Root Cause Analysis for Long-Lived TCP Connections

M. Siekkinen, G. Urvoy-Keller, E. W. Biersack, T. En-Najjary
Institut Eurécom
BP 193, 06904 Sophia-Antipolis Cedex, France
Email: {siekkine,urvoy,erbi,ennajjar}@eurecom.fr

ABSTRACT

While the applications using the Internet have changed over time, TCP is still the dominating transport protocol that carries over 90% of the total traffic. Throughput is the key performance metric for long TCP connections. The achieved throughput results from the aggregate effects of the network path, the parameters of the TCP end points, and the application on top of TCP. Finding out which of these factors is limiting the throughput of a TCP connection – referred to as TCP root cause analysis – is important for end users that want to understand the origins of their problems, ISPs that need to troubleshoot their network, and application designers that need to know how to interpret the performance of the application. In this paper, we revisit TCP root cause analysis by first demonstrating the weaknesses of a previously proposed flight-based approach. We next discuss in detail the different possible limitations and highlight the need to account for the application behavior during the analysis process. The main contribution of this paper is a new approach based on the analysis of time series extracted from packet traces. These time series allow for a quantitative assessment of the different causes with respect to the resulting throughput. We demonstrate the interest of our approach on a large BitTorrent dataset.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Network**]: Network Operations—*Network monitoring*; C.2.5 [**Computer Communication Network**]: Local and Wide-Area Networks—*Internet*

General Terms

Algorithms, Measurement, Performance

Keywords

Internet, TCP root cause, throughput, time series.

1. INTRODUCTION

Motivation: During recent years, the Internet traffic has experienced a massive growth as the number of users skyrocketed. This applies also to the amount of traffic per user, as the capacities of access links have increased by several order of magnitudes. New Internet applications, such as peer-to-peer (P2P), have emerged and the relative importance of HTTP and FTP has decreased. On the other hand, TCP still remains the dominating transport protocol that conveys the vast majority of the traffic – typically more than 90% of the bytes. As a consequence, the behavior and performance of TCP in the Internet is a major concern. The heterogeneity of today's applications running on top of TCP and the diversity of the environments they are used in, imply that a meaningful analysis can only be done during their operation in the Internet.

Throughput is the most important performance measure for long TCP connections. The achieved throughput represents the aggregate effects of the network path, the end points, and the application. Our research tries to find out which of them are responsible for limiting the throughput of a given TCP connection at a given time instant. Knowledge about these root causes can be used by Internet Service Providers (ISP) for quality of service evaluation and troubleshooting. Traffic modeling is another area that would benefit from this knowledge, leading to more accurate workload models of TCP traffic. It is equally important for Internet application designers to know when the limiting factor is the network or the TCP end points and when it is the application.

Approach: We adopt an approach that requires as input bidirectional packet header traces captured at a measurement point along the path from source to destination and produces as output quantitative information about the limitation causes of TCP's throughput per connection. We assign a score to each of these limitations and track the evolution of limitation scores with time. For this purpose, we base our approach on a set of time series generated from the (TCP and IP) packet headers. We focus on long transfers where TCP slow start no longer dominates the throughput achieved.

As stated above, we distinguish three main classes of root causes: (i) limitations due to the application, (ii) limitations due to the TCP end-hosts and (iii) limitations due to the network. We apply a divide and conquer approach to infer the limitation causes. First, the periods where the throughput is determined by the application are isolated. The remaining trace data consists of so called *bulk transfer* periods. We then apply a set of tests to derive the most likely cause (or causes) that explains the performance of each bulk transfer period. Whenever possible, the algorithms are validated using live measurements in the Internet that are compared against the results given by Web100 [10]. We also used NIST Net [4] to create specific conditions for a given transfer.

Zhang et al. [18] performed pioneering research work into the origins of Internet TCP throughput limitation causes. They defined taxonomy of rate limitations (application, congestion, bandwidth,

CoNEXT'05, October 24–27, 2005, Toulouse, France.

sender/receiver window, opportunity and transport limitations) that we build on and introduced the TCP Rate Limitation Tool (T-RAT). T-RAT turned out to suffer from a number of limitations. First, to identify rate limitation, T-RAT needs to identify so called "flights" of packets. These flights often cannot be identified, as we will see in section 3, which undermines the main premise of T-RAT. Second, T-RAT breaks long connections into "flows" of at most 256 consecutive packets. In contrast, we perform true connection level analysis.

Challenges: The problem is very challenging for several reasons. Operating at connection level complicates the analysis because with long connections, we have a higher probability to observe several limitation causes over different periods of time. For instance, some Internet applications such as BitTorrent [8] or HTTP1.1 operate by switching between active transfer periods and passive keep-alive periods. Our first challenge is to detect those different periods and analyze them separately.

The great number of parameters that influence the behavior of a TCP connection is also a major issue: round-trip time (RTT), receiver advertised window, link capacities, available bandwidth, delayed acknowledgment, and TCP version to name a few.

Contributions: The contribution of this paper is threefold: First, in section 2, we discuss the problem of inferring causes for TCP's transmission rate limitation by elaborating more on the limitation concepts themselves with respect to the approach taken by T-RAT. Second, in section 3, we demonstrate through simulations that an important characteristic of a TCP transfer, the so called flights, have in many cases a very different form than the one assumed in T-RAT. Third and most importantly, we provide a set of algorithms to infer causes that limit the throughput of a given TCP transfer (sections 4 and 5), and apply the algorithms to an example set of real Internet traffic (section 6).

2. CAUSES FOR RATE LIMITATION

In this section, we discuss the different rate limitation causes that we want to infer. While the classification is inspired by T-RAT, we extend the scope of their work, and exemplify the difficulties of identifying certain causes or assessing the impact of others. We present the causes in a top down manner, starting from the application level down to the network level.

2.1 Application

The application operating on top of TCP can be the cause for the throughput achieved. In this case, TCP is not able to fully utilize the transport or network layer resources because the application does not produce data fast enough. There exist two scenarios where the application is the limitation cause.

In the first scenario, the application is producing small amounts of data at a relatively constant rate for the TCP layer. This results in small bursts of packets, in the extreme case a single packet of size less than the maximum segment size of the connection. Typical examples are live streaming applications such as Skype [1] that transfer data over TCP at a constant rate of 32 Kbit/s. Also, applications that use permanent TCP connections and send keep-alive packets during inactive periods, fall in this category (BitTorrent exhibits this behavior during choke periods -see [8]).

