Computer Networks 55 (2011) 636–649

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Identifying BGP routing table transfers

Pei-chun Cheng [a,*], Beichuan Zhang [b], Daniel Massey [c], Lixia Zhang [a]

[a] Department of Computer Science, University of California at Los Angeles, CA 90095, United States
[b] Department of Computer Science, University of Arizona, AZ 85721, United States
[c] Department of Computer Science, Colorardo State University, CO 80523, United States

ARTICLE INFO

Article history:
Received 21 February 2010
Received in revised form 19 July 2010
Accepted 7 September 2010
Available online 17 September 2010
Responsible Editor: T. Korkmaz

Keywords:
BGP
Session reset
Routing table transfer

ABSTRACT

BGP routing updates collected by monitoring projects such as RouteViews and RIPE have been a vital source to our understanding of the global routing system. However the collected BGP data contains both the updates generated by actual route changes, and the updates of BGP routing table transfers resulted from BGP session resets between operational routers and the data collection stations. Since the latter is caused by measurement artifact, it is important to accurately separate out the latter from the former. In this paper, we present the design and evaluation of the minimum collection time (MCT) algorithm. Given a BGP update stream, MCT can identify the start and duration of each routing table transfer in the stream with high accuracy. We evaluated MCT performance by using three months of BGP data from *all* RIPE collectors. Our results show that out of the total 1664 BGP resets with 166 monitors, MCT can identify BGP routing table transfers with over 95% accuracy, and pinpoint the exact starting time of the detected table transfers in 83% of such cases. Accurate detection of BGP table transfers enables users to separate out real BGP routing changes and measurement artifacts, and can be used to measure and diagnose the BGP session failures.

1. Introduction

The Border Gateway Protocol (BGP) [2] is the *de facto* inter-domain routing protocol on the Internet. Numerous projects use BGP update data collected by Oregon RouteViews [3] and RIPE RIS [4], the two best known BGP monitoring projects, to monitor Internet routing, diagnose routing problems, and evaluate improvements to BGP. RouteViews and RIPE RIS deploy a number of data collectors around the world. These collectors establish BGP peering sessions with routers in many operational networks. They receive and log BGP routing updates from their operational peers, which we call *monitors*. As shown in Fig. 1, a data collector may co-locate with monitors in the same Internet exchange point and set up single-hop BGP sessions to collect BGP data, or may be far away from the monitors and establish multi-hop BGP sessions to collect the data. In both cases, a data collector is configured to be a passive listener which receives all BGP updates from its peers but does not announce any routing information itself.

Generally speaking, BGP updates can be divided into two categories, table transfer updates and incremental updates. When a BGP peering session is established, a router advertises to the new peer all the routes that are currently in its routing table and match its export policy. We call these updates *table transfer* updates. After the initial table transfer, the router sends out incremental route changes only, which we call *incremental* updates.

In BGP data analysis it is important to be able to distinguish these two types of updates. For example, suppose on a typical day a BGP collector usually records a few tens of thousands of BGP updates, but one day it logs well over a

* Corresponding author. Tel.: +1 3107950801.
E-mail addresses: pccheng@cs.ucla.edu (P.-c. Cheng), bzhang@cs.uarizona.edu (B. Zhang), massey@cs.colostate.edu (D. Massey), lixia@cs.ucla.edu (L. Zhang).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.09.002

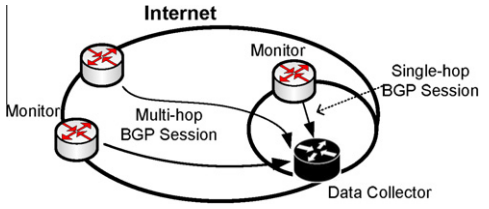


Fig. 1. BGP monitoring.

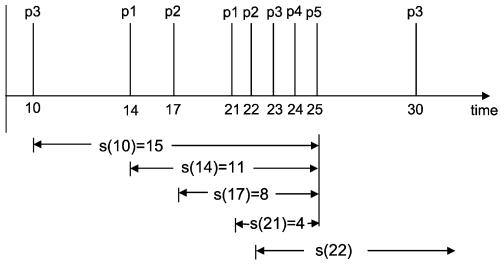


Fig. 2. Update stream and collection time.

few hundreds of thousands of BGP updates. The implication of such a ten-fold increase in daily update counts heavily depends on whether the updates are mainly due to table transfers or incremental changes. At the time of this writing, a typical *full* BGP routing table today contains over 330,000 routes, thus a single BGP session reset between a collector and a monitor results in over 300,000 updates. If the observed ten-fold increase in data logs is due to table transfers, it may suggest the need to improve the stability of peering session with the monitors, because failures of monitoring sessions not only lead to the overhead of session re-establishment on the monitors, but more importantly they can result in missing BGP updates during the session downtime. On the other hand, If the update jump is the result of large spikes of incremental updates, it indicates high dynamics in the global routing system that one needs to pay close attention immediately.

The exact impacts of such data ambiguity on the research results depend on the nature of individual research problem. For example, missed updates during the session downtime may not necessarily affect the results of inferring the Internet topology over a long period of time. However, for analyzing routing dynamics, the results may be highly questionable if the downtime is correlated with routing dynamics. Worse yet, mixing table transfer updates with incremental updates can lead to wrong conclusions. In [5], by observing the large update surges at BGP collectors during worm attacks, Cowie et al. conjectured that worm attacks caused BGP routing instability. However Wang et al. [6] showed later that the observed update surges were mostly due to table transfer updates resulted from the monitoring session resets, and the global routing system did not show significant instability during the worm attack. Had the table transfer updates been identified, the misinterpretation would have been avoided.

Unfortunately, despite the importance to distinguish table transfer and incremental updates, there has been no effective and efficient way to accurately identify table transfers in BGP data. RouteViews data collectors do not log session reset information; RIPE data collectors only have information on when a BGP session reset occurs, but not when a table transfer finishes. Therefore, many research works simply ignore table transfers as [5] did. For those that do attempt to clean up BGP data before use, the methods are developed discretionarily to meet their specific needs [7,8], but may not suite for other projects.

In this paper, we present the minimum collection time (MCT) algorithm that can detect the start and duration of table transfers from a stream of BGP updates with high accuracy. MCT works with all BGP data regardless of whether data collectors log or do not log session reset information. Our goal is to provide a general tool that can accurately differentiate the monitoring artifacts from the operational updates, so that all researchers can make informed decisions regarding monitoring session failures and/or how to treat table transfer updates. Depending on their specific goals, researchers may decide to simply live with the noisy data if the goal is AS-level topology measurement, remove all table transfer updates if the goal is to measure routing dynamics, or even avoid any time period with BGP session failures if the work has stringent requirement on data completeness since there can be missed updates during session downtime.

To evaluate MCT's effectiveness, we utilize the fact that RIPE collectors log session resets and applied MCT to three months of BGP data collected by all RIPE peers, and compared the MCT results against the collector logs. Our results show that MCT can identify routing table transfers with over 95% accuracy, and can pinpoint the exact starts of table transfers in 83% of these cases. Furthermore, MCT can identify the end of table transfers, an important piece of information which has not been available previously. We have made available at `http://bgpreset.cs.arizona.edu/` the MCT source code and the list of detected session resets in RouteViews and RIPE data, so that BGP data users can easily identify table transfer updates and take corresponding measures.

The remainder of this paper is organized as follows. Section 2 presents the basic intuition behind the MCT design and describes the algorithm. Section 3 verifies the detection results against software logs and session state messages, and then applies MCT to RouteViews data. Section 4 discusses possible applications based on the detection results. Section 5 compares MCT with previous approaches, and Section 6 concludes the paper.

2. The minimum collection time algorithm

In this section, we first introduce the intuition behind the basic minimum collection time algorithm, and then addresses a few practical issues in processing real BGP data.

2.1. MCT basic approach

Given a stream of BGP updates, our objective is to detect the existence of all table transfers that may have occurred in the update stream. If a transfer does occur, we want to

identify its *starting time* (i.e., timestamp of the first update of the table transfer) and its *duration* (i.e., how long it takes to finish the transfer). In [6], Wang et al. use session state messages in RIPE data to identify the start of a BGP session re-establishment and thus the beginning of a table transfer, but this method does not tell the duration of a table transfer. Moreover, RouteViews data do not even include BGP state messages. The MCT algorithm aims to identify session resets using only BGP update messages, so that it can be applied to both RIPE and RouteViews data.

