On the Interaction Between Internet Applications and TCP*

M. Siekkinen[1,**], G. Urvoy-Keller[2], and E.W. Biersack[2]

[1] University of Oslo, Dept. of Informatics, Postbox 1080 Blindern, 0316 Oslo, Norway
siekkine@ifi.uio.no
[2] Institut Eurecom, 2229, route des crêtes, 06904 Sophia-Antipolis, France
{urvoy,erbi}@eurecom.fr

Abstract. We focus in this paper on passive traffic measurement techniques that collect traces of TCP packets and analyze them to derive, for example, round-trip times or aggregate metrics such as average throughput. The seminal work of Zhang [1] has shown that for more than 50% of the TCP connections observed, it is not the network bandwidth that limits the throughput but rather the application or mechanisms such as TCP slow start or too small a receiver window. Certain types of analysis of the network characteristics are meaningful only when performed on TCP traffic that experiences minimal interference by the application. To eliminate such interference, we propose a generic method that partitions the packets of a TCP connection in bulk data transfer and in application limited periods: The packets of a bulk data transfer period (BTP) experience minimal interference from the application, while the packets of an application limited period (ALP) experience interference from the application that prevents TCP from fully utilizing the network resources because the application does not produce data fast enough. As a proof of concept, we apply our algorithm to public Internet traffic traces and show that unless the effects of the application are filtered out, studying the end-to-end path and traffic characteristics from a network point of view can produce biased results.

1 Introduction

While the majority of Internet applications today use TCP as a transport protocol, they differ very much in the way they use TCP. Consider, for instance, a P2P application such as BitTorrent that may establish 20-30 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.

Much research has been done to detect anomalies and to characterize TCP traffic in the Internet through passive measurements. This work usually focuses

* This work has been partly supported by France Telecom, project CRE-46126878.
** Work mostly done while at Institut Eurecom.

L. Mason, T. Drwiega, and J. Yan (Eds.): ITC 2007, LNCS 4516, pp. 962–973, 2007.

on the TCP and IP layers, but often ignores the effects of the application on top. When seeking to explain certain characteristics, e.g. burstiness of TCP traffic [2], it is crucial to account for the effects of the application. In addition, it is very important to understand to what extent the applications themselves are responsible for the transmission rates observed in today's Internet, and not limited by the available resources in the network [1].

The contributions of this paper are two fold: First, we present an algorithm that allows to isolate *bulk data transfer periods* (BTP) and *application limited periods* (ALP) within a connection for an application using TCP as transport protocol. We define a BTP as a period where the TCP sender never needs to wait for the application on top to provide data to transfer. Other time periods are defined as ALPs. 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. In this way, it can be applied to traces containing traffic from an unknown mixture of applications. Second, as a proof of concept, we show through examples that unless the effects of the application are filtered out, studying the end-to-end path and traffic characteristics from a network point of view can produce biased results.

Pioneering research work on TCP root cause analysis was done by Zhang et al. in [1] where they identify application limitation as one of the possible causes for achieving a given throughput. Our work differs from theirs by providing a method to isolate the bulk data transfers for further analysis and to evaluate the effect of the application in a quantitative way. Allman [3] recommends to carefully choose the application when evaluating TCP performance. We propose to minimize the effect of the application in the case when it is impossible to make such a choice, e.g. when analyzing traffic traces recorded by third parties.

2 Applications and TCP

Figure 1 describes the way data flows from sender to receiver application using 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. We focus only on the behavior of the sending application since it projects directly the application protocol behavior while the receiving application should always read the buffer `b2` whenever it contains data[1]. When the application sends data constantly, buffer `b1` in Figure 1 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). The interaction between the sending application and TCP manifests itself in the traffic in diverse ways depending on the type of application.

[1] When the receiving application can not read the buffer `b2` fast enough, the receiving TCP will notify the sending TCP by lowering the receiver advertised window.

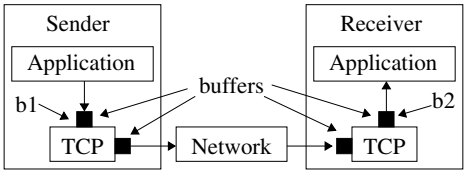

Fig. 1. Data flow from the sender to the receiver application through a single TCP connection

Table 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, typically a single ALP	telnet and instant messaging applications
3	transmission bursts separated by idle periods, applications using persistent connections, BTPs interspersed with ALPs	Web w/ persistent HTTP connections, BitTorrent (choked and unchoked periods)
4	transmit all data at once, single BTP	FTP
5	mixture of 1 and 3	BitTorrent with rate limit imposed by client application

We depict the diverse classes of applications operating on top of TCP in Table 1. This classification illustrates the multiple ways in which the application can influence the TCP traffic. Given such a diversity, it is challenging to design a generic algorithm that separates BTPs from ALPs since the application may interfere on very different time scales.

