University of California

Los Angeles

Understanding the BGP Transport Delay

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Pei-chun Cheng

2012

The dissertation of Pei-chun Cheng is approved.

Federic Paik Schoenberg

Mario Gerla

Songwu Lu

Lixia Zhang, Committee Chair

University of California, Los Angeles

2012

Dedication To my Grand Parents, Dad, Mom and Sister.

And my beloved Wife Yunyin.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my deepest gratitude to my advisor Dr. Lixia Zhang for her guidance and support throughout my dissertation. From her, I see, and I still try to learn a persistent, open, and inquiring aptitude toward both research and life. I also thank Dr. Beichuan Zhang for his seminal idea on studying BGP session behavior, which starts off to be my integral work in this dissertation. I am also grateful to my committee members: Dr. Mario Gerla, Dr. Songwu Lu, and Dr. Federic Paik Schoenberg for their insightful comments and encouragements during my course of research. The life in Internet Research Laboratory at UCLA has always been delightful. I thank Dr. Mohit Lad and Dr. Ricardo Oliveira for sharing their valuable insights in doing research; Dr. Micheal Meisel and Dan Jen for guiding me through my early days in the group; my best labmate Dr. Jong Han Park for valuable discussions and feedbacks on my study. I would like to acknowledge Shane Amante, Keyur Patel, and John Kemp for their valuable effort to provide essential data source for this work. Finally, I would like to thank my grand parents, mom and dad, who provide me with love, trust, and encouragement; my sister, who takes care of the family while I am away; and most of all, my best friend and wife Yunyin, for her supports and always understanding on this long and colorful journey.

Vita

1978	Born, Kaohsiung, Taiwan.
2000,2002	B.A., M.B.A. (Information Management), National Taiwan University.
2002–2007	Software Engineer, Chunghwa Telecom Co., Ltd.
2008	Intern at Cisco Systems, Inc.
2008–2011	Research Assistant, Computer Science Department, UCLA.
2011	Teaching Assistant, Computer Science Department, UCLA.

Publications and Presentations

Pei-chun Cheng, Xin Zhao, Beichuan Zhang, Lixia Zhang, "BGP Route collection session/collector stability", *IETF 75*, July 2009

Pei-Chun Cheng, Kevin C. Lee, Mario Gerla, Jrme Hrri, "GeoDTN+Nav: Geographic DTN Routing with Navigator Prediction for Urban Vehicular Environments", *Mobile Networks and Applications, Volume 15 Issue 1*, February 2010

Pei-chun Cheng, Xin Zhao, Beichuan Zhang, Lixia Zhang, "Longitudinal study

of BGP monitor session failures", *SIGCOMM Computer Communication Review, Volume 40 Issue 2*, April 2010

Kevin C. Lee, Pei-Chun Cheng, Mario Gerla "GeoCross: A geographic routing protocol in the presence of loops in urban scenarios", *Ad Hoc Networks, Volume 8 Issue 5*, July 2010

Jiangzhe Wang, Eric Osterweil, Chunyi Peng, Chiyu Li, Ryuji Wakikawa, Pei-chun Cheng, Lixia Zhang, "Implementing instant messaging using named data", *AINTEC '10: Proceedings of the Sixth Asian Internet Engineering Conference*, November 2010

Pei-chun Cheng, Jong Han Park, Keyur Patel, Lixia Zhang, "Route flap damping with assured reachability", *AINTEC '10: Proceedings of the Sixth Asian Internet Engineering Conference*, November 2010

Pei-chun Cheng, Jong Han Park, Keyur Patel, Lixia Zhang, "Route Flap Damping with Assured Reachability" *NANOG 51*, Miami, Florida, USA, January 2011

Pei-chun Cheng, Beichuan Zhang, Daniel Massey, Lixia Zhang, "Identifying BGP routing table transfers", *Computer Networks: The International Journal of Computer and Telecommunications Networking*, February 2011

Jaeyoung Choi, Jong Han Park, Pei-chun Cheng, Dorian Kim, Lixia Zhang, "Understanding BGP Next-hop Diversity", *14th IEEE Global Internet Symposium (Infocom workshop)*, April 2011

Pei-chun Cheng, Jong Han Park, Keyur Patel, Shane Amante, Lixia Zhang, "Explaining BGP Slow Table Transfers: Implementing a TCP Delay Analyzer" *NANOG 53*, Philadelphia, PA, USA, October 2011

Abstract of the Dissertation

Understanding the BGP Transport Delay

by

Pei-chun Cheng

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2012

Professor Lixia Zhang, Chair

Border Gateway Protocol (BGP) [35] tied together the Internet's global routing infrastructure. In BGP, routers exchange routing updates to adapt to connectivity changes caused by either intentional routing policy changes or, more commonly, unexpected software and hardware failures. To maintain the continuous reachability, BGP update exchange (or delivery) is expected to be reliable and fast. However, it has long been anecdotal knowledge that such BGP update delivery could be slow, particularly in sending the whole BGP routing tables.

Through a longitudinal assessment on BGP data collected by RouteViews and RIPE for over eight years, we show empirical evidence that BGP table transfer can take up to tens of minutes to complete. The observation is prevalent across data collected from different time and Internet locations. Nevertheless, the explanation of these delays remains unclear, and it presents a considerable challenge to identify and understand the causes behind these slow times.

The main goal of this dissertation is to gain a better understanding of these delay times from the perspective of *BGP transport behavior*. 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 exactly an application running on top of TCP. In this work, we thoroughly investigate the actual BGP/TCP data from operation networks. We find recurring transport problems, including those reported in previous literature and new issues identified in this work that collectively induce significant delay. Such problems are due to various reasons, including implementation issues, router features, and the interaction between BGP and TCP, to name a few.

To further quantify and analyze the delay, we develop T-DAT, a tool 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. Applying T-DAT on TCP trace, we reveal and characterize the principal contributing factors behind delay times in the collected BGP dataset. Identifying these delay factors makes a significant step for ISPs and router vendors to diagnose and improve the BGP table transfer performance.

Admittedly, given the scale and heterogeneity of the current BGP network, no single study may answer every question of BGP transport delay in the wild. This work on understanding BGP transport delay not only identifies and explains BGP slow delivery problem in actual operation networks, but also sheds lights on the need of improving BGP monitoring practice based on observing detail TCP level dynamics. As an important contribution of this work, we develop a BGP analysis tool suite to augment the current BGP monitoring settings and enable in-depth studies of BGP transport behaviors.

CHAPTER 1

Introduction

The Internet today has been built upon the successful deployment of Border Gateway Protocol (BGP), which enables service providers to establish routing among each other and maintain the global reachability. In operation, BGP routers, or speakers, setup BGP sessions with neighboring BGP routers [35]. A BGP session is a bidirectional channel through which routers exchange routing information and update local routing tables accordingly. In practice, neighboring BGP routers are commonly connected with high speed lines, and the BGP routing update exchanges are expected to be fast in general.

However, both the research and operation communities have reported alarmingly long delays in BGP update delivery. Previous works show that it could take 5 up to 15 minutes to send the whole routing table [15, 54], which is suspiciously slow considering the current high network speed. In addition, measurement works reported the system-wise slow BGP convergence time [26], which takes a few to hundreds of seconds. While it is possible to attribute specific delay times to BGP protocol features such as Minimal Route Advertise Interval (MRAI) or Route Flap Damping (RFD), [1] there is still a wide variety of *in-between* delay times remaining unanswered [26].

In this work, we seek to solve these puzzles by studying the BGP transport level behavior. More specifically, we use TCP and BGP data traces to identify

[1] for convergence time shorter than 3 minutes or longer than 30 minutes, respectively

potential causes for the slow data transfers between two peering BGP routers. The significance of understanding the delay times 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 one-hop transfer delays (when they exist) introduce measurement artifacts to the collected BGP data, potentially leading to inaccurate analysis results.

Recently, several related studies have looked into the BGP delay via its interactions with TCP. Xiao et al. [50] show that BGP update delivery performance could be seriously degraded upon repeated TCP retransmissions due to network congestion, which can further lead to BGP session failures in the worst case. Zhang et al. [54] 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 their 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 settings?", to list a few.

In this work, we study the BGP transfer delay problem using the data collected from a large ISP and BGP monitoring projects. We first show that BGP data transfer can be surprisingly slow. Then we investigate in-depth TCP trace of slow transfers. The findings shed lights on interesting on-going BGP transport

problems. We further conduct a systematic TCP delay analysis and identify reasons behind the slow times.

The following dissertation is organized to correspond to each stage of this work. In Chapter 2, we provide an overview of BGP sessions running over TCP connections and the potential issues. In Chapter 3, we conducted a longitudinal measurement study of BGP sessions over 8 years of BGP monitoring data (i.e., application level data). The results show that BGP sessions failed rather frequently, and the table transfers triggered by each session failure could be slow (up to tens of minutes), which reveals the prevalence of potential transport delay. Next in Chapter 4, we further show evidence that BGP update transfers are slow in a large ISP and RouteViews. By investigating TCP packet traces, we find on-going TCP transport problems, which induce delays of different magnitude in BGP update delivery. In Chapter 5, we develop a new tool, T-DAT, to identify and measure different factors behind the transfer delay. We demonstrate the T-DAT's usage and look at the results on explaining durations of slow table transfers. We emphasize that, given the scale and heterogeneity of the current BGP network, the results may not 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 itself is general and may also be used for other TCP-based applications. For the rest of this dissertation, we discuss the proposed analysis tool set in Chapter 6, related works in Chapter 7, and conclude the paper in Chapter 8.

CHAPTER 2

Background

The Internet is made of tens of thousands of different Autonomous System (AS). Border Gateway Protocol(BGP) is the de facto protocol used to distribute reachability information across different ASes. More specifically, BGP runs a global dissemination network, within which each *node* is a *BGP router* and each *link* is a *BGP session* established between two neighboring BGP routers. Figure 2.1 shows an example BGP network that consists of four ASes. BGP sessions established between routers of different ASes called external BGP (e-BGP) sessions while those of routers within the same AS are called internal BGP (i-BGP) sessions. Each BGP session (both e-BGP and i-BGP) is a bi-directional channel for two neighboring routers to exchange routing information. In this chapter, we provide a brief overview of BGP and its requirements on the underlying transport service.

2.1 BGP Sessions

By design, BGP is a hard-state protocol [35]. Every BGP router maintains a persistent routing state with its neighbors. Upon successfully establishing a BGP session, two peering routers first exchanges with each other the whole *routing tables*.[1] After this initial synchronization step, the two peers then only send

[1] A full table contains around 370K routes at the time of this writing

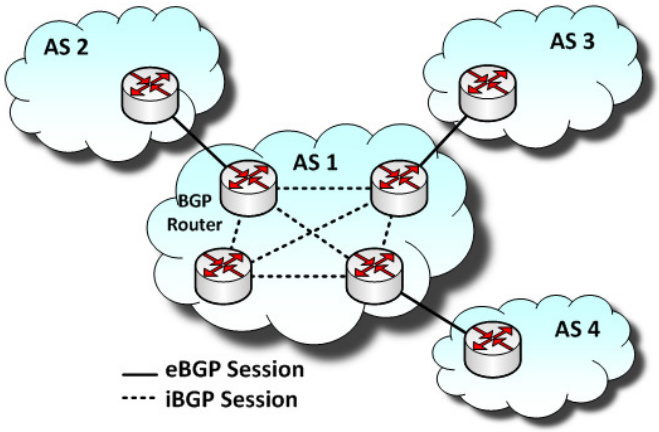


Figure 2.1: BGP router and BGP sessions

incremental updates whenever routing information changes. In this design, the delivery order of BGP updates is crucial. That is, receiving same updates in different orders could result in different routing states. As a result, BGP runs on top of TCP for in-sequence and reliable communication.

A BGP session is expected to be long-lived, and only manually terminated by network operators. However, it may fail due to a variety of reasons, such as malformed BGP updates caused by hardware or software defects, TCP connection problems due to network failures, and prolonged network congestion, etc. Also note that BGP does not rely solely on TCP to detect network failures. BGP employs two timers, *Keepalive* and *Holddown*, whose default values are 60 seconds and 180 seconds respectively. BGP peers send to each other Keepalive messages at every Keepalive timer interval. If no Keepalive message is received before the Holddown timer expires, a BGP router will drop the current session and initiate a new one, which is referred as a *session reset*.

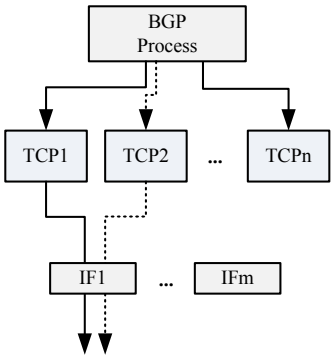


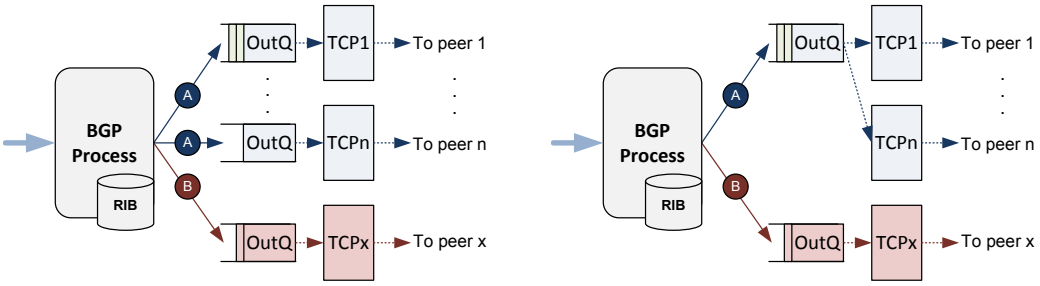
Figure 2.2: BGP/TCP connections and output interfaces

2.2 BGP Transport Issues

As the Internet connectivity becomes denser over time, a BGP router runs an increasing number of peering sessions, both for e-BGP routers between different ASes, and i-BGP routers within large ISPs. In Figure 2.1, each router in AS1 has an external neighbor and three internal neighbors, and BGP updates received from the external neighbor, when changing the routing table, would need to be simultaneously forwarded to other internal neighbors. This synchronized workload imposes bursty stress on the underlying transport service.

This performance issue has long been recognized, and previous works tried to address this problem with different approaches. Jacobson et al. proposed *BGP Scalable Transport (BST)* [31] to solve this problem through application layer multicast. The idea is to maintain a multicast tree for BGP peers that shall receive identical routing updates. Then, one copy of update is generated and forwarded along the multicast tree. This approach has not been widely deployed because of the incurring overhead of maintaining multicast trees.

On the other hand, router vendors address this problem via optimizing the transport performance of the sending router. Figure 2.2 shows an abstract diagram of a BGP router. Each BGP router establishes BGP sessions with all



(a) No Peer-Group

(b) Peer-Group

Figure 2.3: BGP optimization with Peer-Group feature

its peers. Each session is then associated with a TCP connection, while multiple TCP connections could pass through and compete the limited resource (i.e., bandwidth, buffer size) of a handful of physical interfaces. Thus, vendors have suggested guidelines to adjust queue limits and TCP parameters. The general principle is to adjust the queue size, TCP window, and packet size based on the number of BGP peers to prevent packet losses [53]. Moreover, an enhancement feature *Peer-Group* is introduced to optimize the update generation process [53]. In the original BGP implementation, a BGP router has to generate updates individually for each peer. This results in waste of the processing time to generate multiple copies of same updates for peers with the same configuration. To save router CPU for better scalability, Peer-Group is implemented to group together peers with the same export policy. Then updates are generated once per group, and simply replicate to each group member, which substantially reduce the processing overhead. Figure 2.3(a) and figure 2.3(b) show routers with and without using peer-group respectively.

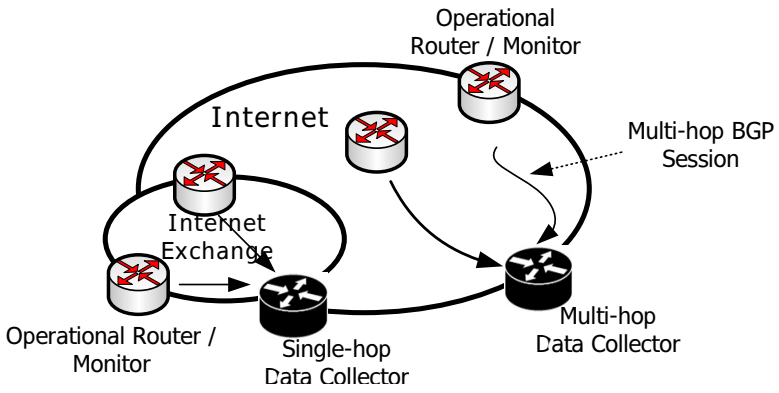


Figure 2.4: BGP monitoring and data collection

2.3 BGP Monitoring

To help understand the Internet routing dynamics, RouteViews and RIPE RIS, the two best known BGP data collection projects, operate a number of *collector boxes* that establish BGP sessions with routers in various networks. We call each operational router connecting to a collector a *monitor* or a *peer*, and the BGP session between the monitor and the collector a *monitoring session*. A monitoring session can be either *single-hop* or *multi-hop* depending on whether the session is across a single or multiple physical hops. As shown in Figure 2.4, single-hop monitoring sessions are usually used in Internet Exchange Points, while multi-hop monitoring sessions are established over wide-area networks.

