# Explaining BGP Slow Table Transfers

Pei-chun Cheng*, Jong Han Park †, Keyur Patel ‡, Shane Amante §, Lixia Zhang *
*University of California, Los Angeles, {pccheng,lixia}@cs.ucla.edu
†AT&T Labs, jpark@research.att.com
‡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 new TCP transport problems, that collectively induce significant delays. 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 BGP's sending and receiving behavior, TCP's parameter settings, TCP's flow and congestion control, and network path limitation. Identifying these delay contributing factors makes an important step for ISPs and router vendors to diagnose and improve the BGP performance.**

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

## I. Introduction

The distributed routing infrastructure of today's Internet is glued together by Border Gateway Protocol (BGP) [18]. 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 communicate reachability changes. Typically neighboring BGP routers are connected by high speed lines, and 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 resets [8, 10, 24, 25, 28]. It remains unclear, given only the BGP level information, what causes these slow table transfers or how to fix them.

Several recent studies have looked into the BGP behavior via its interactions with TCP. Xiao et al. [27] show that BGP performance could be seriously degraded upon repeated TCP retransmissions due to network congestion. Zhang et al. [31] demonstrate that, even without network congestion, the durations of BGP table transfers may take up to an hour under specific low-rate TCP DoS attacks. By investigating TCP packet traces, Houidi et al. [10] 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 their particular settings, and highlight opportunities to reveal BGP problems by studying the TCP behavior, there still remain open issues about their prevalence and impact on today's BGP operations.

In this paper, we use TCP and BGP data traces collected from a large ISP and the RouteViews project [2] to investigate potential causes for the slow BGP table transfers. As part of our work towards investigating the causes, we 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 limitation. Our tool is inspired by pioneering works on TCP rate analysis [21, 30]. However, this work is distinct in that we use the data transfer *delay* rather than transmission *rate* as the main performance measure, which has a direct impact on how quickly the routing information can be propagated to reduce the overall convergence time. This 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 collected in a large ISP and RouteViews (Section II). Second, we develop a new tool, T-DAT, to systematically identify and measure different reasons behind the transfer delay (Section III). We demonstrate the tool's usage and present our preliminary answers on explaining durations of slow table transfers. Some identified issues are due to router implementation or features, which shall affect both BGP monitoring and general operations (Section IV). 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.

## II. Table Transfer Dataset

This section starts off our search for the reasons behind BGP slow table transfer. Compared to previous works that only use BGP data [8, 19, 24, 28], we seek to collect and

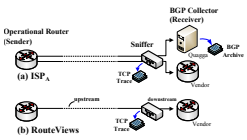



Fig. 1. BGP/TCP data collection

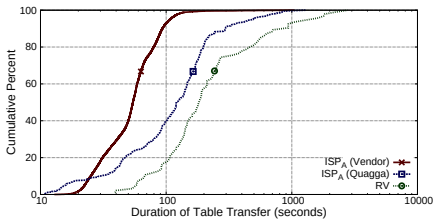



Fig. 2. CDF of table transfer duration

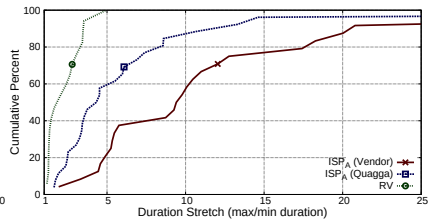



Fig. 3. Stretch of table transfers

analyze the corresponding TCP packet traces, with the goal to reveal distinct *transport protocol* issues.

We collected BGP data at a large ISP ($ISP_A$) and the RouteViews [2] 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) [4] format. The Vendor collector works as a *looking glass* and allows operators to log in and look up the current routing information. In addition to BGP data collection, a TCP packet sniffer (tcpdump) is deployed in front of the collector, and records the pass-through traffic in *both* directions while 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*. From the data flow's perspective, the network path between Sender-Sniffer and Sniffer-Receiver is considered as the *upstream* path and the *downstream* path respectively.

