Explaining BGP Slow Table Transfers: Implementing a TCP Delay Analyzer

Technical Report #110020
UCLA Computer Science Department
Sep, 2011

Pei-chun Cheng*, Jong Han Park *, Keyur Patel †, Shane Amante ‡, Lixia Zhang *
*University of California, Los Angeles, {jpark,pccheng,lixia}@cs.ucla.edu
†Cisco Systems, Inc. keyupate@cisco.com
‡Level-3 Communications Inc. shane.amante@level3.com

***Abstract*— Although there have been a plethora of studies on TCP performance in supporting of various applications, relatively little is known about the interaction between TCP and BGP, which is a specific application running on top of TCP. This paper investigates BGP's slow route propagation by analyzing packet traces collected from a large ISP and RouteViews Oregon collector. In particular we focus on the prolonged periods of *BGP routing table transfers* and examine in detail the interplay between TCP and BGP. In addition to the problems reported in previous literature, our study reveals a number of additional TCP transport problems, that collectively induce significant delay. Furthermore, we develop a tool, named T-DAT, that can be deployed together with BGP data collectors to infer various factors behind the observed delay, including the BGP's sending and receiving behavior, TCP's parameter settings, TCP's flow and congestion control, and the network path limitation. Identifying these delay contributing factors makes an important step for ISPs and router vendors to diagnose and improve the BGP table transfer performance.**

***Index Terms*—BGP; TCP; Delay Analysis; Measurement**

I. Introduction

The routing infrastructure of today's Internet is glued together by the Border Gateway Protocol (BGP) [25]. BGP routers establish BGP sessions with their neighbors to exchange routing information and maintain the global reachability. A BGP session runs over a TCP connection that carries routing updates to reflect reachability changes. In general neighboring BGP routers are connected by high speed lines, thus the BGP routing update exchanges are expected to be very fast in general. However, both the research and operation communities have reported alarmingly long delays (i.e., up to tens of minutes) in *BGP table transfers*, the particular massive BGP updates triggered by BGP session establishments or resets [9, 15, 32, 33, 36]. It remains unclear what are the basic causes behind these slow table transfers or how to fix the problem.

Several recent studies have looked into the BGP behavior via its interactions with TCP. Xiao et al. [35] show that BGP performance could be seriously degraded upon repeated TCP retransmissions due to network congestion, and lead to BGP session failures. Zhang et al. [39] demonstrate that, even without network congestion, the durations of BGP table transfers can increase up to an hour under specific low-rate TCP DoS attacks. By investigating TCP packet traces, Houidi et al. [15] report an undocumented BGP timer-driven implementation, which potentially accounts for more than 90% of table transfer time. While these previous works help explain slow BGP data transfers in individual settings, and highlight opportunities to reveal BGP problems by studying the TCP behavior, there remain open issues about their prevalence and impact on today's BGP operations. In particular, there has been no practical way to answer questions raised by network operators: "Are my table transfers suffering from these reported problems? Are they significant? Are there other problems specific to my particular networks?", to list a few.

In this paper, we use TCP and BGP monitoring data traces collected from a large ISP and the RouteViews project [3] to identify potential causes for the slow BGP table transfers. We also develop a tool which *analyzes TCP bi-directional traces and explains the delay experienced during the BGP table transfer*. More specifically, the tool infers and attributes the table transfer delay to different factors including BGP sender and receiver behavior, TCP parameter settings, TCP congestion and flow control, and network path limit. Our tool is inspired by the pioneering works on TCP rate analysis [28, 38]. However this work is distinct from the previous ones in that BGP, being a routing protocol, concerns data transfer *delay* rather than transmission *rate* as the main performance measure, which requires a new approach of tracking TCP behavior in the time domain.

The significance of understanding the BGP table transfer delay is twofold. From the operation perspective, knowing the causes of the delay helps ISPs and router vendors diagnose and improve the performance of their BGP sessions. From the perspective of passive BGP monitoring efforts like RouteViews, these various transfer delays (when they exist) introduce measurement artifacts to the collected BGP data, potentially leading to inaccurate analysis results.

Our contributions in this work are summarized as follows. First, we show that BGP table transfers are slow, based on BGP

monitoring data in a large ISP and RouteViews. By investigating TCP packet traces, we further find on-going TCP transport problems, which induce delays of different magnitude in table transfers (Section II). Some identifed transport problems are due to router implementation or features, which shall affect both BGP monitoring and general operations. Second, we develop a new tool, T-DAT, to identify and measure different factors behind the transfer delay (Section III). We then demonstrate the tool's usage and look at the preliminary answers on explaining durations of slow table transfers (Section IV).

We emphasize that, given the scale and heterogeneity of the BGP distributed network, our results deem not to answer all the questions of slow BGP table transfers in the wild. However our new tool enables systematic analysis of the BGP table transfer behavior. The tool uses TCP packet traces, which can be collected passively and requires no modification to the BGP operation. Also note that although this work is driven by BGP performance analysis, T-DAT may be potentially used for other TCP-based applications. For the rest of this paper, we discuss the limitation and the applicability of this work in Section V, related works in Section VI, and conclude the paper in Section VII.

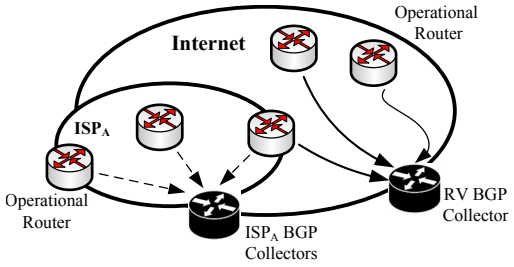


Fig. 1. ISP$_A$ and RouteViews BGP monitoring.

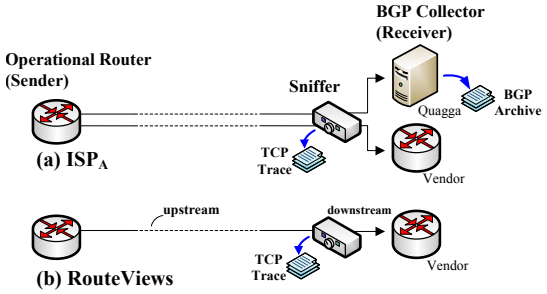


Fig. 2. BGP/TCP data collection.

II. Inspecting Slow Table Transfer

This section starts off our search for the reasons behind BGP slow table transfer. Compared to previous works based solely on BGP data [9, 26, 32, 36], we seek to analyze the TCP packet traces, with the goal to reveal distinct *transport protocol* issues. More specifically, we investigate the TCP dynamics in the *initial BGP routing table transfer* [25]. The duration of such a transfer represents the elapsed time spent to boostrap and converge the routing state between neighboring BGP routers.

A. Datasets and Methodology

We use data collected at a large ISP (ISP_A) and the RouteViews [3] project. As depicted in Figure 1, *BGP data collectors* are deployed to peer with operational routers and passively receive BGP messages.[1] The collector could be a PC-based Quagga router[2] or a vendor router. The Quagga collector records the received BGP updates in the Multi-threaded Routing Toolkit (MRT) [5] format, which has been widely used in studying BGP behavior [2, 3]. The Vendor collector works as a *looking glass* and mainly allows operators to log in and look up the current routing information. As shown in Figure 2, in addition to BGP collection, a TCP packet sniffer (tcpdump) is deployed in front of the collector, and records the pass-through traffic in *both* directions. The whole packet, including the headers and data, is captured. We notice that tcpdump can sometimes drop packets and leaves void periods in the trace. We exclude those periods from the following analysis. Note that the collectors do *not* announce routing information; thus, only the packets from the operational routers to the collectors carry actual BGP updates. In this work, we refer an operational router as *Sender*, the BGP collector as *Receiver*, and the TCP sniffer simply as *Sniffer*. The TCP connection of a BGP session carries both the Sender-to-Receiver *data packets* and Receiver-to-Sender *ACK* packets. From the data flow and Sniffer's perspective, the network path between Sender-Sniffer and Sniffer-Receiver is considered the *upstream* path and the *downstream* path respectively.

Table I summarizes the characteristics of the collected traces. We further separate the ISP_A traces based on the collector type. For each trace, we pinpoint the periods of BGP table transfer with the following steps. *(i)* From the tcpdump trace we first extract individual TCP connections, together with basic connection profiles, including the *connection start time*, *end time*, *estimated round-trip time (RTT)*, *maximal segment size (MSS)*, etc. *(ii)* Based on the TCP connection start time, which also indicates the *start* of the BGP table transfer,[3] we then turn to the BGP archive and apply MCT (Minimum Collection Time) algorithm [36] to identify the *end* of BGP table transfer.[4] *(iii)* For the vendor traces that do not offer the BGP data archive, we develop a side tool, pcap2bgp, to reconstruct TCP data stream from the raw packet trace. This side tool could run either online or offline and takes care of the TCP out-of-order delivery and retransmissions. Compared with wireshark [4] or tcpflow [13], pcap2bgp further extracts and saves individual BGP messages from the constructed TCP byte

[1]RouteViews deployed multiple collectors across the Internet. The collector described in this work is located in Univ. of Oregon, Eugene, USA

[2]Quagga is a software based routing suite, which implements routing protocols such as OSPF and BGP, and is widely used in monitoring BGP behavior [1]

[3]A BGP table transfer starts right after establishing the TCP connection [25].

