The RTT Distribution of TCP Flows in the Internet and its Impact on TCP-based Flow Control

Srinivas Shakkottai, R. Srikant
University of Illinois
sshakkot,rsrikant@uiuc.edu

Nevil Brownlee, Andre Broido, kc claffy
CAIDA, SDSC
nevil,broido,kc@caida.org

ABSTRACT

It is well known that the performance of a TCP flow is affected by its round-trip time (RTT), i.e., the elapsed time between the instant a packet is released by the source to the instant the corresponding ack is received by the source. The distribution of RTTs can dramatically affect not only the data rates realized by individual flows sharing a link but also the utilization of Internet links. In this paper we present a comprehensive study of the RTT distribution for TCP traffic in the Internet.

Because access to measurement points is so limited, one typically has access to a node that may be neither the source nor the destination of most of the TCP flows passing through it. Thus, our primary challenge is to infer the RTT distribution by looking at traffic in only one direction (either source-to-destination or destination-to-source, but not always both since the ack (reverse) path may differ from the data (forward) path). We present three different methods of estimating RTT and show that all three methods provide a consistent description of the RTT distribution. Further, we use this data to validate the use of fluid models that have grown pervasive in TCP analysis. Finally we make a study of burstiness of TCP flows.

Keywords

RTT, Controllability, TCP Dynamics, Fluid Models

1. INTRODUCTION

The round trip time (RTT) seen by a TCP flow is defined as the total time between a sender transmitting a packet and the reception of its corresponding *ack* packet. This interval includes propagation, queuing, processing and other delays at routers and end hosts. A TCP flow's throughput is inversely proportional to its RTT, as described in [1]. Recently, the study of TCP dynamics has been enhanced by the development of fluid models [2, 3, 4, 5]. A common theme of these models is the analysis of the stability of various TCP and active queue management (AQM) schemes in the presence of delayed congestion feedback from the network. These studies conclude that TCP and TCP-like control mechanisms are stable so long as the TCP and AQM parameters are inversely proportional to RTT. More recently, there has been work on protocols such as XCP which outperform TCP even in high bandwidth-delay product cases[6].

With fluid models, one analyzes stability by assuming flows are of infinite duration. The rationale for this assumption is the now accepted fact that most bytes are carried in large flows, a consequence of file-size distributions that exhibit a large (sometimes infinite in heavy-tailed models) variance. However, what matters for controllability is not the flow size but flow duration, which must be long enough to allow the source to obtain sufficient feedback from the network. Since the RTT, as reflected by the arrival of the *ack* back to the source, represents the feedback signal, the important parameter to model is the number of RTTs rather than the absolute size or duration of the flow. In other words, if the duration of a flow is large compared to its RTT, then it gets sufficient feedback from the network and is amenable to the fluid modeling assumption. Another common assumption in the analysis of fluid models is that a flow's RTT is constant throughout the lifetime of the flow. While some models explicitly account for queuing delay variations during the flow lifetime and some recent studies measure the variability of RTT [7], typically the analysis of such models use some constant-RTT approximations to allow application of stability results from linear control theory [8, 9, 10].

Since the dynamics of TCP flows, in terms of performance and stability, are intimately related to their RTTs, accurate methods to measure RTT are essential. It is also important to verify the two assumptions regarding controllability of flows and invariance of RTT in time scales of flow durations. Several researchers have analyzed RTT measurements [11, 12, 13, 14, 15, 16], although typically based on sampling a TCP session at the beginning of a flow and not throughout its duration. The completeness of these results remains in question [17].

Our first challenge is to obtain reliable estimates of RTT. Broadly speaking we may divide TCP flows into *download flows* carrying bulk data and *feedback flows* which are sequences of *ack* packets. We will formally define these two types of flows in a later section. Figure 1 depicts a TCP download flow between hosts X and Y, and corresponding

feedback flow from Y to X. The RTT observed from X to Y is the total time between sending a packet from X and the reception of its *ack* back at X.

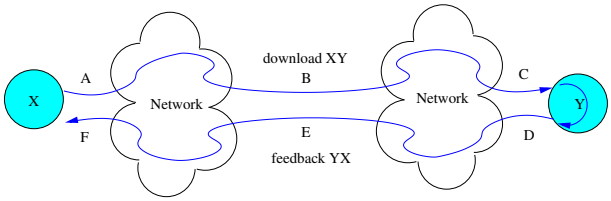


Figure 1: Possible positions of instruments for RTT measurement

The obvious method to measure RTT, as used in [7], is to match the sequence number of a data packet and its corresponding *ack*, and subtract the time stamps, If we receive only one *ack* for every two packets, we would obtain one RTT value for every two packets observed. Overcoming this limitation requires instrumentation at points A and F in Figure 1, which is difficult except in special cases. One case where such instrumentation may be possible is at the access gateway of an organization with negligible internal delays. In order to capture as diverse a data set as possible, our main data sets are from backbone links. These backbone monitor points correspond to points B and E in the figure, which means that sequence matching would yield timing information only for the path $BCDE$. Another problem is that the *acks* may not be seen in the EF direction or indeed, observed at all by the monitor. Since sequence matching under these circumstances is questionable, we do not use this method. Our challenge is to obtain reliable RTT estimates using **unidirectional data**, where we rely on packets observed on a single directional fiber. We will use three different methods of estimating RTT using passive measurements. The first is the well-known *syn-ack* method used in many of the studies mentioned above. The other two methods view TCP flows on a longer time frame using fluid and packet based approaches.

We organize the paper as follows. We first describe the data collection method and types of flows observed. Section 3 considers TCP at two different scales, fluid and packet. This distinction suggests several natural methods for measuring round trip times; we compare three of them in the next section. The three estimates are obtained at three different phases of the TCP flow: one at the beginning, one at the slow start phase and another during the congestion avoidance phase.

Once we have a reliable estimate of RTT, we may use it to generate relevant statistics. The first is to find the RTT distributions experienced by flows in the different data sets. We then show the average RTT in the three phases are very similar and conclude that RTT and hence congestion level on the Internet is a slowly varying quantity. §5 deals with the controllability of flows and time scale separation between long and short flows. We then study burstiness of flows in §6. We conclude with some pointers on how to extend our work.

2. MEASUREMENT METHODS AND DATA TYPES

All data analyzed in this paper was passively collected on different links in the USA. Figure 2 depicts our data collection setup. The monitor captures data carried on two optical fibers, one for each direction of the link. A fiber splitter divides the light on each fiber, siphoning 10% of the signal off to a *DAG 4.11* NIC card [18] in our monitor PC.

The Endace DAG cards have a clock resolution of 15 ns, GPS-synchronized to UTC once per second. The DAG cards record a time stamp when they observe the first byte of a packet on the fiber. When the packet has traversed the link, the card transfers its header bytes and the time stamp into PC memory. Our monitor PCs have two 2.4 GHz Intel Xeon processors and two large, fast RAID disk arrays so that they can run two separate header collection tasks in parallel.

We collect traces using the *dagsnap* program from the dagtools package [19]. Output data from *dagsnap* is piped through *dagsplit*, which breaks it into separate gzipped files at specified intervals. A typical hour of gzipped trace data from an OC-48 (nominal bit rate 2.5 Gb/s) link with about 625 Mb/s (25% utilization) occupies about 15 GB of disk space.

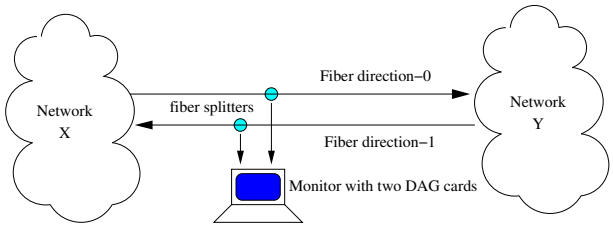


Figure 2: Illustration of data collection

2.1 Different Types of Flows

We refer to a *flow* as a transfer of packets between two ports of a source-destination pair using a particular protocol. A flow is thus uniquely identified by the five-tuple consisting of the source-dest IP addresses, the source-dest ports and the protocol used [20]. Instead of a timeout-based delineation of flow boundaries, we look at all packets associated with a particular five-tuple within the interval of measurement. Each DAG card sees all flows on a particular fiber. A TCP connection is bidirectional and consists of two flows, one in each direction [21]. A TCP connection is initiated by a host (caller) sending a *syn* packet, which generates a response from the end host (callee) with a *syn-ack* packet. The caller responds with an *ack* packet which completes the handshake. Asymmetric routing renders it invalid to assume that a flow will appear on both directions of an observed link. Thus, for flows belonging to a particular TCP connection, data packets seen on fiber-0 may not receive acknowledgments on fiber-1.