The collector could be a commodity vendor router or a PC-based Quagga router. [2] The Vendor collector works as a *looking glass* and mainly allows operators to log in and look up the current routing information. The Quagga collector records the received BGP updates in the Multi-threaded Routing Toolkit (MRT) [27] format, which has been widely used in studying BGP behavior [37,38].

[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 [34]

The Quagga collectors receive BGP routing updates from peers and save the collected BGP updates into files every 15 minutes (RouteViews) or every 5 minutes (RIPE) in the Multi-threaded Routing Toolkit (MRT) [27] format. These files are then made publicly available. The collectors also dump snapshots of the BGP routing table, the RIB, for each of its peers every two hours in the MRT format. Over years, this data has become an essential asset for research community to help understand various aspects of the global routing system. However, note that such BGP data only includes the *application layer routing message*; it does not contain detail TCP layer information. In this work, we would show the limitation of this setting in revealing various important BGP transport problems.

CHAPTER 3

Longitudinal Study of BGP Session Resets and Delays

Since 1999, BGP routing information collected by RouteViews and RIPE RIS has become an indispensable asset for the research community to help understand various aspects of the global routing system, such as Internet topology [43], BGP convergence [29], ISP peering policies [13], and prefix hijack monitoring [21], to name just a few. In this chapter, we start off this work by leveraging such BGP monitoring data. The main challenge in that, the goal is to understand the transport related issues while the BGP monitoring data only contains application level routing updates. Given this mismatch, we propose to *infer BGP session resets and delays from the application updates, and characterize high-level transport behaviors*. The idea is based on the BGP design that the whole process of session reset mainly involves two neighboring routers. The delay times during the session resets thus could indicate the potential transport issues between two routers.

Figure 3.1 illustrates a simplified timeline of BGP session reset behavior from the collector's perspective. Assuming that a monitor has a routing table of 5 prefixes, First, three incremental BGP updates (for prefixes $p1$, $p2$, $p3$) are received at time 10, 14, and 17, respectively. Then the session fails at time 17 and restarts at time 22. The session re-establishment takes time from 22 to 25, during which the collector records three *state messages*. The state message $s1$

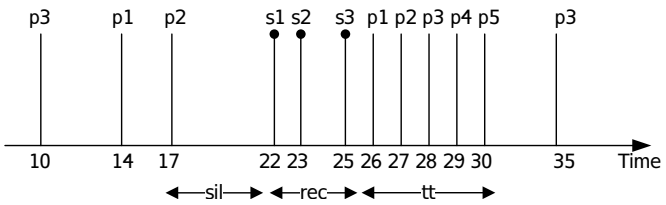


Figure 3.1: BGP message stream (sil: silence period, rec: session re-establishment, tt: table transfer duration)

marks the time when a router initiates a BGP session, while $s3$ marks the time when the session is fully established. We show three state messages here for illustration purpose; in reality establishing a BGP session may require more state changes [35]. Following state messages are the table transfer updates during time period 26 to 30, which include the entire routing table entries ($p1$ to $p5$), followed by incremental updates afterward. We refer the duration (17, 22), (22,25) and (26,30) as *silence period*, *recovery period*, and *table transfer period*, respectively. These three periods all leave time gaps in the BGP routing data. Our goal is to measure these delays and especially focuses on the table transfer duration as it reflects the underlying BGP transport behavior.

Interestingly, note that BGP session resets are commonly considered harmful, as they lead to both missing and duplicate updates during session re-establishment, making analysis results derived from such data inaccurate. Here in this work, we instead utilize these undesired BGP resets to help understand the BGP transport behavior. In the following sections, we present the first systematic assessment and documentary on BGP session failures and delay times of RouteViews and RIPE data collectors from 2001 to 2009. Our results show that monitoring session failures are rather frequent, more than 30% of BGP monitoring sessions experienced at least one failure every month. The table transfers after each session reset could be slow (up to tens of minutes), which indicates the possibility

Table 3.1: BGP monitoring data sources

Collector	Type	Start Date	Location
RRC00	Multi-hop	2001 Jan	Amsterdam
RRC01	Single-hop	2001 Jan	London
RRC02	Single-hop	2001 Mar	Paris
OREG	Multi-hop	2001 Oct	Oregon
LINX	Single-hop	2004 Mar	London
EQIX	Single-hop	2004 May	Ashburn

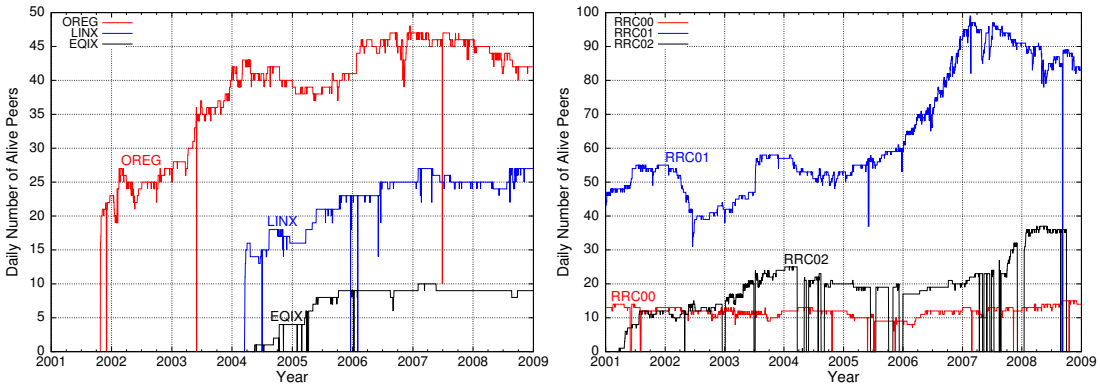
of transport level delays. Furthermore, we observed failures that happen across multiple peer sessions on the same collector around the same time, suggesting that the collector's local failures are a major factor in the session instability.

3.1 Inferring Session Reset

In this section, we discuss data sources and our approach of identifying session resets and delays in the BGP data.

3.1.1 Data Sources

RouteViews and RIPE started collecting BGP data in the late 1990's, and went through a learning period in the first few years before the data collection process stabilized. In this chapter, we use the data from January 2001 onward to December 2009. We take data from six collectors at different locations. The collector information is summarized in Table 3.1. For the eight years period, Figure 3.2 shows the number of peers at each of collector over time. For each day we count the number of unique peers that logged BGP data. The figures show that we cover a representative number of routers in the data (more than 100 routers) The



(a) RouteViews (b) RIPE

Figure 3.2: Number of alive BGP monitors over time

downward spikes in the figures mean that a large number of peers did not log any data on those days, which could be caused by collector outage or maintenance. We will discuss collector failures in the following section.

3.1.2 MCT Algorithm

As Figure 3.1 suggests, session state messages $(s1, s2, s3)$ could potentially help identify session resets. Unfortunately, state messages are only logged by RIPE, but not by RouteViews. Also, state messages do not identify the duration of the following table transfer.

In [47] Wang et al. used syslog messages to detect failures of BGP sessions in a tier-1 ISP. However, syslog information is not available from RIPE or RouteViews collectors. For RIPE, it offers process log for Quagga collectors [34], but such log does not explicitly record BGP session resets. As for RouteViews, it offers Rancid log, but the log is only generated once per hour. As the result, we cannot rely on these logs as the primary method of detecting session resets.

In [51] Zhang et al. developed an algorithm called Minimum Collection Time

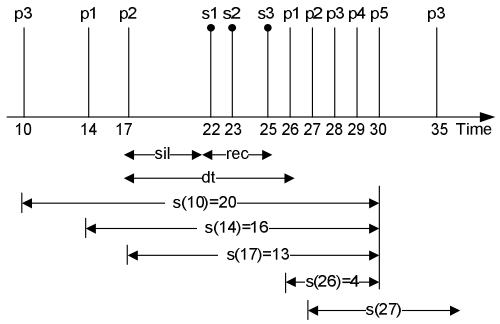


Figure 3.3: Message stream and collection time

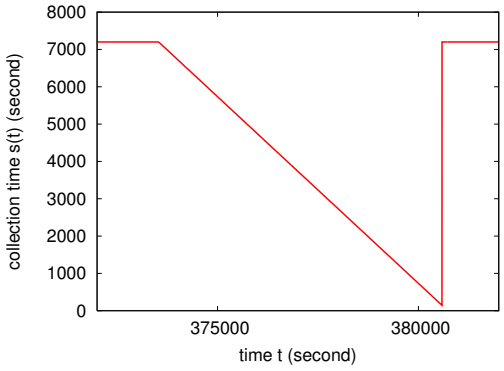


Figure 3.4: Sample collection time

(MCT) that can identify the start and the duration of table transfers from BGP data in the absence of state messages. Based on the fact that all prefixes in the routing table are announced during a table transfer, MCT searches for the smallest time window during which the full table is announced. Using three months of data from 14 different monitored peers, this method successfully detected over 94% of session resets[1]. The strength of MCT is that it only relies on regular BGP updates and has been shown as a practical way to identify session resets in both RIPE and RouteViews BGP data. In this chapter, we further improve the MCT algorithm for even better detection accuracy.

3.1.2.1 Minimum Collection Time

The Minimum Collection Time (MCT) algorithm is based on the observation that, "during a table transfer, *all the prefixes* in the routing table would be announced within a relatively short period of time" [51]. Thus, given the message steam in Figure 3.3, for update (not state message) received at time t, MCT calculates the *collection time*, $s(t)$, as the time it takes for *all* five prefixes to be

[1]The false positive in [51] is lower than 5%.

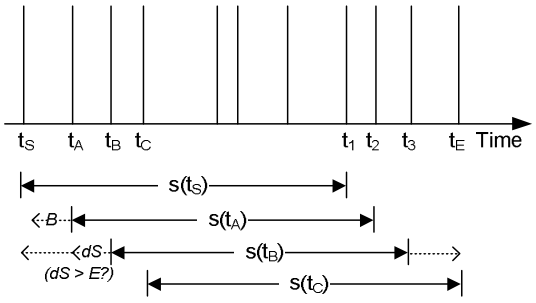


Figure 3.5: Bottom searching

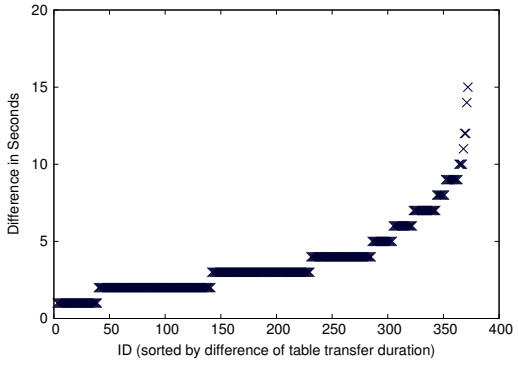


Figure 3.6: Difference of identified table transfer duration

announced. For example, if we assume that the entire routing table consists of entries for five prefixes ($p1$ to $p5$), and the first update of prefix $p3$ arrived at time 10, then the collection time $s(10)$ would be $30 - 10 = 20$, since it takes until time 30 for all five prefixes to be announced. We can further calculate collection times for all the following updates. Note that after time 27, the collection time becomes infinite, since we could not collect all five prefixes afterward.

One important observation about $s(t)$ is that, ideally the collection time for a given update stream shows a monotonically decreasing and then increasing trend, and the local minimum of collection time indicates the true duration of table transfer. Therefore, we can identify the occurrence of table transfer, and thus the session reset, by simply looking for the local minimum of collection time. [2]. for updates arrived before the beginning of the table transfer, $s(t)$ decreases steadily until reaching a minimum value $s(26) = 4$, which indicates the true duration of table transfer. After this minimum value, $s(t)$ steadily increases. From this monotonically decreasing and increasing trend, In reality, to handle

[2]MCT presume a upper limit of 7200 seconds for $s(t)$.

uncertain timing and ordering of updates messages, MCT also introduces several simple tune-ups. We refer readers to [51] for more details about MCT.

3.1.2.2 MCT Improvements

We made two improvements over MCT: better BGP table size estimation and bottom search scheme. First, the MCT results reported in [51] used one sample BGP RIB size as the estimate of the routing table size for each month. This estimation may be imprecise because of the growing or shrinking of the routing table over time. To improve the estimation accuracy, we calculate the table size every day, which gives more accurate table size estimation.

Second, MCT assumes that an update with local minimum collection time flags the start of a full table transfer. However, because the minimum collection time is calculated based on the *estimate* table size, which could be smaller than the *actual* table size, MCT might identify the start of a table transfer with few seconds offset. For example, in figure 3.5, the table transfer starts at t_S and ends at t_E. Assume that the expected table size calculated by MCT is 3 entries smaller than the actual table size. Then, MCT would prematurely conclude t_S to t_1 as one potential table transfer, and so as t_A to t_2, t_B to t_3, and t_C to t_E. Any one of these four could be identified as the start of the table transfer.

To better estimate the start of table transfer, MCT addresses this problem by adopting a backward bottom searching, which looks backward in time for a fixed number of seconds to locate the true start of a table transfer. In Figure 3.5, assume that MCT identifies a minimum collection time $s(t_A)$ at time t_A, then it searches backward in time to locate the earliest update between $t_A - B$ and t_A, and makes this update the new start of table transfer. Note that this backward bottom search allows MCT to better locate the true *start* time, but it may still

miss the true *end* of table transfer.

Thus, we apply *bidirectional bottom searching thresholds* to search both forward and backward. And instead a fixed threshold, we refine the table transfer duration until it finds a relatively large gap E in the collection time.[3] Note that with bidirectional bottom searching, our algorithm is expected to detect session resets with same accuracy compared with the original MCT, but with more accurate estimate of the start time and duration of table transfer. For ease of discussion, we refer our algorithms as eMCT. The high level procedures of eMCT is summarized as follows.

1. Estimate the *table size* for every given day.
2. Calculate *collection time* for all updates based on the expected table size, and find all *local minimum* using MCT.
3. For each local minimum, search both *backward and forward* in time to locate the true start and end of table transfer.

By applying MCT and eMCT on three months of RRC00 data from 2002 Jan to 2002 March, we found that both of them detect the same 510 session resets followed by full table transfers, while eMCT identifies slightly longer table transfer duration. Figure 3.6 shows the difference in seconds, between the table transfer durations detected by eMCT and MCT for the same session reset. For most of the cases, the difference is less than 10 seconds, while there are five cases whose difference is greater than 10 seconds. After manual inspection, we verified that eMCT correctly identified the true table transfer duration, while MCT identified shorter table transfer durations due to imprecise table size estimate.

[3]From our experiments, we set $E = 3$ seconds which seems effective enough in finding true table transfer duration.

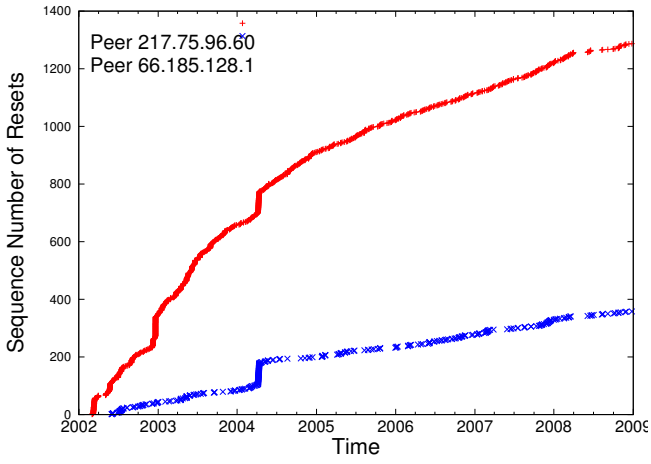


Figure 3.7: Session resets of two example monitors over time

3.2 Session Resets over Time

We present the overall BGP session reset statistics in this section, and then investigate the causes of session resets in the next section.

