Route Diagnosis in Path Vector Protocols

Dan Pei[1], Mohit Lad[1], Beichuan Zhang[1], Dan Massey[2] and Lixia Zhang[1]

UCLA[1] Colorado State University[2]

{peidan,mohit,bzhang,lixia}@cs.ucla.edu massey@cs.colostate.edu

UCLA CSD Technical Report TR040039

***Abstract*—In this paper we present a novel approach to route diagnosis for path-vector routing protocols such as the Internet's Border Gateway Protocol (BGP). Given a sequence of routing updates, our objective is to understand *why a previous path was removed, why the specific new path is chosen among all existing alternatives,* and *whether any of the routers along the path injected false information*. We collect necessary topological information for route diagnosis by inferring topological connectivity from routing updates, enhancing the path vector protocol with *root cause notification*, and using active queries between neighbor routers to learn missing information when necessary. In the absence of false routing updates, our design can reason about route removals and replacements with 100% accuracy. We also use majority votes among independent information sources, combined with active queries, to identify the source of false routing updates. Simulation results show that, in the presence of false updates, our design can achieve high detection rate and attacker identification rate with a low overhead.**

I. Introduction

In this paper we present a novel approach to route diagnosis for path vector routing protocols such as the Internet's Border Gateway Protocol (BGP). A given sequence of routing updates can be a result of valid topology changes, unintentional mis-configurations, or malicious attacks. The objective of route diagnosis is to determine what underlying event triggered these updates, and whether the update sequence corresponds to a valid execution of the protocol. In the specific case of path vector routing, one would like to understand (1) *why the previous path is removed*, and *why the specific new path is chosen among all existing alternatives*, and (2) *whether any of the routers along the path injected false information*.

The distributed nature of path vector routing and the large scale of the Internet topology make the route diagnosis problem difficult. A path vector routing protocol only signals the *effects* of a topology change, but not the *change* itself, through the announcement of a new best path. When a topological change occurs, a router may have multiple alternative paths from which to select. In order to understand why a path is removed and why a specific new path is chosen, route diagnosis requires the knowledge of the network topology and the specific change that triggered the updates. In a large scale, richly connected topology, a wide range of different topological changes can trigger the same set of updates, thus it is difficult to *infer* the change from its effects, as evidenced by the limited accuracy and correctness of passive inference effort[1][2][3] [4], and as confirmed by our simulation results in Section II-B. Furthermore, route diagnosis is made difficult by potentially false information being injected into the routing system, for example by operational errors or malicious attacks. If a router cannot tell why existing paths are removed or why new paths are chosen, it certainly cannot tell whether a newly received path contains false information. Our objective is not only to identify the existence of false information, but also to pin down exactly which piece of the information is false.

Our design is inspired by the diagnosis power of link-state routing algorithms; by letting every node know the complete topology and all the changes, a node's routing table is precisely synchronized with that of the others. In other words, a node can *know* precisely how other nodes make their routing decisions. Similarly, effective path-vector diagnosis could be achieved if a diagnosis station knew the full network topology (and policy) and was notified of all topological (and policy) changes. In this case, the station would be able to predict precisely how each of the routers would adjust its route after a given change. Large network size and rich connectivity can provide multiple alternative paths from the station to each destination. Thus when some node injects a false route into the network, as long as this node is not on all the alternative paths, the station would be able to detect the inconsistency between updates from different neighbors and nail down the origin of the inconsistency. However, it can be costly to learn and maintain the topology of a large, richly connected network. Ideally, one wishes to learn only the *relevant* topology information to achieve diagnosis goals.

In this paper, we present the design of ***D**iagnosis through **R**oot Cause Notification, topology **A**ccumulation, and **Q**uery (**DRAQ**)*. DRAQ aims at providing diagnosis information to network operators, such as raising an alarm after detecting the existence of false routing updates. Thus DRAQ listens to all the received routing updates, but does not block any of them, including those which are detected of carrying false information, as is done in [5][6][7][8]. DRAQ infers topological connectivity from routing updates to build a relevant topology map. To maintain this map updated, DRAQ also enhances the path vector routing protocol to carry a *root cause notification(RCN)* [9][10] in each routing update as a

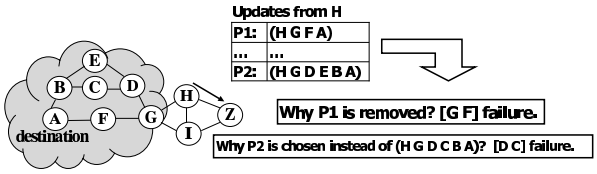


Fig. 1. Path Change Validation Problem

way to collect the latest topological changes. However as we show by examples in Section II, one cannot collect all the necessary topological information through such opportunistic learning from routing updates, even with RCN enhancement, to meet the need for route diagnosis. Therefore DRAQ design also includes active queries between neighbor nodes to acquire the missing topology and change information.

Our analysis and experiments show that, in the absence of false information, DRAQ can determine path removals and replacements with 100% accuracy. False routing updates are detected when the actual replacement path does not match the predicted path. Once the *existence* of false information is detected, a majority vote algorithm is applied to identify nodes with *independent* routing update sources, so as to pin down the misbehaving node. Our simulation results show that, in the presence of false information, DRAQ can achieve detection rate of at least 90%, and identification rate of at least 50%. The overhead cost is at most 4 query/reply messages for 90% of the cases.

The rest of the paper is organized as follows. Section II presents our design and simulation results for understanding why the previous path is removed and why the new path is chosen. Section III presents our design for the false information detection and identification problem, and its simulation results are presented in Section IV. Section V discusses our approach in DRAQ and future work. Finally, Section VI reviews related work, and Section VII concludes the paper.

II. Path Change Validation

To clearly present DRAQ design, we assume a simple path vector routing protocol is running on atomic nodes that are connected by atomic logical links.[1] More formally, the network is represented as a directed connected graph $G = (V, E)$, and every node $v \in V$ runs a path vector routing algorithm. A path to a destination is an ordered sequence of nodes. Paths are calculated using the path vector algorithm rules: each node v stores the latest path received from all its neighbors, and selects the best path to each destination according to its routing policies. Each unidirectional link $[v\ u] \in E$ can be either up or $down$, and in the absence of false information, node v and u always agree on the status of link $[v\ u]$. Nodes adapt to changes of link status and recompute the best paths. In presenting the basic algorithm, we consider one single destination node and assume nodes use a shortest path policy in selecting best paths. with the tie-breaking rule of preferring a next hop with the lowest ID. In section V-C, we will discuss how to handle more general policies that would occur in a path vector routing protocol like BGP.

Unless specified otherwise, in this paper we assume that events (i.e., link failures and link recoveries) are *non-overlapping* in time: when one event triggers some updates, the network converges to a steady state before the next event happens. We further assume that our diagnosis algorithm starts after the network converges and finishes before any new event happens. We will discuss how to relax these assumptions in Section V-B. We assume diagnosis is typically carried out at some external boxes, called *diagnosis stations*. This external box does not modify path selection at the monitored router. For ease of presentation, we sometimes use *diagnosis node* to mean the combination of a router and its diagnosis station.

Our first objective is "path change validation", understanding *why the previous path is removed* and *why the specific new path is chosen among all existing alternatives*. For example, node Z in Figure 1 observes that neighbor H's path has changed from (H G F A) to (H G D E B A). Route diagnosis at node Z should identify the topology change that has triggered H to remove path (H G F A), *and* it should also explain why node H choose backup path (H G D E B A) over path (H G D C B A)

To solve the path change validation problem, DRAQ *accumulates the baseline topology* inferred from topological information from past updates, *uses root cause notification* to propagate the relevant changes, and *sends active queries between neighbor routers* to obtain any missing topological information if necessary. The details of each component are discussed below.

A. Accumulating Baseline Topology

Route diagnosis can benefit greatly from a complete picture of the network topology as in link-state protocols. However, most of topology information flooded by link-state protocols is *not relevant* to a route diagnosis node. A diagnosis node does not have to know the *complete* network topology; it only needs to know the partial topology that affects its neighbor's choice in path selection.

Path vector protocols implicitly provide some topology information by listing the full path used to reach the destination. In the absence of false information, every link in this path either exists in the network now, or must have existed in the network at some time. Using this path information, node Z can accumulate an estimate of the topology graph. More specifically, each node builds a network topology graph $G_T = (V_T, E_T)$ based on updates received. Each edge $[v\ u] \in E_T$ has a status of either *up* or *down*. On receiving an update with a path, each link that is not currently in E_T is added to E_T, and its status is initialized as *up*. We call the topology built solely from path vector updates as the *baseline topology*.