In the second scenario the application is producing 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, causing another transfer period.

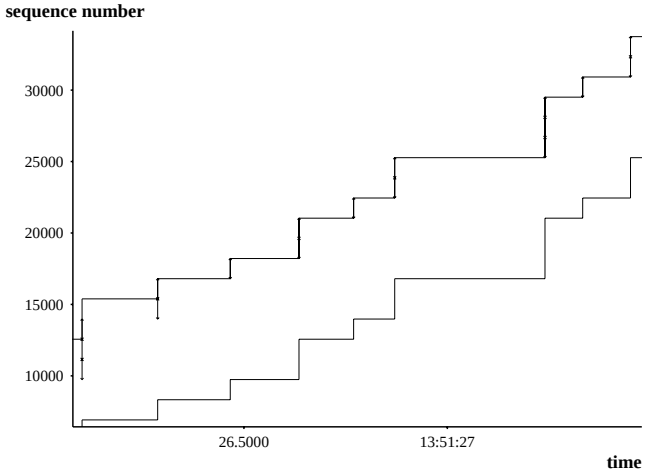


Figure 1: A piece of a receiver window limited connection.

2.2 TCP End-Point Limitations

The achieved throughput of TCP can be limited by the size of the buffers allocated at the two end-points of a connection. The receiver buffer (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. 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. Following the convention of T-RAT, we call the first limitation *receiver window* limitation and the second one *sender window* 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 1 shows a time vs. sequence diagram of an 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 most of the time, 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. As we will see in section 5, identifying a receiver window limitation is possible using the advertised window information carried by TCP packets. On the other side, identifying a sender window limitation is a much more complex task, that we do not address in this work. In [18], the authors use the notion of *flight size* to infer a sender-window limitation. However, we will see in section 3 that identification of flights is, most of the time, impossible. We expect that for most transfers in the Internet, the sender buffer to be at least the size of the receiver window. Indeed, 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) 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. However, a re-

cent study [12] 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.

There is an additional type of limitation at the transport layer that is referred to as opportunity limitation in T-RAT. 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. As will become clear later, we concentrate on analyzing long TCP connections in which case opportunity limitation will not be an issue.

2.3 Network Limitation

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 [7] for a study on the location and lifetime of bottlenecks in the Internet). While other network factors, such as link failures or routing loops [16], might impact a TCP connection, we do not consider them in the present work as we can reasonably expect their frequency to be negligible as compared to the occurrence of bottlenecks.

For the following, we need to borrow a few definitions from [5]. 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:
$C = min_{1,\ldots,H} C_i$, where H is the number of hops in the path

the 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

the average available end-to-end bandwidth A in a path:
$A = min_{1,\ldots,H} A_i$, 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 throughput obtainable by a single TCP connection at that link

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

As in [5], we call the link i with the capacity $C_i = C$ the *narrow link* of the path and the link j with the the average available bandwidth $A_j = A$ the *tight link* of the path. 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. Please note that the bottleneck link is not (necessarily) the same as the tight link. Also the available bandwidth differs from the bulk transfer capacity of a path: The classical example is the case of a link of capacity C used at 100% by a single TCP connection. The available bandwidth is zero on the link while the bulk transfer capacity should be $\frac{C}{2}$.

If the bottleneck link explains the throughput limitation observed for a connection, we declare this connection as *network limited*.

Packet losses are natural indicators of a bottleneck and we will use the packet loss rate as a measure of the impact of the network on a connection. However, note that the packet loss rate by itself does not fully characterize the impact of congestion on the throughput of a given connection. Especially, 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 [11]: $Tput = \frac{MSS}{RTT}\frac{C}{\sqrt{p}}$, 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).

For the above reasons, we will use two metrics to infer if a connection is network limited: (i) the retransmission rate of the connection and (ii) the dispersion ratio (see section 5.2) that can be used to detect if the bottleneck link is shared or not. Figure 2 shows a time vs. sequence diagram of a connection whose throughput is limited by a non-shared bottleneck link. The regular spacing between sent data segments and similarly of the acknowledgments received (tracked by the right limit of the sliding window) is easy to observe.

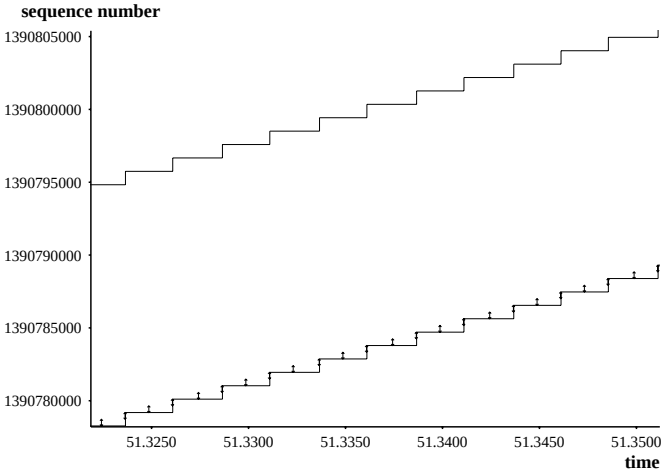
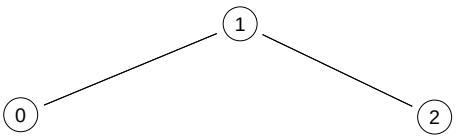


Figure 2: A piece of a bandwidth limited connection where packets are regularly spaced due to the bottleneck link.

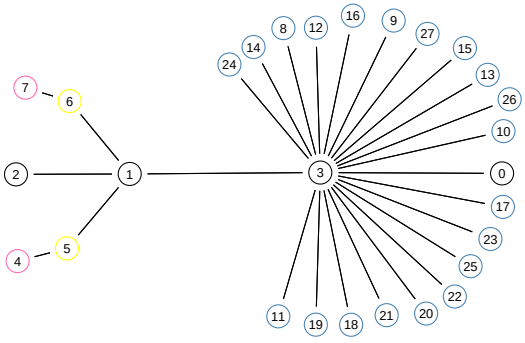
3. ON THE FLIGHT NATURE OF TCP

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, congestion avoidance, and loss recovery. However, in [14], the authors search for flights in Internet traffic traces and arrive to the conclusion that flights can be rarely identified, which means that a tool such as T-RAT is 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 interarrival times (IATs). For example, in the case of a receiver window limited connection one would expect to observe a bimodal distri-



(a) The Basic Configuration.



