On the Correlation between Route Dynamics and Routing Loops

Ashwin Sridharan† and *Sue. B. Moon*†† and *Christophe. Diot*⊕
ashwin@ee.upenn.edu, sbmoon@sprintlabs.com, christophe.diot@intel.com [*]

ABSTRACT

Routing loops are caused by inconsistencies in the routing state of the network. Although undesirable from this aspect, they can provide insight into the routing dynamics that caused them. In this work we present a methodology that utilizes a priori knowledge of loops to study the correlation between routing loops and routing events that could have caused them. We apply our technique to associate route changes with packet loops detected in actual traffic traces collected from the Sprint Backbone. Our study shows that a strong correlation exists between loops and changes in the BGP routing state while link state protocols like ISIS are seldom responsible for such events. Our analysis also identifies factors that influence the distribution of loop path lengths as well as the effectiveness of our detection techniques.

1. INTRODUCTION

As the Internet grows both in size and volume of carried traffic, so does the likelihood of failures, congestions and other anomalies. These in turn increase routing fluctuations which can result in different parts of the network existing in inconsistent routing states. Consequently, there is an increased probability of routing loops being formed which, coupled with the size of the network, may also last longer. Hengartner *et. al.* characterized in [3] the path length and duration of loops as well as their impact on the delay and loss performance of the Sprint network. They proposed a methodology to detect loops and showed that although a significant number of loops can occur in practice, they seldom impact the performance of the network. The focus in [3] however, was on the characteristics of the loops themselves rather than routing events that may have been responsible for their creation.

Extensive work has been done in understanding the dynamics of the popular routing protocols such as BGP [10] and ISIS [1]. In [6] the authors conducted a detailed investigation on the evolution of the BGP routing state. [7] presents an in depth investigation into the frequency and sources of BGP mis-configurations. Both pieces of work however, decoupled the study of route dynamics from their impact on traffic. ISIS dynamics were studied in [4] which investigated the frequency of ISIS state changes in the Sprint backbone and their impact on service convergence for VoIP traffic.

In this paper, we examine routing events from the perspective of traffic anomalies that they generate. Our emphasis is on events that result in inconsistent routing states, thereby creating loops. Towards this end, we identify loops in traffic traces and correlate them with the routing events that may have created them. Our contribution is three-fold. First, we extend the methodology for loop detection presented in [3] to make it more robust. The new methodology allows for detection of up to 10% more loops. Second, a new matching technique is presented to associate both BGP and ISIS routing events with loops detected in traffic traces. Finally, we apply our matching technique to traffic traces as well as routing information collected from the Sprint backbone. The results allow us to study the type of routing events that can create loops, as well as establish a correlation between the path length distribution of loops and their causal events.

Following the classification introduced in [3], routing loops may be broadly divided into either *transient* or *persistent* loops based on their duration and cause. *Transient* loops are formed due to the dynamic nature of the networks which cause routing changes. They occur in normal course of operation due to the finite propagation speed of information which results in inconsistent routing information across different parts of the network. They typically resolve as the routing protocol adapts to the network change and routing states converge, and hence are transient in nature. *Persistent* loops on the other hand are usually caused by mis-configurations (accidental or deliberate) or chronic instability in routing and hence last longer. By their very nature, analyzing *persistent* loops requires a longer history of both traffic traces as well as the routing information. Moreover, even identifying the precise routing event responsible for a loop would require routing state information of all routers involved in the loop over the entire duration of the inconsistency. Obtaining such extensive information is virtually impossible on a network that is as complex and large as the Sprint backbone. As a result, rather than directly identifying the precise event responsible for the loop, we investigate the correlation between routing events and loops. Also, we shall focus only on transient loops and those persistent loops that originated during the trace.

From the perspective of an Autonomous System (AS)[1], changes in the routing state can be caused by both the External (EGP) and

[*]† Ashwin Sridharan, University of Pennsylvania, Philadelphia, PA.
†† Sue B. Moon, Sprint ATL, Burlingame, CA.
⊕Christophe Diot, Intel Research, Cambridge, UK.
This work was done, when Ashwin Sridharan and Christophe Diot were at Sprint ATL.

[1]An Autonomous System refers to a network under a single administrative entity, eg. an ISP

IMC'03, October 27–29, 2003, Miami Beach, Florida, USA.
Copyright 2003 ACM 1-58113-773-7/03/0010 ...$5.00.

Internal (IGP) Gateway Protocols, in response to various events like congestion, policy changes and network element failures. The former can announce new next hops to enter or exit an AS, as well as modify AS paths, while the latter could induce re-computation of a new shortest path within an AS. Hence, in order to establish a correlation between routing state changes and routing loops, we need to study changes in both the EGP and IGP. Towards this end, we collect information about routing events for both BGP and ISIS[2] protocols as well as traffic traces from monitoring points in the Sprint backbone. We then process both traffic and routing information to match the routing events with packet loops observed in the traffic traces.

Our study identifies a strong correlation between routing loops and updates in the BGP state of the network while ISIS events are shown to almost never result in loops. We also detected a large number of persistent loops in some of the traces which are associated with BGP state changes. Finally, we establish a correlation between the path length of a loop with the BGP updates.

The rest of the paper is structured as follows. The methodology to detect and match packet loops to routing state changes is explained in detail in Section 2. Section 3 presents the experiments we conducted and their analysis. Section 4 concludes this work.

2. METHODOLOGY

Studying packet loops from the perspective of routing events requires three steps. First of course, we must collect the traffic and routing data which contains the relevant information. Second, we must detect packet loops in the traffic traces. Finally, we need a technique to match the loops with routing events. In this section we explain the methodology we follow in order to accomplish these three steps.

We will use the following terminology regarding a loop in the rest of the paper. A *packet loop* refers to a singe *packet* caught in a loop. Hence, if multiple packets were caught in a loop, each packet would be identified with a distinct *packet* loop. We define a *routing loop* to be set of *packet loops* which may be associated with a specific routing event.

The first step in the process is the collection of all the relevant information. Packet traces were collected from the Sprint backbone through IPMON machines [2] located at various POPs in the network. The machines dump the first 44 bytes of each packet which contain the IP and underlying protocol headers. Routing information for the BGP protocol was collected using the Zebra [5] listener as well as from the RouteViews Project [8]. ISIS routing information was obtained using the PyRT [9] listener deployed on the Sprint Backbone.