We divide flows into five major categories:

- **TCP-Download Flows (Type-1)**. The callee might be in Network X. In direction-0 we observe a *syn-ack*

packet followed by *data* packets. These flows each carry large amounts of data and together represent a large fraction of overall byte traffic; examples include peer-to-peer file transfers, FTP, and web browsing. Figure 3(a) illustrates TCP download flows.

- **TCP-Feedback Flows (Type-2)** The caller may be in Network X with the callee in Network Y. In direction-0 we observe a TCP flow whose first packet is a *syn* followed by a sequence of responding *ack* packets from the callee, as shown in figure 3(b). Typically the *ack* packets are feedback provided to the callee. However, in some cases such as *telnet* or chat sessions, the traffic in this responding direction actually carries data. We distinguish **data-feedback flows** as those YX flows for which the total bits transferred in direction XY (as seen by the range of sequence numbers) is larger than the bits transferred in the direction YX (as seen by the range of acknowledgment numbers).
- **TCP-Unknown flows (Type-3)**. We observe TCP flows that do not fit into the two above types because they begin outside the interval of measurement so we never observe their leading *syn* or *syn-ack*.

If a TCP-download flow has source-port 80 (HTTP) we call it a **Web** flow. Flows that appear to be TCP-feedback flows may in fact be scan traffic incurred by hackers, which does not involve acknowledgment of a data transfer from the other end. We refer to TCP-feedback flows that are less than four packets in length as **small** and those larger than (or equal) to four as **normal**.

The other two major types of flows are:

- **UDP flows (Type-4)**, often associated with peer-to-peer searches [22], DNS lookups and real-time streaming.
- **Other flows (Type-5)**, using any protocol other than TCP or UDP.

2.2 Data Sets

We use three different packet traces in this paper, all from OC-48 ($\approx 2.5Gb/s$) links:

- **BB1-2002** was collected on a backbone link of a major ISP on Wednesday, 14 August 2002. The data was collected from 10-11 AM local time on a 25% utilization link connecting Seattle to San Jose. This data set is not anonymized, hence we can use the prefixes to identify host locations.
- **BB2-2003** was collected on a backbone link of a different major ISP on Wednesday, 7 May 2003. The (also not anonymized) data set was collected from 10-11 AM local time on a 26% utilization link connecting San Jose to Seattle.
- **Abilene-2002** was collected on a link in the Abilene research network on Wednesday, 14 August 2002. The data set was collected from 10-11 AM local time on an 25% utilization eastbound link between Kansas City and Cleveland, Ohio. This data set is anonymized so we cannot tell anything about the hosts. It is available at [23].

Besides these packet traces, we also obtained a NeTraMet flow data file which we will refer to as **UNI-2002**. A NeTraMet monitor [24, 25] generates flow summaries in real time using packet header data from DAG cards. We configured NeTraMet to observe both directions at the same time and match data packets with corresponding *acks* by matching sequence numbers. NeTraMet was monitoring a research university gateway on 21 August 2002. At the time the university's Internet connection was a single OC-12 ($622Mb/s$) link and so we were in a position to observe all traffic to and from the university. This data set does not have per packet information, but due to the unique location of the monitor has precise timing information which we use to validate our analysis of the other data sets.

Together these data sets represent a high diversity of IP addresses, applications, geographic locations and access types. For instance, the *BB1-2002* data set shows about 30% of bytes destined for locations in Asia, with flows destined to about 30% of all global autonomous systems (AS). The *BB2-2203* also has a fairly high diversity with flows from about 24% of all global ASs. The *Abilene-2002* data has a large fraction of non-web traffic. Figure 4 shows the breakdown of traffic bytes into different flow types. As expected, TCP flows dominate. But whereas the commercial links show a large fraction of *download flows*, this category is less significant in the research network, which also shows a large fraction of *feedback flows*.

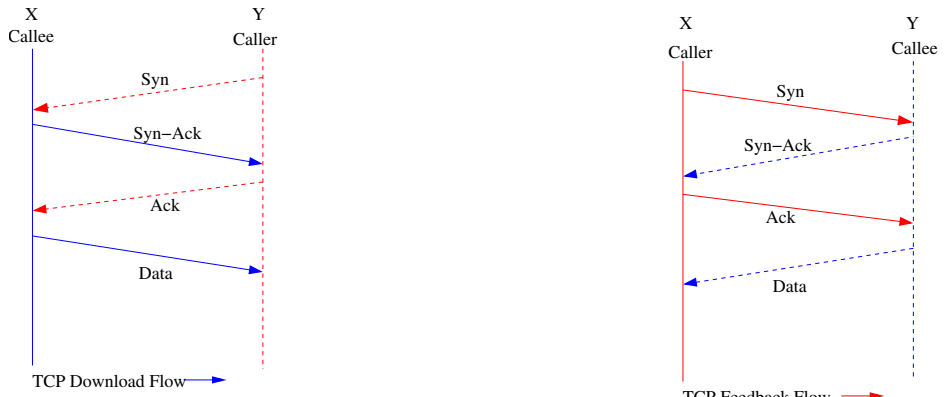
3. TCP BEHAVIOR AT THE FLUID AND PACKET SCALE

We provide here a short description of TCP and two different ways of looking at its behavior. These two approaches lead to different tools that we use later in the paper.

3.1 The Fluid View of TCP

TCP is a window-based protocol with two modes of operation. The windows are calculated in terms of Maximum Segment Size (MSS). MSS indicates the maximum number of data bytes any particular layer-2 technology allows per packet. TCP/IP headers are later added to the packet (normally 40 bytes) and the resulting total is called the Maximum Transmission Unit (MTU). The congestion window size at any time indicates the number of bytes "on the wire", i.e the number of bytes that have been packetized and sent out (assuming that there is data to send and the receive window is large enough to accept it) but for which *acks* have not been received. In *slow start mode*, the congestion window size increases by one MSS per received *ack* which results in its doubling once every RTT. In the *congestion avoidance mode* the window size increases by one MSS every RTT. When a packet is lost, the window size is halved. TCP thus uses an additive increase, multiplicative decrease algorithm for congestion control.

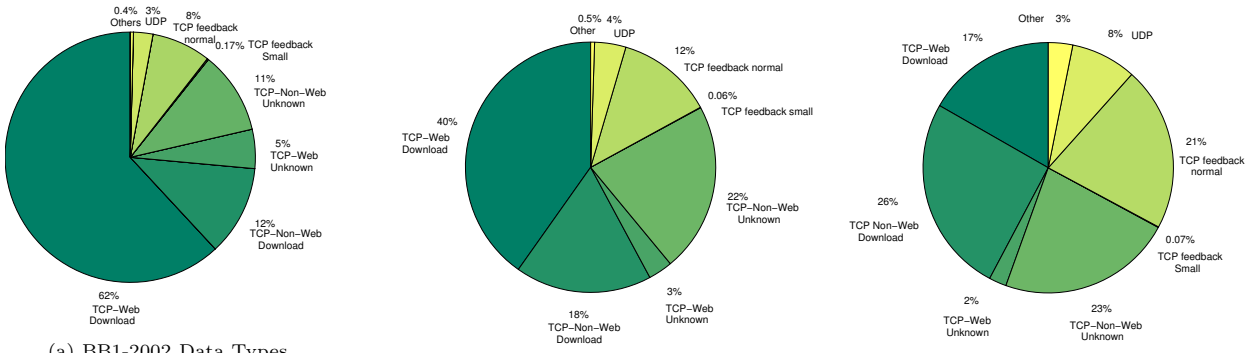
At a coarse time scale we can model TCP as a rate-based algorithm. *Rate* is an average quantity – the number of



(a) TCP-Download Flow

(b) TCP-Feedback Flow

Figure 3: Types of TCP flows. Solid lines represent observed packets.



(a) BB1-2002 Data Types

(b) BB2-2003 Data Types

(c) Abilene-2002 Data Types

Figure 4: Byte percentages of different flow types on three OC-48 data sets. The Abilene profile is dramatically differs in feedback vs. download flow types.

bits transferred over some interval of time – and packets are visualized as a fluid of bits on the wire. Figure 5(a) shows the rate behavior of TCP, i.e., the exponential rate increase in slow start mode and the sawtooth behavior caused by packet drops in congestion avoidance mode. The fluid model has been used extensively [3, 26, 9, 5]. Since we do not know the RTT, it is difficult to divide the flow into the two modes. This paper concerns primarily the congestion avoidance mode, so we assume a fixed threshold above which the flow has entered congestion avoidance.

