A Theory for Networks with Misconfigured Routers

Raj Kumar Rajendran, Dan Rubenstein
Dept. of Electrical Engineering
Columbia University, New York, NY 10025
Email: {raj,danr}@ee.columbia.edu

***Abstract*— For the past several decades, work that develops and analyzes network routing protocols has assumed that each network node properly implements the algorithm that establishes routes through the network. However, there have been several instances in which a trivial misconfiguration in a single router's implementation of the routing algorithm induced undesirable routes within the majority of the network. Given the high likelihood of program error or sabotage in today's networks, there is an urgent need to develop techniques that allow properly-configured routers to identify anomalous behaviors of their misconfigured counterparts.**

In this paper, we describe our development of a general theory that examines routing protocols in environments where some misconfigured routers "misbehave" and (intentionally or unintentionally) issue inaccurate reports that shift routing paths in the network. Application of this theory allows a node to "sense" the presence of a misconfiguration in the network through an analysis of its routing state. We show, using the Distance Vector routing protocol as an example, how this theory can be applied to practical routing protocols. We also describe our plans for a practical tool based on this theory that can be used by network administrators to detect anomalies.

I. Introduction

Routing algorithms, by nature of the task they perform, are implemented in a distributed setting. Over the years, a significant body of research has been developed to implement routing algorithms that converge efficiently to an accurate solution. In addition, research over the years has produced appropriate models and methodologies that can be used to further evaluate and understand the properties of these routing protocols [14].

The assumption in this large body of work is that routers implement the algorithm correctly. However, the distributed, heterogeneous nature of the routing environment and the fact that no single organization controls deployment within today's networks increases the likelihood that somewhere in the network, a router is *misconfigured*, and is not performing the way it is supposed to. In fact, over the years, there have been several instances where a small set of misconfigured nodes in a remote region of the network dramatically shifted the routing paths throughout the entire network [18], [19], [7], [16]. It is unclear whether these and future misconfigurations are accidents caused by buggy code, misinterpretation of proper installation procedure, or are in fact malicious attempts to subvert traffic. What is clear, however, is that we are a long way from being able to protect the network from these misconfigurations.

Ideally, network nodes that participate in a routing protocol would also implement mechanisms to detect misconfigured "rogue" routers elsewhere in the network, or, at the very least, to identify anomalous properties that would indicate the existence of rogues. Doing so would facilitate the tracking and identification of rogue nodes, and would permit appropriate action to be taken to protect/repair damages caused by the rogues. One way to approach developing these mechanisms is *reactive*: wait for misconfigurations to visibly affect network routing, then identify the bug that caused the misconfiguration and take steps (development of code patches, novel techniques) to enable future detection and prevention. *We have the more ambitious goal of developing pro-active techniques that detect misconfigurations before they visibly affect routing and without explicitly knowing the cause.* In other words, properly-configured routers should be enabled with mechanisms that can "sense" when some of their routing state just doesn't seem right.

Is this more ambitious goal even possible? Can nodes infer misconfigurations elsewhere in the network from just inspecting their routing state? Is this computationally feasible? Are there types of misconfigurations that can be sensed and types that cannot? This paper outlines our preliminary investigation of these questions where we show that it may be possible to achieve this goal.

We begin by formalizing our description of this problem. This formalism identifies misconfigurations that are detectable, and reveals a method which, in theory, can detect all detectable misconfigurations. However, the method in its raw form is computationally intractable. Hence, it becomes necessary to refine the method for each routing protocol and, in some cases, to focus on specific sub-classes of misconfigurations.

In addition to the general construction of the theory, we apply this theory to the distance vector routing-protocol, showing how a node, armed with the state it receives from its neighbors, can detect any misconfigurations that are detectable using only this information. We further describe our plans for a practical misconfiguration detection tool, that can be used by network administrators to analyze their nodes and networks for the presence of misconfigurations.

The rest of the paper proceeds as follows. We start by describing our theoretical methodology and formalizing it in Sec. III. In Sec. IV we illustrate the theory by applying it to the distance-vector protocol and outline our experiences in applying it to the Link-State and Path-Vector protocols. We describe our plans for a practical rogue-detection tool and future work in Sec. V. Sec. VI concludes the paper.

II. Prior Work

Broadly, the problem of threats posed by compromised or malicious routers has been categorized into two broad areas called *control-plane* attacks and *data-plane* attacks. When a router disrupts traffic by advertising false routes, it is known

as an attack on the control-plane; on the other hand if a router misroutes data, it is known as an attack on the data-plane. Our work deals with the first of these two classifications: attacks on the control-plane.

