Institut Eurécom
Department of Corporate Communications
2229, route des Crètes
B.P. 193
06904 Sophia-Antipolis
FRANCE

Research Report RR-05-147

On the Impact of Applications on TCP Transfers

10th October 2005

M. Siekkinen, G. Urvoy-Keller, E. W. Biersack

Tel : (+33) 4 93 00 26 26
Fax : (+33) 4 93 00 26 27
Email : {siekkine,urvoy,erbi}@eurecom.fr

[1]Institut Eurécom's research is partially supported by its industrial members: Bouygues Téécom,Fondation d'entreprise Groupe Cegetel, Fondation Hasler, France Télécom, Hitachi, ST Microelectronics, Swisscom, Texas Instruments, Thales

Abstract

TCP is estimated to carry over 90% of the bytes in the Internet. Consequently, the research on TCP traffic analysis continues to be very active. When studying TCP traffic with the objective to learn about the underlying TCP/IP path properties, the effects of the application operating on top should be accounted for. This problem is often neglegted or solved only for particular cases. In this paper we focus on solving the problem for the general case, i.e. for TCP traffic generated by any type of application. The problem is hard because an application may interfere a TCP transfer on a combination of several different time scales, e.g. in the form of rate limited transfers or connections which are transferring only periodically and kept alive at other times. We argue that meaningful analysis on the network characteristics can be performed only by concentrating on TCP traffic that experiences minimal interference by the application. We call these traffic bulk data transfer periods (BTP) of a connection. We define the other traffic as application limited periods (ALP) where TCP is not fully utilizing the network resources because the application does not produce data fast enough. As an example, consider the problem of estimating the available bandwidth of a given path by observing the throughput of a TCP connection. Analysis including ALPs may give false estimates while BTPs convey correct information about the path. We present a generic algorithm that isolates BTPs from ALPs within a TCP connection and allows to quantitatively analyze the impact of the application on the TCP throughput achieved. The algorithm is validated by crosschecking the results with TCP state data extracted directly from an operating system kernel. We apply our algorithm to traffic of several particular applications extracted from public Internet traces. We show that different types of applications exhibit significantly different characteristics when studying the properties of their BTPs.

1 Introduction

The set of applications dominating the Internet has changed over the last couple of years from HTTP and FTP to peer to peer (P2P) applications. However, TCP is still transporting the majority of bytes, typically over 90%. Traffic volumes have also dramatically increased since the emergence of P2P. As a consequence, the analysis of the TCP protocol and TCP traffic is even more vital than before.

Much research have been done to detect anomalies and to characterize TCP traffic in the Internet. This analysis work usually focuses on the TCP and IP layers, but often fails to take into account the effects of the application on top. When seeking to explain certain characteristics, e.g. burstiness of TCP traffic [5] [9], it is crucial to quarantine the effects of the application before making statements concerning the TCP protocol or the TCP/IP data path.

The operational performance of an Internet application depends on several aspects. On one hand the data path, through TCP and IP layers, has a major influence on the throughput achieved, which is typically the most important performance measure. On the other hand, it is important to understand to which extent the application itself influences the throughput. Only with this information can a given data path be meaningfully studied and the performance of different Internet applications be characterized.

1.1 Our Contribution

We consider in this paper the problem of identifying bulk data transfer periods (BTP) within a TCP connection. We define a BTP as a period where the TCP sender never needs to wait for the application on top to provide data to transfer. Other time periods are defined as application limited periods (ALP). We present an algorithm to identify BTPs within a TCP connection, which is *generic* in the sense that it works *regardless of the type of application* on top of TCP. Furthermore, the algorithm enables a quantitative evaluation of the impact of the application on the throughput achieved for a given BTP. The algorithm processes bidirectional TCP/IP headers passively collected at a single measurement point. It may not always be possible to capture the traffic in both directions, e.g. in the backbone where connections may have asymmetric upstream and downstream paths [7]. Nevertheless, we argue that unidirectional traces are often not sufficient for in-depth analysis, as is the case for the round-trip time (RTT) estimation, for instance. The algorithm is validated by cross-checking its results with accurate TCP state data obtained from the operating system kernel.

We apply the algorithm to a variety of traces each of which contains traffic generated by a single application. Each trace is extracted from the same public set of traffic traces of an ADSL access network [1]. We show that the different applications have a very different impact on the underlying flow of TCP packets. We also demonstrate for the case of the RTT estimation the importance of isolating the BTPs when studying properties of a given TCP/IP path.