PROPOSITION 1. *In the congestion avoidance phase of TCP, the time instant $\hat{t}$ at which the average rate in an interval $[t_0, t_1]$ (during which no drops occur) is the same as the instantaneous rate is $\hat{t} = \frac{t_1 - t_0}{2}$.*

PROOF. Consider a time interval $[t_0, t_1]$ in congestion avoidance mode during which no drops occur and $t \in [t_0, t_1]$. Let $x(t)$ be the number of bits transferred in the time interval $[t_0, t]$. Then we have for some constant k

$$\frac{d^2x}{dt^2} = k. \tag{1}$$

The rate at time t is then given by

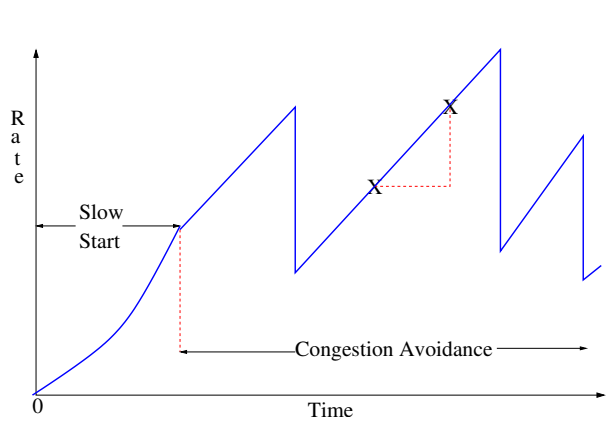
$$\frac{dx}{dt} = r_0 + k(t - t_0), \tag{2}$$

where r_0 is the rate at time t_0. At $t = \frac{t_1 + t_0}{2}$ (the middle of the interval),

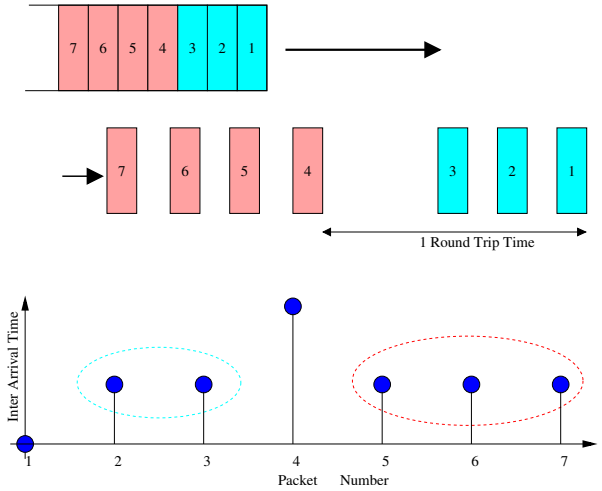
$$\frac{dx}{dt} = r_0 + \frac{k(t_1 - t_0)}{2}. \tag{3}$$

Also we have

$$x = r_0(t - t_0) + \frac{k(t - t_0)^2}{2}. \tag{4}$$



(a) Rate based (fluid) view of TCP



(b) Flight based (packet) view of TCP

Figure 5: Two different views of TCP. In (b) note the inter-arrival time vs. packet number graph. Packets $1-3$ are grouped together as a 2-inter-arrival time unit flight, and so on. The large gap between packets 3 and 4 appears as a singleton.

In particular the total number of bits transferred in the interval $(t_1 - t_0)$ is

$$x = r_0(t_1 - t_0) + \frac{k(t_1 - t_0)^2}{2}. \quad (5)$$

Dividing both sides of the above by $t_1 - t_0$, the average rate during the interval $[t_0, t_1]$ is

$$r_0 + \frac{k(t_1 - t_0)}{2}, \quad (6)$$

which is identical to equation 3. □

Our algorithm for estimating the rate over an interval, is to count the number of bits transferred by ten packets (including the headers) and divide by the time taken for the transfer. As derived above, the time instant at which this rate occurs is the middle of the interval. We call the pair consisting of a time instant and a rate value as a *rate point*. We assume that the flow is in congestion avoidance and that no drops occur while bits are being counted. We ensure this constraint by finding rate points only for sequences of more packets than our threshold. These requirements imply that we can find rate points only for flows with relatively many packets, and even for these we will not obtain many rate points. In the next section we show that although stringent, our requirements allow inferences about a large byte percentage of the traffic. TCP download flows lend themselves well to fluidization as they carry many bits. For feedback-flows, we only fluidize those with significant amounts of data, i.e TCP feedback flows for which the range of sequence numbers is greater than the range of acknowledgment numbers.

3.2 The Packet View and the Controversy regarding Flights

We now consider the steady state characteristics of TCP at a packet level and investigate whether TCP flows have recognizable fine structure that we can label *flight behavior*. We define a *flight* as a sequence of consecutive packets with nearly identical inter-arrival times (IAT) followed by a larger IAT. A TCP flow containing flights would consist of a flight of packets followed by a gap, then another flight of packets, and so on. This structure could arise due to TCP's window-based congestion control. Figure 5(b) depicts a window of packets sent off back-to-back, followed by a pause before acknowledgments arrive.

The phrase *nearly identical inter-arrival times* is defined by means of a threshold. Consider a sequence of packets p_1, p_2, p_3, with IATs δ_1 and δ_2 between the first and second pairs of packets, respectively. Then we consider the ratio $g = |\frac{\delta_2 - \delta_1}{\delta_1}|$. We decide whether a packet belongs to a particular flight depending on whether $g > T$ or $g \leq T$, where T is a threshold value. In our analysis we used several values of T ($\frac{1}{16}$ to 8) with similar results. We use the IAT as a measure of the size of the flight and call the units *IAT units (IATU)*. Thus a flight of 1 IATU means that the observed IAT was different from the preceding and following IATs. An IATU of 2 would mean that two successive IATs were identical. Figure 5 depicts two flights, the first of size 2 IATU and the second of size 3 IATU.

Our flight-finding algorithm is as follows:

1. Start with $IAT = 0$, $flightsize = 1$
2. Compare previous IAT with current IAT

3. If both $IATs$ are within threshold then increment $flightsize$ by one;
   else if $flightsize > 1$ start a new flight of size 0;
   else start a new flight of size 1;

The 'else if' line in our above algorithm means that an out-of-threshold IAT indicates the end of a flight, but a sequence of out-of-threshold IATs indicates consecutive 1-IATU flights. Our flights may therefore have 1 packet (1 IATU), 3 packets (2 IATU), 4 packets (3 IATU) and so on.

At this point we must digress and note a phenomenon that superficially looks similar. Since many TCP implementations [21, 27] implement *delayed-acks*, a host may send two packets for every *ack* it receives. Thus TCP's unit of transmission can be pairs of packets. We do not consider two-packet pairs since that behavior is not derived from the window-based component of TCP. Our algorithm will see a packet pair as two single-packet flights.

Flight behavior of TCP has been a matter of considerable debate. In fact there is not even a standard terminology for the phenomenon; other names for flight-like phenomena are *bursts* [28] and *rounds* [2]. While modeling TCP flows some authors simply assume the flight nature of TCP [16, 2]. In §6 there is a discussion on the validity of the model and how often flights are observed. Our statistics on flight behavior show that although flights are by no means common, they occur often enough to enable us to draw conclusions regarding a large byte percentage of the traffic.

4. RTT MEASUREMENT

We have already outlined the challenge of measuring RTT from unidirectional data. In this section we evaluate three methods of finding RTT. We first need to understand what information about RTT is available to us. Consider the scenario in figure 6 where we have only one instrumented fiber (in direction XY). We observe both download as well as feedback flows in direction XY. But there is a significant difference in timing information carried by each. The RTT of the download flows is of path XYX. One simple way to find the RTT for such flows is (see figure 3(a)) to find the time between the *syn-ack* and the *data* packet. The feedback flows carry slightly different information. Referring to figure 3(b) we note that the time between sending of *syn* and *ack* gives the RTT XYX. However, once the connection has been established, *ack* packets are sent in response to data packets in direction YX and so the timing information now obtained from the feedback flow XY is that of path YXY. There will be differences due to different serialization delays between the two round trips. There are a few other problems with using feedback flows, discussed below. In our analysis we are interested in the RTT XYX.

4.1 Three methods of measuring RTT for unidirectional data

We use three different methods to measure RTT on the unidirectional data sets. The methods are as follows:

- **Syn based Methods**. These are the methods described above. We can use them on both the download and feedback flows in direction XY to obtain RTT information of path XYX. Many authors have used these methods [12, 14, 29] although Zhang *et al.* [16] suggest that the method tends to underestimate RTT.

