` clause. The total execution times for the BitTorrent trace are plotted in Figure B.9. We can see that the execution times without sorting are at most 15% shorter. Indeed, ordering the packets in main memory, especially as it is done while reading them from the disk, should not be the main bottleneck.

Second, we investigated the role of the size of the result set of the query on the execution time. We compared the execution times of the c-query and two modified c-queries: one which computed only the number of packets and another one which selected only the `ipid` attribute[3] as the query result, i.e. `SELECT count(*)` or `SELECT ipid` instead of `SELECT *`, respectively. As Figure B.10 shows, the total execution times for both of the modified c-queries dropped up to 90% from the execution times of the original c-query. This result shows that the amount of CPU time spent on the user level computations is highly dependent on the result set size of the query. Thus, we have reason to assume that this CPU time goes to internal DBMS operations related to handling the result set tuples, e.g. transforming the physical storage format of the data into the logical format of the tuples, which we can not further optimize. This computational overhead is the price to pay for having the structured data and advanced querying facilities provided by the DBMS.

[3]We tried also other types of attributes and obtained similar results.

APPENDIX C

# Descriptions of Isolate & Merge Algorithms

## C.1 Isolate

**Algorithm 2**: The *isolate* procedure.

**function** $MSS(p_i) = \{1$, if $p_i$ has size equal to $MSS$; 0 otherwise$\}$;
**function** *estimate_rtt*(): returns a RTT estimate for the connection;
**input argument** $th \in [0, 10]$;
$rtt :=$ *estimate_rtt*();
$is\_active := 0$, $p_0 :=$ index of first data pkt `// start in inactive state`
**forall** *packets* $p_i \in \{$*sent data pkts* $\cup$ *received acks*$\}$ **do**
  **if** $is\_active == 1$ **then**
    **if** $p_i$ *is data packet* **then**
      $sum := sum + 1$;
      **if** $(\sum_{k \in \{prev\ 10\ data\ pkts\}} MSS(p_k)) \leq th \ \|$
      *(* $!MSS(p_0)$ & $IAT(p_{i-1}, p_i) > \frac{RTT}{2}$ & $p_i$ *not retransmission)* **then**
        **if** $sum \geq 130$ **then**
          store current transfer period as BTP;
        **else**
          store current transfer period as STP;
        $is\_active := 0$, $sum := 0$ `// start a new ALP`
      $p_0 := p_i$;
  **else**
    **if** $MSS(p_{i-2})$ & $MSS(p_{i-1})$ & $MSS(p_i)$ **then**
      store current ALP;
      $is\_active := 1$ `// start a new transfer period`

The *isolate* procedure, sketched in Algorithm 2, scans all the packets of a connection transmitted in a given direction and continuously switches between active and inactive states. Whenever it observes packets smaller than MSS more frequently than a predefined threshold ($th$) allows (e.g. as in Figure 5.6) or encounters a long idle time preceded by a small data packet (e.g. as in Figure 5.7) it switches to the inactive state and stores the current transfer period. If the number of packets belonging to this period is at least 130, a BTP is stored, and a STP oth-

erwise. During the inactive state, all packets observed belong to an ALP until three consecutive packets of size MSS are seen. At this point the algorithm switches back to the active state and stores the ALP.

For the analysis of each connection we require one single RTT estimate to provide a suitable threshold for the idle periods ($IAT > \frac{RTT}{2}$ in Algorithm 2). We specify the inter-arrival time ($IAT$) as the time delay during which no data pkts are sent and no pure acknowledgments received – in case of a two-way data transfer there can be piggybagged acks. We consider only the $IAT$s between a data packet following a data packet or an ack. When computing IAT between data packet following an ack, both the RTT and the location of the measurement point on the path have an influence: $IAT = (time_{data} - time_{ack}) - f \cdot RTT$, where $f \in [0, 1]$ is the "distance" of the measurement point from the TCP sender on the path and can be computed as $f = \frac{d_2}{d_1+d_2}$ from Figure C.1. In other words, we allow a maximum idle time of $\frac{RTT}{2}$ for the application. As the correct estimation of the RTT can be non-trivial (see [46]), the $estimate\_rtt()$ function in the *isolate* procedure tries four different techniques (including the techniques proposed in [116] and [103]) one after another until an estimate is obtained. We leave out further details. In the unlikely case that the RTT can not be estimated using any of the techniques, the connection is not processed at all.

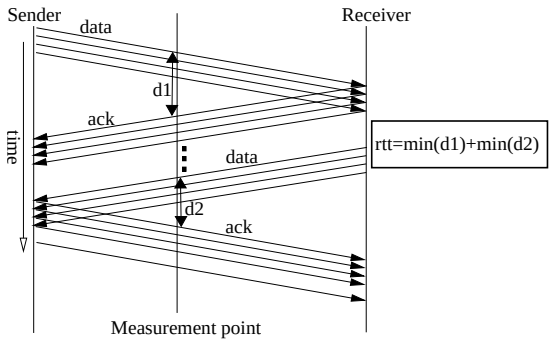



Figure C.1: Round-trip time estimation from a two-way data transfer.

## C.2 Merge

The *merge* procedure, sketched in Algorithm 3, inspects all periods identified by *isolate* and attempts to merge adjacent periods while respecting the given *drop* parameter value. *drop* indicates the maximum tolerated level of decrease in the throughput when merging periods to form a new longer BTP. $tput\_merged$ is the throughput of the newly formed period by merger, whereas $tput\_transfer$ is the throughput obtained when combining together only the BTPs and STPs contained by the merged period.

Note that the throughput of a transfer, especially a long one, is not always stable throughout its lifetime. Consider, for instance, a case where a BTP experiences congestion that degrades its throughput and the next BTP does not. Thus, in these cases a merger may be prevented due to different throughputs achieved by the subsequent transfer periods, which is, in fact, desireable in many cases since they exhibit clearly different network conditions.

**Algorithm 3**: The *merge* procedure.

**input argument** $drop \in [0, 1]$;
**define** struct $Period := \{bytes, pkts, duration, n, type\}$;
**function** $merge(P_1, P_2)$ : returns new $Period\{P_1.bytes + P_2.bytes,$
$P_1.pkts + P_2.pkts, P_1.duration + P_2.duration, null, null\}$;
**initialize** $\forall i$, $n_i := 1$, $Vol := 0$, $i_0 :=$ index of first STP or BTP;
**initialize** $\mathcal{S} :=$ all periods identified by *isolate* procedure, $\mathcal{S}_{new} := \varnothing$;
**initialize** $P_{merge} :=$ new $Period\{0, 0, 0, 1, null\}$;
**repeat**
  **forall** *periods* $P_i \in \mathcal{S}$ **do**
    **if** $P_i.type$ *is STP or BTP* **then**
      $tput\_transfer := \frac{\sum_{k=i_0}^{i} P_k.bytes}{\sum_{k=i_0}^{i} P_k.duration \times P_k.n}$, $P_k.type$ is STP or BTP;
      $tput\_merged := \frac{P_{merge}.bytes + P_i.bytes}{P_{merge}.duration + P_i.duration}$;
      **if** $\frac{tput\_merged}{tput\_transfer} \geq drop$ **then**
        `// merger allowed`
        $P_{merge} := merge(P_{merge}, P_i)$;
        $P_{merge}.n := \frac{tput\_merged}{tput\_transfer}$;
      **else**
        `// merger not allowed`
        **if** $P_{merge}.pkts \geq 130$ **then**
          $P_{merge}.type :=$ BTP;
        **else**
          $P_{merge}.type :=$ STP;
        $\mathcal{S}_{new} := \mathcal{S}_{new} \cup \{P_{merge}, P_{i-1}\}$ `// add also the previous ALP`
        $P_{merge} := P_i$, $i_0 := i$;
    **else if** $P_i$ *is not the first or last ALP* **then**
      $P_{merge} := merge(P_{merge}, P_i)$ `// merge with the interleaving ALP`
  $\mathcal{S} := \mathcal{S}_{new}$;
**until** *no more mergers*;
store $\mathcal{S}$ as the final period set with *drop*;

The outermost loop in Algorithm 3 is required to ensure that eventually all allowed mergers are executed. Consider this likely scenario: A STP is not allowed to merge with the next STP because the $n$ value after a merger would be just below *drop* but the latter STP is allowed to merge with the next BTP. However, on the second round the first STP is allowed to merge with the already merged BTP because the BTP weights much more than the STP in the computation of $tput\_transfer$. Consequently, care must be taken when updating the $tput\_transfer$ value to take into account the decrease due to mergers performed on previous rounds. Otherwise, the algorithm might loop until all periods are merged together. That is why the period structure in Algorithm 3 contains the field $n$ which stores the current reduction in the merged period and is updated after each succesful merger. Subsequently, *we need to divide by* $P_k.n$ *each time when updating* $tput\_transfer$. Finally, a merger is never started or ended with an ALP.