Table I summarizes the characteristics of the collected traces. We 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 life-time*, *estimated round-trip time (RTT)*, *maximum segment size (MSS)*, etc. *(ii)* Based on the TCP connection start time which indicates the *start* of a given BGP table transfer,[3] we then turn to the BGP archive and apply MCT (Minimum Collection Time) algorithm [28] to identify the *end* of the 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 [3] or tcpflow [9], pcap2bgp further extracts individual BGP messages from the TCP byte stream. Then we apply MCT on the extracted BGP messages as in the previous step.

Table I also lists 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 2 shows the CDF of the table transfer duration. The majority of the table transfers finished within a few minutes. The table transfers of 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. For each router-collector pair that has more than two table transfers, we further calculate a *stretch ratio*, defined as the longest table transfer duration divided by the shortest one. A high ratio indicates that the table transfer duration is significantly stretched for some reason, although the amount of data being sent at the BGP-level is the same. Figure 3 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. The stretch could even be an order of magnitude higher, and this observation cannot be fully explained using only the BGP data. In the following sections, we propose a systematic approach to identify potential causes of these slow times by carefully analyzing the matching TCP data collected together with the BGP data.

## III. Explain the Delay with TCP Analysis

To explain the suspicious delays potentially caused by the underlying transport, we introduce a new analysis tool that can capture the TCP connection behavior and help identify the impacting factors that delay the data transfers. The idea is inspired by *TCP rate analysis* [21, 30], 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 a different metric, namely *transfer delay* to identify major contributing factors for each BGP table transfer. [5]

The main idea of T-DAT is to transform, given a TCP trace, the relative 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 uses congestion and receive windows to control packet delivery (i.e., TCP flavors such as Tahoe, Reno, New Reno), which

[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 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 TCP connection [18].

[4] Different from [28], which runs MCT on every BGP message, 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.

[5] We discuss the difference between rate and delay analysis in Section VI.

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 ∼ 2009.04 | Vendor | 1023 / 218 | 24 | Yes | - | 10396 |
| $ISP_A$-2 | iBGP | 2008.05 ∼ 2009.04 | Quagga | 909 / 138 | 27 | Yes | Yes | 180 |
| | | 2009.09 ∼ 2010.09 | | 1296 / 219 | | | | 219 |
| | | 2010.11 ∼ 2011.01 | | 492 / 81 | | | | 37 |
| RV | eBGP | 2010.11 ∼ 2011.01 | Vendor | 176 / 47 | 59* | Yes | - | 94 |

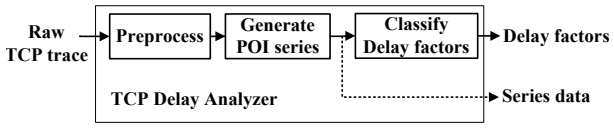



Fig. 4. High level design

is generally true for commercial routers that do not employ extraordinary TCP implementations. Also note that this is a common assumption made in TCP rate analysis studies [21, 30]. It is subjective to our future work to extend T-DAT to support other TCP variations.

Figure 4 depicts a 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 [30] (§III-D). We call the tool T-DAT (TCP Delay Analysis Tool), named after T-RAT in [30], to make a simple yet clear distinction.

### A. Delay Event Series

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 different TCP end-point behaviors, 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 5 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 retransmission delay 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 delay time introduced

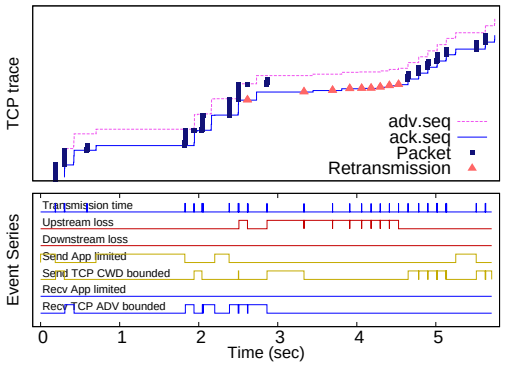



Fig. 5. Example TCP trace and event series

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.

