Institut Eurécom
Networking and Security Department
2229, route des Crêtes
B.P. 193
06904 Sophia-Antipolis
FRANCE

Research Report RR-09-233

Revisiting the Performance of Short TCP Transfers

January 15th, 2009

Aymen Hafsaoui*, Denis Collange+, and Guillaume Urvoy-Keller*
* Institut Eurecom
+ Orange Labs

Email : Aymen.Hafsaoui@eurecom.fr, Guillaume.Urvoy@eurecom.fr,
Denis.Collange@orange-ftgroup.com

[1]Institut Eurécom's research is partially supported by its
industrial members: BMW Group Research & Technology - BMW Group Company,
Bouygues Télécom, Cisco Systems, France Télécom, Hitachi Europe, SFR,
Sharp, STMicroelectronics, Swisscom, Thales.

Revisiting the Performance of Short TCP Transfers

Aymen Hafsaoui, Denis Collange, and Guillaume Urvoy-Keller

Abstract

Performance of short TCP transfers, e.g., Web browsing, has a direct impact on the way users perceive the health of their Internet access. It is a common belief that TCP performs better with large than with short transfers, as the latters are more likely to time-out and their duration is dominated by the RTT.

In this paper, we revisit the performance of short TCP transfers. We highlight the interplay between TCP and the application on top. We show that while losses can have a detrimental impact on short TCP transfers, the application significantly affects the transfer time of almost all short - and even long - flows in a variety of way. Indeed, the application can induce extremely large tear-down times and it can also slow the rate of actual TCP transfers or affect the ability of TCP to recover using Fast Retransmit/Fast Recovery. We illustrate our findings using several traces from realistic networks including DSL, wireless hotspot and a research lab traffic.

[0]Corresponding author: aymen.hafsaoui@eurecom.fr

Contents

1 Introduction **1**

2 Related Work **2**

3 Data Sets **3**
3.1 Well-behaved connections 3

4 Short Transfers **4**
4.1 Definition 4
4.2 Transfer time break-down 6
4.3 Recovery time 6

5 Application Impact **7**
5.1 Synchronism and Losses 9
5.2 Data Pacing 9

6 Conclusion **11**

List of Figures

1 Trace characteristics 4
2 Transfer time break-down 8
3 Conditional ratio of push flags 8
4 Cumulative distribution of transmitted size blocs 10
5 Application impact for the Portland trace 11

1 Introduction

TCP is the dominant transport protocol currently implemented in the Internet and responsible of the majority of packets and flows sent. Recent measurement studies show that TCP accounts for 60% to 90% of today's Internet traffic [1]. It is used by a large range of applications, including web, email, peer-to-peer file sharing and the newly emerging trend of YouTube-like media streaming.

A large majority of TCP flows are short lived, also known as "mice". Mice can contribute up to 97% of total number of flows and 6% of global traffic [2].This phenomenon was attributed to the domination of Internet flows by web data transfers, which are characterized by short connections. More generally, the interactive traffic of the end users often corresponds to short TCP transfers and a change in their performance directly affects the way the user perceives his Internet access.

A closer look at TCP loss recovery mechanisms brings to light some phenomena which can badly impact short connections. TCP detects and recovers from losses using two basic types of mechanisms: retransmissions timeouts (RTO) and fast retransmit/recovery (FR/R). Normally, a sender must receive at least three duplicate acknowledgments (ACKs) before it triggers a fast retransmit [4]. Short flows in the slow start phase often do not have a congestion window large enough to generate three duplicate ACKs, making timeouts the only loss recovery mechanism available to a TCP sender.

Given the above statements, a commonly used definition for a short TCP transfer is a transfer that can not rely on FR/R to recover from a loss. We will use this definition as a starting point and show that the emergence of new mechanisms to speed up short transfers, like *Limited Transmit* [4] and larger initial congestion window [5] prevents the derivation of a universal threshold in number of packets.