(b) The Configuration With Cross Traffic

Figure 3: The Configurations for the Simulation.

bution 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. 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 Δt_1 and one time an IAT of Δt_2. In the following, we show a few examples to demonstrate that this type of simple reasoning rarely holds.

We start with a simple topology with one client (node 0), server (node 2), and one intermediate router (node 1) shown in Figure 3(a). A two-minute long FTP transfer was set up on top of a TCP connection established from node 2 to node 0. Figures 4 and 5 show histograms of IATs of packets where the connection is limited by the receiver advertised window of 20 packets. In Figure 4, delayed acknowledgments were not used by the TCP receiver while in Figure 5 delayed acknowledgments were used. As expected, in Figure 4 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 Δt_1 in Figure 5 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 3(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 6 shows an example evolution of the probability density function (pdf)[1] 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.

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. Therefore, T-RAT [2] [18] will be unable to work properly in many cases.

4. TIME SERIES-BASED APPROACH

Our approach to infer the TCP root causes of a given connection is to generate a number of time series using the packet trace of the connection. We then use these time series to compute scores that characterize the impact of the different causes. As the location of the measurement point on the path impacts the way the time series are generated, we devote the next subsection to this issue. We then briefly present all the time series used in our tests, along with some tests made to validate some heuristics used to derive these time series.

4.1 Measurement Point Location

Most of the time series that we use, are very easy to compute if the packet trace was 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, 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 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 $\frac{d2}{d1} < 0.01$.

More generally, we need to compute the RTT samples for each connection. 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 od a data packet and the first acknowledgment for this data packet[3]. In

[1]We compute the pdf estimates using a kernel density estimation technique [15] with Gaussian kernel function.

[2]No publicly available version of T-RAT has been released so far.

[3]Here 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.

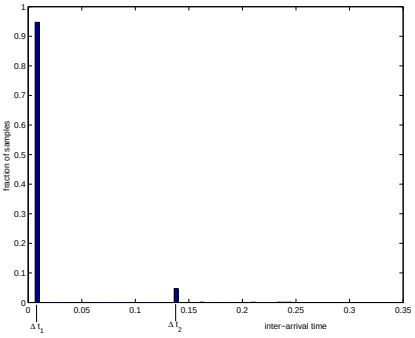


Figure 4: Without Delayed ACKs

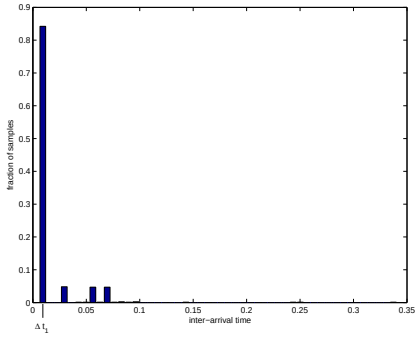


Figure 5: With Delayed ACKs

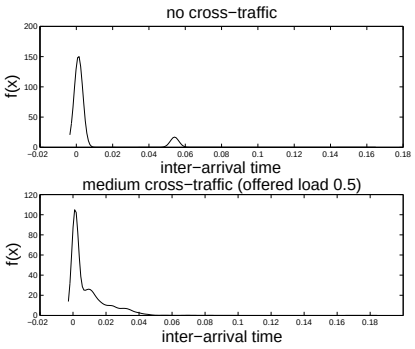


Figure 6: Evolution of the pdf of the inter-arrival times of packets from a receiver window limited connection without and with cross traffic.

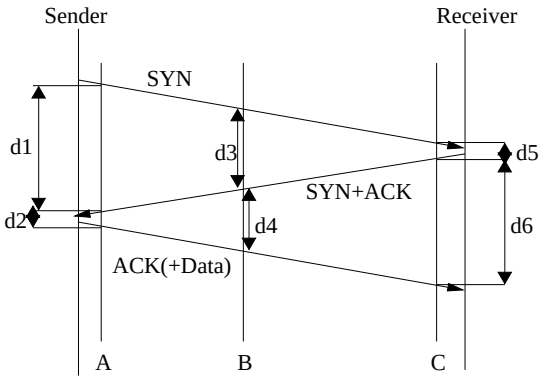


Figure 7: Determining the measurement position from the three-way handshake of TCP

the case that our measurement point is not close to the sender and TCP timestamps are available, we implement the method described in [17].

4.2 Time Series

In this section, we list all the time series that we use in our tests to find the root causes for the throughput seen by a TCP connection.

Fraction of Pushed Packets: A pushed TCP packet is sent with a PUSH flag. RFC-793 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.". Pushed packet thus indicates that the application on top of TCP has for the moment no more data to send. We compute for each direction of a TCP connection the time series of the fraction of pushed packets observed over all consecutive non-overlapping time intervals of fixed duration. To compute those fractions, we only consider packets carrying data and discard pure acknowledgments. If no packets have been seen during a given time window the value is set to -1.

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.

Retransmission Rate: We compute for each direction of a TCP connection the time series of the retransmission rate as the fraction of retransmitted bytes per all (data) bytes transmitted in consecutive time intervals of 1 second. 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 value [2] [3] higher than any previously observed values.

Note that we cannot rely on observing retransmitted packets twice and counting them since the packets may be lost before the measurement point especially if the measurement point is far from the sender. 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.

Receiver Advertised Window: We compute the 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, when the measurement point is at C, we would shift in time the timestamp values of packets sent by the receiver by $+\frac{d6}{2}$, which is the estimated time at which this packet should arrive at the sender. Note that $d6$ will be estimated using the technique borrowed from [17] as indicated in section 4.1

Outstanding Bytes: Another value 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, 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 above algorithms (at the sender and at the receiver side) are heuristics that we tested with real transfers on the Internet. We analyzed `scp` transfers from a Web100 enabled machine to another machine and ran `tcpdump` at the sending or receiving machine. Web100 is a kernel patch that allows to access the actual internal variables of the active TCP connections of a host. It provides the exact values for the sending TCP's retransmission queue size, which corresponds to our definition of outstanding bytes. We did `scp` transfers and compared the values obtained from Web100 and our algorithms: from Institut Eurecom to University of Oslo in Norway and from Eurecom to University of Navarra in Spain. As the transfer to Spain proved to be over a lossy path (with approx. 6% of retransmitted bytes) and the one to Oslo not, we were able to capture two different environments.

