TCP Revisited: A Fresh Look at TCP in the Wild

Feng Qian
University of Michigan

Alexandre Gerber
AT&T Labs Research

Z. Morley Mao
University of Michigan

Subhabrata Sen
AT&T Labs Research

Oliver Spatscheck
AT&T Labs Research

Walter Willinger
AT&T Labs Research

ABSTRACT

Since the last in-depth studies of measured TCP traffic some 6-8 years ago, the Internet has experienced significant changes, including the rapid deployment of backbone links with 1-2 orders of magnitude more capacity, the emergence of bandwidth-intensive streaming applications, and the massive penetration of new TCP variants. These and other changes beg the question whether the characteristics of measured TCP traffic in today's Internet reflect these changes or have largely remained the same. To answer this question, we collected and analyzed packet traces from a number of Internet backbone and access links, focused on the "heavy-hitter" flows responsible for the majority of traffic. Next we analyzed their within-flow packet dynamics, and observed the following features: (1) in one of our datasets, up to 15.8% of flows have an initial congestion window (ICW) size larger than the upper bound specified by RFC 3390. (2) Among flows that encounter retransmission rates of more than 10%, 5% of them exhibit irregular retransmission behavior where the sender does not slow down its sending rate during retransmissions. (3) TCP flow clocking (*i.e.,* regular spacing between flights of packets) can be caused by both RTT and non-RTT factors such as application or link layer, and 60% of flows studied show no pronounced flow clocking. To arrive at these findings, we developed novel techniques for analyzing unidirectional TCP flows, including a technique for inferring ICW size, a method for detecting irregular retransmissions, and a new approach for accurately extracting flow clocks.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols

General Terms

Measurement, Algorithms

Keywords

Network measurement, TCP

1. INTRODUCTION

IP networks today carry traffic from a diverse set of applications ranging from non-real-time email and bulk data transfer like FTP to applications with much more stringent real-time performance and reliability requirements as Voice over IP (VoIP), Internet television (IPTV), Internet games and critical business transactions. A number of intertwined factors have contributed to this material shift in the application mix from even a few years ago when best-effort non-real-time applications like email, FTP and Web dominated. These factors include the rapid deployment of backbone links with 1-2 orders of magnitude more capacity, the increasing reach of broadband access networks, the emergence of bandwidth-intensive streaming applications, and a steady relentless economic-technological move towards transitioning even mission-critical applications from dedicated networks to the Internet using architectures like Virtual Private Networks (VPN).

Interestingly, even though the applications and their requirements have multiplied, the Transmission Control Protocol (TCP) [8] has remained the dominant transport-layer protocol in IP networks, being widely adopted by many of these new applications. Today TCP accounts for majority of the traffic on the Internet. This has happened even though TCP was originally designed to support a reliable, in-order delivery of a byte-stream between two end-points in a bandwidth friendly manner, and is not the ideal transport protocol for applications with real-time constraints. Practical considerations that favored TCP include *(i)* TCP is deployed almost everywhere *(ii)* using TCP helps offload many low-level transport details that an application developer would otherwise have to contend with, and *(iii)* ease of maintaining reachability across firewalls which are routinely configured to allow TCP packets through but block non-TCP flows. Fueled by the need to support more stringent performance requirements of emerging applications, the past few years have also witnessed the massive penetration of new TCP variants or new TCP congestion control algorithms like FAST [34], HSTCP [15] and CUBIC [16], and vendors promoting acceleration boxes that offer proprietary optimizations to TCP. However, these and other changes beg the question of whether, and more importantly, how they impacted the characteristics of TCP traffic in today's Internet, or if the behavior has largely remained the same as found by earlier in-depth studies of measured TCP traffic, the latest of which date to some 6-8 years ago [35, 10]. Given the continuing dominance of TCP, and its central role in preventing congestion collapse in the Internet, understanding its behavior is vital for the proper management, provisioning and capacity planning of these networks and for developing insights to guide protocol design.

In this paper we undertake a detailed exploration of TCP behavior from multiple vantage points in a large tier-1 ISP. We use a predominantly *passive measurement* approach using actual traffic traces for our analysis for reasons of scale, coverage and diversity. Compared to *active probing*, the passive approach is non-intrusive and more scalable requiring no additional coordination, instrumentation or deployment of end host measurement points. We can utilize in-network passive trace collection capabilities that are parts of existing deployed infrastructure. Our trace collection is carefully designed to get a diversity of network traffic mixes including backbone links, broadband access and enterprise traffic. The passive measurement approach allows us to capture the entire spectrum of TCP activity, in the relative proportions it is actually used, without any distortion or artificial biases, over the observation period. Given the existence of many TCP variants, some with multiple versions, and multiple parameters, and the lack of understanding of either the relative distribution of these settings or how they impact behavior, it would be very hard for a purely active probing approach to cover all these possibilities or to focus on the (unknown) interesting ones.

Along with its advantages, an in-network passive measurement approach has its own challenges. For instance, access to bidirectional traces is required by traditional techniques for analyzing certain types of TCP behavior (*e.g.,* tracking the congestion window). However, due to the prevalence of asymmetric routing, such traces are difficult to obtain in practice, especially for backbone links. As one contribution of this paper, we develop new analysis techniques that are suitable for unidirectional flows. Passive measurement also lacks a powerful aspect of active probing – with the latter, it is possible to tailor the probing activity carefully to force the protocol to reveal more details about its actions under different scenarios. We therefore augment our passive measurements with targeted active probes as needed. In particular, we utilize active probing for validation, where we gather the RTT, loss rate, frequency characteristics, *etc.* as ground truth by controlling the active probes.

1.1 Contributions

Using existing techniques where applicable and developing appropriate new methodologies where required, we explore the following main dimensions.

Have TCP flow sizes, durations and rates changed significantly compared to those 6-8 years ago? In particular, what are the corresponding distributions of "heavy-hitter" flows [31, 7] *i.e.,* flows with exceptionally large size, long duration, fast speed and strong burstiness compared to the earlier studies? Heavy-hitters contribute to significant traffic volumes and understanding their behavior is vital to many aspects of network management such as effective traffic measurement [13], scalable load sensitive routing [31], network anomaly detection [22] and usage-based pricing and accounting [12]. We compare our results with two previous studies [35, 10] and pinpoint the evolution of Internet flow characteristics that we observe.

What is the initial congestion window (ICW) distribution? A larger ICW allows a flow to be more aggressive by sending a larger burst of data at the beginning of the flow without any throttling. A large proportion of flows today are short and end before exiting TCP slow start. While such flows can benefit individually from using an inappropriately large ICW size, the widespread use of large ICWs will introduce large traffic bursts and may adversely affect the network performance and is therefore not desired. The existing approach for ICW estimation [26] involves active probing, and therefore, we develop a new passive measurement based ICW estimation scheme that uses only the timestamp information for the first few packets in the connection of a unidirectional data flow. We find that while most flows comply with TCP specifications [6, 5], up to 15.8% of senders in our data have initial congestion window greater than $\min(4 * MSS, \max(2 * MSS, 4380))$, the upper size mandated by the specifications [5]. We also observed ICWs as large as 9KB in our datasets.

When encountering losses, do senders slow down appropriately as mandated by TCP? This is a fundamental requirement for all TCP implementations and its adherence is critical to avoid congestion collapse in the network. The existing approaches to measuring this behavior either use active probing [26], or use bidirectional flows to precisely track the congestion window using a FSM [17]. We develop a passive-measurement based statistical approach to identify situations where the sender does not slow down its transmission rate when the retransmission rate increases. Our approach requires only unidirectional flows and is independent of the particular variant of TCP, unlike the existing schemes. Our findings indicate that in most cases, the sender does slow down its sending rate when retransmission rate increases. Among flows with retransmission rates higher than 10%, we do find 2.5% to 5% of the flows exhibit irregular retransmission behavior. Further investigations revealed that these cases could be attributed to by two main causes *(i)* abnormal retransmission not conforming to RFC-compliant TCP specifications (*e.g.,* retransmitting packets that were not lost); and *(ii)* under-utilization of the congestion window.

What is the distribution of the TCP flow clock and what is its origin? We define the TCP flow clock to be the regular spacing that may exist between flights of packets. The traditional view has been that the RTT dominates the origin of flow clocks for most flows, and existing RTT estimation algorithms [33, 36] implicitly use RTT as the flow clock. However, if the flow clock is not generated by the transport layer, these algorithms will have poor accuracy. One reason this can happen is when applications like streaming media perform their own rate control and packet pacing on top of TCP. We develop a novel frequency domain analysis technique to identify the flow clock independent of its origin. Our analysis indicates that less than 50% flows have distinguishable flow clocks, and reveals that in practice RTT is not the main determinant of flow clocks in many cases. Among our flows with a measurable flow clock, up to 60% have clocks originated by non-RTT factors such as software clocks of applications, periodical OS events (*e.g.,* keyboard scanning), and "retiming" effects [35] of a bottleneck link.

2. RELATED WORK

We describe two areas of related work to our study.

Characterization of Internet Flows: Researchers started to investigate characteristics of TCP connections more than 10 years ago, by passively measuring traffic patterns [32] or actively probing end-to-end packet dynamics [29]. For more detailed characterization, the T-RAT study [35] considers the distribution of flow rates, and further analyzes the origins of different rates such as congestion limited and transport limited. A more recent study [10] examines characteristics of "heavy-hitter" TCP flows [12] in four dimensions, namely size, duration, rate, and burstiness, along with their correlations in detail. There has also been work investigating particular application types of flows such as streaming [11], VoIP [23] and gaming [9] flows. Compared to this, our study serves as a reexamination of observed TCP behavior, motivated by the significant changes experienced by the Internet as mentioned in §1. We compare our results with two most recent previous studies [35, 10] to pinpoint the evolution of Internet flow characteristics that we observe.

Inferring TCP Behaviors: TCP is a complex protocol with various implementation variants and tunable parameters. Researchers have developed many techniques to infer TCP behaviors by actively probing hosts or passively analyzing packet traces. For active probing tools, TBIT [26] strategically drops incoming packets and sends fabricated packets over raw socket to infer Web servers' TCP behavior. TBIT also uses known techniques [14] to identify the TCP flavor (Tahoe, Reno and New Reno) based on sender's response to packet losses. A previous study [24] used TBIT to study the impact of "middleboxes" (NATs, proxies, *etc.*) on TCP. Tools like Nmap [2] and p0f [3] take another approach by using a signature database to fingerprint the OS version of the target host, therefore indirectly inferring the corresponding TCP implementation.