[1] In BGP, one AS can have multiple routers, and one AS-level link can have multiple physical links.

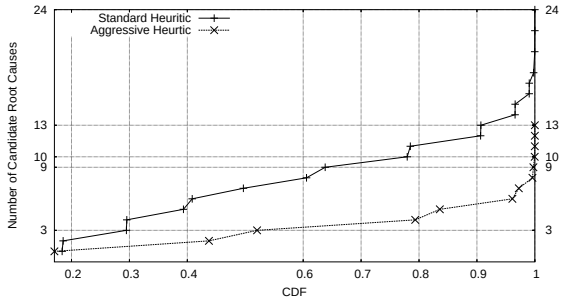


Fig. 2. Accuracy of Passive Inference: the CDF of number of candidate locations of change over all the nodes and all the simulations. We simulated the most conservative heuristic("standard Heuristic") and the most aggressive heuristic("Aggressive Heuristic"), with the risk of missing the real root cause) presented in [4]. For better visual effects, we swapped the x-axis and y-axis in all the CDF figures in this paper. The simulation settings are the same as those in Section II-D.

The baseline topology provides a starting point for route diagnosis, but it is not sufficient. The baseline topology does not necessarily reflect every link's current status. A link can be down in the actual network, but if this information is not explicitly reported in updates, the baseline topology may still consider the link up. Path change validation based on this obsolete baseline topology would be inaccurate. To enhance the baseline topology, we propose to include root cause information in each update to explicitly report the state of some *relevant and important* links.

B. Root Cause Notification

One possible way to identify topology changes is to *infer* from routing updates, but inference is inaccurate at best. Even in the simple example of Fig. 1, the failure of link $[GF]$ and $[FA]$ can both result in the same update sequence as shown. Generally speaking, in a large scale, richly connected network, a wide range of different topological changes can trigger the same set of updates, thus it is difficult to *infer* the change from the effects of the change, as evidenced by the limited accuracy and correctness of passive inference effort[1][2][3] [4]. Fig. 2 shows the results using the state of the art in [4] to infer the location of triggering change in a simulated topology. The figure shows that at best the triggering change can be narrowed to within three locations in less than 45% of the cases. Note also that existing inference approaches[1][2][3][4] only attempt to identify the cause of a path change and do *not* explain why a specific new path is selected.

Rather than attempting to infer which links have failed, we claim routing protocols should carry some essential information to facilitate diagnosis. In the case of path vector routing protocols, the essential information needed to facilitate diagnosis is provided by the Root Cause Notification (RCN) approach recently proposed in [9][10]. RCN was originally proposed to reduce the convergence time of path vector routing. By adding a RCN attribute that identifies the failed link, routers are able to avoid choosing backup paths that contains

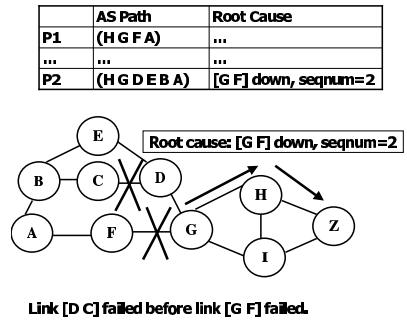


Fig. 3. Root Cause Notification.

the failed link. In effect, the convergence enhancement is attempting to identify failed backup paths. In DRAQ, we adopt the concept of root cause notification as discussed in [9][10]. However, unlike [9][10], we do not assume that routers use RCN in selecting best paths. DRAQ uses RCN solely for route diagnosis. [2]

An RCN attribute has the format $\{[u \quad v], status, seq\_num\}$, where $[u \quad v]$ indicates the root cause link, $status$ indicates whether link is *down* or *up*, and seq_num is the sequence number associated with the link. The seq_num is incremented by 1 each time this link changes its status. If link $[u \ v]$ changes state and cause a node u to modify its best path, node u will announce the new best path and list link $[u \ v]$ as the root cause for this change. If a node w changes its best path due to the receipt of an update message, node w will copy the root cause from the incoming update into w's outgoing update message. Any updates that are triggered by the same link status change will carry the same root cause information. For the purpose of route diagnosis, the root cause for each update is directly identified.

For example, in Figure 3, the RCN attribute $(\{[G \quad F], down, 2\})$ of the updates tells node Z that path $P1$ is removed because link $[G \ F]$ has failed. The root cause information carried in the updates conveys relevant topology information, and we use the root cause information to enhance the baseline topology in diagnosis nodes. The status of an existing edge in the topology is updated by the root cause information with a higher sequence number.

By combining topology information from path updates and root cause notification, a diagnosis node can build a more accurate estimate of the topology. All the information accumulated by the diagnosis node was relevant (impacted path selection) at some point in time. But note this is not equivalent to the link state approach of learning the full topology. In particular, our diagnosis cannot claim to have a complete topology picture. Nevertheless, in later simulations, we will show that this topology estimate is effective for route diagnosis.

[2] If one does choose to use RCN in best path selection at the router itself, our algorithm still works equally well.

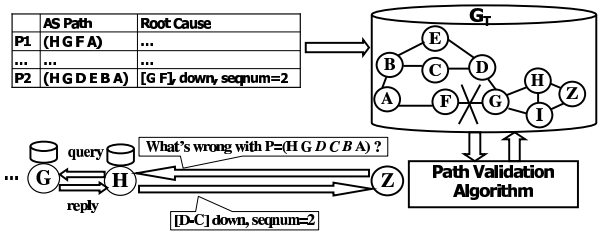


Fig. 4. Topology Accumulation and Query at node Z.

C. Path Prediction and Active Queries

With an accumulated topology enhanced by RCN, the diagnosis node will conduct path prediction after receiving a sequence of updates, and send active query if necessary. The path vector updates and the root cause information do not always provide all the relevant topology information. For example, in Fig. 1, suppose link [D C] fails before link [G F] fails. When D detects the failure of [D C], it sends the new path (D E B A) together with the root cause, $\{[D\ C], down, 2\}$, to node G. Node G stores [D C] failure information in its topology, but since its best path is still (G F A), no update is sent to neighbor H or I. As a result, nodes H, I, or Z are not aware of the failure of link [D C]. At the time of the failure, this information is not relevant to nodes H, I, or Z. However, the information becomes relevant after link [G F] fails. As shown in Fig. 4, node Z updates link [G F] status to be "down" in its topology graph, and the resulting baseline topology is shown in the right-top corner of Figure 4. In this graph, link [D C] is still considered up, even though it is actually down in the network.

Algorithm 1: Path Change Validation algorithm at a node

Data : Update Sequences with RCN at a node
Result : $validated(n)$ for each neighbor n: true or false
Update Topology using updates and RCN;
$GotNewInfo \leftarrow$ true;
while $GotNewInfo$ == *true* **do**
  $GotNewInfo \leftarrow$ false;
  foreach *neighbor* n **do**
    Compute the predicted path for n;
    if *predicted path == received path* **then**
      $validated(n) \leftarrow$ true
    else
      querying n ;
      if *Querying got new info* **then**
        $GotNewInfo \leftarrow$ true;
        add new info into Topology;
      else
        $validated(n) \leftarrow$ false;

With above inaccurate topology, Z's Dijkstra algorithm [11] would predict that the new path for H is $P = (HGDCBA)$.[3] Since the predicted path is different from the received path, either node Z has an outdated topology or the network is not behaving correctly. Node Z will send a query to neighbor H asking about the status of the links on the predicted path. If node H knows more up-to-date status, it will send a reply back to Z; otherwise, as in the case of Figure 4, it propagates the query to the next node on the path. In its query to H, node Z attaches its current sequence number of the links in question. Thus node H can tell the relative freshness of its own link status and that of Z, and H only replies with more up-to-date link status information. Furthermore, as opposed to query about every link in $P = (HGDCBA)$, node Z queries about [D C] and [C B] *only* since all other links in P appears in $P2$, implying these links are not the reason why $P2$ is not used by H.[4]

The recursive query finishes when either the query reaches the node adjacent to the failed link (node D), or reaches a node that has learned of the failure (node G). In Figure 4, node G replies with the $RC = \{[D\ C], down, 2\}$, and this reply is propagated back to Z. Both H and Z update the information in their topology graphs and node Z conduct the path prediction again. If the new predicted path matches the received path, the case in this example, then the received path from node H is *validated*. Otherwise, Z will send a new query asking what's wrong with the *new* predicted path. We show the path change validation algorithm at node Z is presented in Algorithm 1.