The next step is to detect packet loops for which we utilized a modified version of the technique developed in [3]. The original method is a two-step process that involves breaking up the packet trace into chunks and then hashing packets from each chunk into separate hash buckets. Each hash bucket was then processed independently. Packets in a hash bucket were examined to check if they are identical. Identicalness of two packets was established if both header contents sans the TTL and IP checksum fields were identical and the TTL differed by at least 2. A drawback of the original technique is that it ignores loops that span hash bucket boundaries or chunks. In order to account for these loops, we maintain a history of all loops that are within 500 ms of the current hash bucket boundary. The time duration was chosen based on the maximum inter-packet delay observed in loops in [3]. Packets are now matched with loops stored in history before the packets in the current hash bucket. The modified technique accounts for an increase of about 10% in the number of detected loops as compared to the original method. Note that the detection method gives us a set of *packet loops*. The last step is to identify *routing loops* by correlating the *packet* loops with BGP and ISIS events. The matching technique for each protocol is described in the next two sub-sections.

[2]ISIS is the IGP running on the Sprint Backbone

2.1 Matching Loops with BGP events

BGP is the dominant inter-domain routing protocol which allows routing information to traverse Autonomous Systems. It is a distance vector protocol which computes the next-hop to exit or enter an Autonomous System, as well as the sequence of ASes to traverse, otherwise known as the AS Path. A simple example demonstrates how a routing loop can occur due to a BGP route change[3]. Consider the network shown in Figure 1 which consists of two ISPs, namely A and B, and a customer who is multi-homed[4]. Further, assume that the customer gives AS A preference for receiving all traffic, that is, AS A is the preferred route. Initially traffic entering AS A through PoP[5] X will traverse its network to reach the customer. Now assume that due to some policy change, the customer decides to receive traffic via AS B. From the figure, it is easy to see that routers in PoP Y will be the first in AS A to receive this information. Due to transmission latency, there will be a time gap before routers in PoP X update their state to match that in PoP Y. During this duration, all packets destined to the customer will be forwarded from PoP X to PoP Y and then looped back to PoP X until the routing states are synchronized.

This example highlights the salient features of BGP state changes that can cause loops, namely a change in the AS path, a change in the next hop, or both. Note that the changes in next hop or AS path in itself do not represent sufficient conditions for a loop to form, but are only necessary conditions. Indeed, a change could take place without a loop being formed. However, correctly surmising such an event would require knowledge of the BGP tables of all neighbouring ASes, which is not possible to obtain in practice. Hence, for simplicity we assume that observation of such an event from a single point is sufficient evidence. In order to further strengthen this assumption, we impose the condition that the event have occurred in the proximity (in time) of the *packet loop*.

As mentioned previously, BGP updates were collected from both a passive Zebra listener on the Sprint backbone and through the Routeviews project. The Zebra listener created a passive adjacency with a Router Reflector[6] allowing it to receive all BGP updates inside the Sprint network that changed the AS path or next hop of traffic flowing through the PoP. This provides fine grained information of routing dynamics as seen by a single PoP. The Routeviews project collects external BGP updates from various ASes, providing a broader but coarser picture.

The BGP updates were matched with the packet loops in the following fashion :

1. We determine if any packet loop is potentially impacted by a BGP update through a longest prefix match for the destination address of the packet loop on the set of advertised and/or withdrawn routes in the update.

[3]Note that the route change could be either a next-hop change, or an AS Path change, or both.

[4]Multi-homing refers to networks that connect to the Internet through two or more ASes

[5]A PoP or "Point of Presence" is a collection of routers within close geographical vicinity.

[6]A Route Reflector acts as a logical node for a group of routers, participating on their behalf in BGP protocol interactions, thus improving scalability.

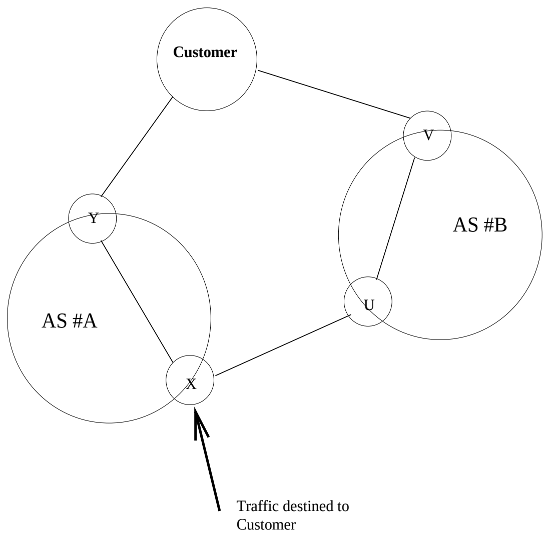


Figure 1: Demonstrating a loop due to BGP update

2. Next we determine if the BGP update lies in the temporal vicinity of the loop. This was set to a value of 2 minutes.

3. If both previous conditions are satisfied, then we examined any change in the current next hop or AS path of the destination prefix by feeding the update to a Zebra router which emulated the BGP decision process.

4. If the first 2 conditions are satisfied and a change in next hop or AS Path is detected we conjecture that the loop was caused by this update. Since the Sprint routing traces were obtained by peering as a Route Reflector client, any change in next hop or AS path from such updates reflect changes in the best next hop or AS Path.

A note is in order here regarding our identification of events related to transient and persistent loops. It should be clear that our technique is equally applicable to both transient and persistent loops. However, as mentioned in Section 1, identification of events related to persistent loops requires a much longer history in terms of the traffic trace and routing information than was available for this study. Hence, in this work, the emphasis is on identifying transient loops. The study in [3] showed that transient loops have very short durations (less than 10 seconds), which was the reason for setting the requirement that the temporal proximity value of the routing event to the loop be 2 minutes. Note that this does not preclude identification of events causing persistent loops, but only limits identification to those loops which started during the trace. Indeed, as we shall in Section 3.2, we were able to associate some persistent loops with BGP updates.