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 such as the maximum segment size (MSS), round trip time (RTT), maximal advertised window size,[6] that are extracted using tcptrace [15]. We also use tcptrace to label packets such as retransmissions, out-of-sequence, and duplicates.

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 [11, 21, 30]. In previous works, the major concern was to estimate RTT at different sniffer locations. Figure 6 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 6) is inferred as the sum of *d1* and *d2*, each representing one part of the round-trip delay

[6] advertised by the receiver

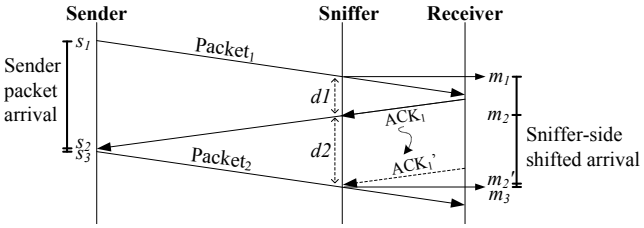
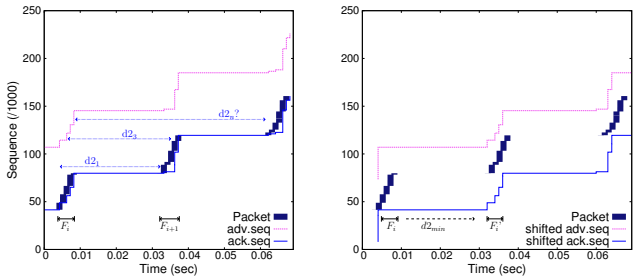



Fig. 6. Inferring the sender-side packet arrival




(a) Original packet trace (b) Shifted packet trace

Fig. 7. TCP trace with original and shifted ACKs.

for Sniffer-to-Receiver and Sniffer-to-Sender [12]. Building upon this approach, we *shift forward* the ACKs with the offset *d2* to match their corresponding data packets (e.g., $\text{ACK}_1 \rightarrow \text{ACK}_1$'), such that the shifted trace (i.e., the original data packets with shifted ACKs) approximates the sender-side behavior. As shown in Figure 6, the goal is to rewrite the *packet-ack-packet* arrival at 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 [12, 14, 21, 23],

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* commonly refers to data packets, here we use the term to refer to ACKs that are sent back to back within a group. In Figure 7(a), we mark a flight of $n$ ACKs, $F_i$, together with their estimated *d2* delays, $d2_1$, $d2_2$, $\ldots d2_n$. Note that for $d2_1$ and $d2_2$, the estimation is relatively more 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 [30]. $(ii)$ 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 [11]. $(iii)$ Last, the whole ACK flight is shifted with $d2_{min}$ (if it exists). Figure 7(b) depicts the shifted ACK flight, $F_i'$.

### C. Series Generation

From collected packet traces, we use three techniques to generate event series, namely *extraction*, *interpretation*, and *operation*. The first step, extraction, is an unsupervised process which generate event series from objective TCP packet information. In the later two steps, we introduce heuristics and thresholds to infer TCP behaviors not directly seen in the packet trace. Internally, the analyzer generates 34 series. Some are only intermediate and serve to derive other series. Due to the space limit, we describe only the representative series in the following discussion to illustrate the idea. Unless 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. This step solely works on the information coded in 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 spends solely on transmitting data packets. As shown in Figure 5, 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 sent packets/bytes and the delay till 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 flow control.

**Upstream and downstream loss.** These two series track the time the TCP connection spends on recovering packet losses. Generally, packet losses could occur along a congested network path. Here, we further differentiate packet losses that happen *upstream* or *downstream* to the sniffer. The idea is based on classifying the packet retransmissions [12]: 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.[7] Later, the Sender retransmit the packet with the same sequence number. We then mark the second packet as a retransmission due to *downstream losses*. On the

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

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. We then mark these retransmissions as due to *upstream losses*. In Figure 5 we depict both series. Just that 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, but the whole retransmission *period*, which could be surprisingly long due to TCP 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.