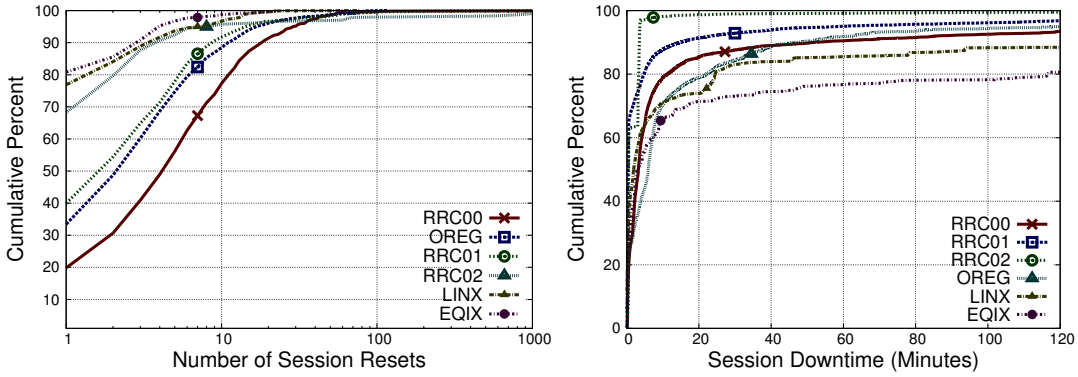
First, since data collectors only passively receive BGP updates from their peering monitors and are not involved in forwarding data traffic, the monitoring sessions between data collectors and monitors have simple configuration, low workload, and requires little maintenance. Thus the monitoring sessions are expected to be stable and long lived, and users of BGP data usually do not pay much attention to possible session resets during their measurement periods.

Our results, however, show that monitoring session resets are relatively frequent. Figure 3.7 shows the cumulative number of resets for two monitoring sessions at the OREG collector, 66.185.128.1 and 217.75.96.60, over the past eight years. The session with 66.185.128.1 has 4.5 resets per month on average, a typical case among the sessions at OREG. The session with 217.75.96.60 is the worst case at OREG, averaging 15.8 resets per month. Although some months

have more BGP session resets than others, overall the resets occur persistently over time.

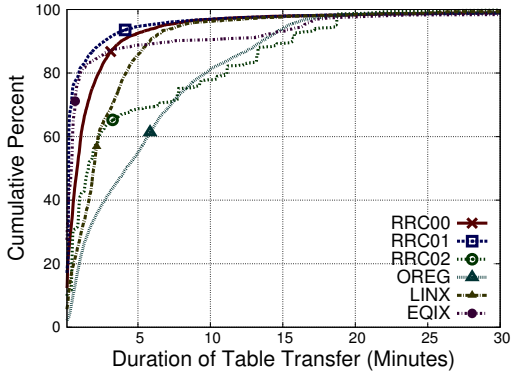
The observation is prevalent across all the collectors, regardless of the type of the session (single-hop or multi-hop), the age of the collector, or its location. Figure 3.8(a) shows the cumulative distribution of the number of resets per peer per month for all the 6 monitors we measured. For the two multi-hop collectors, OREG and RRC00, 10-20% session-months do not have any reset, while the 50-percentile is 3 resets, and the 90-percentile is 12 to 15 resets per session-month. The worst case at OREG is a monitor that had 117 resets in one month, while one of the RRC00 peers had 4205 resets in one month. The single-hop collectors have fewer resets, but the numbers are still alarming. RRC01 and RRC02 also have some sessions that had thousands of resets in a month. These cases were likely caused by hardware problems or mis-configurations that made the sessions up and down constantly before they were fixed.

For each session reset, we then measure the session delay times, including the session downtime and the following table transfers. Here, *session downtime* is defined as the time between when the failure is detected and when the BGP session is fully re-established. Since the failure itself is not logged in the BGP data, we estimate session downtime from the last BGP update preceding a reset and the first BGP update after the session re-establishment as illustrated in Figure 3.1. In Figure 3.8(b), we observe that the majority of session downtimes are within ten minutes, but some cases are much longer. For example, at OREG the session downtime has a 25-percentiles at 1 minute, 50-percentiles at 6 minutes, and 90-percentiles at 48 minutes. All collectors have cases in which the session downtimes are more than 10 days. Users of BGP data can easily spot very long session downtimes (e.g., , days) and take precautions accordingly in their data



(a) Num. Session Resets, per router-month

(b) Session Downtime



(c) Duration of Table Transfers

Figure 3.8: Characteristics of BGP session resets and table transfers

processing. However, given that majority of the session downtimes are within 10 minutes, without knowing the existence of session resets, it is difficult for the BGP data users to identify these short durations of quiet periods as data missing and take proper measures accordingly.

Figure 3.8(c) shows the cumulative distribution of table transfer duration. We observe that over 90% of table transfers finish within around 5 minutes, while table transfers at OREG tend to take longer time to finish, with 50-percentile at 4.5 minutes and 90-percentile at 14 minutes. This is surprisingly slow considering that a routing table usually contains less than 10MB of data. We further calculate

and find that the table transfer time is not significantly correlated with the routing table size, which indicates that the link bandwidth is not the limiting factor. Recently, Houidi et al. [15] reported that slow table transfers are potentially due to routers' timer-driven implementation. We will further discuss the BGP transport problems in the following chapters.

The main point to take away from this section is that the BGP monitoring session resets occur frequently, averaging a few times per peer per month across all the 8 years and 6 collectors that we have examined. Majority of session down-times last within 10 minutes and the following table transfers usually complete within another few more minutes, during this time period actual BGP updates are missing and superfluous table transfer updates are introduced. There exist extreme cases with thousands of resets in a month, or downtime for multiple days, or tens of minutes or longer to finish a table transfer. From the transport point of view, these prolonged delay times are suspicious and call for further analysis, which is the main task for the reset of this work.

3.3 Discussions on Session Resets

In this section, we investigate the potentially causes of observed session resets and delay times.

3.3.1 Collector Failures

Maintaining a stable data collecting service is critical to the quality of logged BGP data. Collecting services may be disrupted by hardware defects, software bugs, network problems, or planned maintenance. For example, RouteViews has reported sporadic collector outages owing to interface malfunctions, memory

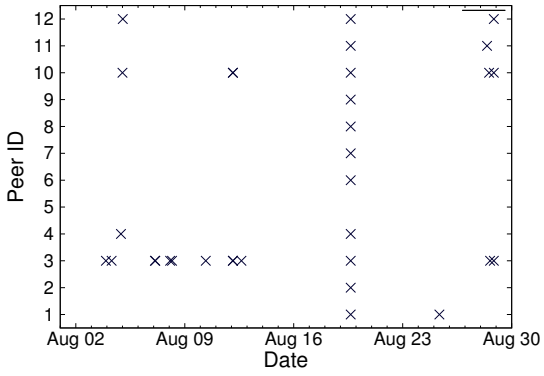


Figure 3.9: An example of synchronized resets

problems, fiber cuts, software upgrades, and other problems [40]. RIPE also occasionally announces degraded service for maintenance [37]. Unfortunately, neither RIPE nor RouteViews maintains complete information about collector outages. In this section, we identify collector problems by correlating session resets on the same collector.

3.3.1.1 Correlating Session Resets

From the session resets identified in the previous section, we find that a collector's session resets across different peers are sometimes clustered within a short time window. For example, Figure 3.9 shows the session resets for RRC00 during August, 2003. On August 19th, almost all peers had session resets. This implies that the collector itself might have experienced a problem.

We define *synchronized session resets* of a collector as a group of resets occurring within a time window w, *synchronized peers* as the peers appearing in synchronized resets, and *synchronization ratio* as the ratio of the number of synchronized peers to the number of total alive peers of the collector at that time. For example, if five out of ten peers have resets within w, these five resets are syn-

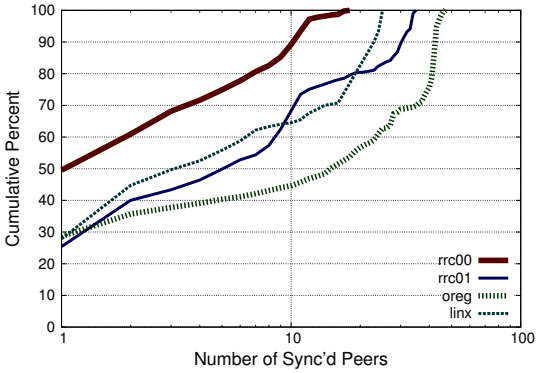


Figure 3.10: CDF of Sync'd Peers

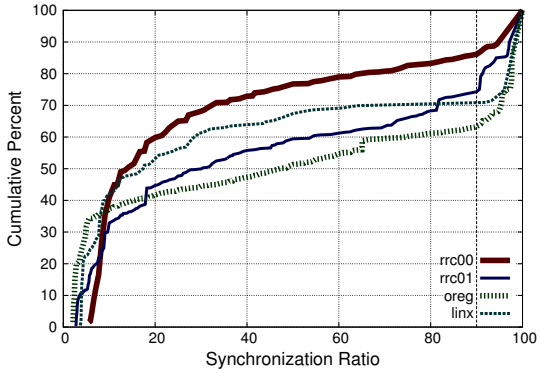


Figure 3.11: CDF of Sync Ratio

chronized resets associated with five synchronized peers, and the synchronization ratio is 0.5.

Figure 3.10 shows the cumulative distribution for the number of synchronized peers for four collectors [4]. For RRC00, about half of the session resets are standalone (i.e., , the number of synchronized peer is 1), and the rest of resets are synchronized to some extent. For other collectors, synchronized resets contribute to more than 70% of all the resets. There is a sharp increase near the tail of the curve, indicating that a significant number of session resets involves most or all peers. Figure 3.11 shows the cumulative distribution of the synchronization ratio. There is a sharp increase among all the four collectors between 0% to 10%. This is because the collectors usually have 10 to 20 concurrently alive peers, which leads to a lower bound on the synchronization ratio of approximately 5% to 10%. After the synchronization ratio passes 90%, there is another sharp increase, which accounts for 10% to 30% of the total session resets.

[4]We use four example collectors to demonstrate the distributions of synchronized resets.

3.3.1.2 Identifying Collector Problems

We assume that if all or most peers have session resets at the same time, the cause is likely to be a local problem at or near the collector. We name such a problem "collector-restart", even though the session resets can be due to different local problems, such as a collector machine reboot, a BGP daemon restart, or network connectivity problems, and so on.

We use a 90% of synchronization ratio as the threshold to detect collector-restart and require that there must be at least five alive peers. As a result, we detected 72 collector-restarts at RRC00 from August 2002 to December 2008. August 2002 is used as the starting time because RIPE started to archive the process log of the collector daemon at that time. The process log records the termination and start of the collector process, and thus can be used to verify our detection results. After matching the *observed* restarts against those *recorded* in collector process logs, we find 7 observed collector restarts that are detected by our method but not recorded in collector process logs. Further inspection finds that 5 of the 7 cases are due to errors in the collector log and 2 cases are due to a large number of BGP re-connections in a short time, which might be caused by network instability. There are also 22 collector restarts that are recorded in collector process logs but our scheme failed to detect. Among these cases, 2 are due to two consecutive collector restarts, so that there is no BGP session successfully established in between. The other 20 cases are due to some peers that disconnected or failed but are still counted as active, so that a collector could not successfully re-establish sessions to these peers after collector restart, and results in a synchronization ratio which is lower than our 90% threshold. Overall this simple algorithm yields over 95% correctness and detects 80% of collector restarts.

Table 3.2: Session resets on collector restarts

collector	no. restarts	no. session resets (%)
RRC00	105	1154 (14%)
RRC01	112	1999 (26%)
RRC02	-	-
OREG	178	6370 (37%)
LINX	29	673 (30%)
EQIX	9	69 (14%)

Table 3.2 shows the number of collector-restarts detected at each collector along with the number of session resets triggered by these restarts. We can see that 14% to 37% of session resets are caused by collector-restarts [5]. The problem is more pronounced for collectors that have many peers, such as OREG, for which 37% of session resets are due to local problems at the collector. Since collectors' local problems are a major contributor to session failures, it is important to improve the stability of the collector, including its network connectivity, software and hardware, in order to reduce monitoring session failures.

3.3.2 BGP Timer Settings

In October 2002 RIPE disables all its collectors' BGP Keepalive/Holddown timers. This was due to the observation that, during periodic RIB archiving, some old collectors stopped sending BGP messages, causing BGP sessions to timeout and triggering a surge of session resets. To alleviate this problem, RIPE disabled BGP timers. However afterward it was noticed that disabling Keepalive/Holddown timers caused BGP to lose the ability to detect connectivity problems such as link failures, and thus introduced long, unexpected session downtimes. Since

[5]Since RRC02 sessions are quite small in general, the number of session resets is not large enough to conclude a collector restart by using synchronization ratio.

Table 3.3: RIPE BGP timer settings

Time Period	Keepalive	Holddown
Before 2002 Oct 17	60 sec	180 sec
After 2002 Oct 17 Before 2006 Nov 23	0 sec	0 sec
After 2006 Nov 23	60 sec	180 sec

later collector software fixed the BGP message blocking problem during RIB archiving, RIPE restored the BGP timers on all its collectors in November 2006. Table 3.3 summarizes the timer settings for RIPE; note that a value of 0 disables a timer. In this section, we document and quantify the impacts of changing BGP Keepalive/Holddown timers on the stability of RIPE monitoring sessions.

One issue we observed is that, while RIPE's plan was to disable the timers for all the BGP monitoring sessions, the Keepalive/Holddown timers for some peers were never turned off. This could be due to the fact that a zero timer value was not allowed on some Juniper routers as of 2002, or due to misconfiguration, which we will discuss later.

No matter what may be the cause, to measure the impact of disabling BGP timers, we need to differentiate between BGP sessions that have the timers disabled, and those that have the timers enabled. We define *Keepalive-enabled (KAE)* sessions as BGP sessions that have the Keepalive timer enabled, and *Keepalive-disabled (KAD)* sessions as the sessions that have the timer turned off.

3.3.2.1 Identifying KAE/KAD Sessions

Differentiating between KAD and KAE sessions poses a challenge since RIPE does not keep historical records for collector configurations. In this section, we proposed a heuristic method to distinguish these two kinds of sessions.

3.3.2.2 Collecting Session Downtimes

For each identified session reset, we further compute the corresponding session downtime. we define *session downtime* as the period during which a failed session has been detected and fully re-established. thus, a session downtime can be further divided into a *silence period* followed by a *recovery period*, which has been briefly described in Figure 3.1.

We define *silence period*, preceding a session re-establishment, as the duration when a failed session remains silent. we measure it as the difference between the timestamp of the last seen regular BGP update and the first succeeding session state message. Figure 3 shows an example silence period, *sil*, between time 17 and 22. ideally, silence periods indicate how long does it take for a data collector to detect failures. when a data collector receives malformed BGP messages from its peers, it may pro-actively reset the session, during which the silence period is extremely short. however, for cases such as link failures, a collector may have to wait for the Holddown timer to expire. this may take more than 90 seconds or 180 seconds depending on the timer configuration. note that the *real* silence period may be shorter than our *measured* period. this is because BGP keep-alive messages are not logged by data collectors. some keep-alive messages might be received from a peer after the last seen BGP updates. for these cases, our measured silence periods serve as the upper bound of the real silence periods.

We define *recovery period* as the duration of BGP session re-establishment. we measure it as the difference between the timestamp of the first session state message and the last session state message which basically mark the begin and end of BGP session establishing process. figure 3.3 shows an example recovery period, *rec*, between time 22 and 25. in general, a BGP session establishment involves setting up a TCP connection, handshaking timers, and negotiating BGP

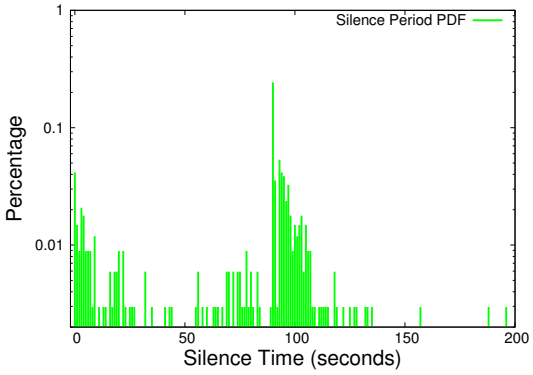


Figure 3.12: Sample silence period distribution

capabilities [35], which requires at least several seconds to recover a BGP session. For failures such as link congestion, a session establishment may include several unsuccessful connection attempts. BGP adapts an exponential back-off approach to increase the interval between each reconnection, which might significantly increase the recovery period.