APPENDIX D

# Formal Definitions for Computed Metrics

## D.1 Inter-arrival Times of Acknowledgments

We denote the arrival time, i.e. the time of capture, and the size of $i^{th}$ data packet as $a_i$ and $s_i$, respectively. We consider only ACKs that acknowledge 1 MSS or 2 MSS worth of data and cancel the effect of delayed ACKs by dividing by two the inter-arrival time of ACKs that acknowledge two data packets. The time series $\mathcal{I}_t$ of inter-arrival times of ACK packets for a connection with $n$ packets is thus obtained as follows:
First, we define

$$T = \{t' = 1, 2, 3, ..., n-1 \mid s_{t'} \in \{MSS, 2 \cdot MSS\}\}$$

Then, we obtain

$$\mathcal{I}_t = \frac{a_{t+1} - a_t}{\frac{s_t}{MSS}} \ , t \in T \tag{D.1}$$

## D.2 Round-trip Time

We define $a_i$ as the arrival time of the $i^{th}$ packet, $seq_i$ and $ack_i$ as the data end sequence number and acknowledged sequence number of the $i^{th}$ packet, respectively. In the case that our measurement point is close to the sender, we obtain the time series $\mathcal{RTT}_t$ of RTT values as follows:
Consider the following notation

$$d_i = \begin{cases} 1 & \text{if } i^{th} \text{ packet was sent by receiver} \\ 0 & \text{if } i^{th} \text{ packet was sent by sender} \end{cases} \tag{D.2}$$

$$L = \{l | d-1| \in \mathbb{N}^* \mid \forall seq_k = seq_l : k < l\}$$

$$L' = \{l \cdot d \in \mathbb{N}^* \mid \forall ack_k = ack_l : k > l\}$$

$$T = \{t' \in L' \mid \forall t' \exists t \in L : ack_{t'} = seq_t\}$$

Finally, we obtain

$$\mathcal{RTT}_t = \{a_t - a_i \mid ack_t = seq_i\} \ , t \in T \tag{D.3}$$

In the case that our measurement point is not close to the sender, we use the methods described in [116] and [72].

## D.3 Receiver Advertised Window

We denote the length of the time window as $T$, and the arrival time of the $i^{th}$ packet as $a_i$. We obtain thus for a specified direction of connection with duration $D$ a time series $\mathcal{R}_t$ as follows:
Consider the following notation

$$a_{t,min} = \min_{i,d_i=1}\{a_i \mid a_i \in [T\cdot(t-1), T\cdot t]\}$$

$$a_{t,max} = \max_{i,d_i=1}\{a_i \mid a_i \in [T\cdot(t-1), T\cdot t]\}$$

$$w_{t,min} = \{w_i \mid a_i = a_{t,min}\}$$

$$w_{t,max} = \{w_i \mid a_i = a_{t,max}\}$$

$$J_t = \{j \in \mathbb{N} \mid a_j \in [T(t-1), Tt]\}\ , t = 1, 2, 3..., \frac{D}{T} \tag{D.4}$$

Using the above definitions we obtain

$$\mathcal{R}_t^1 = (a_{t,min} - T(t-1))w_{t-1,max}$$

$$\mathcal{R}_t^2 = (T\cdot t - a_{t,max})w_{t,max}$$

$$\mathcal{R}_t^3 = \sum_{i\in J_t}(a_i - a_{i-1})w_{i-1}d_{i-1}$$

Finally, we obtain

$$\mathcal{R}_t = \frac{\mathcal{R}_t^1 + \mathcal{R}_t^2 + \mathcal{R}_t^3}{T}\ , t = 1, 2, 3..., \frac{D}{T} \tag{D.5}$$

If the measurement point is away from the sender, we define the arrival time of the $i^{th}$ packet as

$$a_i' = (d_i - 0.5)\cdot d_6 \cdot a_i \tag{D.6}$$

, where $d_6$ is the delay shown in Figure D.1 when the measurement point is C, and we use $a_i'$ in the above calculations instead of $a_i$.

## D.4 Outstanding Bytes

If the measurement point is away from the sender, we do the computation with the help of TCP timestamps.

We denote again the length of the time window as $T$, the arrival time of the $i^{th}$ packet as $a_i$. Additionally, we denote the data end sequence number and acknowledged sequence number of the $i^{th}$ packet as $seq_i$ and $ack_i$, respectively. We obtain for a specified direction of connection with duration $D$ a time series of the outstanding bytes $\mathcal{O}_t$ as follows:
Using (D.2) we define

$$K = \{k|d_l - 1| \in \mathbb{N}^* \mid \forall l < k : seq_l \cdot d_l < seq_k \cdot d_k\}$$

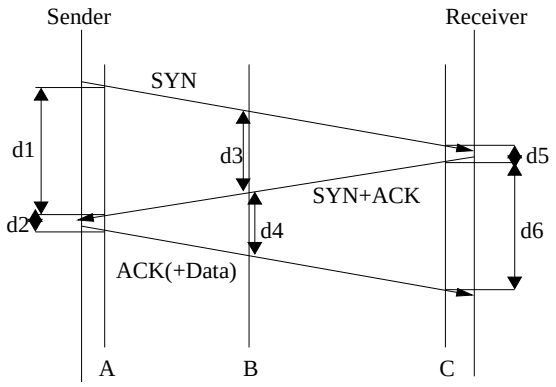



Figure D.1: Determining the measurement position from the three-way handshake of TCP. This figure appears with detailed explanations in Section 7.1.1

$$K' = \{k \cdot d \in \mathbb{N}^* \mid \forall l < k : ack_l \cdot |d_l - 1| < ack_k \cdot |d_k - 1|\}$$

Using the above definitions and (D.4) we obtain

$$\mathcal{O}_t^1 = \sum_{i \in J_t \cap K} seq_i(a_{i+1} - a_i)$$

$$\mathcal{O}_t^2 = \sum_{i \in J_t \cap K'} ack_i(a_{i+1} - a_i)$$

Finally, we obtain

$$\mathcal{O}_t = \frac{\mathcal{O}_t^1 - \mathcal{O}_t^2}{T} \ , t = 1, 2, 3..., \frac{D}{T} \tag{D.7}$$

If the measurement point is away from the sender, as with receiver advertised window, we use (D.6) in the above calculations instead of $a_i$.

## D.5 Retransmissions

We denote the IPID number of the $i^{th}$ packet as $id_i$. First, we define

$$seq_{i,max} = \max\{seq_j \mid j < i\}$$

$$id_{i,max} = \max\{id_j \mid j < i\}$$

Then, we obtain the set of indices corresponding to those packets that were retransmitted for a connection with $n$ packets (note that the two last conditions take into account the wrap around of the sequence and IPID numbers) as

$$\begin{aligned} \mathcal{R}x_t \ = \ & \{t \mid seq_t < seq_{t-1,max}, id_t > id_{t-1,max}, id_{t-1,max} - id_t < 5000, seq_{t-1,max} - seq_t < 10^6\} \\ & , t = 2, 3, 4, ..., n \end{aligned} \tag{D.8}$$

ANNEXE E

# Résumé de la Thèse

## E.1 Introduction

### E.1.1 Internet : une évolution continuelle

L'Internet, qui a débuté comme projet de recherche d'ARPA (Advanced Research Projects Agency) aux Etats-Unis en 1969, est devenu un réseau immense connectant des centaines de millions de machines aujourd'hui. Sa diversité est double. D'une part, les machines reliées à l'Internet comportent des PCs, des serveurs, des téléphones portables, des satellites, des PDAs, des capteurs etc. D'autre part, il y a énormément de services disponibles aujourd'hui, y compris la radio, la télévision, le téléphone, la vidéoconférence, la transmission de messages instantanée, et la distribution de contenu pair-à-pair et, bien sûr, les applications traditionnelles : la messagerie électronique (email) et le Web.

Beaucoup de questions au sujet du comportement et des caractéristiques de l'Internet restent ouvertes. Bien que nous connaissions bien les caractéristiques des différents modules de l'Internet, le système dans son entier est perçu comme une "boîte noire". Par exemple, nous aimerions savoir la taille exacte de l'Internet en termes de n ?uds reliés. Il est également non triviale de découvrir la structure, c.-à-d. la topologie, de l'Internet. Une autre interrogation concerne le choix des métriques quantitatives nécessaires pour répondre aux questions précédentes. Pourquoi sommes-nous dans une telle situation ? Il y a plusieurs raisons à cela. Comme les auteurs de [40] l'on remarqué, l'évolution de l'Internet n'a pas été un effort centralisé. Plusieurs parties y ont contribué avec des objectifs différents.