Figures 8(a) and 8(b) show the comparison in the case where the measurement point is at the sender and receiver side, respectively. Unfortunately, we were unable to dump traffic in the machine located in Oslo and present only the results from the transfer to Spain in the receiver side case. In Figure, 8(a) the two curves for the transfer to Spain are plotted on top of each other, indicating a nearly perfect agreement. In the case of the transfer to Oslo there is a difference of approximately one MSS on average (note that the window scale option was negociated between the two parties in this experiment). The reason for this discrepancy is not clear and may be due to timestamp inaccuracies caused by the high throughput of this transfer. Figure 8(b) for the case where the measurement point is at the receiver side also shows a good agreement between the estimated and actual values.

From the above experiments, we conclude that, in most case, we can expect to observe a maximum a discrepancy between the actual and estimated values that remains below one MSS, a good enough precision for the tests based on these time series (see section 5.2.1).

5. IDENTIFYING AND ANALYZING BULK TRANSFER PERIODS

In this section, we show how we use the time series introduced in section 4 to separate bulk transfer periods from application limited periods. We also introduce the different tests for TCP end host and network limitations.

5.1 Identifying Bulk Transfer Periods

The first operation we perform on a connection is to separate periods limited by the application from other periods. We identify the active phases of a connection where TCP consistently transfers and call them bulk transfer periods.

We identify bulk transfer periods using the time series of fractions of pushed packets using time window of 0.5 seconds. A smaller value for the duration of a time window would risk to interpret the idle time due to the sender waiting for new acknowledgments after having sent a congestion window full of packets, as an indication of application limitation. The algorithm used to separate bulk transfer periods from application limited periods varies between two states: active and inactive. We define the starting state to be inactive. The algorithm switches to the active state (start of a new bulk transfer period) if the fraction of pushed packets is consistently below a value p for Δt_1 consecutive time periods. The algorithm switches back to the inactive state (end of the current bulk transfer period, and start of a new application limited period) if the fraction of pushed packets observed has been consistently above p or if no traffic was sent for Δt_2 consecutive time periods.

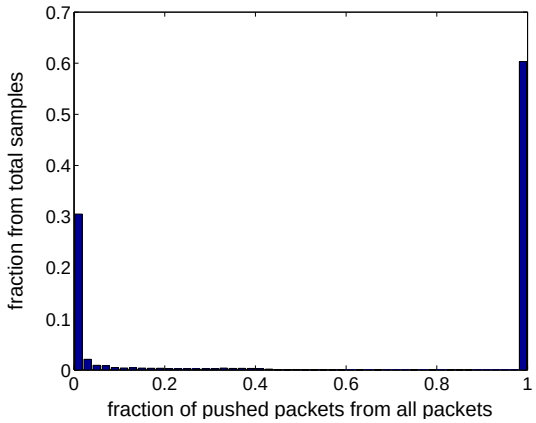


Figure 9: Histogram of the time series values of the fractions of pushed packets for all connections of a 10GB BitTorrent packet trace.

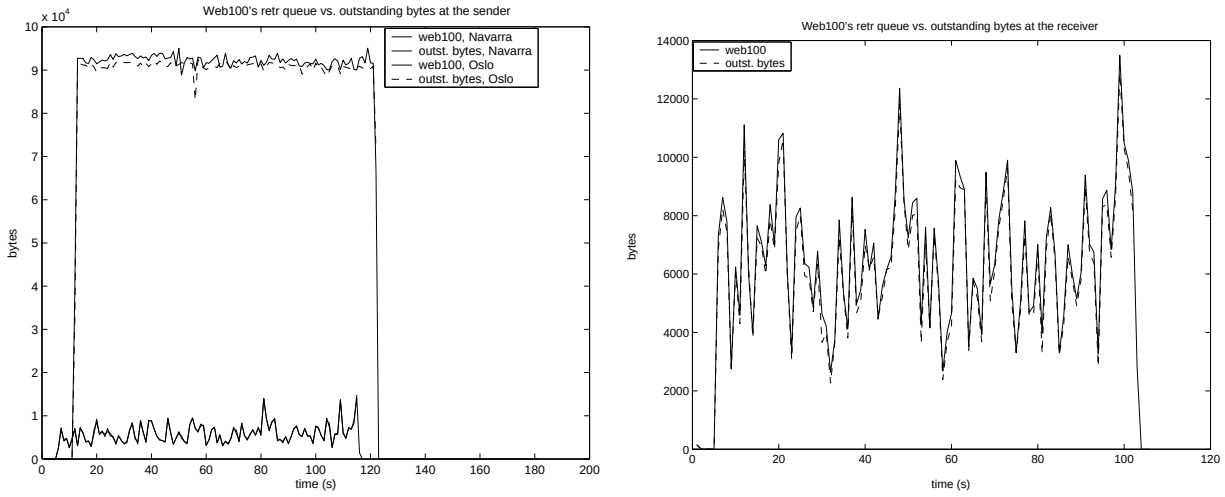
The algorithm is able to recognize both types of application limited periods discussed in Section 2 since we consider both the idle time and the ratio of pushed packets. However, the thresholds p, Δt_1, and Δt_2 need to be tuned according to the type of input traffic and the focus of the analysis. In section 6, we present results for a 10Gbytes BitTorrent trace and we chose p as follows: We extracted the time series of the fractions of pushed packets during one-second time intervals for all those connections of the trace that had more than 10 data packets (we do not process these small connections in any case, see Section 6). We then computed a histogram of all these values, (Figure 9). Based on this histogram we set p to 0.7 but clearly choosing any value from $[0.5, 0.95]$ would practically give the same result. We set $\Delta t_1 = 5$ seconds and $\Delta t_2 = 10$ seconds, which implies that we discard any transfer periods shorter than five seconds.

5.2 Bulk Transfer Period Analysis

After separating bulk transfer periods from application limited periods, we apply the tests for TCP end-point and and the networking limitations on the identified bulk transfer period. These limitation tests are not exclusive. Each of the tests yields a score between 0 and 1 that quantifies the level of the limitation, which is a major improvement over T-RAT [18] that was only providing qualitative results, i.e. a binary answer for each test.