The main characteristic of a table transfer is that, in the update stream, *all* the prefixes in the routing table appear within a short period of time. Fig. 3 shows the number of prefixes contained in update messages in 30-s bins based on one-months update from a router. One can notice the interesting spikes which indicate the potential occurrence of table transfers. In [8], Andersen et al. propose a heuristic method to remove table transfer updates by discarding update spikes. However, although one can easily identify update spikes, it is difficult to pin down their exact starting and ending times. Besides, it is unclear how to tell spikes that correspond to table transfers versus those caused by routing dynamics. In order to automatically and accurately identify table transfers, we developed the following approach.

Assume a stream of updates is received from an established BGP session as shown in Fig. 2, and also assume that the entire routing table consists of routes to five prefixes only, $p1$, $p2$, $p3$, $p4$, and $p5$. The updates at time 10, 14, and 17 are *incremental updates* announcing a change in the route to prefix $p3$, $p1$, and $p2$, respectively. A table transfer happens at time 21 and ends at time 25, during this period the routes to all five prefixes are announced. The resulting updates are table transfer updates and this table transfer lasts $25 - 21 = 4$ s. The update at time 30 is an incremental update again.

For an update received at time t, we define its *collection time*, $s(t)$, as the time it takes to see *the announcements for all prefixes*. For example, consider the update for $p3$ that arrives at time 10. Starting at time 10, it takes until time 25 for all five (unique) prefixes to be announced, and thus $s(10) = 25 - 10 = 15$. Similarly, $s(14) = 25 - 14 = 11$, $s(17) = 8$, and $s(21) = 4$. For updates arriving later than time 21, there is no time for which all five prefixes have appeared in updates, i.e., $s(t) = \infty$. As updates occur closer to the beginning of the table transfer, $s(t)$ decreases steadily until reaching a minimum value at $s(21) = 4$. Note that the table transfer begins at time $t = 21$ and lasts exactly $s(21) = 4$ s. After this minimum value, $s(t)$ steadily increases.

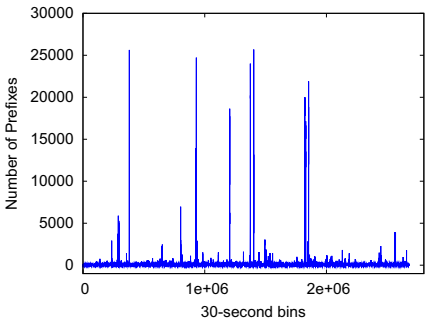


Fig. 3. Number of prefixes in every 30 s.

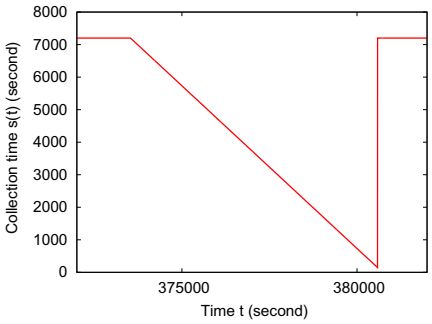


Fig. 4. Sample $s(t) \sim t$.

In general, we expect a trend of decreasing $s(t)$ as the update under consideration approaches the start of a table transfer and increasing $s(t)$ as we move past the start of a table transfer. Calculated from real data, Fig. 4 illustrates the trend of $s(t)$ versus t.[1] Based on this observation, we devise the following basic algorithm to detect table transfers given a stream of updates:

(1) For each update, calculate its collection time, $s(t)$.
(2) Find all local minima of $s(t)$.
(3) Each local minimum is considered a table transfer. Its time t is the starting time of a table transfer, and its collection time $s(t)$ is the duration of the transfer.

If $s(t)$ *monotonically* decreases prior to the beginning of a table transfer and then *monotonically* increases after passing the beginning of a table transfer as shown in Fig. 4, then the basic algorithm works perfectly. In BGP data that we have processed, majority of the time this is the case. However, sometimes the monotonicity does not hold depending on the timing and order of which updates are received. In the following, we introduce some simple tune-ups to adapt the basic algorithm to handle the vagaries of real BGP data.

2.2. Practical tune-ups

The basic minimum collection time approach needs to be adjusted to address a few issues that arise in real BGP data, which is the focus of this section.

2.2.1. Reducing computation load

Incremental updates that are not part of any table transfer can have very long collection times (e.g., on the order of many hours or days). Recall that we are looking for minimum $s(t)$ values and these minimum values correspond to table transfer durations. Computing very large $s(t)$ values requires certain computational cycles but serves

[1] The upper limit of 7200 s on $s(t)$ is explained in the next section.

very little purpose if we are certain that these values cannot be a minimum.

To reduce computation load, we put an upper bound U on the maximum $s(t)$ value. Once $s(t)$ reaches this upper bound, we are certain the update under consideration cannot be part of a table transfer and we set its $s(t)$ to U. In our implementation, we set $U = 7200$ s (i.e., 2 h). That is, for any $s(t) > 7200$ s, we set it to 7200 s. Since we are looking for the minimum of $s(t)$, the value of U does not affect the result, as long as it is larger than any whole table transfer duration. It is very unlikely that any table transfer will last up to 2 h. From our experiments, we observed that over 95% of table transfers were completed in 600 s (i.e., 10 min). Therefore, the 7200 s upper bound seems a reasonable choice. Note that the use of U is simply a computational convenience. If no safe estimate of the maximum table transfer duration can be inferred, we can simply set U to infinity.

2.2.2. Expected table size

In calculating collection times, the basic approach assumes that the set of prefixes to be announced in the table transfer is known in advance. We can collect this set of prefixes by observing updates sent by the router. A typical full-feed router from RIPE or RouteViews announces routes to over 330,000 prefixes at the time of this writing. However this is not a static set. For example, suppose the BGP session between the router $R1$ and $R2$ goes down. During the session downtime, $R2$'s route to prefix p is withdrawn by one of its other neighbors. When the session between $R1$ and $R2$ is re-established, $R2$ will send its current routing table to $R1$, but prefix p will not be part of this transfer. Thus, the table transferred after the session re-establishment may not have exactly the same size as the one before the session breakdown. Therefore we cannot simply expect *all* the prefixes observed so far when calculating the collection time. However, we expect that most of the prefixes will still be present in the table transfer. The number of unique prefixes needed to constitute a full table is a parameter denoted as N. If N is too high, we may miss some table transfers; if N is too low, we may falsely classify a surge of incremental updates as a table transfer.

To derive a reasonable setting of N, we process one year RIPE data in 2008. By utilizing the fact that RIPE collectors log session resets, we calculate the *table transfer ratio* as the routing table size before the reset divided by the table size after the reset. For total 5772 recorded session resets, Fig. 7 shows the cumulative distribution (CDF) of table transfer ratio, We observe that the ratio is very close to 1 for the majority of session resets. Since prefixes might be withdrawn or announced during the session downtime, there are cases in which the ratio is slightly lower or higher than 1. Based on Fig. 7, we choose $N = 0.99$ as the default threshold, which is low enough to include most real session resets, but still high enough to exclude false positives. There are also 10% cases whose ratio is low (i.e., less than 95%). This is due to partial table transfers, where a BGP session goes down again before an on-going table transfer completes. We discuss partial table transfers in Section 2.2.4. Last, note that this algorithm does not require routers to export a *full* BGP routing table. The minimum collection time is calculated with respect to each router's *individual* table size.

2.2.3. Dealing with trend noise

The basic MCT approach assumes that the collection time $s(t)$ decreases *monotonically* prior to the beginning of a table transfer, and then increases *monotonically* after passing the beginning of a table transfer. However, sometimes this monotonicity can be violated. For example, suppose the update stream in Fig. 2 is modified slightly as shown in Fig. 5. We again assume the full routing table consists of prefixes $p1$, $p2$, $p3$, $p4$ and $p5$. The resulting $s(t)$ values are now $s(10) = 14$, $s(14) = 10$, $s(15) = 12$, $s(21) = 6$, $s(22) = \infty$. In this case, $s(t)$ still follows a decreasing trend as we approach the table transfer at time 21, but there is a slight increase between the updates at time 14 and 15. Thus the basic MCT approach will find two local minima, one at time 14, and one at time 21, although only the latter corresponds to a real table transfer.