L'Internet est immense et il est constamment en mouvement. C'est pourquoi c'est un grand défi de le mesurer et de le caractériser. Sa diversité apparaît dans beaucoup de d'aspects. Premièrement, l'Internet évolue constamment. L'ensemble des services disponibles évolue et change tout le temps, le nombre d'utilisateurs et les volumes de trafic augmentent exponentiellement. D'une part, cette évolution rapide augmente la nécessite de mesurer l'Internet. Par exemple, dans [36] les auteurs analysent l'impact de l'apparition de nouvelles applications très populaires sur des caractéristiques du trafic. Ils concluent que l'impact est dramatique et que les implications sont très importantes sur le provisionnement de capacité du réseau. D'autre part, l'évolution évoque de nouveaux problèmes : la quantité de données à mesurer est souvent

immense, ce qui complique le processus d'analyse et pose des problèmes importants pour le stockage. Deuxièmement, un instantané représentatif de l'Internet n'existe pas. C'est à dire que la bonne métrique locale aujourd'hui ne sera pas nécessairement une bonne métrique demain. De manière similaire, la bonne métrique locale n'est peut-être jamais une bonne métrique dans un autre contexte. Par exemple, les applications et le comportement des utilisateurs peuvent être très différents d'un jour à un autre pour un réseau donné, et ils peuvent être complètement différents entre un réseau d'entreprise et un réseau d'accès d'ADSL.

### E.1.2 Analyse des causes du débit de transmission de TCP

Afin de comprendre pourquoi c'est intéressant de se concentrer sur l'analyse de TCP, nous devons revisiter le fonctionnement de l'Internet. Les machines sont reliés à l'Internet se communiquent avec une suite de protocole IP. La figure E.1 montre la structure de cette pile de protocoles. Chaque application donne des données à transférer à la couche inférieure, la couche transport. Cette couche est responsable du transport des données de bout en bout. Aujourd'hui, il existe deux protocoles principaux au niveau de la couche transport de l'Internet : Transmission Control Protocol (TCP) et User Datagram Protocol (UDP). La couche réseau, qui est située dessous de la couche transport, comporte seulement le protocole IP (Internet Protocol). Ce protocole est utilisé par TCP pour transmettre des morceaux de données de la source à la destination.

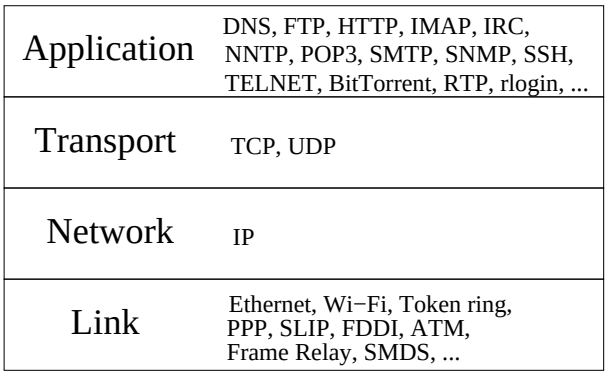



FIG. E.1 – The Internet protocol suite.

Sur la couche la plus élevée dans la figure E.1, il y a eu des changements importants pendant ces dernières années. L'ensemble des applications qui contribuent le plus au trafic Internet s'est transformé du Web et de FTP aux applications pair-à-pair. De plus, il y a des nouvelles applications telles que le RSS ou le PodCast qui apparaissent constamment. Cependant, TCP transporte toujours la majorité de données, typiquement plus de 90% des octets sur un lien. Ce fait ainsi que la croissance rapide des volumes de trafic accentuent la polyvalence de TCP mais ils posent également la question de la façon dont TCP et ces nouvelles applications interagissent dans ce nouvel environnement. Par conséquent, l'analyse du protocole TCP et du trafic TCP est encore bien plus essentielle qu'avant.

Le débit de transmission, c'est-à-dire la quantité de données transmises par périodes de temps, est normalement la métrique de performance la plus importante pour les applications. Considérons, par exemple, le téléchargement d'un fichier avec FTP. Plus le téléchargement est

rapide, plus l'utilisateur est content. Notre objectif dans cette thèse est l'analyse du débit de transmission de TCP et l'identification des raisons qui empêchent une connexion TCP d'obtenir un débit plus élevé. Bien qu'une telle analyse puisse sembler triviale à première vue, nous montrerons dans cette thèse que c'est loin d'être le cas. La variété des conséquences de ces différentes limitations et les contraintes imposées par le contexte des mesures compliquent l'analyse. On ne peut pas mesurer directement la plupart de métriques qui sont exigées pour cette analyse ; elles doivent être estimées en conséquence.

La connaissance des causes du débit de transmission de TCP implique la connaissance des causes du débit de la grande majorité du trafic de l'Internet. C'est pour cela que cette connaissance est très importante et utilisable de manières très diverses. Par exemple, elle pourrait être employée par les ISPs (Internet Service Provider) pour dépanner leur réseau d'accès ou analyser les performances de leurs clients. Cette connaissance pourrait également permettre l'évaluation de la performance opérationnelle d'une application déployée.

### E.1.3 Contributions de la thèse

Cette thèse comporte quatre contributions principales :

I. *Nous pouvons résoudre les problèmes de la gestion et du cycle de processus d'analyse sous optimal dans l'analyse de trafic de traces de paquets avec une approche basée sur un système de gestion de base de données (DBMS).*
   (a) *Les performances de l'implémentation d'une telle approche sont raisonnables.*
   (b) *Nous pouvons améliorer de manière significative les performances d'un tel système en optimisant les entrées-sorties (I/O) à partir des caractéristiques du trafic et de l'évaluation de la popularité des requêtes SQL.*

II. *Il est possible d'identifier les causes de limitation du débit d'une connexion TCP à partir des traces de paquet bidirectionnelles enregistrées de manière passive dans un seul point de mesure situé n'importe où sur le chemin de TCP/IP. De plus, les traces unidirectionnelles sont insuffisantes.*

III. *Les différentes applications se comportent de manière complexe dans leur interaction avec TCP. C'est pourquoi les effets dus aux applications doivent être d'abord filtrés quand on fait les études sur les caractéristiques du chemin de TCP/IP.*

IV. *Nos méthodes pour identifier les causes du débit de transmission de TCP combinées avec notre approche basée sur un système de gestion de bases de données pour l'analyse de trafic permettent de :*
   – *évaluer les performances des protocoles au niveau applicatif,*
   – *évaluer l'utilisation et la charge du réseau,*
   – *identifier certains problèmes de configuration de TCP.*

## E.2 Résumé des Trois Parties de la Thèse

### E.2.1 Première Partie : Méthodologie

Dans la première partie de la thèse, nous motivons, décrivons, et justifions notre méthodologie pour conduire une analyse de trafic Internet.

#### E.2.1.1 Métrologie de l'Internet

Le domaine de recherche de la métrologie de l'Internet se caractérise par des techniques de mesure diverses et variées. Il est cependant possible d'identifier deux principales catégories : les mesures actives ou passives. Les premières consistent généralement à envoyer des paquets vers des sondes dédiées pour déterminer des caractéristiques concernant le chemin que les paquets suivent. Ping et traceroute sont des exemples bien connus de ces outils de mesures actives. D'autres techniques, passives cette fois, sont utilisées pour obtenir des données a posteriori, l'analyse des caractéristiques de réseau étant effectuée après avoir enregistré le trafic collecté depuis une machine donnée ou un routeur.

Cette thèse se focalise sur l'identification des causes de limitation du débit du trafic TCP observé sur un seul point de mesure sur le chemin de TCP/IP. Notre objectif est de parvenir à identifier et analyser ces causes sur la totalité du trafic (potentiellement volumineux) afin d'acquérir une meilleure connaissance du comportement courant de trafic Internet. Nous n'aborderons dans cette thèse que les mesures passives et nous laissons celles actives pour de futurs travaux. Enfin, nous utilisons dans notre analyse des traces du trafic se limitant aux seules en-têtes des paquets TCP/IP.

Il y a généralement trois problèmes dans l'analyse hors-ligne des mesures passives : problèmes de gestion ou d'échelle, et cycle de processus d'analyse qui est sous optimal. Ces problèmes sont principalement dus à trois facteurs : l'état de l'art des outils d'analyse montre que les scripts restent trop spécifiques ou, au contraire, pour les nombreux logiciels spécialisés, l'analyse de trafic est typiquement un processus itératif, et la quantité de données à analyser reste généralement grande. Nous soutenons qu'une solution basée sur un système de gestion de bases de données peut largement résoudre ces difficultés.

#### E.2.1.2 InTraBase : Analyse de trafic intégrée et basée sur un système de gestion de base de données relationnelle orientée objet

Notre méthodologie pour l'analyse des mesures passives du trafic s'appelle InTraBase. Elle est conçue pour un usage intensif et une analyse hors-ligne de traces de trafic au niveau paquet et de taille moyen ($< 50$GB).