1.2 Related Work

Overall, the related work has mainly focused on either not addressing the problem at all or solving it for a specific case, such as for traffic of a particular application. The author in [13] suggests a method to investigate the stationarity of TCP transfers. The transfers were isolated from BitTorrent connections by excluding choked periods [6] that were identified as periods of at least 15 seconds where less than 15 kbytes of data are sent. This heuristic works well for BitTorrent traffic but can not be generalized to other applications.

In [3] the authors recommend to carefully choose the application when evaluating TCP performance.

Pioneering research work on TCP root cause analysis was done by Zhang et al. in [15] where they identify application limitation as one of the possible causes for achieving a given throughput. Our work differs from theirs by providing a method to isolate the bulk data transfers for any further analysis and to evaluate the effect of the application in a quantitative way. Additionally, we extend their definition of application limitation.

In a previous work [12] we have presented a more simple algorithm based on computing a time series with a fixed time window that was specifically designed to analyze TCP traffic generated by BitTorrent. Consequently, it uses thresholds that need to be calibrated for each type of traffic separately. In this paper we present a more general solution designed to work without calibration for any type of application.

2 The Application Impact on TCP Transfers

When the application is the root cause for the throughput achieved of a TCP connection, TCP is unable to fully utilize the network resources because the application does not produce data fast enough. There can be many reasons for this.

One is an application that produces small amounts of data at a relatively constant rate. At the TCP layer, this results in small bursts of packets, in the extreme case a single packet of size less than the allowed maximum segment size (MSS) of the connection. Typical examples are (i) live streaming applications, such as Skype [4] that transfers data over TCP at a constant rate of 32 Kbit/s (if it can not operate over UDP), and (ii) applications that impose transmission rate limits. Figure 1 shows a time vs. sequence diagram of Skype traffic. The plot was created with `tcptrace` (www.tcptrace.org). The bottom line tracks the received acknowledgments and vertical arrows sent data packets. A diamond on top of a black arrow means that the packet was pushed by TCP. The MSS is 1460 bytes but the size of each packet transferred is 42 bytes.

A second case is an application that produces data in bursts separated from each other by idle periods. An example of such behavior is Web browsing with persistent HTTP connections. The user clicks on a link to load a web page, causing a transfer period, reads the page, causing an idle period, and clicks on another

link on the same web site, causing another transfer period. Another example is BitTorrent that uses permanent TCP connections to send blocks of data during transfer periods and keep-alive packets during choked periods [6]. Figure 2 shows an example of a typical BitTorrent connection that oscillates between transfer periods ("vertical" lines) and choked periods ("horizontal" lines). The upper line tracks the receiver advertised window. Keep alive messages, visible as plain diamonds, are sent regularly during the choked periods.

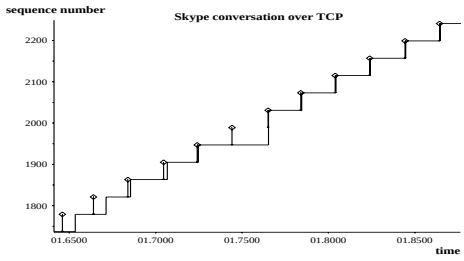

Figure 1: A short piece of Skype connection.

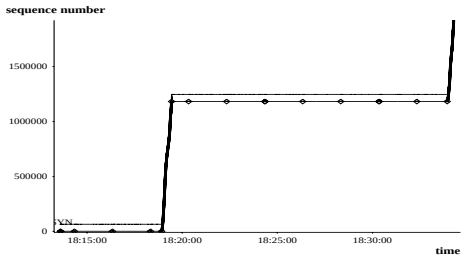

Figure 2: 20 minutes of a BitTorrent connection.

These examples demonstrate that it is challenging to design a generic algorithm to separate BTPs from ALPs since the application may interfere on very different time scales. Some cases may even include combinations of both extreme cases, as is the case of a BitTorrent connection that oscillates between choked and transfer periods where the client application enforces a transmission rate limit during the transfer periods.