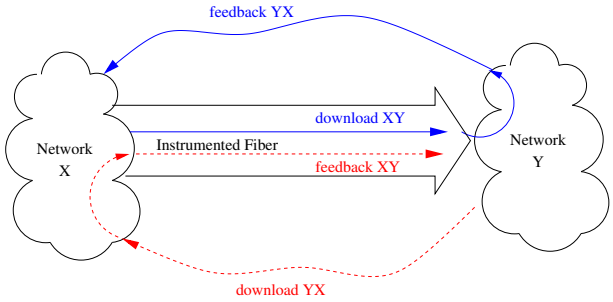


Figure 6: RTT information from different flows with a single instrumented fiber

- **Flight Method**. This method considers TCP dynamics at the packet level described in section 3. Figure 5(b) depicts the time between the leading edge of a flight and the leading edge of the next flight as the RTT. We must be careful when using this method. Rate-limited flows, which send packets at a constant rate constrained by hardware or access limitations, would show up as having large flights, which could potentially yield a large RTT misestimate since their traffic rate is independent of RTT. To avoid this source of false bias we only use flights containing three to five packets. Our statistics show that this causes us to ignore only about 4% of all observed flights. Since each flight yields one RTT value, we average the values found. Feedback flows have a structure different from other flows: since the feedback flow is in response to the data-carrying download flow, a set of back-to-back ack packets from the callee would result in a dispersed sequence of data packets at the caller. *Ack*-packets sent in response to these data packets (forming the feedback flow) would begin somewhat dispersed and further disperse by the time they reach our instrumented link, making the flight difficult to identify. Since this problem is less pronounced in data-feedback flows, we use this method on download flows and data-feedback flows.

- **Rate Change Method**. This method considers TCP dynamics at the fluid level. Let x be the number of bits transferred up to time t from time origin t_0. Then the instantaneous rate is $\frac{dx}{dt}$. Let W be the TCP congestion window at time t.

  PROPOSITION 2. *During the congestion avoidance phase in an interval* $[t_0, t_1]$ *if*

  – *there are no packet drops in* $[t_0, t_1]$, *and*
  – *the RTT does not change in* $[t_0, t_1]$ *then,*

$$RTT = \sqrt{\frac{MTU}{\frac{d^2x}{dt^2}}} \tag{7}$$

Data Sets	Byte Percentages		
	Syn Based	Flight Based	Rate Change
BB1-2002	82%	53%	58%
BB2-2003	70%	48%	52%
Abilene-2002	64%	44%	59%

Table 1: Byte percentages in flows for which each of the three methods yields an RTT estimate

PROOF. In congestion avoidance mode the congestion window increases by one MSS every RTT if no drops occur, which results in an increase of one MTU every RTT. So we have

$$\frac{dW}{dt} \approx \frac{MTU}{RTT}. \tag{8}$$

If there are no drops on an average, one window of bytes is transmitted every RTT, giving

$$\frac{dx}{dt} \approx \frac{W}{RTT}. \tag{9}$$

Hence we have

$$\frac{d^2x}{dt^2} \approx \frac{MTU}{RTT^2}. \tag{10}$$

Rearranging, we have the expression (7). □

Implementing this method requires fluidization of packets as described in §3. To ensure that the flow has entered congestion avoidance mode, we arbitrarily set a threshold of 15 packets (or the first out of sequence packet, whichever occurs earlier) as a threshold. We assume that the flow has entered congestion avoidance mode after this. We remark at this point that we also tried values of 25 and 30 for our threshold and noticed no appreciable changes in our results. The quantity $\frac{d^2x}{dt^2}$ derives from considering two rate points and finding the slope of the ramp between them as shown in figure 5(a). We assume that the RTT does not change in the small interval between successive rate points. We approximate the MTU with the maximum packet size observed in the duration of the flow. In the case of rate limiting, the slope will be close to zero or negative. We flag such cases and do not use them. Toward the end of flows transferring large files, the rate naturally comes down as there are not enough bytes to fill the congestion window. To avoid inaccuracies we do not use the final rate point obtained. The algorithm yields multiple RTT values (one value corresponding to each pair of rate points) and we average these values. We use this method for TCP download flows and data-feedback flows which, as mentioned in the last section, lend themselves well to fluidization.

We ran the three methods on the download flows and feedback flows. Table 1 shows the percentage bytes for which each method yields results. Recall that for TCP-unknown flows we did not see any *syn-ack*. Although the other two methods would yield results for such flows, we do not use them since we have no values produced by the *syn based* method to compare. We see from the table that although the flight-method and rate-change method are applicable to a specific subset of flows, namely those containing flights or ones large enough for fluidization, these methods do provide results for a significant portion of the traffic bytes. If we were to normalize the TCP-download and feedback flows to 100% (i.e., set the byte percentage for the *syn based* method to 100%), then the byte percentage for which the other two methods yield values is of the order of $60 - 70\%$. In other words these methods work for the flows that count.

We now obtain RTT by the three methods and compare the values obtained. Figure 7 shows the difference statistics. The first column compares the RTT values for which the syn based and Flight methods gave results, the second for which the Flight and Rate change methods yield results, etc. We have binned the differences so that 0% to 1% difference would be in the 1% bin, etc. We see that about 90% of the time all three methods yield values within 1% of each other. This surprisingly strong result suggests two things:

- The syn based method does not underestimate RTT. Since two other methods relying on entirely different facets of TCP dynamics yield the same values, the syn based method likely gives correct results. We will further justify this claim in section 4.2. The results imply that passive monitors such as NeTraMet may use the syn based method to reliably estimate RTTs in real time.
- The three methods yield RTT values at three different phases of the flow, the syn based at the starting epoch, the flight method in the early (in general slow start) phase and the rate change yields the average in the later (congestion control) phase. This suggests that the average RTT experienced by a flow is very close to what it experiences in the early part. So although the variance of RTT might be high (which in fact is an observation in [7]) the average RTT during the different phases is very similar. This in turn suggests that average queuing delay is a slowly varying quantity with respect to the duration of a flow.

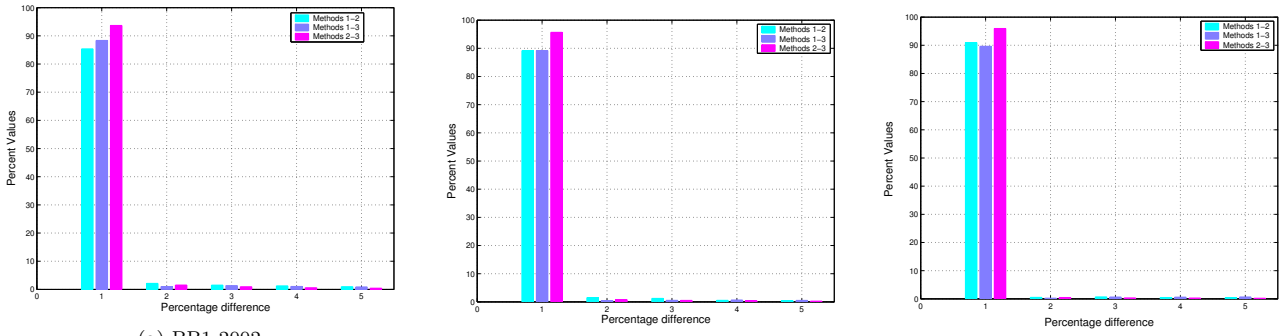
4.2 RTT distribution

Finally we find the actual RTT distribution for the packet traces (BB1-2002, BB2-2003, Abilene-2002). The algorithm we use to determine RTT is,

- If only one method yields results, use the value so obtained.
- If two methods yield results, use their average.
- If all three yield results, use the average of the lower two. This rule minimizes any artificial overestimates of RTT.

For the UNI-2002 data set we present RTT statistics obtained by sequence matching. Such values are most precise as they are what the flows actually see.

Figure 8 shows that all of the data sets show similar RTT distribution. The main features of the distributions are:



(a) BB1-2002 (b) BB2-2003 (c) Abilene-2002

Figure 7: Differences in RTT calculated by the three methods for the three data sets. The left column in each graph is the comparison between the *syn based* and the *flight* methods, the middle column between *syn based* and *rate change* methods and the right column is between the *flight* and *rate change* methods. We see that about 90% of the time the difference between any two methods is below 1%.