A wide range of passive analysis techniques also exist. For example, *tcpanaly* [27] infers TCP implementation from packet traces based on observed differences among 8 TCP implementations. *tcpflows* [17] keeps track of the sender's congestion window based on predefined finite state machine of TCP Tahoe, Reno and New Reno. Work by Lu and Li [21] statistically infers the correspondency between the arrived ACK packet and the data packets sent from packet traces using maximum-likelihood criterion. T-RAT [35] also focuses on unidirectional packet traces, separating the trace into flights, then inferring the TCP state of each flight (*e.g.*, slow start or congestion avoidance).

Compared to these previous studies, our work develops methodologies requiring minimum information from the trace which are unidirectional traces containing only timestamp, sequence number, acknowledgement number, packet size and TCP flags. Furthermore, we address a new problem that has not been well explored, *i.e.*, accurately extracting flow clocks originated from either RTT or other factors.

3. DATA CHARACTERIZATION

We describe the data used in our study and perform basic characterization of the data set.

3.1 Datasets Used

As summarized in Table 1, we use seven quite diverse datasets named BU, BS1 to BS4, VPN and DSL in this study. BU (Backbone Unsampled) is a 30-minute unsampled TCP trace collected from a 10Gbps backbone link of a Tier-1 ISP on June 17, 2008. BS1 to BS4 (Backbone Sampled) are sampled TCP traces from the same link collected on June 26, 2008. The duration of each dataset is approximately 1 hour. Sampling was performed on a per-flow basis with a sampling rate of 50% (one in two flows), so that all packets from a sampled flow were captured. The VPN dataset is an unsampled bidirectional trace collected from all the uplinks of a VPN provider edge router (PE) on January 5, 2009. The DSL dataset, also bidirectional and unsampled, was collected from a BRAS (Broadband Remote Access Servers; an aggregation point for approximately 20,000 DSL lines) on January 9, 2009. For each packet, we record the following fields of IP and TCP headers, plus a 64-bit timestamp: source and destination IP, source and destination port number, packet length, IPID, IP fragment flag/offset, sequence number, acknowledgment number, receiver's window size, TCP payload length, TCP flags, and TCP options. From each dataset, we extract flows based on a 5-tuple of src/dst IP, src/dst port numbers, and protocol (always TCP). Following previous studies [35, 10], we use a threshold of 60 seconds to decide that a flow has terminated.

We discuss two limitations of our datasets. First, similar to previous measurements using passive traces (datasets in [35] were 30min to 2hours, plus a 1-day sampled trace; two datasets in [10] were 20min and 2hours), our finite sampling durations (30min to 2h46min) may influence the distribution of flow characteristics. Another limitation is that our datasets only contain TCP traffic, while both [35] and [10] use a mixture of TCP and UDP traffic. However, basic statistics of VPN dataset[1] shows that UDP contributes only 7.8% of the traffic, comparable with the fractions in [10] (4% and 15%). We believe the above limitations (finite dataset length and a lack of non-TCP traffic) do not qualitatively affect our comparison with [35] and [10].

3.2 Flow Characteristics

We analyze four basic flow characteristics: size, duration, rate, and burstiness. Size is simply defined as the total number of bytes in the flow including headers; and duration is the time span between first and last packet of the flow. Flow rate is computed by dividing flow size by flow duration. Similar to previous work [35, 10], we focus on longer-lived flows by ignoring flows with durations of less than 100ms when computing the flow rate. We give the definition of burstiness later. We first characterize the distributions of size, duration, and rate of *all* flows before focusing on "heavy-hitter" flows [31, 7] *i.e.*, flows with exceptionally large size, long duration, high speed, and strong burstiness. Understanding the behavior of heavy-hitter flows are useful for various applications [13, 31, 22, 12], as described in §1. Due to the well-known heavy-tailed distribution of Internet flows, the number of such flows is very small. They however may contribute to significant traffic volume.

Figure 1 plots the complementary cumulative distribution (CCDF) of flow size, duration, and rate across 7 datasets (shown in thick lines), compared with the T-RAT study which is a most recent TCP study similar to ours on understanding the origin of Internet flow rates conducted in 2001 [35], whose eight datasets are shown as thin lines[2]. For flow size and duration, no qualitative difference exists in log-log plot. We do observe much higher flow rate in our dataset. For instance, in their datasets, only 4% to 10% flows are faster than 100kbps; the percentage increases to at least 17% in ours. This is mostly explained by higher speed backbone links and increasingly popular broadband Internet access. Correspondingly, high-speed TCP variations were deployed accommodating faster link speed. For instance, BIT-TCP was used as the default TCP implementation since Linux 2.6.8 in 2004 [16]. BIT-TCP makes the congestion window grow faster by modifying the linear growth function for existing congestion avoidance, thus mitigating the under-utilization problem for high speed and long distance paths.

Note that the higher rate is observed by ignoring considerable number of flows with duration less than 100ms. We will see a much more significant increase of flow rate by looking at a smaller number of "heavy hitter" flows later.

Given the significant similarities across our seven datasets as evidenced by closely placed thick lines representing them in Figure 1, we attempt to quantify the similarity between two datasets by detecting flows with the same unordered pair (IP1/24, IP2/24) *i.e.*, only preserving 24-bit prefix. In particular, we define the Similarity Index (SI) between dataset X and Y as $SI_{X,Y} = |\Omega(X) \cap \Omega(Y)| / \max\{|\Omega(X)|, |\Omega(Y)|\}$, where $\Omega(X)$ denotes the set containing unordered (IP1/24, IP2/24) pairs of all flows in X. We find that the similarity index for any pair of datasets is less than 33%, therefore we claim that our datasets are reasonably heterogeneous.

Next we examine heavy-hitter flows and compare our results with a most recent study analyzing traces of a regional network and Abilene OC48 backbone link collected in April 2003 and August 2002 respectively focusing on heavy-hitter flows [10]. This study

[1]We only have simple statistics of UDP traffic for VPN dataset.

[2]For both Figures 1 and 2, to ensure effective comparison, we superimposed our data onto figures obtained from the original papers preserving their scale.

Table 1: Basic statistics of datasets used

Name	Date	Length	# Packets	Volume	Bidir	Sampled	# Flows	Description
BU	Jun 17, 2008	30min	643 M	377G	No	No	32.6 M	Unsampled unidirectional backbone link trace
BS1	Jun 26, 2008	62min	635 M	371G	No	1 in 2 flows	30.6 M	Sampled unidirectional backbone link trace
BS2	Jun 26, 2008	59min	637 M	424G	No	1 in 2 flows	30.0 M	Sampled unidirectional backbone link trace
BS3	Jun 26, 2008	58min	636 M	346G	No	1 in 2 flows	29.6 M	Sampled unidirectional backbone link trace
BS4	Jun 26, 2008	47min	633 M	360G	No	1 in 2 flows	24.0 M	Sampled unidirectional backbone link trace
VPN	Jan 05, 2009	2h46min	522 M	245G	Yes	No	29.6 M	Unsampled bidirectional VPN link trace
DSL	Jan 09, 2009	2h18min	745 M	496G	Yes	No	25.3 M	Unsampled bidirectional DSL link trace

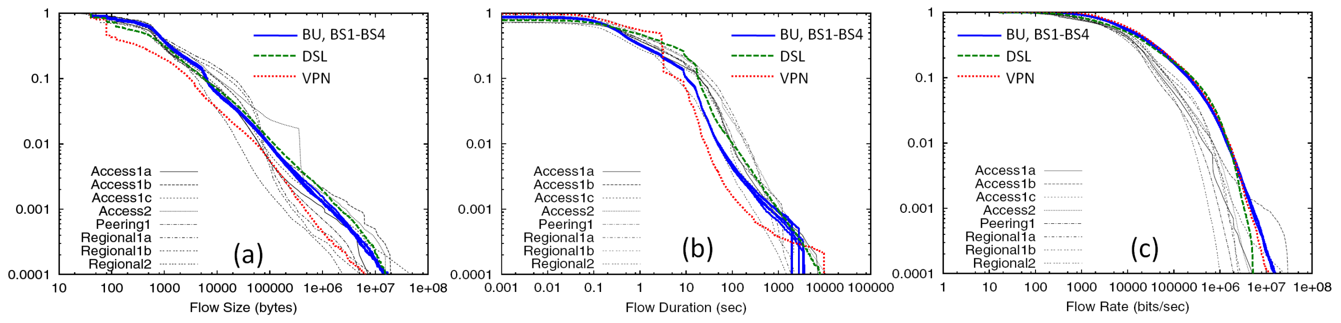


Figure 1: Complementary distribution of flow size, duration and rate (comparing with the T-RAT study in 2001). When computing the flow rate, we only examine flows with durations of 100ms or longer.

Table 2: Contribution of heavy-hitter flows in flow count and volume

Our datasets		Elephant	Tortoise	Cheetah	Porcupine
BU	Flows	0.09%	0.52%	0.60%	0.26%
	Vol.	49.4%	37.3%	23.8%	0.83%
BS1 (Similar to BS2-BS4)	Flows	0.08%	0.24%	0.60%	0.15%
	Vol.	55.8%	38.7%	22.0%	1.35%
VPN	Flows	6e-3%	0.42%	1.15%	0.04%
	Vol.	55.3%	50.4%	49.7%	0.24%
DSL	Flows	0.06%	0.34%	1.51%	0.12%
	Vol.	58.9%	48.7%	35.0%	1.07%
Two datasets in [10]		**Elephant**	**Tortoise**	**Cheetah**	**Porcupine**
LosAngeles Regional'03	Flows	1%	4%	2%	0.9%
	Vol.	71%	43%	16%	39%
Abilene OC48 Aug 2002	Flows	4%	4%	2%	1%
	Vol.	82%	45%	36%	40%

by Lan *et al.* finds strong correlations among some combinations of size, rate and burstiness, explained using transport and application-level protocol mechanisms. Four types of heavy-hitter flows are coined in that work: elephant, tortoise, cheetah, and porcupine, corresponding to flows with size, duration, rate, and burstiness greater than the mean plus three standard deviation of the respective measurement. In particular, the *burstiness* of a flow is calculated by multiplying the average burst rate by the average inter-burst time, where a *burst* consists of a train of packets whose inter-arrival time are less than 1ms. Only bursts with more than one packet are considered. As shown in Table 2, although the number of heavy-hitter flows is very small (except for porcupine flows), they contribute to significant traffic volume.