[4]Different from [36], which runs MCT on every BGP message and is intractable in this work due to a huge data volume; here we use TCP start time as an indicator to quickly locate the occurrences of a table transfer, and only use MCT to estimate the duration of the BGP table transfer. This greatly reduces the processing time.

TABLE I
SUMMARY OF BGP/TCP DATASETS AND IDENTIFIED BGP TABLE TRANSFERS

Trace Name	Type	Duration	Collector	# Pkts/Bytes (M/GB)	# Rtrs	TCP (tcpdump)	BGP (MRT)	# BGP Tables Transfers
ISP_A-1	iBGP	2008.05 $\sim$ 2009.04	Vendor	1023 / 218	24	Yes	-	10396
ISP_A-2	iBGP	2008.05 $\sim$ 2009.04	Quagga	909 / 138	27	Yes	Yes	180
		2009.09 $\sim$ 2010.09		1296 / 219				219
		2010.11 $\sim$ 2011.01		492 / 81				37
RV	eBGP	2010.11 $\sim$ 2011.01	Vendor	176 / 47	59*	Yes	-	94

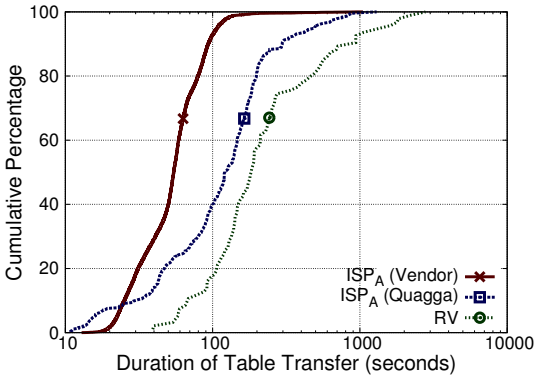


Fig. 3. CDF of table transfer duration

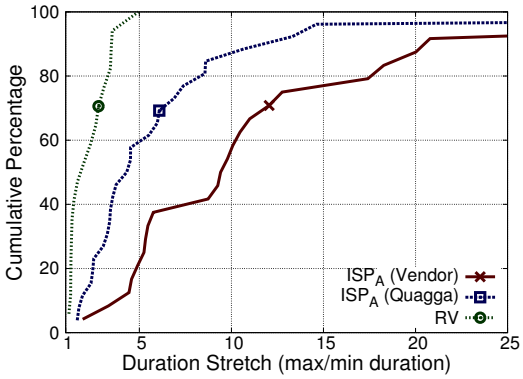


Fig. 4. Stretch of table transfers

stream. Then we apply MCT on the extracted BGP messages as in the previous step.

B. BGP Transport Problems

Table I lists also the number of identified BGP table transfers, ranging from tens to a few hundreds in each trace. One exception is the ISP_A-1 (Vendor) trace, which contains an alarmingly high number of table transfers. We confirmed with the operator that this is due to a vendor bug which triggered frequent BGP session resets. Figure 3 shows the CDF of the table transfer duration. The majority of the table transfers finished within a few minutes. The table transfers of the ISP_A (Quagga) and RouteViews tend to take longer time to finish, with 50-percentile at 2.5 minutes and 80-percentile at 5 minutes. We can also observe that some table transfers are taking longer than 10 minutes. This is generally slower than one would expect: considering the amount of data to send (i.e., 5 $\sim$ 8 MB for the full BGP table) and the underlying link bandwidth (i.e., up to tens of Gbps in ISP_A), table transfers shall finish mostly in a few seconds [15]. Previous works have made similar observations that table transfers can take even up to tens of minutes [9, 15].

Note that each router's table transfer duration can be different due to its distance (hops, RTT) to the collector. For each router-collector pair that has more than two table transfers, we calculate the *stretch ratio*, defined as the longest table transfer duration divided by the shortest one. We check and make sure that these two transfers carry similar amount the data. A high ratio indicates that the table transfer duration is significantly stretched, for some reason, while sending the same table. Figure 4 shows the results. We observe that in general, a router could send a routing table 2 to 5 times slower compared to its own fastest one (22%, 59% and 100% respectively for the ISP_A-1, ISP_A-2, RV traces). The stretch could be more than an order of magnitude for the distribution tail.

TABLE II
OBSERVED TRANSPORT PROBLEMS

	Observation	Potential Cause	Num.
1	Gaps in table transfers	Timer implementation [15]	25
2	Consecutive retransmission	Bursty BGP dynamics [22]	58
3	BGP peer-group blocking	BGP scaling feature [37]	15
4	Misc. issues	Bugs, delay acks, etc	-

To identify potential causes of the slow times, we inspect the TCP packets exchanged in the table transfer. Given that it is nearly impossible to check *all* table transfers individually, we take for each router, slow table transfers, whose duration are longer than the average transfer duration plus three standard deviations. If no such slow transfer exists, the router's slowest table transfer is selected instead. We ended up with investigating of 172 table transfers.

Table II lists the transport issues we identified from the table transfers. For each problem, we discussed with the operators and vendors to find the root causes, which range from implementation bugs to router specific features. In the following discussions, we skip miscellaneous minor problems due to the limited space. We emphasize that the analysis in this section ***does not intend to be complete nor system-wise prevalent***, as we only study the sample table transfers, and our dataset based on BGP monitoring settings inherently represents a limited view of the entire BGP network. Our goal is to demonstrate the real on-going problems that otherwise went unnoticed, and discuss their impact on BGP research and operations, which highlights the necessity of a new tool for systematic analysis.

1) Gaps in Table Transfers: Houidi et al. [15] investigate the slow table transfer problem. They found that *the sender regularly stops sending routes to the receiver and creates gaps in the table transfer* in a VPN provider backbone. Through further experimenting in a testbed with routers from 3 vendors,

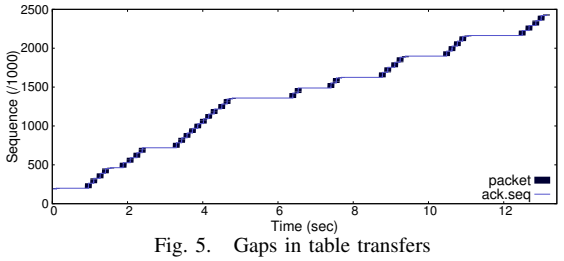


Fig. 5. Gaps in table transfers

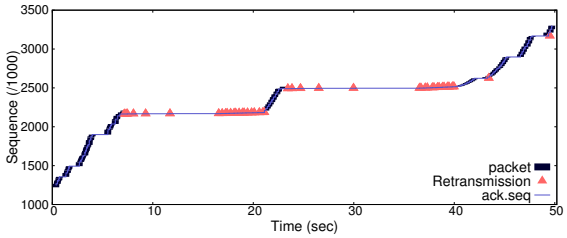


Fig. 6. Consecutive packet losses / retransmissions

they show that the gaps are caused by the undocumented timer-driven router implementation, which results in sending a limited number of messages per timer expiration.

In our dataset, we make similar observations for 25 sample table transfers. Figure 5 shows an example piece of one BGP table transfer that contains the prolonged gaps (i.e., much longer than the RTT) between packet transmissions. However the distribution of gap length is less regular compared to the timer values reported in [15]. We observe various gap lengths of tens of milliseconds up to a few seconds. Note that table transfers are captured from operational networks, and could be affected by various factors, including the router load, end-to-end network path, traffic level, etc. Moreover, we checked multiple table transfers from the same router or across different routers, and find that the presence of the gaps is not always pronouncing. Compared to [15] which shows that gaps can represent more than 90% of table transfer time, instead we observe that table transfers could be still slow without suffering from such timer gaps. This motivates our search on other explanations for the slow transfer described in the following sections.

2) Consecutive Retransmissions: In our dataset, another common observation is the consecutive TCP retransmissions in a short period of time, and we check that the retransmissions are caused by packet loss. Figure 6 shows an example of TCP connection that experiences two episodes of consecutive packets retransmissions. Table III lists BGP updates received during the first retransmission episode. Note that the router attempted to send all these updates at the same time at 1235728587 (unix timestamp), but due to packet retransmissions, they arrive at the receiving BGP with different delay, from 1 to 13 seconds. Without inspecting the packet trace, these delay gaps could be falsely attributed to the result of BGP protocol dynamics.