3 The Isolate & Merge (IM) Algorithm

The IM algorithm identifies BTPs for a single direction of a connection at a time, since a TCP connection may have two-way data transfers. It processes only connections consisting of at least 130 data packets, because connections with fewer packets are very likely to be dominated by the TCP slow start algorithm and therefore (arguably) convey little information about the TCP/IP data path for future analysis. When in slow start mode, the TCP sender needs to transmit approximately 130 data packets (assuming a MSS of 1460 bytes) in order to reach a congestion window size equal to 64 Kbytes, the most common size of a receiver advertized window [11]. We also define a short transfer period (STP) as a transfer period that is not application limited and contains less than 130 data packets.

The IM algorithm consists of two phases: First, it partitions the connection into BTPs and STPs separated by ALPs. In the second phase the algorithm attempts to merge a BTP or STP with its adjacent BTP or STP including the ALP in between to create a new BTP. These mergers are controlled with a parameter lim that directly relates to the decrease in the BTP throughput due to a merger. There are

two main reasons for the merge phase: After the isolate phase, a connection may be divided into many BTPs and STPs separated by very short ALPs. It would be often desirable to combine these periods into one long BTP for subsequent analysis if the effect of these short ALPs on the throughput achieved is small. Also, experimenting with different values of the control parameter lim allows for quantitative analysis of the application impact on the throughput achieved as we will show in the next section. The procedures corresponding to these two phases are called *isolate* and *merge*.

3.1 Isolate

Algorithm 1: The *isolate* procedure.

```
define MSS(p_i) = {1, if p_i has size equal to MSS; 0 otherwise};
define estimate_rtt(): returns a RTT estimate for the connection;
input argument th ∈ [0, 10];
rtt := estimate_rtt();
is_active := 0, p_0 := index of first data pkt // start in inactive state
forall packets p_i ∈ {sent data pkts ∪ received acks} do
    if is_active == 1 then
        if p_i is data packet then
            sum := sum + 1;
            if (Σ_{k∈{prev 10 data pkts}} MSS(p_k)) ≤ th ||
            ( !MSS(p_0) & IAT(p_{i-1}, p_i) > RTT/2 & p_i not retransmission) then
                if sum ≥ 130 then
                    store current transfer period as BTP;
                else
                    store current transfer period as STP;
                is_active := 0, sum := 0 // start a new ALP
            p_0 := p_i;
    else
        if MSS(p_{i-2}) & MSS(p_{i-1}) & MSS(p_i) then
            store current ALP;
            is_active := 1 // start a new transfer period
```

The *isolate* procedure, sketched in Algorithm 1, scans through all the packets of a connection traversing in a given direction and continuously switches between active and inactive states. Whenever it observes packets smaller than MSS more frequently than a predefined threshold (th) allows (e.g. as in Figure 1) or encounters a long idle time preceded by a small data packet (e.g. as in Figure 2) it switches to the inactive state and stores the current transfer period. If the number of packets belonging to this period is at least 130, a BTP is stored, and a STP otherwise. When in the inactive state, all packets observed belong to an ALP until three consecutive packets of size MSS are seen. At this point the algorithm switches back to the active state and stores the ALP.

3.1.1 Obtaining an RTT estimate

For the analysis of each connection we require a RTT estimate to provide a suitable threshold for the idle periods ($IAT > \frac{RTT}{2}$ in Algorithm 1). We specify the inter-arrival time (IAT) as the time delay during which no data pkts are sent and no pure acknowledgments received – in case of a two-way data transfer there can be piggybagged acks. We consider only the IATs between a data packet following a data packet or an ack. When computing IAT between data packet following an ack, both the RTT and the location of the measurement point on the path have an influence: $IAT = (time_{data} - time_{ack}) - f \cdot RTT$, where $f \in [0, 1]$ is the "distance" of the measurement point from the TCP sender on the path and can be computed as $f = \frac{d_2}{d_1+d_2}$ from Figures 3 and 4, for example. In other words, we allow a maximum idle time of $\frac{RTT}{2}$ for the application. As the correct estimation of the RTT is important for the algorithm and far from trivial, the $estimate\_rtt()$ function in the *isolate* procedure uses one of the four following different techniques since none of the techniques alone can guarantee an estimate in all cases:

- the three-way handshake technique (a.k.a. SYN-ACK technique) [8]
- two way data/ack association technique as visualized in Figure 3.
- technique relying on TCP timestamps [14]
- technique relying on observing at least one Fast Retransmission [2], see Figure 4

If the RTT can not be estimated using any of the techniques, the connection is not processed at all. This would be the case for connections not observed from the start, that do not support TCP timestamps, transmit data purely to one direction, and experience no loss, a very rare case as we will see in Section 5.