3 The Isolate & Merge (IM) Algorithm

3.1 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. The algorithm processes bidirectional TCP/IP traffic passively collected at a single measurement point[2].

[2] 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 [4]. Nevertheless, we argue that unidirectional traces are often not sufficient for in-depth analysis, as is the case for round-trip time (RTT) estimation.

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 this threshold 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, which is a common size for the receiver advertised window [5]. For the same reasons, 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.

3.2 Procedures

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

The key insight of the Isolate phase is to use small-size packets and idle times between packets as indications that the application does not provide data fast enough to TCP. Therefore, a large fraction of transmitted packets that are smaller than MSS or an idle time longer than a RTT following a packet smaller than MSS trigger the beginning of an ALP. Similarly, many consecutive MSS packets mark the beginning of a BTP.

In order to understand the reasoning behind the Merge phase, let us consider how the ALPs differ from the TPs. ALPs achieve by definition a lower throughput than TPs, since the application prevents TCP from fully using the network resources. Thus, the application interference is visible as a reduced 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 throughput of the periods involved in the merger.

The mergers are controlled with the threshold parameter $drop \in [0, 1]$. Figures 2 and 3 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

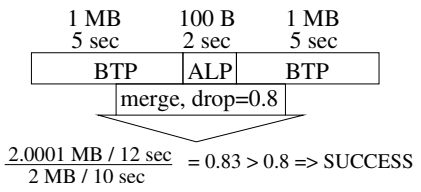

Fig. 2. Successful merger

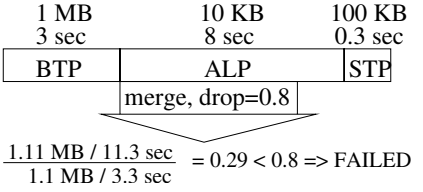

Fig. 3. Failed merger

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, as we show in the extended version of this paper [6], can provide interesting input for characterizing the application behavior. Due to space limitations, we refer the reader for an algorithmic description of the IM procedures to our technical report [6]. The procedures are more complex that one would think at first sight because they are designed to work correctly regardless of the location of the measurement point on the TCP/IP path (i.e. at sender/receiver or middle of path). Because of that, the measurement point location needs first to be determined, as it is not always known a priori, and furthermore, the location has implications to RTT estimation and calculations of the inter-arrival times (IAT) between packets.

We validated the IM algorithm with the help of the Web100 software [7] that allows querying the TCP state information of active TCP connections. We cross-checked the results of applying IM on a large trace of Internet traffic with the Web100 data captured at the sender and observed that the results matched in more than 95% of the cases. For more details please consult [6].

4 Data Sets

We applied the IM algorithm to eight application-specific traffic traces in order to investigate the impact of different applications in our studies in Section 5. Except for the SSH data set, all of the application specific traces were extracted from the same original public ADSL access network traces (the first 19 days from Location 4 traces in the M2C Measurement Data Repository at http://m2c-a.cs.utwente.nl/repository/, two traces per day, 15 minutes each) by filtering on the well-known TCP port numbers of these applications. 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 by P2P applications to bypass firewalls, for example. The SSH traffic data set consists of `scp` downloads from various locations all over the world to a single destination.

Table 2. 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

Table 2 summarizes the characteristics of the traces. Regardless of the application type, BTPs were 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. However, 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.

5 Distortion Due to ALPs on End-to-end Path Studies

BTPs can capture the TCP/IP path properties in a 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 will be necessary 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 due to the presence of ALPs. This distortion is the biasing effects due to the application protocol in measures/estimations of metrics that typically measure the end-to-end TCP/IP path properties. RTT and throughput are generally the two principal ones among such metrics. That is why they are the focus of our two case studies. In the first case study, we study the impact of the application protocol on the characteristics of TCP transfer rates, i.e. the throughput achieved on a given data path. In the second case study, we investigate the impact of the application protocol on the accuracy of RTT

estimation of TCP connections. Our goal is not to demonstrate that connection-level measurements (vs. BTP-level measurements) yield necessarily wrong results (especially in the first case study). Instead, we want to underline the fact that one should carefully consider what is the role and impact of the application running on top of TCP on the measurements when drawing conclusions from them. While there are cases when connection-level measurements are desirable, there are also other cases where they can be misleading.