2.2 Matching ISIS Loops

We next turn our attention to ISIS, a link state routing protocol. Link state routing protocols typically collect information about the entire logical topology (within an AS) by exchanging Link State Packets (LSPs) and then run the shortest path algorithm on the topology to compute routes. In order to keep this information updated, LSPs are generated and flooded whenever events like link failures and weight changes modify the logical topology. Since LSPs need to be flooded to every part of the network, there is an interval during which the databases of different routers are momentarily unsynchronized which may result in transient loops.

Similar to detection of loops caused by BGP updates, directly detecting loops caused by ISIS events is intractable, since it would require knowledge of every router's database at every time instant. Instead we use an indirect method to infer if a routing loop can be caused by ISIS state changes. The method, similar to the one used to detect loops caused by BGP updates, essentially consists of monitoring changes in the forwarding path. However, it differs from the detection technique used in BGP in that knowledge of the entire topology is required to identify the forwarding path. On the positive side, it provides a more reliable technique to decide whether a loop could have been caused by the ISIS event.

In order to to compute the forwarding paths, a passive ISIS listener [9] established an ISIS adjacency with a SprintLink router. This enabled it to receive all LSPs flooded throughout the network. Knowledge of the LSPs was used to construct the ISIS topology and consequently compute shortest paths from any node. We now explain the detection procedure in further detail.

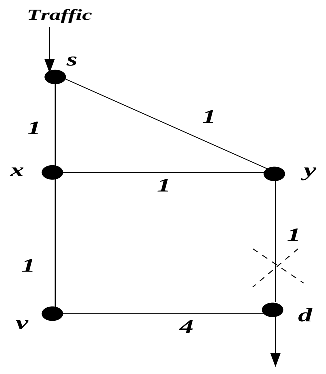


Figure 2: A loop caused due to ISIS Failure

Figure 2 represents an example network, where for simplicity each link is assumed to be bi-directional with the same weight in either direction. Let link (x, y) be the link from which packet traces are being collected. Let there be a flow from ingress node s to egress node d. With the weight setting specified in the figure, it will take the shortest path $s - y - d$. Now suppose that at some time t the link $y - d$ goes down. Until node s receives the LSP regarding this event, it will continue to regard y as its next-hop to forward packets to d. However y will recompute its shortest path to d as $y-x-v-d$ and will forward packets destined to d via node x. Similar to s, until the time node x receives an LSP from either y or v about the failure *and* processes it, x will forward all packets headed to d along the original shortest path $x-y-d$, thus creating a loop $x-y-x$. To further illustrate this, we show the sequence of how nodes x and y respond to the reception of an LSP, in the time-line of Figure 3.

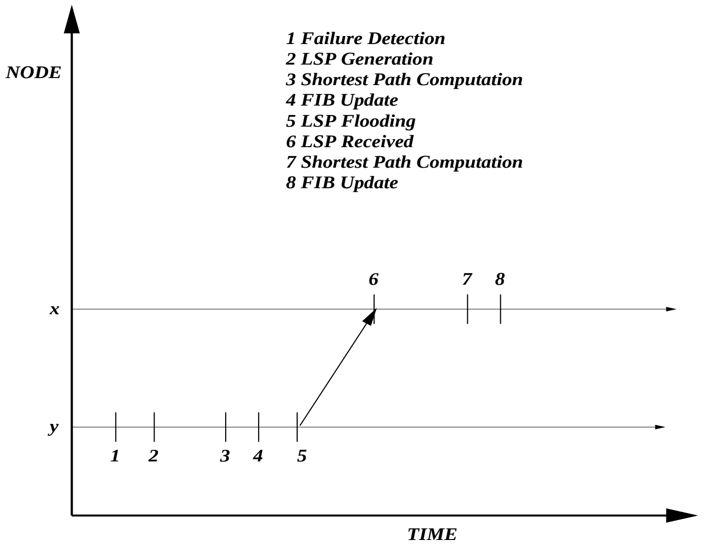


Figure 3: Time-line at nodes x and y after ISIS Failure

Note from the time-line that, between time epochs 5 and 8 (that is, from the time node y forwards the LSP to node x, to the time node x receives the LSP and updates its FIB), the LSP database at nodes x and y will be inconsistent with x assuming its shortest path to d is still through y. Hence a loop will be created for this duration.

The above example illustrates two important points. First, the creation of the loop involves a change in the forwarding path of node x. In this case, from $x-y-d$ to $x-v-d$. This in itself is not sufficient however since it is possible that forwarding paths may change without loop creation (for example, in the presence of equal cost multiple paths). The second observation is that at some point in time, nodes on the new shortest path of x pointed back to x itself. To see this, observe that the loop was formed because the new path $y-x-v-d$ contains node x whose previous path ($x-y-d$) went through node y.

We now formalize these two observations in the form of two conditions to determine when loops can be formed and when they cannot. In our subsequent discussion, the link from which packet traces were collected will be referred to as the *observed link* and the ingress node of the link will be referred to as the *observation node*. All paths are with respect to a specific destination node. The path to a destination node e from node x at a time instance t will be denoted by $P_{x,e}(t)$ and the next hop by $NH_{x,e}(t)$.

Condition 1: A necessary condition for a routing loop involving the *observed link* to occur in an SPF protocol is a change in forwarding path of the *observation node*.

The proof is quite straightforward. If the loop occurred *because* of a change in the forwarding path of the *observation* node, then the condition clearly holds true. Else, let at some time t the *observed* node, say x, has a loop free path $P_{x,e}(t)$ to another node, say e. Let at some time $t+\epsilon$ an event causes a loop that involves packets forwarded by x to e. Then clearly, there exists a node $l \in P_{x,e}(t)$ which is also involved in the loop, thereby violating the requirement for a loop-free path. This will result in a change in the forwarding path of node x.

Note that this is a necessary and not sufficient condition for loop formation, that is, a change in the forwarding path need not always result in a loop. However, unlike in the detection of loops due to BGP updates, we can look for stronger conditions that must be true for a loop to occur. This is because the ISIS protocol collects information regarding the entire topology for path computation.

Condition 2: Let the *observation node*, say x, received the LSP related to the event that caused a loop, at time $t+\delta$. Subsequently, node x changed its path to the destination, say e, from $P_{x,e}(t)$ to $P_{x,e}(t+\delta)$. Then, an *observed link* may be involved in the loop, occurring at time $t+\epsilon$, if and only if one of the following condition holds :

1. Case 1: The *observation node* has updated its path but a set of nodes, say $N' \subseteq P_{x,e}(t+\delta)$ on the *new* path, that were originally pointing to the *observation node* at time t, have not yet updated their paths in response to the change. In other words, the time of update for x is $t+\delta < t+\epsilon$, and the time of update for a node $l \in N'$ is $t+\delta_l > t+\epsilon$, so that there exists a walk
$$W = \{NH_{l,e}(t+\epsilon) : l \in N'\}$$
that returns to x.