Figure 2 illustrates distributions of size, rate, and burstiness for four types of heavy-hitter flows in the BU trace (thick curves with labels). We did not present the duration distribution due to the bias introduced by limited duration of the BU trace (30min *vs* 2hr). Measurements from Lan's study [10] are also kept for comparison. We observe qualitatively similar distributions compared to other datasets and highlight key observations below.

As illustrated in Figure 2(a), the sizes of elephant, cheetah, and tortoise flows increase by about an order of magnitude. Since most are HTTP data flows with the source port 80, such increase is likely due to file size increase on Web servers and the trend of using HTTP to transfer large multimedia data. The growth in file size in both local and network file systems are well documented by previous measurements [4, 20]. The decrease in flow size for the overall dataset is caused by the 30min duration of our trace compared to the 2hr duration of data used in Lan's study.

As depicted in Figure 2(b), the rate of elephant flows that contribute to at least half of the traffic volume along with that of tortoise flows increases by one magnitude compared to Lan's study [10], most likely accounted for by the tremendous growth in link speed within both the core and edge networks. In Lan's study [10], the authors explain the rate increase of tortoise flows with user behavior (*e.g.,* one types a character every 30 seconds), as only 6% of tortoises are flows with size greater than 100KB. While in our results, we observe a trend in more long-lived flows (about 10% of them) are likely multimedia streaming or gaming as verified using the IP addresses and port numbers.

The most striking difference is the burstiness. First, as shown in Table 2, the very small number of porcupines contribute to less than 1.5% of the traffic volume, providing a sharp contrast to [10]. Second, as shown in Figure 2(a)(b), the size and rate of porcupine flows differ significantly from previous study: there is an increase in rate, but decrease in size. Finally, in Figure 2(c), the burstiness of heavy-hitter flows also deviate much from Lan's study, deserving further investigation we plan to conduct as future work.

Similar to [35] and [10], we also observe correlation between duration, rate and size. For each dataset, $(logS, logR)$ has correlation coefficient between 0.54 and 0.57, smaller than the values observed in [35] and [10]; while $(logR, logD)$ shows a stronger negative correlation between -0.69 and -0.60. $(logS, logD)$ are slightly positively correlated with correlation coefficient between 0.21 and 0.40.

To summarize our major findings, we observe significant increase of average flow rate. In particular, the rate of elephant flows in-

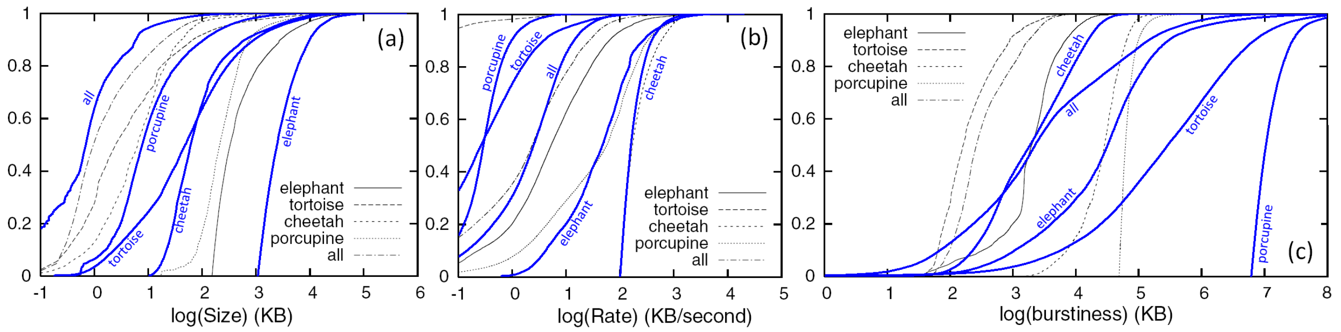


Figure 2: Distributions of size, duration, rate and burstiness for four types of heavy-hitter flows in BU. The base of logarithm is 10.

creases by a factor of 10, comparing with that of 6 years ago. Also, the sizes of elephant, cheetah, and tortoise flows increase by an order of magnitude. Our observations indicate the trend of upgrading Internet infrastructure at core and edge, the deployment of new TCP variants, and the trend of using TCP to transfer larger content such as video streaming and long-time online gaming.

4. NEW METHODS FOR FLOW ANALYSIS

In this section, we present three new algorithms for analyzing unidirectional TCP flows. As discussed before, we focus on long-lived flows which contribute much more traffic volume compared to short-lived ones. An inherent challenge in analyzing unidirectional flows is a general lack of information to reconstruct the TCP states at both the sender and receiver side. For example, if there is no pronounced flow clock or the flow clock does not originate from the transport layer (RTT), it is challenging to accurately estimate RTT. To overcome this difficulty, in §4.2 and §4.3, we adopt statistical approaches that require observing enough packets to infer certain properties with sufficient accuracy and high confidence. Such approaches are fundamentally different from existing deterministic approaches [28, 18] that precisely track the TCP state, given bidirectional trace. We validate the accuracy of our approaches in §5.

4.1 Inferring Initial Congestion Window Size

Inferring the initial congestion window (ICW) size helps detect aggressive TCP flows that send a larger burst of data at the beginning of the connection without any throttling. The value of ICW is the number of bytes a TCP sender can send immediately after establishing the connection, before receiving any ACKs from the receiver. We devise the following algorithm shown as Algorithm 1 to measure ICW given a unidirectional flow trace. The key task of this algorithm is to examine the normalized inter-arrival times (IAT) between the first $k + 1$ data packets. The first "large" gap (*i.e.,* larger than θ, a threshold of normalized IAT as shown in Algorithm 1) indicates that the sender has reached its congestion window and is waiting for ACKs from the receiver. If such a gap is not detected, we pick the maximum gap among all k IATs. Note that in Algorithm 1, I_1, the IAT between the SYN-ACK packet and the first data packet usually includes the extra delay caused by the server OS and is therefore discarded. As a result, Algorithm 1 only examines $I_2, ..., I_k$. Based on our empirical findings in §5.1, we choose $k = 8$ and $\theta = 0.2$.

There are several limitations in Algorithm 1. *(i)* Our approach only works for server-to-client data flows (*i.e.,* the trace starts with a SYN-ACK packet) due to lack of sufficient data from client to server; *(ii)* there must be no retransmissions in the first $k + 1$ packets; *(iii)* the only factor that prevents the sender from sending more data should be reaching the congestion window. This is ensured by requiring that the first $c - 1$ packets have same size (*i.e.,* equal to the maximum segment size MSS), therefore $c \times MSS$ is the inferred ICW (Line 6 to 7); *(iv)* if flight-based IAT is too small (*e.g.,* RTT $<$ 2ms) or packet-based IAT is too large, then the large gap will be blurred. Note that the active probing technique for inferring ICW described in the TBIT work [26] also requires that conditions *(i)*,*(ii)*,*(iii)* hold. The accuracy (higher than 98%) of our method is evaluated in § 5.1.

Algorithm 1 Infer the Initial Congestion Window Size

```
Require: Packet trace starts with SYN-ACK packet
Require: There are no retransmissions in first $k + 1$ packets
1: $c \leftarrow null$; Calculate inter-arrival time $I_2, ..., I_k$
2: for $j = 2$ to $k$ do
3:     if $I_j / \sum_{t=2}^{k} I_t > \theta$ then $c \leftarrow j - 1$; exit for; endif
4: end for
5: if $c = null$ then $c \leftarrow \operatorname{argmax}_{2 \le j \le k}\{I_j\} - 1$; endif
6: if first $c - 1$ packets have the same packet size $MSS$ then
7:     return $c \times MSS$; else return unknown ICW; endif
```

4.2 Detecting Irregular Retransmission

Slowing down during retransmission (especially during periods with many retransmissions) is one of the fundamental requirements for all RFC-compliant TCP implementations [8], since retransmissions indicate packet loss as inferred by the TCP sender [17]. We denote a TCP flow that does not slow down its sending rate during retransmission as a flow with *irregular retransmission*.

We devised a new tool called the *Rate Tracking Graph* (RTG), based on a statistical algorithm for detecting irregular retransmission behavior. The basic idea behind RTG is an intuitive observation that holds for all known TCP implementations: when retransmission rate increases, the sender should decrease the *upper bound* of its sending rate by reducing the congestion window [8]. This implies a negative correlation between the retransmission rate and the sending rate, *i.e.,* a positive correlation between the retransmission rate r and the time t required to successfully transfer a fixed size of data (*e.g.,* 50 KB). RTG samples all pairs of (t, r) by sliding a tracking window W along the flow to test whether t and r exhibit any strong positive correlation.

The details of our RTG tool are described in Algorithm 2. The input is a unidirectional TCP flow trace (server to client) with high retransmission rate (*e.g.,* >10%). This is because RTG requires sufficient number of sample points (t, r) to generate statistically confident results. A lower threshold of retransmission rate may detect more irregular retransmissions, but the confidence of accuracy decreases as well.

Algorithm 2 first identifies retransmitted bytes by examining repeated sequence numbers (Line 1 to 4). Subsequently, given a fixed tracking window size W, it samples all pairs of (t, r) by sliding the window of varying length t along the flow, where t is determined by the requirement that there are W non-retransmitted bytes in the window, and where r is the retransmission rate of the window (*i.e.,* retransmitted bytes in the window divided by W). In the case of regular retransmissions illustrated in Figure 5(a), t and r have strong positive correlation. We are interested in RTGs with small positive or negative correlation coefficients, as illustrated with the example shown in Figure 5(b).

It is important to point out cases where a well-behaved flow does not exhibit a strongly positive correlation coefficient. If the flow rate is not limited by the congestion window, it is not necessary for the sender to slow down its rate, even if the congestion window is reduced. In fact, in our scenario with high retransmission rate, the rate limiting factor can also be [35] *(i)* the server application (it does not generate data fast enough); *(ii)* the server's sending buffer in OS kernel; *(iii)* the receiver's window; *(iv)* the bottleneck link. In particular, case *(i)* accounts for 30% to 50% irregular retransmissions in our datasets, and will be discussed in detail in §6.2.