Generally, multiple packet losses could occur along the congested network path. Here, we further differentiate packet losses that happen *locally* to the receiver, but not somewhere deep in the network. This is made possible due to the fact that *Sniffer* is immediately next to *Receiver*. To find local losses,

TABLE III
RETRANSMISSION DELAY OF BGP UPDATES (SECONDS)

Timestamp	Delay	Prefix	Path
1235728588	1	66.154.112.0/24	19080 22298 30092
1235728588	1	66.154.104.0/22	19080 22298 30092
...			
1235728592	4	138.247.0.0/16	1239 13576 14263 23122
1235728592	4	205.151.56.0/24	174 16532
...			
1235728597	9	206.209.232.0/21	7018 16910
1235728597	9	219.239.44.0/23	10026 7497 7497 7497 17964
...			
1235728601	13	92.255.72.0/22	8342 20632 47168

we first check whether a packet is lost between Sender and Sniffer (i.e., upstream marked in Figure 2), or Sniffer and Receiver (i.e., downstream), respectively. The idea is based on classifying the packet retransmissions [17]: if a retransmission is due to the loss between Sniffer and Receiver, then Sniffer would first see a packet that is not acknowledged in time by Receiver.[5] Later, the Sender sends another packet with the same sequence number. We then mark the second packet as a retransmission due to *downstream losses*. Given that Sniffer is simply co-located with Receiver, these downstream losses shall occur *locally*, either at the Sniffer-to-Receiver link or at Receiver's interface. Figure 7 depicts an example connection. As shown in the figure, the sniffer sees a complete packet flight (the left-most one), but multiple packets are lost between the sniffer and the receiver (i.e., the Receiver only acknowledges up to half of the flight). This triggers multi-rounds of successive retransmissions.

On the other hand, if a retransmission is due to the loss between Sender and Sniffer, the sniffer would not see the dropped packet, but many out-of-order packets following the missing sequence gap. Figure 8 depicts such an example connection. We then mark these retransmissions as due to *upstream losses*. However, in this case, we could not further tell whether the packets are lost at the sender side or along the path. Out of the 172 sample table transfers, there are 58 consecutive retransmissions and 49 and 37 of them contain upstream and downstream (receiver-local) losses respectively.

The problem of BGP scalability and local losses has long been recognized [12, 22, 37]. In large networks, a BGP router peers with tens and up to hundreds of neighboring routers [10, 22]. Upon massive route changes (i.e., router or link failures, scheduled maintenance, etc.), the router could send thousands of route updates to all its peers at the same time. In our dataset, this usually results in tens of thousands of packet exchanges (e.g., around 57K packets in one sample instance). This can result in sustaining packet drops on router interfaces [11]. To alleviate this situation, router vendors suggested to increase interface buffer size based on the number of BGP peers. But as BGP peering is used with an ever-growing number of neighbors to advertise an ever-growing number of routes, the buffer space required may still increase far beyond the available router resources. Note that this problem is not specific to BGP. Recent works report

[5]Either because the packet is lost from Sniffer to the Receiver, or the ACK is lost in the opposite direction

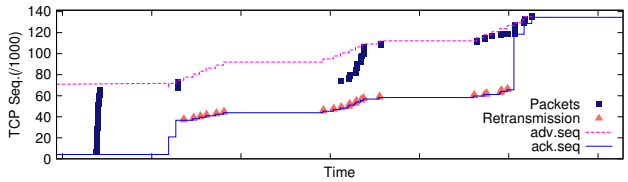


Fig. 7. Downstream (Receiver-local) consecutive losses

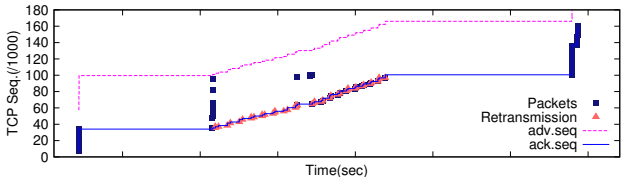


Fig. 8. Upstream consecutive losses

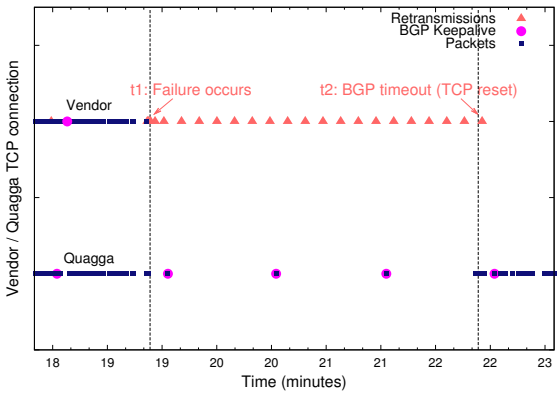


Fig. 9. Session failures and Peer-Group blocking

similar phenomenon of *TCP incast congestion* in datacenters, when multiple synchronized servers send data to the same receiver concurrently [8, 30]. Here, the impact of consecutive retransmissions on BGP has long been overlooked due to lack of available data to the community.

3) BGP Peer Group Blocking: This section describes an interesting syndrome captured specifically in the ISP_A settings. During the measurement period of May 2008 to April 2009, each operational router is configured to peer with both the Quagga and Vendor collector. From the traces, we observe that two connections proceed in a lockstep. That is, even with more pending updates to send, the faster connection often pauses and waits for the slower one to catch up. We verified with the vendor and find that this is due to a specific BGP *peer-group* feature [37]. The purpose is to group together peers with identical outbound policies. The router then generates routing updates once, places in a common queue, and simply replicates the updates to all group members' TCP connections. Note that the queued common updates would be cleared only after being successfully delivered to all peers. This reduces the router processing load, but with the cost that the whole group is now dragged down by the slowest member.

Our observation shows that the peer-group delay is generally in the order of milliseconds, but it could be pathologically long upon connection failures as depicted in Figure 9. At t_1, an error occurred at the Vendor collector, causing the router to keep retransmitting packets, but never being acknowledged[6] till the faulty BGP session eventually timed out at t_2. We can see that, during the whole retransmission period of the Vendor connection, the router also stopped the transmission of the Quagga connection. The Quagga connection immediately resumed after the Vendor connection timed out and was removed from the peer group. Based on the BGP keep-alive and hold-down [25] timer setting in ISP_A, the timeout took 180 seconds ($t_1 \sim t_2$) in this example and could have significantly hampered the BGP convergence.

C. Lessons Learned

In the previous sections, we report transport problems that impact the BGP table transfer performance. We observe that these problems went unnoticed in ISP_A and RouteViews for months or even years. Here, TCP plays an important role in reliably delivering BGP messages, and hides from BGP the details of various lower layer bugs, bad parameter settings, transient network congestions, etc. As a result, BGP only gets to see the prolonged delay in update arrivals. However, this could be undesirable in that people may *falsely attribute the transport-induced delay to BGP's distributed nature*, and draw questionable conclusions for the BGP convergence behavior, and overlook the necessity to address the underlying transport problems.

We reiterate, to ensure our point is clear, that limited by the specific datasets, this work does not target at finding *all* possible BGP transport issues. Instead, the problems reported in this study, together with those of previous works [15, 35, 39], serve as hard evidence that motivates us to answer the fundamental question: how to *detect* and *quantify* transport problems more efficiently without the tedious trace inspection? This is challenging as we learned that there exist various reasons behind transport problems, which could result in complex interaction between BGP and TCP.

In this work, we propose a *reactive* approach. That is, instead of searching for a seemingly impossible way to recognize unexpected transport problems, we are focusing on staying alert to their *consequences* (i.e., suspicious delays) in the packet trace. We develop a delay analysis tool to systematically measure and classify the transfer delays in TCP packet traces. Users' further attention and investigation is only needed when a significant and suspicious delay factor has been reported.

III. TCP Delay Analyzer

In this section, we describe a new analysis tool that can capture the TCP connection behavior and identify factors that delay the data transfers. The idea is inspired by TCP rate analysis [28, 38], which classifies the rate limit of TCP connection by application, TCP end-points, and network paths. Based on a similar taxonomy, our goal is an analysis tool which focuses on the different metric, namely *transfer delay*, and identifies major contributing factors. We will discuss the major difference between previous rate analysis and T-DAT later in Section VI.

[6]We suspect that it is due to a software bug

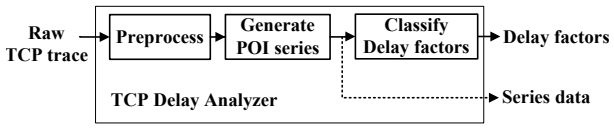


Fig. 10. High level design

The basic idea of T-DAT is to to transform the ralative data and ACK arrivals into multiple event series, and from the series we infer the reasons behind transfer delay. This requires one basic assumption that TCP use congestion and receive windows to control packet delivery (i.e., TCP flavours such as Tahoe, Reno, New Reno), which is generally true for commercial routers which do not emply exotic TCP implementations. Note that this is a common assumption made in TCP rate analysis studies [28, 38]. It is subjective to our future work to extend T-DAT to support other TCP variations.

Figure 10 depicts the high-level tool operations. The delay analyzer first pre-processes the raw packet trace, collects the connection level information, and restructures the trace if necessary (§III-B). Then, the packet trace is transformed into multiple event series, each is designed to represent one specific type of TCP connection behavior (§III-C). Based on the event series, the tool measures the induced delay and classifies the delay factors similar to [38] (§III-D). We call the tool T-DAT (TCP Delay Analysis Tool), named after T-RAT in [38], to make a simple yet clear distinction.

A. Data Structure for Delay Analysis

In this section we first introduce an important time-range based data structure used by the tool. From the raw packet trace, consider the arrival of each packet (including data and ACK) as an *event* that potentially affects diffrent *types* of TCP end-point behavior, such as a packet loss that triggers retransmissions, or an ACK that changes the advertised window size. We represent each event with the 2-tuple notation $(event\_duration, event\_data)$. The $event\_duration$, represented as $[start\_time, end\_time]$, records the event start and end time in microseconds. The second field, $event\_data$, is a reference pointing to the detail trace data. Events of the same type are then organized in *an ordered set of time durations*, i.e., a special set container in which each element is a continuous time duration (or time range). We name these ordered sets as *event series*.