2. Case 2: This is the exact opposite of Case 1. Some nodes on the original shortest path of x at time t, say $N' \subseteq P_{x,e}(t)$, have updated their shortest paths in response to the LSP, to point to the *observation node*, x, but the *observation node* is yet to record this change. In other words, the time of update for node x is $t+\delta > t+\epsilon$, and the time of update for a node $l \in N'$ is $t+\delta_l < t+\epsilon$ so that there exists a walk
$$W = \{NH_{l,e}(t+\delta) : l \in N'\}$$
that returns to x.

It is easy to see that if either case of *Condition 2* were to hold true, a loop would be formed. Vice versa, by definition of a loop, we must have one of these two cases to hold true if a loop was formed.

Note that the above conditions imposes an order on the events for a loop to occur and require knowledge of the *sequence* of events from every router's perspective. However, as mentioned previously, it is intractable to obtain this information, Instead, we only check for a looser version of *Condition 2*. Specifically, for every LSP, we check if either case of *Condition 2* could have held before or after the LSP was received, *independently* for each node. Clearly, this can result in false positives but they can be eliminated by other means, e.g., by checking the length of the loop and the possible set of loop paths on the ISIS topology as well as the proximity (in time) of the event to the loop.

We now describe our implementation of the procedure for associating loops with ISIS events. The first step in determining if a loop was caused by an ISIS event is to resolve the egress ISIS router for each destination address in the loop. We then proceed with the ISIS event detection phase for the duration of the traffic trace from which the loops were extracted, as enumerated below.

1. On reception of an LSP, we compute the shortest path from the *observation node* to the ISIS egress router of the looping packets.
2. If the forwarding path has changed, *and* we are in the temporal vicinity[7] of a loop, we check to see if either case of *Condition 2* holds.
    - For case 1, we compute shortest paths of all nodes on the *new* forwarding path of the *observation node*, using the topology *before* the LSP was received. We then check if the *observation node* lies on any of the nodes' shortest paths.
    - For case 2, we compute the shortest paths of all the nodes on the *original* forwarding path of the *observation node* using the topology *after* the LSP was received. We then check if the *observation node* lies on any of the nodes' shortest paths.
3. If either case holds true, we conjecture that the ISIS event caused a loop.

Before we end this section, we must point out that our discussion regarding the applicability of the matching technique to BGP updates and transient and persistent loops also holds for ISIS events. Another note worth mentioning, is that since BGP updates can change the egress router for a destination, this should be taken into account while tracking ISIS event changes. Specifically, changes in the forwarding path due to egress router (BGP next hop) changes should not be attributed to an ISIS event. This is accomplished by a look ahead scheme which notes the time epoch of the current and *next* BGP update and tracking ISIS evens between these two epochs. Any change in the forwarding path during this epoch would then be directly attributable to ISIS events only.

3. EXPERIMENTS AND RESULTS

In this section, we present the results obtained by applying our matching technique to the packet loops and the routing information. For our experiments we collected packet traces from OC-48 links in the SprintLink network. All links were between backbone routers. Three of the traces are 1 hour long, while the other three are 12 hours long. Table 1 shows the number of packet loops detected in each trace using the loop detection technique explained in Section 2. Observe that the number of detected loops varies over an order of magnitude.

Trace Name	No. of Packet Loops	Duration (hrs)
NYC-20	2476	1
NYC-21	3838	1
NYC-23	1895	1
NYC-22	8672	12
NYC-24	719	12
NYC-25	1691	12

Table 1: Number of Packet Loops in Each Trace

In subsequent discussion, we define the TTL Δ of a packet loop to be the round trip decrement in the TTL field of the packet, that is, the length of the loop. For purposes of classification, we define loops that lasted over 1 week from the point of collection of the trace as persistent loops. This choice is used to simply illustrate the difference in duration between persistent and transient loops. In order to identify such loops, traceroutes were performed at several time instances over a duration of 1 week on the destination address of all the identified packet loops from the traffic trace to confirm that the loops still persisted.

[7]set to 2 minutes

We first present our findings in regard to matching loops with ISIS events in Section 3.1. Section 3.2 investigates the efficacy of our matching technique for BGP updates. Finally, Section 3.3 identifies a correlation between the distribution of TTL Δ for loops and the BGP updates causing them.

3.1 Loops related to ISIS changes

In [4] it was observed that on an average there are 2 ISIS failures per hour that occur on the Sprint Backbone. Although we observed similar failure rates, none of the loops in any of the traces could be correlated with an ISIS event based on the criteria described in Section 2.2. This can be attributed to the extensive use of equal cost multiple paths between PoP pairs on the Sprint Backbone for load balancing. The presence of multiple forwarding paths provides the ability to switch over to another path in case of link failures without overlapping with the previous path of another node. Consequently neither case of *Condition 2* of Section 2.2 holds, resulting in loop-free operation. Another possible factor contributing to the lack of ISIS events that could cause loops is that ISIS is a link state protocol which uses the entire topology to compute paths. This is known to greatly reduce the duration of inconsistent routing state as compared to a distance vector protocol like BGP [11].

The observation that ISIS events do not create loops lends weight to the observation that convergence time for ISIS is not critical, at least from the perspective of routing convergence. By maintaining equal cost multiple paths one can quickly switch over to an alternate path without allowing loops to be formed.

3.2 Matching Loops to BGP Routing Events

In the previous sub-section, we saw that no loops could be correlated to ISIS events. This implies that the only other routing protocol that could have caused the loops is BGP[8]. In Table 2, we show the match ratios for BGP updates collected from the Sprint backbone with packet loops from the various traces. The first column displays the fraction of loops classified as transient that were associable with BGP updates. The second column displays the fraction of loops that were persistent but still were identifiable with BGP updates while the third column displays the fraction of persistent loops with no updates. The final column displays the number of "accounted" loops, that is, the sum of the previous three columns. Persistence of loops was identified using the definition established at the beginning of the section and transience was established through elimination of persistent loops.