In the absence of false information, the querying process in Algorithm 1 will always terminate, and it always terminate with the new path "validated." This is because, as shown above, each query about a particular path will always get an answer, and eventually the topology will be accurate enough such that each predicted path is the same as the received one. This implies that if Algorithm 1 terminates with $validated(n) = false$ for some neighbor n, it is a clear indication of the existence of false information. We conclude this section by analyzing the effectiveness of DRAQ assuming there is no false information and then the false information identification algorithm discussed in Section III.

D. Accuracy and Overhead

To evaluate the performance of DRAQ's path change validation, we use SSFNET [12] to simulate BGP updates in a 110-node (with 572-links) Internet-like Topology, also from SSFNET project [13]. We injected 5720 non-overlapping random link changes. The path change validation algorithm runs at each node.

[3] lowest ID tie-breaking

[4] Above discussion assumes the predicted path is more preferred than the received path. If the predicted path is less preferred than the received path, it means that some links in the received path are marked "down" in the accumulated topology, but actually are "up" in the network. Thus, only those links in the received path that are marked "down" in the topology are needed to be queried.

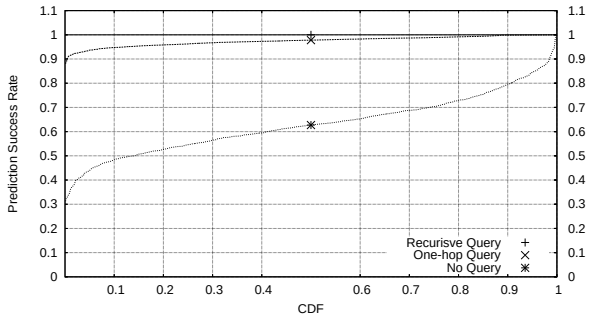
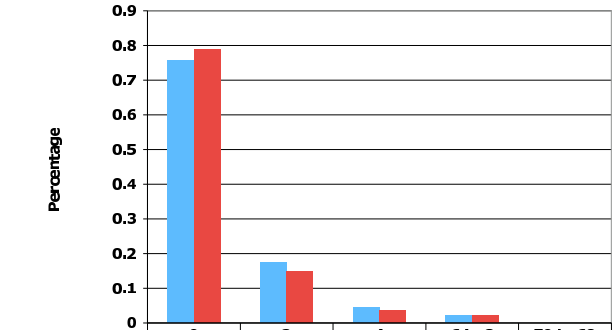


Fig. 5. Path Change Validation Accuracy



# of Messages	0	2	4	6 to 8	50 to 68
Recursive Query	0.757	0.176	0.0438	0.0229	0
One-hop Query	0.789	0.148	0.0362	0.0226	0.000054

Fig. 6. Querying Overhead for Path Change Validation

For each event observed at a diagnosis node Z, the algorithm should terminate with $validated(n) = true$ for each neighbor n (since no false information was injected). We say Z's path change validation is a success if Algorithm 1 returns true for all n; otherwise, it's a failure. We define node Z's accuracy of path change validation as the success rate of all the events it observed. Figure 5 shows the CDF of accuracy over all nodes in the network, and it shows that DRAQ's accuracy of path change validation is 100%, confirming our earlier analysis.

We also measure how many query messages in total are triggered by node Z for this event. Note that by "message", we mean a message between two neighbor routers, either a query or a reply. For example, in Figure 4, Z triggered 4 messages, one query from Z to H, one query from H to G, and two corresponding replies. Figure 6 shows the PDF of query message overhead of all $(Z, event)$ cases. The overhead is very low: about 93% of events needs at most 1 query and 1 reply, and about 75% of events needs no query at all.

To show how much active query helps path change validation, we compared DRAQ with the approach where no query is used, and the approach with one-hop query only (i.e., the neighbor will not propagate the query if it does not know the answer). Figure 5 shows that for the no-query approach, half of nodes have higher than 60% accuracy. This shows that, although not as accurate as in link-state protocol, the topology accumulated from updates and root cause information is fairly adequate for route diagnosis. For the one-hop query, each node has higher than 88% accuracy, which demonstrates that active query, even just one-hop, can dramatically increase the path change validation success rate.

III. Attack Detection and Attacker Identification

In the previous section, we saw that a combination of accumulating topology, adding root cause notification, and sending queries when necessary, provides us with effective diagnosis *in the absence of false information*. In this section, we consider how to *detect false information* and *identify the source* of the false information. False routing information may be injected due software faults, unintentional mis-configurations, or malicious attackers who have compromised routers. In addition to injecting false information, a misbehaving router may simply choose not to forward required information. In this paper, we use the term "attacker" to mean any misbehaving router and we use "attack" to mean the action of injecting false information or blocking the propagation of genuine information.

Our objective is detect the following two types of attacks and identify the source of this misbehavior:

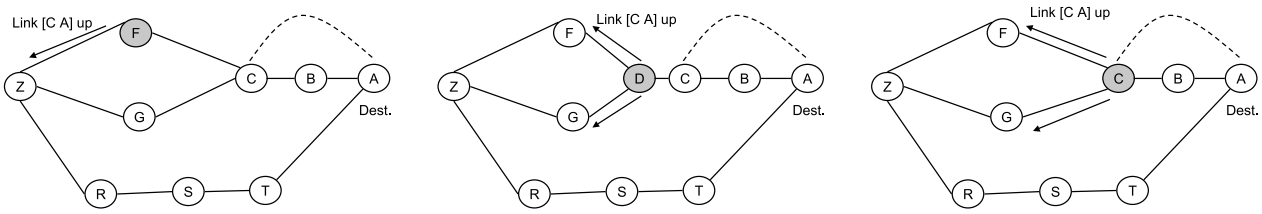
1) Insertion Attack: The attacking router announces a path that contains a false link and/or false RCN message. More precisely the announced state of some link in the path and/or the state of RCN is the opposite of the actual state known by the router. As a result of such an attack, some of the nodes in the network receive path vector updates triggered by the attacker, and these updates all carry the false information.
2) Blocking Attack: A legitimate root cause triggers updates in the network, but the attacker simply withholds any new path announcements. For example, the attacker fails to withdraw a path even though some link in the path has failed.

As a first step we simplify the attack scenario by restricting the number of attackers at any given time to at most one. We further restrict the attacker to inserting false information on the existence or status of a single link. Even with a single attacker and single false information, attack detection and attacker identification is non-trivial and presents many challenges. Directions to deal with multiple attackers are discussed in Section V-E.

Finally, we distinguish between *third-party attacks* and *direct attacks*. In a third-party attack, the disputed link is not directly adjacent to the attacker. In a direct attack, the disputed link is adjacent to the attacker. In Figure 7(a), a third-party attack occurs when F inserts a false link, [C A], by announcing the path (F C A) with root cause$\{[CA], up, 1\}$. In Figure 7(c), a direct attack occurs when node C inserts a false link [C A].

A. Overview of Our Approach

Our approach has two mechanisms for attack detection. First, we note the Path Change Validation from the previous



(a) Third-party Attack: $validated(G)$=false (b) Third-party Attack: no validation failure (c) Direct Attack: no validation failure

Fig. 7. Third-Party Attack and Direct Attack. The shaded node is an attacker who inject a false link shown in dashed line.

section will finish with 100% rate in the absence of attacks. Therefore, a failure of path change validation indicates an attack has occurred. We later show that, in a richly connected graph, failure of the path change validation is an effective step in detecting third-party attacks. But not all attacks will result in a failure of the path change validation algorithm. In particular, path change validation does not detect any direct attacks. In a direct attack where node v announces link [v u], only node u can detect the false information, but path vector routing does not signal this information to u. To resolve uncertainties such as those that result from a direct attack, we send active queries to neighboring nodes to solicit information about link status. Once the *existence* of attack is detected, a majority vote algorithm identifies *independent* nodes and pins down the attacker.