Based on these definitions, the idea is to infer the BGP Holddown timer value based on the distribution of session downtime. For session resets triggered by Holddown Timer expiration, the duration of silence period should be close to the length of the Holddown Timer. Figure 3.12 shows the distribution of silence time for session resets from an example RRC00 session with 90 second Holddown Timer, which shows that a significant number of session resets are associated with a 90 second silence period. We then identify KAE sessions as those with a single silence period duration length which is associated with more than 10% of session resets. This 10% threshold is chosen conservatively based on the measurement result in [47], which observed that more than 20% of session resets are triggered by the expiration of BGP Holddown Timers.

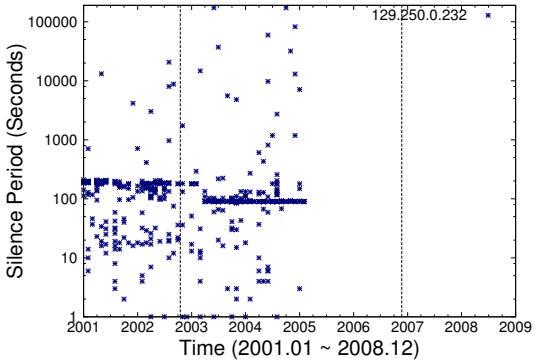


Figure 3.13: KAE silence period

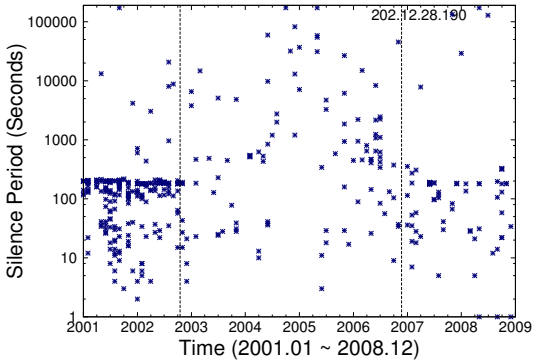


Figure 3.14: KAD silence period

Table 3.4: KAE / KAD Peers

Collector	Total Peers	KAE	KAD
RRC00	42	9	33
RRC01	57	5	52
RRC02	15	2	13

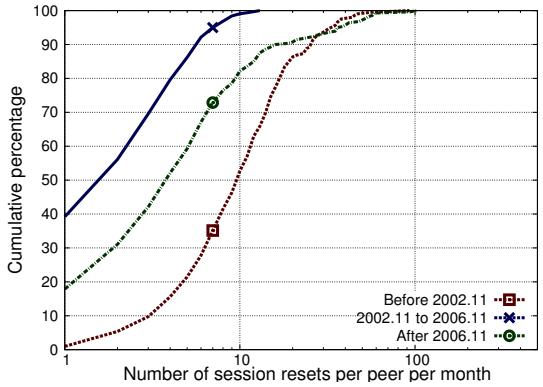
Applying this algorithm on RRC00 data, we identified 9 KAE sessions out of total 42 BGP sessions. Figure 3.13 and Figure 3.14 show the distribution of silence time for one identified KAE session and one KAD session, respectively. The vertical lines mark the dates when RIPE disabled and enabled BGP timers. These two figures verify that, after RIPE disabled timers on Oct 17, 2002, the identified KAE session continued to trigger session resets after a 90 second silent period, but the KAD session did not. Table 3.4 summarizes the inference results for three RIPE collectors. In the remaining of this section we only consider session resets from the KAD sessions.

3.3.2.3 Number of Session Resets

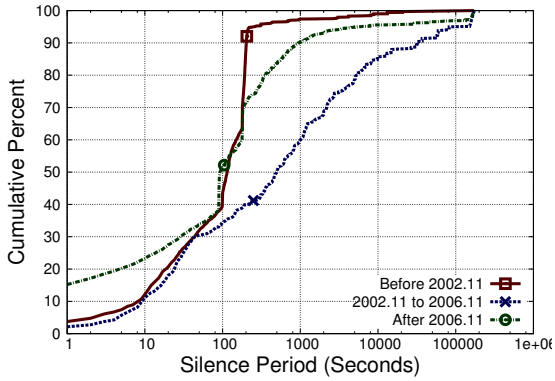
We first measure the number of session resets before and after disabling timers. Figure 3.15(a) shows the cumulative distribution of the number of session resets per month for KAD sessions. We group session resets into three periods based on the dates RIPE disabled and enabled timers: *Before 2002.11*, *2002.11 to 2006.11*, and *After 2006.11*. After disabling BGP timers in 2002.11, we can observe a left shift of the distribution, which indicates a drop in the number of session resets. The median number of session resets of "*Before 2002.11*" is about 4 times of that of "*2002.11 to 2006.11*". This shows that disabling BGP timers did reduce the number of session resets. After 2006.11, when RIPE restored the timers, the distribution shifts right, but with a smaller magnitude. This is because the newer version of the collector software fixed the BGP message blocking problem during RIB archiving. Thus there should not be as many resets as before Nov, 2002. We observed a similar distribution of the number of session resets for other RIPE collectors.

3.3.2.4 Session Downtime

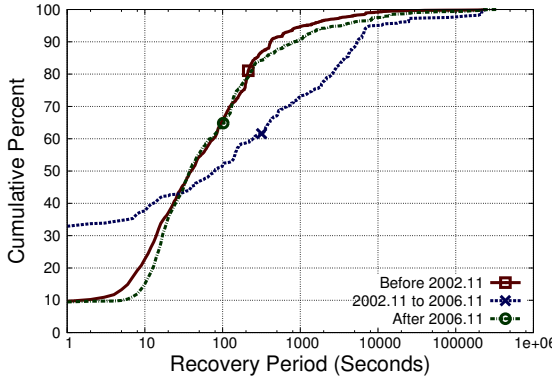
In this section, we measure the *silence period* and *recovery period* for the unnoticed side effect of disabling BGP timers. Figure 3.15(b) shows the CDF of the silence period for KAD sessions. Before disabling BGP timers, there are two consecutive sharp jumps at around 90 and 180 seconds silence time, which represent session resets trigger by Holddown timers with 90-second and 180-second values. After disabling Keepalive timers, these two jumps basically disappeared and the CDF of the silence period began to follow a long-tail distribution. This is because, with Keepalive timers disabled, BGP sessions could no longer detect failures by the timeout interval. These failures either went on unnoticed, or were



(a) Num. Session Resets



(b) Silence Period



(c) Recovery Period

Figure 3.15: Impact of disabling keepalive timer, RRC00.

eventually detected by external signals such as TCP errors, at much later time.

Figure 3.15(c) shows the cumulative percentage of recovery time for session resets. We observed that disabling BGP timers did change the distribution of recovery time. This seems counter-intuitive because Keepalive/ Holddown timers are expected to only affect the silence time but not the recovery time. One possible explanation is that, though disabling timers does not change the recovery time for a given session failure, it could potentially change the *visibility* of some session failures. More specifically, [47] observed that session failures can mainly be categorized into 4 groups. The first and second groups contains failures such as *admin*

resets and *peer closed sessions*, these types of resets can recover fast. The third group contains *local holddown timer expired*, which results in moderate downtime. The fourth group contains *local router shutdown* and *peer de-configured*, which have very long recovery times. As a result, disabling Keepalive timers would make a BGP session *blind* to the third group of failures, and skew the distribution of recovery time towards the other three groups, which have either much shorter or longer recovery times. This explains the increase in percentage of both short recovery times and long recovery times in Figure 3.15(c).

In this section, we analyzed RIPE BGP data to show that disabling Keepalive timers indeed reduced the number of session resets. At the same time, it also led to a long-tail distribution of session silence time, during which session failures went unnoticed and real BGP updates were lost. Thus we recommend not to disable Keepalive and Holddown timers, even though this is allowed in the BGP specification [35]. In addition, when interpreting historical RIPE data, users need to be aware that long silence times might be the result of unnoticed BGP session failures, rather than live BGP sessions suddenly became quiet.

3.4 Summary

In this chapter, we report the first systematic assessment on the BGP session failures and the associated delays of RouteViews and RIPE data collectors over the eight years. The results show that failures of the BGP monitoring sessions are relatively frequent, averaging a few session resets per monitor per month. How to make BGP sessions robust against transient packet losses remains an open problem both in BGP monitoring projects and operational networks. The measurement also shows that failures local to the data collectors contributed between 14% to 37% of the total session resets. Although some cases could be due

to intended administrative maintenance, they nevertheless affect the quality of the data being collected. In the process of analyzing BGP session resets using the historical data, we also found that disabling BGP's Keepalive timer leads to negative consequence of unnoticed session failures. We proposed an efficient algorithm to detect ISP peers that turned off BGP timers. Users of historical RIPE BGP data should take into account the potential long downtime and missing updates for the affected peers in order to achieve reliable results. More importantly, this chapter shows the prevalence of BGP transport delay and raises an important open question: *What are the causes behind the slow table transfer duration?* For the rest of this dissertation, the main task is to explain these BGP transfer delay times.

CHAPTER 4

Diagnose BGP transport problems

In Chapter 3, we make a common observation that BGP table transfer is slow. The observation is prevalent across BGP data collected at different topological locations. In this chapter, the main idea is to search deeply from the TCP perspective, the explanations behind BGP slow transfers. This is challenging as virtually no BGP over TCP data is publicly available at the time of this study. Also, researchers commonly do not have direct access to the operational routers nor the router source code. As a result, BGP researches mostly focus on understanding BGP application level behaviors using BGP data [7, 36, 48, 52], [1] or study BGP transport issues using controlled environment or simulations [12, 15, 49].

In this chapter, we seek to identify actual transport problems by leveraging TCP trace provided by one courtesy large service provider and RouteViews. We first discuss our approach and data sources, and then report a number of identified BGP transport problems. To clarify, the analysis in this chapter does not intend to be complete nor system-wise prevalent, as the dataset inherently represents a limited view of the entire BGP network. The goal is to demonstrate the real ongoing problems that otherwise run unnoticed, and discuss their impact on BGP research and operations.

[1] collected by RIPE [28] or RouteViews [38] as mentioned in Section 2.3

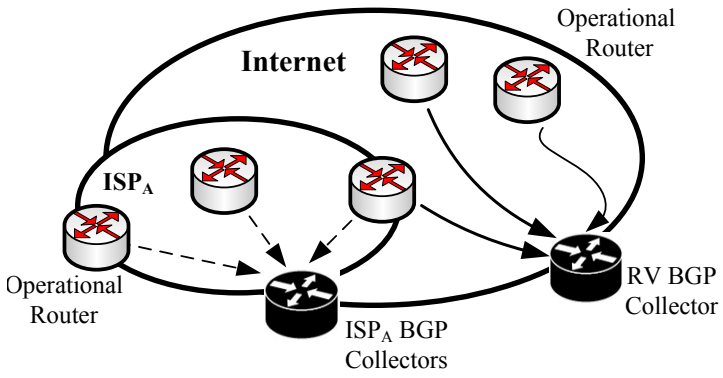


Figure 4.1: ISP$_A$ and RouteViews BGP monitoring.

4.1 Data Characterization

We first describe the data collected and its high level flow characteristics.

4.1.1 Datasets

We use BGP and TCP data collected at a large ISP (ISP_A) and the RouteViews [38] project. The collection setting is similar to the BGP monitoring setting discussed in 2.3. As depicted in Figure 4.1, *BGP data collectors* are deployed to peer with operational routers and passively receive BGP messages.

ISP_A deployed, at the same site, one Vendor collector and one Quagga collector, while the Vendor collector only collected data from one year from 2008 to 2009. Each BGP collector peers with around 25 BGP routers, including the closest router in the same pop to the farthest one in a different continent. RouteViews deployed multiple collectors across the Internet, the particular one used in this work is a vendor collector located in University of Oregon, Eugene, USA.

As shown in Figure 4.2, in addition to BGP update 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

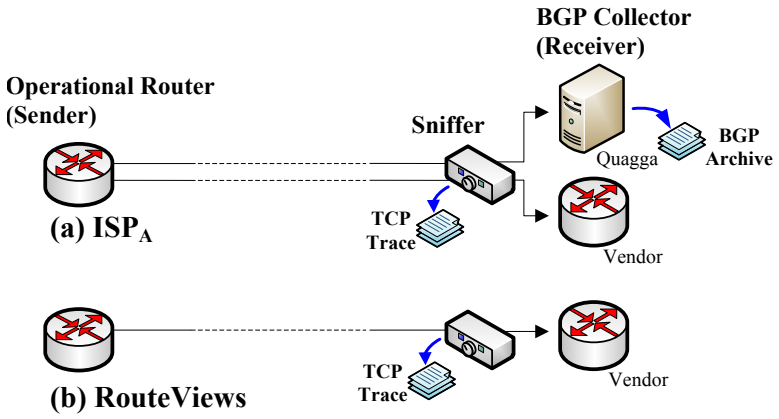


Figure 4.2: BGP/TCP data collection.

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 the following description, we refer an operational router as *Sender*, the BGP collector as *Receiver*, and the TCP sniffer simply as *Sniffer*. The recorded TCP connection contains both the Sender-to-Receiver *data packets* and Receiver-to-Sender *ACK* packets.

Table 4.1 summarizes the basic information 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,[2] we then turn to the BGP archive and apply MCT [52] to identify the *end* of BGP table

[2]A BGP table transfer starts right after establishing the TCP connection [35].

Table 4.1: Summary of BGP/TCP datasets

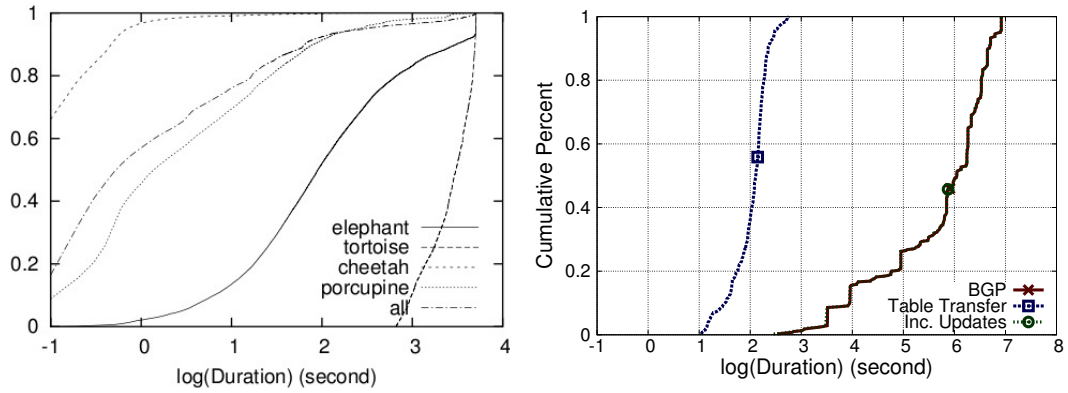
Trace	Duration	Collector	# Pkts/Bytes (M/GB)	# Rtrs	Data `PCAP/MRT`	# Table Trans.
ISP_A-1	2008.05 ∼ 2009.04	Vendor	1023 / 218	24	Yes / -	10471
ISP_A-2	2008.05 ∼ 2009.04	Quagga	909 / 138	27	Yes / Yes	180
	2009.09 ∼ 2010.09		1296 / 219			219
	2010.11 ∼ 2011.01		492 / 81			37
RV	2010.11 ∼ 2011.01	Vendor	176 / 47	59	Yes / -	94

transfer. Different from [52], 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. *(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 and extract BGP messages. This side tool could run either online or offline and takes care of the TCP out-of-order delivery and retransmissions. Then we apply MCT on the extracted BGP messages as in the previous step.

The number of identified BGP table transfers (listed as the last column of Table 4.1) ranges 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.