5.1 Studying Characteristics of Rates

Throughput is a key performance metrics for many 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 directly at the connection level vs. first filtering out 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 3. 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 sign of particular user behavior. For example, a large difference means that the user spends a long time reading the current page before clicking on a new link.

The authors found in [1] 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 4 contains the coefficients of correlation between the rates (throughput) and sizes (number of bytes transferred) computed for entire connections and BTPs of our data sets. In the case of BTPs, the average throughput and the sum of bytes transferred was computed for each connection. As in [1], we compared the logarithms of the rates and sizes because of the large range of values.

While we observe correlation throughout our data sets, the amount of correlation varies a lot depending on the application. Furthermore, when we compare correlation at connection- and BTP-level, the difference in the degree of correla-

Table 3. Mean Values of the Throughput Ratio

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

Table 4. 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

tion 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, manage to exchange more bytes than slower peers. Slow peers are more likely to be choked more often and, thus, exchange 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 first reason is that the choked periods, during which the peer is idle, are identified as ALPs, which, therefore, decrease the connection-level throughput but do not affect the throughput of the BTPs. In this way, connections of slow peers mainly contribute to the difference of the correlations between connection and BTP level for BitTorrent traffic in Table 4. The second reason is the BitTorrent download connections that are not used simultaneously to upload 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.

For SSH traffic, the large difference in the degree of correlation in connection-level and BTP-level 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 [1] 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 [1] 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.

5.2 Case Study on RTT Estimation

The case of RTT estimation is particularly interesting, since the ALPs may distort the estimates in yet another way when the measurement point is in the middle of the TCP/IP path, a common situation in traffic monitoring [8]. In this case, when estimating the RTT from passively collected packet headers and using techniques based on observing bidirectional traffic, the RTT needs to be computed in two parts (see Figure 4): (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 4), i.e. to associate a data packet to an ack packet that triggered its transmission. However, as Figure 4 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 affects every RTT estimation technique that is based on bidirectional traffic observations, such as the technique relying on TCP timestamps to do the ack-to-data association (the authors of [9] acknowledge the problem in Section 4.1) or the technique described in [4] which reconstructs the TCP state machine to track the sender's congestion window, which in turn enables the ack-to-data association.

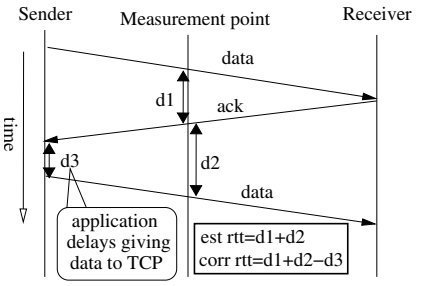

Fig. 4. Problem with RTT estimation during an ALP

In the case that data is constantly transferred in both directions of a connection, an RTT estimate can be obtained by simply computing `d1` in Figure 4 for both directions and then summing them up. Unfortunately, it is rare to have constant data transfers in both directions within a connection.

Other RTT estimation techniques based only on unidirectional traffic observations have been proposed in [9] and in [1]. 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 primarily the technique from [9] relying on TCP timestamps and secondarily the technique from [4] when TCP timestamps were not available. For each connection, we computed the average for two sets of estimated RTT samples: first, including only the BTPs and second, including only the ALPs. We used $drop = 0.9$. The CDF plot in Figure 5 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% of them 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 4).