While several works have addressed control-plane attacks by misconfigured routers, their solutions focus on ensuring the authenticity of route updates by using encryption based schemes [4], [8], [10], [13], [27], [1], [11] and on detecting the consistency of route updates by actively probing the forwarding-path of advertised routes using secure tools that locate the source of routing misbehavior[29], [22]. Additionally many current works have focused on misconfiguration problems in a single protocol: [28] and [3] are empirical studies about BGP Policy and BGP router response to loads, while [15] and [9] deal with the problem of BGP convergence. Some work has focused on attacks by misconfigured routers on the data-plane [20], [2], [5], [12]. These works propose a distributed solution where multiple cooperating routers attempt to detect and avoid the disruptions caused by the misbehaving routers. These works differ from ours in that they concentrate on the data-plane and require multiple cooperating nodes.

Yet other bodies of work have attacked broadly similar questions. Competitive routing [21] is one such approach where selfish users in a communications network attempt to maximize their flow by controlling the routing of their flows and attempt to find if equilibrium states exist. This body of work differs in that they are concerned with traffic flows rather than reachability and with a competitive environment rather than a misconfigured environment.

III. The Theory

Our theoretical methodology evaluates routing protocols in environments where nodes can misbehave due to misconfigurations in the routing protocol implementation. To detect misconfigurations, a router examines its routing *state* which consists of all the information it has available about the current routing infrastructure. State is what is constructed via a combination of communicating with neighboring nodes and local computation on the exchanged information. Each node's state may be complete and contain the topology of the whole graph as in the Link-State routing algorithm or incomplete as in the Distance-Vector algorithm, where each node's view is limited to the distance to its neighbors and their shortest path distances to all other nodes in the network.

To facilitate our preliminary study, we make several assumptions about the information stored within the router's state:

- The routing protocol has converged, i.e., the state is stable.
- The router does not obtain additional information beyond what resides within its state. If it were to do so, this additional information would simply become part of its state.
- Our techniques described below apply to nodes that properly implement the routing protocol. Clearly, we cannot make any claims about the validity of our techniques when applied by misbehaving nodes.

Our assumptions rule out our ability to detect "dynamic" errors that can be induced by other types of misconfigurations. For example, by continually changing the distances it reports to a neighbor, a node can prevent the stabilization of the routing map within the network. A node could also introduce misconfigurations in the data-plane where it correctly computes routes but disrupts routing by dropping packets, modifying packet destinations, introducing phantom packets, or forwarding packets to incorrect neighbors. Dynamic misconfigurations and data-pane attacks are beyond the scope of our work which focuses on static misconfigurations in the control-plane.

At a high level, the basic procedure we will use to detect misconfigurations is fairly straightforward. A node n, armed only with the knowledge of its own state s and the (correct) implementation of the routing protocol, examines its state and asks itself "Does a feasible network exist where, if every node implements the routing protocol correctly, would result in my having state equal to s?" If no such network exists, then clearly, then there is no way in which n could wind up with state s, and hence some node must have reported incorrect information, indicative of a misconfiguration. On the other hand, suppose n can identify at least one feasible network. Then it may be the case that a misconfiguration occurred, yielding this state, or it may be the case that this feasible network yielded this state without misconfiguration. In this latter situation, if a misconfiguration led to this state, n cannot conclusively determine that a misconfiguration exists.

Using this procedure we can classify the misconfigurations of each protocol to be detectable or undetectable as shown in Fig. 1. The plane of the paper, in the figure, represents the space of possible misconfigurations for a particular protocol and the axis perpendicular to that plane represents the space of protocols. The misconfigurations of a particular protocol are shown to be broken up into two sets: detectable and undetectable misconfigurations. Ideally, for each protocol, we would know these two sets. But the ideal is hard to achieve because the set of all possible misconfigurations may be enormous. What we can do, however, is to take classes of misconfigurations and prove them to be undetectable - represented as ovals in the figure- using the procedure above. Else, if a class is detectable, we can provide techniques that detect them -shown as rectangles in th figure. In the next sections we show in detail how classes of misconfigurations can be proved to be detectable or undetectable for a protocol. We provide an illustrative example, using the distance-vector protocol, of a technique that identifies the presence of detectable misconfigurations.