InTraBase est une solution entièrement intégrée et basée sur un système de gestion de base de données relationnelle orientée objet. Figure E.2 présente une vue architecturale du système. Nous stockons des données de différentes sources dans le système de base de données (DBS). On appelle les données téléchargées dans le DBS les *données de la base*. Par exemple, des traces de paquets capturés avec `tcpdump` sont une forme de données de la base.

Une fois que les données de base sont téléchargées dans le DBS, nous les traitons afin de dériver les nouvelles données qui sont également stockées dans le DBS. Tout le traitement est fait dans le DBS en utilisant les fonctions et les requêtes SQL du DBS (voir la figure E.2). Le système de gestion de base de données *relationnelle orientée objet* permet de faire évoluer la

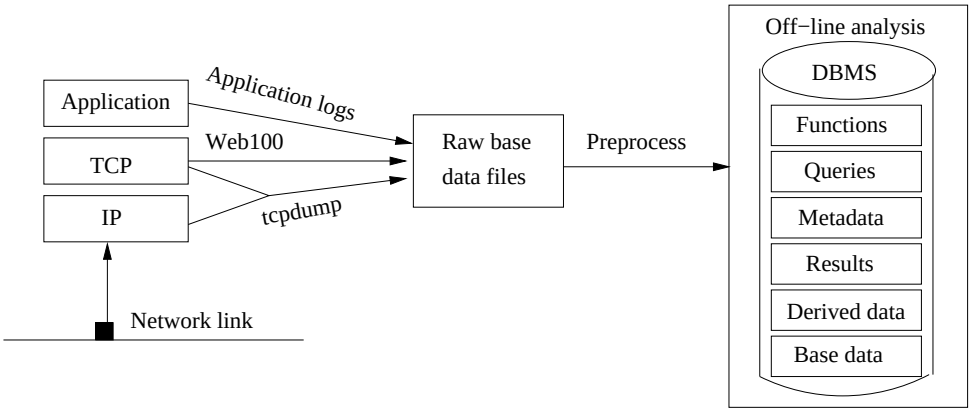



Fig. E.2 – Architecture du système de base de données (DBS).

fonctionnalité en ajoutant de nouvelles fonctions, ce qui n'est pas possible avec un système de gestion de base de données relationnelle standard comme MySQL.

Une approche basée sur un DBMS permet d'organiser et de gérer des données, les meta-données, des résultats d'analyse, et des outils. Le principal avantage tient au fait que, dans une base, les données sont stockées de manière structurée et suivant une sémantique. Cela implique que le traitement des données est facilité car les outils parviennent à mieux manipuler et interpréter les données. De plus, il est plus efficace de chercher des données particulières avec des index, et il est toujours envisageable de stocker des résultats intermédiaires réutilisables. Enfin, cela permet de combiner différentes sources de données (par exemple à travers différentes couches de piles protocolaires).

Le prototype de l'InTraBase, pgInTraBase, fournit la base pour l'analyse des traces de paquets TCP capturés avec tcpdump ou une carte d'Endaces DAG. PgInTraBase est développée avec PostgreSQL [10] qui est un logiciel libre de gestion de base de données qui dispose d'une grande communauté d'utilisateurs. C'est le nature 'relationnelle orientée objet' de PostgreSQL qui permet de faire évoluer la fonctionnalité par l'ajout de fonctions dans les langues procédurales chargeables (PL).

Les tables de base utilisées dans PgInTraBase sont décrites dans la figure E.3. Il existe cinq étapes distinctes pour traiter une trace de paquets :

I. Stocker l'annotation au sujet de la trace dans la table *traces*.

II. Créer la table dédiée aux paquets.

III. Charger les en-têtes des paquets dans le table de paquets.

IV. Calculer les statistiques des connexions à partir de la table de paquets puis les stocker dans la table de connexions.

V. Insérez l'information liant le 4-tuple au *cnxid* dans la table *cid2tuple*.

#### E.2.1.3 Évaluation et optimisation de l'InTraBase

Nous avons évalué la performance et la consommation en terme d'espace disque du prototype. Comme il est possible de voir dans la figure E.4, le temps d'exécution des étapes de base évolue

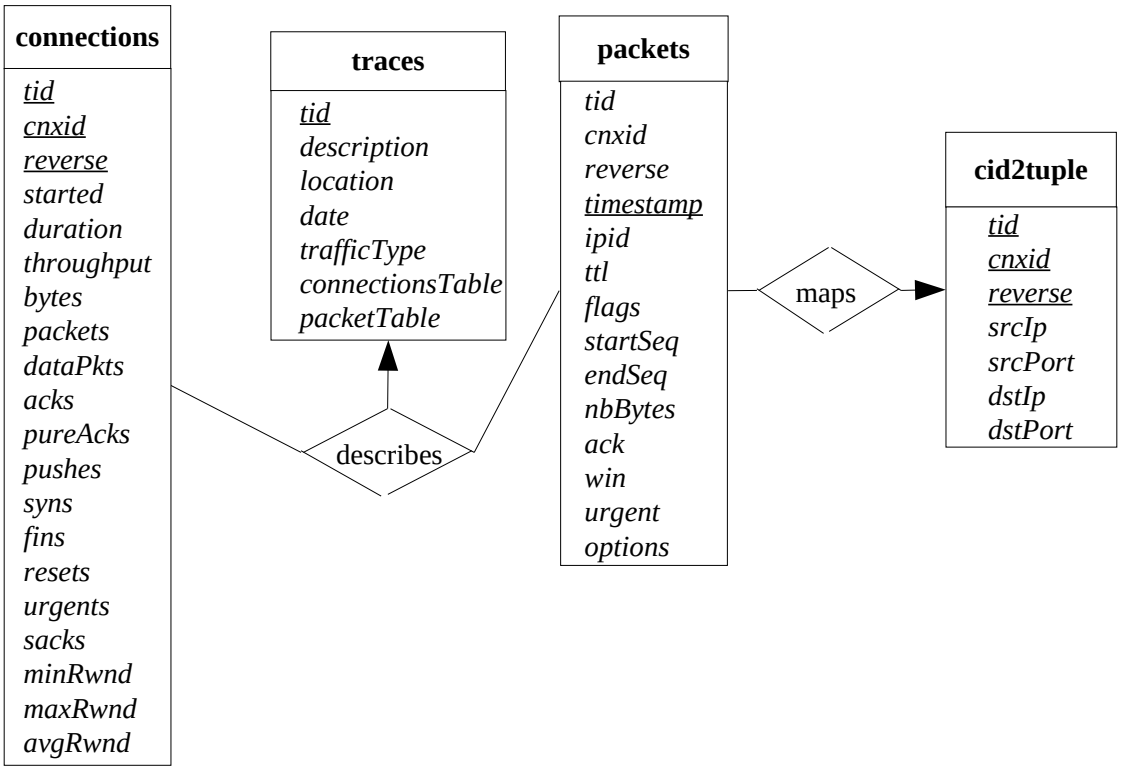



FIG. E.3 – Les tables de base utilisées dans PgInTraBase. Les paramètres sous-lignés forment un clef unique pour chaque ligne de la table.

linéairement jusqu'à 10GB. Il faut environ quatre heures pour traiter une trace de 10GB, ce qui est assez rapide pour nos besoins. Il est important de rappeler que l'InTraBase n'est pas conçu pour une analyse en ligne (active). Quant à la consommation d'espace disque, elle se caractérise par un surplus mesuré d'environ 50%. Un tel surplus ne pose normalement pas de problèmes. L'espace supplémentaire est le prix à payer pour avoir de données structurées.

L'optimisation de la performance d'un système tel que PgInTraBase est très importante car un système de gestion de base de données typique qui s'achète aujourd'hui n'est pas optimisé pour la gestion des données scientifiques. C'est pourquoi la configuration de défaut ne fournit généralement pas une bonne performance. Cette dernière dépend également des caractéristiques des données. Dans l'analyse des mesures du trafic au niveau paquet il y a souvent quelques requêtes spécifiques qui sont exécutées très fréquemment. Par voie de conséquence, le système doit être optimisé pour ces requêtes.