One may notice that the falsely perceived table transfer at time 14 and the actual table transfer at time 21 have an interesting relation. The falsely perceived table transfer starts at time 14 and ends at time $14 + s(14) = 14 + 10 = 24$ (see Fig. 5). This *conflicts* with the second local minimum, which says a table transfer starts at time 21. In other words, a table transfer is starting at the same time when another table transfer is still in progress. Given two local minima $s(t_1)$ and $s(t_2)$ of the collection times, we say they are *conflicting* if $t_1 + s(t_1) > t_2$. In the event of such a conflict, the shorter transfer time is taken to be the real table transfer. In our example, there is a conflict between $s(14)$ and $s(21)$. Since $s(21) < s(14)$, we discard $s(14)$ and keep $s(21)$ as the real table transfer.

More formally, assume we have computed $s(t)$ for all updates and have found all local minima in the resulting $s(t)$ values. We then check for and resolve conflicts as follows.

(1) Set t_m to the first local minimum.
(2) Check all other local minima. If there is another local minimum at t, which conflicts with the local minimum at t_m, and $s(t) < s(t_m)$, discard this local minimum t_m.
(3) Otherwise, report that a table transfer starts at t_m and lasts $s(t_m)$ seconds.
(4) Set t_m to the next local minimum, repeat step 2, until all local minima have been either reported as table transfers or discarded.

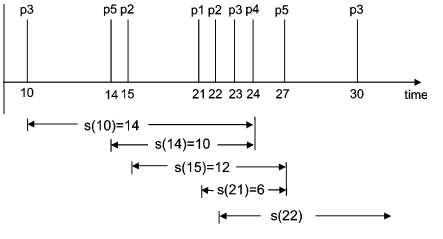


Fig. 5. Trend noise and conflicting transfers.

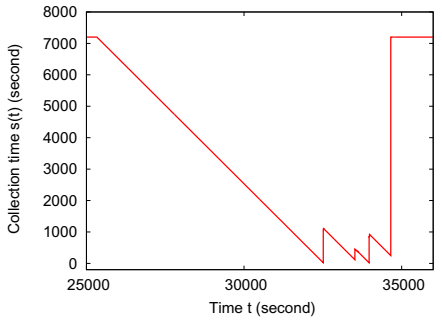


Fig. 6. Multiple table transfers.

2.2.4. Multiple table transfers

In the event that multiple table transfers occur close to each other in time, as long as one table transfer completes before another one starts, MCT can still correctly identify each of them. Fig. 6 shows an example from real BGP data where four table transfers happened within 35 min (from 32,900 to 34,500 s), and they are correctly identified by four local minima in $s(t)$.

A table transfer may not necessarily completes before another one starts. Suppose a BGP session goes down before an on-going table transfer completes. When the session is up again, a new table transfer will start. In this case, we will see two local minima in the collection time, $s(t_1)$ for the partial transfer and $s(t_2)$ for the complete transfer. These two will conflict with each other, $t_1 + s(t_1) > t_2$, since the partial transfer's collection time must extend into the complete transfer in order to include all prefixes. This is the same characteristic of trend noise, and we handle it by the same technique of choosing the one with shorter collection time as a true table transfer. Thus, MCT can correctly identify complete table transfers even in the presence of multiple partial table transfers close in time. In addition, note that for updates within a partial table transfers, their measured collection times would be similar and form a *plateau*. As the result, our tool could also mark these potential partial table transfers together with the detected complete table transfers.

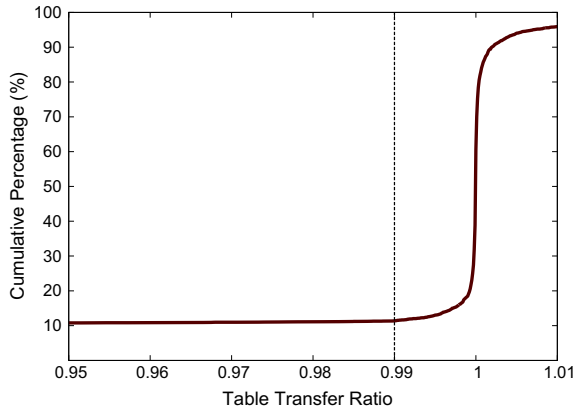


Fig. 7. Table transfer ratio.

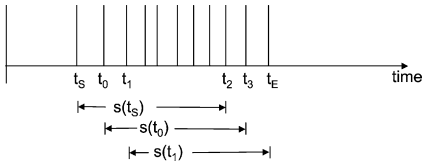


Fig. 8. Bottom searching.

2.2.5. Bottom searching

The basic algorithm assumes that a local minimum $s(t_m)$ (after removing trend noises and partial transfers) corresponds to the start of a full table transfer. As a result of imprecise estimate of expected table size, t_m may not be the exact starting time, and the true starting time could be earlier than t_m. As illustrated in Fig. 8, the table transfer starts at t_S and ends at t_E. But $s(t_S)$ ends earlier at t_2, because we have already seen 99% of the table. A similar early end occurs for $s(t_0)$ and $s(t_1)$. These collection times, $s(t_S)$, $s(t_0)$, and $s(t_1)$, will have similar values, and all appear at the bottom of the valley in $s(t) \sim t$ plot. Depending on the timing of updates, any one of the three can potentially be the local minimum, but only $s(t_S)$ is the true start of the table transfer.

To accommodate this situation, we apply a *bottom searching threshold*, E. After finding the minimum collection time $s(t_m)$ and identifying the potential start t_m and end t_n of table transfer, we look *backward* in time starting from t_m to include more updates into the table transfer until there is a gap of more than E seconds between the next earlier update. Where we stop will be regarded as the true start of the table transfer. Similarly, to find the true end of the table transfer, we look *forward* in time starting from t_n to include more updates into the table transfer until there is a gap of more than E seconds between the next later update, and where we stop will be the true end of the table transfer. From our experience of real data, the true start t_S is usually only a few seconds before t_m when t_S itself is not the detected local minimum. When t_S is the actual detected local minimum, our bottom searching would not find a false start, since right before the table transfer there must be a relatively large time gap with no update in it.

The gaps before table transfers are due to the fact that, in BGP, there exist timers to regulate the consecutive session establishments: ConnectRetryTimer and IdleHoldTime, during which the BGP connections are held in silence to suppress unstable sessions. The default value of these timers are vendor specific, but the suggested value are in the order of tens of seconds. In this work, we conservatively choose E depending on the inter-arrival time of updates within table transfers. From our data, we found that over 99% of updates (11,160,383 cases) in a table transfer are associated with zero inter-arrival time, which means most updates are sent together in the same second.[2] This confirms the bursty nature of table transfers. For the 1% of non-zero inter-arrival times (31,336 cases), Fig. 9 shows

[2] The time granularity of RIPE/RouteViews' BGP data is 1 s. Therefore, zero inter-arrival time means the elapsed time between two BGP messages is less than 1 s.

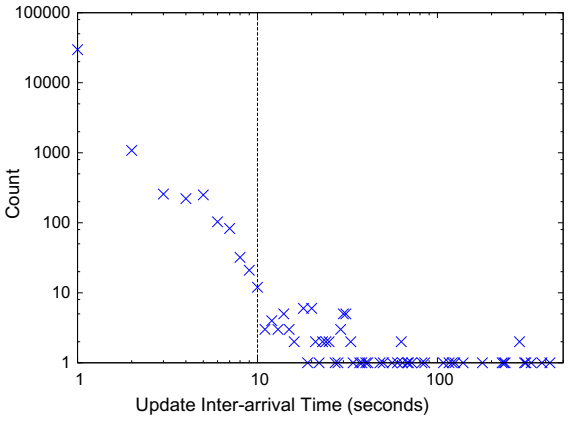


Fig. 9. Non-zero inter-arrival time of table transfer updates.

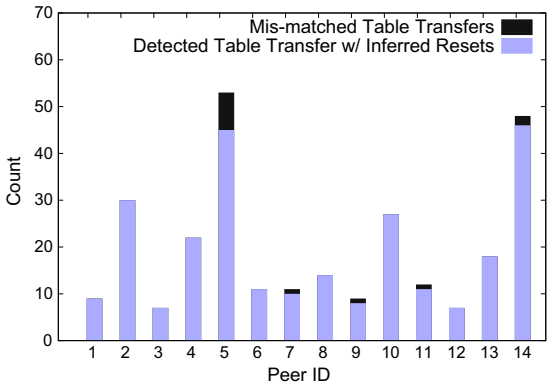


Fig. 10. Table transfers and implicit resets, RRC00, October–December 2008.

the distribution of inter-arrival times. We choose $E = 10$ s, which is effective in our experiments to locate the more precise start and end of a table transfer. Note that the setting of E gap does not affect the accuracy of identifying table transfers, but incrementally fine-tune the estimated transfer start-time and end-time.