**Network and Sender/Receiver local loss.** We check whether packet losses occur *locally* to the end hosts, but not somewhere deep in the network. That is, if the sniffer is close to either the sender side (e.g., neglectable $d2$ in Figure 6) or the receiver side (e.g., neglectable $d1$), we can then further interpret the series *UpstreamLoss* and *DownstreamLoss* to indicate the local packet losses as *SendLocalLoss* and *RecvLocalLoss* respectively. For example in our particualr case, Sniffer is simply co-located with Receiver, thus the downstream losses shall occur *locally* to the Receiver. However, for the upstream losses, we could not further tell whether the packets are lost at the sender side or along the path.

One interesting question is how to determine the location of the sniffer. Note that 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 existing series. In addition, we introduce inferences and heuristics which are necessary to track the TCP dynamic behavior.

**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 5 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. Note that in the context of BGP, this series represents the delay induced by the sending end BGP application process.

**Small/Large adv. window.** This series track the small and large advertised 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 indicates 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, adopted from [21, 30].

**Adv. bounded outstanding.** This series compares 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 rarely the outstanding bytes aligns perfectly with the advertised window. We determine that the outstanding is bounded by the advertised window if the difference is less than $3 \cdot MSS$ [21].

**Cwd. bounded outstanding.** This series tracks the periods that the outstanding packets are bounded by the congestion window. Using the *Outstanding* and *Adv. bounded outstanding* series as our input, 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 example in Figure 5, before the retransmissions, we can see 6 flights of packets that are bounded by the receiver window (the bottom square curve). While during and after the retransmissions, the outstanding packets become bounded by the sender congestion window (the fifth square curve).

Last, we derive additional series by applying set algebra on existing series. 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. With minimal effort, these series are generate by set intersections.

$$SmallAdvBndOut := AdvBndOut \cap SmallAdv \quad (1)$$

$$LargeAdvBndOut := AdvBndOut \cap LargeAdv \quad (2)$$

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 conclusive series, called *delay factors*. We then map these factors to the ones proposed in [21, 30]: *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 delay 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 time durations in a series (e.g., the length of 9 square waves of the Upstreamloss series in Figure 5). Each ratio represents the time fraction that the TCP connection exhibits a specific behavior. A raw ratio vector is output for a given analysis period.

$$\overrightarrow{V} = (r_1, r_2, \ldots, r_8), \quad r_i = \frac{size(Factor_i)}{AnalysisPeriod}, \quad i = 1 \ldots 8 \quad (3)$$

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.

$$\overrightarrow{G} = (R_s, R_r, R_n), \quad R_g = \frac{size(\bigcup Factor_i)}{AnalysisPeriod}, \quad g \in \{s, r, n\} \qquad (4)$$

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. The object is to demonstrate two important steps of the tool on $(i)$ surveying major delay factors and $(ii)$ identifying specific BGP transport problems.

### A. Surveying Major Delay Factors

In this first step, we assume that the user does not yet have knowledge about transport problems in 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 8 we depict the scatter plots for the sender ($R_s$) and receiver ($R_r$) delay ratios. 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] for clarity; otherwise the data points would be too crowded. We checked other periods and the observation is similar.

Figure 8(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 transfers 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 transfer bounded by the network factors. To understand the overall trend in more detail, we infer whether a table transfer is triggered by a sender or receiver [9] using the method in [8], further marked as the solid square points. Figure 8 (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 8(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.

[8]includes 3038 table transfers

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

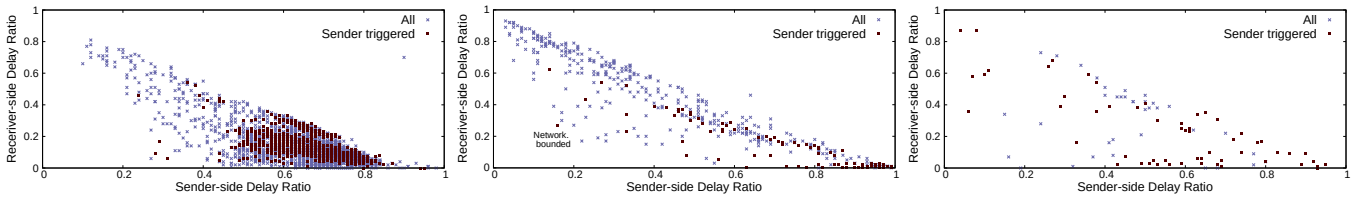
TABLE II
DISTRIBUTION OF MAJOR DELAY FACTORS FOR TABLE TRANSFERS, WITH THE THRESHOLD OF 30% TRANSFER DURATION.

| | 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 |

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.