A. Problem-Definition

We now describe our theory in a more formal manner. Let the network in which the routing protocol operates be a weighted graph $G = (N, E, W)$ where N is a set of nodes $\{n_i\}$ and E is a set of edges where each edge e_{ij} uniquely connects two nodes i and j. W is a set of weights w_{ij} corresponding to each edge. We let $\mathcal{G}$ denote the set of all *valid* graphs, G. In this network each node i in the graph G applies a routing protocol, generating its state S_i. A routing algorithm in a correctly-configured network can thus be defined formally as a function f that maps a weighted-graph $G(N, E, W)$ to the

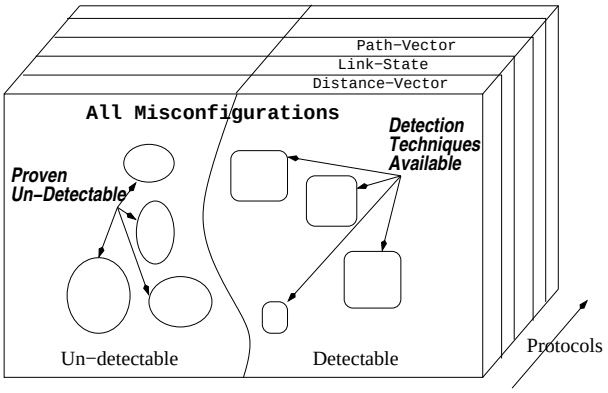


Fig. 1. The Space of Misconfigurations

sequence of states $(S_1, \ldots, S_{|N|})$, where the ith component is the state maintained by node i upon stabilization.

$$f : G(N, E, W) \rightarrow S = (S_1, \ldots, S_{|N|})$$

We define $f_i(G)$ to be the state S_i that results at node i when f is applied to G, i.e., $f(G) = (f_1(G), f_2(G), \ldots, f_{|N|}(G))$. We also define the routing algorithm in a misconfigured network to be a function h that maps G to an alternate sequence of states, $T = (T_1, \ldots, T_{|N|})$ where each T_i can be the same as S_i, but need not be. We define $h_i(G)$ with respect to h such that $h(G) = (h_1(G), h_2(G), \ldots, h_{|N|}(G))$.

We assume that each correctly-configured node i knows f (i.e., f can be derived from the routing algorithm), but does not necessarily know G in its entirety. If, however, a node i is given any graph, G', it can apply f to G' to determine its state $f_i(G')$ (as well as the state of all other nodes in G').

In theory, with unlimited computational resources, node n_i can perform the following process to identify misconfigurations:

1) For each graph $G_j \in \mathcal{G}$, node i computes $f(G_j) = (S_1^j, S_2^j, \ldots, S_{|N|^j})$.
2) If there exists a graph G_j for which $S_i^j = T_i$, then node i cannot detect the misconfiguration because it is plausible that the node resides within the valid graph G_j.
3) If, after exhaustively checking all $G_j \in \mathcal{G}$, no G_j is found that satisfies step 2, then the misconfiguration is detected: there no valid graph with correctly configured nodes that could produce the state T_i for i. The state can only have been produced via a misconfiguration in the routing protocol somewhere in the network.

Of course, limited computational resources make the above procedure infeasible - one cannot expect a node to apply the above procedure to the potentially infinite number of graphs in $\mathcal{G}$. To proceed in a computationally feasible manner, we must either reduce the search space by considering a subclass of graphs or a subclass of misconfiguration scenarios (or both). In addition, we can look for specific methods to detect misconfigurations: for example we can identify properties of the state that must hold; a violation of this property implies the presence of a certain class of miconfigurations.

1) Detecting Misconfigurations: There are two methods to detect misconfigurations for a specific protocol. One method is by identifying a property of the state that should be exhibited under a correct operation of the protocol. It is this approach that we take in our recent technical report [24] to identify misconfigurations in BGP traffic announcements. As a simple example, consider a network where a node i has two neighbors, j, and k with edge-weights w_{ij} and w_{ik} to them. According to the state of i, let d_{jm}, d_{km} be the distances reported by j and k to a fourth node m. If either $d_{jm} > (d_{km} + w_{ij} + w_{ik})$ or $d_{km} > (d_{jm} + w_{ij} + w_{ik})$ the state of node i exhibits a violation of the triangle inequality $d_{ab} \leq d_{ac} + d_{bc}$ indicating a misconfiguration somewhere in the network.