2.3. Summary of the algorithm

After applying the tune-ups mentioned above, the final MCT algorithm and can be summarized as follows.

(1) Calculate *collection time* $s(t)$ for all updates. Use $U = 7200$ s as the upper limit of $s(t)$, and use $N = 99\%$ of the last known table size as the expected table size.
(2) Find all *local minima* of $s(t)$.
(3) Resolve *conflicts*, which can be caused by trend noises or incomplete table transfers.
(4) For each local minimum, search for the *true start* and *end* of the table transfer using bottom searching threshold $E = 10$ s.

3. Evaluation

In this section, MCT is verified with two reference data sources provided by RIPE: Quagga software log and BGP state messages. We then apply MCT to RouteViews data. Since RouteViews does not store such additional information, MCT provides the first practical way to accurately identify table transfers in RouteViews data.

3.1. Verification with Quagga software logs

Quagga routing software suite [9] is used by RIPE and RouteViews as the main platform for data collectors. The Quagga process log records the start, termination, and operational status of the collector daemon. Since 2002, RIPE started to archive the Quagga process log, which can be used as one data source to verify MCT detection result.

One challenge is that, Quagga does not explicit record each BGP session reset; rather, it records each individual TCP connection attempt. For example, if the collector establishes a BGP session with a monitor after two failed TCP connection attempts, three TCP connection attempts would be recorded in the Quagga log instead of one BGP session reset. Since exactly one TCP connection can be maintained for a BGP session at any given time [10], any observed TCP connection attempt indicates that there has been a BGP session failure. Thus, we infer session resets by grouping consecutive TCP connection attempts that happened within a short period of time, and compare these inferred session resets with those detected by MCT. We use three recent months of RIPE RRC00 data, from October 2008 to December 2008, which includes update streams from 14 different IPv4 peers.[3]

In this three months RRC00 data, we have 265 cases that the detected table transfers match the inferred session resets. There are 13 cases where MCT detects a table transfer but there is no inferred session resets. Fig. 10 shows the counts over different peers. Overall most detected table transfers (95%) match corresponding inferred session resets. Out of the 14 peers, 9 peers have the perfect match, 3 peers has one mismatch case, 1 peer has 2 mismatch cases, and only 1 peer has 8 mismatch cases. Due to the lack of detail session information in Quagga log, it is infeasible to identify the cause of mismatches. In the following section, we further make use of BGP state messages, which explicitly record the occurrence of session resets, and serve as a more reliable data source for evaluating MCT.

3.2. Verification with session state messages

In addition to regular BGP updates, RIPE also logs BGP session state messages, which record BGP session breakdown and re-establishment. Since a session reset would trigger a table transfer, we use MCT to detect table transfers, and compare the results with session establishment messages. The same three months period, October–December 2008, is used for verification. We calculates the routing table size for each day, and uses 99% of this value as the expected table size for the entire day. RIPE archives the

[3] This study does not include IPv6 peers.

routing tables every 8 h, and RouteViews does it every 2 h. We could re-calculate the expected table size every 8 or 2 h, but it does not make much difference to the final results.

In the three months data, we detect 265 cases that the detected table transfers match the session establishment messages. These confirmed cases are referred as *verified table transfers/resets (V)* in the following sections. Also, there are 13 cases where MCT detects a table transfer but finds no session establishment message, and 40 cases that have a session establishment message but no table transfer is detected, referred as *mismatched table transfers (mT)* and *mismatched session resets (mR)*, respectively.

To quantify MCT's performance, we define *correctness* and *effectiveness* as follows.

- *Correctness:*
  Fraction of detected table transfers that can be matched with a logged session resets, calculated by $\frac{V}{V+mT}$.
- *Effectiveness:*
  Fraction of logged session resets that can be matched with a detected table transfer, calculated by $\frac{V}{V+mR}$.

From the three months of RRC00 data, MCT is able to achieve 95% correctness and 87% of effectiveness from the three months of data. Fig. 11 shows the detection result for each peer. Most mismatches were contributed by two particular peers, which might be caused by peers' specific behaviors or events. In the following sections, we investigate in detail the mismatched table transfers and session resets.

3.2.1. MCT correctness

For detected table transfers without corresponding session establishment messages, they could be due to missing session state messages or "soft reset". First, though session state messages are generated locally by the collector, its logging may not be precise due to the presence of a software bug [9]. In our data, we found evidence of missing session state messages. For example, one session went through two consecutive session establishment processes without a session breakdown message in between. Second, "Soft reset" [11] can be used by router operators to signal changes after router reconfiguration. After a change such as a change in routing policy, the router may re-announce the entire table without resetting the BGP session. Finally, it is possible that the BGP peer may suffer from connection stability issues with all its upstream peers and, after some unfortunate failures, the router may lose and then re-learn (and hence re-announce) almost all of its routing table. In this study, only the last scenario results in the false positives of MCT.

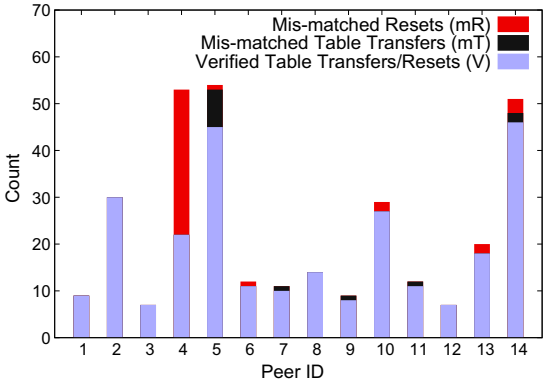


Fig. 11. Table transfers and session resets, RRC00, October–December 2008.

To further verify whether the 13 mismatched table transfers are indeed false positives, two additional metrics are defined. First, "*duplicate ratio*" is defined as the similarity between *detected* table transfers and *known* routing tables. This is based on the fact that a routing table and its derived table transfer updates would not only include the same number of prefixes, but also the route information for each prefix shall be identical. In this evaluation, for a mismatched table transfer, we calculate its duplicate ratio by comparing its route (i.e., AS Path) to each prefix with a logged routing table.[4] In other word, 95% duplicate ratio indicates that the routes to 95% of the prefixes are identical in the detected table transfer and the known routing table.

Moreover, "*sequencing ratio*" is defined as the extent that prefixes are received following the similar order in detected table transfers and verified table transfers. This is based on our observation that, depending on the implementation, BGP routers may send table transfer updates following a particular order, which could be persistent within a short period of time. For example, Quagga organizes routing tables with a prefix-based tree structure and generates table transfer updates by traversing the RIB tree in postorder [9]. In this case, if two table transfers occur close in time, and there are no significant changes to the routing table tree, the updates received during these two table transfers shall follow the similar prefix ordering. Therefore, we can confirm a mismatched table transfer by comparing its prefix ordering with the ordering of a verified table transfer. Note that in real BGP data, we could not always locate two nearby mismatched and verified table transfers; there are always a few minutes to days of time offset in between. As the result, instead of directly comparing the ordering of two entire table transfers, we randomly sample 100 prefix sequences from the mismatched table transfer (200 prefixes per sequence), and check if we could find such sequences in a verified table transfer using the *longest common sequence* algorithm [12]. The sequencing ratio is calculated by the length of common sequences divided by the size of sample sequence (i.e., 200).

Fig. 12 shows the duplicate ratio and average sequencing ratio for each of 13 mismatches. There are 10 cases that the duplicate ratio ranges from 85% to 99% and the sequencing ratio ranges from 45% to 83%. These cases are very likely caused by missing session state messages or "soft reset", since compared with a verified table transfer, these mismatched table transfers not only carry the comparable number of prefixes with same route information, but also follow a similar prefix ordering.

[4] To be consistent with MCT, we use the first known routing table at the beginning of each day.