One way to visualize the series is to present them using binary square curves. Figure 11 gives a preview of the graphical output of the tool. Here, the figure includes an example piece of packet trace and multiple derived series, which represent different TCP behaviors. For instance, the series *UpstreamLoss* captures the retransmissions due to upstream packet losses. Each of 9 packet retransmissions (shown as red triangles) in the TCP trace is represented by a corresponding time range (shown as 9 square waves), and the duration of each wave indicates the retransmission delay introduced to the TCP connection. In addition, each wave records the actual number of retransmitted packets and bytes within itself (not shown in the figure). We will describe in detail how to generate these series in Section III-C.

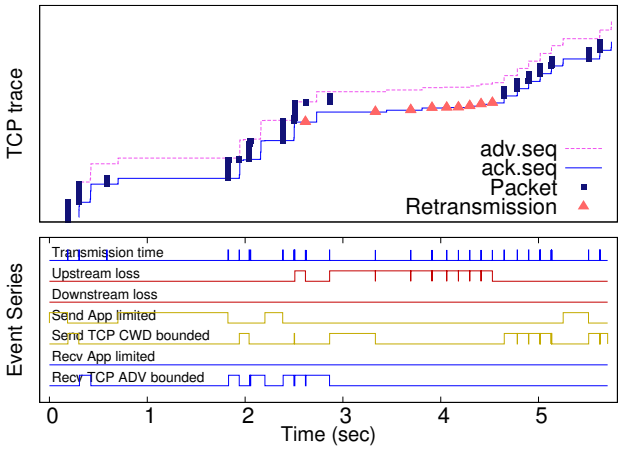


Fig. 11. Example TCP trace and event series

Note that the set-based data structure enables both the high-level quantification and detailed inspection. On one hand, measuring the coarse-grained transfer delay induced by a particular series is now equivalent to calculating the *set size (or set cardinality)*, which has been widely supported by software libraries. On the other hand, the series faithfully preserves the exact packet timing information as the raw trace. This provides essential cross-reference when we make interesting high-level observations, and decide to further investigate in depth the TCP packet trace.

B. Input: TCP Packet Trace

The analyzer takes as input the raw packet traces in pcap format, together with connection level parameters, including the maximum segment size (MSS), round trip time (RTT), maximal advertised window size,[7] which we extract using tcptrace [21]. We also use tcptrace to label packets such as retransmissions, out-of-sequence, and duplicates.

1) Accommodate the Sniffer Location: For the dataset used in this work, one important limitation is that the sniffer is close to the receiver end, while the data transfer by and large depends on the sender behavior. This is not a new problem and the impact of the sniffer location on interpreting the TCP trace has been widely acknowledged [16, 28, 38]. In previous works, the major concern was how to estimate RTT at different sniffer locations.

Figure 12 illustrates the common idea of RTT estimation. When the sniffer is in the middle of the path, the sender's true perceived RTT (in the left of Figure 12) is inferred as the sum of *d1* and *d2*; each represents one part the round-trip delay for Sniffer-to-Receiver and Sniffer-to-Sender [17]. Building upon this approach, our processing in this step to *shift forward* the ACKs with the offset *d2* to match their corresponding data packets (e.g., $ACK_1 \rightarrow ACK_1$'), such that the resulted new trace (i.e., the original data packets with shifted ACKs) approximates the sender-side behavior.

[7] advertised by the receiver

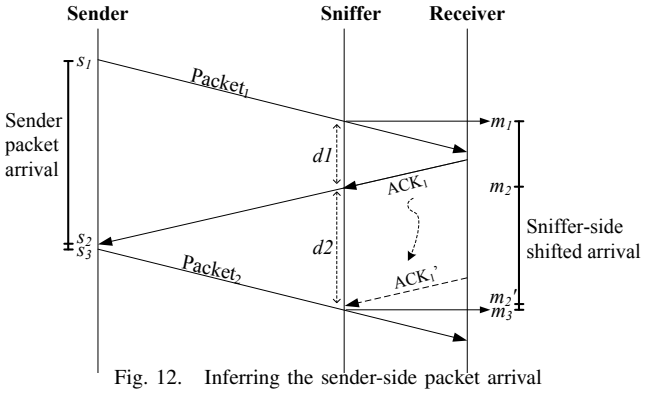


Fig. 12. Inferring the sender-side packet arrival

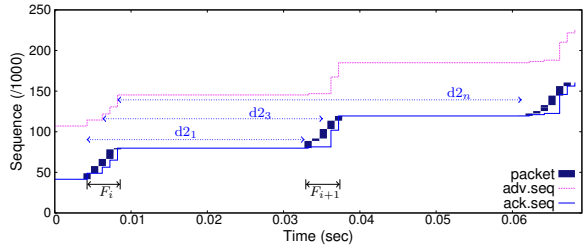
As shown in Figure 12, the goal is to rewrite the *packet-ack-packet* arrival at the Sniffer from m_1-m_2-m_3 to m_1-m_2'-m_3, which more accurately reflects the sender-side arrival s_1-s_2-s_3. Unfortunately, measuring *d2* is challenging. Multiple ACK and data packets may be concurrently in transmission, and often there is no clear association between the ACK and the following data packets [17, 20, 28, 31],

Our observation is that, *it could be easier and more accurate to measure d2 for a group of ACKs, instead of each individual ACK*. As the term *flight* is commonly referring to data packets, here we use it to refer to ACKs that are sent back to back within a group. In Figure 13(a), we mark a flight of n ACKs, F_i, together with their estimated *d2* delays, $d2_1$, $d2_2$, $\dots d2_n$. Note that for $d2_1$ and $d2_2$, the estimation is relatively accurate, as these ACKs explicitly free the window space, which is soon filled by the data packets in the next round trip. On the other hand, $d2_n$ is rather a loose estimation; the n_{th} ACK could arrive anytime between 0.04 $\sim$ 0.06 second and still leads to the same packet arrivals. Thus, the idea is to shift the whole ACK flight with the most precise (shortest) $d2$ of each ACK in the flight. The algorithm is summarized as follows. (i) Based on the similar technique used in grouping data packets, we first separate ACK packets into flights based on the inter-arrival time [38]. (ii) For each ACK in the same flight, we then estimate its delay $d2$ and select the minimal, $d2_{min}$, among all ACKs in the flight. Note that there exist different studies on measuring $d2$, we implement an algorithm similar to the one described in [16]. (iii) Last, the whole ACK flight is shifted with $d2_{min}$ (if it exists). Figure 13(b) depicts the shifted ACK flight, F_i'.

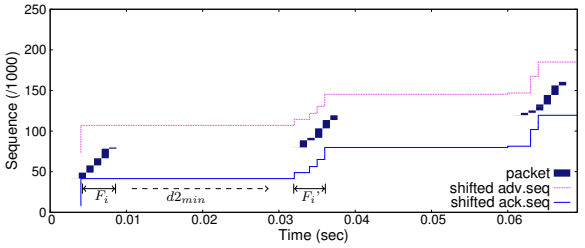
To draw a clear distinction between the previous works, here we do not infer the exact time that ACKs shall arrive at the sender (e.g., s_2 in Figure 12). Instead, we shift the ACKs forward in time with the purpose to explain the following packet arrivals. This is essential to analyze concurrent data packets from multiple senders arriving at the receiver. If the TCP trace is already taken at the sender side, this step could be skipped, or safely executed without effect.

C. Series Generation

From collected packet traces, this section describes three fundamental techniques used in generating series, namely *extraction*, *interpretation*, and *operation*, expressed with the following abstract rules:



(a) Original receiver-side packet trace



(b) Shifted packet trace

Fig. 13. TCP trace with original and shifted ACKs.

$$(Extraction) \quad series := func(trace) \tag{1}$$
$$(Interpretation) \quad series := series \tag{2}$$
$$(Operation) \quad series := func(series, \cdots args, \cdots) \tag{3}$$
$$series := series \oplus series \cdots \tag{4}$$

The first step, *Extraction*, is an unsupervised process which generate event series using TCP packet information. The later two steps then introduce heuristics and thresholds to infer TCP behaviours that otherwise do not manifest directly in the packet trace. Internally, the analyzer generates 34 series. Some are intermediate and serve only to derive other series. Due to the space limit, we describe only the representative series in the following discussion to illustrate the idea. Without otherwise specified, the series mentioned but not discussed are assumed to be properly generated using the techniques described in this section.

1) Extraction: First, we generate base series from *objective observations*, i.e., the events that we can directly extract from the packet trace. As *Rule 1* implies, this step solely works on packet traces. This is possible because the information required is coded in the protocol headers (IP or TCP) or lies in the packet arrival pattern. Example series in this category include the series of the receiver window size, packet transmission, and retransmission, etc.

Transmission time. This series tracks the transmission duration, indicating the time that TCP really spends on transmitting data packets. As shown in Figure 11, this series usually contributes an insignificant amount of time, and the inter-transmission gaps dominate the whole transfer period. The main task of T-DAT is to construct different series to explain the reasons behind these inter-transmission gaps.