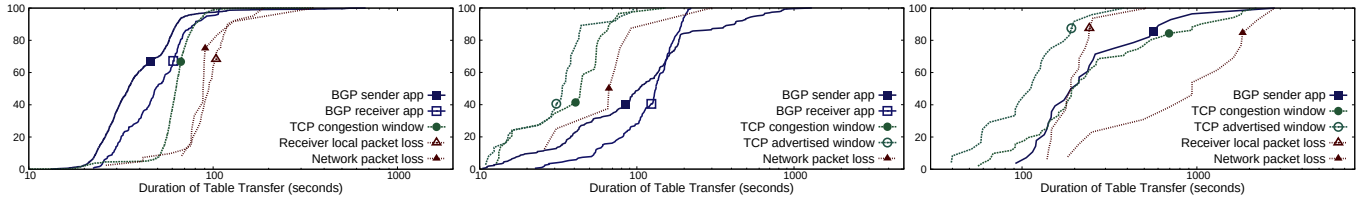
Table II shows that the sender-side factors are the most prevalent, identified as the major factors for 83%, 67%, and 84% of table transfers in the three traces respectively. The receiver-side factors are the second most common, identified as the major factors of 42%, 61%, 43% of table transfers. The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

For each major group, Table II further shows the breakdown results for individual factors. 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 10, 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




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

Fig. 8. Sender-side, receiver-side and network delay ratios of table transfers




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

Fig. 9. Table transfer duration by delay factors. Y axis is the cumulative percentage

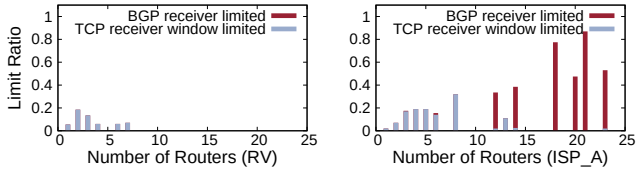



Fig. 10. Affect of concurrent table transfers

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 categorized 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 interesting observation 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 9. 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 pushing 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.

### *B. Detecting BGP Transport Problems*

Based on the identified major delay factors from the previous step, we futher check the packet trace that is affected by these delay factors. This section describes the transport problems we successfully identified, and the high level algorithms to detect and quantify them in the table transfers.

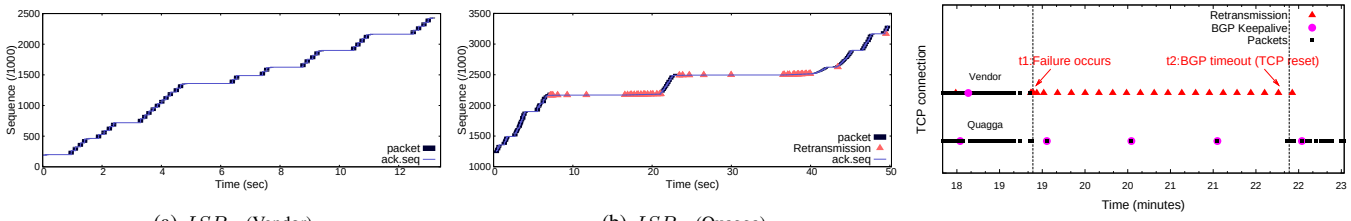
**BGP timer gaps.** Houidi et al. [10] investigate the slow table transfer problem. They observed that *the sender regularly stops sending routes to the receiver and creates gaps in the table transfer* in a VPN provider backbone. They further found that the gaps are caused by the undocumented timer-driven router implementation, which results in sending a limited number of messages per timer expiration. In our dataset, we make similar observations. Figure 11(a) shows an example piece of one BGP table transfer that contains the prolonged gaps between packet transmissions. However, 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 [10] 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.