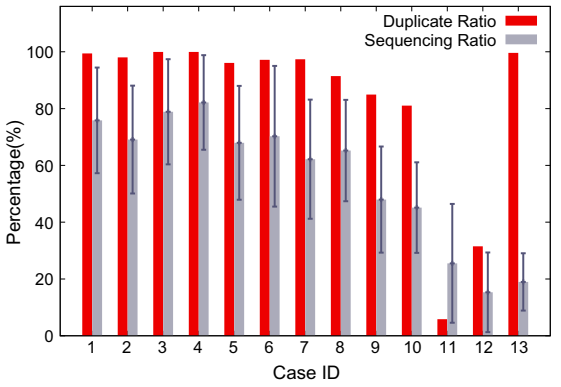


Fig. 12. Mismatched table transfers, RRC00, October–December 2008.

There are 2 cases that have both low duplicate and sequencing ratio, which indicate that the monitored router loses and re-announces almost all prefixes of its routing table with new transient routes.

This is possible when the monitored router solely select one particular peer as the nexthop for all prefixes. Thus, when the session between such peer fails, the monitored router needs to change it routing decision and send a large amount of routing updates. This situation is relatively rare for the RouteViews and RIPE projects, since a monitored router typically has tens of peering routers, and choose multiple peers as the nexthop routers instead of just one. In our evaluation, there exists 2 out of 278 cases (0.7%) belong to this situation. These cases however represent the true false positives of MCT, since for better computational efficiency, MCT detects table transfers by counting the received number of prefixes, but not keeping track of routes to each prefix. In the future work, we are extending this idea to detect these table transfers in the operational networks.

Last, there is one interesting case that has 99% duplicate ratio but only 19% sequencing ratio, in which the table transfer carry exactly the same routes in a routing table but with quite different order. By further investigation, we found that this might be caused by particular implementation of one peer (91.103.24.1), which does not send table transfer updates in persistent order; even two nearby consecutive table transfers reveal two different prefix ordering. We suspect that such peer had employed some specific data structure or RIB traversing technique, such as parallelism, to improve performance, which also introduces randomness in the prefix ordering.

3.2.2. MCT effectiveness

The 40 resets with no detected table transfers could be due to the fact that, not all session resets lead to *complete* table transfers. During persistent failures, BGP session may fail multiple times before it could finish sending the whole routing table. Fig. 13 shows one such example from real BGP data. There are four consecutive session resets followed by 33681, 65, 148, and 107133 announced prefixes. In the first three cases, the table transfer could not complete since the session went down again. Only the last one completed a table transfer, which is correctly identified by MCT. We define the three incomplete table transfers as *partial table transfers*. Moreover, note that in this paper, we conservatively choose 99% of *estimated* table size at the beginning of each day as the threshold to detect table transfers. If the transient table size shrinks significantly within one day, it is still possible that MCT fails to identify such complete but smaller table transfers.

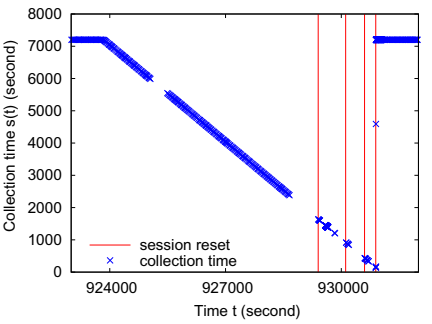


Fig. 13. Multiple incomplete table transfers.

Fig. 14 shows the number of prefixes received for each mismatched reset, calculated by the number of prefixes received following these session resets, compared to the size of known complete routing tables. Most mismatches are due to partial table transfers since the number of prefixes received is less than 80% of a complete routing table. There are two ambiguous cases which cover 95% and 97% of prefixes.

We manually examine these two cases and confirm that they are also caused by partial table transfers. In these two cases, the update message stream stopped suddenly and remained silent for 90 s, which triggered another BGP session reset. This indicates that a transient network failures or congestion happened just before the BGP session can complete 99% of the table transfer, and thus cannot be detected by the MCT algorithm. Fig. 15 shows the session downtime that follows these partial table transfers. (Note that the same ID in these two figures, and other figures in this section does not correspond to the same case.) Most partial table transfers were terminated promptly by 90 s session downtime, which is the default holddown timeout for Juniper routers, implying that these table transfers ended prematurely with a BGP timeout. In addition, some BGP sessions were reset pro-actively by monitors, instead of waiting for BGP timeouts, which might be caused by malformed or corrupted BGP messages [10]. This explains

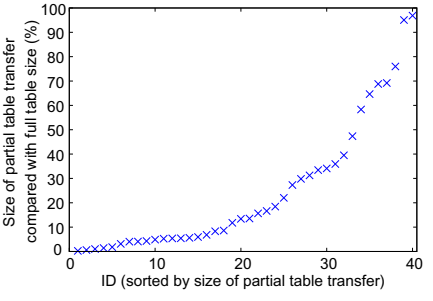


Fig. 14. Partial table transfer size.

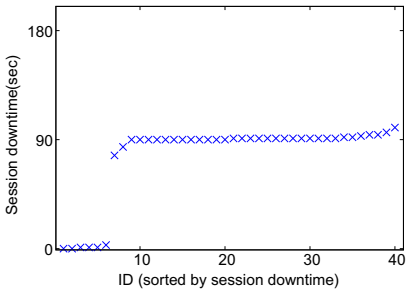


Fig. 15. Session downtime following partial table transfer.

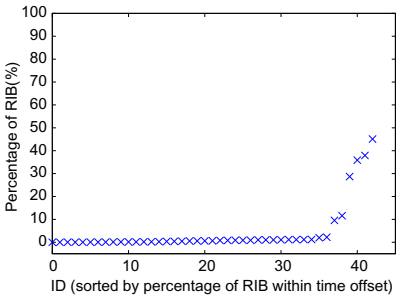


Fig. 17. Percentage of routing table sent within non-zero time offset.

the few cases whose session downtime is extremely short in Fig. 15.

3.2.3. Accuracy of table transfer start time

For the detected start time of table transfer, we quantify the accuracy for the 265 cases that have both resets recorded and table transfer detected. We take the *first routing update message* after session establishment as the real start of a table transfer, as opposed to the session establishment message itself. We then compare this real start with the one our method finds. The difference between these two is called "offset", and is measured in terms of number of seconds and number of updates. Out of the 265 cases, 221 (83%) cases have offsets of *zero* second, i.e., our method finds the exact starting point of the table transfer with no error.

For the 43 cases with non-zero offset, Fig. 16 shows their time offset and Fig. 17 shows the number of prefixes (in the percentage of routing table size) sent within this time offset. For most of the cases the offset is small. For example, in 23 cases, MCT misses the real start by less than 30 s, and in 27 cases we miss the real start by less than 1% of routing table size. Note that these small offsets around or less than 1% are expected. As discussed in the previous section, since MCT calculates the expected table size by 99% of last known routing table size, the true start of table transfer might be 1% offset from the MCT detected table transfer. Depending on the actual trend noise in real BGP data, the bottom search technique may not fully compensate such offsets.

We also found a few cases to have large offsets. The largest time offset is 871 s but it has only small percentage of routing table sent (0.82%) within this offset, which means there are few large gaps between the updates within this period, which might be caused by transient network congestions. There are four extreme cases with very large number of prefixes, 28%, 35%, 37%, 45%, respectively. After careful inspection we found evidence of imprecise session state logs and we suspect that there were in fact two quick session resets and a missing state message for the second reset.

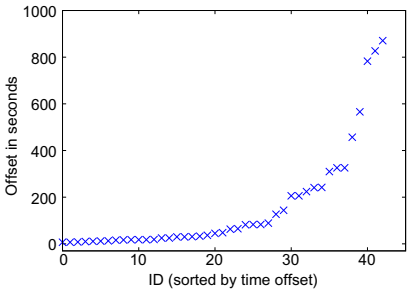


Fig. 16. Non-zero time offset.

3.2.4. Verifying MCT with more collectors

We have verified and investigated the MCT detection results by using RIPE RRC00 data. In this section, we extend the evaluation for all other RIPE collectors. Based on the detail understanding of mismatched cases in the previous section, we filter out cases that are caused by *partial* table transfers and missing session messages to focus on the performance of MCT algorithm.

Table 1 lists the verification result for all RIPE collectors during October–December 2008, except 4 collectors, RRC02, RRC08, RRC09, RRC16, which did not archive BGP data during these three months. Depending on the number of peers, MCT detected different number of table transfers, ranging from tens to hundreds of transfers per collector. The results show that MCT is able to achieve over 95% of correctness and effectiveness for most collectors, while the MCT correctness for RRC07 is around 80%. This is caused by a specific peer (194.68.123.76) which kept announcing most of its prefixes repeatedly in November 20 and November 30. We verified that these announced prefixes are associated with different routes, which indicates that such session was suffering from connectivity problem with its upstream peers and kept lose and relearn its routing table. Overall MCT detected full routing table transfers triggered by session resets with both over 95% correctness and effectiveness.