There exist other factors that may affect the correlation coefficient of RTG. First, the tracking window W should be large enough to include more than one RTT, and be small enough so that sliding the tracking window covers various retransmission rates. We empirically choose 4 tracking window sizes: 50KB, 100KB, 200KB, and 400KB, and conservatively claim an irregular transmission if all tracking windows yield correlation coefficients less than 0.1. Second, at the sender's side, there may be pauses that enlarge t. For interactive Web applications, *e.g.,* the server may be idle for seconds, with no data to send. We devise an *Entropy-based cutting algorithm* that removes large gaps by separating the flow into segments. We then generate RTGs for each sufficiently large segment (*i.e.,* greater than 1MB) whose IATs are less intermittent than those of the original flow. A flow's IAT-Entropy is defined as the following (P_i denotes the i^{th} packet):

$$E_{IAT} = -\sum_{P_i, P_{i+1}} \frac{iat(P_i, P_{i+1})}{d} \log\left(\frac{iat(P_i, P_{i+1})}{d}\right)$$

where d is the flow duration. The algorithm iteratively cuts a segment S into S_1 and S_2 as long as $\max\{E_{IAT}(S_1), E_{IAT}(S_2)\} > E_{IAT}(S)$. Here, an increase in entropy indicates that the IATs in the newly generated segments are more homogeneous. In practice, such entropy-based cutting requires no tuning parameters and succeeds in removing large gaps. The remaining small gaps may add "noise" to the RTG, but usually they do not significantly change the correlation coefficients. The third factor concerns dramatic changes in the sending rate, as illustrated with an example in Figure 10(g). Given the rare occurrence of this case in our datasets, we will deal with this as future work.

4.3 Flow Clock Extraction

We define the *TCP flow clock* to correspond to the regular spacing between flights of packets. The most commonly accepted cause of TCP flow clocking is RTT-based and hence inherently linked to the transport layer [33, 36]. By devising a methodology for accurately extracting TCP flow clock information from unidirectional packet traces and applying it to actual data, we observe that TCP flow clocking can also originate from the application layer or even the link layer. Understanding the different root causes for TCP flow clocks has far-reaching implications. For one, if the flow clock is not generated by the transport layer, existing algorithms [33, 36] that implicitly associate RTT with flow clock will suffer from low

Algorithm 2 Rate Tracking Graph

```
Input: Unidirectional Packet Trace $T$, Window size $W$
Output: Rate Tracking Graph
Require: $T$ has significant retransmissions ($> 10\%$)
 1: for all byte $b \in T$ do
 2:     if ∃byte $b'$ : $(b'.seq = b.seq) \wedge (b'.ts > b.ts)$ then
 3:         $b.lbl \leftarrow 0$; else $b.lbl \leftarrow 1$; endif
 4: end for
 5: $head \leftarrow 0$; $tail \leftarrow 1$
 6: while $tail \le T.len$ do
 7:     $head \leftarrow head + 1$
 8:     while $(tail \le T.len) \wedge (\sum_{i=head}^{tail} byte(i).lbl < W)$ do
 9:         $tail \leftarrow tail + 1$
10:     end while
11:     if $tail \le T.len$ then
12:         $r = tail - head + 1 - W$; $t = byte(tail).ts - byte(h).ts$
13:         Plot $(t, r)$ on Rate Tracking Graph
14:     end if
15: end while
```

accuracy. Second, we find that flows with large non-RTT based flow clock tend to have more consistent flight size. Also, these flows are more likely to transfer data with an inappropriately large congestion window, due to a larger timeout value not complying with RFC [5], as illustrated in §6.3. Third, we envision that flow clocks can serve as a new feature for traffic classification and network anomaly detection.

The main idea behind our method for accurately extracting the dominant flow clock (if it exists) is as follows. We view a packet trace as a sequence of pulse signals in temporal domain. Next we transform the signal into the frequency domain via Fourier Transform. In the frequency domain, we design an algorithm that combines pattern recognition techniques with our empirical knowledge about TCP clocking to detect peaks (spikes) within relevant frequency bands. Lastly, the flow clock is defined to be the *fundamental frequency i.e.,* the lowest frequency in a harmonic series [25].

Our detailed implementation of this flow clock extraction algorithm consists of 6 steps. *(i)* Given a unidirectional packet trace T, the algorithm discretizes timestamps of T into a binary array B using a sampling frequency of 500Hz; $B(i) = 1$ if and only if there is at least one packet that arrived between times $2i$ and $2i+2$ msec. *(ii)* We use the Discrete Fourier Transform (DFT) to transform B into the frequency domain: $F = DFT(B, 2^{\lceil log_2^{B.len} \rceil})$, then downsample F to 1,000 points (resolution of each point is 0.25Hz). *(iii)* Detect the local maxima (candidate peaks) by sliding a window of size w and sensitivity s along the spectrum, and mark points whose amplitude is larger than $\mu + s\sigma$ as candidate peaks (μ: mean, σ: standard deviation of the points within the window). In our implementation, we apply 3 pairs of (w, s) to discover both narrow and wide peaks: $w = 4, 8, 16$ and $s = 8, 16, 32$. *(iv)* Cluster consecutive candidate peaks (distance of less than 5 points) into a single peak; remove peaks whose amplitude is less than $\mu_0 + 3\sigma_0$ (μ_0: mean, σ_0: standard deviation of all 1,000 points). *(v)* For each peak with frequency f, test whether f is a fundamental frequency: for $k = 2, 3, 4$, if there exists a peak with frequency $f' \in (kf - \delta, kf + \delta)$ where the tolerance parameter δ is set to three[3]. *(vi)* Return the minimum fundamental frequency if found.

In the above approach, after downsampling the spectrum to 1,000 points, the resolution of each point is 0.25Hz. Therefore the ex-

[3]In our implementation, for a fundamental frequency, we only require 2 out of 3 values of k correspond to peaks to increase robustness to errors.

tracted fundamental frequency may be inaccurate when the flow clock is large. We solve this problem by performing additional postprocessing if the fundamental frequency is less than 5Hz. First, we separate the flow into flights based on the rough estimation of flow clock using the algorithm introduced in §4.1 of [35], except that here we rely on an estimation of the flow clock instead of using blind search as it is in [35]. Next, the refined flow clock is calculated as the average time difference between the beginning of consecutive flights after removing outliers falling outside $(\mu - 3\sigma, \mu + 3\sigma)$.

We tuned the above parameters based on the empirical findings described in §5.3. In rare cases, a flow may possess two or more fundamental frequencies *e.g.*, both RTT-based clocks and application-based clocks are observable in the flow. We find that the smallest fundamental frequency usually obscures the detection of larger ones, so that discovering a second or third fundamental frequencies may not be accurate or informative in practice. We intend to pursue this issue in future work.

5. METHODOLOGY VALIDATION

We systematically evaluate our algorithms introduced in §4. We first validate the ICW estimation algorithm by comparing with active probing in the TBIT approach [26], followed by an analysis of false positives in RTGs by triggering retransmissions through injected packet losses to thousands of HTTP downloading sessions, and finally validate flow clock detection by comparing with ground truths obtained from flow traces of different types.

The same dataset for active probing, consisting of 3,131 URLs, each pointing to a file with size greater than 1MB, is used for first two sets of experiments in §5.1 and §5.2. We performed DNS lookup for the domain part of each URL and replaced it with one or more IP addresses to eliminate DNS based server load balancing. This expanded the dataset to 5,844 URLs. We set up a testbed for URL query experiments based on the TBIT [26] tool which infers TCP behavior of Web servers by active probing. For example, TBIT infers ICW by sending an HTTP GET request packet and not acknowledging any further packet. The Web server will only be able to send packets that fit within its ICW before retransmitting the first data packet. We added two new tests to TBIT: `ICWPassive` and `RTG`. After establishing the connection to the Web server, `ICWPassive` receives $k+1$ packets, closes the connection by sending a TCP RST and estimates ICW passively as described in §4.1; `RTG` receives data as a normal TCP receiver but randomly drops packets at a certain loss rate, then generates RTG based on §4.2 after connection termination. Besides these enhancement, we also improved TBIT in several other aspects, *e.g.*, made the format conform to Konqueror 3.5.8 for FreeBSD 7.

5.1 Inferring Initial Congestion Window Size

As shown in Table 3, the experiment was performed using 3 different MSS values: 1460B, 512B and 128B. For each MSS, each URL was probed 5 times. We eliminate cases where probing fails due to HTTP errors (less than 30%), or either algorithm reports inconsistent results in 5 trials (less than 0.7%). For the remaining URLs, we regard a probing as accurate if both algorithms produce the same result. We report the accuracy for MSS=1460B, 512B and 128B to be 98.4%, 98.8% and 99.2%, respectively. Inconsistent cases are conservatively considered as inaccurate.

Algorithm 1 has two parameters k and θ. For k, we tried $k = 7, 8, 9, 10$ and finally chose $k = 8$ since it results in the highest accuracy for all three MSS. We chose $\theta = 0.2$ based on the distribution of normalized IAT for $I_2, ..., I_8$ where $I_j^{norm} = I_j / \sum_{a=2}^{8} I_a$ for $2 \le j \le 8$ (Figure 3(a)), since $\theta = 0.2$ well separates two frequently occurred ranges $(0, 0.1)$ and $(0.4, 0.55)$, which correspond to packet-based IAT and flight-based IAT, respectively. The complementary CDF of normalized IAT including I_1 where $I_j^{norm'} = I_j / \sum_{a=1}^{8} I_a$ for $1 \le j \le 8$ is shown in Figure 3(b). In most cases I_1 is much larger, due to the extra delay caused by the server OS / applications, explaining the need for discarding I_1 in Algorithm 1. Also in Figure 3(b), curves of I_3 and I_6 depict the typical increase in congestion window size at the beginning of slow start *i.e.*, from 2 MSS to 3 MSS (instead of 4 MSS due to delayed ACKs [6]).

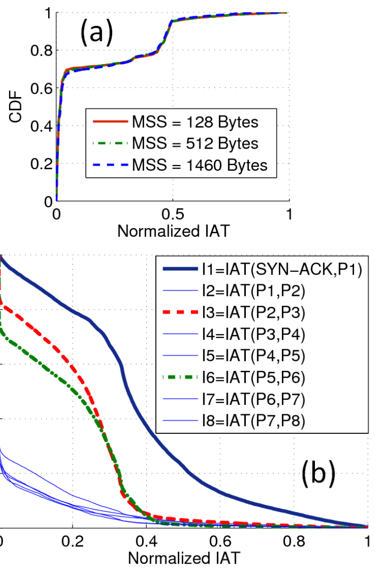


Figure 3: (a) Distribution of normalized IAT for the first 9 packets (excluding IAT of SYN-ACK and first data packet) (b) Distribution of normalized IAT for the first 9 packets

Table 3: Compare the ICW algorithm with TBIT

MSS	1460B	512B	128B
Total URLs	5,844	5,844	5,844
HTTP errors	1,523	1,494	1,783
Inconsistent results by TBIT	11	11	6
Inconsistent results by Algorithm 1	9	6	8
Both return inconsistent results	20	8	12
Remaining URLs	4,281	4,325	4,035
Accuracy	98.4%	98.8%	99.2%

5.2 Rate Tracking Graph

We first validate whether under high retransmission rate, RTGs of most flows exhibit positive correlation coefficients. Our testbed downloaded each URL described previously. During the file download, the testbed dropped packets at loss rate of 5%, 10%, and 15% respectively, and also generated RTG for the downloading trace. Such approach only introduces random losses not congestion losses. However, all known TCP implementations do not distinguish them (both are triggered by duplicated ACKs or timeout)[4]. Next we show results for the loss rate of 10%. For other loss rates, qualitatively similar observations are made.