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 12 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 [10], 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 [20] to automatically identify the knee points in the gap distribution. 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 variation in packet arrivals.

In general, we found that the timer lengths (if exist) are around a few specific values: 80ms, 100ms, 200ms, and

TABLE III
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) |




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

Fig. 11. Transport problems

400ms, while 200ms is the most prevalent. Note that 200ms is the default timer used by a major vendor reported recently in [10]. We could not verify as the authors had not revealed the actual vendor. In Table III, 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.** In our dataset, another common observation is the consecutive TCP retransmissions due to packet loss in a short period of time. Figure 11(b) shows an example piece of table transfer that experiences two episodes of consecutive packets retransmissions. Note that the router attempted to send all these updates at the same time. But due to 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.

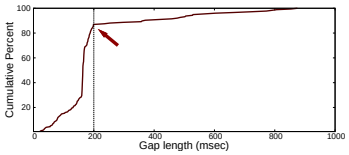
For this problem, we take as input all series that are related to packet losses: *SendLocalLoss*, *RecvLocalLoss* and *NetworkLoss*. We union these three series to construct a new series which captures all instances of packet losses. From such series we check if there exist more than *8* consecutive 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 III 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 II.

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 back-off 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 II (16+13). Note that the TCP retransmission delay is determined collectively by 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.** During the measurement period of May 2008 to April 2009, operational routers in $ISP_A$ peer with both the Quagga and Vendor collector. From the traces, we observe that two connections proceed in a lockstep. That is, the faster connection often pauses and waits for the slower one to catch up. We found and verified with the vendor that this is due to a specific BGP *peer-group* feature [29]. 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 delay could be long particularly upon connection failures as depicted in Figure 11(c). At $t_1$, an error occurred at the Vendor collector, causing the router to keep retransmitting packets, but never being acknowledged[10] 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 timeout took 180 seconds ($t_1 \sim t_2$) in this example and could have significantly hampered the BGP convergence.

To identify these pathological cases, we find table transfers that completely paused due to the failure of another 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. Also, 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

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




| | Timers (ms) |
|---|---|
| ISP$_A$ (Vendor) | 200 400 |
| ISP$_A$ (Quagga) | 100 200 |
| RouteViews | 80 400 |

Fig. 12. Infer BGP timers from the gap distribution

series from two different TCP connections as the following:

$$Quagga.SendAppLimited \cap Vendor.Loss \quad or \tag{5}$$

$$Vendor.SendAppLimited \cap Quagga.Loss \tag{6}$$

As shown in Table III, 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.

So far we demonstrate how to identify transport problems by investigating both packet trace and the series data. Here, the event series themselves can be suffice to 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 specific series:

$$ZeroAckBug \quad := \quad ZeroAdvBndOut \cap UpstreamLoss \tag{7}$$

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 [16]. 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.

## V. Discussion and Future Work

In this section, we discuss the limitation, T-DAT implementation and potential usage .

### A. Limitation and Source of Inaccuracy

To clarify, our 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. This is not a particular problem to our work, but rather a common limitation shared by BGP studies based on the monitoring data [24, 32]. This work contributes to the community in that: first, the identified transport problem (especially those of router implementation and features) shall persist independently 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 to BGP routing behavior and lead to questionable conclusions at best.

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 [11, 14, 23] and [12, 21, 30]. 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.

### B. 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 [11, 17, 21, 30], which can be less effective with respect to their specific goals. Qian et al. [17] 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. [11] 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.