The main contribution of this paper lies in the study of the interplay between TCP and the application on top of it. Indeed, the application can slow down a TCP transfer by: (i) being stalled waiting for data to be crafted by back-end servers or from the end user, (ii) shaping the traffic to a specific rate, or (ii) delaying the closing of the transfer. In addition, the application can worsen the impact of losses by preventing TCP from sending large enough burst of packets. We adopt an application agnostic approach, i.e., we do not make any assumption on the way the application is working, to develop a set of techniques that delineate the impact of the application from other causes that explain a given transfer duration, including the data transfer itself and the recovery time if any.

We rely on a passive study of more than 35,000 TCP connections to assess the impact of the application and the recovery mechanisms of TCP. Those connections originate from a variety of environments: one trace from an ADSL platform of a European ISP collected in 2005, one wireless trace from a public hotspot captured in Portland in 2007 (publicly available on Crawdad [7]) and one trace from a research lab (Eurecom) collected in 2008.

Overall, we find that while losses can significantly impact the performance of short TCP transfers, only a small fraction of the short flows actually experience

losses. In contrast, the application tends to affect the vast majority of the transfers, resulting in a significant drop of performance as compared to a TCP transfers where all the bytes to be sent are present in the application buffer at the onset of the transfer.

The reminder of the paper is organized as follows: related work is reviewed in Section 2. We present the main characteristics of the traces we used in Section 3. Section 4 reports on how to identify short TCP connections. In Section 5, we focus on the many different ways an application can impact a TCP transfer. Finally, Section 6 concludes the paper.

2 Related Work

The study of short TCP connections, a.k.a mice, has been the focus of several studies over the past two decades. The exact definition of a short transfer varies from one publication to the other. Some works rely on a fixed threshold: 10 KB [8], [9], which corresponds to 7 segments with a typical maximum segment size (MSS) of 1460 bytes, 13.5 KB in [10], i.e., 9 segments, or 32 KB [11], which is chosen equal to the median size of HTTP responses with status code 200 (indicates that the client request was successfully received). In [12] authors define short connection as data transfer comprising a number of packets less than or equal to 20 packets, assuming that the maximum congestion size is 8 KB and delayed acknowledgment is turned off. Unlike previous studies, the authors in [13], [14] define short transfers as connections that never leave the slow start phase of TCP.

Modeling short TCP transfers latency has received considerable attention. Several approaches have been proposed that take into account round trip time(RTT) estimation and losses impact. In [9] Cardwell et al. compare analytic models to understand how well several TCP performance models fit TCP behavior under realistic loss rate in the Internet. They propose a first model when the loss rate is zero. When the loss rate is strictly positive, they adapt the model based on the well-known TCP throughput formula of [15] to the case of short flows. In [10], a recursive analytical model is proposed to predict the TCP performance of short lived flow in the presence of losses. Completion time is computed using the connection establishment time and the duration of previous data transfers. Results in [10] show that TCP performance was mainly dominated by time outs.

More recently, the authors in [11] investigated the use short transfers latency prediction techniques based on the TCP throughput formula proposed in [9] and historical observations, being done at the server end. They demonstrated using real traces that prediction based on previous transfers systematically outperforms the analytical approach. Hence, they propose an hybrid approach: an equation based estimation for the first-contact transfer and a smoothed mean of the client previous bandwidths for subsequent transfers.

Few works have focused on the interplay between the transport and the application layers. In [16], the authors analyze passively captured TCP connections

of more than 128 packets. They propose a technique to break each connection into time intervals where the application explains the transfer rate or not, based on the silences and also the rate of push flags observed. In contrast, we focus on small transfers and provide a deeper analysis of the impact of the application on the transfer time (Section 5.2) and also on the impact of the application on the recovery mechanisms of TCP (Section 5.1).

3 Data Sets

Table 1 summarizes the main characteristics of the packet level traces (in tcpdump format) used in this paper. These traces were collected from several different environments: the network of a DSL ISP, a wireless hotspot in Portland and a research lab (Eurecom). Those traces are interesting because of their diversity in terms of access technology but also because of their diversity in terms of applications. For instance, p2p transfers are banned from the Eurecom network while it represents a large fraction of the bytes for the DSL trace. A wireless hotspot should differ from a DSL network in that users tend to focus more on interactive application in such environment and tend to refrain themselves from generating large transfers, e.g. application updates or p2p transfers.