3.3. Applying to RouteViews data

The RouteViews project has collected valuable BGP data for a number of years, however the routing updates do not contain session state messages. We applied MCT similarity on three months of data starting October–December 2008 collected from the Oregon collector of RouteViews.

There are totally 325 table transfers detected in these three months for 40 peers. For each individual peer, Fig. 18 shows the number of table transfers, and Fig. 19 shows the percentage of table transfer updates in the total

Table 1
Verification result using state messages.

Collector	Type	Location	Number of peers	Verified table transfers	Correctness (%)	Effectiveness (%)
RRC00	Multi-hop	Amsterdam	14	265	99.3	100.0
RRC01	Single-hop	London	30	223	100.0	97.4
RRC02	Single-hop	Paris	–	–	–	–
RRC03	Single-hop	Amsterdam	31	415	99.0	100.0
RRC04	Single-hop	Geneva	3	5	100.0	100.0
RRC05	Single-hop	Vienna	12	84	94.4	93.3
RRC06	Single-hop	Otemachi	1	4	100.0	80.0
RRC07	Single-hop	Stockholm	6	45	80.4	100.0
RRC08	Single-hop	San Jose	–	–	–	–
RRC09	Single-hop	Zurich	–	–	–	–
RRC10	Single-hop	Milan	7	36	100.0	100.0
RRC11	Single-hop	New York	12	32	100.0	100.0
RRC12	Single-hop	Frankfurt	21	310	100.0	99.0
RRC13	Single-hop	Moscow	13	185	95.4	98.9
RRC14	Single-hop	Palo Alto	13	47	100.0	100.0
RRC15	Single-hop	Sao Paulo	3	13	100.0	100.0
RRC16	Single-hop	Miami	–	–	–	–

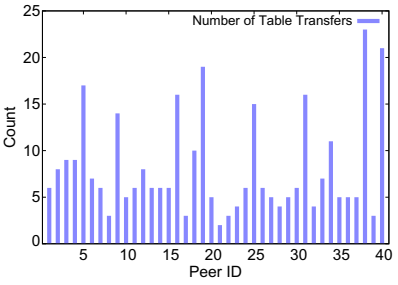


Fig. 18. Number of table transfer.

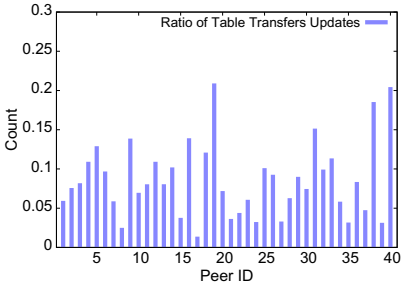


Fig. 19. Ratio of table transfer updates.

number of updates. We observed that the majority of peers had less than 10 table transfers during the three months period, while a few peers had more frequent table transfers than others. In addition, the ratio of table transfer updates ranges from 2% to 20% and the ratio is proportional to the number of table transfers as expected. RouteViews users may need to differentiate table transfers and incremental updates since the number of table transfer updates is significant enough to impact their BGP analysis. Without requiring additional software logs or state messages, MCT provides the first practical way to accurately identify table transfers.

Preceding each detected table transfer, we further measure the session downtime which indicates that how soon a BGP instance could recover a failed session. For example, if there are transient network failures during which no BGP messages were received, it might take 90 s[5] to timeout the failed session [10] and a few seconds to re-establish a BGP session, which may lead to hundreds of seconds of session downtime.

However, since there are no explicit logs on when the BGP session failed, we estimate the session downtime by calculating the elapsed time between the last seen BGP update and the beginning of table transfer.

Fig. 20 shows the cumulative distribution of estimated session downtime. We observed that around 30% of session downtime is shorter than 202 s, while 50% and 90% of session downtime is shorter than 829 and 3432 s, respectively. There are also extreme cases that session downtime is even longer than 2 h, which left significant gaps in the stored BGP data.

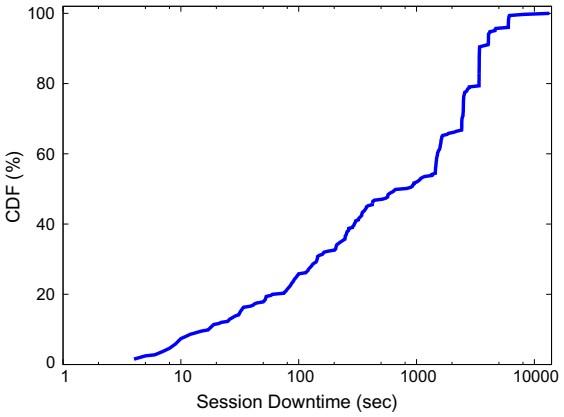


Fig. 20. Session downtime.

[5] Depending on BGP implementation and configuration.

It is important for users of BGP data to be aware of such large session downtime as well as the corresponding table transfers. The collector will miss valuable BGP routing updates during the downtime; and when the session comes back up, there will be a large number of extra updates due to the initial transfer of the entire routing table. If one does not take data deficiency into consideration, the results from analyzing the data may very well be questionable.

4. MCT applications

In this section, we present three applications stemmed from the results of identifying table transfers.

4.1. Clean collected BGP data

BGP routing data collected by RouteViews and RIPE RIS have become an essential asset to both the network research and operation communities. However, the BGP monitoring session failures have introduced noisy artifacts such as missing update messages as well as duplicate updates during session re-establishment, making analysis results derived from such data inaccurate at best. Unfortunately, since there have been no records of these session failures, data users have to sanitize the data according to their individual needs. Our MCT tool helps correctly identify the table transfers, including their occurrence as well as the start and end time of table transfers.

Depending on the specific types of study, users may decide to live with the noisy data, if their works focus more on static BGP properties, such as inferring the Internet topology over a long period of time. On the contrary, users could clean up the table transfers from the update stream. More specifically, the pure duplicates introduced by the table transfer shall be discarded while the withdrawn and newly announced prefixes during the session failures should be restored. By investigating the difference between the routing table before and after the detected session reset, one can infer the missing announcements and withdrawals during the downtime.

Admittedly, such inference could not perfectly restore the actual receiving time or ordering of updates. In addition, as described in Section 2.2.4, there exist cases that a BGP session might experience a severe network problem, and a number of full and partial tables would be observed consecutively in a short period of time. In these cases, precisely cleaning the BGP data becomes more difficult. As RIPE and RouteViews provide routing table snapshots which record the correct routing state periodically. If users are strict on the data deficiency, they are suggested to skip the period of detected noisy data between two snapshots, or may rather use data from other stable monitored peers. Note that such *noise avoidance* is made possible only after MCT first detects the existence of table transfers.

The MCT implementation is available online. In addition, for the public available RouteViews and RIPE BGP data, we further provide our identification results for all historical BGP data. Without additional processing overhead, researchers can utilize our detection results to clean the public BGP data. To our best knowledge, MCT has been used in various works to clean up BGP table transfers [13–17].

4.2. Diagnose monitoring sessions

Since BGP monitoring sessions only passively receive BGP updates from the monitored routers, they tend to have simple configuration and low workload, and are expected to be stable and long lived. In [18], we have conducted a longitudinal study using the RouteViews and RIPE data over eight years. We use the MCT tool to detect table transfers and session resets. The detected table transfers are also verified with the session state messages for the RIPE data. In this section, we briefly summarize our findings of two data collectors, RRC00 and OREG, from RIPE and RouteViews, respectively.

First, frequent session resets are observed across all the monitors and collectors, regardless of the age of the collector, or its location. Fig. 21 shows the cumulative distribution of the number of resets per monitor-month. For both OREG and RRC00, 10–20% monitor-months do not have any reset, while the 50-percentile is 3 resets, and the 90-percentile is 12–15 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 alarming 4205 resets in one month. These cases were likely caused by hardware problems or misconfigurations that made the sessions up and down constantly before they were fixed.