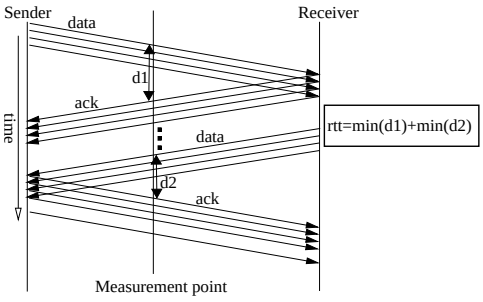

Figure 3: Round-trip time estimation using naive data-ack association.

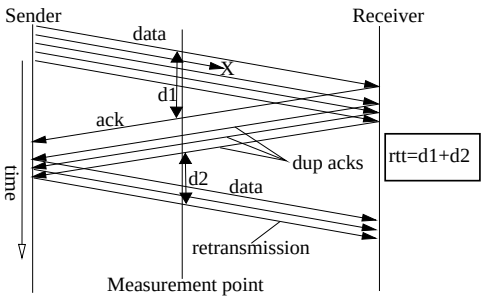

Figure 4: Round-trip time estimation using 3rd duplicate ack-retransmission association.

Algorithm 2: The *merge* procedure.

input argument $lim \in [0, 1]$;
define struct $Period := \{bytes, pkts, duration, n, type\}$;
define $merge(P_1, P_2)$: returns new $Period\{P_1.bytes + P_2.bytes, P_1.pkts + P_2.pkts, P_1.duration + P_2.duration, null, null\}$;
initialize $\forall i$, $n_i := 1$, $Vol := 0$, $i_0 :=$ index of first STP or BTP;
initialize $\mathcal{S} :=$ all periods identified by *isolate* procedure, $\mathcal{S}_{new} := \varnothing$;
initialize $P_{merge} :=$ new $Period\{0, 0, 0, 1, null\}$;
repeat
  forall *periods* $P_i \in \mathcal{S}$ **do**
    if $P_i.type$ *is STP or BTP* **then**
      $tput\_transfer := \frac{\sum_{k=i_0}^{i} P_k.bytes}{\sum_{k=i_0}^{i} P_k.duration \times P_k.n}$, $P_k.type$ is STP or BTP;
      $tput\_merged := \frac{P_{merge}.bytes + P_i.bytes}{P_{merge}.duration + P_i.duration}$;
      if $\frac{tput\_merged}{tput\_transfer} \geq lim$ **then**
        `// merger allowed`
        $P_{merge} := merge(P_{merge}, P_i)$;
        $P_{merge}.n := \frac{tput\_merged}{tput\_transfer}$;
      else
        `// merger not allowed`
        if $P_{merge}.pkts \geq 130$ **then**
          $P_{merge}.type :=$ BTP;
        else
          $P_{merge}.type :=$ STP;
        $\mathcal{S}_{new} := \mathcal{S}_{new} \cup \{P_{merge}, P_{i-1}\}$ `// add also the previous ALP`
        $P_{merge} := P_i$, $i_0 := i$;
    else if P_i *is not the first or last ALP* **then**
      $P_{merge} := merge(P_{merge}, P_i)$ `// merge with the interleaving ALP`
  $\mathcal{S} := \mathcal{S}_{new}$;
until *no more mergers*;
store $\mathcal{S}$ as the final period set with lim;

3.2 Merge

The *merge* procedure, sketched in Algorithm 2, inspects all periods identified by *isolate* and attempts to merge adjacent periods while respecting the given lim parameter value. lim indicates the maximum tolerated level of decrease in the throughput when merging periods to form a new longer BTP. $tput\_merged$ is the throughput of the newly formed period by merger, whereas $tput\_transfer$ is the throughput obtained when combining together only the BTPs and STPs contained by the merged period.

Note that the throughput of a transfer, especially a long one, is not always stable throughout its lifetime. Consider, for instance, a case where a BTP experiences congestion that degrades its throughput and the next BTP does not. Thus, in these cases a merger may be prevented due to different throughputs achieved by the subsequent transfer periods, which is, in fact, desireable in many cases since they exhibit clearly different network conditions.