5.2.1 Receiver Window Limitation

We use two time series to test for receiver window limitation: the outstanding bytes time series and the receiver advertised window time series. 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. 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 $n * MSS$ and zero otherwise (in section 6 we discuss the impact of the n value). The receiver limitation score is the average value of the resulting binary time series for the analyzed bulk transfer period.



(a) Measurement point is at the sender.

(b) Measurement point is at the receiver.

Figure 8: Validation of the outstanding bytes algorithms.

5.2.2 Network Limitation

We use two metrics to infer whether the network limits the throughput of a connection: (i) the retransmission score and (ii) the dispersion score.

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, retransmission score does not exactly correspond to the loss rate. However, we can expect these quantities to be close to one another 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 on the throughput of a connection. We introduce this factor starting from the simple case of a non-shared bottleneck in the network, next moving to the case where there is a shared bottleneck which is the narrow link of the path up to the general case where the bottleneck link is not the narrow link of the path. The dispersion score is computed from the times series of the inter-arrival times of acknowledgments and the average throughput $tput$ of the bulk transfer period under consideration.

Let us first consider the case where the network limitation consists of a non shared bottleneck on the path. The bottleneck is evidently the narrow link of the path. Let r be the capacity of the narrow link. The histogram of the inter-arrival times of acknowledgments (computed as explained in Section 4.2) should exhibit a mode located at $\frac{MSS}{r}$ that contains most of the mass. Since the bulk transfer period is network limited and the narrow link is not shared, the ratio of $tput$ to r should be approximately equal to 1, i.e. $\frac{tput}{r} \sim 1$. We define the *dispersion score* as $1 - \frac{tput}{r}$.

Consider now the more complex case where the bottleneck link is still the narrow link of the bulk transfer period but it is now shared [4]. The histogram of the inter-arrival times of acknowledgment should still exhibit a mode located at $\frac{MSS}{r}$. However, since the bottleneck is now shared, this mode will contain a smaller fraction of the total mass of the histogram. Also, the ratio $\frac{tput}{r}$ represents the share the connection obtains at the narrow link during this bulk transfer period.

Consider now the more general case where the bottleneck is not the narrow link. The mode at $\frac{MSS}{r}$ in the histogram of the inter-arrival times of acknowledgments should still persist, though less pronounced, and it is thus still possible to identify the capacity of the narrow link. (We refer the reader to [9] for a related work that takes advantage of the distribution of the inter-arrivals of packets to identify link capacities.) In this case, $\frac{tput}{r}$ does not represent any more the share that the connection obtains at the bottleneck. However, the dispersion score can still be seen as an indicator of the distortion (or dispersion) introduced by the network.

We validate the method to infer the narrow link capacity by setting up an artificial narrow link with NIST Net at Institut Eurecom, the receiving end of the path, and transferring a large file with scp to Eurecom from University in Oslo (UiO), University of Navarra (UN), and Helsinki University of Technology (HUT). We performed two experiments with three parallel transfers for two different narrow link capacities (2 Mbit/s, 5 Mbit/s) in order to observe the effect of cross traffic, and three experiments with a single transfer each to observe the impact for dispersion score. The results in table 1 show that the accuracy of the estimated r value in these tests is good regardless of the retransmission score on the path. We also observe that the low dispersion scores correctly reveal that there is no sharing at the bottleneck.

6. EXPERIMENTAL RESULTS

6.1 Dataset

We applied our algorithms to a 10 Gbytes `tcpdump` packet trace

[4] In practice, detecting this case is not an easy task and would require the use of tools like `Pathneck` [7] that detects bottlenecks in combination with a tool that measures the capacity of a path [5]. This study is out of the scope of the present work

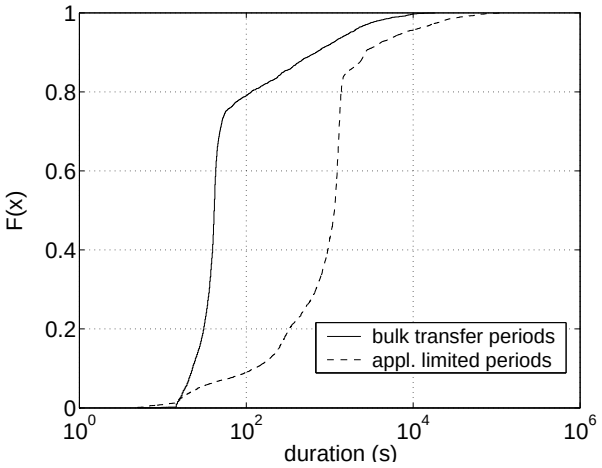


Figure 10: Cdfs of durations of the periods

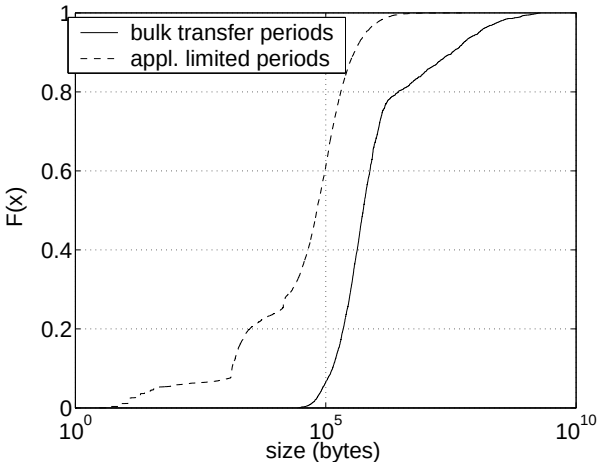


Figure 11: Cdfs of the size of the periods in bytes

Table 1: Validation results from inferring the capacity of the narrow link.

src	set r	r estimate	$1 - \frac{tput}{r}$	retr score	RTT
UN	2.0 Mbit/s	2.0 Mbit/s	0.89	0.175	56 ms
UiO	2.0 Mbit/s	2.0 Mbit/s	0.43	0.027	71 ms
HUT	2.0 Mbit/s	2.0 Mbit/s	0.69	0.037	71 ms
UN	5.0 Mbit/s	5.1 Mbit/s	0.92	0.089	56 ms
UiO	5.0 Mbit/s	5.1 Mbit/s	0.35	0.008	72 ms
HUT	5.0 Mbit/s	5.1 Mbit/s	0.78	0.013	71 ms
UN	2.0 Mbit/s	2.0 Mbit/s	0.21	0.055	57 ms
UiO	2.0 Mbit/s	2.0 Mbit/s	0.10	0.014	70 ms
HUT	2.0 Mbit/s	2.0 Mbit/s	0.03	0.005	71 ms

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.