Another approach would be to significantly limit the space of graphs $\mathcal{G}$ that need to be considered. Then, it becomes computationally feasible to apply the method described above to detect misconfigurations. More formally, one must construct a function ψ_i that maps a node's state T_i to a (small) set of graphs $\{G_j\}$, upon which the 3-step process described above can be applied to identify a misconfiguration. This is the approach we use to identify misconfigurations in the Distance Vector Protocol, described below. Of course, if one does not prove that only this small set of graphs needs to be considered, this process can falsely detect a misconfigurations since there may be a graph that has not been considered which would yield T_i.

2) Proving Undetectable Misconfigurations: If we wish to show that a particular class of misconfigurations cannot be detected by node i, we can extend on our second idea above. Assume it is possible to define a function h that maps each valid graph G_j to a sequence of states $(T_1^j, \ldots, T_{|N|^j})$ where h implements the "misconfigured" routing protocol. We then construct a mapping, ϕ_i that maps each valid graph $G_j \in \mathcal{G}$ to another valid graph $\phi_i(G_j) \in \mathcal{G}$, such that $h(g_j) = f_i(\phi(G_j))$.

By doing this, we show for any valid graph G, each properly-configured node i can find some other valid graph $\phi_i(G)$ that would have produced the state it holds in a properly configured network. Hence, it is impossible for node i to disprove the correctness of the protocol. [1]

As a simple example, consider a graph where two non-neighbor nodes m and n both falsely claim an edge e_{mn} of weight w_{mn} between them and advertise distances as if this edge existed. No other node i in the network will be able to detect this misconfiguration because for any graph G_j, node i can always construct the new graph $\phi_i(G_j) = (G_j \cup e_{mn})$ where $f_i(\phi_i(G_j))$ will produce T_i^j, node i's state due to the lies of m and n. Therefore this misconfiguration is undetectable.

This method is also used to prove that our approach to identify misconfigurations in the Distance Vector Protocol catches all misconfigurations that can indeed be observed using a node's available state.

[1]Note one subtlety in our construction: a different mapping, ϕ_i is permissible for each node. This is because each node uses all the state it has available to it to form its own view of possible valid graphs. Were two nodes i and j to exchange notes in such a way to ensure that $\phi_i = \phi_j$, this exchange would be part of the state, and would simply restrict the set of feasible functions one could consider for ϕ_i and ϕ_j.

Definition We define a class of misconfigurations to be *undetectable* if we can derive a sequence of ϕ_i that satisfy the above properties for each i for each valid graph G and any function h that falls within this class of misconfigurations. We define a class of misconfigurations to be *detectable* when, for any function h that falls within this class of misconfigurations, there is a properly-configured node i such that for any ϕ_i that maps valid graphs to valid graphs, $f_i(\phi_i(G)) \neq h_i(G)$.

IV. Detecting Distance Vector Misconfigurations

In this section, we apply the general methodology to the Distance Vector protocol. We consider a network where each node's state consists of the information that is immediately available after the Distance Vector protocol converges. Namely, a node knows its immediate distance to each of its neighbors, as well as each neighbor's shortest path distances to all nodes in the network, and its own shortest path distances to all nodes in the network (note this last group of information is easily derived from the previous two).

Given a node's state T_i that results from participating in Distance Vector routing in the network, we define a function ϕ_i, described as an algorithm, that maps this state to a single graph, G'. We call this graph the canonical graph. Distance vector is simulated by node n_i in its memory upon the canonical graph, G'. *If the state of node n_i in the simulation does not match T_i, we can prove that a misconfiguration must exist within the protocol. If the states match, then we can prove that if a misconfiguration exists, then it cannot be detected using the state in node n_i!*

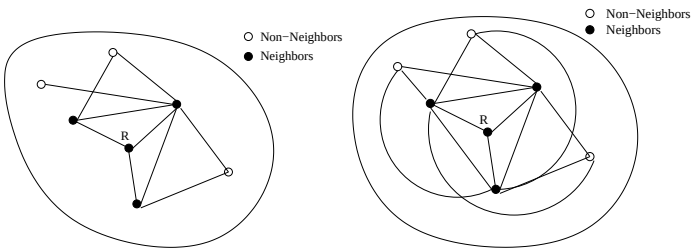
First, we briefly describe distance vector routing. In the distance vector algorithm, each node i periodically exchanges a vector of distances to each node in the graph with each neighbor n. On each exchange, node i recomputes its distances to node j according to $\hat{d}^i_j = \min(d^i_j, d^n_j + d^i_n)$ where d^q_p is the distance to node p according to node q. After a finite number of rounds of such exchanges, the distance vectors of each node will stabilize and each node will have a vector of its shortest distances to every other node in the graph.