## 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 [22], video streaming [13], online gaming [6], and data centers [7], to name a few. However, until recently,

there has been marginal attention on investigating the BGP over TCP behavior. Xiao et al. [26] model the BGP session survivability under severe TCP congestion. Zhang et al. [31] instead demonstrate a low-rate DoS attack to defeat TCP retransmissions and trigger BGP session resets. Houidi et al. [10] 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 investigate the BGP traces collected from an ISP and BGP monitoring networks. We confirm previous observations and find more potential transport problems, which 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. [30] proposed to identify TCP rate limiting factors for packet flights. Siekkinen et al. [21] addressed the same 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 the context of analyzing BGP. In addition, as rate is a relative measure calculated by transmission size and interval, rate analysis commonly operates on every fixed number (e.g., $\theta$) of packets [30] or interval [21], which could be affected by the selection of $\theta$. One similar work is TCP Critical Path Analysis (CPA) [5]. From TCP packet trace, Barford et al. proposed to construct a path that connects data and ACK packets based on the *happen-before* relationship. Then, each link on the constructed path indicates a particular type of delay. Note that the technique requires the sender's TCP implementation and initial parameters information to accurately simulate the change to TCP windows. Only TCP Reno is illustrated and supported in [5]. T-DAT does not have this requirement. The result may not be as precise as CPA, but can support common TCP versions that adopt window-based congestion control (e.g., TCP Tahoe, Reno, New Reno).

## 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. 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. The tool would be made publicly available at http://irl.cs.ucla.edu/bgpmicro.

## References


[1] Quagga Software Routing Suite. http://www.quagga.net/.
[2] The RouteViews project. http://www.routeviews.org/.
[3] wireshark. http://www.wireshark.org/.
[4] MRT routing information export format. http://www.ietf.org/internet-drafts/draft-ietf-grow-mrt-14.txt, 2011.
[5] P. Barford and M. Crovella. Critical path analysis of TCP transactions. In *Proc. of ACM SIGCOMM*, 2000.
[6] 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.
[7] 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.
[8] P.-c. Cheng, X. Zhao, B. Zhang, and L. Zhang. Longitudinal study of BGP monitor session failures. *SIGCOMM Comput. Commun. Rev.*, 40, 2010.
[9] J. Elson. tcpflow. http://www.circlemud.org/jelson/software/tcpflow.
[10] Z. B. Houidi, M. Meulle, and R. Teixeira. Understanding Slow BGP Routing Table Transfers. In *Proc. of Internet Measurement Conference (IMC)*, 2009.
[11] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proc. of IEEE INFOCOM*, 2004.
[12] 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.
[13] T. Kim and M. Ammar. Receiver buffer requirement for video streaming over TCP. In *Proc. of SPIE*, 2006.
[14] G. Lu and X. Li. On the correspondency between TCP acknowledgment packet and data packet. In *Proc. of Internet Measurement Conference (IMC)*, 2003.
[15] S. Ostermann. tcptrace. http://www.tcptrace.org/.
[16] J. Postel. Transmission Control Protocol. RFC 793 (Standard), Sept. 1981.
[17] 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.
[18] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), Jan. 2006.
[19] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP routing stability of popular destinations. In *Proc. of ACM SIGCOMM IMW*, 2002.
[20] 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.
[21] 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.
[22] J. Touch, J. Heidemann, and K. Obraczka. Analysis of HTTP performance. *ISI Research Report ISI/RR-98-463, USC/ISI*, 1998.
[23] 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*, 2005.
[24] 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.
[25] 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.
[26] L. Xiao, G. He, and K. Nahrstedt. BGP session lifetime modeling in congested networks. *Computer Networks*, 50, 2006.
[27] L. Xiao and K. Nahrstedt. Reliability models and evaluation of internal BGP networks. In *Proc. of IEEE INFOCOM*, 2004.
[28] B. Zhang, V. Kambhampati, M. Lad, D. Massey, and L. Zhang. Identifying BGP routing table transfers. In *Proc. of ACM SIGCOMM workshop (Minet)*, 2005.
[29] R. Zhang and M. Bartell. *BGP Design and Implementation*, chapter Tuning BGP Performance, pages 74–81. 1999.
[30] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of Internet flow rates. In *Proc. of ACM SIGCOMM*, 2002.
[31] Y. Zhang, Z. Mao, and J. Wang. Low-rate TCP-targeted dos attack disrupts internet routing. In *Proc. of Annual Network & Distributed System Security Symposium*, 2007.
[32] 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.