[8]BGP and ISIS are the only two protocols which propagate routing state changes within the Sprint Network. Hence even manual routing changes should result in BGP or ISIS state changes

Trace	% Transient & BGP Updates	% Persistent BGP Updates	%Persistent No Updates	Total
NYC-20	40.1	0	50.8	90.8
NYC-21	80.2	0	7.5	87.9
NYC-23	3.3	0	0	3.3
NYC-22	18.8	0	80.6	99.4
NYC-24	70.0	0	0	70.0
NYC-25	43.7	15.5	0	59.2

Table 2: BGP Update Matches for Loops using Sprint Link BGP Information

As can be seen from the table, in most cases we were able to account for more than half the loops, as either identifiable with a BGP event or persistent (and unidentifiable with a BGP event), although the fractions vary quite a bit. Only in the case of NYC-23 were we unable to identify more than 10% of the loops as being caused due to BGP dynamics. There are several factors which can potentially contribute to this varying success ratio in matching transient loops to routing events. An important factor that can affect the match ratio is the possibility that a large fraction of loops were persistent. Such loops are often caused by mis-configurations and as mentioned before, may require a much longer measurement interval than available for this work. Indeed, as can be seen from Table 2, we found a significant number of persistent loops. For NYC-20, 50.8% of the loops were persistent loops. For NYC-22 and NYC-25, the numbers were 80.2% and 15.5% respectively. NYC-25 presents an interesting case in that we were able to associate *all* the persistent loops with BGP events. That is, the persistent loops originated during the packet trace and hence our technique was able to identify the BGP events that may have created them. However, for all the other traces, none of the persistent loops could be attributed to BGP events, indicating that they originated before the collection of the trace. In fact, in the case of NYC-23, we did not detect any persistent loops, which implies that other factors exist that can affect the match ratio.

One of these factors that may adversely affect the effectiveness of our technique, particularly for the NYC-23 trace, could be that the changes in BGP tables occur external to the Sprint AS and hence are not visible in the BGP routing tables. In order to verify this conjecture, we applied our matching technique to updates obtained from the RouteViews project [8] to three traces, NYC-20, NYC-21 and NYC-23. Whereas, updates collected by the Zebra router peered within the Sprint Network would only contain iBGP updates passed on by a Route Reflector, RouteViews stores updates obtained from other ASes. If the loops were indeed caused by factors external to the network, it is plausible that RouteViews should offer a better perspective. The match ratio (for transient loops) using RouteViews updates are shown in Table 3.

Trace	% Sprint Matches	% RouteViews Matches
NYC-20	40.1	43.1
NYC-21	80.2	82
NYC-23	3.3	10.6

Table 3: BGP Update Matches for Loops using RouteViews Information

Observe that the largest increase in correlation is for the NYC-23 trace, by about 7%. This seems to indicate that the loops in the NYC-23 trace were caused by BGP changes outside the Sprint AS and hence not visible to the measurement node in question[9]. To further support this assertion, in Table 4, we show, for each trace, the average number of ASes traversed from the Sprint network to reach the destinations of the loops. This was computed by dumping 10 BGP routing tables from the monitored node over a period of 3 months and averaging the AS paths over these samples for each loop destination. The path length indicates that loops of the NYC-23 trace traverse the largest number of ASes, providing further evidence for our reasoning behind the low match ratio. We also note that this trend holds true for the other traces. Loops from traces that traversed longer AS paths (eg. NYC-24 and NYC-25) had poorer match ratios as compared to loops that were inside or close to the Sprint AS (eg. NYC-20, NYC-21, NYC-22).

Yet another factor that affects the success ratio of our technique could be the geographical distribution of loop destination addresses. Intuitively, if the loops were spread over a large number of ASes, then it is less likely that we would be able to identify the causal BGP events. This is because the Sprint Network may be peering (if at all) with these ASes at different points on the backbone and the PoP containing the measurement node need not participate in all of them. Consequently, given the large spread of the ASes, it would be less likely for the measurement PoP to be directly involved in BGP route changes that caused loops, even though it lies on the loop path.

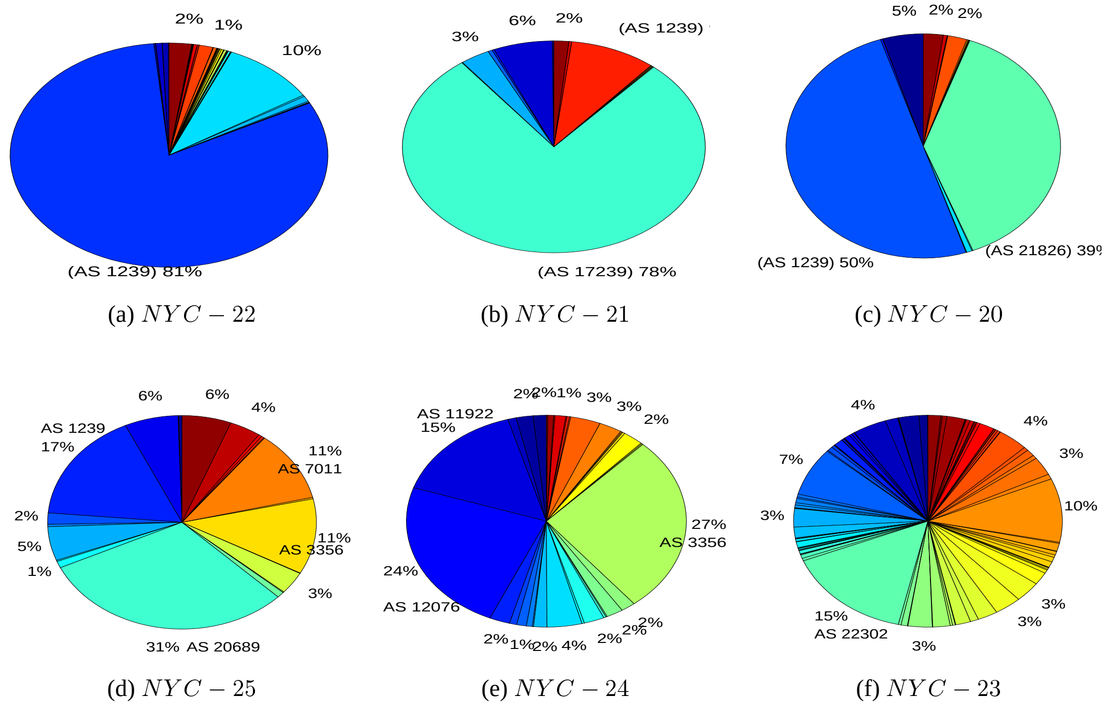
We tested this hypothesis by plotting the Origin AS of each loop destination address for all the 6 traces in Figure 4. An immediate observation is that a large fraction of the loops for NYC-21 (78%) and NYC-22 (81%), belong to one AS. In case of NYC-21, it was a customer, and for NYC-22, it was within the Sprint AS itself, which increases the likelihood of receiving all the updates that resulted in loops. We note that for both of these traces, we were able to account for most of the loops (either transient or persistent). Loops from NYC-20 are distributed over a larger number of ASes as compared to NYC-22 and NYC-21, but there are still 2 dominant ASes which account respectively for 50% and 39% of the loops, again Sprint and a customer. In this case also, most of the loops are accounted for. The next step in this trend is seen in the Origin AS distribution for NYC-24 and NYC-25. Unlike the first 3 cases, there is no single dominant AS which accounts for most of the loops. Instead we have multiple dominant ASes, that account for small fractions of destination blocks. This observation along with poorer loop match ratios as compared to NYC-20, NYC-21 and NYC-22 supports our original hypothesis that wider destination distribution may lead to poorer matching ratios. Lastly, NYC-23, whose loop destination addresses are spread over the largest number of ASes has the poorest match ratio.