B. Detection: Staring from Algorithm 1

After the Path Change Validation algorithm (Algorithm 1) is done, the diagnosis node runs our attack detection algorithm, shown in Algorithm 2. The detection algorithm first checks whether any path has been marked invalid(i.e., $validated(n)$ is false for neighbor n's path) by Algorithm 1. Consider Fig. 7(a), where an attacker F fakes a false link [C A] by announcing the false path (F C A) with root cause$\{[CA], up, 1\}$. F cannot influence node G's path, and G still reports path (G C B A). Upon receiving F's false path and root cause, node Z first temporarily adds root cause $\{[CA], up, 1\}$ into its topology, then runs path change validation algorithm (Algorithm 1) to compute the predicted paths for all its neighbors. The predicted path for node G is (G C A), different from the received path (G C B A). As part of Algorithm 1, Z sends a query to G, asking what's wrong with path (G C A). Neither G nor C has information that can explain why (G C A) is not used by G, thus Algorithm1 at Z terminates with $validated(G)$=false, $validated(F)$=true, and $validated(R)$=true. Since at least one neighbor failed to validate, Z has detected an attack.

In general, a node detects an attack anytime the path from one (or more) neighbors is not validated. When one third-party node x lies about the status of link [v u], another third-party y might not agree. An attack is detected provided that the diagnosis node receives a path from such a node y. In topologies with reasonable connectivity, it is likely that there exists at least one such node y. But no such y exists in the case of a direct attack.

Algorithm 2: Attack Detection Algorithm

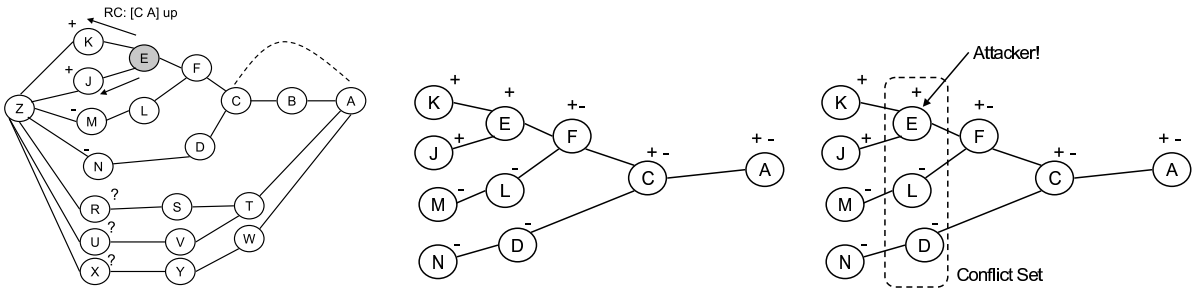
```
Data   : validated(n) result for each neighbor n
         from From Algorithm 1. RCN link [v u]
Result : attack_detected: true or false
if validated(n)==false for any neighbor n then
    attack_detected ← true;
    Call Algorithm 3;
else
    attack_detected ← false ;
    if link [v u] ∉ G_T or is timed out then
        if has a neighbor n' whose path includes u but
        does not include v then
            query this neighbor n';
            if u's reply is the opposite to RCN then
                attack_detected ← true;
                Call Algorithm 3;
```

C. Detection: Adding Queries

In a direct attack, the disputed link is adjacent to the attacker as shown in Figure 7(c). and the path change validation algorithm cannot detect any inconsistency. Under our a single destination prefix model [5], a unidirectional link [v u] can be announced only by v. Node u does not signal anything about whether link [v u] is genuinely "up" or "down" and thus there is no inconsistency to detect.[6] Fundamentally, verifying the state of link [v u] requires get some information from node u. In addition to direct attacks, some third-party

[5] We discuss the potential advantage of a multiple-prefix model in Section V-D.

[6] Under our single-prefix model, it does not help to try to infer [v u] from [u v]'s status either. At most one of v or u announces either link [v u] or [u v] in a new best path, thus when v is announcing link $[v\ u]$, then node u must not be announcing [v u] or [u v].



(a) Attack (b) Conflict Tree at Z (c) Conflict Set at Z

Fig. 8. Identifying Attacker

attacks can pose the same problem. For example, attacker node D in Figure 7(b) announces a false link [C A], and both neighbors of Z will use link [C A]. The paths received at Z, (F D C A) and (G D C A), are consistent and match the prediction.

Unlike other BGP security work[5][6][7][8] that require node u to *pro-actively* propagate information about unidirectional link [v u]'s status, our approach attempts to detect attacks while limiting the amount of link status propagation. Given the output of Path Change Validation algorithm, node Z can determine what information is relevant and then verify only that information. For example, node Z in Figure 7(c) can determine that the potential false information is the state of link [C A]. Information for other links such as [F C] is not needed. We use a query mechanism to *reactively* obtain the "needed" information regarding link [C A] from node A. Of course, our query must avoid the potential attacker and neighbors who rely on the attacker. To do this, node Z identifies neighbors whose path include A but not C and then sends the query to one of these neighbors. For example, in Figure 7(c), node Z identifies node R as such a neighbor and sends a query inquiring about link [C A]. Node R propagates the query along its current path (R S T A) to A. When A receives this query, it replies indicating link [C A] does not exist, and the reply is propagated back to node Z, allowing node Z to detect the attack. The query mechanism and overall detection algorithm are summarized in Algorithm 2.

There are two elements of the algorithm that merit further discussion. First, note that if the Path Change Validation algorithm has already detected $validated(n) = false$ for some neighbor n, a query is not triggered by the *detection* algorithm. In this case, an attack has been detected and we proceed directly to the problem of the attacker identification. Second, note that if a link genuinely keeps going up and down, the query overhead to verify this link could be high. To counter this, we use a simple heuristic: a query is sent only when a link appears in the topology for the *first* time. As a result, the query overhead is related to how frequently a new *physical* link appears over time.To flush out stale information, we set a timer for each link and a link is "timed-out" if it has been "down" or has not been updated before the timer expires and is then treated as a new link if it reappears in a path update.

D. Attacker Identification: majority vote among independent nodes

Once Algorithm 2 detects the existence of an attack, the next step is to identify the attacker as precisely as possible. Our identification algorithm is motivated by the assumption that there are relatively few attackers and large numbers of correct nodes in the network.

In the event of the detection of an attack, one straightforward approach is to use a majority vote among conflicting neighbors. We label neighbor n as "+" if $validated(n)$ is true and "-" if $validated(n)$ is false. In other words, those nodes with "+" agree with the topology information in the root cause, and those with "-" do not. In addition, if a neighbor n's $validated(n)$ is true, but its predicted path is not affected by the new root cause information, a "?" mark is assigned. For example, node Z in Fig. 8(a) receives updates that were triggered by the attack by node E. Nodes K and J are assigned "+", nodes M and N are assigned "-", and node R, U and X are assigned "?" since they are not affected by the attack. The resulting straightforward majority vote among node Z's conflicting neighbors results in a draw. But a closer look at the Figure 8(a) shows that conducting a simple neighbor count is unfair because node K and J are both influenced by node E. That is, K and J are not independent with respect to E. For a more accurate majority vote, we actually seek a majority from *independent* nodes instead of a simple majority.

To determine independence, we combine the paths from all the neighbors marked either "+" or "-'" and obtain a shortest path tree rooted at the destination. We call this tree a "conflict tree" (shown in Figure 8(b)). Leaves of this tree are neighbors of the diagnosis node and are labeled based on whether $validated(n)$ is true or false. A non-leaf node that has descendants with both "+" and "-" is labeled with "+-", indicating its descendants cannot agree on whether the root cause information is true or not. In Figure 8(b), node F, C

and A are labeled with "+-". All other nodes in the network, if independent of the attacker, should all agree with each other and disagree with the attacker. Therefore, the attacker must be the descendant of those nodes with '+-" marks.

More formally, we construct a conflict set S that contains the attacker and other independent nodes. S is initialized to be empty. From each leaf node of conflict tree(Fig. 8(b)), we traverse up the tree until reaching the first node x that satisfies the following two conditions:

1) x is marked with either "+" or "-"
2) its parent node is marked with "+-" mark

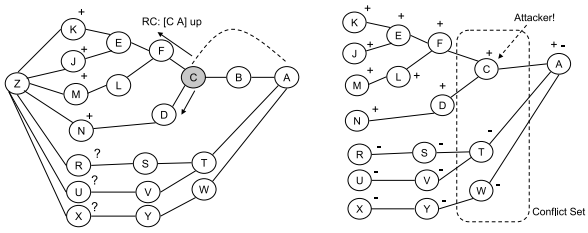
Node conflicts with at least one other node and thus x is added to S, but none of node x's descendants are added to S since they are influenced by x. We repeat this procedure for each leaf node of conflict tree to obtain S. Nodes in the conflict set S are *independent* of each other and we now can do a majority vote among nodes in S. For example, the node E, L, and D in Figure 8(b) are all added to S. The mark with the least votes loses, and the node with losing mark is identified as the attacker. In our example, mark "+" loses, and node E is identified as the attacker and the attack is an insertion attack of link [C A].