Figure 6 shows an example of the error that can be introduced by the application. The figure contains estimated RTT samples from a short piece of an 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 of each BTP after an ALP is clearly longer than the other samples and corresponds to the situation depicted in Figure 4. These RTTs are erroneously inflated by the delay due to the application (`d3` in Figure 4), which makes them on the average much larger than the actual RTT. In this example, this delay `d3` due to the application inflates the first RTT sample of each BTP (right after the dotted vertical line in Figure 6) because no packets

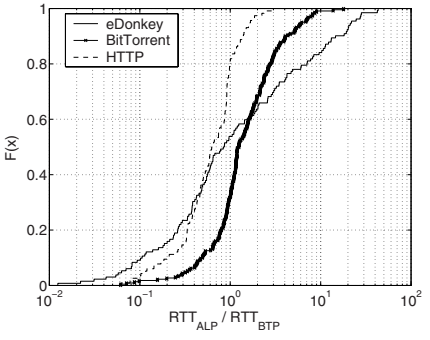

Fig. 5. CDFs of ratio of the mean RTTs: $\frac{\overline{RTT}_{ALP}}{\overline{RTT}_{BTP}}$ ($drop = 0.90$)

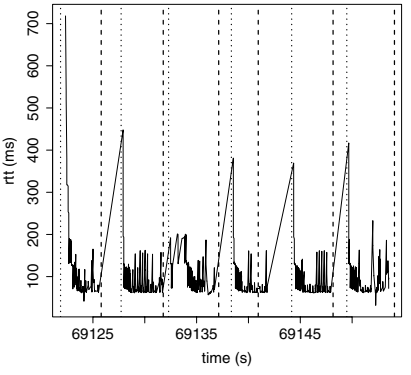

Fig. 6. Piece of an HTTP connection. Dashed and dotted vertical lines start an ALP and BTP ($drop = 0.95$), respectively

are transmitted and, thus, no samples obtained during the ALPs. As a consequence, to correct the error introduced by the application, the first sample of each BTP should be filtered out. This example may partially explain why in Figure 5 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 this paper whose goal is to raise awareness on the potential impact of the application on the analysis results when studying the TCP/IP path and to propose an application agnostic solution for these studies.

6 Conclusions

TCP is the dominant transport protocol carrying a large fraction of the total traffic in the Internet. It is therefore quite natural to use TCP packet traces for network tomography purposes. However, as we have shown in this paper, the application can interfere in various ways with the flow of packets injected into the network. The IM algorithm is able to isolate the "contribution" of the application for TCP packet traces containing traffic of any kind of application. We applied the IM algorithm to a variety of different application traffic and demonstrated the impact of the application when studying the TCP/IP path properties in the case of throughput achieved and RTT estimation.

As a continuation of this work, we have applied the IM algorithm on traffic traces in order to study the role of applications as origins of the rates and throughput performance observed by today's Internet applications. In specific, we have evaluated to what extent clients of ADSL access network experience

throughput limitation by the application [10]. As future work, we would also like to investigate some of the TCP traffic and network path properties, such as RTT and burstiness [2], using publicly available traffic traces (such as the ones used in [2]) in order to quantify the impact of applications on these properties.

References

1. Zhang, Y., Breslau, L., Paxson, V., Shenker, S.: On the characteristics and origins of internet flow rates. In: Proceedings of ACM SIGCOMM 2002 Conference, Pittsburgh, PA, USA (2002)
2. Jiang, H., Dovrolis, C.: Source-level IP packet bursts: causes and effects. In: IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement, New York, NY, USA, pp. 301–306. ACM Press, New York (2003)
3. Allman, M., Falk, A.: On the effective evaluation of TCP. Comput. Commun. Rev. 29, 59–70 (1999)
4. Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., Towsley, D.: Inferring tcp connections characteristics from passive measurements. In: Proc. Infocom 2004 (2004)
5. Medina, A., Allman, M., Floyd, S.: Measuring the evolution of transport protocols in the internet. Comput. Commun. Rev. 35, 37–52 (2005)
6. Siekkinen, M., Biersack, E.W., Urvoy-Keller, G.: On the interaction between internet applications and tcp. Technical report, Institut Eurecom (2006) http://www.eurecom.fr/šiekkine/pub.hml
7. Mathis, M., Heffner, J., Reddy, R.: Web100: extended TCP instrumentation for research, education and diagnosis. Comput. Commun. Rev. 33, 69–79 (2003)
8. Jaiswal, S.: Measurements-in-the-middle: Inferring end-end path properties and characteristics of TCP connections through passive measurements. PhD thesis, Univ. of Massachusetts, Amherst (2005)
9. Veal, B., Li, K., Lowenthal, D.: New methods for passive estimation of TCP round-trip times. In: Proceedings of Passive and Active Measurements (PAM) (2005)
10. Siekkinen, M., Collange, D., Urvoy-Keller, G., Biersack, E.W.: Performance limitations of ADSL users: A case study. In: Proceedings of the Eighth Passive and Active Measurement Conference (PAM) (2007)