3.3 TTL Δ Distribution of Routing Loops

[9]In fact, even updates, outside the measurement PoP need not be visible, since they may be filtered by the Route Reflector.

Trace	Avg. No. Of ASes traversed
NYC-20	1.34
NYC-21	1.04
NYC-23	1.74
NYC-22	0.513
NYC-24	1.61
NYC-25	1.63

Table 4: Average Number of ASes traversed



(a) $NYC-22$ (b) $NYC-21$ (c) $NYC-20$

(d) $NYC-25$ (e) $NYC-24$ (f) $NYC-23$

Figure 4: Distribution of Origin AS

We next turn to studying the distribution of the TTL Δ of the loops and how they are correlated to the BGP updates that were associated with the loops. The main goal of this section is to demonstrate that the *range* of distribution of TTL Δ of the loops is correlated with the *number* of BGP updates that may be associated with the loops. If only a small number of BGP updates were responsible for the loops, then the TTL Δ of the loops will be spread over a narrow range as compared to when a large number of updates caused all the loops. The intuition behind this argument is as follows. If the number of updates responsible for loops is very small, then it is likely that the distribution of destination addresses of the loops is also quite small. Consequently a large fraction of the loops will follow the same path resulting in a closely clustered TTL Δ distribution.

To test this hypothesis, we have plotted three relations for 5 traces, NYC-21, NYC-20, NYC-22, NYC-24 and NYC-25 in Figures 5, 6, 7, 8, 9 respectively. For each trace, we show 3 figures, namely, the number of *packet* loops as a function of the TTL Δ, the number of *packet* loops associated with each BGP update, and finally the average TTL Δ of all the packet loops associated with a BGP update, or in other words, the TTL Δ of a *routing* loop. For notational convenience, they are labeled (a), (b) and (c) respectively for each trace. Since we were able to associate very few loops from the NYC-23 trace with BGP updates, we omit discussion of the impact of BGP updates on this trace.

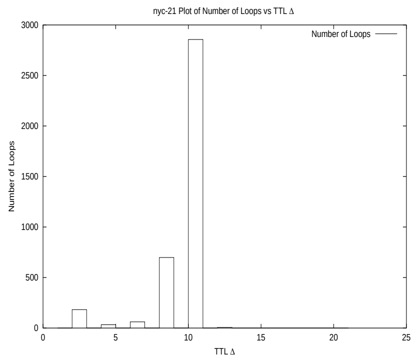
We begin our discussion of these plots with the NYC-21 trace which had the largest fraction (80.2%) of loops accounted for by BGP updates. Figure 5(a) shows the number of packet loops as a function of the TTL Δ. Observe that almost all loops are closely clustered around just two TTL Δ values. 74% of the loops have a TTL Δ of 10, while 18% of the loops have a TTL Δ of 8. In accordance with our hypothesis, such a narrow TTL Δ distribution should result from only a few updates being correlated with all the loops. We confirm this observation from the next two figures. In Figure 5(b), we have plotted the number of *packet* loops that are correlated with each BGP update. A strong correlation is observed with only a few BGP updates. In fact most loops (approximately 2900 (74%)) are correlated with a single update (update index 8). In Figure 5(c) we have plotted the average TTL Δ of the packet loops that were correlated with each BGP update. If we look at the average TTL Δ of loops caused by update index 8, we see that it is close to 10 (the update also accounts for some loops with a TTL Δ of 8) coinciding with the dominant TTL Δ mode of the trace.

Similar plots of the NYC-20 trace are shown in Figure 6. Observe from Figure 6(a) that the dominant TTL Δ of 14 which accounts for around a 1300 *packet* loops (50.8%), is *not* reflected in the BGP updates, i.e., no BGP update could be associated with these routing loops. The reason for this is that they were persistent loops that originated before the trace. Our analysis indicates that the persistent loops belonged to the same address block and shared the same path resulting in a common TTL Δ.

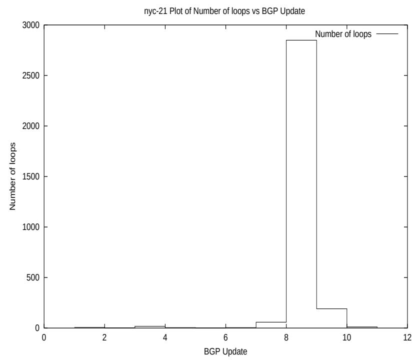
Turning our attention to the accounted loops (which number around a 1000), from Figure 6(b) observe that most of them (about 950 loops of them) can be associated with the BGP update indexed 2. The average TTL Δ of of this routing loop[10] is 9.5[11](from Figure 6(c)). On Figure 6(a), this turns out to be the second major cluster, centered around a TTL Δ of 10. These observations again support our hypothesis that a closely clustered TTL Δ occurs due to most loops being associated with a single or few updates.