Definition When $f : G(N, E, W) \rightarrow S = \{S_1, \ldots, S_{|N|}\}$ represents the distance vector algorithm, the view or state S_i of each node i consist of $|\mathcal{N}(i)|$ vectors of distances to each node in the graph where each vector corresponds to one of i's neighbors denoted as $\mathcal{N}(i)$.

Each node i's state is represented by an $|N| \times |\mathcal{N}(i)|$ table where N is the set of all nodes in the network and $\mathcal{N}(i)$ is the set of node i's neighbors.[2]. In node i's distance vector table, the entry in the (n, m)th position indicates the distance from node m to node n by taking the path with next hop m. We write the value (distance) of the entry at (n, m) as $v(n, m)$. Note that we assume that the distance vector table does not include the edge-distance from i to the neighbor m but that the table contains $v(i, m)$.

A node i that wishes to identify a misconfiguration constructs a graph G' whose set of nodes is identical to those in G but whose edges differ significantly. The new graph G' contains edges $e_{im} \forall m \in \mathcal{N}(i)$ from i to each of its neighbors with weights w_{im} as in G. It also contains edges $e_{mm'}$ from each neighbor $m \in \mathcal{N}(i)$ to every node $m' \in N$ in the graph with weight $w_{mm'}$ equal to $v(m, m')$. In other words, i has a 2-hop path to each destination m' that is not its neighbor, where the length of this path matches what is reported in its table.

The construction of a graph G' from the original G is pictured in Fig. 2.



(A) Original Graph G (B) Newly Constructed Graph G'

Fig. 2. Constructing G'

Node i subsequently computes the shortest path for each neighbor $m \in \mathcal{N}(i)$ to all destinations within G'. We represent these values as $v'(n, m)$.

Lemma 4.1: If there is some n, m such that $v(n, m) \neq v'(n, m)$ then one or more misconfigurations exists in G. If there is no such n, m, even if misconfigurations exist in G it is impossible for i to detect them.

Proof: The proof is provided in Section VI. ■

We illustrate the Lemma in Fig. 3. Consider the graph in Fig. 3(A) where the nodes and weights are as indicated (solid-lines indicate edges). Consider the case where a misconfiguration causes the node R to report a distance to D of 99 to its neighbor $N1$, rather than the true shortest-distance of 3 through F and $N2$. Because of this misconfiguration $N1$ will believe that its shortest-distance to D is 10 through its direct edge rather than 4 through R, F and $N2$. Therefore $v(D, N1) = 10$ as viewed by I.

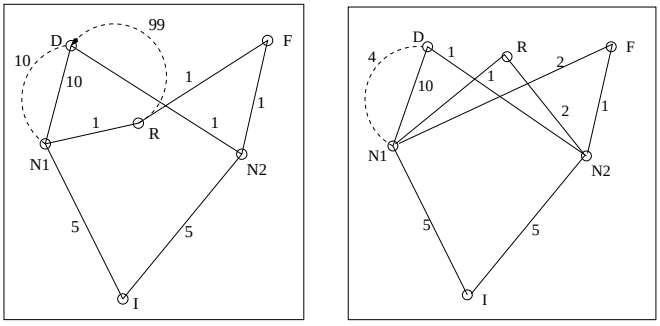
To verify if a misconfiguration exists, node I constructs a graph G' from the view as detailed above. This graph will be as pictured in Fig. 3. When we run the distance vector protocol on this new graph G', the protocol will report $v'(D, N1) = 4$! From Lemma 4.1 we will conclude correctly that the original graph G is misconfigured.

A. Application of the Technique to Other Routing Protocols

We briefly summarize our experiences applying the techniques within the domains of link-state and path-vector (BGP routing). For link-state routing, since nodes know the entire topology a priori, the theory is rather trivial. For BGP routing, the lack of a well-defined policy greatly complicates the construction of a theory to detect misconfigurations. Nonetheless, we briefly describe our efforts in these two areas:

1) Link-State: OSPF is an example of a protocol that uses Link-State routing where each node maintains a list of the neighbors and distances to neighbors for each node in the network. Periodically nodes exchange their neighbor-list with

[2]For simplicity of presentation, we assume a node maintains state indicating the distance to itself through its neighbors.



(A) Misconfigured Graph G (B) Constructed Graph G'

Fig. 3. The Construction

each neighbor. This process continues until each node is aware of the neighbors for every node in the network. Each node then independently computes its distance and shortest path to each node in the network.

