Control Plane Resilience: The Method of Strong Detection

Raj Kumar Rajendran, Vishal Misra, Dan Rubenstein
Columbia University
New York, NY
Email: {raj,danr}@ee.columbia.edu,misra@cs.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 called *Strong Detection* 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 strong-detection allows a node to "sense" the presence of *any* detectable misconfiguration in the network through an analysis of its routing state. We also demonstrate a practical algorithm that allows Strong Detection to be applied to the Distance-Vector protocol.

I. Introduction

Over the years, a significant body of research has been developed to implement distributed routing algorithms that converge efficiently to an accurate solution. Additionally, research has produced ways to evaluate and understand the properties of these routing protocols [1]. 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 makes it likely that somewhere in the network, a router is *misconfigured*, and is misimplementing the algorithm. 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 [2]–[5]. While the cause of these misconfigurations is unclear, it is clear that we are a long way from being able to protect the network from such misconfigurations.

Ideally, network nodes that participate in a routing protocol would also implement mechanisms to detect misconfigured "rogue" routers elsewhere in the network. 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 believe that a better approach is to develop *pro-active* techniques that detect misconfigurations *before they visibly affect routing and without explicitly knowing the cause*. In other words, we believe that 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 investigates these questions and shows that it is possible to achieve this goal.

We begin by introducing a formalism called Strong-Detection which, in theory, identifies 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. We demonstrate a computationally feasible algorithm for the Distance-Vector routing-protocol, and show how a node, armed with the state it receives from its neighbors, can detect any detectable misconfigurations using only this information.

II. Prior Work

While several works have identified that disruption due to incorrect implementation in routing-protocols is an important problem, the approaches to solutions have been different. In [6] and related work the authors set out to identify nodes that show erroneous behavior, but in contrast to our work, they do so by analyzing traffic patterns (in routing parlance, they analyze data-plane data while we analyze control-plane data). Others emphasize the need for reliable communication [7] and use centralized public-key infrastructures or key-distribution mechanisms to address the problem [8]–[14]. However these works do not attempt to harness the self-monitoring capabilities that protocols possess.

The authors in [15], [16] address the question of decentralized security in networks. They propose a toolkit of primitives that can be added to a routing-protocol to make it more secure. Our work differs in that we attempt to detect incorrect behavior without modifying the protocols. Other

The first and last authors are with the Dept. of Electrical Engineering while the second author is with the Dept. of Computer Science

works [17], [18] propose tools that can locate the source of routing misbehavior in the face of uncertainty and insecure environments.

Some bodies of work have attacked broadly similar questions. Among these is competitive routing [19] where selfish users in a communications network attempt to maximize their flow by controlling the routing of their flows and the works attempt to find if there exist equilibrium states. 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.

The brief announcement [20] on strong-detection was among the first works that provided a technique for detecting static inconsistencies while being grounded in a theoretical framework. It complements other studies that have sought to understand the nature of misconfigurations [21] and yet others that aim to detect misconfigurations through observance of dynamic behavior [6], [22]–[28].

Some recent work that has addressed this problem uses heuristics to make the best educated guess possible about the state of the network, but can incorrectly infer that there are problems when in fact none exist [29]–[32]. Other works take advantage of simultaneous analysis of multiple perspectives [33] when nodes are willing to share their routing table information.

III. Detecting Misconfigurations

In this section, we develop the idea of Strong Detection which will enable us to understand the types of misconfigurations that various classes of protocols can and *cannot* detect.

We consider a network $G = (V, E, W)$ where V is the set of nodes, E is the set of edges that connect pairs of nodes, and W the corresponding edge-weights. Each node $n \in V$ is aware of all other nodes in the network and communicates directly with a set of *neighbor* nodes, represented as $N(n) = \{N_1, \cdots, N_{|N(n)|}\}$. Each node n also maintains a protocol-specific *state*, d_n where it records some of the information exchanged. The information contained in d_n will vary depending on the protocol used.

A node n's state can be thought of as a table of size $|V|$ x $|N(n)|$, whose entry, $d_n(i, j)$ in the ith row and jth column is the information reported to it by neighbor n_j about the path to node i. This general framework does fairly well at classifying a wide variety of routing protocols that "learn from their neighbors". An example state for the Distance-Vector protocol is given in Sec. III-A.

Consider the perspective of node n in the network. Suppose that some other misconfigured or malicious node $\hat{n}$ in the network incorrectly implements the protocol, and reports erroneous routing information to its neighbors. This erroneous information could propagate through the network, possibly altering d_n, and lead other nodes, including n, to select routes that are non-optimal.