Nous avons exécuté des optimisations d'entrée/sorie (I/O) pour la requête qui est exécutée le plus fréquemment dans PgInTraBase (on l'appelle c-requête). Nous appliquons et détaillons une méthode d"indexation et de regroupement (appelée clustering par la suite). Ce sont des techniques standards du système de gestion de bases de données afin d'améliorer la performance d'I/O. Les index permettent la consultation rapide des lignes spécifiques des tables. En revanche,

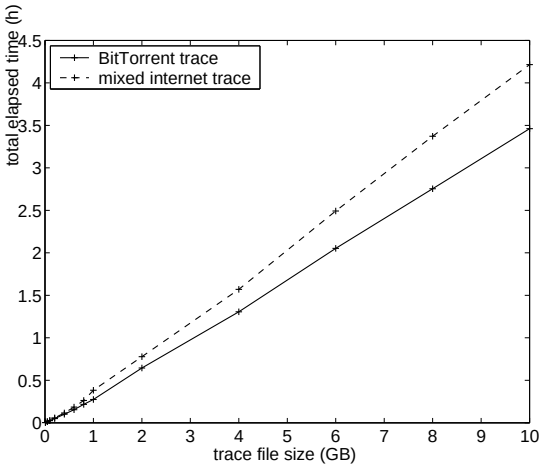



Fig. E.4 – Le temps total pour traiter une trace de `tcpdump` d'une taille variée.

le clustering regroupe physiquement le disque dur des données qui sont sémantiquement reliées (par exemple tous les paquets d'une connexion) pour un accès plus rapide. Nos résultats de mesure attestent que ces optimisations peuvent diminuer le temps d'exécution de la c-requête pour toutes les connexions d'une trace de 5 GO de 8 jours à 27 minutes dans le meilleur des cas.

### E.2.2 Partie 2 : Analyse des Causes du Débit de Transmission TCP

La partie 2 est dédiée aux techniques et aux algorithmes pour l'analyse des causes du débit de transmission TCP, ce qui constitue les contributions principales de cette thèse.

#### E.2.2.1 Causes de limitation des transferts de TCP

**Application :**
Dans cette situation particulière, l'application n'essaie même pas d'utiliser toutes les ressources de réseau disponibles. La figure E.5 décrit les flux de données de l'application émettrice (E) à l'application qui réceptionne (R) par une simple connexion TCP. L'interaction se produit par des buffers : du point de vue de (E) l'application stocke des données à transmettre par TCP dans le buffer `b1`, alors que du le côté de (R), l'application stocke les données TCP qui sont correctement reçues en séquence dans le buffer `b2`, et lues par (R). Les données qui sont reçues hors séquence sont maintenues dans le buffer `b4` jusqu'au moment où elles pourront être fournies dans l'ordre et être stockées dans le buffer `b2`.

Les connexions TCP se divisent suivant deux types de périodes : Les périodes de transfert (BTP), pendant lesquelles l'application fournit constamment des données à transmettre a TCP, i.e. b1 n'est jamais vide, et les périodes limitées par l'application (ALP), où TCP doit attendre des données parce que b1 est vide.

**Le récepteur d'une connexion TCP :**
Quand la limitation provient du récepteur d'une connexion TCP, c'est la fenêtre annoncée par le récepteur TCP qui limite le débit de transmission. Ceci peut se produire pour deux raisons : premièrement, l'application (R) est sur le point de déborder l'application de réception (le buffer

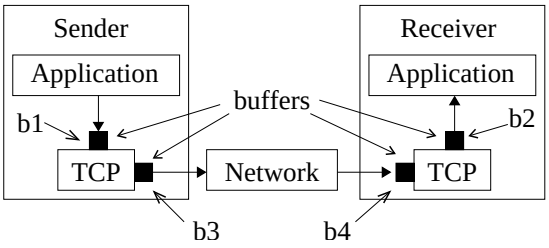



FIG. E.5 – Flux de données de l'application émettrice à l'application qui réceptionne par une simple connexion TCP.

b2 est plein). Dans ce cas, la fenêtre du récepteur est utilisée pour le contrôle de flux, comme prévu à l'origine. Deuxièmement, il est possible que, involontairement, la taille par défaut de cette fenêtre soit configurée pour avoir une valeur qui s'avère trop petite pour le logiciel d'exploitation, voire même que le "window scaling" n'est pas supporté ou n'est pas autorisé pour une quelconque raison.

**Réseau :**
Cette limitation correspond au cas où un lien limitant (bottleneck) perturbe le débit de transmission. Nous distinguons deux types de limitation de réseau. Le premier est un lien bottleneck non partagé qui correspond au cas où une connexion simple utilise toute la capacité du lien bottleneck. L'autre type et un lien bottleneck partagé qui se produit quand plusieurs connexions profitent de la capacité du lien bottleneck.

**TCP protocol :**
La cause de débit de transmission peut également être due aux algorithmes de TCP qui s'appellent congestion avoidance et slow start. Dans ce cas, le transfert finit avant que le débit augmente au point d'être limité par le réseau ou la réception TCP.

#### E.2.2.2 Identification des causes de limitation

Nos techniques sont basées sur l'inspection à fine échelle des traces bidirectionnelles de paquets capturés en un seul point de mesure. Nous analysons chaque connexion de TCP séparément. Pour appliquer nos algorithmes, nous avons besoin des connexions de TCP qui portent au moins 130 paquets de données, car nous décidons de négliger les connexions trop courtes qui peuvent être influencées par l'algorithme slow start.

Notre approche consiste à diviser puis conquérir : Nous classifions dans une première étape les paquets d'une connexion dans BTPs et ALPs. Dans un second temps, nous analysons le BTPs pour déterminer s'ils sont limités par le récepteur de la connexion TCP, par le réseau, ou par le protocole TCP.

**La première étape : Séparation de BTPs des ALPs :**
Nous appelons notre algorithme servant à identifier les BTPs et ALPs l'algorithme Isolate and Merge (IM) dû à la manière dont il procède. L'algorithme est générique parce qu'il peut être appliqué sans calibration, au trafic de n'importe quelle application. De plus, il ne dépend pas de la version de TCP utilisée. L'algorithme IM se compose de deux phases : tout d'abord, la phase d'isolation (Isolate) divise la connexion dans BTPs séparé par ALPs. Ensuite, la seconde phase, Merge, essaie de fusionner deux BTPs consécutifs comprenant les ALPs qui séparent ces

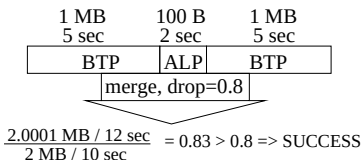



Fig. E.6 – Fusion réussie.

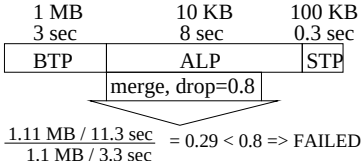



Fig. E.7 – Fusion échouée.

deux BTPs afin de créer un nouveau BTP. L'étape de fusionnement est nécessaire parce qu'une connexion peut être divisée en nombreux BTPs séparés par des ALPs très courts après la phase d'isolation. Il serait même souhaitable de combiner ces périodes dans un long BTP si l'effet de ces ALPs courts sur le débit global est négligeable. Les fusions sont contrôlées avec le paramètre de seuil $drop \in [0, 1]$. Les figures E.6 et E.7 démontrent des fusions réussies ou échouées.

Nous avons appliqué l'algorithme IM aux études du biais du aux ALPs sur des analyses pour des chemins TCP/IP. Nous avons étudié les caractéristiques du débit et l'estimation du RTT des connexions TCP générées par des applications différentes. Nous avons enfin prouvé que si les effets de l'application ne sont pas filtrés au préalable, les études du chemin de TCP/IP et les caractéristiques du trafic d'un point de vue de réseau peuvent produire des résultats biaisés.

**La seconde étape : Analyse des BTPs :**
Notre approche pour analyser les BTPs comporte deux phases. Dans la première phase, nous calculons pour chaque BTP plusieurs points de limitation. Ces points sont une métrique quantitative pour évaluer le niveau d'une cause de limitation pour un BTP. Nous calculons tous ces points à partir de l'information qui se trouve dans la trace des paquets. Il peut y avoir plusieurs causes de limitations à un instant donné. Notre approche consiste donc à déterminer celle dominante, en s'appuyant sur un schéma de classification basé sur différents points de limitation dans la deuxieme phase.

Les points de limitation calculés dans la première phase sont :

- points de limitation de fenêtre de récepteur : la fraction du temps où le nombre des octets transmis sont limités par la fenêtre annoncée par le récepteur
- points de retransmission : la fraction des octets retransmis
- points de dispersion : ces points indiquent si le débit est proche de la capacité du chemin ou pas
- b-points : ces points mesurent la sporadicité du transfert (voir les figures E.8 et E.9 pour des exemples de b-points hauts et bas)

Apres avoir calculé l'ensemble des points de limitation, nous appliquons notre méthode basée sur des seuils pour classifier les BTPs : Il y a un seuil attaché à chaque type de points. Le schéma de classification est décrit dans la figure E.10.