	Capture day	Duration connection	No. of connection	Well-behaved connection	Size in MB	Size in packets
ADSL	2005-05-31	1 min and 29 s	37790	5873	357.51	743683
Portland Hotspot	2007-09-14	2 h and 20 min	5051	3798	174.13	352569
Research Lab	2008-10-20	1 h and 1 min	32153	26837	1567.42	2867321

Table 1: Trace description

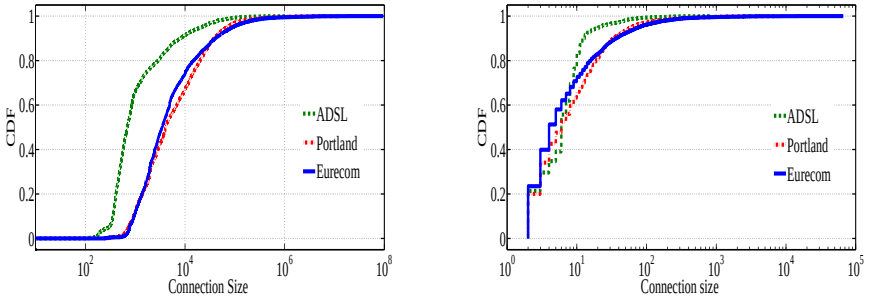
3.1 Well-behaved connections

While analyzing the performance of TCP transfers, we focused on the connections that correspond to valid and complete transfers. Specifically, well-behaved TCP connections must fulfill the following conditions: (i) A complete three-way handshake; (ii) At least one TCP data segment in each direction; (iii) The connection must finish either with a FIN or RESET flag.

When applying the above heuristics, we are left with a total of over 35,000 TCP connections when summing over the three traces (detailed values are given in Table 1). The DSL trace is the one offering the smallest fraction of well-behaved connections, 5873 over 37,790, because of a large number of unidirectional transfers (SYN without a reply). P2p applications tend to generate such abnormal connections (contacting a non available p2p server to download a content), as well as malicious activities.

Figure 1 depicts the cumulative distribution of well-behaved connection size using bytes and data packets of the 3 traces. We observe that the Eurecom and Portland traces offer a similar connection profile that significantly differs from the DSL trace. For instance, 65% of the DSL connections are less than 1 kbytes and 25% are between 1 Kbytes and 1 Mbytes, unlike Portland and Eurecom traffic which offers larger values at similar connection percentiles. A reason behind this obervation is the small duration of the DSL trace. However, our focus is on short transfers, and from this perspective, the DSL trace offers valuable information.

When focusing on the performance of TCP transfers, the number of data packets to be transferred is a key element to consider. We can already observe from Figure 1 that irrespectively of the trace, a significant portion of connections (between 53% and 65%) have less than 7 data packets.



(a) Size in Bytes (b) Size in Packets

Figure 1: Trace characteristics

4 Short Transfers

4.1 Definition

In this section we introduce a first definition of a short TCP connection, which is commonly used in the literature.

A short TCP connection is a connection unable to perform fast retransmit/recovery (FR/R), after a packet loss detection.

While simple, the above definition does not lead to a unique threshold value in terms of number of data packets for a short TCP transfer. Indeed, various TCP implementations and connection characteristics can affect this definition: the initial congestion window, the use of delayed ACK, the number of duplicate acks that triggers a FR/R. For instance, Windows Vista implements Limited Transmit, which means that only 2 duplicate ACKs are enough to trigger a fast retransmit. We estimated for the 3 traces, the number of segments observed in a duration equal to one

RTT after the sending of the first data packet, and this for each direction - see Table 2. The obtained value provides a lower bound on the initial congestion window that the transport uses as the application may not provide TCP with enough data to send at the beginning of the transfer. This is especially true for the initiator side in the case of Web transfer where the GET might fit in a single data packet. Overall, we observe that values of 1 and 2 MSS (and possibly higher values) seem to be common initial congestion windows. Initial congestion windows larger than 2 MSS (we observed values up to 12 MSS) might be due to specific optimizations of operating systems that cache TCP level variables of previous transfers for a few minutes, see e.g. `http://www.csm.ornl.gov/~dunigan/netperf/auto.html`.