In a link-state routing protocol, each node's state is a snapshot of the entire graph. We write $S_i = G_i$ where G_i is a graph $G_i = (N_i, E_i, W_i)$. This property trivializes the analysis of detecting misconfigurations for link-state protocols. In fact, we have that $\phi_i(G_i) = \{G_i\}$. In other words, imagine if node i were to enumerate the set of possible network graphs which, after correctly running the link-state protocol would produce G_i. The only graph in this set would be G_i. In fact, if $n_1, n_2, \cdots, n_k$ are neighbors of G_i and all neighbor's states match i's state (i.e, $G_{n_j} = G_i$ for all neighbors j, then clearly node i cannot detect a misconfiguration. In contrast, if two neighbors, x and y have states that remain fixed yet do not match, $G_x \neq G_y$, (or $S_x \neq S_y$) then i will detect a misconfiguration.

The above observation can easily be extended to the following Lemma by applying the neighbor argument above along paths of properly-configured nodes:

Lemma 4.2: Let $G = (N, E, W)$ be a graph where a subset of nodes, $N' \subset N$ are properly configured. Then a misconfiguration is detectable iff there exist two nodes $x, y \in N'$ where $S_x \neq S_y$ and there is a path from x to y through a series of nodes $n_1, n_2, \cdots, n_k$ where $n_j \in N'$ for $1 \leq j \leq k$.

B. Path Vector Protocols

In the path-vector routing algorithm, each node maintains paths, or lists of nodes to be traversed, to each node in the network. Periodically each node exchanges *some* paths with each neighbor and recomputes its most-desirable path to each node in the network based on its individual *policies*. Therefore each node's state or view consists of the most-desirable paths each of its neighbors claims to subsets of nodes in the network. BGP is an example of a protocol that uses path-vector routing.

We determine potential misconfigurations in BGP by analyzing the state of each node for properties that should hold if the nodes, known as ASes, in a BGP network operate according to the standard BGP routing policy. We prove that if a route contains a set of ASes in a certain order and we number these ASes in increasing order, then another route can never contain the same ASes in an order that first increases and then decreases. ASes can use this property to check if other ASes are indeed operating according to the standard BGP policy. Details and experimental results can be found in our technical report [24].

V. Further Work

We plan to extend our work in two different directions: theory and practice. We first plan to apply our current theory to other protocols and identify classes of misconfigurations that can be proved detectable or undetectable and provide techniques to point out detectable misconfigurations. Second, we plan to produce a verification tool, based on our theory, that can be used by administrators of networks running various protocols to analyze the state of routers and individual nodes for the presence of misconfigurations elsewhere in the network. We describe both below.

A. Extending the Theory

In this paper we have applied our theory to what we call *tight* routing-protocols, where each node is required to run exactly the same algorithm. However *loose* protocols, such as BGP, where different nodes may run different algorithms in choosing and communicating routes are widely used. Since each node in a loose protocol may run a different routing algorithm, it is not obvious how misconfigurations can be defined, let alone verified. In such protocols, it is necessary to identify "normal" behavior, if they exist, model these behaviors and then attempt to detect nodes that deviate from these norms.

There has also been a growth in ad-hoc and sensor networks which use distributed algorithms to estimate various parameters at each node by iteratively exchanging information with other nodes within their communication radii [25], [6], [17], [26]. Even though these algorithms compute parameters such as geographic location and time rather than the shortest path, they function in a manner very similar to distributed routing algorithms. We plan to apply our theory to these algorithms and attempt to estimate the amount by which rogue-nodes can distort the parameter that is being estimated.

We also plan to extend our theory to situations where multiple well-behaved nodes can cooperate and share information. We would like to study how such cooperation changes the classes of detectable and undetectable misconfigurations. We also wish to extend our theory so that, in addition to detecting the presence of misconfigurations, it provides insight into the identity of the violators. As more well-behaved nodes cooperate we believe that the location of rogue-nodes can be determined more accurately. We plan to explore the theories behind such determination and implement practical tools that can be used to identify rogues.

B. A Tool for Rogue Detection

We would like to provide a means by which network practitioners can apply our methodology and protocol-specific state-checking algorithms in practical situations to verify the presence of rogues in their networks. To this end we plan to create a practical tool that implements our theory and makes it applicable in practical situations. This goal of this tool will

be to provide administrators of networks running well-known routing protocols the ability to verify the correctness of the state of individual nodes or routers.