Les seuils doivent être calibrés. Cela nécessite des données de référence. C'est une tâche difficile de produire de bonnes données de référence car, pour vérifier leur qualité, il faudrait s'appuyer sur une supposition que cette thèse essaie de valider. Le problème est de trouver une méthode qui produit des données avec assez de diversité et, dans le même temps, qui comporte un niveau de contrôle assez élevé pour les expérimentations. D'une part, les simulations ne sont pas représentatives et restent assez diverses. D'autre part, les mesures « en aveugle » donnent les données avec des caractéristiques inconnues qui ne sont guère exploitables. Notre approche est de produire des données de référence avec des expériences légèrement contrôlées afin de produire

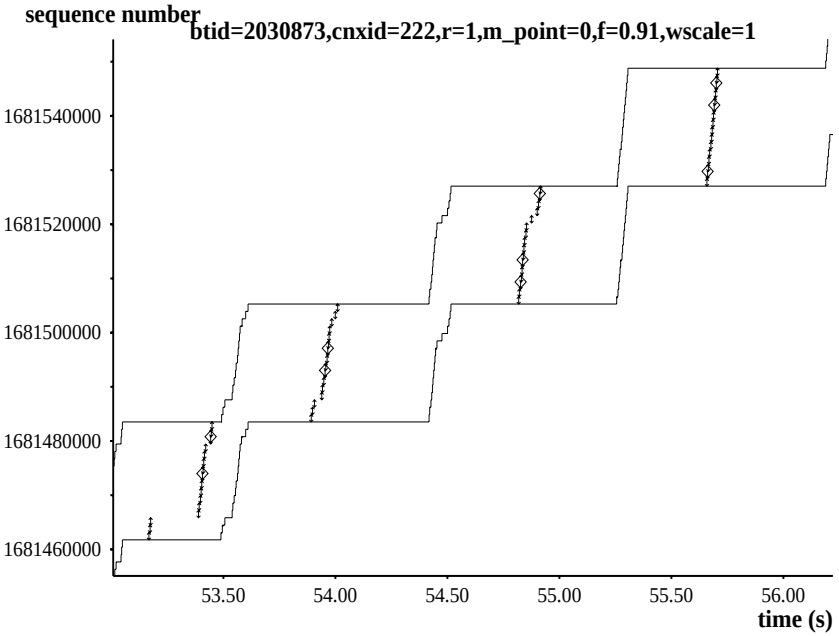



FIG. E.8 – Le numéro de séquence en fonction du temps pour un transfert limite par la fenêtre de récepteur avec les b-points hauts.

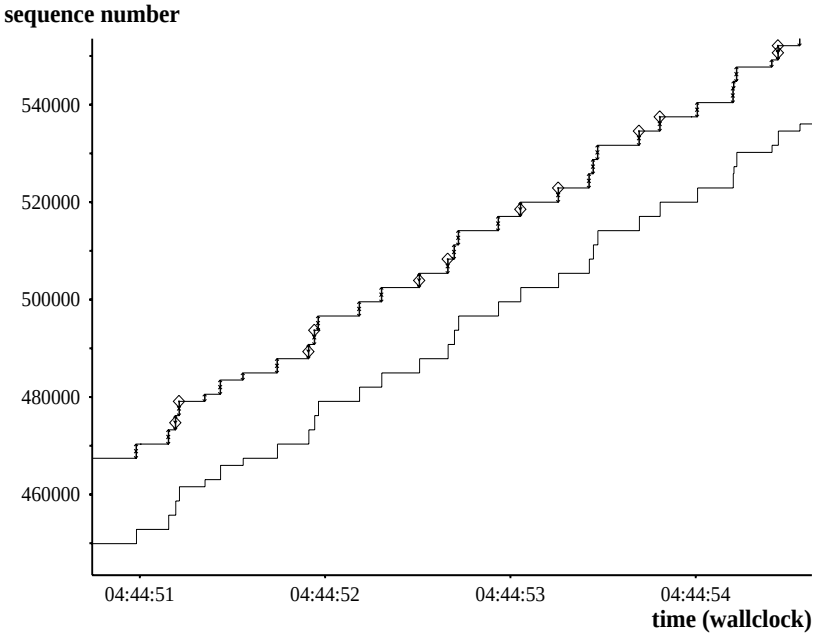



FIG. E.9 – Le numéro de séquence en fonction du temps pour un transfert limite par la fenêtre de récepteur avec les b-points bas.

des transferts limités par certaines causes avec une probabilité élevée.

Nous avons généré des téléchargements de FTP de 232 sites de miroir de système d'exploita-

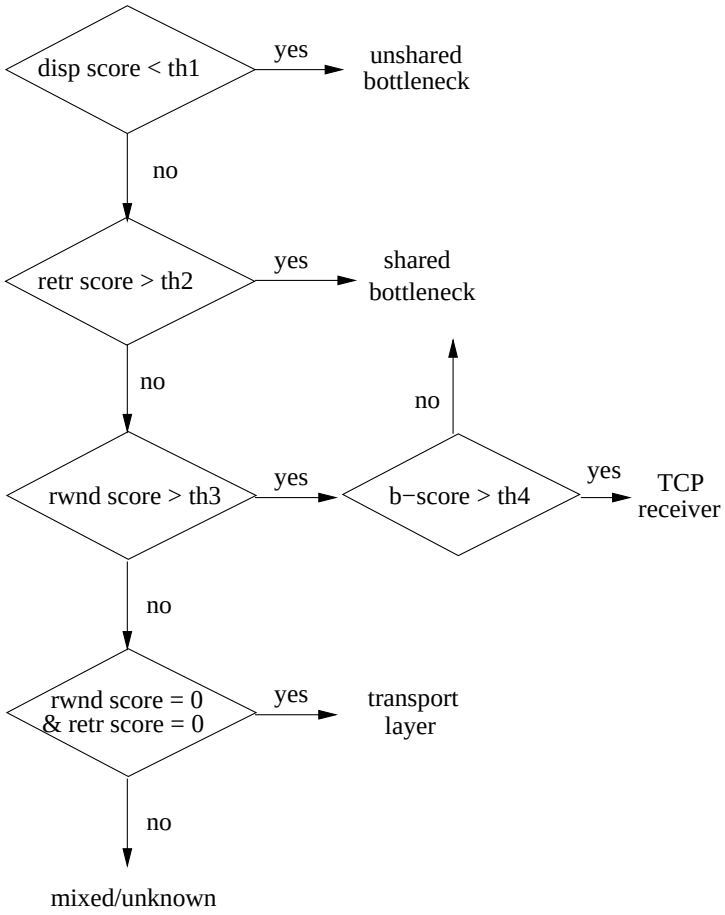



Fig. E.10 – Le schéma de classification.

tion Fedora Core qui ont couvert tous les continents. Pendant les téléchargements, nous avons mis en place des liens bottleneck artificiels avec rshaper [12] pour produire des transferts limités par le réseau. On a également utilisé NISTNet [5] pour retarder des paquets afin d'augmenter le RTT et créer la limitation par récepteur de TCP. Nous avons aussi contrôlé le nombre de téléchargements simultanés pour produire les deux limitations, par un lien bottleneck partagé et non partagé. Nous avons ensuite analysé les données de référence pour découvrir les seuils appropriés pour le schéma de classification.

### E.2.3 Partie 3 : Etude de Cas sur l'Analyse du Trafic d'un Réseau d'Accès d'ADSL

La partie 3 est une étude de cas concernant l'analyse du trafic d'un réseau d'accès d'ADSL de France Telecom. Le but est de montrer la capacité et l'utilité de l'InTraBase équipée avec nos techniques d'analyse de débit de transmission de TCP.

### E.2.3.1 Adaptation de l'InTraBase pour l'analyse des causes de débit de transmission de TCP

Nous utilisons le prototype PgInTraBase et étendons la conception pour incorporer les tables nécessaires pour l'analyse des causes de débit. La figure E.11 montre quelques tables. Les tables *bulk_transfer* et *app_period* stockent le BTPs et les ALPs identifiés par l'algorithme IM. Les points de limitations sont stockés dans les trois dernières tables, *bnbw_test*, *retr_test*, et *rwnd_test*. Tous nos algorithmes sont mis en oeuvre de telle sorte que les fonctions PL et ces fonctions peuplent ces trois tables.