Trace	Initiator			Remote party		
	1 pkt	2 pkts	$>$ 2 pkts	1 pkt	2 pkts	$>$ 2 pkts
DSL	99%	1%	0%	80%	18%	2%
Portland	82%	17%	1%	64%	24%	2%
Eurecom	90%	10%	0%	65%	24%	1%

Table 2: Estimated initial congestion window

Given the estimated initial congestion window of Table 2, we report in Table 3 the main scenarios we focus on to find the threshold in terms of number of data packets that triggers a FR/R. A short connection is thus, for each scenario, one with a number of packets strictly smaller than the threshold. Those scenarios cover, to the best of our knownledge, all the most commonly encountered cases.

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
initial cwnd	1	1	2	2
Delayed ACK	no	yes	yes	yes
Duplicate ACK	3	3	3	2
Minimum connection size (data packets)	7	9	8	7

Table 3: Minimum connection size to perform fast retransmit/recovery

Based on the results presented in Table 3, we observe that:

- Different scenarios lead to different thresholds, from 7 to 9 data packets;
- A connection size with less than 7 data packets can't recover from packet loss using FR/R, whatever the exact scenario is;
- When considering a given scenario and a connection whose size is one packet over the threshold, we observe that this connection is able to perform a FR/R for only a single packet in its last round. The loss of any other packet will lead to timeout. A connection is thus not able always to perform FR/R if it is over the threshold.

Based on the result obtained from this section, we adopt a first definition of a short TCP transfer as a connection of size less than 7 data packets. This definition, while

simple, relies on the implicit hypothesis that the application on top of TCP does not impact the way TCP sends packets. As we will see in Section 5, this assumption can be too strong in practice, as even long TCP transfers can be divided into short bursts that prevent TCP from relying on FR/R in case of losses.

4.2 Transfer time break-down

To understand the factors that affect the performance of TCP transfers, we rely on the following decomposition of each transfer into 3 different phases:

Set-up time: this is the time between the first control packet and the first data packet. Since we consider only transfer which have performed a complete three-way handshake, the first packet is a SYN packet while the last one is a pure ACK in general. The connection set-up time is highly correlated to the RTT of the connection. For the three traces we consider, we have a correlation coefficient of 70% for the DSL trace, 60% for the Portland trace, and 39% for the Eurecom trace.

Data transfer time: this is the time between the first and the last data packet observed in the connection. Note that it includes loss recovery durations, if any.

Tear-down time: this is the time between the last data packet and the last control packet of the connection. We impose, as explained in Section 3.1, that at least one FIN or one RESET be observed, but there can multiple combinations of those flags at the end of the transfer. Unlike set-up, tear down is not only a function of the RTT of the connection, but also a function of the application on top of TCP. For instance, the default setting of an Apache Web server is to allow persistent connection but with a keep alive timer of 15 seconds, which means that if the user does not post an new GET request after 15 seconds, the connection is closed. A consequence of the relation between the tear-down time and the application is a weak correlation between tear-down times and RTT in our traces: 40% for the DSL trace (which is still quite high), 0.7% for the Portland trace, and -2% for the Eurecom trace.

Using the above decomposition, we analyze, in the remaining of this article, the impact of losses (Section 4.3) and also of the application (Section 5) on the data transfer time.