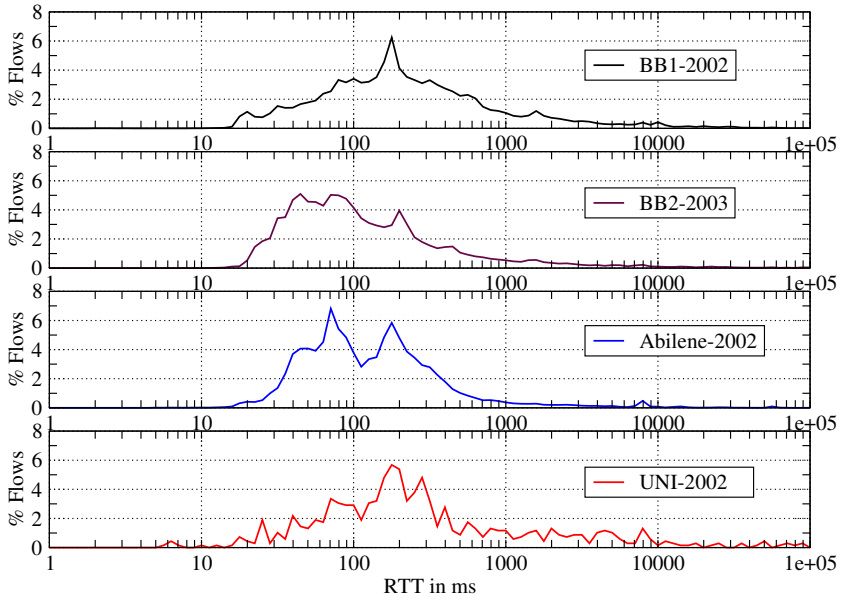


Figure 8: RTT distribution for the four data sets. The two spikes correspond to coast-to-coast US and Asian/European traffic.

- **BB1-2002** Since the trace was not anonymized we could check the geographical locations of the hosts. We attribute the spike at about 180 ms to traffic between Seattle and Beijing on that link. The pure propagation delay between these locations (smallest RTT) is of the order of 130 ms, which yields a queuing delay estimate of 50 ms.

- **BB2-2003** Again, this is a non-anonymized trace. This data set has a much larger spread in the lower RTTs, consistent with the dominance of US traffic.

- **Abilene-2002** The trace Abilene-2002 has two marked spikes. Since this trace is anonymized we are unable to identify the sources of the spikes, but the position of the link in the Eastern United States is consistent with spikes that correspond to coast-to-coast US traffic ($74ms$) and European/Asian traffic ($180ms$). The propagation delay for coast to coast in the US is of the order of 60 ms, giving a queuing delay estimate of about 14 ms.

- **UNI-2002** For comparison we also show the distribution of the UNI-2002 data set, which has fewer points resulting in a jagged appearance. This plot is from direct sequence matching of packets.

We draw the following conclusions from the RTT distributions:

- We see that there are queuing delays present in the system which are accentuated by distance (number of queues increases). This would imply that implementation of AQM mechanisms [4, 8] which aim at reducing queuing delays to zero, might possibly improve the RTT and hence the TCP performance for physically distant locations.

- TCP is inherently unfair to high RTT flows. The problem of stability and fairness is bound to crop up more often as Asian traffic increases in volume (the data set BB1-2002 for instance has 30% of bytes going to Asia). Calls for TCP fixes are likely to occur more often as this happens.

- If viewed in log-log scales, the tails of all the distribution fit the tail scaling law $500T^{-1.5}$ laid down in [15]. The law may be explained as a consequence of Asia and Europe being similar in distance to the US. Decay of RTTs after this point (about 180 ms) is likely to be due to the fact that countries in Eastern Europe, Africa and Southern Asia do not have very much traffic. The far end of the tail (over 500 ms) is unlikely to be due to queuing delays and might be worth investigating.

5. CONTROLLABILITY OF FLOWS ON THE INTERNET

Internet congestion control has been of timeless interest. Kelly [3, 26] proposed a framework that views congestion control as a mechanism for fair resource allocation in a network of users with elastic requirements, such as the Internet. This framework, and more generally, differential equation models of congestion control, can be used to study the stability of congestion control and active queue management (AQM) schemes using control-theoretic methods [4, 30, 6, 31, 32]. All these are based on a **fundamental premise: flows are controllable**.

The primary means of controlling flows on the Internet is by feedback control mechanisms, of which TCP is the best example. The object of the feedback controller is to attain the operating point which allocates the resource fairly. The time taken for the system to change by $\frac{1}{e}$ is called the time constant of the system. A system is considered to have changed its state after the elapse of three time constants, which corresponds to a 95% change in state. We define the *lifetime* of a flow to be the time interval between the transmission of the first and last packet. If the lifetime of a flow is greater than three time constants then the control mechanism would be effective. But what is the time constant of a TCP flow? Intuitively there seems to be a close link between RTT and the time constant for TCP flows. If the RTT is large then it is possible that the *acks* are not received during the flow's lifetime. In this case, the flow is never under the auspices of TCP's feedback control. However, if the flow's lifetime is long then even for large RTTs the feedback mechanism would be effective.

We illustrate the connection between RTT and the time constant for TCP flows by looking at the fluid model of TCP. Consider the fluid model of TCP with a single bottleneck link and a single source. Using the same notation as in section 3, we have the instantaneous rate as $\frac{dx}{dt}$. Define $\nu \stackrel{\triangle}{=} \frac{dx}{dt}$. Let the RTT experienced by the flow be T. We use the well known fluid approximation [33]

$$\dot{\nu} = \frac{1}{T^2} - \frac{1}{2}\nu(t)\nu(t-T)p(\nu(t-T)).$$

The first term models the rate of additive increase and the second term models the multiplicative decrease factor. The term $p(x)$ is the probability of marking/dropping at the link. The rate at which marked packets are received is then $\nu(t-T)p(\nu(t-T)$. So the the rate at which the bit rate is halved is given by the second term. Discretizing the above equation with a sampling time of $k = t/T$ yields

$$\nu[k+1] - \nu[k] = \frac{1}{T} - \frac{1}{2}T\nu[k]\nu[k-1]p(\nu[k-1]).$$

Linearizing the discrete time controller about the equilibrium ν^* found by setting $\dot{\nu} = 0$, we obtain

$$y[k+1] = y[k]\left(1 - \frac{1}{2}T\nu^* p(\nu^*)\right)$$
$$-y[k-1]\left(\frac{1}{2}T\nu^{*2}p'(\nu^*) + \frac{1}{2}T\nu^* p(\nu^*)\right).$$

This is of the form

$$y[k+1] = ay[k] - by[k-1],$$

whose eigenvalues are given by

$$\lambda = \frac{a \pm \sqrt{a^2 - 4b}}{2}.$$

The time constant of the system (assuming stability, $|\lambda| < 1$) is given by $\frac{-1}{log|\lambda|_{max}}$.

We illustrate the effect of RTT on the time constant derived above using a marking function $p = \left(\frac{\nu - C}{\nu}\right)^+$, where C = capacity of the link = 4.5 $MTU/ms \simeq 45\ Mb/s$. The figure 9 shows time constants corresponding to only stable, real eigenvalues. The relation between RTT and time constant can be understood from the graph. We see that an RTT of 100 ms corresponds to a time constant of 300 ms, i.e the flow's lifetime would have to be 900 ms or 9 RTTs for convergence. Thus, increasing RTT causes the system takes longer to converge to the final value.

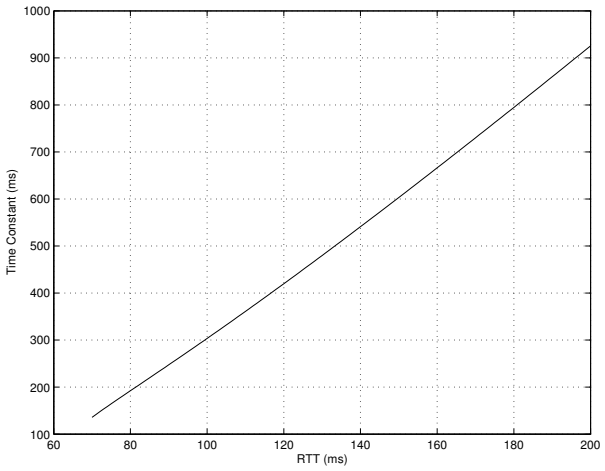


Figure 9: Variation of Time Constant of a TCP flow with RTT. Notice that it monotonically increases with RTT.

The above discussion suggests that from the viewpoint of a flow, time is a relative quantity. What is really relevant to for the transmission rate to converge and the system to reach stability, is that the flows' lifetimes be at least three time constants. Considering the close correspondence between time constant and RTT, we change the measure of time for each flow, in particular using a a dimensionless quantity:

$$controllability \triangleq \frac{lifetime}{RTT}.$$

For instance in the graph, a time constant of 300 ms would imply that the flow would require a controllability of 9 for it to converge. A high controllability factor would give the flow many time constants to converge, while a low one would mean that the flow is uncontrollable. We may also think of controllability as the minimum number of *acks* received in time interval. For instance a controllability factor of 10 would indicate that at least 10 *acks* would be received in the lifetime of the flow.