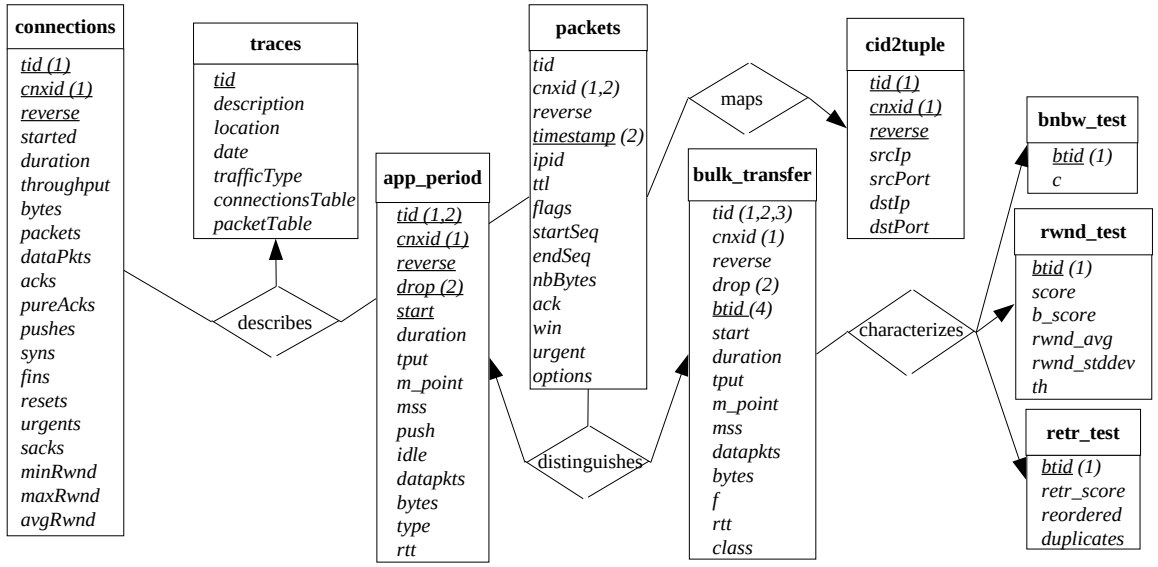



FIG. E.11 – La conception du prototype PgInTraBase avec les tables nécessaires pour l'analyse des causes de débit.

### E.2.3.2 Étude de cas sur des limitations de performance des clients d'ADSL

Pour un ISP, la satisfaction des clients est très importante. Si un client perçoit un débit de téléchargement inférieur à ce qui a été indiqué dans son abonnement, il risque d'être mécontent. Il est donc important que l'ISP comprenne la source de ce problème.

Nous avons analysé une grande trace de paquets des clients reliés à l'Internet par l'accès ADSL dans le but d'étudier les causes de limitation de débit observées par les clients. Nous avons capturé tout le trafic sur une journée entière (vendredi mars 10, 2006) généré par environ 3000 utilisateurs ADSL. Les données capturées en ce jour représentent environ 290 GO du trafic de TCP au total, dont 64% est descendant et 36% montant. Ce jour peut être considéré comme un jour normal en termes de volumes téléchargés par des clients. Seulement 1335 de ces 3000 clients ont produit suffisamment de données pour permettre d'appliquer nos techniques d'analyse de causes de débit de transmission. Nous considérons ces clients uniquement dans l'analyse qui suit.

**Ensemble de données :**
Nous regardons d'abord les caractéristiques générales de l'ensemble de données. La figure E.12 montre les volumes de trafic par application observées pendant toute la journée. Nous observons que le volume de données moyen montant est relativement constant pendant toute la journée ; ce qui n'est pas le cas pour le volume de données téléchargées. Seulement 5 applications ont produit plus de 5% du montant total des octets : eDonkey, les applications utilisant les ports 80/8080, BitTorrent, les échanges de messagerie électronique (email y compris SMTP, POP, et IMAP) et telnet. Nous avons identifié les applications par les numéros de port de TCP. De cette façon, la plus grande catégorie du trafic est l'autre catégorie avec plus de 50% du trafic total. Par ailleurs, nous ne voulons pas déclarer le trafic du port 80/8080 comme Web car il peut contenir du trafic pair-à-pair.

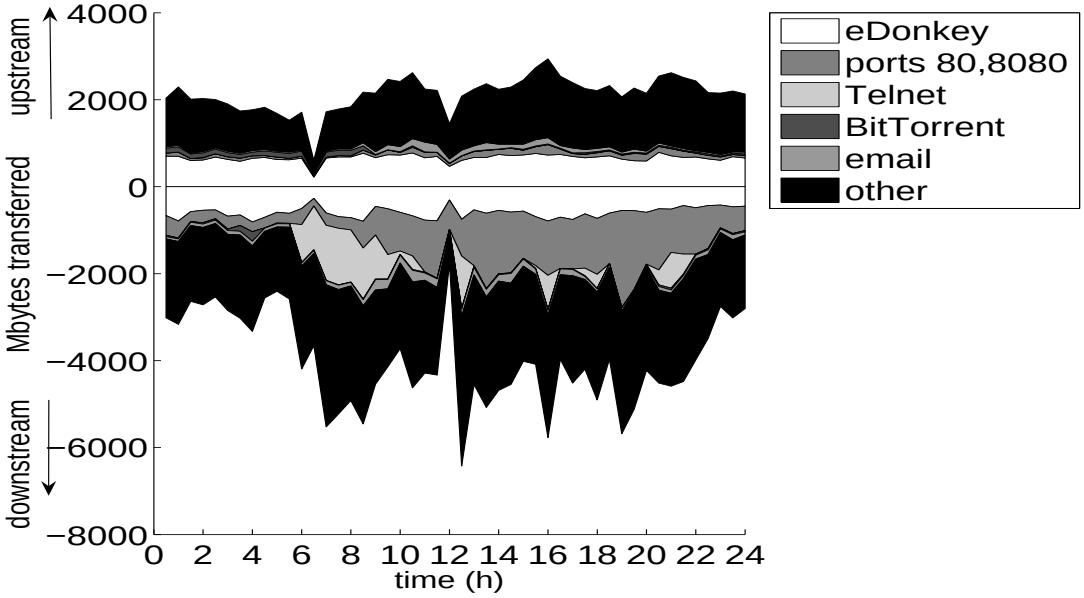



Fig. E.12 – Les volumes de données transmis par différents applications pendant la journée.

La distribution des tailles de connexions est fortement biaisée par beaucoup de petites connexions et de quelques grandes connexions, ce qui reste conforme aux résultats publiés depuis une dizaine d'années [41]. Par conséquent, nous utilisons dans notre analyse seulement 1% de connexions qui représente plus de 85% de tout le trafic en terme de volumes.

La distribution des clients – nous les identifions par l'adresse IP – en terme de données transmises est également biaisée. 15% de clients produisent 85-90% du trafic, montant et descendant. Nous appelons ces clients heavy-hitters. L'utilisation de lien d'accès est étonnamment basse pour tous les clients. Nous observons que 80% de clients utilisent leur lien descendant à 20% du maximum et le lien montant moins que 40% du maximum pendant une période de 30 minutes.

**Analyse de performance des clients :**
Après avoir constaté que la plupart des clients utilisent peu leurs liens d'accès, nous en étudions les causes. D'abord, nous devons définir une taxonomie des causes de limitation au niveau du client qui est basée sur la taxonomie au niveau connexion. Les causes qui peuvent limiter la

performance des clients d'ADSL sont :

- l'application elle-même
- un lien d'accès saturé
- une limitation de réseau due à un lien bottleneck éloigné
- la configuration de TCP

Pour identifier ces causes avec nos techniques de l'analyse des causes du débit de transmission des connections TCP, nous établissons les relations qui existent entre les causes au niveau d'une connexion et les causes au niveau du client. Plus précisément, nous associons des octets transmis pendant des ALPs et BTPs aux causes de limitation au niveau client de la façon suivante : tous les octets transmis pendant les ALPs de toutes les connexions du client sont associés à la limitation de l'application. Tous les octets transmis pendant les BTPs qui sont marqués comme "limité par le réseau" (lien bottleneck partagé ou non partagé) et une utilisation du lien d'accès, au cours de la connexion, au-dessus de 90% de maximum sont associés à une saturation de lien. Tous les octets transmis pendant le reste de BTPs limité par le réseau avec une utilisation au cours de la connexion en-dessous de 90% sont associés à la limitation de réseau due à un lien bottleneck éloigné. Tous les octets transmis par les BTPs descendant et limité par le récepteur de TCP sont associés à la limitation par configuration de TCP. Les octets transférés qui ne correspondent à aucun des critères précédents sont associés à une catégorie 'indéterminé'.

Les résultats de l'analyse indiquent que la majeure partie du trafic côté utilisateur est en fait limitée par application. Plus exactement, ce trafic limité par l'application est la plupart du temps est impacté par les applications pair-à-pair. Ces faits impliquent que les utilisateurs des applications de pair-à-pair imposent les limites de débit de transmission pour les téléchargements montant qui sont très conservatifs. Les autres causes de limitation de débit, comme la limitation par le réseau et par le récepteur TCP, qui ont été observées dans des études précédentes [118], ne ressortent pas de notre analyse. En limitant sévèrement le taux global de téléchargement de leurs applications de pair-à-pair, les clients parviendraient à assurer que leur trafic de pair-à-pair n'interfère pas des activités concourantes telles que le Web ou la téléphonie d'IP. Cependant, ceci implique des durées de téléchargement plus importantes, ce qui rend les stratégies de limitation de débit couramment utilisées par des clients de pair-à-pair inefficaces du point de vue d'un utilisateur. L'implication d'une telle basse utilisation de lien d'accès par les clients est l'utilisation basse du réseau d'accès entier, qui est profitable pour le fournisseur de service. Cependant, l'utilisation et les volumes de trafic peuvent nettement changer si un nouveau type d'application populaire de pair-à-pair est déployé ou une application déjà existante est améliorée pour utiliser le lien d'accès montant d'une manière plus efficace.