Outstanding. This series tracks the number of outstanding

packets/bytes and the duration till they are acknowledged by the receiver, which is usually around one round-trip time. The duration varies due to transient network or receiver delay. This is a base series designed to answer questions about the number of unacknowledged packets at any time instance.

Receiver advertised window. This series tracks the changes of the receiver advertised window. Every time an ack is observed, the advertised window size and the inter-ACK time would be recorded. This represents the ever-changing upper bound of the outstanding packets enforced by the TCP receiver flow control.

Upstream and downstream loss. These two series track the time the TCP connection spends on recovering packet losses. We differentiate the upstream and downstream losses using the idea described in Section II-B2. Note that the upstream losses are detected by out-of-sequence packet arrivals, which could be caused by actual in-network reordering rather than packet drops. We further filter out those cases by implementing the algorithm in [17]. In Figure 11 we depict both series. But in this example piece of connection, there exist only instances of upstream packet losses. One important clarification to make is that these series do not track the time *instance* at which the packets are dropped. Instead, they capture the whole retransmission *period* spent in recovering the loss, which could be surprisingly long depending on the retransmission timeout.

2) Interpretation: In this step, new series are not generated from the packet trace, but instead from users' interpretation of the existing series. More specifically, we clone an existing series and annotate it with a more meaningful name with respect to our analysis purpose.

Network and Sender/Receiver local loss. As described in Section II-B2, if the sniffer is close to the sender side (e.g., neglectable $d2$ in Figure 12), then the *UpstreamLoss series* also indicate the local packet losses at the sender. So we construct series.

$$SendLocalLoss \quad := UpstreamLoss$$

On the other hand, if the sniffer is close to the receiver side (e.g., neglectable $d1$), we construct another series to represent the receiver local loss with the downstream loss.

$$RecvLocalLoss \quad := DownstreamLoss$$

One interesting question is how to know the location of the sniffer. Note that it is possible to infer the location based on the inter-arrival time of packets and ACKs ($d1$ and $d2$) [28]. For T-DAT, we leave this as a configurable setting, assuming that the user has prior knowledge of the data collection settings, including the sniffer location. Moreover, the definition of *local* could be subject to users' discretion. For example, suppose a case that the sniffer is at the ingress point of a large local network. Then, even there could exist substantial delay between the sniffer and the end hosts; the user may still consider the downstream (or upstream) losses as local to their own domain. We solicit from the users to specify what to be considered as local.

3) Operation: In this step, event series are generated based on operations among multiple series. In this step, we introduce inferences and heuristics which are necessary to track the TCP dynamic behavior (*Rule 3*).

Send application limited. Here we track the sender idle time, characterized by the idle period between the moment the sender receives the ACKs and sends the following data packets. Figure 11 shows three such idle instances in the trace (the middle line of *Send App limited*). During these periods, the sender already received the ACK for all its outstanding packets and is not bounded by the TCP windows. But the connection simply remains silent as the application may not produce data fast enough [38] or subject to the application rate limit [15]. Note that in the context of BGP, this series represents the delay induced by the sending end BGP application process.

Small/Large adv. window. These series track the size of advertised window, specifically for the small and large open windows. While the former indicates that the receiving application is unable to keep up with the sending rate and closes up the advertised window, the later indicates the opposite meaning. In the context of BGP, this series indicated the processing load of the receiving end BGP process. We consider the advertised window to be small or large if it is less than $3 \cdot MSS$ or greater than the maximum advertised window - $3 \cdot MSS$, respectively. The threshold is adopted from [28, 38].

Adv. bounded outstanding. This series is constructed from comparing the *Outstanding* and *Receiver advertised window* series. The purpose is to track the periods that the number of outstanding packets is bounded by the receiver window. Note that in this case, rarely the outstanding bytes aligns perfectly with the advertised window. In between there is always a small sequence gap, mostly smaller than one MSS. We determine that the outstanding is bounded by the advertised window if such difference is less than $3 \cdot MSS$ [28].

Cwd. bounded outstanding. This series tracks the periods that the outstanding packets are bounded by the sender congestion window. We take as input the *Outstanding* and *Adv. bounded outstanding* series. We consider a flight of outstanding packets to be congestion window bounded if it is not bounded by the advertised window, and another flight of packets are emitted immediately upon receiving the ACKs of current flight.

For the example in Figure 11, before the retransmissions, we can see 6 flights of outstanding packets that are bounded by the receiver window (shown as 6 square waves in the bottom curve). While during and after the retransmissions, the outstanding packets become instead bounded by the sender congestion window (shown by the fifth square curve).

Last, we generate series by applying set algebra on existing

series (*Rule 4*). This is possible because all series are uniformly presented in sets of time ranges.

Small/Large Adv. bounded outstanding. We further differentiate, for the *Adv. bounded outstanding* series, whether it is bounded by small or large receiver windows, as they indicate different receiver behavior as mentioned previously. With minimal effort, these series are generate by set intersection as the following.

$$\begin{aligned} SmallAdvBndOut &:= AdvBndOut \cap SmallAdv \\ LargeAdvBndOut &:= AdvBndOut \cap LargeAdv \end{aligned}$$

Note that in this work, the series are designed particularly for the purpose of delay analysis. T-DAT allows users to construct additional series for their specific needs.

D. Output: Contributing Delay Factors

In this step, out of 34 internal series, we arrive at 8 conslusive series, called delay *factors*. We then map these factors to the ones proposed in [28, 38]: *application limited*, *TCP window limited*, and *network path limited*. We further extend the taxonomy with the *local packet losses* as observed in our dataset.

For each factor, the tool outputs a quantitative measure *delay ratio*, defined as the *series size* divided by the duration of analysis period (i.e., the BGP table transfer duration in this work). We calculate the series size as the sum of all time durations in a series (e.g., the length of 9 square waves of the Upstreamloss series in Figure 11). Each ratio represents the fraction of time that the TCP connection exhibits a specific behavior. A raw ratio vector is output for a given analysis period.

$$\vec{V} = (r_1, r_2, \ldots, r_8), \quad r_i = \frac{size(Factor_i)}{AnalysisPeriod}, \quad i = 1 \ldots 8$$

In addition to the raw vector, we sort factors into three top level *factor groups*: *Sender*, *Receiver*, and *Network limited*, based on whether each series represents the sender, receiver, or network behavior. For each group, we calculate a *group delay ratio*, defined by the *union size* of all series in the group, divided by the analysis period. This results in a compact 3-vector, representing the fraction of delay contributed by three top-level groups.

$$\vec{G} = (R_s, R_r, R_n), \quad R_g = \frac{size(\bigcup Factor_i)}{AnalysisPeriod}, \quad g \in \{s, r, n\}$$

For example, a vector $(0.8, 0.1, 0.1)$ indicates that the sender-side factors collectively accounts for 80% of the transfer delay. In the next section, we present the classified delay factors together with the experiment results.

IV. Results

We applied our tool on the three traces described in Section II-A. The object is to demonstrate two different flavors of the tool on (i) surveying delay factors and (ii) identifying specific known problems.

A. Identifying Major Delay Factors

In the first scenario, we assume that users do not have prior knowledge about transport problems in their BGP table transfers. In this case, the delay analyzer serves to advise for each table transfer its dominant delay factors. This sheds light on *where (sender, receiver, network)* and *which (BGP, TCP)* could be the potential reason of the transfer delay.

We apply the tool on all table transfers and collect the 3-vector *group delay ratios* (R_s, R_r, R_n). We find that the network delay ratio, R_n, is close to zero in most cases. Thus, in Figure 14 we depict the scatter plots for the sender (R_s) and receiver (R_r) delay ratios. Please note that we mark the data points with solid crosses. For the ISP$_A$ (Vendor) trace, we only show the result for the period of March 2009 to May 2009[8]; otherwise the data points would be too crowded. We check other periods and the observation is similar.

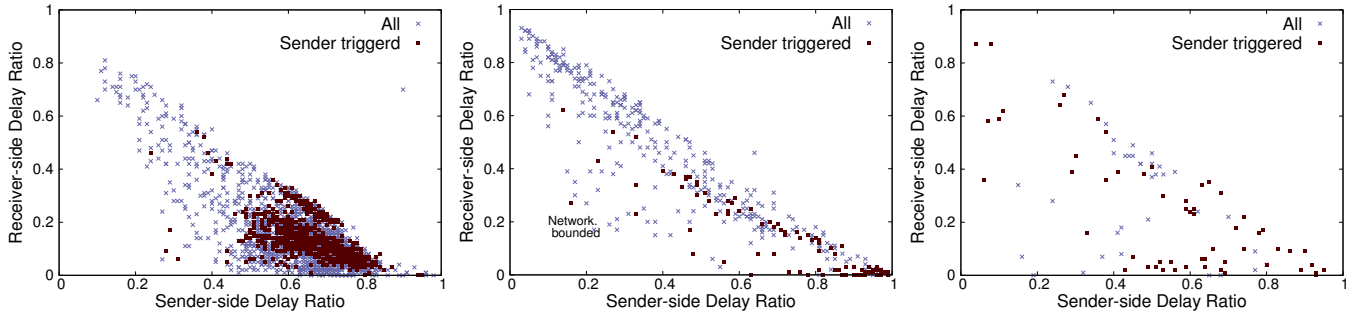
Figure 14(a) shows that the ISP$_A$ (Vendor) table transfers are more bounded by the sender-side factors, clustered between the ratio from 0.4 to 0.9. On the other hand, the ISP$_A$ (Quagga) table transfer are bounded by either the sender or the receiver-side factors (i.e., close to the line of $x + y = 1$). Also marked in the figure is a sample table transfers bounded by the network factors. To understand the trend, we further infer whether a table transfer is triggered by a sender or receiver [9] using the method in [9], further marked as the solid square points. Figure 14 (a) and (b) suggest that the triggering end could account more on the table transfer delay. This is as expected in BGP. If a BGP router fails, it would need to re-establish BGP sessions and exchange routing tables with all its peers. This imposes much stress on itself and is likely to become the bottleneck of table transfer performance. In Figure 14(c), we observe that RouteViews table transfers have more spread-out delay ratios, which could be due to the fact that, compare to ISP_A, RouteViews monitors are from different vendors and managed by different ISPs all over the Internet. This is under our on-going investigation.