We apply these ideas to the three traces, to answer three questions:

1. What fraction of Internet traffic is in controllable flows?
2. What fraction of Internet traffic is in flows which carry a large number of bytes?
3. Do byte-rich flows get higher transmission rates?

The first column of graphs in figure 10 shows how timescale separation occurs by means of controllability plots. *Weighted by flow* implies that each flow gets a weight of one unit, whereas *weighted by bytes* implies that each flow is weighted by the number of bytes it transfers. On average over 90% of the bytes are in flows with a controllability factor of 10 and above. On the other hand about 75% of flows have a controllability factor below 10. Flows which have a controllability factor of 10 and above have a mean controllability of 5000. This means that if a flow lasts over ten RTTs, on an average, it lasts for an interval of the order of 5000 RTTs. Thus there is a division between numerous short (uncontrollable) flows carrying a small number of bytes and a small number of long (controllable) flows carrying a large number of bytes.

The second column of graphs in figure 10 depicts the separation between large (elephant) and small (mouse) flows. Using $10kB$ as a threshold puts an average of over 90% of the bytes in flows over this threshold. However, about 70% of the flows are below $10kB$ in size. We also observe that if a flow is larger than $10kB$, then on an average it is of the order of $5MB$ in size.

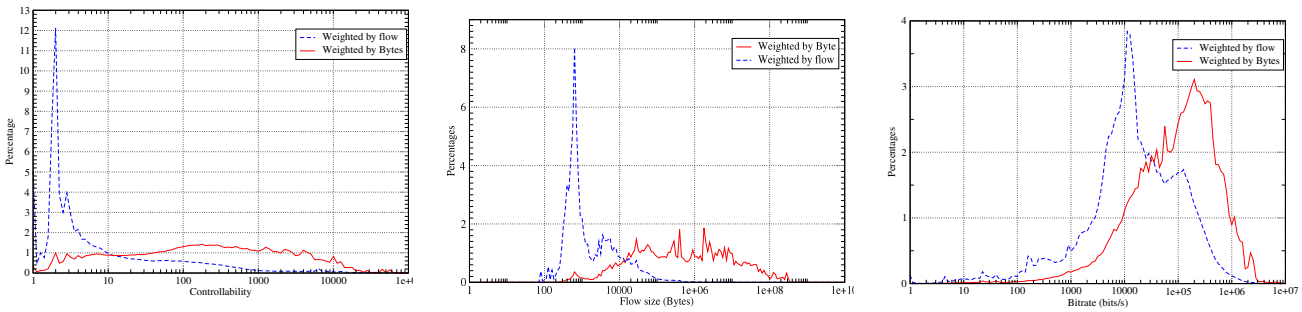
The third column in figure 10 shows the distribution of rates for the three data sets. There is no substantial separation of rates when looked at by flows or bytes, suggesting that large flows do not exhibit higher rates than small flows. We conclude that a large size flow is also likely to be a highly controllable flow, but does not have any particular bias ratewise. Similarly, a small size flow is likely to be uncontrollable regardless of its rate. These results lend validity to models that assume the division of Internet traffic into a small number of highly controllable flows (long running streams) transferring a large byte fraction and a large number of uncontrollable flows (noise) transferring a small byte fraction.

6. A CLOSER LOOK INTO TCP FLIGHTS

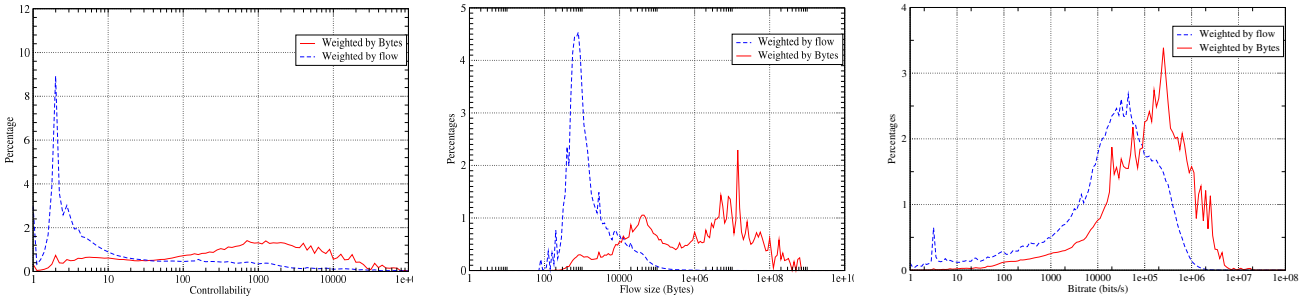
In previous sections we discussed TCP flight behavior and how to estimate RTTs. We have not yet studied the nature of flights, how often they occur, how well they reflect TCP behavior and what causes them. We devote this section to answering these questions.

We first obtain an idea of how flights appear at the point of instrumentation by looking at a few sequence number plots. Figure 11 shows two such plots, each for a TCP-download flow. The first example is a flow with a rate of about 300 kbps and the second at about 23 kbps. Flights are observed in both but do not show either doubling (corresponding to slow start) or single increases (corresponding to congestion avoidance) in flight sizes with any regularity. However, they are both parabolic in shape in accordance with equation 4. The curvature of the parabola is indicative of the RTT. Example 1 has a much lower RTT than example 2 as its curvature is much greater. The lower RTT of example 1 can also be gauged by looking at the inter-flight times.

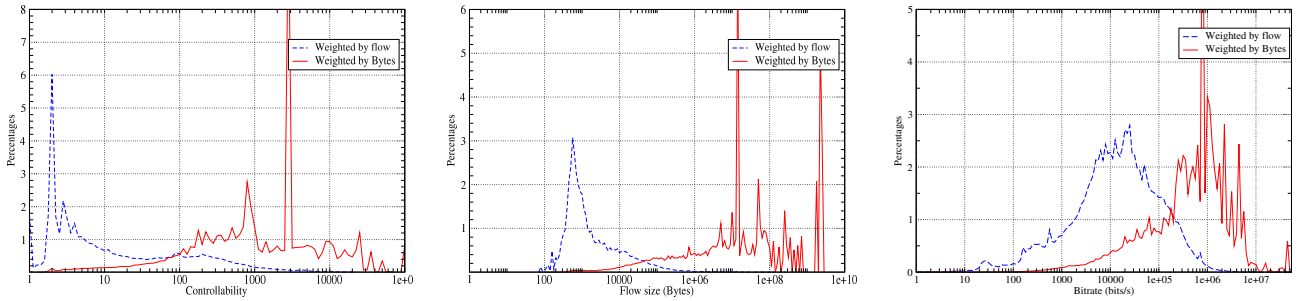
To go into more detail, figure 12 shows the distribution of flight sizes for download-flows. The unit of flight size is IATU. We can convert this into packets by recalling that a 1 IATU flight is a single packet flight, a 2 IATU flight has three packets and so on. We see that regardless of the



(a) Controllability (b) Flow Size (c) Bitrate

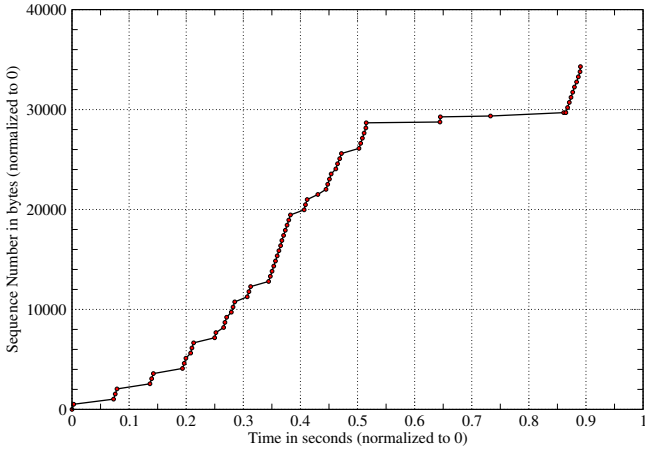


(a) Controllability (b) Flow Size (c) Bitrate

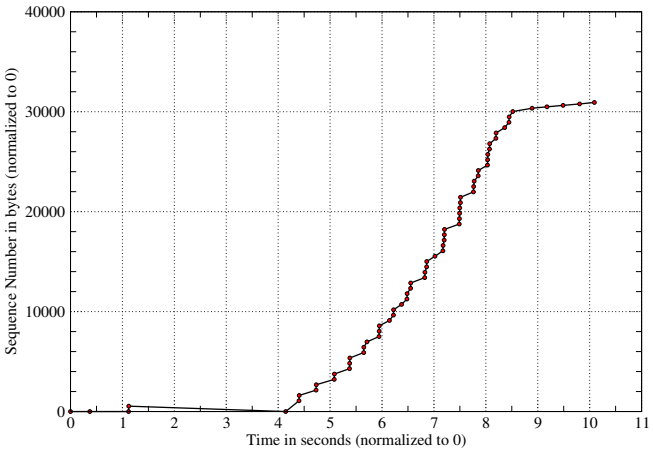


(a) Controllability (b) Flow Size (c) Bitrate

Figure 10: Controllability, flow size and rate distributions for the three data sets BB1-2002 (top row), BB2-2003 (middle row) and Abilene-2002 (bottom row). *Weighted by flow* implies that each flow gets a weight of one unit and *weighted by bytes* implies that each flow is weighted by the number of bytes transferred. We see separation based on controllability and flow size but there is no significant separation by rate. Note that the point of intersection of the two curves on each plot has no physical meaning, however it is the point which achieves the best separation of the two measures