We successfully downloaded 4,462 out of 5,844 URLs. For each downloading trace, four tracking window sizes of 50KB, 100KB,

[4]We created a 1Mbps bottleneck link to increase congestion losses. In that case, well behaved flows also exhibit strong positive correlation coefficients when both congestion and random losses exist.

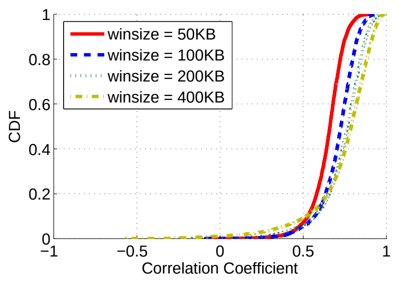


Figure 4: Distribution of correlation coefficients of RTGs

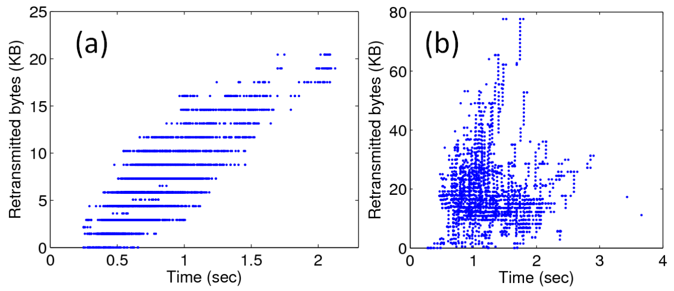


Figure 5: (a) A normal Rate Tracking Graph (window size $W = 50KB$, correlation coefficient=0.88) (b) An abnormal Rate Tracking Graph (window size $W = 100KB$, correlation coefficient=-0.04)

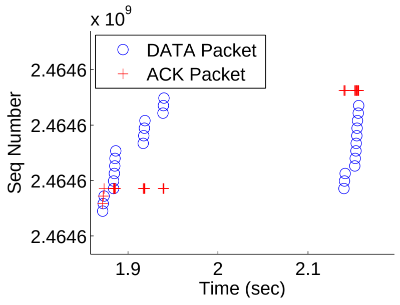


Figure 6: Irregular retransmission detected by RTG

Table 4: Validation of the flow clock extraction algorithm

Application	Flow Clocks	Errors
Web/FTP bulk transfer	RTT	0 / 20
Interactive Web session	RTT / NoClk	1 / 20
SSH	NonRTT / NoClk	0 / 20
Remote Desktop	RTT / NoClk	0 / 20
VoIP (Skype)	NonRTT / NoClk	1 / 20
Multimedia streaming	RTT / NonRTT	0 / 20
Gaming	NonRTT / NoClk	0 / 20

200KB and 400KB are used, generating 17,848 RTGs. Figure 4 plots the distribution of correlation coefficients for each window size, clearly indicating that in most cases, the sender slows down the rate when retransmissions increase. Correlation coefficients for any pair of tracking window sizes (W_x, W_y) are positively correlated between (0.75, 0.92). After entropy-based cutting, the average entropy only slightly increases from 6.06 to 6.11 (0.8%), because it is unlikely that large gaps occur in these HTTP download traces. However, for our seven passive collected datasets, the average entropy increases by 5% to 9%.

We examine the maximum of correlation coefficients for four window sizes to discover irregular retransmissions. All flows have $max\{CC_{50}, CC_{100}, CC_{200}, CC_{400} > 0.3\}$ except for one with $CC_{50} = 0.09$, $CC_{100} = 0.08$, $CC_{200} = -0.04$, and $CC_{400} = -0.27$ (CC_W denotes RTG's correlation coefficient for tracking window size W) as shown in Figure 5(b), which provides a contrast for a typical RTG with high positive correlation coefficient illustrated in Figure 5(a). From the sequence diagram of the irregular flow shown in Figure 6, we observe that in fast retransmission, instead of retransmitting the lost packet (indicated by the duplicated ACK), the server retransmits all packets from the lost packet to the current packet with the maximum sequence number. This can be caused by problematic TCP implementation. In fact, the OS fingerprinted by Nmap [2] looks very strange ("`HP 9100c Digital Sender multifunction printer`" with confidence of 93%)[5].

5.3 Flow Clock Extraction

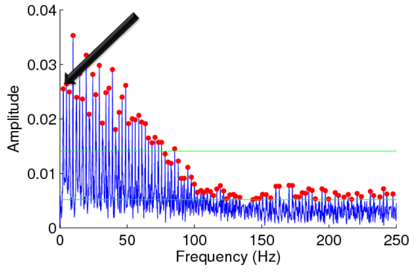
To evaluate the flow clock extraction algorithm (§4.3), we capture flows of different applications (listed in Table 4) where the flow clocks and their origins are known, then compare the ground truth with extracted flow clocks. We collected 10 flows for each application type. By measuring the RTT using `ping` and examining the frequency spectrum and packet sequence diagram, it is easy to determine the dominating flow clock and its origin. Similar to the algorithm described in §4.3, we declare the existence of flow clock by observing at least two human-observable harmonic frequencies among $2f_0, 3f_0, 4f_0$, where f_0 is the human-observable fundamental frequency[6]. We declare that our algorithm correctly extracts the flow clock if the difference between human judgment (choosing the local maximum) and algorithm output is less than 10%. We declare that the clock is originated from transport layer if the difference between flow clock and RTT is less than 10%. The whole experiment was conducted twice at one author's department (optical fiber connection for campus network) and home (broadband cable connection). The validation results are reported in Table 4, from which we select eight representative cases illustrated in Figures 7(a) to (h). For each plot in Figure 7, the bullets on spikes denote candidate peaks (after clustering) as described in §4.3 step *(iv)*; the arrows point to the extracted flow clocks (fundamental frequencies); and two horizontal lines indicate μ_0 and $\mu_0 + 3\sigma_0$ (explained in §4.3 step *(iv)*).

For Web/FTP bulk transfer, we collected flows downloading or uploading files larger than 1MB with RTT varying from 20ms (Figure 7(a)) to 400ms (Figure 7(b)). Clocks of all flows clearly correspond to RTT. For interactive Web sessions such as GMail, RTT-based clocks in both directions are blurred by user's interaction at varying degrees (Figure 7(c)). For SSH flows[7], we observed intense amplitude at 62.5Hz (16ms) from client to server (Figure 7(d)) since IATs of most packets are multiples of the fundamental frequency at 16ms regardless of RTT. As shown in Figure 7(e), Skype flows from caller to callee exhibit dominant frequency characteristics at 50Hz (other peaks in Figure 7(e) are not fundamental frequencies), while we did not observe such behavior for reverse flows (callee to caller, Figure 7(f)). Such small non-RTT based clocks

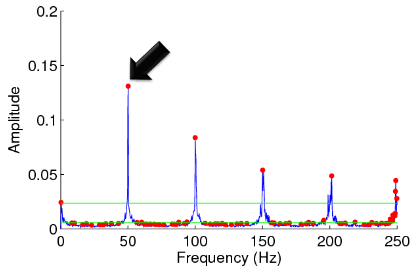
[5]We use Nmap 4.85 with `-O -host-timeout 600000`.

[6]We admit that such an approach introduces subjective elements; however, in most cases, such determination by human is trivial.

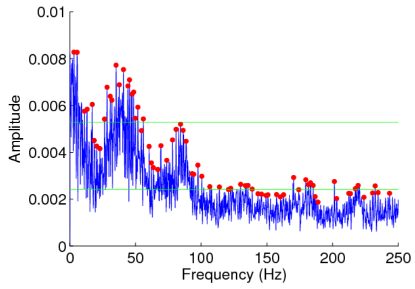
[7]We use SSH Secure Shell version 3.2.9 on Windows XP SP3 as client; we tried both Solaris 10 and Linux 2.6.20 as server.



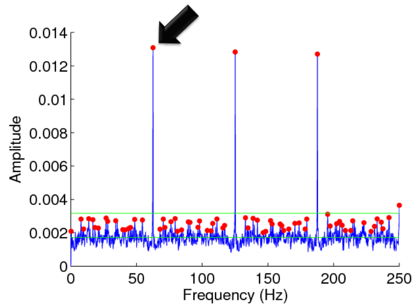
(a) Bulk HTTP Transfer
Flow Clock: 2.5Hz (RTT)



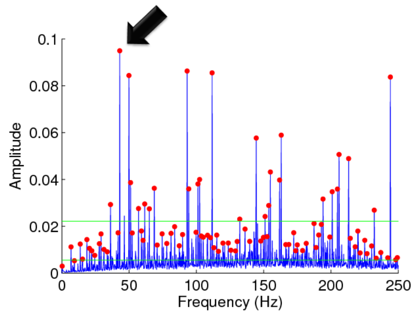
(b) Bulk HTTP Transfer
Flow Clock: 50Hz (RTT)



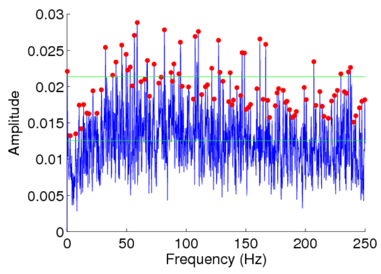
(c) Interactive Web Session (GMail)
No Flow Clock



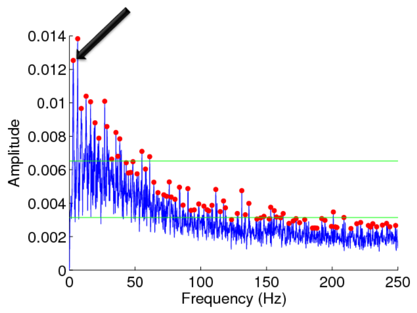
(d) SSH (Clinet→Server)
Flow Clock: 62.5Hz (Application)



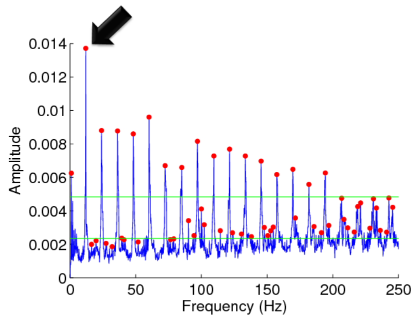
(e) Skype (Caller→Callee)
Flow Clock: 50Hz (Application)



(f) Skype (Callee→Caller)
No Flow Clock



(g) PPLive Streaming
Flow Clock: 5.6Hz (RTT)



(h) Winamp Radio Streaming
Flow Clock: 12Hz (Application)

Figure 7: Frequency spectrum and flow clocks for different applications

may be caused by software clocks of the user application or OS. For multimedia flows, as examples shown in Figures 7(g)(h), their clocks can either be RTT-based (*e.g.*, PPLive) or application-based (*e.g.*, Winamp Radio).