When a monitoring session fails, the observed session downtime usually ranges from one or a few minutes to a few tens of minutes, during which routing updates will not be received from the peer. Fig. 22 shows the cumulative distribution of session downtimes, we observe that the majority of session downtimes are within 10 min, but some cases are much longer. For example, at OREG the session downtime has a 25-percentiles at 1 min, 50-percentiles at 6 min, and 90-percentiles at 48 min. There are cases in which the session downtimes are more than 10 days. While Users can easily spot very long session downtimes (e.g., days) and take precautions accordingly in their data processing, the majority of the session downtimes are within 10 min. 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.

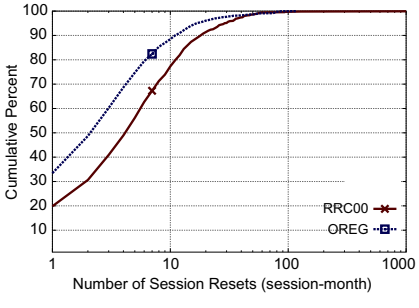


Fig. 21. Number of resets per session-month.

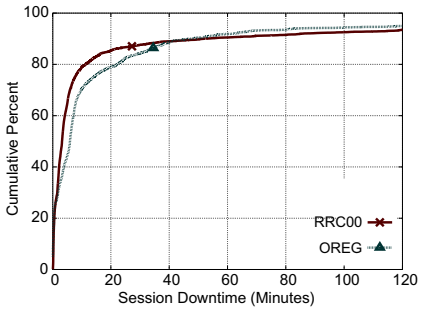


Fig. 22. Session downtime.

Furthermore, we often observed failures that happen to multiple peer sessions on the same collector around the same time, suggesting that the collector's local problems are the cause for the session instability. Verifying with the collector logs, Table 2 shows the number of detected collector-restarts along with the number of session resets triggered by these restarts. We can see that 14% and 37% of session resets are caused by collector-restarts for the RRC00 and OREG collector, respectively. The problem is more pronounced for OREG which has many peers. As collectors' local problems are a major contributor to session failures, it is important for the monitoring projects to improve the stability of the collector, including its network connectivity, software and hardware, in order to reduce monitoring session failures.

Note that in [18], we use MCT specifically to understand the public monitoring session failures. To our knowledge, network operators often archive private BGP data over time, applying the MCT tool to such operational data could help understand and improve the internal routing performance.

4.3. Understand slow table transfers

In the previous section, we briefly describe the characteristics of monitoring session resets, which help reveal that a significant number of session resets are caused by collectors' local problems. In addition, understanding the table transfers are as well important, as transmitting and processing large volume of table transfer updates impose severe stress on the routing operations.

One interesting question is how fast (or slow) are the table transfers, since it has long been speculated by the network operators that table transfers are slow. Fig. 23 shows the cumulative distribution of table transfer duration collected from our longitudinal study [18] over 8 years. For RRC00, over 90% of all table transfers finish within around 5 min, while table transfers at OREG tend to take longer time to finish, with 50-percentile at 4.5 min and 90-percentile at 14 min.

Table 2
Session resets on collector restarts.

Collector	Number of restarts	Number of session resets (%)
RRC00	105	1154 (14%)
OREG	178	6370 (37%)

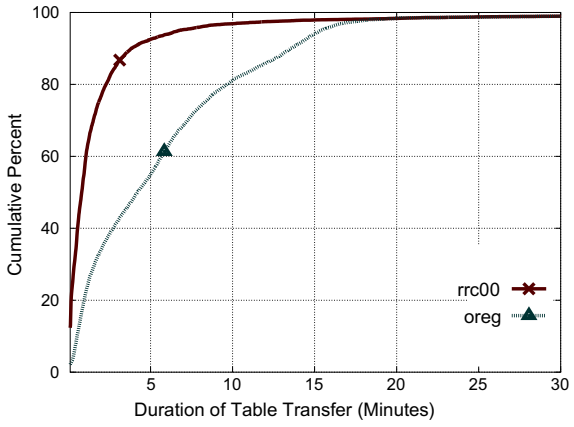


Fig. 23. Table transfer time.

Surprisingly, we found that such prolonged table transfer time is not correlated with the routing table size, and is much slower compared to the link bandwidth, which suggests the existence of other limiting factors in the router implementation. Recently in [14], Houidi et al. has discovered in control experiments that slow table transfers are largely caused by commercial routers' timer-driven processing in sending bulk BGP updates, which particularly introduce many idle gaps within BGP table transfers. More following experiments and studies are necessary to verify that the slow table transfers of actual monitoring sessions are caused by this particular decision of the router implementation.

5. Related work

Prior works that used BGP updates has highlighted the need to clean the BGP data to differentiate table transfers from incremental updates and fall into two classes.

In the first class, rather than identifying table transfers, researchers propose data cleaning methods specific to their works. The method employed in [7] removes all duplicate announcements from the update stream. Based on the fact that a reset of BGP session triggers a complete table transfer and since the session downtime is usually short compared with routing changes, BGP updates sent after session re-establishment should consist primarily of duplicate announcements. However, with our three-month RIPE data set, Fig. 24 shows that duplicate announcements are not solely produced by table transfers. The percentage of duplicate announcements due to other factors is generally small, but could be significant for some peers (e.g. peer 3). Eliminating all duplicate announcements removes both table transfer updates and updates due to other factors, which could be necessary in studying routing dynamics. Although this shortcoming does not affect the result in [7], it limits the applicability of this method as a general approach to deal with table transfers in BGP logs.

The method in [8] makes use of a aforementioned observation that updates due to table transfers occur in bursts. This method splits the update stream into 30-s bins and discards any bin that contains more than 1000

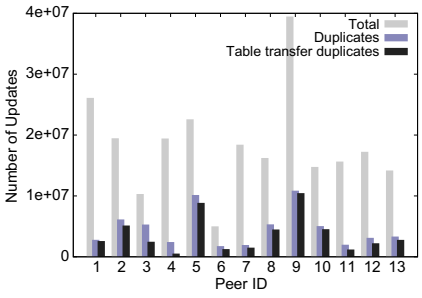


Fig. 24. Duplicate discard.

prefixes, regarding them as part of table transfers. In Fig. 25, we plot the number of valid updates that this method discards, and table transfer updates this method misses using the three-month RIPE data set. It is clear that this method does not miss many table transfer updates, but it falsely discards a significant number of legitimate updates. This could be due the conservative default threshold (1000 prefixes in 30 s). Lower the bin-based threshold may reduce the false positives (valid updates discarded) but inevitably increase the false negatives (invalid updates considered).

One problem of these methods is that they are developed based on specific data cleaning needs, which may not be applicable to other research works. For example in [10], Rexford et al. study the path change for popular prefixes, thus simply removing duplicate updates fits their need. But it could not work for researches which do need the duplicates to investigate their causes [13] or understand the route MED oscillations [19].

The second class of prior works, including this paper, propose to detect explicitly BGP session resets or table transfers. In [20], Wang et al. uses session state messages in BGP logs to identify the start of a session re-establishment and thus the beginning of a table transfer. However, RouteViews logs, which have many years of valuable data, do not contain such state messages. Maennel and Feldmann [21] presents a bin-based heuristic to detect session resets in the Internet, not limited to BGP sessions between the monitor and its peers. Wu et al. [22] presents a heuristic to detect session resets between a network's border routers and their external peers. Their scheme takes into account a majority of routes shifting from one neighbor to another, in a small interval of time as an indication of session reset or restoration. Both these approaches present heuristics for inferring session resets, but do not directly address all the issues arising with session resets between monitoring points and its direct peers.

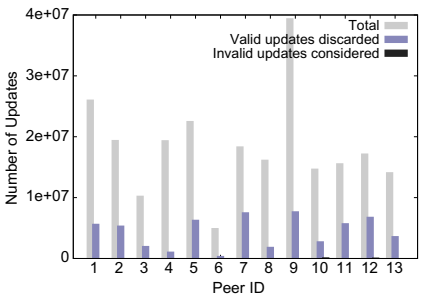


Fig. 25. Bin-based discard.

The work reported in [23] may be considered most relevant to ours. While our tool focuses on accurately identifying table transfers embedded in the update traces, Flayel et al. intend to provide a more general framework, *CleanBGP*, which detects various inconsistency in the collected BGP RIBs and updates, including not only the table transfers, but also update re-ordering, missing updates, etc. However, with respect to detecting table transfers, the authors still resort to the heuristic bin-based approach which we have shown to be imprecise in general. It is possible that CleanBGP incorporates our MCT algorithm for much more accurate detection.