4.3 Recovery time

As explained above, the data transfer time possibly includes loss events. We estimate the time spent by TCP in recovering from losses using the *recovery time*. Specifically, for a given transfer, each time the sequence number in the stream of data packet decreases, we record the duration between this event and the observation of the first data packet whose sequence number is larger than the largest observed sequence number seen so far. For instance, assuming that we associate a unique sequence number to each packet, if we observe the sequence 1,2,3,4,7,6,5,6,8, we will record the duration between packet 7 and packet 8. This duration is added to the *recovery time* of the transfer. To filter out reorderings that occur at the net-

work layer, we discard each recovery time smaller than one RTT. Rewaskar et al. [17] developed algorithms to assess whether an observed loss event could be attributed to a time out or a FR/R. We were not able to use this technique as it requires to perform a passive OS finger printing of the sender of the data. However, in our traces, most losses occurred in the data stream issued by the remote party and not the local clients. While p0f (`http://lcamtuf.coredump.cx/p0f.shtml`), which is recommended in [17], is effective when used on SYN packets, it fails when working on SYN/ACK packets, which limits the applicability of the techniques proposed in [17].

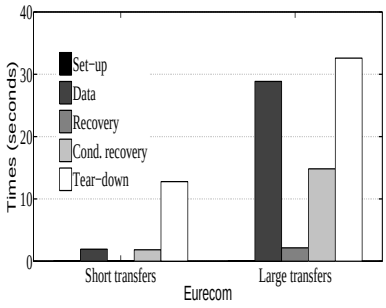
Figure 2 presents the break-down of the small and large TCP transfers for the three traces. We first observe from Figure 2 that while set-up durations are consistently small for all taces and transfer sizes, tear-down take very high values, between 2.5 and 27.5 seconds on average. The tear-down phase in itself often represents the majority of the connection time. Note however, that the tear-down time should have no impact on the performance perceived from the application on top as the data transfer is completed.

As for losses, we present two distinct values for the recovery time: the average conditional recovery time and the average recovery time. The latter is computed over all transfers of the category while the former is computed only for the transfers that experience at least one recovery event. Since only a small fraction of the transfers experience losses (9.4% for DSL trace, 13.2% for Portland and 6.8% for Eurecom) but dramatically increase the data time, the average conditional recovery time is often much larger than the average transfer time. This impact is clearly more pronounced for small than for large flows, over the three traces, most probably because of the predominance of time-outs for short transfers.

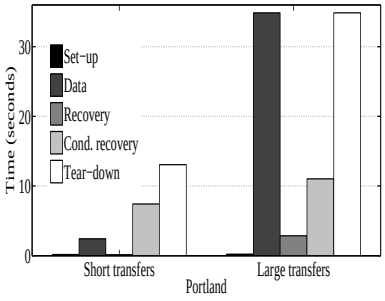
5 Application Impact

In this section, we are interested in assessing the impact of the application on the transfer time of a TCP connection. There are many ways by which the application can influence the pace at which data flows in a network. First, the user might be involved in the transfer, as the case in a persistent HTTP connection, where the download of a new page is triggered by an HTTP Get message issued by the client browser. Second, the application might cap the rate at which information is sent to the TCP layer. This is typically what p2p applications do to limit the congestion on the uplink of the user. A third possibility where the application might affect a TCP transfer is when the generation of data is done online. For instance, when querying Google for a specific keyword, several tens of machines are involved in this operation.

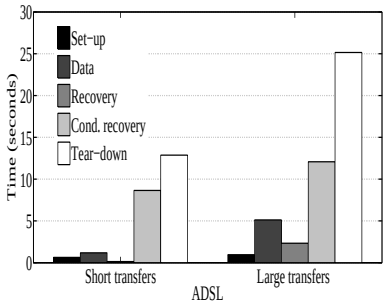
From the above discussion, we observe that the application may affect the transfer of data in many different ways. A first simple assessment that can be made to infer the impact of the application on a TCP transfer is to compute the fraction of packets with PUSH flags. The PUSH flag is a way for the application to



(a) Eurecom



(b) Portland



(c) ADSL

Figure 2: Transfer time break-down

specify that it has no more bytes to send at the moment and the current segment can be sent. We plot in Figure 3 the ratio of PUSH flags as a function of the transfer size for the three traces. We observe that the impact of application as captured by the PUSH flags decreases with increasing transfer size. For the short connections, the push flag ratio is extremely high, between 74% and 86%.