We investigated the origin of the 16ms-clock for SSH flows. By hooking the socket `send()` API and `WM_KEYDOWN` message (a keyboard event) in `SSHClient.exe` using Detours [1] (a binary interception tool for Windows functions), we observe that both events happen at a granularity of 16ms, indicating that the clock is caused by the timing granularity of keyboard scanning event in Windows XP.

Clearly, Figure 7 only lists several possible but not all flow clock configurations. Flow clocks are affected by multiple factors including link speed, packet loss rate, RTT, applications and user interaction. In §6.3, we present characterizations of flow clocks observed in our datasets.

6. ANALYSIS OF LONG-LIVED FLOWS

In this section, we apply methodologies introduced in §4 on our datasets and present the results. We only examine long-lived flows defined to be with duration of more than 30 sec and size larger than 1MB (The numbers of such flows are shown in the first row in Table 6). There are several reasons that we focus on long-lived flows. First, they contribute to considerable traffic volume, as for each dataset, the long-lived flows accounts for at most 0.16% of all flows, but contributes at least 55% of traffic volume; second, long-lived flows provide enough information as required by our statistical approaches; third, the reduction of the number of flows significantly saves analysis time without losing the global view on the datasets. §6.1, §6.2 and §6.3 discuss the results for ICW inference, irregular retransmission and flow clocks, respectively.

6.1 Initial Congestion Window Size

All our passively collected datasets exhibit IAT distributions very similar to those of active probing datasets as shown in Figures 3(a)(b). So we choose the same parameters $k = 8$ and $\theta = 0.2$.

Table 5: Distributions of Initial Congestion Window

ICW	Total	1-2	3	4	5	6	7
BU	18234	88.3%	8.0%	3.1%	.2%	.3%	.08%
BS1	18609	86.0%	10.2%	3.4%	.1%	.3%	.05%
BS2	18342	86.2%	9.2%	4.1%	.2%	.3%	.02%
BS3	18468	83.9%	9.1%	6.4%	.2%	.3%	.05%
BS4	15763	85.1%	8.6%	5.6%	.3%	.3%	.04%
VPN	2135	94.5%	3.2%	2.0%	0%	.4%	0%
DSL	18004	77.2%	6.9%	14.2%	.6%	1.0%	.03%

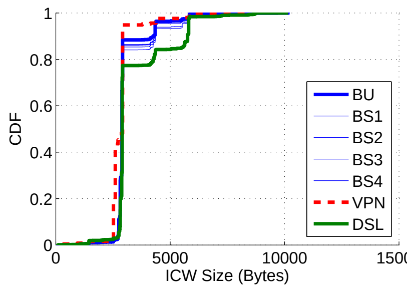


Figure 8: Distribution of ICW Size

RFC 2581 [6] requires that ICW must be less than or equal to 2*MSS bytes and not exceed 2 segments. RFC 3390 [5] updates RFC 2581 by changing the upper bound to $\min(4*MSS, \max(2*MSS, 4380 \text{ bytes}))$[8]. In our measurement results shown in Table 5, most flows have ICW of 2 MSS, while we also observe small fraction of flows (0.4% to 1.63%) whose slow start begins with ICW of more than 4 MSS. Figure 8 plots the distribution of ICW size in bytes, where ICWs mainly concentrate in two clusters: 2520 to 2920 bytes and $1460 \times 3 = 4380$ bytes, corresponding to 2*MSS and the numerical upper bound defined in RFC 3390, respectively. For DSL, about 14% flows form a third cluster around 5800 bytes (4*MSS, as shown in Table 5), which is an inappropriately large ICW. We also observe extreme cases where ICW is as large as 9KB. For each dataset, we report the percentage of flows with ICW greater than $\min(4*MSS, \max(2*MSS, 4380 \text{ bytes}))$ as follows. BU: 3.6%, BS1: 3.8%, BS2: 4.6%, BS3: 6.9%, BS4: 6.2% VPN: 2.3%, DSL: 15.8%. OS detection results (Nmap only fingerprints 24% of servers) show that almost all OS implementations of flows with inappropriately large ICWs are Linux 2.6.x or FreeBSD 6/7 (ICW is controlled by `sysctl_tcp_iw` variable in Linux kernel).

6.2 Irregular Retransmission

Recall that in §4.2, irregular retransmissions deviate from the usual case where the sender slows down its sending rate when the retransmission increases. Rate tracking graph (RTG) statistically detects irregular retransmissions for flows with overall high retransmission rate. We first present characterizations of retransmission rate and an overview of correlation coefficients of RTGs, then analyze the irregular retransmissions detected by our algorithm.

Figure 9(a) plots the distribution of retransmission rate. More than 55% of flows have almost no (less than 0.01%) retransmission. At least 80% of flows have retransmission rate of less than 1%. There exists little diversity in retransmission behavior across seven

[8]Some network cards (*e.g.,* Intel Pro 1000 NIC) provide a function called TCP Segmentation Offload (TSO) that allows the kernel to transmit packets with very large MSS (*e.g.,* 64KB). However, the NIC will eventually break the data down into proper MTU-sized packets (*e.g.,* 1448 bytes).

Table 6: Distribution of different types of irregular retransmission

	BU	BS1	BS2	BS3	BS4	VPN	DSL
Long-lived	31K	29K	29K	29K	26K	4.1K	33K
High Retran	977	1009	908	821	692	16	1672
Breakdown of flows with high retransmission rate							
Non-irregular	928	953	847	778	651	15	1622
Irregular	49	56	61	43	41	1	50
Breakdown of flows with irregular retransmission							
Non-Conform	16	19	20	12	9	0	16
Sender Limit	17	18	25	20	16	1	29
Partial Overlap	5	5	5	3	6	0	0
Gaps/Rate Chg	2	4	2	3	2	0	3
Unknown	9	10	9	5	8	0	2

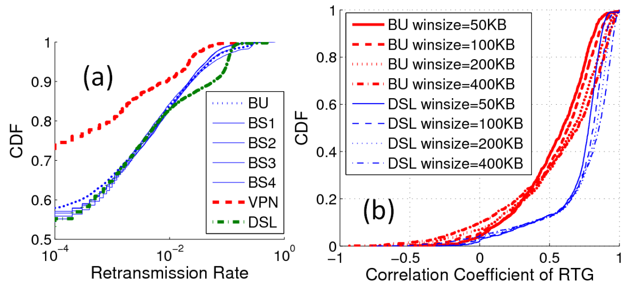


Figure 9: (a) Distribution of packet retransmission rate (b) Distribution of correlation coefficient of RTG for different window size W

datasets. The retransmission rate of VPN is lowest on average, while DSL has more flows with retransmission rate higher than 5%.

Next, we pick flows with retransmission rate higher than 10% and generate their RTGs by applying Algorithm 2 with preprocessing described in §4.2. Figure 9(b) plots the distribution of correlation coefficients of all RTGs for BU (lowest on average) and DSL (highest on average), using tracking window sizes of 50KB, 100KB, 200KB and 400KB. Clearly, for each window size W, majority of flows exhibit strong positive correlation between the transmission time for W bytes and the retransmission rate. On the other hand, we are more interested in understanding the opposite part, irregular retransmissions, which are conservatively defined here as $\max\{CC_{50KB}, CC_{100KB}, CC_{200KB}, CC_{400KB}\} < 0.1$. As shown in Table 6, those irregular flows account for 2.5% to 5% of flows with retransmission rate higher than 10%. By carefully analyzing each irregular flow, we classify them into five categories.

Category 1. *There exists clear indication that the retransmission behavior does not conform to RFC-compliant TCP specifications.* In particular, we observe cases where *(i)* the sender retransmits a train of packets within one RTT; *(ii)* the sender retransmits packets not lost; *(iii)* the sender injects large duplicated bursts to the link. Except for VPN, they account for 20% to 50% cases in each dataset. Three examples are shown in Figures 10(a) to (c). In Figure 10(a), at $t_1 = 25.54s$, the sender retransmits 18 identical packets sent at $t_2 = 25.44s$. Note that the interval is $t_1 - t_2 = 0.1s$, less than $RTT = 0.2s$ that can be measured from the slow starts observed in the flow. In Figure 10(b), at $t = 11.27s$, the sender retransmits received packets (indicated by the ACKs) sent between $t = 10.58s$ to $t = 10.81s$, resulting in a large number of duplicated ACKs observed from $t = 11.84s$. In Figure 10(b), there is no observed duplicated ACKs that may trigger retransmission. In Figure 10(c),