Empirically, we say that the sender-side factors (as well as the receiver-side and network factors) are *major* if they collectively accounts for more than 30% of the table transfer duration (i.e., delay ratio $>$ 0.3). The 0.3 threshold is an engineering choice to allow more than one major factors been selected for a table transfer, which is rather common based on our observations. We test the threshold between 0.3 to 0.5, and it doest not qualitatively affects the relative importance among delay factors.

Table IV shows that the sender-side factors are the most prevalent, identified as the major factors for 83%, 67%, and 84% of table transfers in the three traces respectively. The receiver-side factors are the second most common, identified as the major factors of 42%, 61%, 43% of table transfers. The network factors dominate a relatively small number of table transfers. There are also a few cases that we do not find a major factor.

[8] includes 3038 table transfers

[9] that is, whether a transfer is due to a failure of the TCP sender or receiver



(a) ISP_A (Vendor) (b) ISP_A (Quagga) (c) RouteViews (Vendor)

Fig. 14. Sender-side, receiver-side and network delay ratios of table transfers

TABLE IV
DISTRIBUTION OF MAJOR DELAY FACTORS FOR TABLE TRANSFERS, WITH THE THRESHOLD OF 30% TRANSFER DURATION.

	ISP$_A$ (Vendor)	ISP$_A$ (Quagga)	RV
Table Transfers	10396	436	94
Sender-side limited	8525	295	79
Receiver-side limited	4210	242	40
Network limited	24	10	13
Unknown	20	5	2
Breakdown of Sender-side factor group			
BGP sender app	5740	266	28
TCP congestion window	2785	29	51
Local packet loss	-	-	-
Breakdown of Receiver-side factor group			
BGP receiver app	3391	204	0
TCP advertized window	758	37	24
Local packet loss	61	1	16
Breakdown of Network factor group			
Bandwidth limited	1	2	0
Network packet loss	23	8	13

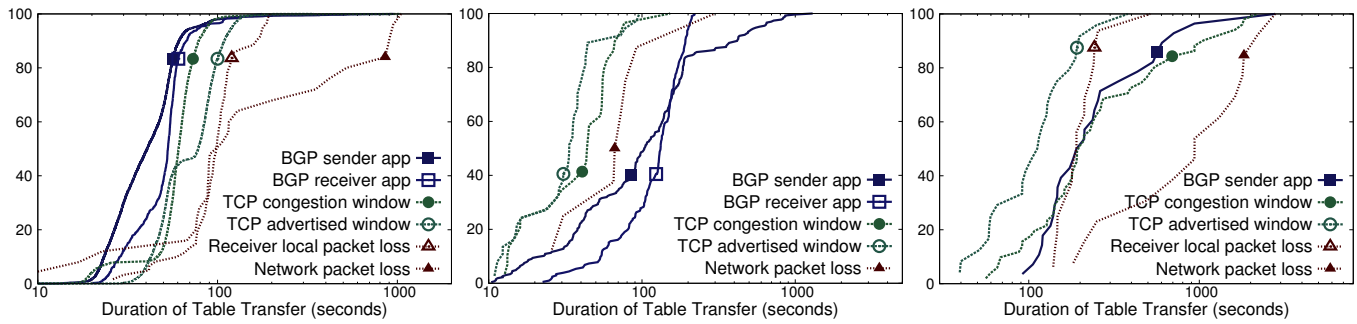
For each major group, Table IV further shows the breakdown results for individual factor. In ISP_A, more table transfers are limited by BGP than by TCP, with ratios between 2:1 and 7:1. This observation holds for both the sender-side and receiver-side limited table transfers. Though infrequently, we also observe the evidential impact of receiver local losses on 61 table transfers. Interestingly, the results for RouteViews shows that TCP, on the contrary, is more prevalent than BGP, especially for the receiver-side limited table transfers. One possible explanation is the different settings of TCP maximal advertised window: ISP_A uses 65KB while RouteViews use a much smaller 16KB window which is more likely to limit a connection at the TCP transport level. Another possible reason is that in our dataset, the ISP_A collectors failed from time to time, which triggered concurrent table transfers from multiple routers toward the collector. In Figure 15, we show the effect of number of concurrent table transfers to the receiving BGP and TCP delay ratio. We can observe that when less than 10 concurrent table transfers, the table transfers are slightly bounded by the TCP receiver window. However, as the number increase, the BGP receiver starts to become the bottleneck. Note that in the 3 month RV trace, we are not able to find any case of high concurrent-number table transfers to make the same (or different) observation. Last, for the network limited cases, the effect of packet losses is more prevalent than the bandwidth. In fact, we expect *none* of the table transfers should be limited by the more-than-sufficient link bandwidth in the ISP_A and RouteViews network. These three rare bandwidth-limited cases are actually receiver-limited. They are falsely categoried because their *whole* table transfer is TCP receiver flow controlled, and thus confuses T-DAT's inference on the link bandwidth. This would not happen as long as the table transfer includes some non-receiver controlled periods.

Another question is the association of these factors with the table transfer duration. Given the identified major delay factors, we re-plot the CDF of the transfer duration in Figure 16. Overall, the table transfers limited by the TCP receiver window have the shortest duration, followed by the ones limited by the congestion window. This is because in these cases, TCP keeps pumping out packets roughly every RTT; only that the amount of outstanding packets is limited by the window size, which is still relatively fast. Otherwise, if table transfers are limited by packets losses (local or network), they waste time in TCP timeout and retransmissions, which could take up to hundreds of seconds to finish. Generally, the table transfers limited by BGP application processes could also have longer durations, which reflects the processing limitation.

Another interesting observation lies in ISP_A (Vendor) and RouteViews (Vendor) traces, in which TCP retransmissions take much longer delay. In this case, we observe that packet retransmissions are particularly slower when both the sender and receiver are vendor routers. This is subjective to our furture investigation.

B. Revisiting the Transport Problems

This second scenario describes the usage of the tool on investigating *known* problems. That is, the users are aware of specific transport problems, and the purpose is to check whether these problems do affect their BGP operations. In this case, T-DAT facilitates the process by converting the raw packet traces into multiple unified series of time ranges. The users then only need to focus on the relevant series with respect to their analysis need. To demonstrate, based on our understanding of transport problems in Section II, we develop the following algorithms to identify and quantify them in the table transfers.



(a) ISP_A (Vendor) (b) ISP_A (Quagga) (c) RouteViews (Vendor)

Fig. 16. Table transfer duration by delay factors. Y axis is the cumulative percentage

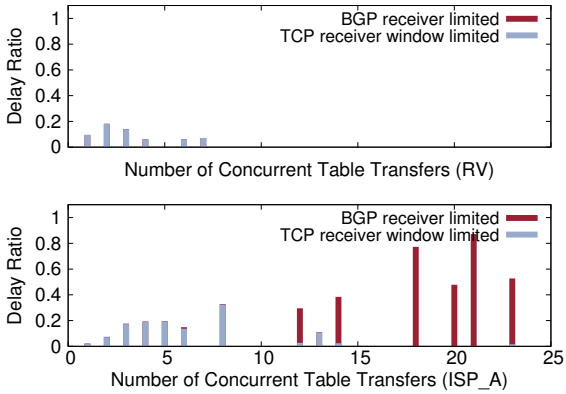


Fig. 15. Affect of concurrent table transfers

BGP timer gaps. To detect the repetitive timer gaps in table transfers, we take the *Send Application Limited* series, which captures the periods that the BGP sender remains idle. We then draw the length distribution of each gap in the series. Figure 17 shows the gap distribution for one example table transfer that contains 200ms timer gaps. The idea is that if a table transfer does contain repetitive gaps due to a specific BGP implementation timer [15], there would be a knee point in the curve indicating the timer value (e.g., the 200ms marked in the figure). We use the method in [27] to automatically identify the knee point and, thus, BGP timers. For ISP_A trace from May 2008 to April 2009, during which there were two collectors, we checked and found that the two collectors observe similar timers from a same router. However, the timer values are not exactly the same, or one timer could be the multiple of the other, which might be due to delay variaion in packet arrivals.