[10]Recall that the routing loop is a collection of packet loops that may be associated with a single event.

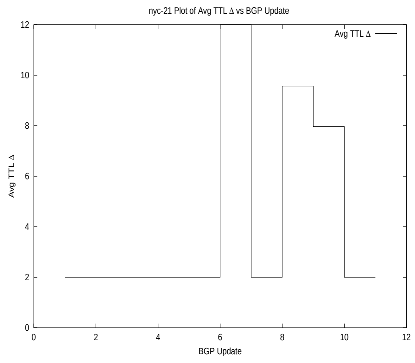
[11]The BGP update actually accounts for loops with TTL Δs varying from 8 to 10



(a) Distribution of TTL Δ of Loops

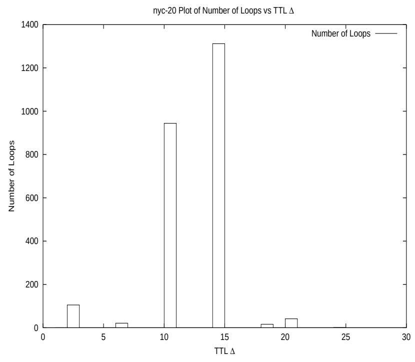


(b) Loops caused per BGP Update

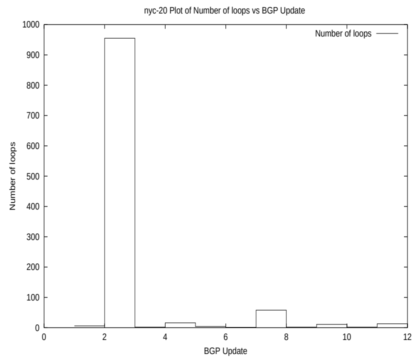


(c) Avg. TTL Δ per BGP Update

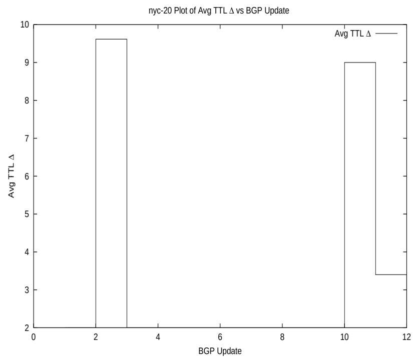
Figure 5: NYC-21: Routing Loop characteristics



(a) Distribution of TTL Δ of Loops



(b) Loops caused per BGP Update



(c) Avg. TTL Δ per BGP Update

Figure 6: NYC-20: Routing Loop characteristics

Plots for the NYC-22 trace are shown in Figure 7. 80% of the loops, around 6000 in number, are persistent loops and represented by the dominant TTL Δ mode of 10 in Figure 7(a).

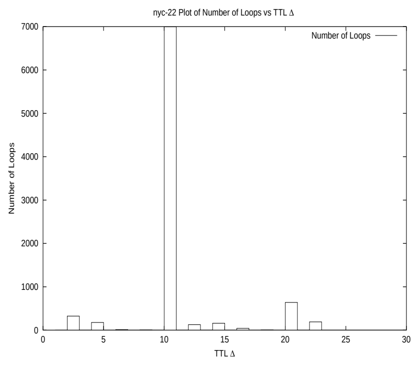
From Figure 7(b), the largest number of loops caused by a BGP update is approximately 800 loops and from Figure 7(c), they can be associated with the next dominant TTL Δ cluster of around 800 loops centered around 20. We also see that as compared to the previous 2 traces, there are more BGP updates that result in loops (Figure 7(b)), albeit in small numbers. In accordance with our hypothesis, an increase in the number of BGP updates that cause loops would imply an increase in the distribution of loop destinations, which in turn implies a more diverse loop path length distribution. This is reflected in Figure 7(a), where we see a number of TTL Δ modes around 10-15, although their relative size is dwarfed by the presence of 6000 loops with TTL Δ of 10.

Further support for this hypothesis is obtained from the TTL Δ distributions for the other two traces, NYC-24 (Figure 8) and NYC-25 (Figure 9). From Figure 8(b) and Figure 9(b), we note that a large number of BGP updates contribute to creating loops for NYC-24 and NYC-25 respectively. Hence, one would expect a wider range of destination addresses to be affected, as is confirmed from the Origin AS distributions of NYC-24 (Figure 4(e)) and NYC-25 (Figure 4(f)). This manifests itself by increasing the range of of TTL Δ distributions as seen from Figure 8(a) and Figure 9(b).

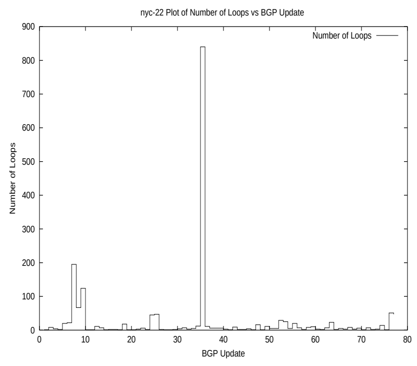
4. CONCLUSIONS

Although routing loops constitute a very small fraction of traffic, they present an interesting opportunity to study changes in routing behaviour and their impact on traffic. We have presented a methodology to associate packet loops with the routing events that may have caused them. In order to determine the efficacy of our technique we tested it on routing information as well as packet traces collected from high speed OC-48 links on an operational IP backbone. Our studies indicate that a strong correlation exists between loop instances and changes in the BGP routing state. Indeed, for most traces, we were able to associate a large fraction of loops with BGP updates. In practice, the technique was found to be sensitive to the distance of the BGP update origin from the point of measurement. Specifically, our results show that the technique is able to identify a large fraction of loop causing updates that originated in or near the measurement point. However loops that may have been caused by BGP changes farther away from the observation point may not be identified due to the updates getting filtered out.