6.2 Separating the Wheat From the Chaff

Out of the $60,000$ initial connections, we first filtered out the $57,118$ connections with less than 10 data packets. We then applied our algorithm to isolate the bulk transfer periods and the applications limited periods to the remaining 2882 connections. We discarded the connections that consisted of a single application limited period (our algorithm discards transfer periods of less than 5 seconds, refer to Section 5.1). We ended up with *only* 686 connections (to 413 hosts) consisting of 3295 bulk transfers and $10,365$ application limited periods.

Figures 10 and 11 show the cumulative probability density functions (cdf) of the durations and sizes in bytes of both types of periods. We observe that even though BitTorrent is sending only small protocol messages (e.g. to request a block or a piece, or to keep connection alive) during the periods when it is non active or only downloading data, i.e. the application limited periods, the duration of those periods is so large as compared to the bulk transfer periods (figure 10) that eventually the total amount of bytes of some of the application limited periods can be non negligible as compared to the amount carried by some of the bulk transfer periods (figure 11). Indeed, a closer look revealed that the ones transferring up to 2.5 Mbytes are several hour-long connections sending small packets with push flags at a very low rate ($< 5Kbit/s$).

For the bulk transfer periods the coefficient of correlation between the throughput and the size is 0.65 and between the throughput and the duration is 0.52. Such strong correlations are the consequence of the BitTorrent protocol that favors fast transfers between peers. Hence, the faster the transfer, the more likely it is to last, and thus to be large.

6.3 Receiver Window Limitation

Figure 12 shows a cdf of the receiver window limitation score for different values of n (see section 5.2.1). Clearly, the choice of n is not critical as the shape of the curve remains practically the same for $n \in \{1, 2, 3\}$. We use in the following analysis $n = 2$. We observe that approximately 65% of the transfers are never limited by the receiver window and 17% are limited half of their life time. Only a small fraction of the transfers are limited more than 90% of the time by the receiver window. In Figure 13, the receiver window limited score is plotted against the mean value of the receiver advertised window size. The three most common advertised window values are distinguishable from Figure 13 as horizontal stripes: 8, 16, and 64 Kbytes, an observation that agrees with [12]. The coefficient of correlation between the limitation score and average advertised window size is -0.37 which indicates that it is more probable to be receiver window limited when the average advertised window value is smaller. However, we note that there is a significant amount of transfers with a high limitation score and an average advertised window of 64 Kbytes, the largest usable value without window scaling. This observation suggests that perhaps, in some cases, a higher throughput could be obtained by using the window scaling option, though it is not certain and depends on the amount of available bandwidth on the path. Indeed, using a larger window value might equally lead to congestion and lower throughput [13].

We found that all of the 413 client hosts used window scaling or supported its usage in our dataset. Nevertheless, approximately 93% of the hosts did not scale their own advertised window, while 6% used a value of 2 and 1% a value of 7. This agrees with the results in [12] where 97% of the hosts that support window scaling set the value to zero. On the other hand, only 26.6% of the hosts in