If a node n analyzes route information received from other nodes for incorrect implementations we call it a *monitor*. A monitor node performs its usual routing protocol duties, but it also *self-monitors* its own state for misconfigurations.

Before continuing, we state the assumptions we make to keep the theoretical problem both tractable and well-specified. We assume that the analysis is conducted after routes have stabilized, since transient inconsistencies may exist during stabilization. We also assume that modifications to the protocol are not allowed, and that the node conducting the analysis is functioning correctly. Finally we only attempt to detect the presence or errors. Identifying the source of the error is beyond the scope of this work.

A. Weak Detection

Can other nodes detect when a misconfigured node $\hat{n}$ introduces an anomaly? This problem has been studied previously for specific protocols. For instance, in [34], it is shown that the triangle inequality can be applied within RIP [35] (a specific Distance-Vector implementation) to detect certain misconfigurations.

A shortcoming of this previous method is that it identifies a specific property (the triangle inequality in the case above) that the state at a node (or set of nodes) should exhibit, and then looks for violations only of this specific property within the node's state. If a violation of this property is identified, then clearly this is sufficient evidence that the network is exhibiting a misconfiguration. However when a violation is *not* found, this does not necessarily mean that a misconfiguration does *not* exist.

n	a	b	e
a	0	1	12
b	4	0	7
c	12	13	8
d	5	9	6
e	9	6	4
f	12	15	13
g	4	9	2

Fig. 1. Violation of Symmetry

n	a	b	e
a	0	1	1
b	1	0	13
c	12	13	8
d	5	9	6
e	1	3	0

Fig. 2. Violation of the Triangle-Inequality

Consider the example state-table in Fig. 1 where nodes a, b, c, d, e, f and g execute the Distance-Vector Protocol in an undirected network. We know that in an undirected network, the distance $d(X, Y)$ reported by node X to node Y must be equal to $d(Y, X)$, the distance reported by node Y to node X. However in the state-table of Fig. 1, $d(a, b) \neq d(b, a)$ indicating a misconfiguration. Checking for the symmetry property identified a misconfiguration in this example.

Now consider the distance-vector state-table of node n shown in Fig. 2 where a reports that its shortest-path distances to nodes a,b,c,d and e to be 0, 1, 12, 5 and 1 respectively. Similarly b reports that its shortest-path distances to the nodes a,b,c,d and e are 1, 0, 13, 9, 3 while node e reports its shortest-path distances to be 1, 3, 8, 6 and 0. In this state-table, symmetry is not violated since a and b both report distances of 1 to each other, b and e report distances of 3 to each other and a and e report distances of 1 to each other. We next check the triangle

inequality $d(X,Z) \leq d(X,Y) + d(Y,Z)$ where as before $d(X,Y)$ indicates the distance reported by X to Y. A simple check reveals that $d(b,e) = 3$, $d(b,a) = 1$, $d(a,e) = 1$ and therefore $d(b,e) > d(b,a) + d(a,e)$ violating the triangle-inequality, indicating the presence of a misconfiguration.

n	a	b
a	0	2
b	2	0
c	3	1
d	3	3

Fig. 3. Node N's state

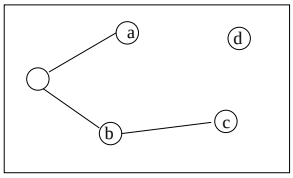


Fig. 4. Undetectable Report

To see why checking for properties such as symmetry and the triangle inequality is *weak* consider the example state-table in Fig. 3 where node n executes the Distance-Vector Protocol along with four other nodes, a, b, c and d. Suppose that the length of every edge in the network is known to equal either 1 or ∞ (i.e. no edge). a and b are n's neighbors and are connected to it by edges of distance 1. The distance-vector state-table of node n is shown in Fig. 3 where a reports to node n that its shortest-path distances to nodes a,b,c and d are respectively 0, 2, 3, and 3. [1] Similarly, node b reports distances of 2, 0, 1, and 3 respectively to these four nodes.

In this state-table, symmetry is not violated since a and b both report distances of 2 to each other. Similarly the triangle inequality is not violated in node n's state table. For instance, a claims its shortest path to node c has length 1. If the sum of claimed shortest path lengths from a to b and b to c was less than 3, the triangle inequality would be violated, but in this graph, this sum equals 3.