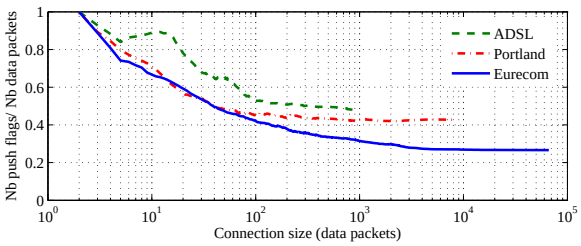


Figure 3: Conditional ratio of push flags

In the remaining of this section, we want to assess in more details the way the application influence the transfer time. We will first show that the application tends to fragment the transfer in small flights of packets that prevent TCP from relying on FR/R in cases of losses. In a second stage, we focus on the way the application forces TCP to pace the data.

5.1 Synchronism and Losses

For client/server applications, one often observes that even if the server is sending a large amount of bytes/packets, the actual exchange is fragmented: the server sends a few packets (hereafter called a train of packets), then waits for the client to post another request and then sends its next answer. If such a behavior is predominant in TCP transfers, it can have a detrimental impact if ever the train size is too small as it might prevent TCP from performing FR/R in cases of losses.

The question we raise is thus: are the two parties involved in a transfer synchronized or not? Proving synchronism requires an a priori knowledge of the application semantics. We can however prove that the synchronism hypothesis cannot be rejected as follows: for a given transfer, each time we observe a transition from one side sending packets, say A, to the other side sending packets, say B, we observe if the first packet from B acknowledges the reception of the last packet from A. If this is not the case, then there is no synchronism, otherwise, synchronism can not be rejected. Applying this methodology to the three traces, we obtained that for each trace, the fraction of connections for which synchronism could not be rejected was extremely high: 88.6% for the ADSL trace, 94.4% for the Portland trace and 95.3% for the Eurecom trace.

For the connections for which synchronism could not be rejected, we looked at the distribution of the size of the trains of packets sent. We distinguished between the initiator of the connection and the remote party, as we expect the latter to be some kind of server that usually sends larger amount of packets than the former that simply posts requests. As illustrated by Figure 4:

- Trains size sent by the remote part are larger than those sent by the initiator, in line with our hypothesis that the remote party be a server;
- More than 97% of initiator trains are less than 3 data packets, which leaves TCP unable to trigger any Fast Retransmit, even if Limited Transmit is used;
- More than 75% of remote part bloc are less than 3 data packets, which again leaves TCP unable to trigger the fast recovery/retransmit, even if Limited Transmit is used;.

Taking a broader perspective, the fraction of connections that have a maximum train size of 3 packets is 85.2% for the DSL trace, 40.5% for the Portland trace and 54% for the Eurecom trace. Sizes of those connections remain quite in line with our definition of Section 4.1 as about 87% of those Eurecom and Portland connections have less than 7 packets. It falls to 62% for the DSL trace. For all 3 traces, we observe the vast majority (over 97%) of those connections have less than 20 packets.

5.2 Data Pacing

In this section, we focus on the transfers that obey to the definition of synchronism introduced in the previous section. For those transfers, we want to assess how

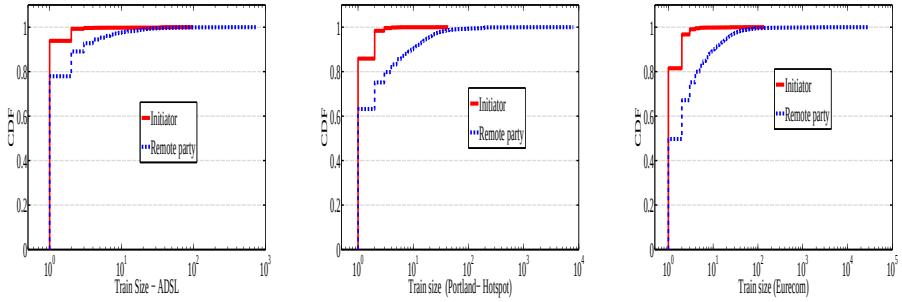


Figure 4: Cumulative distribution of transmitted size blocs

the application[1] slows down the actual data transfer. To do so, we term A and B the two parties involved in the transfer (A is the initiator of the transfer) and we break down the data transfer times into a set of components:

- $T^i_{\text{train time}}(A)$: time needed to transfer the i-th train of the initiator;
- $T^i_{\text{train time}}(B)$: is the time needed to transfer the remote part data train;
- $T^i_{\text{warm-up}}(A)$: time between receiving the last data packet from B and sending train i. The warm-up accounts either for the user thinking time or for some latency to generate the data at the server time for instance;
- $T^i_{\text{warm-up}}(B)$: time between receiving the last data packet from A and sending train i;

Note that to obtain accurate estimates of those durations that are related to the sender or receiver side, we have to shift in time the time-series of packets received at the probe. Specifically, we assume that a packet received from A at probe P was sent $\frac{RTT_{P-A}}{2}$ in the past and will be received $\frac{RTT_{P-B}}{2}$ in the future, where RTT_{P-A} (resp. RTT_{P-B}) is the RTT between P and A (resp. B) . While doing this operation, we assume that the RTT of the transfer stays constant.

The above breakdown strategy results in a complete partition of the total transfer time. The application can impact both warm-up and train times. Concerning train times, we sum for each party, A or B, the total train times, from which we substract the recovery times if any. We term those values $T_{\text{train time}}(A)$ and $T_{\text{train time}}(B)$. We also record the total number of distinct data packets sent by A or B. We next compute the duration that an ideal TCP layer with an initial congestion of 1, delayed acknowledgment turned on, an infinite capacity, an RTT equal to RTT_{A-B} and the same number of packets to send as A or B would take to complete the transmission of all those packets. We term those duration $T_{\text{theory}}(A)$ and $T_{\text{theory}}(B)$. The difference between theoretical quantities and the total train time, $T_{\text{train time}}(A) - T_{\text{theory}}(A)$ and $T_{\text{train time}}(B) - T_{\text{theory}}(B)$, represent estimates of the delay

[1]We consider the application in a broad sense, including the user interactions.

introduced by the application on top of TCP. We term them as pacing time in the remaining of this section.

Figure 5 presents the result of applying the above methodology to the Portland trace. The two other traces offer qualitatively similar results. We observe when looking at Figure 5 that the warm up time of A (initiator) and the pacing time of B (remote party) represent the largest shares of the train time of A and B respectively. A possible explanation behind this observation is that the "average" connection features characteristics close to a client/server application with a large thinking-time of the user, that leads to large warm-up values for A, and a server whose rate is limited either by some back-end server or the use of a rate policy. A precise assessment of the causes behind those phenomena clearly calls for more advanced studies, that we leave for future work. For the time being, the major lesson learned from this study is that the application slow down most transfers in many different ways and this impact is observable for both small and large transfers. This is somehow in contrast to losses, which can an even more detrimental impact, but only for a minority of transfers.

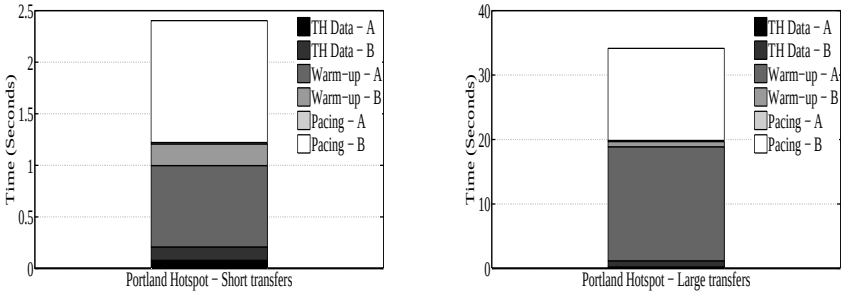


Figure 5: Application impact for the Portland trace

6 Conclusion

In this paper, we have analyzed on three different traffic traces the performance limitations of short and of interactive TCP transfers.