The outermost loop in Algorithm 2 is required to ensure that eventually all al-

lowed mergers are executed. Consider this likely scenario: A STP is not allowed to merge with the next STP because the n value after a merger would be just below lim but the latter STP is allowed to merge with the next BTP. However, on the second round the first STP is allowed to merge with the already merged BTP because the BTP weights much more than the STP in the computation of $tput\_transfer$. Consequently, care must be taken when updating the $tput\_transfer$ value to take into account the decrease due to mergers performed on previous rounds – hence the division by $P_k.n$ when updating $tput\_transfer$ in Algorithm 2. Finally, a merger is never started or ended with an ALP.

4 Validation

Validation of our algorithm was done using the Web100 software [10] that allows querying of the state variables of active TCP connections locally on a Web100-enabled machine. We generated traffic with a BitTorrent client by seeding, i.e. uploading to other clients, a very large and popular torrent. We recorded packet headers with `tcpdump` and simultaneously sampled the current state of application write buffer for all active connections with Web100. This Web100 variable gives the current number of bytes of application data buffered by TCP and that is pending first transmission. We ran our algorithm on the collected `tcpdump` data and compared the results to those from Web100.

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

We analyzed 99 connections amounting to 11.7 GB of transfer in total. Each connection was broken on the average into 279 periods by our algorithm. Figure 5 shows a CDF of d for these connections. The periods identified by the Algorithm 1, i.e. those with $lim = 1$, were used. Overall the match is very good but not perfect. Indeed, we can identify several sources of error due to the way Web100 is operated: The ability to query the Web100 variables and to write the data on the disk is limited by the speed of CPU and disk I/O. We were able to achieve a maximum sampling rate of approximately 67 samples/s, i.e. one sample each 15 milliseconds, during the experiments. As the changing rate of the measured variable is in reality limited only by the arrival rate of packets, that can be easily ten times higher than our sampling rate, we failed to capture all dynamics with Web100. Consequently, as Figure 5 shows, the coarser the granularity, the better

the accordance because of smoothing of the smallest time-scale variance in both time series. Means of d are 0.9713, 0.9570, and 0.9517 for time windows 1000, 100, and 10 milliseconds, respectively. In addition, the timestamps for the two data sources might not be 100% in sync because we used plain commodity hardware.

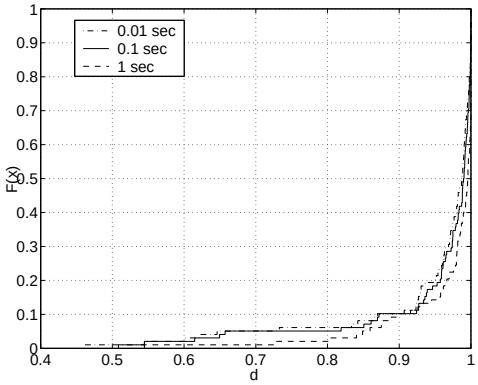

Figure 5: CDF of d for the periods with $lim = 1$.

5 Per-Application Analysis

We applied the IM algorithm to eight different traffic traces each containing traffic from a specific application. All of the application specific traces were extracted from the same original public ADSL access network traces (the first 19 days from Location 4 traces in [1]) by filtering on the well-known TCP port numbers of these applications. This gives in most of the cases solely the traffic from the expected application except for some cases where well-known TCP ports are used, for example, by P2P applications to bypass firewalls. In the following, we will only highlight the most interesting observations that our analysis has brought up.

Table 1 summarizes the characteristics of the traces. Only one connection in the traces could not be analyzed due to the lack of a RTT estimate. We used $th = 3$ with Algorithm 1. Regardless of the application type, BTPs were found only in a small fraction of the connections. However, BTPs generally carry the majority of the bytes. The average size of the connections including no BTPs was below 30KB for all applications except for FTP which had an average of 220KB. Oddly enough, the largest ones of these FTP connections, carrying up to 90MB, appeared clearly to be rate limited by the application sending constantly small packets. These unexpected examples clearly emphasize the need to identify the BTPs even for "bulk transfer applications" such as FTP.