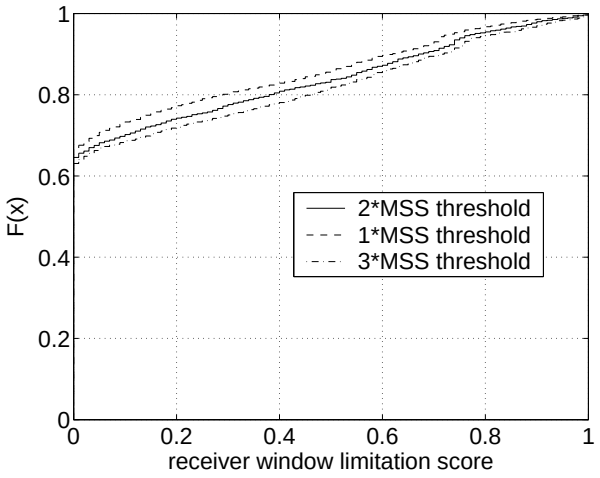


Figure 12: Cdf of receiver window limitation score

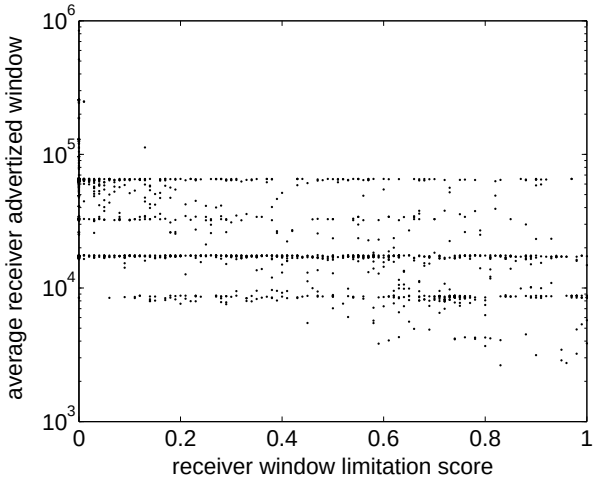


Figure 13: Receiver window limitation score vs. mean advertised window size

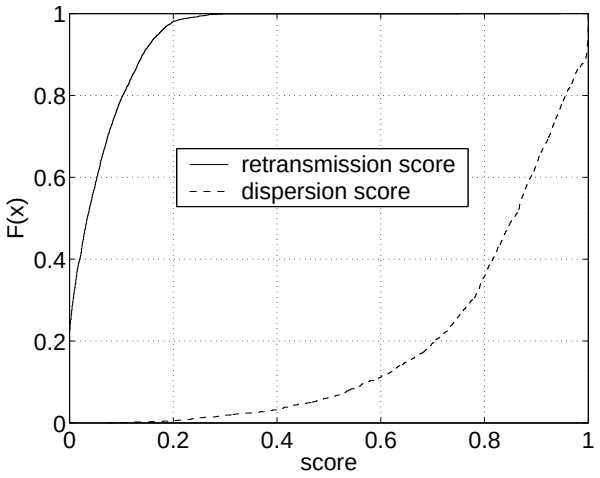


Figure 14: Cdfs of network limitation scores

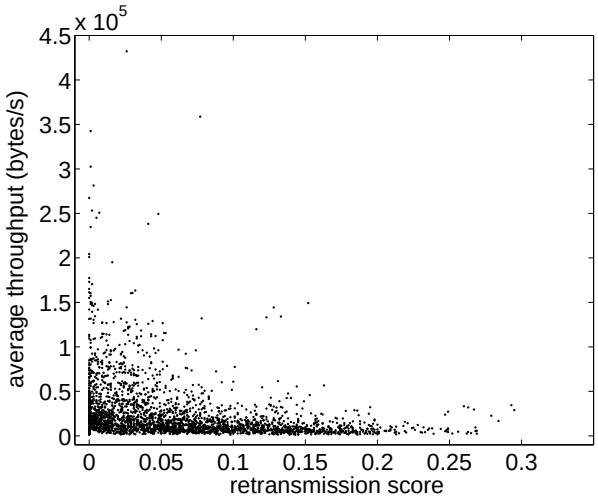


Figure 15: Retransmission score vs. throughput

their dataset support window scaling, as compared to 100% in our case.

6.4 Network Limitation

We observed surprisingly elevated levels of network limitations in our dataset as can be seen from Figure 14. In 20% of the transfer periods, at least 10% of the bytes were retransmitted. When we plot the retransmission score against achieved throughput in Figure 15, we observe that the higher the retransmission score the lower the throughput as stated by the TCP throughput formula [11]. The coefficient of correlation between the retransmission score and the throughput is -0.21.

The cdf of the estimated capacities of the narrow link is presented in Figure 16. The most dominant capacity is around 2 Mbit/s with 16% of the values (highlighted with a box). The values around 1 Gbit/s are erroneously inferred since the capacity of the access link of our measurement host was less than 1Gbit/s and may be due to ack compression. Out of the 1791 narrow links for which we were able to infer a capacity we identified only 10 potential non-shared bottlenecks, i.e. cases where the dispersion score was smaller than 0.2. As the retransmission score was high throughout the dataset and the inferred narrow link capacities fairly modest (in more than 70% of the cases below 2.5 Mbit/s), a possible explanation for high dispersion scores (see Figure 14) could be a very congested high capacity link close to our measurement host. Figure 17, which plots the retransmission score against the dispersion score for bulk transfer periods with a receiver window limitation score lower than 0.5, reveals that there is a connection between these two scores. The coefficient of correlation between them was 0.25. More precise interpretations of this phenomenon would require more information about the cross-traffic, available bandwidth, and bottleneck locations on the path and is left as future work.

We looked more closely at some of the bulk transfer periods with zero retransmission score and a high dispersion score. We found 30 to 60 seconds-long bulk transfers where TCP is in congestion avoidance and experiences very long RTTs that prevents it from growing its congestion window large enough to reach the limit set by the receiver window or available bandwidth before the end of the transfer. The connection depicted in Figure 18 had an initial RTT of $65ms$ while during the transfer period the RTT grew up to

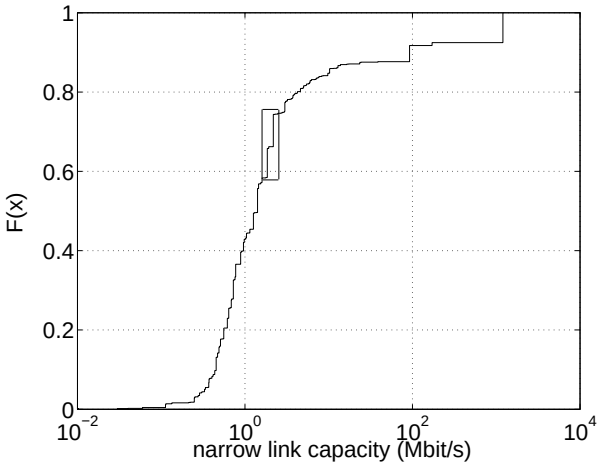


Figure 16: Cdf of inferred narrow link capacities. The box highlights the 2Mbit/s transfers.

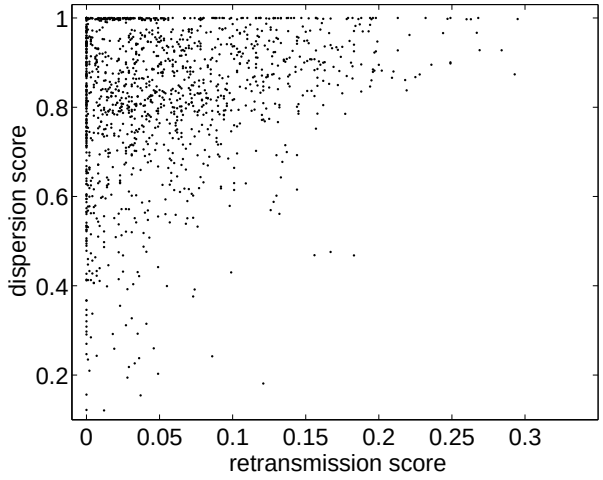


Figure 17: Retransmission score vs. dispersion score

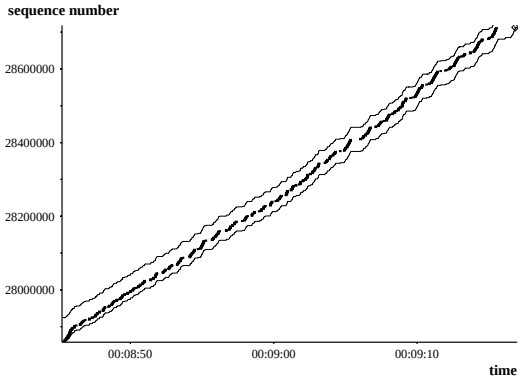


Figure 18: A complete bulk transfer period with a high dispersion score and no retransmissions.

$1.3s$. This suggests that somewhere along the path large queuing delays were introduced causing a network limitation that could not be detected by only observing retransmissions.

6.5 Exclusiveness of the Limitation Causes

We want to shed more light on the dynamics of bulk transfer periods that experience both network and receiver window limitation. Figure 19(a) reveals that even though the trend is, as expected, that a high score in one test excludes a high score in the other test, there are still quite a few transfer periods with significant scores for both limitations. We had a closer look at the three such transfer periods highlighted with a box in Figure 19(a). Two of them exhibited occasional retransmissions alternating with periods where the senders were receiver window limited as visible in Figure 19(b). Retransmissions are marked with vertical arrows with an R on top. Non-retransmitted data segments often hit the upper limit of the sliding window. In contrast, the third transfer period was receiver window limited until just before the end of the transfer where it retransmitted a large number of packets.