(a) Example 1



(b) Example 2

Figure 11: Sequence number plots showing flight behavior for TCP download flows. Notice that flights do not follow any particular pattern that suggests slow-start or congestion avoidance mode. The examples show linear rate increases as seen by their parabolic shapes.

number of packets in the flow, flight sizes are usually quite small, flights larger than 7 IATU are rare. Figure 13 shows the distributions of number of flights in download-flows. We see that flights are much more common in flows with a large number of packets.

Now that we have established that flights are not a common phenomenon, we study the relation between them and the bandwidth-delay product (BDP). [28] suggests that flights are associated with high bandwidth, low delay flows. Figure 14 shows the relation between BDP and flight behavior. The bi-modal graph common to all the data sets is striking. Values of BDP at which flights are common are of the order of 1 kb and 16 kb. However, when we plot the BDP distribution (figure 15), we find that it shares the same bi-modal nature as the previous graph. So all the plots reveal is that Internet flows have two distinct regimes of BDPs, most likely corresponding to access methods (possibly home users with low BDP and office users with high BDP). To further understand the relation among flights, bandwidth, and delay, figure 15 plots 3D graphs of RTT, bandwidth and flight probability. Flights occur along two distinct lines on each figure (the scale is log-log), corresponding to the constant BDPs seen in the previous figure.

We are now in a position to draw inferences about flight behavior. We have seen that flights are not common and usually not large. They occur with greater frequency in two regimes: large RTT with medium bandwidth, and large bandwidth with medium RTTs. The results lead us to conclude that flights in general are a small window phenomenon. This would correspond in a majority of flows to the slow-start phase. When window sizes are small and network conditions correspond to one of the above regimes, TCP is forced to wait for *acks* causing flights.

6.1 The T-RAT tool

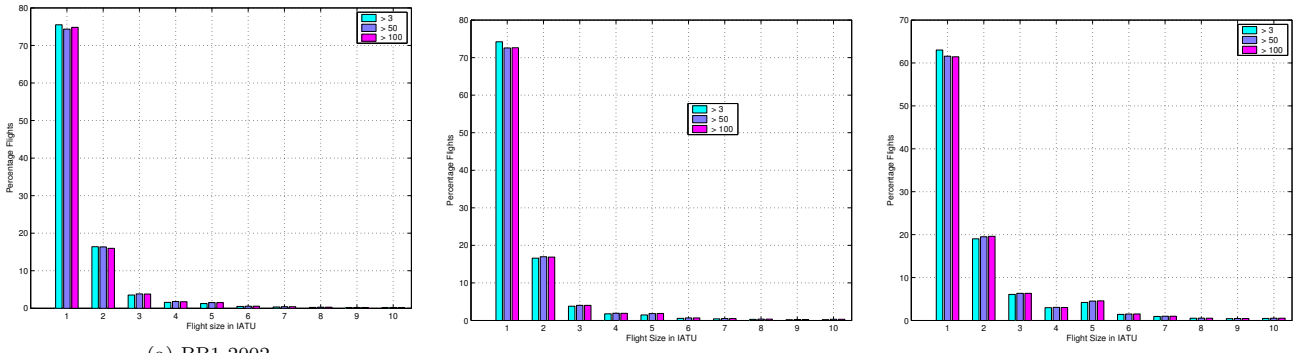
Recently Zhang *et al.* [16] proposed a new tool called T-RAT. The tool is built with the assumption that TCP sends packets in flights. It further assumes that by observing flight sizes, one can determine the mode at which TCP operates: a doubling of flight size indicates slow-start and an increase by one MTU indicates congestion avoidance.

The authors provided no statistics as to how often T-RAT actually finds flights, how often expected increases of window sizes corresponding to different TCP modes are observed, or a distribution of RTTs obtained. The tool is validated using a single data set $\mathcal{N}_2$, collected in Nov-Dec 1995 [27] to study the dynamics of different TCP flavors. The data consists of packet traces of specific transfers of 10 kB between select cooperating pairs of sites. The data set does not represent a particularly rich mix of Internet traffic. The authors claim that the syn-based method under estimates RTT, which is at variance with our results as well as [7]. In fact [14] observe that it has potential to overestimate RTT. This result is counterintuitive and requires explanation that has not been provided.

7. CONCLUSION

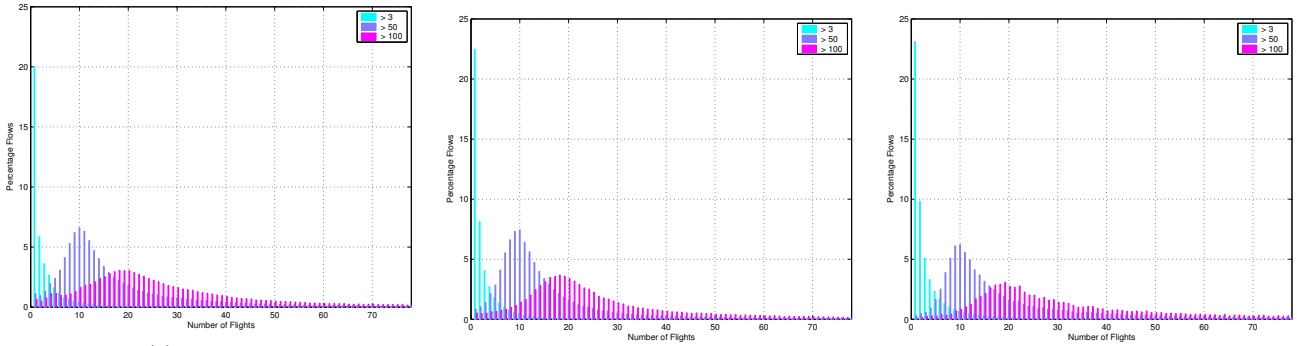
In this paper we have investigated the performance of TCP as a delayed feedback system. We examined how the dynamics of a TCP flow is affected by the RTT it sees and emphasized the intimate relation between TCP and RTT with regard to performance and stability.

As a measurement tool contribution, we implemented three methods of determining RTT from unidirectional packet trace files, and found insignificant differences in results. We concluded that the *syn based* method provides a good estimate of RTT that can be exploited in real time analysis of Internet traffic.



(a) BB1-2002 (b) BB2-2003 (c) Abilene-2002

Figure 12: Flight size distribution for the three packet traces. The left column in each graph is for flows greater than 3 packets in length, the middle for those greater than 50 and the right column is for flows greater than 100 packets in length. We notice that flights are usually small irrespective of the number of packets in the flow.



(a) BB1-2002 (b) BB2-2003 (c) Abilene-2002

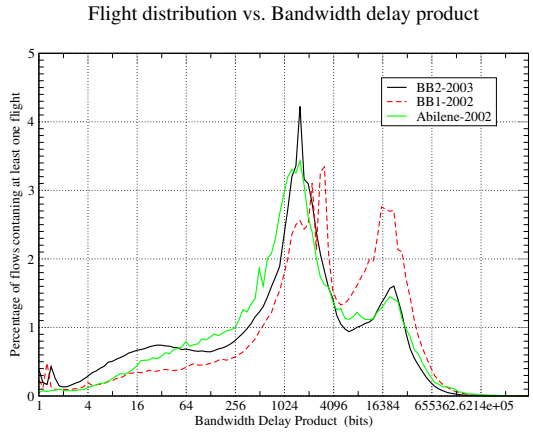
Figure 13: Number of flights in download flows on a per flow basis. The left histogram in each graph is for flows greater than 3 packets in length, the middle for those greater than 50 and the right histogram is for flows greater than 100 packets in length. We notice that flights are more common in flows with a larger number of packets.