Table 1: Trace characteristics.

traffic type	BitTorrent	eDonkey	FTP data	SSH	Gnutella	HTTP(S)	FastTrack	WinMX
port numbers	6881-6889	4661,4662	20	22	6346,6347	80,443	1214	6699
duration	4d 22h	4d 22h	18d 22h	18d 22h	18d 22h	4d 22h	18d 22h	18d 22h
packets	31M	44M	9M	1M	8M	14M	20M	13M
bytes	19GB	20GB	7GB	978MB	2GB	9GB	14GB	5GB
cnxs	150K	1.6M	5.9K	4.2K	410K	590K	360K	6.3K
cnxs with BTPs	12K	7.5K	430	180	1.2K	3.3K	5.9K	720
bytes in BTPs (lim=1)	3.5GB	2.9GB	4.7GB	630MB	700MB	4.8GB	5.2GB	490MB
bytes in BTPs (lim=0.9)	9.0GB	7.4GB	5.9GB	720MB	1.3GB	5.1GB	10GB	1.6GB
avg BTP size (lim=1)	780KB	460KB	3.3MB	2.5MB	600KB	1.6MB	700KB	480KB
avg BTP size (lim=0.9)	590KB	1.1MB	13.3MB	8.4MB	1.2MB	1.8MB	1.6MB	2.6MB
avg BTP dur. (lim=1)	45s	2m 33s	1m 2s	1m 17s	1m 5s	41s	1m 54s	1m 50s
avg BTP dur. (lim=0.9)	1m 50s	6m 12s	4m 57s	5m 9s	3m 7s	59s	4m 56s	8m 23s

5.1 Properties of the Identified BTPs

Figures 6 and 7 show for some selected applications the number of identified BTPs and the fraction of all bytes carried by them, respectively, as a function of the threshold parameter lim. The number of identified BTPs varies differently with the lim value. The reason is that we require a BTP to contain at least 130 data packets. Thus, sometimes new BTPs are formed by merging together only STPs when lim value is decreased, which increases the total BTP count. Indeed, we checked that for all the applications the total count of BTPs and STPs together is always increasing when increasing lim value. Similarily in Figure 7, all the byte ratios are increasing when decreasing the lim value because mergers always bring more bytes into BTPs. eDonkey clients often limit the maximum upload rate and we observe that majority of the bytes are in ALPs. Also BitTorrent clients often throttle back their upload rate. However, as Figures 9 and 10 show, eDonkey and BitTorrent clients seem to implement rate limitation differently. While eDonkey clients regulate the rate by varying constantly the amount of bytes passed to TCP (packet sizes in Figure 9 are systematically below MSS), BitTorrent clients give data in larger blocks and stay idle between the transfers of two blocks. The effect is also visible as different shapes of curves in Figure 7: The small chunks of data sent by a BitTorrent client generate STPs that are then merged together as BTPs when lim value is low enough while the small packets sent by eDonkey client generate connections consisting only of ALPs. In contrast, with FTP the vast majority of bytes are carried by BTPs and the byte ratio in Figure 7 is quite insensitive to the lim value indicating that most connections compose of only a single BTP. In other words, FTP exhibits the purest form of "bulk transfer application". This is confirmed by Figure 8, which shows the number of BTPs per connection. BitTorrent connections are commonly broken into many BTPs due to the oscillation between the choked and unchoked state of the protocol.

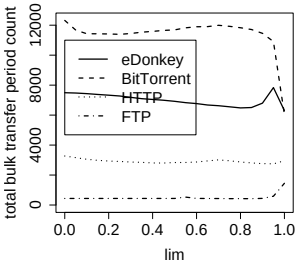

Figure 6: Number of identified BTPs vs. lim.

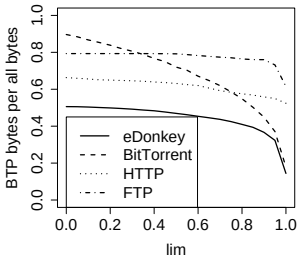

Figure 7: Fraction of all bytes in BTPs vs. lim.

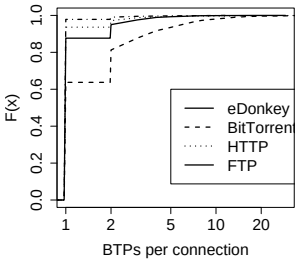

Figure 8: Number of identified BTPs, $lim = 0.9$.

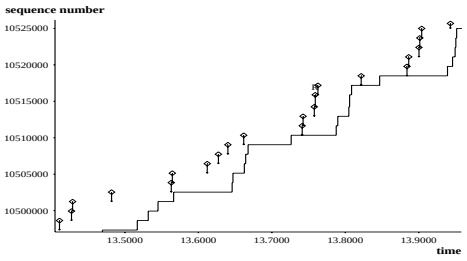

Figure 9: Rate limited eDonkey connection.