E. Attacker Identification: Query

The above identification algorithm did not make use of query messages. In Figure 7(b) and Figure 7(c), the above algorithm results in $S = \{D^+\}$ and $S = \{C^+\}$, respectively. Both conflict sets only have a size of 1 unless query results are added into the algorithm.In general, when we query node u about a link [v u] through a neighbor n, there are several nodes in the path from n to u. We only add the node adjacent to u into the conflict set, since replies of all the other nodes on the same path are not independent. In the case of Figure 7(b), the conflict becomes $S = \{D^+, T^-\}$, and in the case of Figure 7(c), $S = \{C^+, T^-\}$.

Once the identification algorithm is called by Algorithm 2, and the conflict set after query result is incorporated, the conflict set size is at least 2. Given we assume there is only one attacker, the attacker can be directly identified if the conflict size is larger than 2. If the size is 2, we clearly cannot rely on majority vote. At this point, the diagnosis node may choose to send additional queries through other neighbors in an attempt to further pin down the attacker. Again, we need to find the *independent* nodes to query. In Figure 9(a), node Z has queried node R and obtained the conflict set $S = \{C^+, T^-\}$. A query to neighbors U would go through T and thus wouldn't contribute to the conflict set. But a query neighbor X would produce the conflict set $S = \{C^+, T^-, W^-\}$, and identify node C as the attacker, as shown in Figure 9(b).

After including queries, our attacker identification algorithm is summarized in Algorithm 2. Finally, note that our attacker identification algorithm deals with the case that the link is genuine but query results are false(i.e., an attacker fake a reply for a query). Given there is only one attacker, this case is naturally solved by majority vote among *independent* nodes in our identification algorithm.



(a) Attack (b) Conflict Set at Z

Fig. 9. Attacker Identification with Query

Algorithm 3: Attacker Identification Algorithm

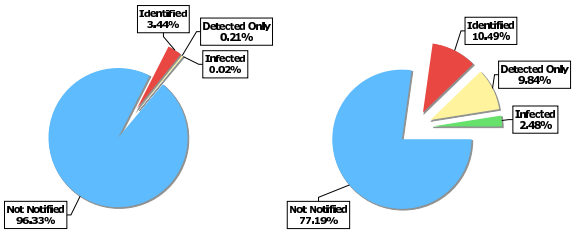
Data : From algorithm 2. root cause link $[v\ u]$;
Result : $attacker\_identified$: true or false
$attacker\_identified \leftarrow$ null;
repeat
  construct conflict set;
  if *size of conflict set >2* **then**
    $attacker\_identified \leftarrow$ true ;
  if *has a neighbor n whose path does not include v, but include u; not queried yet; and independent of other querying results* **then**
    query this neighbor n;
  else
    $attacker\_identified \leftarrow$ false;
until $attacker\_identified \neq$ *null*;

IV. Evaluation of Attack Detection and Identification

In this section, we evaluate our attack detection and identification algorithms using simulations. We simulate path-vector protocol and our algorithm in an Internet-like topology from [13] with 409 nodes. Throughout this section, we will use the following notations: A for the origin node(destination), X for the attacker, and Z for the diagnosis node. In a third-party attack, the attacker announces a false link between its current next hop and the origin; in a direct attack, the attacker announces a false link between itself and the origin. Such attack patterns are to approximate maximal damage that an attacker can do to a specific origin. Note that in each simulation run, the attacker attacks only *one* origin at a time.

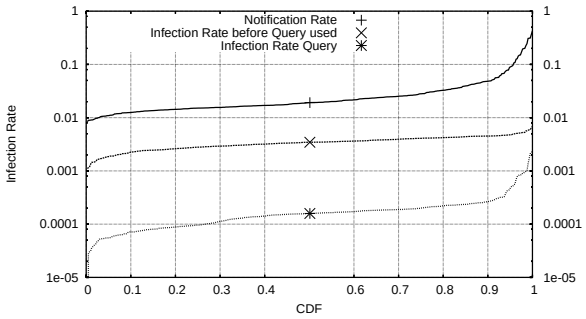
A. Overall Results

Given A and X, Z may or may not be affected by the attack depending on their locations in the network. If Z receives updates triggered by the attack, we say Z is *notified*

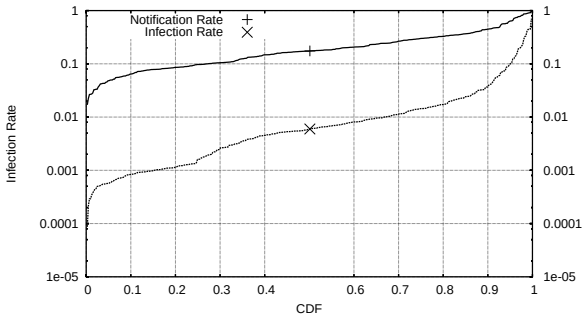


(a) Third-party Attack (b) Direct Attack

Fig. 10. Overall Results



(a) Third-Party Attack



(b) Direct Attack

Fig. 11. Notification Rate and Infection Rate over All origins.

by the attack (A, X); otherwise Z is *not notified*. For our attack pattern, Z is notified if X is on the old path or new path(to A) of at least one of Z's neighbors.

An attack has impacts only on notified nodes. For a notified Z, if it can both detect the attack and identify the attacker, we say the attacker X is *identified* by Z. According to our algorithm, identification is only possible when the conflict set size is at least 3. This means node Z must receive paths from at least 3 *independent* information sources regarding the disputed link, and node Z's degree must be at least 3.

If a notified Z can detect the attack but cannot identify the attacker, we say the attack (A, X) is only *detected* by Z. A node Z can detect the (A, X) if the new path of at least one of Z's neighbors does not go through X to reach to A. If a notified Z cannot even detect the attack, we say node Z is *infected* by attack (A, X). A node Z is infected by (A, X) if *all* Z's neighbors' new paths go through X.

Since the locations of A, X, and Z have great impact on the effectiveness of diagnosis, we enumerated all possible combinations of (A, X, Z) in the 409-node topology. Fig. 10 shows the overall results.

We can see that the overall notification rate is low. Among all combinations of (A, X, Z), Z is notified in only 3.67% of the third-party attack cases, and 22.81% of the direct attack cases. Since our detection algorithm does not block any update, it doesn't affect the notification rate at all. Therefore, the relatively small notification rate reflects the network's intrinsic immunity against attacks even without any detection mechanism. This is mainly due to the Internet topology's rich connectivity, and the path vector routing protocol, in which a node only sends its best path to neighbors.

Without a detection mechanism, the infection rate would be equal to the notification rate. The difference between the notification rate and the actual infection rate is how much the system benefits from adopting a detection mechanism. Fig. 10 shows that our algorithm can reduce the infection rate to 0.02% for third-party attack, and 2.46% for direct attack, less than 10% of the notification rate in both cases.

We now examine more detailed results from the origin AS' point of view and the diagnosis node's point of view respectively.

B. Results for Origin ASes

Given an origin node A, we measure the notification rate $n(A)$, which is the ratio of the number of cases that Z is notified over the number of all possible (X, Z) cases. Similarly, we measure the infection rate $i(A)$, which is the ratio of the number of cases that Z is infected over the number of all possible (X, Z) cases. Fig. 11 shows $n(A)$ and $i(A)$ as CDF over all possible A.

For third-party attack (Fig. 11(a)), our detection algorithm reduces the infection rate by two orders of magnitude compared with the notification rate. For instance, the median infection rate(Y value when X=0.5 in our CDF Fig. 11(a)) is 0.0159%, while the median notification rate is 1.92%. If we only use path change validation algorithm without active queries, the infection rate (e.g., the median is 0.345%) is still about one order of magnitude smaller than the notification rate. For direct attack, our detection algorithm also reduces the infection rate by one to two orders of magnitude (Fig. 11(b)). For instance, the median infection rate is 0.597%, while the notification rate is 17.5%.