We foresee several challenges in implementing such a tool. We list some of the issues that we believe need to be solved to create a practical and useful tool:

- It would be very attractive to have a single tool that works across all well known protocols. However our theoretical methodology applies across protocols while our misconfiguration-detection techniques are protocol-specific. It it viable or feasible to produce a single tool that can be used across protocols?
- We plan to continually apply our methodology to new routing-protocols, identify new classes of misconfigurations and provide algorithms that detect them. How can we ensure that the tool, once deployed, is kept up to date with the latest algorithms. Do we need to build in an ability for the tool to to update itself from a central location, much like virus software?
- How can this tool read in a node's state information? Different implementations of a single protocol may store their state-information in different formats. Do we need a way for the network-administrator to describe the layout of a node's state information? Are there ad-hoc standards for the way this information is stored in different protocols?
- In its current form, our methodology is applicable to a single node. However we plan to extend our theory so that multiple cooperating nodes can pool their information to increase the probability of detecting rogues, and to accurately pinpoint offending nodes. How will the tool deal with this extension? If the solution is to deploy the tool at multiple nodes, how will these copies of the tool communicate? Should they use an overlay network to share information among themselves since the underlying network's routing may get compromised?

We anticipate, as indicated by the challenges listed above, that building such a practical tool will be a challenging but satisfying experience.

VI. Conclusion

We have described our development of a theoretical methodology that evaluates routing protocols in environments where nodes can misbehave. This theory can be applied by routers to analyze their routing-states and pro-actively sense misconfigurations before these misconfigurations visibly affect routing. Additionally, the theory allows classes of misconfigurations in a protocol to be classified detectable or undetectable. This feature can be used by network researchers and routing protocol designers to evaluate the robustness of protocols in rogue environments and to provide insight into possible modifications that would further limit the sets of undetectable misconfigurations. In a larger context, this theory contributes to our understanding of the design and evaluation of self-healing, anomaly-tolerant networks.

References

[1] W. Aiello, J. Ioannidis, and P. McDaniel. Origin authentication in interdomain routing. In *Proc. of 10th ACM conference on Computer and communication security*, pages 165–178. ACM Press, 2003.

[2] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *1998 IEEE Sym. on Security and Privacy*, pages 115–124, 1998.

[3] D.-F. Chang, R. Govindan, and J. Heidemann. An empirical study of router response to large bgp routing table load. In *Proceedings of the second ACM SIGCOMM Workshop on Internet measurment workshop*, pages 203–208. ACM Press, 2002.

[4] S. Cheung. An efficient message authentication scheme for link state routing. In *ACSAC*, pages 90–98, 1997.

[5] S. Cheung and K. Levitt. Protecting routing infrastructures from denial of service using cooperative intrusion detection. In *New Security Paradigms Workshop*, 1997.

[6] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Fifth Symposium on Operating System Design and Implementation (OSDI) 2002*, 2002.

[7] J. Farrar. C&w routing instability. nanog mail archives. Available at http://www.merit.edu/mail.archives/nanog/2001-04/msg00209.html.

[8] M. Goodrich. Efficient and secure network routing algorithms. In *Patent Application*, January 2001.

[9] T. G. Griffin and G. T. Wilfong. An analysis of BGP convergence properties. In *Proc. of SIGCOMM*, Cambridge, MA, August 1999.

[10] Y. Hu, A. Perrig, and D. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *The 8th ACM International Conference on Mobile Computing and Networking*, September 2002.

[11] Y.-C. Hu, A. Perrig, and D. B. Johnson. Rushing attacks and defense in wireless ad hoc network routing protocols. In *Proceedings of the 2003 ACM workshop on Wireless security*, pages 30–40. ACM Press, 2003.

[12] J. R. Hughes, T. Aura, and M. Bishop. Using conservation of flow as a security mechanism in network protocols. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 132–141. IEEE Computer Society Press, Los Alamitos CA, USA, 2000.

[13] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4), 2000.

[14] J. F. Kurose and K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Inc., 2002.

[15] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 6(5):515–528, 1998.

[16] C. Labovitz, G. R. Malan, and F. Jahanian. Origins of internet routing instability. In *Proc. of IEEE INFOCOM 1999*, Mar 1999.

[17] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 120–130, Aug. 2000.

[18] R. Mahajan, D. Wetherall, and T. Anderson. Understanding bgp misconfiguration. In *Proc. of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press, 2002.

[19] S. Misel. Wow, as7007! Available at http://www.merit.edu/mail.archives/nanog/1997-04/msg00340.html.