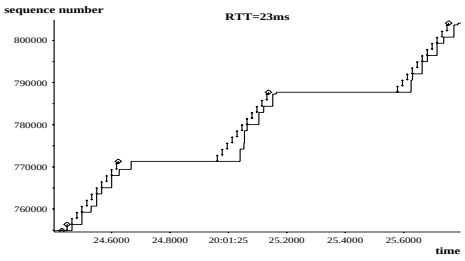

Figure 10: Rate limited BitTorrent connection.

5.2 Comparing BTPs to Entire Connections

We investigated also the relation between the durations of BTPs and the bytes carried by them with respect to the duration and volume of the entire connection. For the connections with at least one BTP, we computed ratios of durations ($\frac{\sum BTP\_duration}{cnx\_duration}$), volumes ($\frac{\sum BTP\_bytes}{cnx\_bytes}$), and throughputs ($\frac{\left(\frac{cnx\_bytes}{cnx\_duration}\right)}{\left(\frac{\sum BTP\_bytes}{\sum BTP\_duration}\right)}$) between BTPs (with $lim = 0.9$) and the entire connections shown in Figures 11, 12, and 13. BitTorrent connections often experience long choke periods that are identified as ALPs, which explains the small duration ratio observed for most connections. Note that Figures 11, 12, and 13 do not include the connections with no BTPs at all. That is why eDonkey has relatively few connections with low volume ratio – the rate limited eDonkey connections typically consist only of ALPs. There are more BitTorrent connections with small volume ratio than with eDonkey, which is again due to the difference in the way they limit the transmission rate. BTPs of HTTP traffic contain the vast majority of the bytes of a connection but can be short when compared to the whole connection. Web surfing with persistent HTTP connections would explain this phenomenon (see Section 2). Interestingly enough, Figure 13 reveals the dual nature of SSH traffic. On one hand, *secure copy (scp)* runs over SSH and similarily any application (e.g. X server application) may be

tunneled through a SSH pipe. Establishing a SSH connection requires authentication, key exchange, and negotiation of a set of parameters, which commonly produces an ALP of a few seconds. That is why scp, with small and moderate size files generate BTPs preceded by ALPs. Similarily, some ssh-tunneled applications which update the screen periodically (e.g. web browser when loading a new page) generate BTPs separated with ALPs. Hence, the connections with low throughput ratios. On the other hand, the high throughput ratios may be caused by other more interactive applications (e.g. telnet) that are run over SSH and generate low rate BTPs, while they generally consist entirely of ALPs. Indeed, the coefficient of correlation between BTP throughput and throughput ratio for SSH traffic is -0.43.

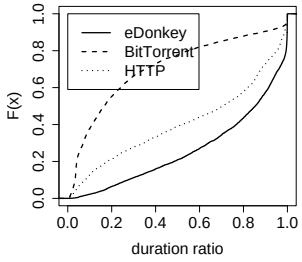

Figure 11: CDF of duration ratio

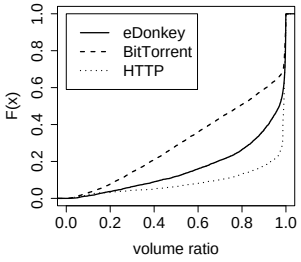

Figure 12: CDF of volume ratio

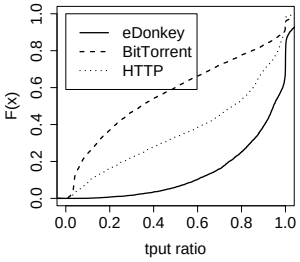

Figure 13: CDF of tput ratio

5.3 BTPs vs. ALPs in the analysis of a TCP/IP path

We have argued that it is important to focus on BTPs when studying network characteristics. In general, BTPs manifest the TCP/IP path properties in a very different way than do the entire connections, which is illustrated in Figure 13. As a concrete example, we estimated RTT samples of all connections having at least one BTP. To compute the RTT samples we used either the technique relying on TCP timestamps [14] or the technique from [7] when TCP timestamps were not available. We computed for each connection averages and coefficients of variation ($CoV = \frac{stddev}{mean}$) for two sets of estimated RTT samples: first, including only all the BTPs and second, including only all the ALPs. We used $lim = 0.9$. Table 2 shows how the relative differences for the average RTTs ($d_{avg} = \frac{avg(RTT_{BTP}) - avg(RTT_{ALP})}{min(avg(RTT_{BTP}), avg(RTT_{ALP}))}$) and differences for the CoVs ($d_{CoV} = CoV_{BTP} - CoV_{ALP}$) distribute. We can make two main observations concerning the average RTTs. First, the differences are striking: For instance, more than 20% of the eDonkey connections have ten times shorter or longer average RTT during ALPs than during BTPs. A possible explanation for the inflated RTT values during the ALPs may be that when an ack is assumed to trigger a sending of data packet, TCP may delay the sending of a data packet because it has no more data to send at the arrival time of the ack, thus, generating a longer RTT sample (the problem is