Short transfers sending less than seven packets are not able to apply Fast Retransmit. Thus, they are really sensitive to loss events in the network. These short transfers represent the majority of transfers. We have also observed very long tear-down delays, between the last data packet of the connection and the last control packet. This tear-down delay does not influence the user perception, but it may affect the measurement of response times of short transfers in network management functions.

The sensitivity to loss concerns also many long transfers as many of them are a sequence of alternate exchanges and the vast majority of these bursts are less than

3 packets. Such a feature has a direct influence on the ability of TCP to recover from a loss using Fast Retransmit.

We have also highlighted that the delay to transfer a burst is usually much larger than the pure transmission time. Causes behind this slow downs can be found at the sender, e.g., rate shaping, and also at the receiver side, e.g., thinking time. To the best of our knowledge, this work is the first of its kind to pinpoint and quantify the impact of the application on top of TCP. An important lesson learned from this study is that the while losses can have a highly detrimental impact on the transfer times, losses occur in fact (and hopefully) very rarely. In contrast, the application affect almost all flows and leads to a substantial slow down of the transfers.

References

[1] M. Fomenkov, K. Keys, D. Moore, and k. claffy. Longitudinal study of Internet traffic in 1998-2003. Technical Report, Cooperative Association for Internet Data Analysis (CAIDA), 2003.

[2] M. Mellia, I. Stoica, and H.Zhang, TCP model for short Lived Flows, IEEE Communications Letters, 2002.

[3] J.R. Janardhan, A.L. Caro Jr, P.D. Amer, Dealing with Short TCP Flows: A Survey of Mice in Elephants Shoes, Technical Report, 2007.

[4] M. Allman, H. Balakrishnan, S. Floyd, Enhancing TCP's Loss Recovery Using Limited Transmit, RFC:3042, 2001.

[5] M. Allman, S. Floyd, C. Partridge, Increasing TCP's Initial Window, RFC:3390,2002

[6] J. Nagle, Congestion Control in IP/TCP Internetworks, RFC:896, 1984.

[7] David Kotz and Tristan Henderson and Ilya Abyzov, CRAWDAD data set dartmouth/campus , http://crawdad.cs.dartmouth.edu/

[8] U. Ayesta, K. Avrachenkov, "The Effect of the Initial Window Size and Limited Transmit Algorithm on the Transient Behavior of TCP Transfers", *ITC Specialist Seminar on Internet Traffic Engineering and Traffic Management*, July 2002.

[9] N. Cardwell, Stefan Savage, T. Anderson, "Modeling TCP Latency", *INFOCOM*, 2000.

[10] M. Melia, I. Stoica, H. Zhang, "TCP Model for short Lived Flows", *IEEE COMMUNICATIONS LETTERS*, February 2002.

[11] Martin Arlitt, Balachander Krishnamurthy, Jeffrey C. Mogul, "Predicting short-transfer latency from TCP arcana: A trace-based validation", *ACM SIGCOMM*, 2005.

[12] N. Ben Azzouna, F. Guillemin, "Analysis of ADSL traffic on an IP backbone link", *GLOBECOM*, 2003.

[13] S. Ebrahim-Taghizadeh, A. Helmy, S. Gupta, "TCP vs. TCP: a Systematic Study of adverse Impact of Short-lived TCP Flows on Long-lived TCP Flows", *INFOCOM*, 2005.

[14] C. Barakat, E. Altman, "Performance of short TCP transfers", *Networking*, 2000.

[15] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", *In Proceedings of ACM SIGCOMM 98*,pages 303-314,Vancouver, BC, September 1998.

[16] M. Siekkinen, G. Urvoy-Keller, and E. W. Biersack, "On the interaction between internet applications and TCP", In Proceedings of ITC20, 2007.

[17] S. Rewaskar, J. Kaur and FD. Smith, "A Passive State-Machine Approach for Accurate Analysis of TCP Out-of-Sequence Segments", In the ACM SIGCOMM Computer Communication Review, July 2006.