4.1.2 Flow Level Characteristics

Before we delve into the detail TCP trace, in this section we first analyze, for BGP over TCP connections, the four basic flow (or connection) characteristics,



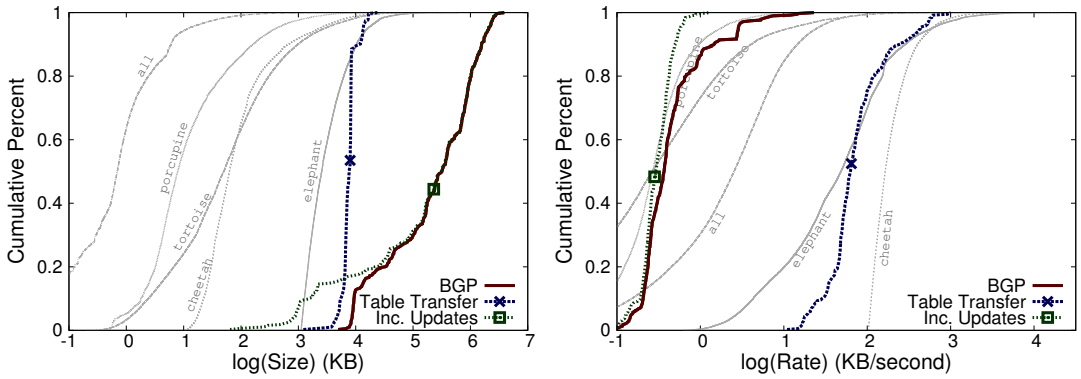
(a) Internet flows [22]

(b) BGP flows

Figure 4.3: Distribution of flow duration

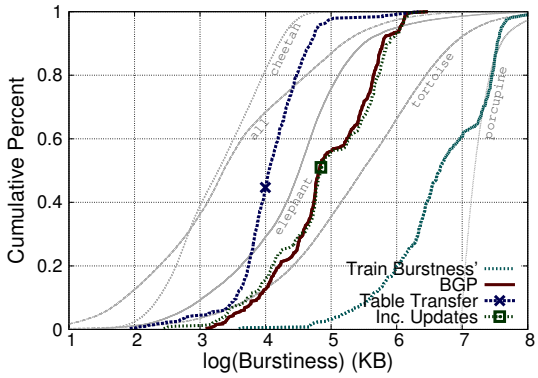
namely *duration*, *size*, *rate*, and *burstiness*. In previous works, Lan et al. and Qian et al. have studied the flow characteristics of long-lived Internet flows [22,33]. Note that BGP generally runs internally in ISPs and was not included in their work. Here, we extend the previous analysis with new results of BGP or TCP connections. We calculate the flow characteristics, and then compare to the four Internet flow types coined in [22]: *tortoise*, *elephant*, *cheetah*, and *porcupine*, corresponding to the *"flows with duration, size, rate and burstiness greater than the mean plus three standard deviation of the respective flow measurement"* [33]. Also, in addition to per TCP connection analysis, we separate a TCP connection into *table transfer* and *incremental update* periods, (i.e., based our identified table transfer period), and measure the flow characteristics respectively.

We first present the distribution of BGP connection duration. Figure 4.3(a) and Figure 4.3(b) show the duration distribution of Internet flows presented in [22] the our result of BGP flows, respectively. As expected, the BGP flow duration is much longer (3 orders of magnitude) than even the longest Internet connections (i.e., tortoise). Note that for Internet flows, there is a cap of flow lifetime at 2 hours due to the record duration in [22], but this contributes to less than 10%



(a) Size

(b) Rate



(c) Burstiness

Figure 4.4: Distribution of size, rate, and burstiness of BGP and Internet flows

of the tortoise flows. Note that the BGP connection duration is by and large determined by the incremental update period. The table transfer period generally finished in tens to hundreds of seconds. Also for BGP distribution, there exist vertical jumps in the distribution curve. These are the cases of collector failures as discussed in Section 3.3.1, in which multiple TCP connections reset at the same time and result in similar flow durations.

In Figure 4.4, we overlay the results of BGP flows on top of Internet flows from [33] [3] for ease of comparison. As shown in Figure 4.4(a), BGP connections

[3]We obtain the data points from the authors and re-plot the figure

send more data than the Internet elephant flows; the difference is in two to three orders of magnitude. Similarly, the incremental update period contributes to more traffic due to long duration. The table transfer period contribute to a stable amount of data, corresponding to the BGP routing table size, which slowly increases in a long term. At the time of this writing, a full BGP routing table contains 370K prefixes, which roughly correspond to 8MB to 10MB worth of data ($\sim$ 95 percentile of both the elephant flows and BGP flows).

For BGP flows, Figure 4.4(b) shows that the transmission rate of table transfer is relatively high, while that of incremental update is low. This is as expected, as during the incremental update period, BGP (and TCP) only sends data upon occasional routing events, and the transmission rate is amortized by the idle periods between events. Note that TCP transmission rate also depends on the underlying link speed, which we could not know and directly compare. But generally we observe that BGP table transfer is slightly slower than the fastest Internet cheetah flows, while the increment update is about the same speed with the porcupine flows.

Last, we measure the burstiness of a BGP flow by multiplying the average update burst rate by the inter-burst time [22]. Figure 4.4(c) shows that the incremental update period is burstier than the table transfer. For the top 40% of BGP incremental update periods, the burstiness distribution aligns with the distribution of the porcupine flows, which are the burstiest Internet flows. On the other hand, the table transfer period, though has a high transmission rate, has pretty low burstiness ratio.

Similar to [22] and [33], we observe correlations between the flow characteristics (shown in Table 4.2). For the table transfer periods, the flow duration and flow rate have the correlation coefficient around -0.74, while the flow size and rate

Table 4.2: Correlation between flow characteristics

	Duration,Rate	Size,Rate	Duration,Size
Table transfer	-0.748	0.782	-0.171
Inc. updates	0.087	0.274	0.982
Qian et al. [33]	-0.69∼-0.60	0.54∼0.57	0.21∼0.40

have the correlation coefficient around 0.78. This reflects the bulk transfer nature of BGP table transfer. On the contrary, The incremental update period does not have such correlations. Instead, it has positive correlation between the flow duration and flow size (with coefficient 0.98). This reflects the another fact that the incremental update period terminates on failures; the flow size is proportional to the flow duration, and rather independent to the transmission rate.

To summarize our observations. We compare the high level flow characteristics of BGP with those of Internet flows. The results illustrate the unique distinction between BGP's two phases: the initial short bulk table transfer and the long low rate update exchange. This also implies that specific TCP tunings for either phase might not work well for BGP over TCP connection. In the following sections, we focus on investigating the table transfer period, with the goal to reveal the potential transport issues and delay times.

4.2 Identify BGP Transport problems

In Figure 4.5 we plot again the distribution of the table transfer duration, but separately for each trace in a finer scale. 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 took longer than 10 minutes. This is generally slower than one would expect:

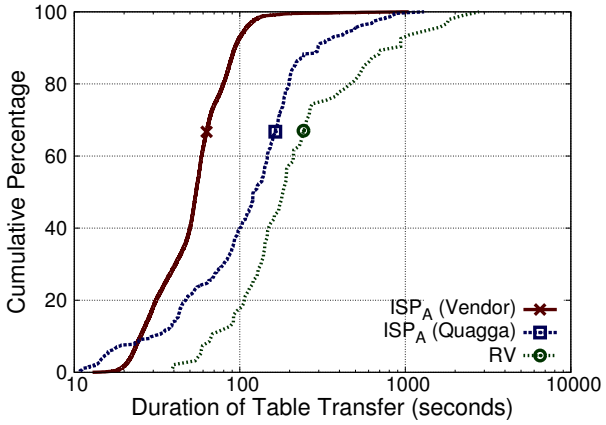


Figure 4.5: CDF of table transfer duration

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. Previous works have made similar observations that table transfers can take even up to tens of minutes [7, 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 4.6 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 three traces). The stretch could be more than an order of magnitude for the distribution tail.

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,

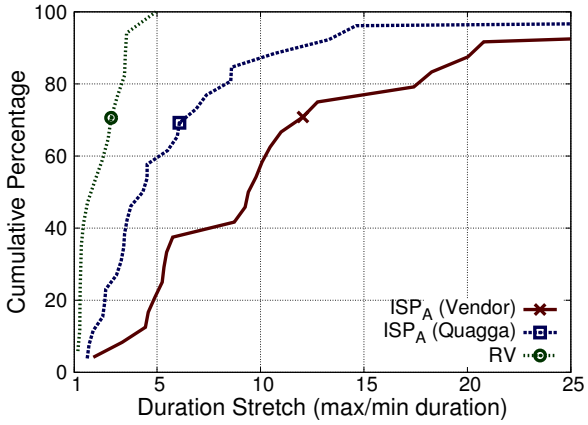


Figure 4.6: Stretch of table transfers

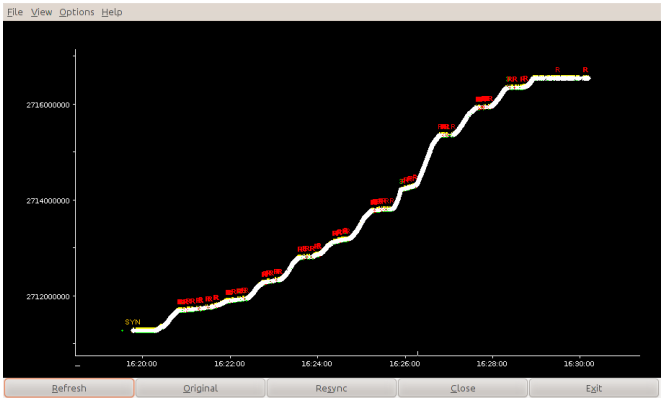


Figure 4.7: Screenshot of `BGPlot`

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. For these slow tables, we develop and use a visualization tool (`BGPlot`) to fast locate interesting TCP events (retransmissions, out-of-order packet delivery, duplicate acknowledgment, etc.). Then we develop corresponding scripts to quantify the occurrences or identify the cause of problem. Figure 4.7 depicts a screenshot of `BGPlot`. In the plot, the tool shows a sample TCP connection with multiple packet retransmissions.

Table 4.3: Observed transport problems

	Observation	Potential Cause
1	Gaps in table transfers	Timer implementation [15]
2	Consecutive retransmission	Bursty BGP dynamics [31]
3	BGP peer-group blocking	BGP scaling feature [53]
4	Misc. issues	Bugs, delay acks, etc

Table 4.3 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 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.

4.2.1 Gaps in Table Transfers

Houidi et al. [15] 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*. Through experimenting in a testbed with routers from 3 vendors, 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. Figure 4.8 shows an example piece of one BGP table transfer that contains the prolonged gaps (i.e., much longer than the RTT) between packet transmissions.

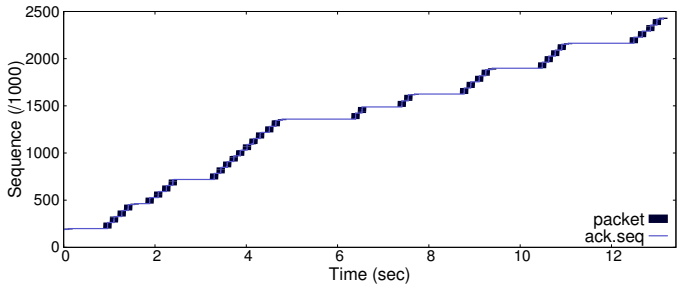


Figure 4.8: Gaps in table transfers

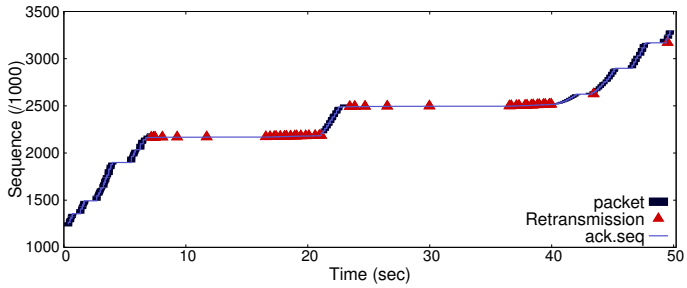


Figure 4.9: Consecutive packet retransmissions

However the distribution of gap length is less regular compared to the testbed results reported in [15]. We observe various gap lengths of tens of milliseconds up to a few seconds. Note that table transfers studied in this work are captured from actual 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.

Table 4.4: Retransmission Deley 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

4.2.2 Consecutive Retransmissions

In our dataset, another common observation is the consecutive packet retransmissions (or packet losses) in a short period of time. Figure 4.9 shows an example of TCP connection that experiences two episodes of consecutive packets retransmissions. Table 4.4 lists BGP updates received during the first loss episode. Note that the router attempted to send all these updates at the same time at 1235728587 (unix timestamp), but due to packet losses and 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*.

4.2.2.1 Sender-side loss (Multiple out-of-order packets)

To find local losses, we first check whether a packet is lost between Sender and Sniffer (i.e., upstream), or Sniffer and Receiver (i.e., downstream), respectively. The idea is based on classifying the packet retransmissions [18]: if a retrans-

mission is due to the loss between Sniffer and Receiver, then Sniffer would first see a packet that is not acknowledged in time by Receiver.[4] 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 4.10 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.

4.2.2.2 Receiver-side loss (Multiple 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 4.11 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, we checked that 27 (15%) and 35 (20%) tables have experienced consecutive upstream and downstream (receiver-local) losses respectively.

The problem of BGP scalability and local losses has long been recognized [9, 31, 53]. In large networks, a BGP router peers with tens and up to hundreds of neighboring routers [8, 31]. 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

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

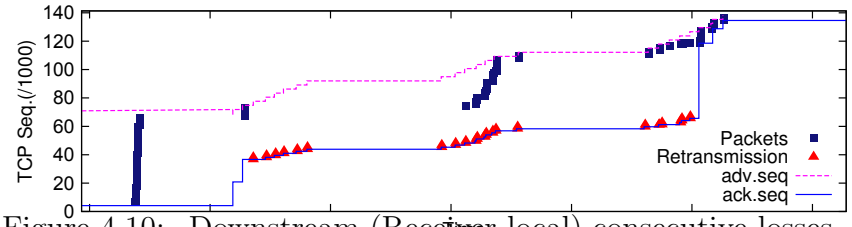


Figure 4.10: Downstream (Receiver-local) consecutive losses

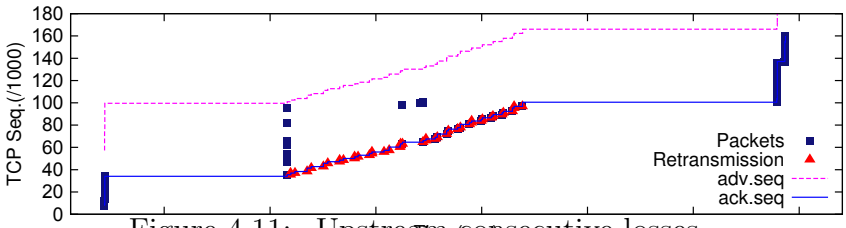


Figure 4.11: Upstream consecutive losses

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 [10]. To alleviate this situation, router vendors suggested to increase interface buffer size based on the number of BGP peers. However, 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 similar phenomenon of *TCP incast congestion* in data centers, when multiple synchronized servers send data to the same receiver concurrently [6, 45].

4.2.3 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 [53]. 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 4.12. At t_1, an error occurred at the Vendor collector, causing the router to keep retransmitting packets, but never being acknowledged[5] 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 [35] 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.

Not that such blocking is a common problem for applications which maintains shared queues for better scalability. As an example, BGPMon [4], a new BGP monitoring service which collects and provides real time BGP data stream to the research and operation community, also recognizes this problem, and propose to pace or even skip the update generation to mitigate the blocking effect.

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

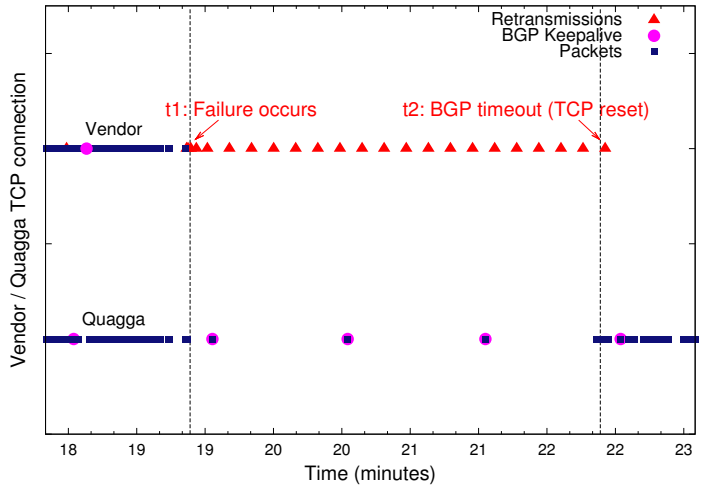


Figure 4.12: Session failures and Peer-Group blocking

4.2.4 Summary

In this section, we observe recurring BGP transport problems that went unnoticed for months or over years. By further investigation, we find that these problems are due to various causes, including bugs, implementation decisions, unexpected effect of optimization features, etc. Interestingly, these problems, to some extent, escaped from the BGP router testing and monitoring, which highlights the need for better BGP monitoring and analysis. In the following sections, we further measure the occurrences of identified problems and quantify their potential negative impact on the BGP transport performance.

4.3 Quantification Results

In this section, we measure *how many* (occurrence) and *how bad* (slowness) are the identified transport problems.