acknowledged in Section 4.1 in [14]). Second, the results vary significantly from one application to another. In order to fully explain these findings requires further investigation and is therefore left as future work. As for the CoV values, we observe systematically less RTT variation during the ALPs than during the BTPs for all the applications. Larger RTT variation during BTPs could originate from the fact that the higher transmission rates of the BTPs compared to the ALPs (refer to Figure 13) may cause their data packets to experience varying queuing delays with a higher probability. Again, we leave further investigation as future work.

Table 2: Differences in RTT estimates during BTPs and ALPs.

traffic type	d_{avg} <-10	d_{avg} <-1	d_{avg} <-0.1	d_{avg} >0	d_{avg} >0.1	d_{avg} >1	d_{avg} >10
eDonkey	14%	37%	44%	54%	50%	35%	7.5%
BitTorrent	0.4%	24%	49%	42%	36%	14%	0.9%
HTTP(S)	0%	2.6%	16%	81%	66%	37%	2.6%
traffic type	d_{CoV} <-10	d_{CoV} <-1	d_{CoV} <-0.1	d_{CoV} >0	d_{CoV} >0.1	d_{CoV} >1	d_{CoV} >10
eDonkey	0%	2.0%	11%	86%	80%	27%	0%
BitTorrent	0%	3.1%	25%	67%	59%	13%	0%
HTTP(S)	0%	0%	0.9%	97%	89%	24%	0%

6 Conclusions

Much of the research done on TCP traffic analysis neglects the effects of the application operating on top. We argue that when trying to characterize the behavior of a TCP/IP data path, the impact of the application must be accounted for. We focused in this paper on the problem of identifying bulk transfer periods, i.e. periods of traffic where the application impact on the TCP throughput is minimal, within a TCP connection. We provided an algorithm that isolates these BTPs from the rest of the traffic and, moreover, allows to quantify the impact of the application on the throughput perceived for a TCP connection. We then applied the algorithm to a variety of different application traffic extracted from public Internet traces and showed that different applications exhibit clearly different characteristics when studying their BTPs. We also showed the impact of the application operating on top of TCP when studying the TCP/IP path properties through an example use case on RTT estimation.

As future work we are going to extend our work on the root cause analysis of TCP throughput [12] with the help of the IM algorithm to include all types of application traffic. We would also like to investigate some of the TCP traffic properties, such as burstiness [5] [9], in order to quantify the impact of applications on these properties. Finally, we also want to examine the many connections that contain no BTPs.

Acknowledgments

This work has been partly supported by France Telecom, project CRE-46126878.

References

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

[2] "RFC 2001: ftp://ftp.rfc-editor.org/in-notes/rfc2001.txt".

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

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

[5] E. Blanton and M. Allman, "On the Impact of Bursting on TCP Performance", In *Proceedings of Passive and Active Measurements (PAM)*, 2005.

[6] B. Cohen, "Incentives to Build Robustness in BitTorrent", Technical Report, http://bitconjurer.org/BitTorrent/bittorrentecon.pdf, May 2003.

[7] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP Connections Characteristics from Passive Measurements", In *Proc. Infocom 2004*, March 2004.

[8] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times", *Comput. Commun. Rev.*, 32(3):75–88, 2002.

[9] H. Jiang and C. Dovrolis, "Source-level IP packet bursts: causes and effects", In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pp. 301–306, New York, NY, USA, 2003, ACM Press.

[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] 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.

[12] M. Siekkinen, G. Urvoy-Keller, E. Biersack, and T. En-Najjary, "Root Cause Analysis for Long-Lived TCP Connections", In *Proceedings of CoNEXT*, October 2005.

[13] G. Urvoy-Keller, "On the Stationarity of TCP Bulk Data Transfers", In *Passive and Active Measurements 2005*, March 2005.

[14] 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.

[15] 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.