## E.3 Conclusions

## E.4 Contributions

Cette thèse comporte quatre contributions principales que nous avons présentées dans l'introduction. Maintenant, nous revisitons et évaluons chacune d'entre elles. Les contributions de cette thèse sont :

I. *Nous pouvons résoudre les problèmes de la gestion et du cycle de processus d'analyse sous optimal dans l'analyse de trafic de traces de paquets avec une approche basée sur un système de gestion de base de données (DBMS).*
   (a) *Les performances de l'implémentation d'une telle approche sont raisonnables.*
   (b) *Nous pouvons améliorer de manière significative les performances d'un tel système en optimisant les entrées-sorties (I/O) à partir des caractéristiques du trafic et de l'évaluation de la popularité des requêtes SQL.*

   Nous avons d'abord expliqué les problèmes de l'analyse de trafic passive et nous avons proposé une approche basée sur un système de gestion de bases de données qui peut potentiellement aider à les résoudre. Ensuite nous avons décrit notre approche basée sur un système de gestion de bases de données objet relationnelle et nous avons expliqué comment cette approche aide à résoudre ces problèmes. De plus, nous avons implémenté un prototype d'une telle approche basée sur PostgreSQL. Nous avons démontré avec des mesures de performance qu'un tel système est faisable et utilisable. Nous avons également identifié le goulot d'étranglement des performances de notre prototype, et nous avons montré comment améliorer considérablement les performances par des optimisations au niveau I/O.

II. *Il est possible d'identifier les causes de limitation du débit d'une connexion TCP à partir des traces de paquet bidirectionnelles enregistrées de manière passive dans un seul point de mesure situé n'importe où sur le chemin de TCP/IP. De plus, les traces unidirectionnelles sont insuffisantes.*

   D'abord nous avons présenté les causes de limitation de débit de transmission potentielles d'une connexion de TCP. Nous avons également montré sur des exemples comment ces causes se manifestent dans le trafic, par exemple en tant que modèles spécifiques des temps d'inter-arrivées. Ensuite nous avons présenté nos techniques pour identifier ces causes de limitation. Les techniques basées sur un ensemble de métriques sont calculées à partir des traces de paquets capturées avec un seul point d'observation. Nos métriques sont des mesures quantitatives et donc elles exigent de calibrer notre schéma de classification afin de produire des résultats qualitatifs (par exemple la cause dominant de débit d'un transfert donné de TCP est un goulot d'étranglement non partagé sur le chemin de réseau). Nous avons validé nos techniques et nous les avons comparé avec le seul autre outil actuellement existant, T-RAT [118]. Cette comparaison a démontré que nos techniques fonctionnent correctement dans plusieurs cas qui sont trop complexes pour T-RAT.

III. *Les différentes applications se comportent de manière complexe dans leur interaction avec TCP. C'est pourquoi les effets dus aux applications doivent être d'abord filtrés quand on fait les études sur les caractéristiques du chemin de TCP/IP.*

   Nous avons énuméré les différents types d'applications basées sur leur manière de communiquer avec TCP. Nous avons décrit notre algorithme IM pour diviser une connexion

TCP en deux types de périodes : Périodes de transfert (BTP) et périodes limitées par application (ALP). Les BTPs capturent seulement les propriétés du chemin bout à bout de réseau, alors que les ALPs peuvent capturer les propriétés de l'application. Nous avons montré sur plusieurs exemples, en utilisant des traces du trafic de différentes applications, comment la présence de l'application peut biaiser les mesures du débit et du délai.

IV. *Nos méthodes pour identifier les causes du débit de transmission de TCP combinées avec notre approche basée sur un système de gestion de bases de données pour l'analyse de trafic permettent de :*
   - *évaluer les performances des protocoles au niveau applicatif,*
   - *évaluer l'utilisation et la charge du réseau,*
   - *identifier certains problèmes de configuration de TCP.*

Nous avons présenté une étude de cas sur l'analyse de performance des clients d'un réseau d'accès commercial ADSL. Nous avons utilisé notre prototype, InTraBase, adapté à l'analyse des causes de débit de transmission de TCP et nous avons appliqué ce système à une trace de trafic d'un jour entier capturée au niveau d'un réseau d'accès de l'ADSL de France Telecom. Nous avons défini un ensemble de causes de limitation de performances au niveau client et nous avons appliqué nos techniques d'analyse de cause de débit au niveau des connexions de TCP pour identifier ces causes de limitation de client. Nous avons découvert que les performances globalement faibles des applications pair-à-pair sont la plupart du temps dues aux limites de débit de téléchargement par les applications. De plus, nous avons montré que les clients saturent rarement leurs liens d'accès. Nous avons également vu que les performances de quelques clients étaient très probablement limitées par les problèmes de configuration de TCP, la fenêtre de récepteur trop petite par défaut.

## E.5 Perspectives

Jusqu'ici, nous avons mis en application un prototype d'InTraBase seulement basé sur PostgreSQL (PgInTraBase). Il serait intéressant du point de vue de la comparaison des performances d'implémenter ce prototype dans un autre système de gestion de bases de données, par exemple Oracle. Nous envisageons d'avoir un prototype qui fonctionner sur Oracle début 2007. Après cela, nous pouvons comparer les performances des deux prototypes.

Nous avons limité notre travail à l'analyse des connexions longues de TCP. Plusieurs travaux de recherche se sont concentrés spécifiquement sur les connexions TCP courtes. Nos techniques pourraient potentiellement fournir des résultats intéressants une fois appliquées à ces cas. Dans [120], les auteurs affirment que le trafic de l'Internet a été dominé par les connexions courtes HTTP. Cependant, ce n'est plus le cas depuis l'apparition de HTTP/1.1 qui emploie principalement des connexions persistantes. Par conséquent, il serait d'intéressant de faire une étude comparative de traces anciennes et récentes.

Nous avons supposé dans nos travaux une politique de service de type FIFO. Bien que FIFO soit toujours beaucoup utilisée dans l'Internet, il y a un nombre de plus en plus important de cas où FIFO n'est pas utilisé ; par exemple, dans les réseaux d'accès par modem câblé et dans les réseaux d'accès sans fil (par exemple 802.11). Puisqu'il y a un intérêt croissant pour les réseaux sans fil, il serait d'intéressant d'évaluer comment on peut appliquer nos techniques d'analyse des causes du débit de transmission dans ces scénarios. Un des défis serait alors l'estimation de la capacité du chemin TCP/IP. Nous utilisons actuellement l'outil PPrate [47]. Cet outil assume une

politique de service FIFO et peut en conséquent produire des estimations de capacité incorrectes pour le trafic d'un réseau d'accès 802.11.

Nos techniques d'analyse des causes limitantes impliquent beaucoup de calculs. Par conséquent, il n'est pas directement possible de les appliquer dans les cas où l'analyse en ligne est nécessaire. Cependant, il serait intéressant d'étudier s'il est possible de transformer une partie des techniques d'analyse en des algorithmes qui marchent en ligne. Il peut être impossible de les appliquer dans une situation où l'on surveille un lien haut débit qui génère une quantité énorme de données à traiter. Par contre, il pourrait être faisable d'avoir outil d'analyse chez le client, par exemple. Un tel logiciel pourrait servir à informer l'utilisateur au sujet des limitations et réagie automatiquement en conséquence. Par exemple, si l'outil observe fréquemment une limitation de niveau transport, le seuil de slow start pourrait être augmenté.

Pour de grandes quantités de données, il est possible d'adopter une autre approche. Après avoir analysé les causes de débit de transmission en utilisant tous nos algorithmes, nous pourrions essayer de trouver des métriques plus simples à calculer, qui pourraient être utilisées dans un scénario particulier. Par exemple, si nous devions surveiller un réseau d'accès, comme dans l'étude de cas d'un réseau ADSL, nous pourrions essayer de regarder des variations de RTTs local (entre une machine de client local et le point d'observation) et distant (entre un centre serveur éloigné et le point d'observation) pour détecter la présence des goulots d'étranglement locaux et distants. De cette façon, nous ferions la première fois des calculs lourds afin d'inférer les causes du débit de référence et utiliser cette connaissance pour trouver des métriques qui soient faciles à calculer dans de futures analyses.