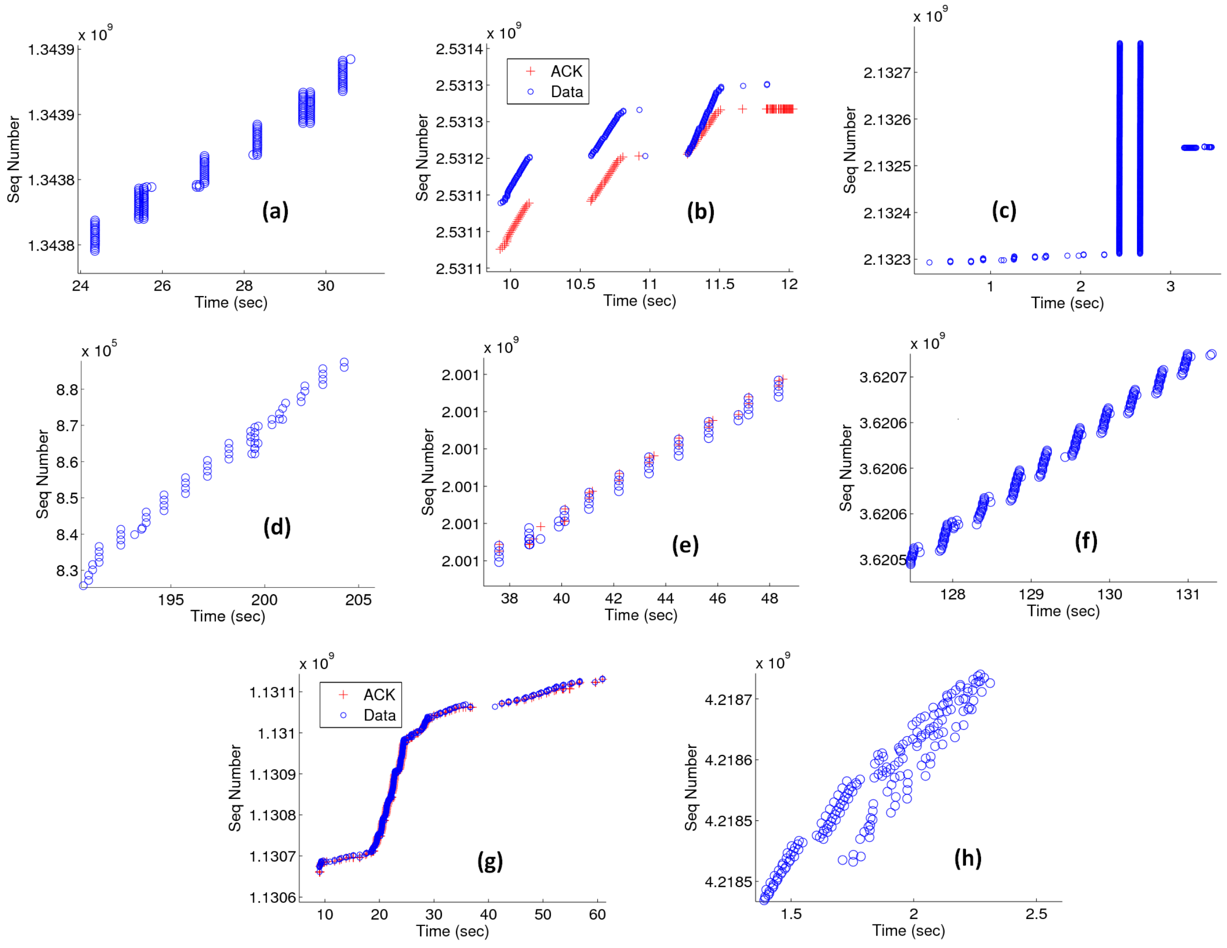


Figure 10: Examples of irregular retransmissions

the sender injects two large overlapped bursts of 500KB into the link at $t = 2.44s$ and $t = 2.66s$. Above behaviors may be caused by bugs or intentionally aggressive implementation of TCP.[9]

Category 2. *Rate limited by sender*. As discussed in §4.2, the sender does not fully utilize the congestion window even if the congestion window is reduced due to packet loss, since the sender does not produce data fast enough. Therefore, when the loss is detectable, the sender can possibly keep retransmitting packets without slowing down. Two examples are illustrated in Figures 10(d) and (e). We identify this category by observing *(i)* the flow clock is detected; thus we can separate the flow into flights based on the clock; *(ii)* the last packet of each flight is not transferred in MSS; *(iii)* the flow does not fall into Category 1 or 3. This category accounts for 30% to 50% of irregular flows for each dataset except for VPN. Note that the congestion window reduction described in RFC 2581 [6] is based on *FlightSize*, defined as is the amount of outstanding data on the wire. Even if an application is not fully utilizing current congestion window, a loss should also cause a reduction in the observed transmission rate. We therefore believe that this category also relates to non-standard TCP implementations.

[9]Nmap shows that flows in Category 1 are mostly generated by Linux 2.6.x and Windows 2003, while we also observe other OS versions such as firmware OS for embedded network devices (*e.g.,* routers). It is difficult for us to reproduce the buggy TCP behaviors because many servers return HTTP 4xx codes, and the bugs seem to be triggered nondeterministically.

Category 3. *Partial Overlap of Sequence Numbers.* Irregular flows in this category have strong frequency characteristics based on which we can separate the flow into flights. The flow exhibits a strange pattern that the sequence numbers of consecutive flights partially overlap. For example, in Figure 10(f), each flight contains $16 \le k \le 20$ packets; after sending flight i : $[m, m+k)$, the sender retransmits packet $m+k-c$ and the next flight starts from $m+k-c+1$. We observe 3 to 6 such flows in each unidirectional dataset.

Category 4. *Gaps or Rate change (false positives)*. The generated RTG shows a negative correlation due to gaps that were not removed, or due to a dramatic rate change. An example is shown in Figure 10(g). Before $t = 30s$, the sending rate and retransmissions are high; both decrease after $t = 30s$, causing the undesirable negative correlation. The overall false positive rate is 16/301.

Category 5. *Unknown cases*. It includes other cases that do not fall into the above four categories. We are unable to infer the cause of irregular retransmission, especially for unidirectional datasets. An example is shown in Figure 10(h).

6.3 Flow Clocks

We make four key observations from our analysis regarding flow clocks. In our datasets, *(i)* more than half of our flows do not have distinguishable flow clocks; *(ii)* a significant number of flows have non-RTT based flow clock around 100ms; *(iii)* flows with large non-RTT based flow clock tend to have more consistent flight size;

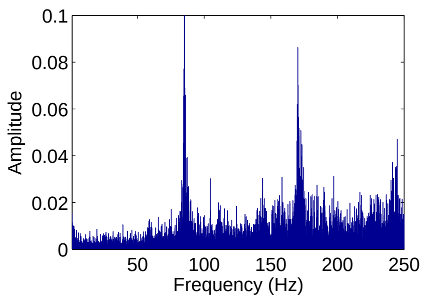


Figure 11: Freq. spectrum of a flow with a bottleneck link of 1Mbps

(iv) flows with non-RTT based clocks are more likely to transfer data with an inappropriately large congestion window (violating RFC 2581 [6] and RFC 2988 [30]) after a relatively long period of idle time.

RTT is a key parameter for understanding the origin of flow clocks. We tried three ways to estimate RTT: (1) measure the delay between SYN-ACK and first data packet; (2) measure the delay between SYN and first ACK packet; (3) measure the delay between first two flights in slow start. However, none of them yields satisfactory results, since (1) overestimates RTT for most flows due to the reason explained in §4.1; (2) may overestimate RTT for some flows due to delayed first ACK [19]; and (3) also overestimates RTT in many cases, compared with (2). Finally we picked the minimum value of (2) and (3) as an approximation of RTT. Such compromise requires bidirectional data, so we did not report RTTs for BU and BS1 to BS4. We clearly cannot use previous methods [33, 36] that implicitly assume RTT as flow clocks to calculate RTT.

Table 7 shows the existence of flow clocks in our datasets. For unidirectional traces BU and BS1 to BS4, more than half of the flows do not have distinguishable flow clocks; nearly 69% of flows fall into this category for VPN and DSL. For these latter two bidirectional datasets, we further classify flow clocks into RTT based and non-RTT based, using the empirically selected criteria that $|clock - RTT|/RTT < 20\%$. The ratio of RTT based and non-RTT based clocks are 1:0.73 and 1:1.44 for VPN and DSL, respectively.

Figure 12(a) plots the distribution of flow clocks. For each dataset, among flows with a measurable flow clock, about half of flows have clocks less than 150ms, while considerable number of flows have larger clocks up to 2000ms. We found a significant number of flows (15% for BU and BS1 to BS4, 10% for DSL) with flow clock around 100ms (10Hz). Based on Figure 12(b), which plots RTT and non-RTT based clocks for DSL[10], these clocks are mostly non-RTT based. By examining their IP and port numbers, we found that many of them are flows from video/audio streaming servers such as imeem, Pandora and streamtheworld[11]. Furthermore, in Figure 12(b), 30% of flows in DSL dataset have non-RTT based clock around 18ms. They are from a wide range of Web servers and CDN servers. However, we suspect that such non-RTT based flow clock is caused by the link layer, as it is known that if a flow's rate is limited by its bottleneck link, then the packets will be nearly equally-spaced [35]. In our controlled experiment, we created a bottleneck link of 1Mbps using a Linksys WRT54GL broadband router. As shown in Figure 11, HTTP downloading flows going through the 1Mbps bottleneck link exhibit flow clock around 12ms,

[10] Samples in VPN are too few to draw confident conclusion.
[11] Many IPs are from CDN servers, so we cannot infer their sources.

Table 7: Existence of flow clocks.

Has Clock	BU	BS1	BS2	BS3	BS4
Yes	42.5%	43.3%	44.7%	46.3%	45.9%
No	57.5%	56.6%	55.3%	53.6%	54.1%
VPN: No flow clk		68.5%		DSL: No flow clk	69.0%
VPN: RTT based clk		2.6%		DSL: RTT based clk	8.8%
VPN: non-RTT based clk		1.9%		DSL: non-RTT based clk	12.7%
VPN: Unknown RTT		27.0%		DSL: Unknown RTT	9.5%

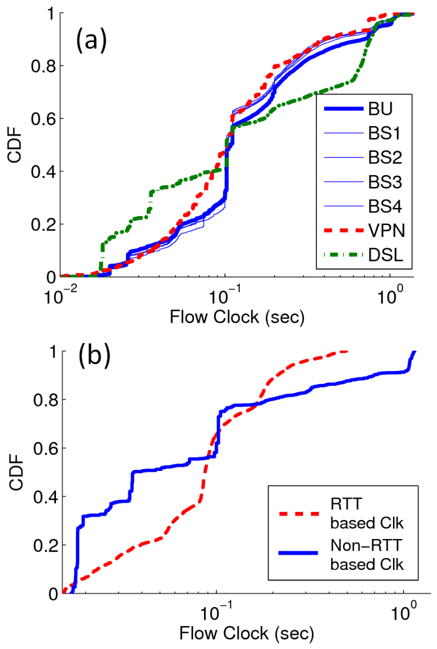


Figure 12: (a) Distribution of flow clocks (b) Distribution of RTT based clocks vs. non-RTT based clocks for DSL

which equals to the inter-packet time regardless of RTT. Finally, in Figure 12(b), 7% of flows with clocks larger than 1 second appear to be video streaming applications sending at constant bit rate.