In general, we found that the timer lengths (if exist) are around a few specific values: 80ms, 100ms, 200ms and 400ms, while 200ms is the most prevalent. Note that 200ms is the default timer used by a major vendor reported recently in [15]. We did not verify as the authors had not revealed the actual vendor. In Table V, we list the number of table transfers in which we successfully detect a pronouncing timer. Moreover, these timer gaps introduce 7.31 to 19.40 seconds of delay in the table transfer in average.

Consecutive packet losses. For this problem, we take as input all series that are related to packet losses: *SendLocalLoss*, *RecvLocalLoss* and *NetworkLoss*. We union these three series to construct a new series which captures all instances of packet losses. From such series we check if there exist more than 8 consecutive losses. We choose 8 as a conservative threshold which is sufficiently large to reduce the TCP congestion window and the slow start threshold to the minimum 1 or 2 MSS, assuming the maximal 64KB window and the 1400 byte MSS. Surprisingly, Table V shows that more than 20% of table transfers experienced at least one consecutive losses. However, in ISP_A, the incurring delay is relatively short with the average around 5 seconds. This is the reason why we have detected many cases of consecutive losses but they do not surface as the major delay factor as shown in Table IV.

On the other hand, the incurring delay is much longer in RouteViews with the average of 31 seconds. We check and find that the RouteViews' TCP connections backoff more aggressively. In many cases, the TCP retransmission timeout (RTO) increases promptly to a few seconds after two or three timeouts. The detected 29 cases match the number of loss-limited table transfers in Table IV (16+13). Note that the TCP retransmission delay is contributed by various factors such as TCP versions, window size, RTO, etc. T-DAT can help detect the delay, while investigating the causes of this delay is beyond the scope of this paper.

Peer Group blocking. For this problem, we only focus on the pathological blocking as described in Section II-B3. More specifically, we identify the cases that the table transfer to a peer-group is completely paused or blocked because of the failure of one member peer. During the pause, only the keep-alive messages are periodically exchanged. Here, we take again the *Send Application Limited* series, and find the suspicious long idle gaps that match the BGP keep-alive timers. We then query the *Outstanding* series to make sure that only BGP keep-alive messages are seen within the whole idle period. Again, for ISP_A trace from May 2008 to April 2009, during which we have two collectors in the same peer-group, we check if the other session has experienced packet losses and blocked the group. This can be achieved by intersecting the series from two different TCP connections as the following:

$$Quagga.SendAppLimited \cap Vendor.Loss \quad or$$
$$Vendor.SendAppLimited \cap Quagga.Loss$$

TABLE V
IDENTIFY PROBLEMS DESCRIBED IN SECTION II, AND THE AVERAGE INTRODUCED DELAYS.

	ISP_A (Vendor)		ISP_A (Quagga)		RV	
Table Transfers	10396		436		94	
Gaps in table transfers	857	7.31 (sec)	74	16.25 (sec)	7	19.40 (sec)
Consecutive losses	2092	5.14 (sec)	176	4.52 (sec)	29	31.15 (sec)
BGP peer-group blocking (upon resets)	8	134.53 (sec)	8	129.72 (sec)	3	94.37 (sec)

As shown in Table V, we detect 8, 8, and 3 such cases in the table transfers, which appears to be infrequent. However, note that whenever this problem occurs, it introduce a long delay. This is because the paused table transfer resumes only after the failed peer timed out and has been removed from the peer group. Depending on the default BGP timeout setting, this would take about 90 to 180 seconds. Also note that the effect of this problem would be amplified by the number of routers in the group, which ranges from several to tens of members in the current practice.

So far we demonstrate how to identify known transport problems using the series data. Here, the event series can actually help capture new problems as well. Recall that we generate event series to represent different states of a TCP connection. It is possible that series themselves do not agree with each other. We take the intersection of different series, and find a conflict between two series:

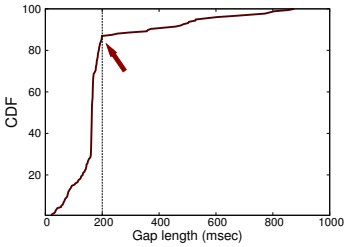
$$ZeroAckBug \quad := \quad ZeroAdvBndOut \cap UpstreamLoss$$

That is, in the dataset we find controversial cases of slow TCP connections, which experience both zero receiver window and persistent packet losses at the same time. This is suspicious in that packets get constantly dropped even under low transmission rate. It turns out that the sending TCP has a implementation bug: upon receiving a zero-window ACK, the sender creates a 1-byte probe packet [23]. However, if another ACK arrives again and opens up the window before the sender transmits the probe, the probe gets incorrectly discarded by the sender. This triggers repetitive retransmissions. We found that this bug was left in the operational routers for years.

To conclude, we emphasize that it is possible to directly work on the raw packet trace and develop individual ad-hoc technique to capture table transfer problems. However, the event series offers a few unique advantages. First, as shown above, it allows the users to focus on the series (and the associated BGP data transfer behavior) that are closely related to their specific analysis need. Second, series are stored using the set data structure, for which we provide a rich set of lookup and manipulation operations. Last, the set presentation also facilitates the cross-connection inspection, which could be nontrivial when working on several raw traces.

V. DISCUSSION AND FUTURE WORK

We have presented the tool and the application results. In this section, we discuss the limitation, T-DAT implementation and potential usage .



	Timers (ms)
ISP_A (Vendor)	200 400
ISP_A (Quagga)	100 200
RouteViews	80 400

Fig. 17. Infer BGP timers from the gap distribution

A. Limitation

To clarify, our analysis results do not intend to fully represent BGP table transfers between operational routers, as we only get to access traffic between operational routers and BGP collectors. Note that this is not a particular problem to this work, but rather a common limitation shared by BGP studies based on the monitoring data [32, 39]. This work contributes to the community in that: first, the identified transport problem (especially those of router implementaion and features) shall persist indepently of settings. Second, this work helps differentiate, in the BGP monitoring data, the TCP induced delay from the BGP routing convergence delay. The slow times may otherwise be attributed the BGP routing behavior and lead to questionable conclusions at best.

B. Source of Inaccuracy

As discussed earlier, the tool operates on analyzing the relative arrivals between data and ACK packets. The inaccuracy of the results comes from two sources of errors: (i) the uncertain information in the packet trace due to the sniffer location, and (ii) the various heuristics introduced in the analysis algorithms. Understanding and quantifying these two errors individually are already challenging and deserve their own research venues respectively in [16, 20, 31] and [17, 28, 38]. In this work, we design the tool carefully to proceed in two separate steps. We first rewrite the original packet trace to a new inferred sender-side trace when necessary. Then the rest of the tool works strictly assuming the input of sender-side packet traces. This isolation helps prevent the complicated aggregated effect of these two error sources, and allows us to reuse many techniques and parameter settings established in previous TCP inference and rate analysis (described along with the tool in Section III), which we observe also work empirically well in this work. As the next important step, we expect to explore the effect of different parameter settings, for both BGP and other application packet traces.

C. Implementation

This work is built around the idea of sets and time-ranges. As the preliminary proof-of-concept, we implement in Perl

TABLE VI
ANALYSIS TOOL SUITE

	Description
tcptrace'	Patched from the original tcptrace [21]. Modify the I/O processing to handle huge data volume
pcap2bgp	Reconstruct TCP stream from tcpdump packet trace. Extrace BGP messages from the data stream and store in the MRT format.
t-dat	TCP delay analyer. Output delay factors and series data.
BGPlot	Extended from SCNMPlot [19] Visualize the TCP sequence and the event series.

language the time-ranges with integer-ranges. We convert the tcpdump second-based timestamps to micro-seconds. Then we store them in the set container of big integers. Each set is developed to support *range* query and update operations. We also implement set operations, including set *intersection*, *union*, and *complement*. The T-DAT analyzer is then wrapped in a Perl script, consisting of approximately 5,500 lines of code. It processes the RouteViews trace (47GBytes) in 64 minutes (26 seconds per TCP connection in average). [10] In addition, as the outcome of this study, we also develop and include several utility tools as listed in Table VI, and would be made publicly available at http://irl.cs.ucla.edu/bgpmicro.

D. Prospective Deployment and Usage

This work is driven by BGP delay analysis. To our knowledge, most ISPs have deployed their internal BGP collectors to monitor the BGP operation. It shall require minimal effort to record TCP traffic together at the collector boxes and feed into T-DAT for detail performance analysis. More importantly, the tool only requires passively collected TCP trace, and allows operators to capture traffic trace between two operational routers without any modification to network settings.

In addition, considering the proposed T-DAT as a generic tool, the series data can serve as the **sanitized input to other TCP analysis studies**. Currently, TCP analysis is mostly conducted on the raw packet trace [16, 24, 28, 38], which can be less effective with respect to their specific goals. Qian et al. [24] extract various non-RTT flow clocks caused by application timers. Clearly, such application timers are often concealed by the much more pronouncing RTT, and only reveal during which the connection is application limited. Jaiswal et al. [16] proposed to infer TCP flavors by comparing the number of outstanding packets against the projected congestion window size. The approach is more effective when the TCP connection is bounded by the congestion control, which is not always true throughout the connection lifetime. For these two analysis, instead of processing the raw trace, it could be more effective to take in as input the series *SendAppLimited* and *CwdBndOut*, which exactly point to the periods of their research interests, respectively.