First, based on our understanding of the transport problems, we design various automatic detection algorithms. For example, to capture timer gaps in the table

transfer, we calculate the elapsed delay between the ACK and data packets, and record the gaps if the delay is longer than $1.5 * RTT$; to capture consecutive retransmissions, we maintain a sliding window, and see if more than 8 consecutive packets in the window are retransmissions. We apply the algorithms on the TCP trace, and measure the *occurrence* as the number of problems identified.

Next, measuring the *slowness* is more challenging, In [15], Houidi et al. use TCP throughput model to establish a baseline for expected message delivery rate. However, this does not work well here, as the TCP throughput model is derived from long term TCP periodical oscillation, but in this work we focus on the short term TCP performance, probably concerning only tens of packet exchanges.

In this section, we define and measure the slow times by calculating BGP *epoch* slowness. A BGP epoch is a time period which contains a consecutive train of BGP packets with Maximum Segment Size (MSS), which indicate a burst of continuous BGP update transmission. BGP epochs are then separated by sporadic smaller packets (i.e., idle times).

For each separated epoch, we calculate T_e^+ as the time for the whole BGP epoch gets delivered. We then calculate T_e by removing from T_e^+ the time introduced by transport problems. The value T_e is an estimation of the ideal epoch duration assuming no transport problem in it. Last, we calculate *slowness ratio* as $\frac{T_e^+ - T_e}{T_e}$, which represents the extra delivery caused by transport problems. Note that this proposed slowness ratio serves as a lower bound. Take packet retransmission as an example, we consider the direct retransmission time but not the following shrinkage of TCP congestion window. Our purpose is to demonstrate the evidential impact caused by bad transport behaviors.

Because of the large volume of data, in the following section, we choose three representative routers. These three routers have the same configuration, the only

Table 4.5: Target routers

	geo. distances	avg. RTT(ms)	RTO(ms)	Prefixes
R1	same city	~ 150	$\sim$300	$\sim$ 303k
R2	same continent	~ 200	$\sim$	$\sim$ 303k
R3	different continent	~ 275	$\sim$	$\sim$ 303k

Table 4.6: Occurrence, 2009 March

	R1		R2		R3	
	Vendor	Qua.	Vendor	Qua.	Vendor	Qua.
Gaps in table transfers	245	1	234	27	60	3
Consecutive retransmissions	30	5	40	4	28	8

difference is their distance to data collector. Table 4.5 summarizes the basic information of these three routers. We do not reveal the exact name/number to confirm the usage term of dataset. We use 6 months of data starting from September 2008, after the operator made a TCP configuration change to increase TCP MSS and window size.

4.3.1 Measuring the Occurrence

In this section, we measure the occurrence of identified transport problems. Table 4.6 shows the result of three routers in 2009 March. The Vendor session has more timer gaps and retransmissions than the Quagga session. We check that this is due to the fact that the Vendor session has more session resets.

Figure 4.13 shows the number of session resets of Vendor sessions over the 6 months period. The router R1 always has the most number of session resets and there shows no correlation between the distance of routers and the number of session resets. Figure 4.14 shows the number of consecutive retransmission of Vendor sessions over the 6 months period. We observe that there are at least 20 consecutive retransmissions losses every month, range from 20 to 60 per month. Except 2008 November, during which there are lots of packet losses, we

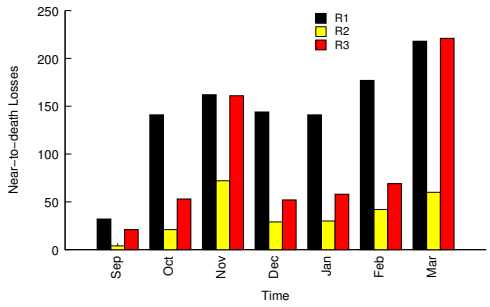


Figure 4.13: Number of session resets

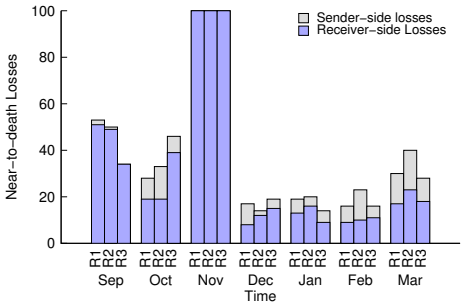


Figure 4.14: Number of near-death losses

Table 4.7: Slowness (R3)

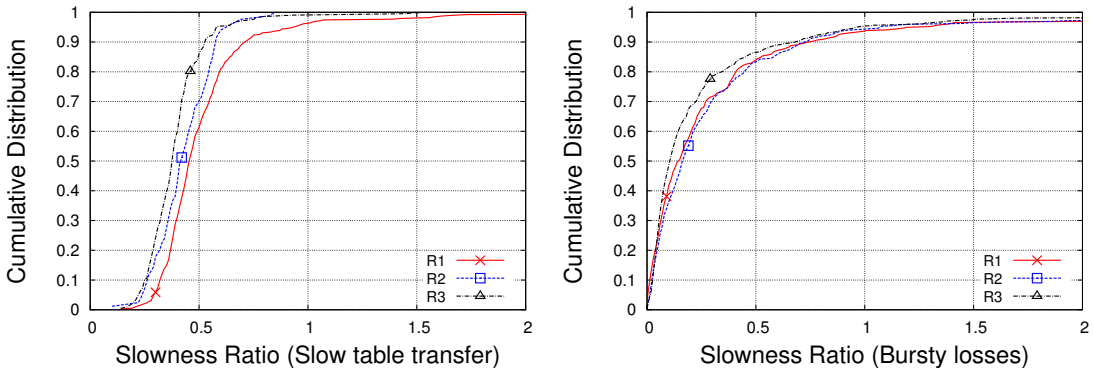
Problem	T_{epoch}(sec)	Inc. Delay(sec)	Slowness Ratio
Gaps in table transfers	39∼54	16.46 (23.44)	+46% (+72%)
Consecutive retransmissions	2.2∼14.8	0.68 (4.06)	+11% (+68%)

are investing the cause for such losses. Moreover, we observe similar number of sender-side and receiver-side losses. But we do not draw conclusion from such distribution, since our result is limited by the fact that the collector is deployed at the receiving end, and thus we can capture *all* receiver-side losses but only *partial* sender-size losses.

4.3.2 Measuring the Slowness

In this section, we measure the BGP epoch slowness of identified transport problems. Table 4.7 shows the results for R3 for illustration. We check that other routers give the similar results.

Figure 4.15(a) and figure 4.15(b) show the slowness ratio for *timer gaps* and *consecutive retransmissions*, respectively. We observe that the timer gaps have more impact on the update delivery, with delay ratio from 40% to 50% in average. while retransmissions delays the packets transfer for 10% to 50% in average. But there are some pathological case for packet retransmissions to slow down a epoch



(a) Timer gaps (b) Consecutive retransmissions

Figure 4.15: Slowness

for up to 400%. Another interesting observation is that the result for three different routers are quite similar. We believe that this is because these routers all have the same configuration and the RTT (in the milli-second level) plays little difference here.

Note that we could not quantify the exact impact of BGP peer-group blocking since we do not have all data from every router in the same group. However, we expect that impact of peer-group blocking shall be proportional to the number of members in the group.

4.4 Discussion

In this section we discuss possible improvements and the lessons learned for the identified transport problems.

4.4.1 Suggested Improvements

In Table 4.8 we list the *fixes* for each individual problem discussed in the previous sections. First, in [15] Houidi et al. describe and evaluate several approaches to mitigate the time gaps in the table transfer, including workarounds such as

Table 4.8: Improving BGP slow transport

Problem	Improvements
Gaps in table transfers	Houidi et al. [15]
Consecutive retransmissions	TCP window validation [14] BGP-aware TCP tuning
Peer Group blocking	Dynamic re-grouping

modifying the router timer implementation, or increasing the TCP window size, etc. We refer interested readers to their work for detail discussion.

Next, in the previous sections, we attribute TCP consecutive retransmissions to a potential collective effect of BGP bursty workload, large number of peering sessions, widely open TCP congestion window, and the mismatch between TCP window size and interface buffer space. Among these factors, Handley et al. [14] propose to decay TCP congestion window when the sender is idle or application-limited for a sufficient long period of time. While this approach applies to general TCP applications, it could be more effective to modify TCP congestion control specifically to BGP protocol behaviors. For example, we can reduce the TCP window size upon receiving *Keepalive* messages, given that a BGP router only send Keepalive messages when there is not pending BGP update and the TCP connection is idle.

Last, for the BGP Peer Group blocking problem. Note that the cause of the blocking is that a peer group is simply configured by administrative export policies, without considering the heterogeneity among BGP sessions. In practice, an administrators could configure a peer group based on the transport performance (i.e. RTT between BGP peers, the underlying physical bandwidth) or the purpose of sessions (i.e. monitoring or operation session, primary or backup). This could prevent the negative impact due to slow group members.

Recently, a router enhancement is introduced to dynamically adjust group members based on their real-time performance. A BGP router keeps monitoring the transmission rate of group members; slow peers would be moved to a new group, or merged back when they catch up back with the fast peers. There are a few open issues. For example, this approach works better when there exist only a few particular slow peers. Otherwise, if all BGP peers behave differently, then many small groups could be created and introduce additional overhead.

4.4.2 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 congestion, 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 dataset, here we do not target at finding *all* possible BGP transport issues. Instead, the problems reported in this chapter, together with those of previous works [15, 50, 54], 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.

To address this problem, in the following chapter we propose a *reactive* approach. That is, instead of searching for a seemingly impossible way to recognize all unexpected transport problems, we focus 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 are only needed when a significant and suspicious delay factor has been reported.

CHAPTER 5

BGP Transport Delay Analysis

In Chapter 4, we discuss BGP transport problems, and the corresponding transport delay observed in the BGP and TCP trace. In this chapter, we further propose a new systematic analysis tool to capture such TCP transport delay and identify reasons behind these delay times. The idea is inspired by TCP rate analysis [42, 55], which classifies the rate limit of TCP connection by factors such as *application*, *TCP end-points*, and *network paths*. Based on a similar taxonomy, the goal is an analysis tool which focuses on the different metric, namely *transfer delay*, and identifies major contributing factors. We call the tool T-DAT (TCP Delay Analysis Tool), named after T-RAT in [55], to make a simple yet clear distinction.

5.1 T-DAT: TCP Delay Analyzer

Figure 5.1 illustrates the high-level T-DAT operations. The delay analyzer first pre-processes the raw packet trace, collects the connection level information, and restructures the trace if necessary (§5.1.2). Then, we transform the packet trace into multiple event series, each is designed to represent one specific type of TCP connection behavior (§5.1.3). Based on the event series, the tool measures the transport-induced delay and classifies the delay factors similar to [55] (§5.1.4).

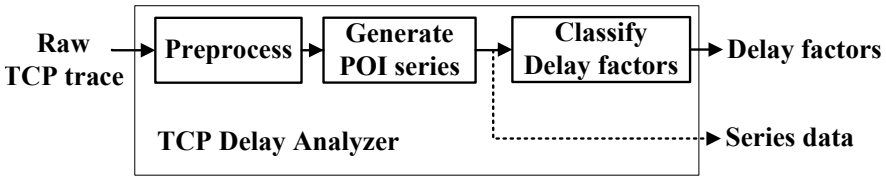


Figure 5.1: High level T-DAT design

5.1.1 Series-based Structure for Delay Analysis

In this section, we first introduce an important time-range based data structure used by the tool: *POI series*. More specifically, the main task of T-DAT is to transform the packet trace into multiple POI series, and analyze the series to infer the reasons behind transfer delay.

From the raw packet trace, consider the arrival of each packet (including data and ACK) as an *event* that potentially affects the behavior of the TCP end-points, such as a packet loss that triggers retransmissions, or an ACK that changes the advertised window size. We call each event as a *Point-of-Interest* (POI) and represent it with the 2-tuple notation $(time\_range, event\_data)$. The time range, 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 event data. POIs generated by the same event are then organized in *an ordered set of time ranges*, i.e., a special set container in which each element is a continuous time period. We name these sets as *POI series*.

One way to visualize the POI series is to present them using binary square curves. Figure 5.2 gives a preview of the graphical output of the tool. Here, the figure includes an example piece of input packet trace and multiple derived output 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

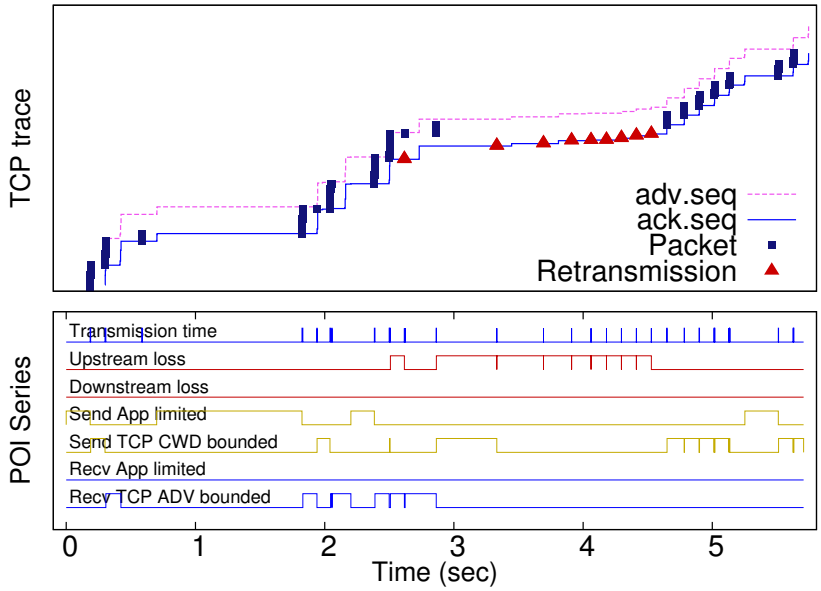


Figure 5.2: Example TCP trace and POI series

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 would describe in detail how to generate these series in Section 5.1.3.

Note that the set-based data structure enables both the high-level quantification and detail inspection. On the one hand, measuring the gross-grain 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 preserve 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.

5.1.2 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,[1] which we extract using `tcptrace` [30]. We also use `tcptrace` to label packets such as retransmissions, out-of-sequence, and duplicates.

5.1.2.1 Accommodate the Sniffer Location

Note that for the BGP monitoring trace used throughout 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 [17, 42, 55]. In previous works, the major concern was how to estimate RTT at different sniffer locations.

Figure 5.3 illustrates the common idea of RTT estimation. When a sniffer is in the middle of the TCP end-to-end path, the sender's true perceived RTT (in the left of Figure 5.3) is inferred as the sum of *d1* and *d2*; each represents one part the route-trip delay for Sniffer-to-Receiver and Sniffer-to-Sender [18]. 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. As shown in Figure 5.3, 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

[1] advertised by the receiver

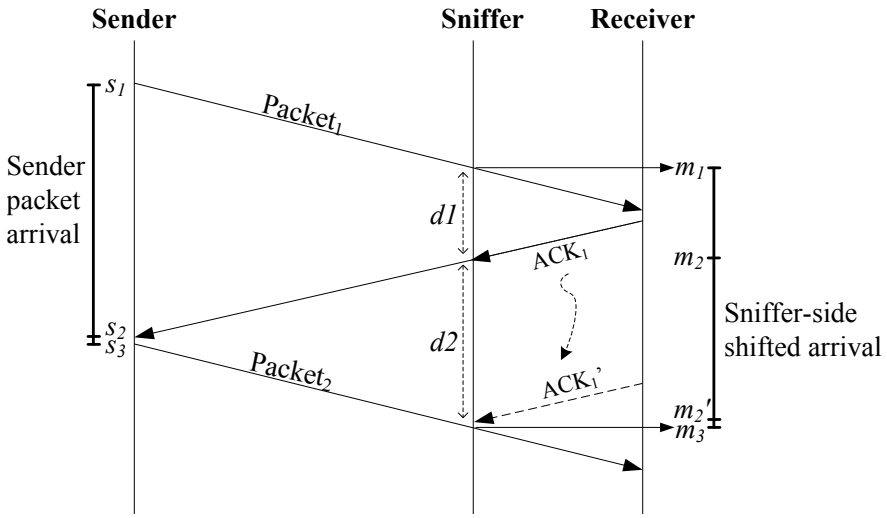
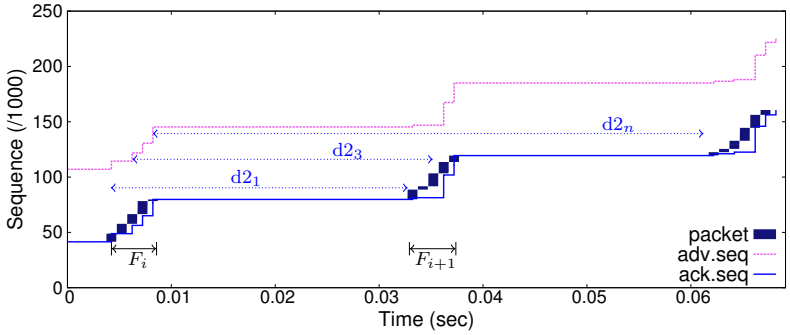


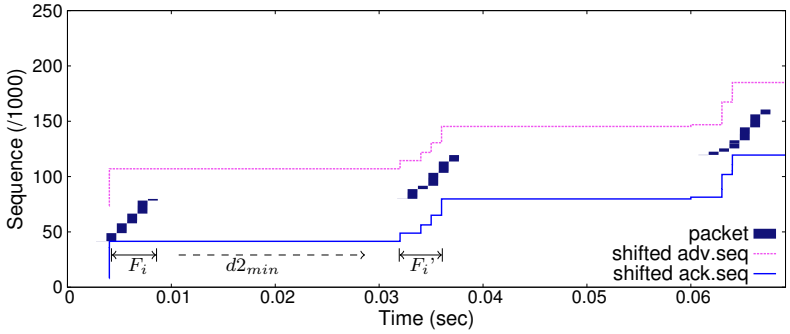
Figure 5.3: Inferring the sender-side packet arrival

be concurrently in transmission, and often there is no clear association between the ACK and the following data packets [18, 25, 42, 46],

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 5.4(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 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 [55]. (ii) For each ACK in the same flight, we then estimate its delay $d2$ and select the minimal, $d2_{min}$, among all



(a) Original receiver-side packet trace



(b) Shifted packet trace

Figure 5.4: TCP trace with the original and shifted ACKs

ACKs in the flight. Note that there exist different studies on measuring $d2$, we implement an algorithm similar to the one described in [17]. (*iii*) Last, the whole ACK flight is shifted with $d2_{min}$ (if it exists). Figure 5.4(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 5.3). 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.