Our next observation relates to the consistency of flight size. Recall that in §4.3, given that a flow has its clock, we separate the flow into *flights*, each of which consists of a train of packets sent within one clock. We find that, flows with large non-RTT based flow clock tend to have more consistent flight size. To quantify the consistency of flight sizes, we define a flow's *flight entropy* as: $E_F = -\sum_k \frac{N_k}{N} \log\left(\frac{N_k}{N}\right)$ (N: total number of flights, N_k: the number of flights containing k packets). Intuitively, a smaller E_F indicates that the flight sizes are more consistent (in all seven datasets, packet count and flow size are highly correlated with correlation coefficient higher than 0.99). The scatter plot in Figure 13 illustrates a trend that the flight entropy tends to decrease as flow clock increases, given that the flow clock is greater than 100ms. In each dataset, for flows with clock greater than 100ms, the correlation coefficients between flow clock and flight entropy lie between -0.5 and -0.3, since as flow clock increases, the proportion non-RTT based clock increases correspondingly, causing the decrease in the average flight entropy.

We further observe that flows with non-RTT based clocks are more likely to transfer data with an inappropriately large congestion window after a long idle period. Based on RFC2581 [6], if a TCP has no transmission for more than one retransmission timeout

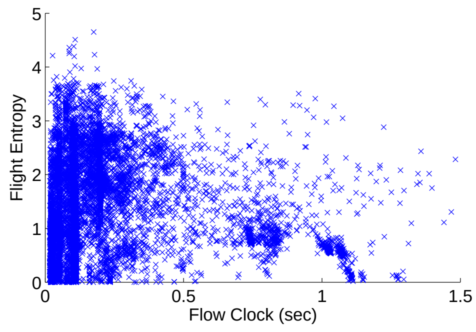


Figure 13: Correlation between flow clock and flight entropy for BU

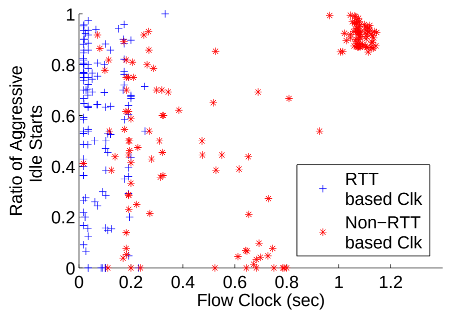


Figure 14: Scatter plot of flow clock and aggressive idle starts

(RTO), it should reduce the congestion window to no more than the restart window, which equals to initial congestion window, before next transmissions. The standard way to compute RTO is defined in RFC 2988 [30] as $RTO = RTT_{mean} + 4 \times RTT_{std}$, then rounded up to 1 second if needed[12]. To test whether the idle start behavior of a flow conforms to [6] and [30], we count the number of aggressive idle starts (*i.e.*, the server does not perform slow start after RTO), which is then divided by the total number of idle starts observed in the flow to get an aggressive ratio.

We selected 288 flows from DSL datasets where there exists at least 10 idle starts and the RTT can be estimated from TCP handshake. We calculate the aggressive ratio for each flow, based on an overestimation of RTO as $\max\{5 * RTT, 1.1sec\}$. The scatter plot of flow clocks and aggressive ratios is shown in Figure 14. On one hand, flows with non-RTT based clocks are more likely to perform aggressive idle start. In particular, among flows with aggressive ratio higher than 0.8, 75% have non-RTT based clocks. On the other hand, non-RTT based clocks with high aggressive ratio are mostly large, as 87% have clocks greater than 0.95sec, forming a cluster at upper-right corner of Figure 14. Most flows in the cluster originate from multimedia streaming servers. For flows with large non-RTT based clocks, a clear motivation to use a longer RTO is to keep the constant sending rate by avoiding slow start. However, by doing so, TCP can potentially send a large burst into the network after an idle period.

[12]Based on RFC 2988, exceptional cases include: RTO should be set to 3 seconds before first RTT is estimated, and RTO must be doubled when retransmission timer expires due to packet loss.

6.4 Summary of Results

We summarize our findings as follows. *(i)* The majority of the flows have ICW of 2*MSS. However, from 2.3% to 15.8% flows in our data have a large ICW violating RFC 3390. Almost all aggressive flows are from two open source OSes: Linux 2.6.x and FreeBSD 6/7. *(ii)* Among flows with high retransmission rate (higher than 10%), 5% exhibit irregular retransmission behavior, which is observed to have two main causes: abnormal retransmission not conforming to RFC-compliant TCP specifications, and under-utilization of the congestion window. *(iii)* Less than half of our flows have distinguishable flow clocks. Among flows with a measurable flow clock, up to 60% have clocks originated by non-RTT factors. In particular, we observe several clusters of clocks such as 100ms and 18ms differing significantly from RTT values. We found that besides RTT, many factors such as user interaction, application defined software clocks, periodical OS events (*e.g.*, keyboard scanning), or "retiming" effects of a bottleneck link may shape or blur the flow clock. *(iv)* Flows with large non-RTT based flow clock tend to have more consistent flight size. Also, flows with non-RTT based clocks are more likely to transfer data with an inappropriately large congestion window due to a larger RTO. Both observations are motivated by keeping constant sending rate at the application layer.

7. CONCLUSION

Given the critical importance of the TCP protocol for shaping the traffic characteristics on the Internet, our work reexamines key properties of TCP behavior observed on the Internet today, using traces collected at multiple vantage points from a tier-1 ISP. We reveal the evolution of TCP's characteristics by comparing with two previous studies conducted 6 to 8 years ago. Furthermore, we go beyond the basic characterization to study within-flow packet dynamics. In particular, we studied three problems: how to determine the initial congestion windows of TCP senders; how to capture sender's change in sending rate in response to packet retransmission; how to accurately obtain TCP flow clocks. To answer these questions, we have designed several novel methodologies, especially addressing the challenges of analyzing passively collected unidirectional TCP flows. By applying our methods on long-lived flows in our datasets, we characterized the popular TCP behavior, and identified unexpected flows not conforming to TCP specifications as well. Our findings also suggest that the popularity of TCP's use for streaming and gaming applications would greatly change the traffic dynamics especially because most flows with non-RTT based flow clocks are found to belong to this application class. Our study is an important step towards better understanding Internet traffic dynamics, ensuring protocol conformance, and understanding the interaction between the transport layer and the application layer.

8. REFERENCES

[1] Detours, Binary Interception of Win32 Functions. `http://research.microsoft.com/en-us/projects/detours/`.

[2] Nmap, Free Security Scanner for Network Exploration and Security Audits. `http://nmap.org/`.

[3] p0f, a Versatile OS Fingerprinting Tool. `http://lcamtuf.coredump.cx/p0f.shtml`.

[4] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A Five-Year Study of File-System Metadata. In *Proc. of USENIX Conference on File and Storage Technologies (FAST)*, 2007.

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

[6] M. Allman, V. Paxson, and W. R. Stevens. TCP Congestion Control. RFC 2581, 1999.

[7] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft. Pop-level and Access-link-level Traffic Dynamics in a Tier-1 POP. In *Proc. of Internet Measurement Workshop*, 2001.

[8] V. Cerf, Y. Dalal, and C. Sunshine. Specification of Internet Transmission Control Program. RFC 675, 1974.

[9] C. Chambers, W. chang Feng, S. Sahu, and D. Saha. Measurement-based Characterization of a Collection of On-line Games. In *Proc. of Internet Measurement Conference (IMC)*, 2005.

[10] K. chan Lan and J. Heidemann. Measurement Study of Correlations of Internet Flow Characteristics. *Computer Networks*, 50, 2006.

[11] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing Residential Broadband Networks. In *Proc. of Internet Measurement Conference (IMC)*, 2007.

[12] C. Estan, S. Savage, and G. Varghese. Automatically Inferring Patterns of Resource Consumption in Network Traffic. In *Proc. of ACM SIGCOMM*, 2003.

[13] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. In *ACM Transactions on Computer Systems (TOCS)*, 2003.

[14] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. In *ACM Computer Communication Review*, 1996.

[15] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649, 2003.

[16] S. Ha, I. Rhee, and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In *Proceedings of the third PFLDNet Workshop*, 2008.

[17] S. Jaiswal. *Measurements-in-the-Middle: Inferring end-end path properties and characteristics of TCP connections through passive measurements*. PhD thesis, University of Massachusetts Amherst, 2005.

[18] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP Connection Characteristics through Passive Measurements. In *Proc. of IEEE INFOCOM*, 2004.

[19] H. Jiang and C. Dovrolis. Passive Estimation of TCP RoundTrip Times. In *ACM Computer Communication Review*, 2002.

[20] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. Measurement and Analysis of Large-Scale Network File System Workloads. In *USENIX Annual Technical Conference*, 2008.

[21] G. Lu and X. Li. On the Correspondency between TCP Acknowledgment Packet and Data Packet. In *Proc. of Internet Measurement Conference (IMC)*, 2003.

[22] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. In *ACM Computer Communication Review*, 2002.

[23] A. P. Markopoulou, F. A. Tobagi, and M. J. Karam. Assessment of VoIP Quality over Internet Backbones. In *Proc. of IEEE INFOCOM*, 2002.

[24] A. Medina, M. Allman, and S. Floyd. Measuring Interactions Between Transport Protocols and Middleboxes. In *Proc. of Internet Measurement Conference (IMC)*, 2004.

[25] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-Time Signal Processing (2nd Edition)*. Prentice Hall, 1999.

[26] J. Padhye and S. Floyd. Identifying the TCP Behavior of Web Servers. In *Proc. of ACM SIGCOMM*, 2001.

[27] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *ACM Computer Communication Review*, 1997.

[28] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *Proc. of ACM SIGCOMM*, 1997.

[29] V. Paxson. End-to-end Internet Packet Dynamics. In *ACM Computer Communication Review*, 1997.

[30] V. Paxson and ark Allman. Computing TCP's Retransmission Timer. RFC 2988, 2000.

[31] A. Shaikh, J. Rexford, and K. Shin. Load Sensitive Routing of Long-lived IP Flows. In *Proc. of ACM SIGCOMM*, 1999.

[32] K. Thompson, G. J. Miller, and R. Wilder. Wide-area Internet Traffic Patterns and Characteristics. In *IEEE Network Magazine*, 1997.

[33] B. Veal, K. Li, and D. Lowenthal. New Methods for Passive Estimation of TCP Round-Trip Times. In *Proc. of Passive and Active Measurement conference (PAM)*, 2005.

[34] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. FAST TCP: motivation, architecture, algorithms, performance. In *Proc. of IEEE INFOCOM*, 2004.

[35] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the Characteristics and Origins of Internet Flow Rates. In *Proc. of ACM SIGCOMM*, 2002.

[36] Y. Zhang and Z. Lei. Estimate Round-Trip Time of TCP in a Passive Way. In *Proc. of International Conference on Signal Processing (ICSP)*, 2004.