Fig 11 also shows that both the notification rate and the infection rate actually vary a lot for different origin ASes. Suppose an origin A has t direct neighbors. The effect of a third-party attack or direct attack is to add one fake neighbor. Intuitively, an attack can affect about $\frac{1}{t+1}$ fraction of

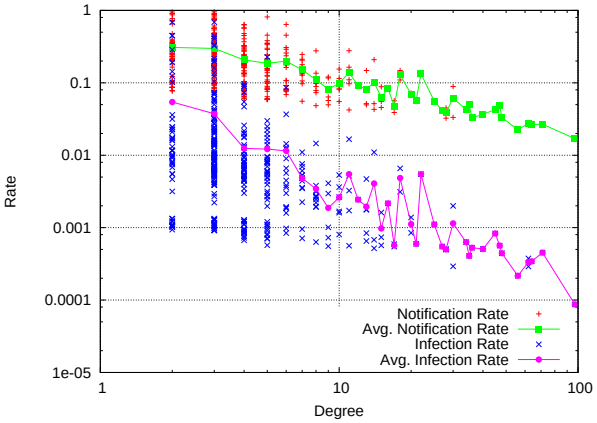


Fig. 12. Notification Rate and Infection Rate vs. Origin's Degree for Direct Attack. The result for third-party attacks has the same trend.

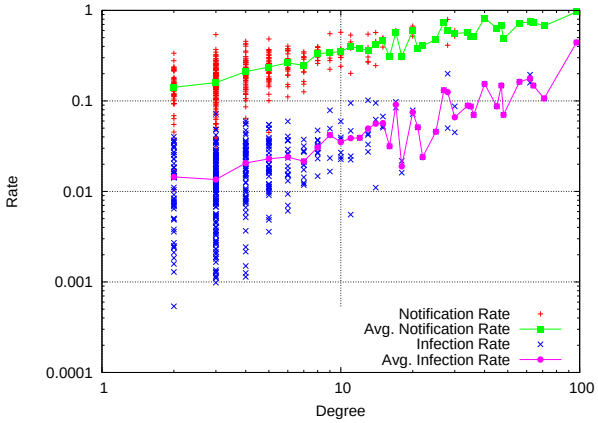


Fig. 13. Notification Rate and Infection Rate vs. Attacker's Degree for Direct Attack. The result for third-party attacks has the same trend.

all nodes on average, that is, the average notification rate is around $\frac{1}{t+1}$. Therefore, the higher an origin's degree, the more immune to an attack this origin is, even without any detection mechanism. With our detection algorithm, for a given (X, Z), the higher an origin's degree, the more chance that at least one of Z's neighbors does not take X's path which enables Z to query the origin and detect the attack.

Fig. 13 shows the notification rate and infection rate versus origin's degree for direct attacks. The result for third-party attacks has the same trend. As analyzed above, both notification rate and infection rate decrease as origin degree increases, and the infection rate decreases sharper than the notification (a node not notified will never be infected according to the definitions). The implication is that a well-connected node is better protected by both path-vector itself and our detection algorithm.

Note that at the top-right corner of Fig. 11(b) the notification and infection rates for some origins are quite high. In addition, the scatter plots in Fig. 12 also show that origin's degree is not the only factor, since origins with the same degree can have different notification rate. Our investigation shows that those cases are mainly caused by the combination of origin's low degree and being close to high-degree attackers. Fig. 13 shows the notification rate and infection rate versus attacker's degree for direct attacks. The result for third-party attacks has the same trend. It is clear that as the attacker's degree increases, its damage to the network increases too. The higher a node's degree, the more likely that other nodes will rely on this node to reach the origin. If such a high-degree node is compromised, it is likely to have bigger negative impact on the system.

C. Results for Diagnosis Nodes

We now show the results from the diagnosis node's point of view. Given a diagnosis node Z, if its notification rate is $n(Z)$, and infection rate is $i(Z)$, then its *relative infection rate* is $ri(Z) = \frac{i(Z)}{n(Z)}$. Similarly, if its identification rate is $d(Z)$, its *relative identification rate* is $rd(Z) = \frac{d(Z)}{n(Z)}$.

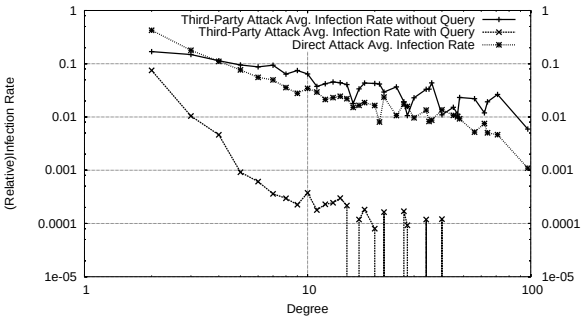


Fig. 14. Relative Infection Rate vs. Diagnosis Node's Degree

Since both our detection algorithm and identification algorithm benefit from high-degree diagnosis nodes, we draw the average $ri(Z)$ as a function of Z's degree in Fig. 14. We see that in general infection rate decreases as the degree of diagnosis node increases. When node degree is 10 or higher, the average relative infection rate is no more than 0.0375% for third-party attack, and no more than 3.45% for direct attack. This shows that a well-connected diagnosis node can be extremely effective in detecting attacks (e.g., more than 99.9% in third-party attack, and more than 96.5% in direct attack).

Fig. 15 shows the relative identification rate as CDF over

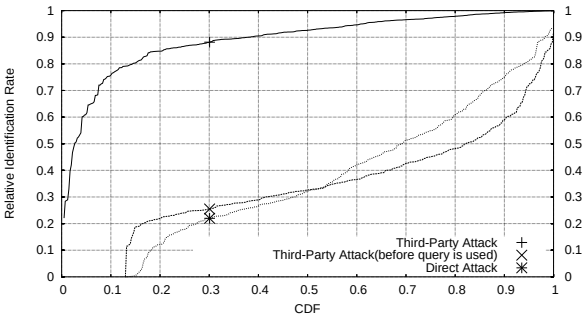


Fig. 15. Relative Identification Rate at Diagnosis Nodes

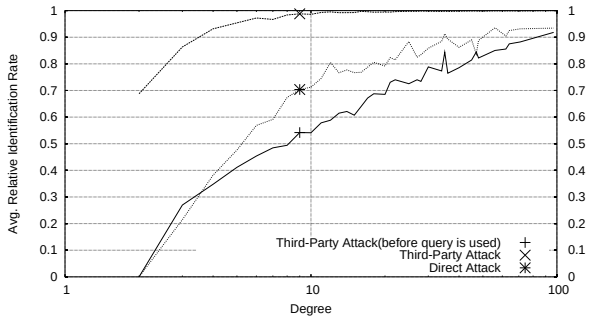


Fig. 16. Relative Identification Rate vs. Diagnosis Node's Degree

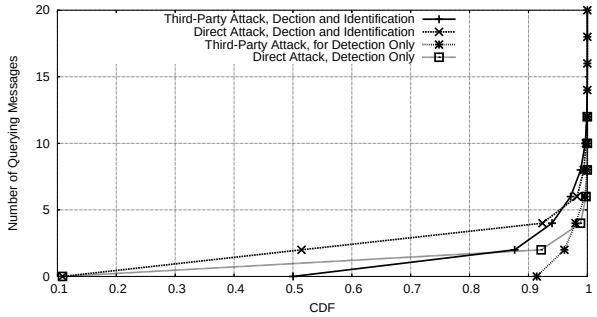


Fig. 17. Querying Overhead for Attack Detection and Identification.

all diagnosis nodes. For third-party attack, our identification algorithm achieves high median $rd(Z)$ of 93%. For direct attack, the median $rd(Z)$ is around 30%. According to our algorithm, identification is only possible when the conflict set size is at least 3, which means node Z's degree must be at least 3. However, in our topology, there are 48% of nodes with degree of 2 or 3. Therefore, it is no surprise to have lower relative identification rate for direct attacks.

Fig. 16 shows the average relative identification rate versus diagnosis node's degree. When Z's degree is 10 or higher, average $rd(Z)$ is more than 98% for third-party attack, and more than 70% for direct attack. Also, when Z's degree is 10 or higher, using only the path change validation algorithm without active query can achieve average $rd(Z)$ of more than 50% for third-party attack. This demonstrates that the query mechanism significantly improves the identification of attackers.

D. Querying Overhead

Fig 17 shows the CDF of the query mechanism's message overhead. With our optimization, the message overhead is very low. In 90% of all cases, the query overhead is at most 2 messages (one query and one reply) for detection *only*. In 90% of all cases, the query overhead is at most 4 messages for both detection and identification.

V. Discussions and Future Work

In this section, we discuss our future work and some issues not covered by previous sections.