[20] A. T. Mizrak, K. Marzullo, and S. Savage. Brief announcement: detecting malicious routers. In *Proc. of the 23rd ACM symposium on Principles of distributed computing*, pages 369–369. ACM Press, 2004.

[21] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Trans. Netw.*, 1(5):510–521, 1993.

[22] V. N. Padmanabhan and D. R. Simon. Secure traceroute to detect faulty or malicious routing. *SIGCOMM Comput. Commun. Rev.*, 33(1), 2003.

[23] R. K. Rajendran and D. Rubenstein. A theory for networks with misconfigured routers. Available from http://www.ee.columbia.edu/ kumar/papers/p2004-03.pdf.

[24] R. K. Rajendran, D. Rubenstein, and M. Wasserman. A theoetical method for bgp policy verification. Available from http://www.ee.columbia.edu/ kumar/papers/p2004-02.pdf.

[25] K. Romer. Time synchronization in ad hoc networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing*. ACM Press, 2001.

[26] A. Savvides, C.-C. Han, and M. B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Mobile Computing and Networking*, pages 166–179, 2001.

[27] B. Smith and J. Garcia-Luna-Aceves. Securing the border gateway routing protocol. In *Proc. Global Internet'96*, November 1996.

[28] N. Spring, R. Mahajan, and T. Anderson. The causes of path inflation. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 113–124. ACM Press, 2003.

[29] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz. Listen and whisper: Security mechanisms for bgp. In *Proc. First Sym. on Networked Systems Design and Implementation*, March 2004.

APPENDIX

Lemma 6.1: If there is some n, m such that $v(n,m) \neq v'(n,m)$ then one or more misconfigurations exists in G. If there is no such n, m, even if misconfigurations exist in G it is impossible for i to detect them.

Proof: First, we show that i cannot detect a misconfiguration if $v(n,m) = v'(n,m)$ for all n and m. Suppose this property holds true. Then G' is a potential topology of the network. Hence, it is not possible for a misconfiguration to be detected.

We will now show that the existence of some $v(n,m) \neq v'(n,m)$ implies a misconfiguration in G. Our proof will be by contradiction. First, consider the case where both n and m are neighbors of i. Since the graphs are undirected, clearly we have $v'(n,m) = v'(m,n)$, such that if the reports in G directly reveal an asymmetry (i.e., $v(n,m) \neq v(m,n)$), this must be the result of a misconfiguration and we are done.

Assume now that $v(n,m) = v(m,n)$ whenever both n and m are neighbors of i, and also that there exist n and m where $v(n,m) \neq v'(n,m)$ yet no misconfigurations exist in G. Let $p' = (n_0, n_1, n_2, \cdots, n_k)$ be a shortest path from node m to node n (hence $n_k = n$ and $n_0 = m$) in G' with length $v'(n,m)$. Since the direct path m, n has length $v(n,m)$ and $v'(n,m)$ is the length of the shortest path from m to n, it must be the case that $v'(n,m) < v(n,m)$. Also, note that in G', there is no edge between two nodes x and y when $x, y \min \mathcal{N}(i) \cup \{i\}$. Hence, if any node n_j in p' is not a neighbor of i, then n_{j-1} and n_{j+1} both are neighbors of i. For each j, since either n_{j-1} or n_j is a neighbor of i or i itself, the length of the shortest path between n_{j-1} and n_j is either given as $v(n_{j-1}, n_j)$ (if n_j is a neighbor of i) or as $v(n_j, n_{j-1})$ (if n_{j-1} is a neighbor of i) or as both when both are neighbors of i. We let $t(n_{j-1}, n_j)$ equal this length.

The length of the path p' in G' is therefore $\sum_{j=1}^{k} t(n_{j-1}, n_j)$, and since p' has length $v'(n,m)$, we have that $\sum_{j=1}^{k} t(n_{j-1}, n_j) = v'(n,m) < v(n,m)$. Consider a path p in G that starts at node m and follows a path to n that is a conjunction of sub-paths, $p_1, p_2, \cdots, p_k$ where p_j is a shortest path in G from n_{j-1} to n_j. It is permissible that p has cycles or "backtracks", but each p_j is cycle-free. Since there are no misconfigurations in G, the length of path p in G is given by the (correct) reports of i's neighbors: $\sum_{j=1}^{k} t(n_{j-1}, n_j) = v'(n,m) < v(n,m)$. Hence, we have a contradiction: path p is a valid path in G of length $v'(n,m)$, yet node m reports to i that its shortest path to n has length $v(n,m) > v'(n,m)$. ■