Last, in addition to the research community's efforts on separating out the monitoring artifacts for more reliable research results, the industry has also recognized the negative effect of operational session resets. In [24], Sanli et al. describe a mechanism, *Graceful Restart*, which allows a BGP speaker to preserve forwarding state during session reset, and uses an End-of-RIB marker to indicate the completion of the following table transfer. Note that *Graceful Restart* does not eliminate table transfers: it mainly improves the forwarding plane performance by using the stale (while valid) routing entries. After successful session re-establishments, the routers still require to exchange the whole routing table to replace the preserved state. The End-of-RIB marker, though, really helps the identification of table transfer as it explicitly indicates the end of transfer. Nevertheless, as valuable BGP data without such marker has already been archived over a decade, our tool provides a practical and efficient way to sanitize the existing historical data.

6. Conclusion

In this paper, we presented the design of the minimum collection time (MCT) algorithm that can identify the start and duration of BGP routing table transfers resulted from BGP session resets from a stream of BGP updates with high accuracy. We evaluated MCT performance by applying it to three months of BGP data from all RIPE collectors. Our results show that MCT can identify routing table transfers across heterogeneous monitoring sessions with over 95% accuracy, and in 83% of the cases it can pinpoint the exact start without any error.

MCT efficiently locates table transfers by tracking the collection time for prefixes, without relying on the availability of BGP session logging information. Therefore, it can work with all BGP data, and is particularly useful in processing RouteViews BGP update logs which do not contain session state messages.

Based on the detected table transfers, we are able to estimate and identify failures with long session downtime. These failures negatively impact the quality of BGP logs as they result in missing BGP updates during session downtime as well as superfluous updates during the following table transfer. MCT is the first practical tool for users to filter out these problems before interpreting and drawing conclusions from BGP data obtained from RouteViews or

RIPE collectors, and can also be used to detect and diagnose BGP session failures. The MCT source code is publicly available at http://bgpreset.cs.arizona.edu/ [18]. To our best knowledge, MCT has been used in various works to clean up BGP table transfers [13–17].

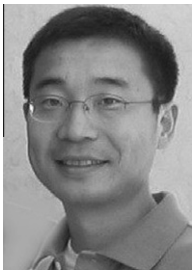
Acknowledgements

A previous version of this manuscript appeared in SIGCOMM'05 Mining the Network Data (Minet) Workshop [1]. This work was partially supported by the US National Science Foundation under Contract No. CNS-0721859, CNS-0721863 and DHS Grant N66001-08-C-2028.

References

[1] B. Zhang, V. Kambhampati, M. Lad, D. Massey, L. Zhang, Identifying BGP routing table transfers, in: MineNet'05: Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data, ACM, New York, NY, USA, 2005, pp. 213–218. <http://doi.acm.org/10.1145/1080173.1080188>.
[2] Y. Rekhter, T. Li, S. Hares, Border Gateway Protocol 4, RFC 4271, Internet Engineering Task Force, 2006.
[3] The RouteViews Project. <http://www.routeviews.org/>.
[4] RIPE Routing Information Service. <http://www.ripe.net/projects/ris/>.
[5] J. Cowie, A. Ogielski, B. Premore, Y. Yuan, Global Routing Instabilities Triggered by Code Red II and Nimda Worm Attacks, Renesys Corporation, Hanover, New Hampshire, USA, 2001. Available from: <http://www.renesys.com/projects/papers/renesys_bgp_instabilities2001.pdf>.
[6] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S.F. Wu, L. Zhang, Observation and analysis of BGP behavior under stress, in: IMW'02: Proceedings of the Second ACM SIGCOMM Workshop on Internet Measurement, ACM, New York, NY, USA, 2002, pp. 183–195. <http://doi.acm.org/10.1145/637201.637231>.
[7] J. Rexford, J. Wang, Z. Xiao, Y. Zhang, BGP routing stability of popular destinations, in: ACM SIGCOMM Internet Measurement Workshop (IMW), 2002.
[8] D. Andersen, N. Feamster, S. Bauer, H. Balakrishnan, Topology inference from BGP routing dynamics, in: ACM SIGCOMM Internet Measurement Workshop (IMW), 2002.
[9] Quagga Software Routing Suite. <http://www.quagga.net/>.
[10] Y. Rekhter, T. Li, S. Hares, A Border Gateway Protocol 4 (BGP-4), RFC 4271 (Draft Standard). <http://www.ietf.org/rfc/rfc4271.txt>, 2006.
[11] Cisco Documentation: Configuring BGP, 2003.
[12] J.W. Hunt, T.G. Szymanski, A fast algorithm for computing longest common subsequences, Commun. ACM 20 (5) (1977) 350–353. <http://doi.acm.org/10.1145/359581.359603>.
[13] J. Park, D. Jen, M. Lad, S. Amante, D. McPherson, L. Zhang, Investigating occurrence of duplicate updates in BGP announcements, in: Passive and Active Measurement, Springer, 2010, pp. 11–20.
[14] Z.B. Houidi, M. Meulle, R. Teixeira, Understanding slow BGP routing table transfers, in: IMC 2009, 2009.
[15] Y. Zhu, J. Rexford, S. Sen, A. Shaikh, Impact of prefix-match changes on IP reachability, in: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, ACM, 2009, pp. 235–241.
[16] R. Oliveira, B. Zhang, D. Pei, L. Zhang, Quantifying path exploration in the internet, IEEE/ACM Trans. Netw. (TON) 17 (2) (2009) 445–458.
[17] A. Elmokashfi, A. Kvalbein, C. Dovrolis, BGP churn evolution: a perspective from the core, in: INFOCOM'10: Proceedings of the 29th Conference on Information Communications, IEEE Press, Piscataway, NJ, USA, 2010, pp. 1208–1216.
[18] P.-c. Cheng, X. Zhao, B. Zhang, L. Zhang, Longitudinal study of BGP monitor session failures, SIGCOMM Comput. Commun. Rev. 40 (2) (2010) 34–42. <http://doi.acm.org/10.1145/1764873.1764879>.
[19] T. Griffin, G. Wilfong, Analysis of the MED Oscillation Problem in BGP.
[20] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S.F. Wu, L. Zhang, Observation and analysis of BGP behavior under stress, in: ACM SIGCOMM Internet Measurement Workshop (IMW), 2002.
[21] O. Maennel, A. Feldmann, Realistic BGP traffic for test labs, in: Proceedings of ACM SIGCOMM, 2002.
[22] J. Wu, Z.M. Mao, J. Rexford, J. Wang, Finding a needle in a haystack: pinpointing significant BGP routing changes in an IP network, in: Symposium on Networked System Design and Implementation (NSDI), 2005.
[23] A. Flavel, O. Maennel, B. Chiera, M. Roughan, N. Bean, CleanBGP: verifying the consistency of BGP data, in: Internet Network Management Workshop 2008, 2008.
[24] S. Sangli, E. Chen, R. Fernando, J. Scudder, Y. Rekhter, Graceful Restart Mechanism for BGP, RFC 4724 (Proposed Standard). <http://www.ietf.org/rfc/rfc4724.txt>, 2007.

Pei-chun Cheng has been pursuing the PhD in Computer Science at University of California, Los Angeles, since 2007. He received his B.S. and M.S. degress from National Taiwan University in 2000 and 2002.



Beichuan Zhang is an assistant professor in the Department of Computer Science at the University of Arizona. His research interests include Internet routing and topology, multicast, network measurement, and security. He received PhD in Computer Science from the University of California, Los Angeles in 2003 and B.S. from Beijing University, China in 1995.

Dan Massey is an associate professor at Computer Science Department of Colorado State University. Dr. Massey received his doctorate from UCLA and is a senior member of the IEEE, IEEE Communications Society, and IEEE Computer Society. His research interests include protocol design and security for large scale network infrastructures, and he is currently the principal investigator on research projects investigating techniques for improving the Internet's naming and routing infrastructures. He is a co-editor of the DNSSEC standard (RFC 4033, 4034, and 4035).

Lixia Zhang received her PhD in Computer Science from the Massachusetts Institute of Technology. She was a member of the research staff at the Xerox Palo Alto Research Center before joining the faculty of UCLA's Computer Science Department in 1995. In the past she has served as the vice chair of ACM SIGCOMM, Co-Chair of IEEE Communication Society Internet Technical Committee, and on the editorial board for the IEEE/ACM Transactions on Networking. She also served on the Internet Architecture Board.