A. Decoupling Diagnosis Function from Router

DRAQ does not modify best-path selection algorithm run by routers. The only required change to the routing process is that routers attach and propagate the root cause in the updates to facilitate the diagnosis. All the other diagnosis functionalities are conducted in an auxiliary device, called *diagnosis station*, without sending feedback to routers.

To conduct route diagnosis, the diagnosis station receives a copy of each update the corresponding router receives, and then runs the diagnosis algorithm. The topology graph is stored inside the diagnosis station using inexpensive storage device (such as a hard disk). Necessary queries are sent and propagated among diagnosis stations.

The output of route diagnosis is used by network operators. When false information is detected, an alarm is raised, and identified attackers are provided to the operators. Compared with alternative approaches [5][6][7][8] which provide feedback to the routing process to actively block the detected false information, DRAQ does not require timely detection decision, adds little CPU overhead to th routers, and no feedback loops are created that could make diagnosis more difficult.

B. Overlapping Events

We now discuss our assumption in Section II that our diagnosis algorithm starts after the network converges and finishes before any new event happens.

To start running the diagnosis algorithm, the diagnosis node first needs to know whether the current event has finished. To determine whether all the updates in an event have been received, the diagnosis node can use the root cause information and some timing clustering algorithm as used in [1][2][4]. A node determines that the current event converges if no update is received within a time interval $threshold$ after the last message with the same root cause. As long as no new event occurs when our diagnosis algorithm is still in progress, our algorithm works as expected. In case the timing heuristic prematurely declares the current event has converged (e.g., because $threshold$ is too small), some updates with the same root causes will be received by the diagnosis node, thus the diagnosis algorithm needs to start over again.

In the case that a *new* event happens before the current diagnosis completes, the diagnosis node will receive updates with a new root cause. Thus, the diagnosis station declares the two events are too close to each other, and the current diagnosis process is canceled and then starts over when the new event converges. Similarly, in the case that events overlap, the diagnosis algorithm runs after all the overlapping events converge. In these cases, the diagnosis algorithm needs to deal with two or more root cause changes. The path change validation algorithm and the attack detection algorithm still work, since the path prediction and query are not affected. However, the attacker identification algorithm needs to be generalized to deal with multiple root causes. One way to do that might be to build the conflict tree and

conflict set for *each* root cause, and identify the attacker individually for each root cause. We plan to investigate the generalization of attacker identification algorithm as part of future work.

C. Path Prediction with More General Policy

So far we have assumed shortest-path policy in presenting DRAQ. However, some components of DRAQ are independent of the policy used, and some others can be generalized to apply to more general routing policies. First, the root cause notification is per-prefix based, thus not affected by policy. The querying mechanism is not affected by policy either. Second, our topology graph is per-prefix based and is built based on received updates. In a path-vector protocol with more general policies, a link in the received paths regarding one prefix are allowed by the policy, thus links accumulated in this prefix's topology are allowed by the policy too.

Both path change validation and attack detection/identification use the path prediction algorithm, and so far we have used shortest-path algorithm on the accumulated topology graph to do the prediction. Similarly, prediction can be done if the network policy is "known" to the diagnosis node. For example, currently the most common policy in the Internet is the so-called "no-valley" policy based on the relationship between neighboring ASes, where a path learned from a peer or a provider can be sent only to customers, a path learned from a customer can be sent to any neighbors, and a path from customer is always preferred to a path from a peer. Under such a policy, we can first run the AS relationship inference algorithm, such as those in [14], then the path prediction is done using the policy rule. As long as a diagnosis node knows the policy (e.g. no-valley policy), and knows the input to the policy (e.g., AS relationship), path prediction can still be done accurately. In the case that policy and/or its input is not known to diagnosis node or is inaccurate, the diagnosis node can keep a list of past paths ever used by each neighbor, and calculate their relative preference based on history and some partial policy information, and use this ordered list of paths to do the path change validation. We plan to generalize our DRAQ along above directions to deal with general policies as part of our future work.

D. Taking Advantage of Multiple Destination Topology

In this paper, we build topology on a per-destination base. However, in reality, there are lots of destinations, and the failures and recoveries of one physical link can simultaneously affect paths to multiple destinations. Also, we have assumed a link is unidirectional, but it's often the case that link [u v] and link [v u] fail and recover at the same time. Correlating diagnosis results for different destinations can potentially increase our diagnosis power, but there are some issues, especially with a general routing policy. For example, a link legitimate to one destination might be illegitimate to another destination due to policy reason.

To take advantage of multiple destinations, we can build a topology graph combining all the per-destination topologies, while maintaining the individual topologies as before. With this combined topology and the policy information that we described in Section V-C, we can do some path prediction for one destination based on path changes to another destination. With this cross-checking among different destinations, we can achieve more diagnosis power. One simple example is, if an attacker fakes a link for one destination, but the combined topology and the policy say such a link, if genuine, should be used for another destination, then some inconsistency is detected. Also, our querying mechanism can be extended to query about multiple destinations in the same message. Furthermore, the attack detection and attacker identification results of different diagnosis nodes can be correlated. For example, a node that is identified as an attacker with multiple destinations is more suspicious than others. For the root cause notification mechanism, currently it is per-destination based, and it can be enhanced to carry some policy information, specifying the link status change is applicable to one destination only, a list of destinations, or all the destinations. We plan to investigate these directions as our future work.

E. Dealing with Multiple Attackers

We have assumed that there is at most one attacker in the network at any given time. In case there are multiple *isolated* attackers in the network at the same time, the problem is equivalent to dealing with overlapping events, which is discussed in Section V-B. We now discuss how to deal with multiple *colluding* attackers at the same time.

Our attack detection and attacker identification algorithms take advantage of the rich connectivity of the network, and can detect and identify the attacker with high probability if there is only one attacker. If there are two or more colluding attackers, as long as there are less attackers than normal nodes, the same detection and identification algorithm in general can still work. However, the detection rate and identification rate might be decreased, especially for those low-degree nodes close to the attackers, since query replies and majority votes might be dominated by the false information from colluding attackers.

To deal with colluding attackers, we propose that neighboring diagnosis stations share the detection and identification results. In the current DRAQ design, only the pure topology information is queried and replied among neighbor diagnosis nodes. To share the detection and diagnosis results, each node keeps two sets of attack detection and attacker identification results, one from its own diagnosis algorithm, and one learned from neighbors. An enhanced detection algorithm can then be done based on the *learned* detection and identification results. Even if one might want to cut overhead by exchanging only the local results between direct neighbors, a node can still benefit from neighbor's combined de-

grees and redundancy. Therefore, with reasonably rich network connectivity, such a collaboration among neighbors can be an effective defense against colluding attackers. We plan to investigate this direction in the future.

VI. Related Work

While there is a lot of existing work related to identification of the root cause change and false information detection, there is little work targeting at false information identification and we are not aware of any work targeting at understanding why a specific path is chosen among alternative path.

A. Related Work for Root Cause Identification

Root cause identification of BGP updates has received a lot of research attention lately. [1],[2],[3], and [4] infer the root cause solely based on current BGP messages. As reviewed in [4], there are three dimensions of update information for root cause inference: time, different peers, and prefixes, and these approaches use different combinations and different orders of these three dimensions. However, the insufficient information for diagnosis has limited effectiveness of passive inference [15][16]. Furthermore, [4][15][16] have shown that passive inference could even reach the wrong conclusion in terms of the root cause location. As [15][16] have argued, we believe that some network support is needed for an effective solution to root cause identification problem, and we have proposed root cause notification as the solution for the root cause identification problem. In terms of utilizing *path* information in the updates, these approaches only use two snapshots of the paths in the routing table at the steady state to infer the change, while we utilize much more historical information by accumulating a topology graph, although we haven't deeply explored the possibility of taking advantage of dimension of multiple prefixes.

[15][16] propose some network support for the root cause identification problem by first maintaining local root cause at dedicated servers and then using queries between servers. Both approaches focus on how to maintain the detailed local root cause such as IGP distance changes and session failures, while we focus on Inter-AS level root cause, therefore our approach complements these two approaches and vice versa. For root cause identification problem, we use RCN to signal the root cause, while they use query among servers between neighboring ASes to pin-down the root cause. These two approaches did not deal with understanding the choice of new path.

B. Related Work for Attack Detection and Identification

One of the first routing security work is by Perlman [17]. [17] presents a secure link state protocol utilizing public key cryptography for authentication of link state information. Some theoretical results on Byzantine-robustness of this secure link state protocol are provided.