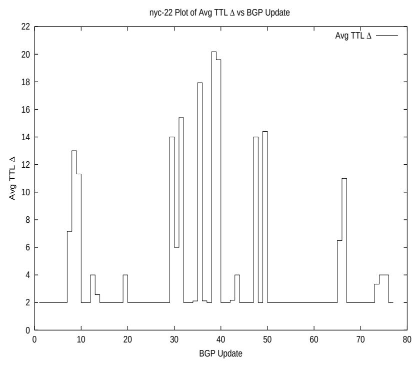
We were also able to establish a correlation between the path length distribution of the loops and the number of associated BGP updates. We conjecture that packet loops associated with only a few BGP updates should have a narrower path length distribution as compared to packet loops from traces that are associable with a



(a) Distribution of TTL Δ of Loops

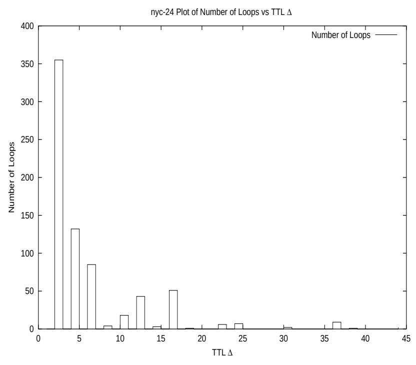


(b) Loops caused per BGP Update

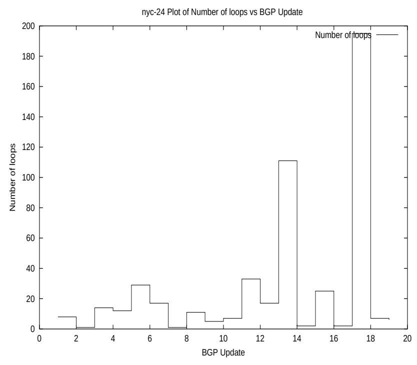


(c) Avg. TTL Δ per BGP Update

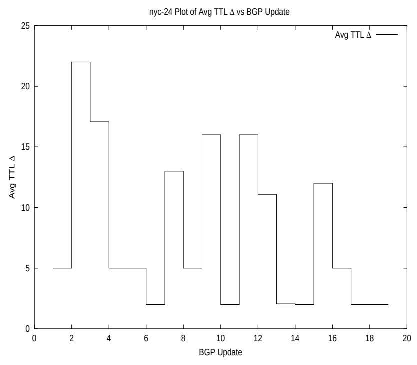
Figure 7: NYC-22: Routing Loop characteristics



(a) Distribution of TTL Δ of Loops

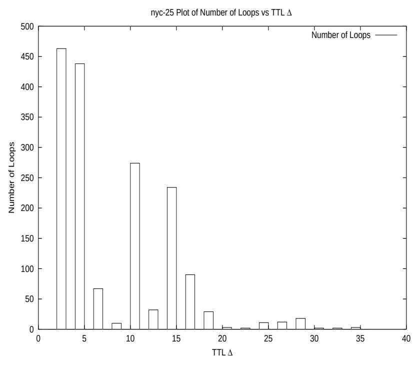


(b) Loops caused per BGP Update

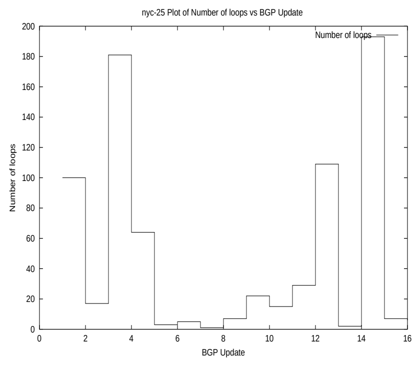


(c) Avg. TTL Δ per BGP Update

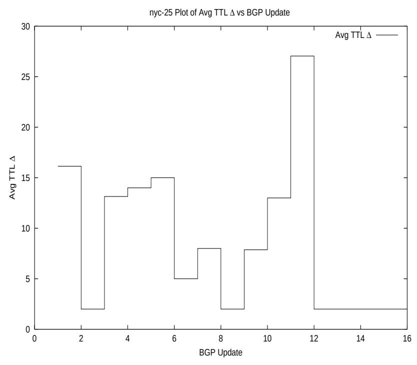
Figure 8: NYC-24: Routing Loop characteristics



(a) Distribution of TTL Δ of Loops



(b) Loops caused per BGP Update



(c) Avg. TTL Δ per BGP Update

Figure 9: NYC-25: Routing Loop characteristics

larger number of updates. The reason being that the fewer the updates, the more likely it is that the loops share common destination addresses. This in turn would result in a common network path. Analysis of the BGP updates correlated with the packet loops from 5 traces was found to support this argument.

There are however several threads that need to be explored further. Although our matching technique is effective, there are cases in which it does not perform well. We have offered a plausible explanation in that the loops may originate outside the observed AS, but a more complete study would be desirable. Finally, a more in depth study of persistent loops is required to determine their causes as well as their duration.

5. REFERENCES

[1] R. Callon. Use of OSI IS-IS Routing in TCP/IP and Dual Environments , 1990. Request for Comment. RFC 1195, available at http://www.ietf.org/rfc/rfc1195.txt.

[2] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet Level Traffic Measurements from the Sprint Backbone. In *IEEE Network*, 2003.

[3] U. Hengartner, S. Moon, R. Mortier, and C. Diot. Routing loops: Detection and analysis of their impact on loss and delay. In *ACM Sigcomm, Proceedings of IMW*, Marseille, November 2002.

[4] G. Iannaconne, C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of link failures in an IP backbone. In *ACM Sigcomm, Proceedings of IMW*, Marseille, November 2002.

[5] K. Ishiguro. Gnu zebra – routing software. Available at http://www.zebra.org.

[6] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *Proceedings of ACM Sigcomm*, Stockholm, 2000.

[7] R. Mahajan, D. Wehterall, and T. Anderson. Understanding BGP Misconfigurations. In *Proceedings of ACM Sigcomm*, Pittsburgh, 2002.

[8] D. Meyer. University of Oregon Route Views Project. http://antc.uoregon.edu/route-views/.

[9] R. Mortier. The python routeing toolkit, 2001. Available at http://www.sprintlabs.com/Department/IP-Interworking/Routing/PyRT/.

[10] Y. Rekhter. A Border Gateway Protocol 4 (BGP-4) , 1995. Request for Comment. RFC 1195, available at http://www.ietf.org/rfc/rfc1771.txt.

[11] S. Keshav. An Engineering Approach to Computer Networking, 1997. Addison-Wesley Publications.