7. CONCLUSIONS AND OUTLOOK

In this paper, we have revisited the issue of the root cause analysis of TCP connections introduced in [18]. We have first demonstrated the weakness of the flight-based approach adopted in [18]. We have provided a thorough discussion on the different limitation causes, i.e. the application, the TCP end point parameters and the network, emphasizing the need to account for the impact of the application on the observed traffic. We then came up with a new analysis method based on various time series extracted from the headers of a packet trace. Our technique is robust and allows to precisely assess the impact of each limitation. A score representing the limitation level between 0 and 1 is provided for each of the causes (as opposed to T-RAT [18] that provided only a binary answer, yes or no, to each test). A first application of the tool on a large BitTorrent dataset has demonstrated the interest of the technique.

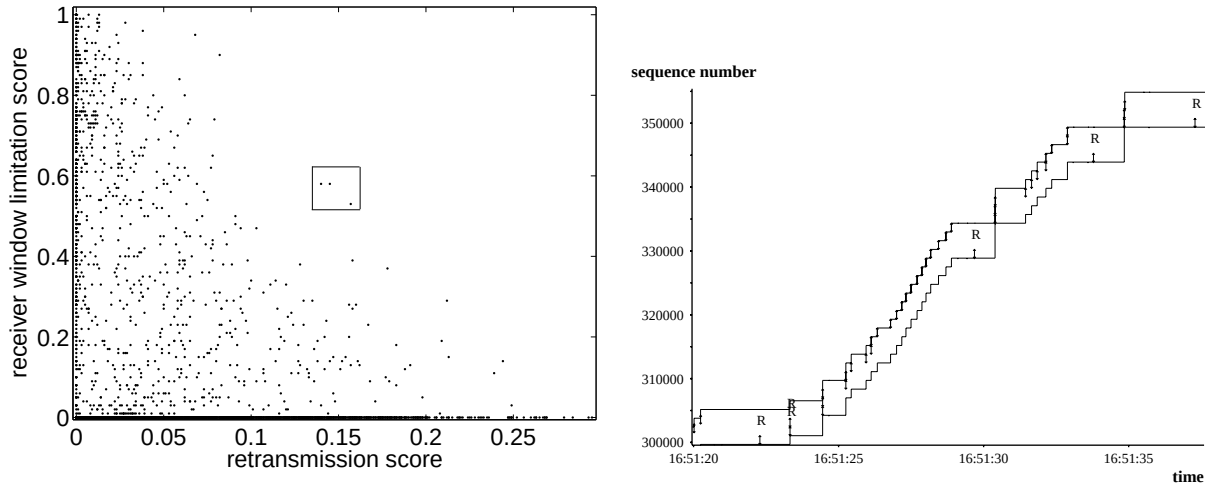
Future work includes applying our tool to publicly available traces that contain data of various applications. As explained in Section 5.1, there are currently several parameters that need to be tuned according to the application type. We are working on a reimplementation of the algorithm for bulk transfer identification that requires neither a time window nor the thresholds and could therefore work regardless of the time scale of the transfer and the type of the application. We want to analyze other applications because we believe that the ”bulk transfer applications” (P2P file transfers, FTP, scp, etc.) do not form a homogeneous class of applications but can generate many different traffic patterns due to a number of factors, e.g. application-level mechanisms, compression, and encryption, which all have an impact on how data is delivered to TCP and subsequently transferred. Another challenge in analyzing publicly available traces is that RTT estimation is still difficult in case the measurement point is not at the sender and TCP timestamps are not available [6]. We are currently investigating whether other methods for RTT estimation are suitable in this case.

We would also like to analyze the evolution of the TCP root causes of bulk transfer periods within a connection, and eventually, study in more depth the temporal dynamics of the causes and their interaction within a bulk transfer period.

Acknowledgments

This work has been partly supported by the European Union under the E-NEXT project FP6-506869 and by France Telecom, project CRE-46126878.

The authors would like to thank Mikel Izal from University of



(a) Retransmission score vs. receiver window limitation score

(b) Close up of a receiver window and network limited transfer period.

Figure 19: Network and receiver window limitation

Navarra, Spain for providing the BitTorrent dataset used in our experiments.

8. REFERENCES

[1] S. Baset and H. Schulzrinne, "An Analysis of the Skype P2P Internet Telephony Protocol", CUCS-039-04, Department of Computer Science, Columbia University, 2004.

[2] 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.

[3] S. M. Bellovin, "A technique for counting natted hosts", In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 267–272, New York, NY, USA, 2002, ACM Press.

[4] M. Carson and D. Santay, "NIST Net: a Linux-based network emulation tool", *Comput. Commun. Rev.*, 33(3):111–126, 2003.

[5] K. Claffy, R. S. Prasad, M. Murray, and C. Dovrolis, "Bandwidth Estimation: Metrics, Measurement Techniques, and Tools", *IEEE Network*, 17(6):27–35, November 2003.

[6] M. Dyrna, "Network Tomography Tools", M.S. Thesis, TU Muenchen/Eurecom, September 2005.

[7] 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.

[8] 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 2004*, April 2004.

[9] 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.

[10] M. Mathis, J. Heffner, and R. Reddy, "Web100: extended TCP instrumentation for research, education and diagnosis", *Comput. Commun. Rev.*, 33(3):69–79, 2003.

[11] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", *Comput. Commun. Rev.*, 27(3):67–82, July 1997.

[12] 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.

[13] 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.

[14] S. Shakkottai, R. Srikant, N. Brownlee, A. Broido, and kc claffy, "The RTT Distribution of TCP Flows in the Internet and its Impact on TCPbased Flow Control", , Cooperative Association for Internet Data Analysis (CAIDA), University of Illinois, 2004.

[15] B. Silverman, *Density Estimation for Statistics and Data Analysis*, CRC Press, 1986, ISBN 0412246201.

[16] 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.

[17] 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.

[18] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the Characteristics and Origins of Internet Flow Rates", In *Proceedings of ACM SIGCOMM 2002 Conference*, Pittsburgh, PA, USA, August 2002.