In IRV [18] approach, each AS designates a server that answers queries regarding BGP security. Queries and replies are sent out-of-band of BGP, thus the authenticity of reply remains a challenging problem. Our query aims to solicit only relevant information along a chosen path, and our majority vote approach can naturally deal with false reply if any. IRV does not provide any attacker identification functionality.

The rest of work reviewed in this section, including [5][6][7][8], all use cryptography, and require the signing and verifying the path in the router, which increasing router's CPU overhead. DRAQ does not use any cryptography, and it requires little CPU overhead on router since it only needs routers to generate and propagate the root cause. All other things, including topology accumulation, querying, attack detection and identification can be done in separate devices(diagnosis stations). Furthermore, these approaches to some extent all depend on some out-of-band knowledge such as PKI, PGP-like web of trust, or globally known hash function, which can be important deployment hurdle. In addition, they focus on *blocking* invalid *updates*, and the attacker identification has not been the particular design goal of existing approaches, although potentially identification can be done based on the some approaches such as S-BGP. Furthermore, DRAQ approach handles the Blocking Attack where an attacker blocks genuine updates from being propagated, while these approaches don't. Note that our DRAQ approach can be used together with these BGP security approaches and when they fails to *protect* BGP path, DRAQ can still provide another fence of defense.

S-BGP (S-BGP)[5] provides a comprehensive BGP security solution using public key cryptography. A route attestation(RA) is signed by AS_x specifies that AS_x authorized AS_{x+1} to advertise the path of $(AS_x, AS_{x-1}, \ldots, AS_0)$. The recipient AS uses the public key of the router's along the path to verify the each link in the AS path. It requires significant router CPU overhead to sign and verify the signatures, and building and maintaining PKI required by S-BGP is difficult and this further delays the S-BGP's deployment.

The Secure Path Vector (SPV) protocol prevents attackers from truncating an ASPATH or changing any AS number in the ASPATH. In SPV, the owner of a prefix generates a sequence of one-time signatures and pass them with the update to other ASes. As the update propagates, each AS along the path uses one one-time signature to sign itself into the ASPATH. Hash trees are used to authenticate and verify the signatures, and one-way hash chains are used to reduce the size of cryptographic information. In most of its operations, SPV uses symmetric key cryptography instead of asymmetric key cryptography as in S-BGP, which makes it more computationally efficient. Nevertheless, SPV still requires some use of asymmetric cryptography and certificate hierarchy in order to authenticate the ownership of prefixes and verify the ASPATH. SPV also doesn't consider attacker identification.

In SoBGP approach [6], each AS use a special BGP message to propagate a list of its AS neighbors. Routers can use this information to build a directed graph that constructs a su-

perset of possible AS level links and then check any routing update against this superset to see whether a link is possible. However, even if each link in the update exists in the directed graph, the update could still be invalid since this link might not be available for the specific prefix in the update due to policy reason, or because currently the link is down. Our topology is built for each particular prefix based on updates, root cause, and query results, and is more accurate. Furthermore, SoBGP does not support attacker identification.

In "Whisper" approach, each node v uses symmetric cryptography to include some secret $s(v)$ into the BGP updates. At a receiving each node, the $s(v)$ learned from different peers should be the same, otherwise, an alarm is raised. This approach shares the similarity with DRAQ in that both approaches utilize the network redundancy and use some form of consistency checking. Whisper's attacker identification relies on the condition that the attacker simultaneously announces routes to multiple destinations so that penalty values of attackers becomes higher and thus identified. When an attacker only attacks one destination, Whisper cannot identify the attacker or block the invalid update from propagating, while DRAQ does not have this problem since it deals with different prefixes separately.

VII. Conclusion

Internet route diagnosis is of great need in practice. However, as with all protocol designs, the importance of route diagnosis is truly appreciated only *after* the protocol has been developed and deployed. Current BGP routing update messages carry only *functional* information, *i.e.*, information needed to compute the best path in the network. No information is *designed* into the protocol to facilitate route diagnosis.

In this paper, we proposed a simple and effective path vector routing diagnosis solution DRAQ. DRAQ collects necessary information for diagnosis through the following three means: (1) fully utilizing the existing connectivity information carried in the path vector protocol to build a partial topology graph; (2) enhancing path vector protocol with *root cause notification* to carry topological change information to relevant nodes; and (3) using active queries to obtain missing information on demand to achieve diagnosis goals. With all the necessary topology and topological change information in hand, and in the absence of false information and overlapping events, a diagnosis node can determine path removal and replacement with 100% accuracy. In the presence of a false routing update, DRAQ takes advantage of a network's rich connectivity and the attacker's inability to block *all* the correct routing updates from reaching a diagnosis node in most cases. The diagnosis node checks the consistency between received paths to detect the existence of false information, and uses majority votes among independent informatoin sources to identify the attacker. Our simulation results show that DRAQ can achieve high detection rate and attacker identification rate with low overhead.

We would like to draw the following conclusions from our DRAQ design and evaluation experience. First, DRAQ design shows that effective routing diagnosis can be achieved with minor additions to path vector routing protocols, in combination with creative use of information already embedded in the current protocol. Second, rich topological connectivity of a large network can be utilized for effective attack detection. As our simulation results show, attack against destinations with high node degree are less likely to succeed, and diagnosis node with high node degree is more effective in detecting and identifying attacks. By the same token, high degree nodes also need better protection because a compromised high degree node can also be more dangerous. Finally, DRAQ design shows a proof of evidence that one can achieve effective attack detection and identification in the absence of cryptographic mechanisms. We believe the exploration of such non-cryptographic mechanisms (such as DRAQ) is important. They are not designed to compete with cryptographic mechanisms but to complement them in deployment, so that the system can be well protected by these non-cryptographic mechanisms even when the crypto protection layer is broken.

References

[1] D. Chang, R. Govindan, and J. Heidemann, "The Temporal and Topological Characteristiccs of BGP Path Changes," in *Proceedings of ICNP*, November 2003.

[2] M. Caesar, L. Subramanian, and R. H. Katz, "Root Cause Analysis of Internet Routing Dynamics," Tech. Rep., UC Berkeley CSD, 2003.

[3] M. Lad, A. Nanavati, D. Massey, and L. Zhang, "An algorithmic approach to identifying link failures," in *PRDC*, 2004.

[4] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs, "Locating internet routing instabilities," in *Proceedings of ACM Sigcomm*, August 2004.

[5] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (s-bgp)," *IEEE JSAC Special Issue on Network Security*, 2000.

[6] J. Ng, "Extensions to BGP to Support Secure Origin BGP," ftp://ftp-eng.cisco.com/sobgp/drafts/draft-ng-sobgp-bgp-extensions-02.txt, April 2004.

[7] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz, "Listen and whisper: Security mechanisms for bgp," in *Proceedings of ACM NDSI 2004*, "March" 2004.

[8] Y.-C. Hu, A. Perrig, and M. Sirbu, "SPV: Secure path vector routing for securing bgp," in *Proceedings of ACM Sigcomm*, August 2004.

[9] D. Pei, M. Azuma, N. Nguyen, J. Chen, D. Massey, and L. Zhang, "BGP-RCN: Improving BGP Convergence Through Root Cause Notification," Tech. Rep. TR-030047, UCLA CSD, October 2003.

[10] J. Chandrashekar, Z. Duan, Z.-L. Zhang, and J. Krasky, "Limiting path exploration in path vector protocols," Tech. Rep., University of Minnesota, 2003.

[11] Dimitri Bertsekas and Robert Gallager, *Data Network*, Prentice-Hall, 1992.

[12] "The SSFNET Project," http://www.ssfnet.org.

[13] B. Premore, "Multi-as topologies from bgp routing tables," http://www.ssfnet.org/Exchange/gallery/asgraph/index.html.

[14] L. Gao, "On inferring automonous system relationships in the internet," *IEEE/ACM Transactions on Networks*, vol. 9, no. 6, 2001.

[15] R. Teixeira and J. Rexford, "A measurement framework for pinpointing routing changes," in *ACM SIGCOMM Network Troublestooting Workshop*, August 2004.

[16] J. Chandrashekar, Z.-L. Zhang, and H. Peterson, "Fixing BGP, One AS at a time," in *ACM SIGCOMM Network Troublestooting Workshop*, August 2004.

[17] Radia Perlman, *Network Layer Protocols with Byzantine Robustness*, Ph.D. thesis, MIT Lab. for Computer Science, 1988.

[18] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin, "Working around bgp: An incremental approach to improving security and accuracy of interdomain routing," in *NDSS*, 2003.