5.1.3 POI Series Generation

From collected packet traces, this section describes three fundamental techniques that we use to generate POI series, namely *extraction*, *interpretation*, and *operation*, expressed with the following abstract rules:

$$(Extraction) \quad series := func(trace) \tag{5.1}$$
$$(Interpretation) \quad series := series \tag{5.2}$$
$$(Operation) \quad series := func(series, \cdots args, \cdots) \tag{5.3}$$
$$series := series \oplus series \cdots \tag{5.4}$$

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.

5.1.3.1 Extraction

First, we generate base series from *objective observations*, i.e., the events that we can directly extract from the packet trace. As *Rule 5.1* 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, the packet transmission and retransmission, and the outstanding packets, etc.

Transmission time. This series tracks the transmission duration, indicating the time that TCP really spends on transmitting data packets. As shown in Figure 5.2, 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 4.2.2. 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 [18]. In Figure 5.2 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.

5.1.3.2 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 4.2.2, if the sniffer is close to the sender side (e.g., neglectable $d2$ in Figure 5.3), then the *UpstreamLoss series* also indicate the local packet losses at the sender. So we construct series.

$$\begin{aligned} SendLocalLoss &:= UpstreamLoss \\ NetworkLoss &:= DownstreamLoss \end{aligned}$$

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.

$$\begin{aligned} NetworkLoss &:= UpstreamLoss \\ RecvLocalLoss &:= DownstreamLoss \end{aligned}$$

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 $d2$) [42]. 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.

5.1.3.3 Operation

In this step, POI 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 5.3*).

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. We generate this series by checking whether the interval between each outstanding period is much longer than RTT [55]. Figure 5.2 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 [55] or subject to the application rate limit [15].

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. 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 [42, 55].

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 $3 \cdot MSS$ [42].

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 5.2, 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 5.4*). 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.

5.1.4 Output: Contributing Delay Factors

Last, out of 34 internal series, we arrive at 8 conclusive series, called delay *factors*, corresponding to the limit factors proposed in [42, 55]: *application limited*, *TCP window limited*, and *network path limited*. We 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). 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.

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

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.

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

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.

5.2 Analysis Results

We applied our tool on the three traces described in Section 4.1.1. In particularly, we analyze the delay in the *initial BGP routing table transfer* [35]. The duration of such a transfer represents the elapsed time spent to boostrap and converge the routing state between neighboring BGP routers. Our object is to demonstrate two different flavors of the tool on (*i*) surveying delay factors and (*ii*) identifying specific known problems.

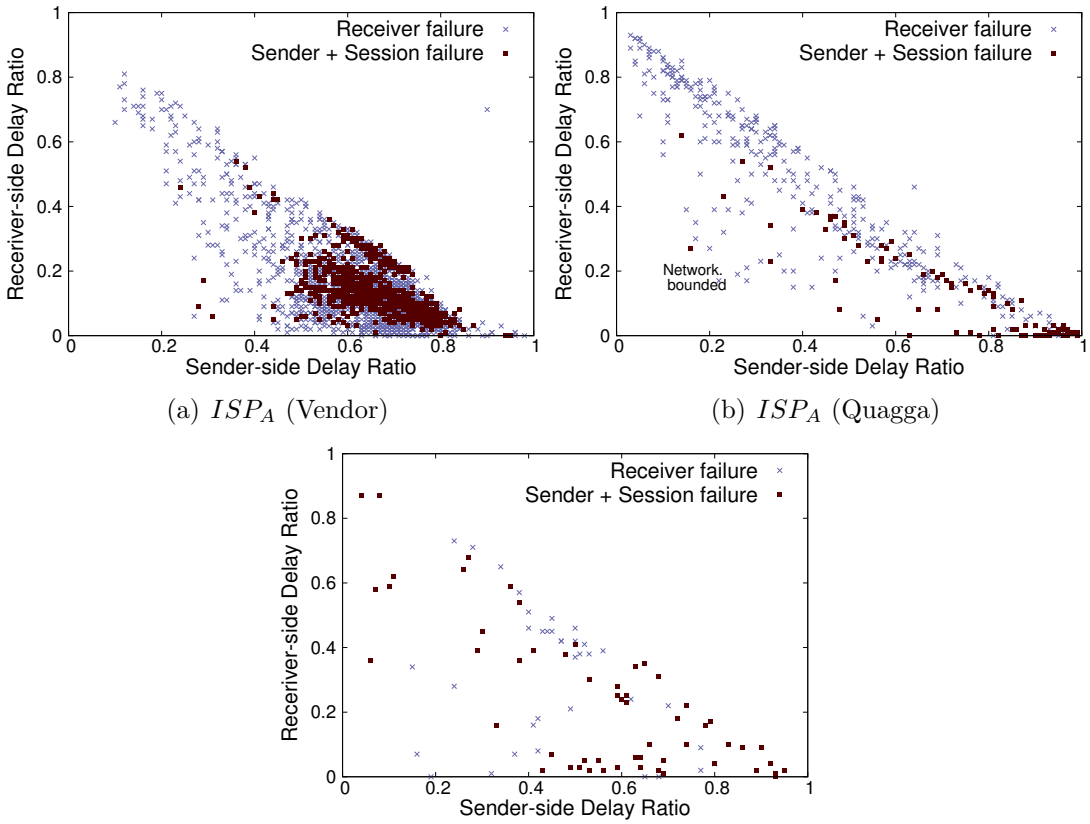
5.2.1 Identifying Major Delay Factors

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 5.5 we depict the scatter plots for the sender (R_s) and receiver (R_r) delay ratios. For the ISP_A (Vendor) trace, we only show the result for the period of March 2009 to May 2009[2]; otherwise the data points would be too crowded. We check other periods and the observation is similar.

Figure 5.5(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

[2]includes 3038 table transfers



(a) ISP_A (Vendor)

(b) ISP_A (Quagga)

(c) RouteViews

Figure 5.5: Sender-side, receiver-side and network delay ratios of table transfers

understand the trend, we further infer whether a table transfer is triggered by a sender or receiver failure using the method in [7], marked as the solid square points. Figure 5.5 (a) and (b) show that the failing 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 the failing side and is likely to become the bottleneck of table transfer performance. In Figure 5.5(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

Table 5.1: 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

vendors and managed by different ISPs all over the Internet. This is subjective to future 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 ration > 0.3). We choose the 0.3 threshold to allow more than one major factors been selected for a table transfer, which is rather common based on our observations. Table 5.1 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 5.1 further shows the breakdown results for in-

dividual 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 62 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 5.6, 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 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 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

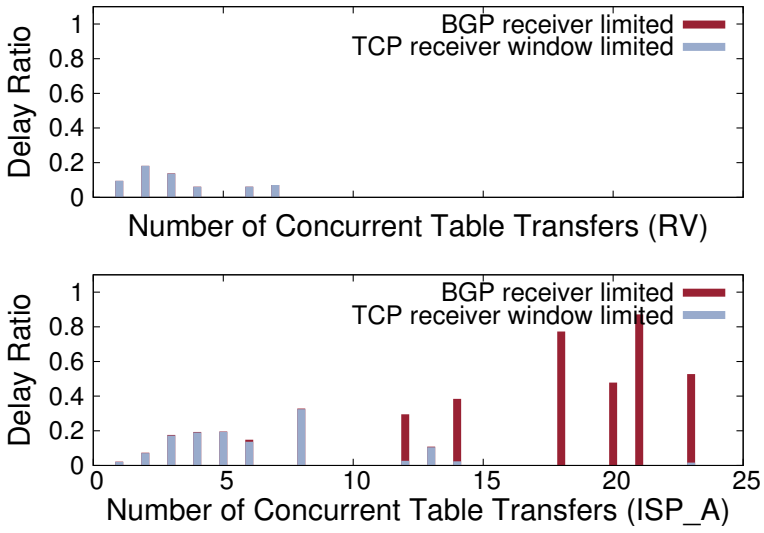
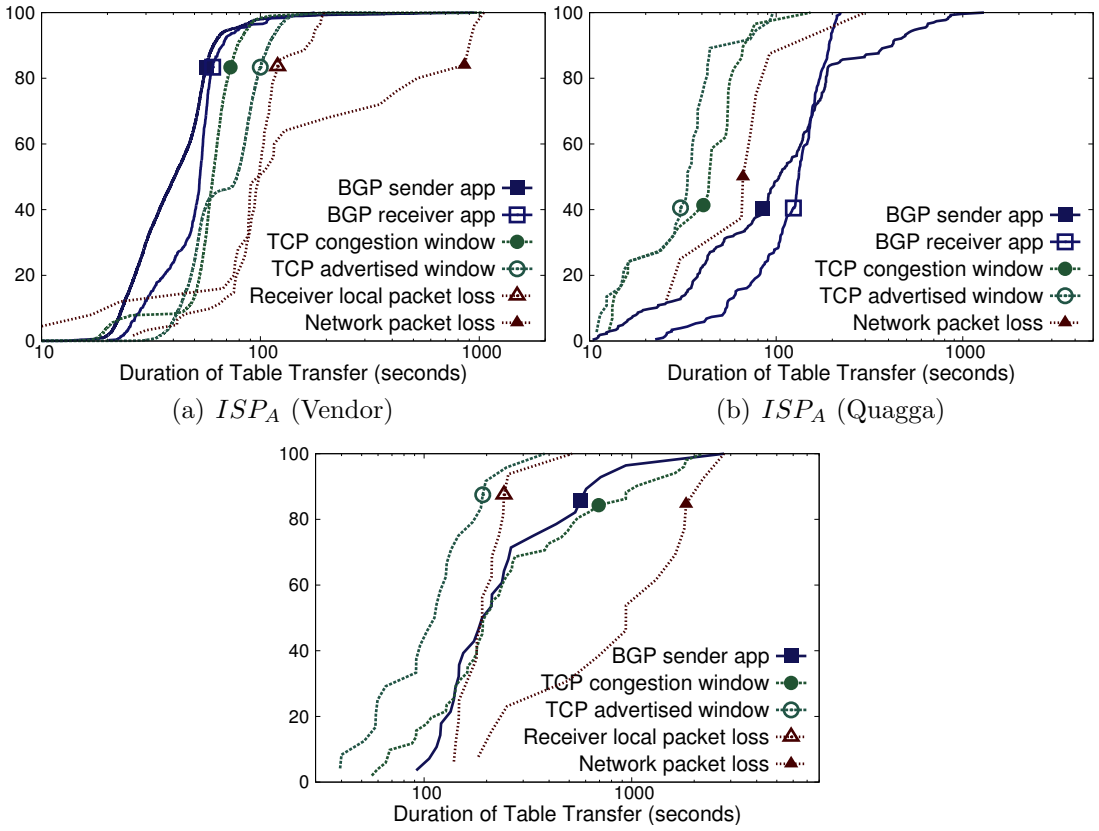


Figure 5.6: Affect of concurrent table transfers

transfer duration in Figure 5.7. 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. One interesting difference lies ISP_A (Vendor), in which the application limited transfers are relatively fast. In this case, we observe that if table transfer are throttled by the sending or receiving applications, the packets are sent with a lower and smooth transmission rate, and are less likely to cause packet drops, which would otherwise result in much longer delay as observed in ISP_A (Vendor) trace.



(a) ISP_A (Vendor)

(b) ISP_A (Quagga)

(c) RouteViews

Figure 5.7: Table transfer duration by delay factors. Y axis is CDF

5.2.2 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 4.2, we develop the following algorithms to identify and quantify them in the 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 5.8 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 [41] to automatically identify the knee point and, thus, BGP timers. 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 5.2, 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 5.2 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

Table 5.2: Identify problems and avg. delay described in Section 4.2.

	ISP_A (Vendor)		ISP_A (Quagga)		RV	
BGP Table Transfers	10336		436		90	
Timer Gaps	857	7.31 (s)	74	16.25 (s)	7	19.40 (s)
Consecutive losses	2092	5.14 (s)	176	4.52 (s)	29	31.15 (s)
BGP peer-group blocking [3]	8	134.53 (s)	8	129.72 (s)	3	94.37 (s)

shown in Table 5.1.

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 5.1 (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 4.2.3. 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. In addition, 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:

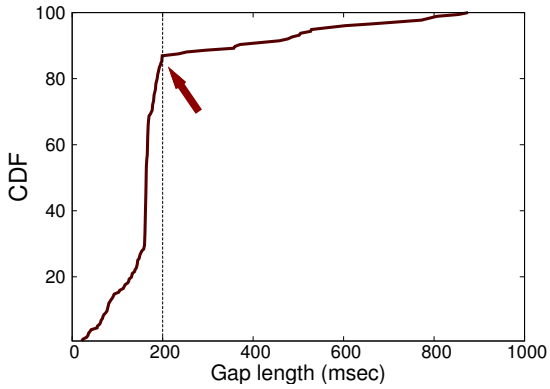
$$Quagga.SendAppLimited \cap Vendor.Loss \quad or$$
$$Vendor.SendAppLimited \cap Quagga.Loss$$

As shown in Table 5.2, 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.

We emphasize that without this delay analyzing tool, it is possible to directly work on the raw packet trace and develop individual ad-hoc technique to capture table transfer problems. However, the tool offers a few unique advantages. First, as demonstrated 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.

5.3 Discussion

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



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

Figure 5.8: Infer BGP timers from the gap distribution

5.3.1 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 [17,25,46] and [18,42,55]. 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. Such an isolation helps prevent the complicated aggregated effect of these two error sources. This allows us to reuse many techniques and parameter settings established in previous TCP inference and rate analysis (described along with the tool in Section 5.1), which we observe also work empirically well in this work. As a future step, we expect to explore the effect of different parameter settings, for both BGP and other application packet traces.

5.3.2 Prospective Usage

In addition, considering the proposed T-DAT as a generic tool, here we briefly discuss a few future applications. First, the tool can potentially help **troubleshoot the protocol implementation**. Note that we generate POI series to represent different states of a TCP connection. Our observation is that series themselves do not always agree with each other. For example, 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 an implementation bug: upon receiving a zero-window ACK, the sender creates a 1-byte probe packet [32]. 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. This tool can help detect such a situation by intersecting the two seemingly conflict series:

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

Second, the series data can serve as the **sanitized input to other analysis studies**. Currently, TCP analysis is mostly conducted on the raw packet trace [17, 33, 42, 55], which can be less effective with respect to their goals. Qian et al. [33] 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. [17] proposed to infer TCP flavors by comparing the number of outstanding packets against the projected congestion window size. The approach is effective when a TCP connection is bounded by the congestion control, which

is, unfortunately, 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.