In the context of modeling and simulation, we considered models of TCP fluidization and flights and showed that average RTT in different phases of a flow is a slowly varying quantity. We also provided distributions of RTT, which simulations could use as a source of realistic RTTs for flows.

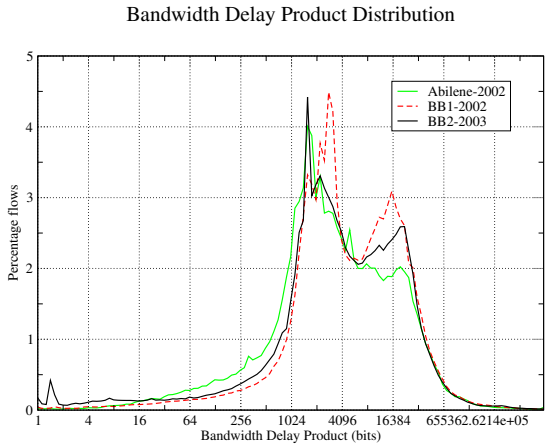
With regard to network control, we examined the time scale separation on the Internet by introducing a measure of flow lifetimes in terms of their associated RTTs. Our empirical analysis implies that that the paradigm of numerous-short-small flows and few-long-large flows is valid and that rate-wise the two modes have similar behavior. We made a case for implementation of AQM schemes to improve TCP performance in large RTT scenarios. The XCP paradigm assumes that there are short-large flows. Our work indicates that there are very few such flows. However, the fairness and stability issues with TCP in the high RTT regime is a case for XCP.

A byproduct of our work is a simple algorithm for flight identification based on packet inter-arrival times and we made use of this to collect statistics on flight behavior.

We plan to continue studying TCP dynamics, starting with identification of packet drops by observing rate halving. We also hope to use NeTraMet in a project to observe RTTs over a large time scale and map the congestion topology of the Internet. This would enable us to find the points at which AQM schemes would have maximum impact.

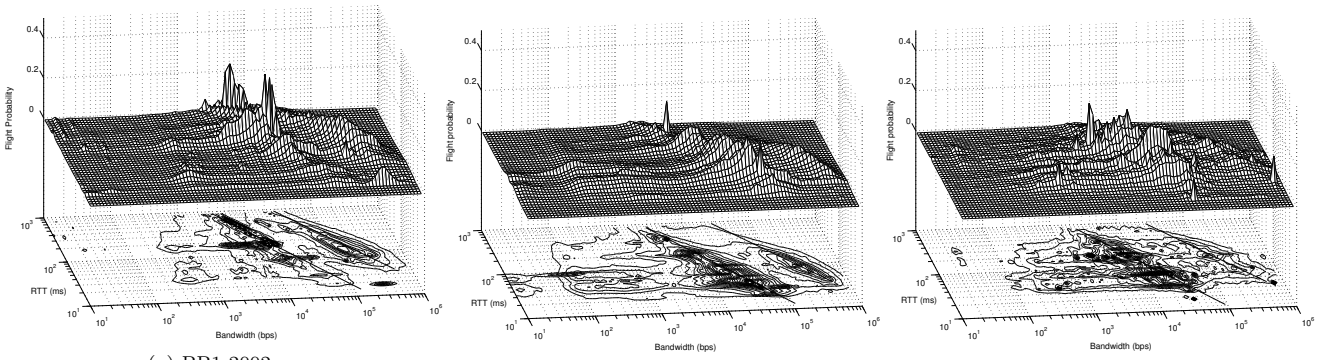


(a) Distribution of Flights versus BDP



(b) BDP distribution for flows in the three data sets

Figure 14: Plots showing the relation between BDP and flights



(a) BB1-2002

(b) BB2-2003

(c) Abilene-2002

Figure 15: Flight behavior for the three data sets as a function of RTT and bandwidth

8. ACKNOWLEDGMENTS

We would like to thank all those who made this work possible: Jörg Micheel of NLANR and Endace, Ruomei Gao of GaTech and Dan Andersen of CAIDA.

9. REFERENCES

[1] V. Jacobson, "Congestion avoidance and control," in *Proceedings of ACM Sigcomm*, August 2003.

[2] D. Towsley and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proceedings of ACM Sigcomm*, October 1998.

[3] F. P. Kelly, "Models for a self-managed Internet," *Philosophical Transactions of the Royal Society*, vol. A358, pp. 2335–2348, 2000.

[4] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual queue algorithm for active queue management," in *Proceedings of ACM Sigcomm*, August 2001.

[5] F. Paganini, J. Doyle, and S. Low, "Scalable laws for stable network congestion control," in *Proceedings of Conference on Decision and Control*, December 2001.

[6] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for future high bandwidth-delay product environments," in *Proceedings of ACM Sigcomm*, August 2002.

[7] F.D. Smith J. Aikat, J. Kapur and K. Jeffay, "Variability in TCP round-trip times," in *Proceedings of IMW*, October 2003.

[8] V. Misra, W. Gong, and D. Towsley, "A fluid-based analysis of a network of AQM routers supporting

TCP, flows with an application to RED.," in *Proceedings of ACM Sigcomm*, August 2000.

[9] G. Vinnicombe, "On the stability of networks operating TCP-like congestion control," in *Proceedings of the IFAC World Congress*, July 2002.

[10] R. Johari and D. Tan, "End-to-end congestion control for the Internet: Delays and stability," *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 818–832, December 2001.

[11] A. Mukherjee, "On the dynamics and significance of low frequency components of Internet load," *University of Pennsylvania, Report MS-CTS-92-83/DSL-12*, 1992.

[12] S. Martin, A. McGregor, and J. G. Cleary, "Analysis of Internet Delay Times," in *PAM*, April 2000.

[13] C. J. Bovy, H. T. Mertodimedjo, G. Hooghiemstra, H. Uijtervaal, and P. Van Mieghem, "Analysis of end-to-end delay measurements in Internet," in *PAM*, 2002.

[14] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times," in *Computer Communications Review*, July 2002, vol. 32, pp. 75–88.

[15] A. Broido and K. C. Claffy, "Invariance of Internet RTT spectrum," in *Proceedings of ISMA, Leiden*, October 2002.

[16] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of Internet flow rates," in *Proceedings of ACM Sigcomm*, August 2002.

[17] S. Floyd, "Sally floyd: Questions," 2002, www.icir.org/floyd/questions.html.

[18] "Endace measurement systems web page," http://www.endace.com.

[19] "DAG-University of Waikato Computer Science Department," 2001, http://dag.cs.waikato.ac.nz.

[20] K. C. Claffy, H. W. Braun, and G. Polyzos, "A parametrizable methodology for Internet traffic flow profiling," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1481–1494, October 1995.

[21] R. Stevens, "TCP/IP Illustrated, vol.1. Addison-Wesley," 1994.

[22] T. Karagiannis, A. Broido, N. Brownlee, and K. C. Claffy, "Estimating the true intensity of P2P traffic," *To be published*.

[23] "Abilene-I data set, *NLANR measurement and network analysis group*," 2002, http://pma.nlanr.net/Traces/long/ipls1.html.

[24] "ITSS Internet development group: NeTraMet," 1999, http://www.auckland.ac.nz/net/NeTraMet/.

[25] N. Brownlee, "Using NeTraMet for production traffic measurement," in *Proceedings of IM2001*, May 2001.

[26] F. P. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: Shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[27] V. E. Paxson, *Measurements and Analysis of End-to-End Internet Dynamics*, Ph.D dissertation, University of California, Berkeley, April 1997.

[28] S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level analysis and modeling of network traffic," in *Proceedings of IMW*, November 2001.

[29] Z.-L. Zhang, V. Ribeiro, S. Moon, and C. Diot, "Small-time scaling behaviors of Internet backbone traffic: An empirical study," in *IEEE Infocom*, March 2003.

[30] S. Kunniyur and R. Srikant, "A time-scale decomposition approach to adaptive ECN marking," *IEEE Transactions on Automatic Control*, vol. 47, no. 6, pp. 2024–2029, June 2002.

[31] S. Kunniyur and R. Srikant, "End-to-end congestion control: utility functions, random losses and ECN marks," in *Proceedings of IEEE Infocom*, March 2000.

[32] F. Paganini, Z. Wang, J. Doyle, and S. Low, "A new TCP/AQM for stable operation in fast networks," in *Proceedings of the IEEE Infocom*, April 2003.

[33] R. Srikant, *The Mathematics of Internet Congestion Control*, Birkhauser, 2003.