[10]Not including the pre-processing time of tcptrace

VI. RELATED WORK

This work follows the lead of two research threads.

Understanding the BGP transport behavior: The impact of the transport layer dynamics on the application performance have long been recognized and studied in the literature, including HTTP [29], video streaming [18], online gaming [7], and data centers [8], to name a few. However, until recently, there has been marginal attention on investigating the BGP over TCP behavior. Feldmann et al. [14] measure BGP pass-through times using controlled environment, and quantify the impact of individual router delays on the overall convergence time. Their work focuses on exploring the factors that introduce delay within the router itself. Xiao et al. [34] model the BGP session survivability under severe TCP congestion, and propose to improve BGP robustness by more aggressive TCP retransmissions. Zhang et al. [39] instead demonstrate a low-rate DoS attack to defeat TCP retransmissions and trigger BGP session resets. Houidi et al. [15] examine the packet trace of BGP slow table transfers and find a potential cause to be the timer-driven router implementations. In this work, we investigates the BGP traces collected from the operational ISP and BGP monitoring networks. We confirm the observations made in these previous works, and more importantly report new potential transport problems, which has motivated our design of a new diagnose tool.

Identifying limiting factors of TCP connection: There have been several studies on analyzing the rate limiting factors of a TCP connection. Zhang et al. [38] proposed to classify rate limiting factors as *application limit, congestion, TCP window*, etc. The idea is to separate TCP packet trace into flights, and test whether each flight is limited by specific factors. Similar to this work, Siekkinen et al. [28] addressed this problem with a time series approach, which offers a more detail quantitative score for the level of the limitation. Compared to this, our work targets on a different measure, *delay*, driven by our research context of analyzing BGP routing protocol, which more concerns the routing message delay. In addition, as rate is a relative measure calculated by transmission size and interval, rate analysis commonly operates on every fixed number (e.g., θ) of packets [38] or interval [28], which could be affected by the selection of θ. In this work, the event series instead preserve the original packet timestamp and the inter-packet delay.

One similar work indeed focusing on TCP delay is Critical Path Analysis (CPA) [6]. From TCP packet trace, Barford et al. proposed to construct a path that associates data and ACK packets based on the *happen-before* relationship. Then, each link (i.e., association between two packets) along the path indicates a particular type of delay. Note that the technique requires to know in advance the sender's TCP implementation and initial parameters to accurately simulate the change to TCP windows. Only TCP Reno is illustrated and supported in [6]. Compare to this, T-DAT operates on relative inter-arrivals of TCP data and ACK packet flights. This may not be as accurate as CPA, but can support common TCP versions

that adopt window-based congestion control (e.g., TCP Tahoe, Reno, New Reno). Also, previous works focus on identifying limiting factors for individual TCP connection. As we show in Section II-B3, there could exist intervention among BGP connections, and this work offers representation of connections in unified time ranges, which allows efficient analysis across multiple connections.

VII. Conclusion

This paper seeks to solve the BGP slow table transfer puzzle from the view of TCP packet level dynamics. By investigating BGP and TCP data collected in a large ISP and RouteViews, we found various transport problems, not yet reported in previous works, that introduce transfer delay up to a few and tens of seconds. Without the evidential packet trace, such transport induced delay could be overlooked and easily attributed to the system-wise BGP slow convergence. Stemmed from the venue of TCP rate analysis, we develop a new delay-centric tool with the goal to explain various reasons behind the table transfer duration, which we believe is an important step toward diagnose and improve the overall BGP transport performance. We demonstrate the tool usage in identifying major delay factors as well as investigating specific problems in our BGP dataset. As part of our future work, we would like to investigate the general BGP transport behavior in addition to the initial table transfer, specifically the massive updates triggered upon the inter-domain routing failures. Moreover, as the tool itself is BGP agnostic, we would also like to explore its potential usage for other delay sensitive applications.

References

[1] Quagga Software Routing Suite. http://www.quagga.net/.

[2] RIPE Routing Information Service. http://www.ripe.net/projects/ris/.

[3] The RouteViews project. http://www.routeviews.org/.

[4] wireshark. http://www.wireshark.org/.

[5] MRT routing information export format. http://www.ietf.org/internet-drafts/draft-ietf-grow-mrt-14.txt, 2011.

[6] P. Barford and M. Crovella. Critical path analysis of TCP transactions. In *Proc. of ACM SIGCOMM*, 2000.

[7] K. Chen, C. Huang, P. Huang, and C. Lei. An empirical evaluation of TCP performance in online games. In *Proc. of the ACM SIGCHI*, 2006.

[8] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. In *Proc. ACM workshop on Research on enterprise networking (WREN)*, 2009.

[9] P.-c. Cheng, X. Zhao, B. Zhang, and L. Zhang. Longitudinal study of BGP monitor session failures. *SIGCOMM Comput. Commun. Rev.*, 40, 2010.

[10] J. Choi, J. Park, P. Cheng, D. Kim, and L. Zhang. Understanding BGP Next-hop Diversity. In *Proc. of IEEE INFOCOM Workshop Global Internet Symposium*, 2011.

[11] Cisco. Understanding Selective Packet Discard (SPD). www.cisco.com/image/gif/paws/29920/spd.pdf, 2005.

[12] Cisco. Troubleshooting High CPU Caused by the BGP Scanner or BGP Router Process. http://www.cisco.com/application/pdf/paws/107615/highcpu-bgp.pdf, 2008.

[13] J. Elson. tcpflow. http://www.circlemud.org/jelson/software/tcpflow.

[14] A. Feldmann, H. Kong, O. Maennel, and A. Tudor. Measuring BGP pass-through times. *Lecture notes in computer science*, pages 267–277, 2004.

[15] Z. B. Houidi, M. Meulle, and R. Teixeira. Understanding Slow BGP Routing Table Transfers. In *Proc. of Internet Measurement Conference (IMC)*, 2009.

[16] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proc. of IEEE INFOCOM*, 2004.

[17] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. *IEEE/ACM Transaction on Networking*, 2007.

[18] T. Kim and M. Ammar. Receiver buffer requirement for video streaming over TCP. In *Proc. of SPIE*, 2006.

[19] LBL. scnmplot. http://www-didc.lbl.gov/SCNM/SCNMPlot.html.

[20] G. Lu and X. Li. On the correspondency between TCP acknowledgment packet and data packet. In *Proc. of Internet Measurement Conference (IMC)*, 2003.

[21] S. Ostermann. tcptrace. http://www.tcptrace.org/.

[22] K. Poduri, C. Alaettinoglu, and V. Jacobson. BSTBGP Scalable Transport. http://www.nanog.org/meetings/nanog27/presentations/van.pdf, 2003.

[23] J. Postel. Transmission Control Protocol. RFC 793 (Standard), Sept. 1981. Updated by RFCs 1122, 3168.

[24] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP revisited: a fresh look at TCP in the wild. In *Proc. of Internet Measurement Conference (IMC)*, 2009.

[25] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), Jan. 2006.

[26] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP routing stability of popular destinations. In *Proc. of ACM SIGCOMM IMW*, 2002.

[27] S. Salvador and P. Chan. Determining the number of clusters/segments in hierarchical clustering/ segmentation algorithms. In *Proc. of IEEE International Conference on Tools with Artificial Intelligence*, 2004.

[28] M. Siekkinen, G. Urvoy-Keller, E. W. Biersack, and T. En-Najjary. Root cause analysis for long-lived TCP connections. In *Proc. of ACM CoNEXT*, 2005.

[29] J. Touch, J. Heidemann, and K. Obraczka. Analysis of HTTP performance. *ISI Research Report ISI/RR-98-463, USC/ISI*, 1998.

[30] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *Proc. of ACM SIGCOMM*, 2009.

[31] B. Veal, K. Li, and D. Lowenthal. New methods for passive estimation of TCP round-trip times. In *Proc. of Passive and Active Network Measurement (PAM)*, 2005.

[32] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. Observation and analysis of BGP behavior under stress. In *Proc. of ACM SIGCOMM IMW*, 2002.

[33] J. Wu, Z. M. Mao, J. Rexford, and J. Wang. Finding a needle in a haystack: pinpointing significant BGP routing changes in an IP network. In *Proc. USENIX NSDI*, 2005.

[34] L. Xiao, G. He, and K. Nahrstedt. BGP session lifetime modeling in congested networks. *Computer Networks*, 50, 2006.

[35] L. Xiao and K. Nahrstedt. Reliability models and evaluation of internal BGP networks. In *Proc. of IEEE INFOCOM*, 2004.

[36] B. Zhang, V. Kambhampati, M. Lad, D. Massey, and L. Zhang. Identifying BGP routing table transfers. In *Proc. of ACM SIGCOMM workshop on Mining network data*, 2005.

[37] R. Zhang and M. Bartell. *BGP Design and Implementation*, chapter Tuning BGP Performance, pages 74–81. 1999.

[38] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the

characteristics and origins of Internet flow rates. In *Proc. of ACM SIGCOMM*, 2002.

[39] Y. Zhang, Z. Zhang, Z. M. Mao, C. Hu, and B. MacDowell Maggs. On the impact of route monitor selection. In *Proc. of Internet Measurement Conference (IMC)*, 2007.