CHAPTER 6

Instrument the BGP monitoring

In previous chapters, we identified BGP transport problems and analyzed actual transfer delays in a large operation network. However, this works does not intend to cover all aspects of BGP transport behavior. First, given the current distributed and heterogeneous BGP networks, these results show only a limited view of the global routing system. Second, data captured for BGP monitoring sessions, though essential to the research community, could not fully represent the BGP sessions in operation. However, to clarify, this issue is not specific to this work, but rather a common limitation of current BGP studies based on the monitoring data. As an important contribution along the progress of this work, we develop a series of software tools that augment the current BGP monitoring practice, and allow in depth studies of BGP transport behaviors.

6.1 BGP Microscope

We put together a collection of analysis tools, named `BGP Microscope`, to investigate BGP transport behavior and help identify table transfer delays. Figure 6.1 shows the software components and the corresponding input and output data flow. In this section, we briefly discuss each component. Note that all the tools are available at `http://irl.cs.ucla.edu/bgpmicro`.

tcptrace'. The first component is a patched version of tcptrace [30]. The `tcptrace` tool reads the given pcap file(s) and *produce TCP connection infor-*

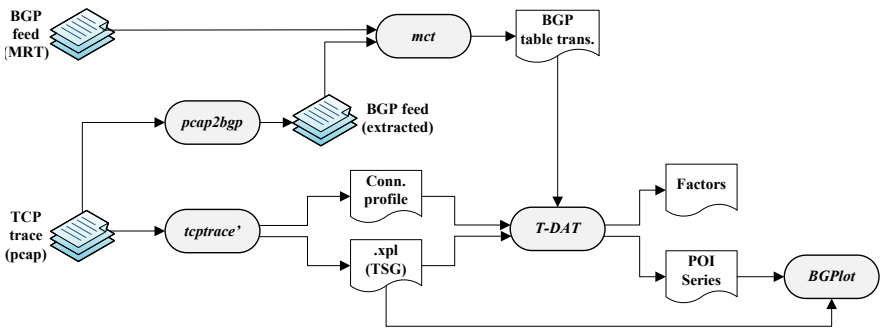


Figure 6.1: BGP Microscope components

mation, including connection life time, TCP segments sent and received, retransmissions, round trip times, window advertisements, throughput, and more [30]. It also produce a number of graphs for further analysis. In this work, we modify the I/O processing of the tool to handle the huge data volume of our dataset.

mct'. the `mct` tool is used to extract the period of BGP table transfers out of the stream of BGP updates message. Please refer to Section 3.1 and [7] for detail descriptions.

pcap2bgp. `pcap2bgp` reads the given pcap file(s), assembles TCP data stream, and extracts the embedded BGP messages. The tool understands TCP sequence numbers and construct data stream regardless of TCP retransmissions, duplication, and out-of-sequence delivery. By default, the tool outputs BGP messages in the plain BGP text format [39]. We also support to save BGP messages in MRT format, which can be then read by common MRT parsers, such as `bgpdump` [3] or `bgpparser` [16]. This is particularly useful if the BGP endpoints are commodity boxes, which are not able to export BGP messages in MRT format.

T-DAT. The TCP delay analyzer is used to classify the delay factors for a TCP connection. This is the main tool we used in Chapter 5. Note that the tool is built around the idea of sets and time-ranges. We implement in Perl language

the time-ranges using integer. We convert the `pcap` second-based timestamps to micro-seconds, and 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*.

BGPlot. The component `BGPlot` is an extended version of `SCNMPlot` [23], which is in turn based on `jPlot`. SCNMPlot extends *"the ability of jPlot by allowing the user to overlay multiple plots in one view and remove color layers from the individual plots"*. In BGPlot, we add new features to help analyze BGP and TCP data traces, including (1) parse and display BGP messages correlated to TCP packets, and (2) synchronized time ranges of multiple plots.

6.2 Deployment

To our knowledge, most ISPs have deployed their internal BGP collectors to monitor the BGP operation. It shall require minimal effort to record BGP and TCP traffic together at the collector boxes and feed into the analysis tools for detail performance analysis. At the time of this writing, we capture the BGP and TCP traffic from a large ISP, RouteViews, and one UCLA core router. The data collecting starts from May 2008, November 2010, and September 2011, respectively. For the large ISP and RouteViews, we mirror the BGP traffic using an inline network switch, while we use `tcpdump` on a PC-based router to capture UCLA BGP traffic. Though we could not make public the saved data due to the usage term of date source, our point is to show that recording TCP data is practical and easy on top of the existing BGP operation. Lastly, note that the proposed BGP Microscope is a tool set. It does not assume or require specific deployment scenarios. However, to fully utilize the potential of the analysis tool, we suggest establishing BGP sessions to the same router from two different monitor boxes,

preferably routers from different vendors. By monitoring two sessions concurrently, this deployment yields a unique advantage in finding transport problems that could only be revealed by cross-checking two update streams.

CHAPTER 7

Related work

This work follows the lead of a few research threads.

7.1 BGP Monitoring Data Quality

The quality of BGP data collected by RouteViews and RIPE is far from perfect due to measurement artifacts and missing data. Specifically, a number of previous works have recognized the need to identify table transfers after monitoring session resets. These table transfers, while contributing a large amount of data, are essentially the monitoring artifacts and do not reflect the actual BGP behavior.

Wang et al. [48] use BGP session state message to identify the start of a BGP session re-establishment but this works only for RIPE.[1] Rexford et al. [36] remove all duplicate BGP announcements from the update stream which is an aggressive way to remove updates due to table transfers, though it also removed real duplicates. Anderson et al. [1] identify table transfers using a rough estimate, it splits the BGP update stream into 30-second bins and discards any bin that contains more than 1000 prefixes. Zhang et al. [51] develop MCT to accurately detect the occurring and duration of table transfers from BGP update messages. All these efforts focus on cleaning up BGP data by removing table transfer updates, rather than quantifying and understanding the table transfer delay, which is the goal of this work.

[1] RouteViews data does not contain session state messages

7.2 Understanding BGP and TCP Interaction

The impact of the transport layer dynamics on the application performance has widely been recognized and studied in the literature, including HTTP [44], video streaming [19], online gaming [5], data centers [6], and file system [24], to name a few. These previous works studied the mismatch between TCP and the application usage patterns, quantify the performance impact, and commonly suggest improvements or tunings for a particular application or TCP protocol. However, until recently, there has been marginal attention to investigate the BGP over TCP behavior.

Feldmann et al. [12] measure BGP pass-through times using controlled environment, and quantify the impact of pass-through delay on the overall convergence time. Their work focuses on exploring the factors that introduce delay within the routers' internal processes. In contrast, this work studies the message delay between two neighbor routers. Xiao et al. [49] model the BGP session survivability under severe TCP congestion, and propose to improve BGP robustness by more aggressive TCP retransmissions. Zhang et al. [54] demonstrate a scenario of low-rate DoS attack to defeat TCP retransmissions and trigger BGP session resets. Houidi et al. [15] examine the BGP slow table transfer caused by suspicious gaps and find a potential cause to be the timer-driven router implementations. In this work, we investigate 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 additional transport problems, which motivate our design of a new analysis scheme.

There are a few additional works addressing the BGP-TCP interaction from different perspectives. Kong et al. [20] verifies the consistency between BGP and captured TCP data and reports the deficiency in BGP data collection tool.

Instead of studying the TCP behavior, Fang et al. [11] propose to fundamentally replace TCP with SCTP as the transport service to enhance the BGP session stability and efficiency.

7.3 TCP Behavior Analysis

There have been several studies on analyzing the rate limiting factors of a TCP connection. Zhang et al. [55] 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. [42] 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 concerns the routing message delay. Also, previous works focus on analyzing limiting factors for individual TCP connection. As we show in Section 4.2.3 that there could exist intervention among BGP connections, this work offers to represent TCP connections in unified time series, which enables efficient analysis across multiple connections.

One similar work is TCP Critical Path Analysis (CPA) [2]. 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 to know in advance the sender's TCP implementation and initial parameters to simulate accurately the change to TCP windows. Only TCP Reno is illustrated and supported in [2]. 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).

CHAPTER 8

Conclusion

Owning to the distributed nature of the global BGP network and limitations of existing BGP monitoring practice, understanding the BGP transfer delay, specifically caused by the interaction between BGP and TCP, presents a considerable operational and research challenge.

In this dissertation, we first report a systematic assessment on the BGP monitoring session failures and table transfer delay of RouteViews and RIPE data collectors over eight years. The results indicate the prevalence of BGP sessions failures, averaging a few resets per monitor per month. How to make BGP sessions more robust remains an open issue both for BGP monitoring projects and ISPs. More importantly, the main point is to show that BGP table transfers (i.e., massive update delivery) could be surprisingly slow. The observations are made possible using the inference technique to identify table transfers from the BGP update stream.

Further to understand the application level slow times, we collect and investigate TCP traces in a large ISP and RouteViews. We successfully locate recurring transport problems which result in prolonged BGP update delivery delay. Such transport problems are due to various reasons, including router implementation bugs, optimization features, and the interaction between BGP and TCP, to list a few. In general, these problems would slow down the BGP update delivery by 10% to 70%. Note that without the evidential TCP trace, these one-hop transport delays could be overlooked and easily attributed to otherwise the system-wise BGP

slow convergence. Unfortunately, transport problems went unnoticed due to the limitation of current *application-level-only* BGP monitoring settings, which highlights the need for a better monitoring practice to detect and analyze the TCP transport delay.

Thus, stemmed from the venue of TCP rate analysis, we propose a new delay-centric tool, T-DAT, to characterize various factors behind the transport delay, which we believe is a significant step toward diagnose and improve the overall BGP transport performance. We demonstrate the tool usage in identifying principal delay factors as well as investigating specific problems in the BGP dataset. Also, as T-DAT itself is designed as BGP agnostic, it can potentially analyze other delay sensitive TCP applications.

We emphasize that, given the global scale and heterogeneous nature of the current BGP network, this work may not answer all the questions of BGP transport delay in the wild. As a major contribution, we propose an analysis tool set, which enables systematic collection and analysis of the BGP transport behavior. The tool uses TCP packet traces, which can be collected passively and requires no modification to the BGP operation. This offers a future opportunity for both the operation and research community to better understand BGP transport behaviors.

References

[1] David G. Andersen, Nick Feamster, Steve Bauer, and Hari Balakrishnan. Topology inference from bgp routing dynamics. In *Proc. of ACM SIGCOMM Workshop on Internet Measurment (IMW)*, 2002.

[2] Paul Barford and Mark Crovella. Critical path analysis of TCP transactions. In *Proc. of ACM SIGCOMM*, 2000.

[3] bgpdump. `http://www.ris.ripe.net/source/`.

[4] BGPMon. BGP Monitoring System. `http://bgpmon.netsec.colostate.edu/`.

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

[6] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. In *WREN '09: Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 73–82, New York, NY, USA, 2009. ACM.

[7] Pei-chun Cheng, Xin Zhao, Beichuan Zhang, and Lixia Zhang. Longitudinal study of BGP monitor session failures. *SIGCOMM Comput. Commun. Rev.*, 40, 2010.

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

[9] Troubleshooting High CPU Caused by the BGP Scanner or BGP Router Process. [online] `http://www.cisco.com/application/pdf/paws/107615/highcpu-bgp.pdf`.

[10] Understanding Selective Packet Discard (SPD). [online] `http://www.cisco.com/image/gif/paws/29920/spd.pdf`.

[11] Kevin Fang Fan and Feng Cai. BGP-4 message transport over SCTP. [online] `https://datatracker.ietf.org/drafts/draft-zhiyfang-fecai-bgp-over-sctp/`, 2008.

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

[13] Lixin Gao. On inferring autonomous system relationships in the Internet. *ACM/IEEE Transactions on Networking*, 9(6):733–745, 2001.

[14] M. Handley, J. Padhye, and S. Floyd. TCP Congestion Window Validation. RFC 2861 (Experimental), June 2000.

[15] Zied Ben Houidi, Mickael Meulle, and Renata Teixeira. Understanding Slow BGP Routing Table Transfers. In *Proc. of Internet Measurement Conference (IMC)*, 2009.

[16] IRL. BGP parser. `http://irl.cs.ucla.edu/software/bgpparser.html`.

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

[18] Sharad Jaiswal, Gianluca Iannaccone, Christophe Diot, Jim Kurose, and Don Towsley. Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. *IEEE/ACM Transaction on Networking*, 2007.

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

[20] Hongwei Kong. The Consistency Verification of Zebra BGP Data Collection. *Technical Report, Agilent Labs*, 2003.

[21] Mohit Lad, Dan Massey, Dan Pei, Yiguo Wu, Beichuan Zhang, and Lixia Zhang. PHAS: A Prefix Hijack Alert System. In *USENIX Security Symposium*, July 2006.

[22] Kun-chan Lan and John Heidemann. A measurement study of correlations of Internet flow characteristics. *Comput. Netw.*, 50(1):46–62, 2006.

[23] LBL. SCNMPlot. `http://www-didc.lbl.gov/SCNM/SCNMPlot.html`.

[24] Sang Seok Lim and Kyu Ho Park. TPF: TCP Plugged File System for Efficient Data Delivery over TCP. *IEEE Transaction on Computers*, 2007.

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

[26] Z.M. Mao, R. Bush, T.G. Griffin, and M. Roughan. BGP beacons. In *Proc. of Internet Measurement Conference (IMC)*, 2003.

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

[28] RIPE NCC. Routing Information Service.
http://www.ris.ripe.net/.

[29] Ricardo Oliveira, Beichuan Zhang, Dan Pei, Rafit Izhak-Ratzin, and Lixia Zhang. Quantifying path exploration in the Internet. In *ACM SIGCOMM Internet Measurement Conference (IMC)*, 2006.

[30] Shawn Ostermann. tcptrace.
http://www.tcptrace.org/.

[31] Kedar Poduri, Cengiz Alaettinoglu, and Van Jacobson. BSTBGP Scalable Transport. http://www.nanog.org/meetings/nanog27/presentations/van.pdf, 2003.

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

[33] Feng Qian, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, Oliver Spatscheck, and Walter Willinger. TCP revisited: a fresh look at TCP in the wild. In *Proc. of Internet Measurement Conference (IMC)*, 2009.

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

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

[36] Jennifer Rexford, Jia Wang, Zhen Xiao, and Yin Zhang. BGP routing stability of popular destinations. In *Proc. of ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2002.

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

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

[39] The RouteViews project - Data format.
http://www.routeviews.org/data.html.

[40] The RouteViews project - BGP Data Archives.
http://www.routeviews.org/update.html.

[41] Stan Salvador and Philip Chan. Determining the Number of Clusters/Segments in Hierarchical Clustering/ Segmentation Algorithms. In *Proc. of IEEE International Conference on Tools with Artificial Intelligence*, 2004.

[42] 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.

[43] Georgos Siganos, Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. Power-Laws and the AS-level Internet Topology. In *ACM/IEEE Transactions on Networking*, August 2003.

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

[45] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Brian Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *Proc. of ACM SIGCOMM*, 2009.

[46] 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.

[47] Lan Wang, M. Saranu, J.M. Gottlieb, and Dan Pei. Understanding BGP Session Failures in a Large ISP. In *Proc. of IEEE INFOCOM*, 2007.

[48] Lan Wang, Xiaoliang Zhao, Dan Pei, Randy Bush, Daniel Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang. Observation and analysis of BGP behavior under stress. In *Proc. of ACM SIGCOMM workshop on Internet measurment (IMW)*, 2002.

[49] Li Xiao, Guanghui He, and Klara Nahrstedt. BGP session lifetime modeling in congested networks. *Computer Networks*, 50(17):3315–3333, 2006.

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

[51] Beichuan Zhang, Vamsi Kambhampati, Mohit Lad, Daniel Massey, and Lixia Zhang. Identifying BGP routing table transfers. In *Proc. of ACM SIGCOMM workshop on Mining network data (MineNet)*, 2005.

[52] Beichuan Zhang, Vamsi Kambhampati, Mohit Lad, Daniel Massey, and Lixia Zhang. Identifying BGP routing table transfers. In *Proc. of ACM SIGCOMM workshop on Mining network data*, 2005.

[53] Randy Zhang and Micah Bartell. *BGP Design and Implementation*, chapter Tuning BGP Performance, pages 74–81. Cisco Press, 1999.

[54] Y. Zhang, Z.M. Mao, and J. Wang. Low-rate TCP-targeted dos attack disrupts internet routing. In *Proc. of Annual Network & Distributed System Security Symposium*, 2007.

[55] Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. On the characteristics and origins of Internet flow rates. In *Proc. of ACM SIGCOMM*, 2002.