However, one can see that a misconfiguration must in fact exist as we will find that it is impossible to place edges of length 1 between the nodes pictured in Fig. 4 in a way to satisfy the state-table of Fig. 3. Consider the following argument. If all the information in n's state were correct, then there clearly is an edge of length 1 from b to c (the only way to get length 1 between them). Now since a is distance greater than 1 from the three nodes b, c, d it must attach to b through n (i.e. through the edges $a-n$ and $n-b$). So we have the edges $a-n$,$n-b$ and $b-c$. Since b is distance 3 from d and n and c are distance 1 from b, it must be that d attaches to a. But then $d(a,d) = 1$ and not 3 as shown in Table 3. Therefore this network cannot exist! Here, checking the symmetry and triangle inequality properties failed to identify the misconfiguration even though it was possible for node n to detect a misconfiguration through an analysis of its state.

We say that the triangle-inequality based method described above belongs to a class of methods that apply *Weak Detection*. A method provides Weak Detection when, given a node's state, an existing misconfiguration is not detected for one (or more) of two reasons:

- The misconfiguration is undetectable, regardless of what property is explored.
- The misconfiguration is detectable by checking some property, but the Weak Detection method did not check the appropriate property.

B. Strong Detection

Our work investigates what is called *Strong Detection* [20] where the goal is to construct methods that, like Weak Detection methods, detect misconfigurations. However, Strong Detection methods *must* detect any misconfiguration that is detectable by *any* property.

We start by providing a bird's-eye view of how Strong Detection functions at a node n, and then we show how it can be practically implemented. The basic idea behind implementing Strong Detection at a monitor node n is to try to identify a network that could yield the state d_n it obtained. More formally, let $\mathcal{G}$ be the (possibly infinite) set of *valid* network configurations, i.e., the actual network must be described accurately by some $G \in \mathcal{G}$. For instance, if distance is computed in terms of the number of hops, then $\mathcal{G}$ would be the set of networks with edges of length 1. If edge lengths equal propagation delay, then $\mathcal{G}$ could be the set of all graphs, each of whose edge lengths are all less than 500 (i.e., a conservative upper bound on the propagation delay between a pair of nodes).

Suppose n checks each network $G \in \mathcal{G}$. To check a particular network G, n builds a "toy" network that describes G, and then, in its local memory, simulates the routing protocol upon this (network) graph G to obtain a state, d_n^G for the node in G that corresponds to n. n then compares the state d_n^G in the "toy" network G to its actual state d_n in the real network.

There are two outcomes to consider:

- If no network $G \in \mathcal{G}$ satisfies $d_n^G = d_n$, then a misconfiguration must have occurred: there is no valid network that would have generated the obtained state.
- At least one $G \in \mathcal{G}$ satisfies $d_n^G = d_n$. Then n cannot detect the misconfiguration, if one exists. This is because, from n's perspective, the actual network may be described by G, and when the routing protocol was run correctly, the returned state was d_n^G. On the other hand, the network might be some other G' where $d_n^{G'} \neq d_n$ when the routing protocol runs correctly, but a misconfiguration produced d_n.

The problem with Strong Detection, as described above, is the time needed to either find a G that produces a matching state, or the (potentially infinite) time needed to demonstrate that there is no matching graph.

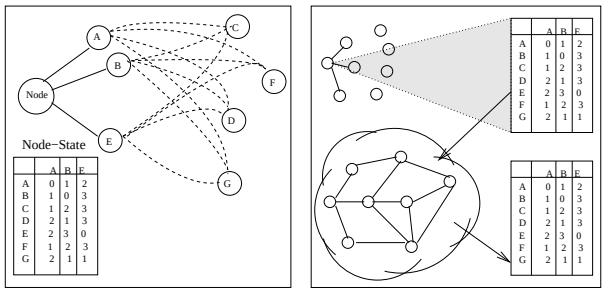
C. Computationally Feasible Strong Detection

We now show, in general, how Strong Detection can be implemented within a reasonable (i.e., low-degree polynomial) amount of time for a variety of protocols. The key idea is to identify how to construct a single special graph, G from within the space of valid graphs, $\mathcal{G}$, which we call

[1] Note here that we are assuming that the value reported in the table indicates the distance from the neighbor to the destination. An alternative form often used is to have the value indicate the distance from the node n itself to the destination through that neighbor. Since n knows the edge-length to its neighbor, the two forms provide equivalent information.

the *canonical graph*. Node n with state d_n pictured in Fig. 5(a) runs the following procedure:

- n executes an algorithm that takes as input its state, d_n, and outputs a particular graph G' with edge-weights w', which we refer to as the *canonical graph*. This process is pictured in Fig. 5(b).
- If G' is a valid graph ($G' \in \mathcal{G}$), then n next simulates the routing-protocol on G', producing simulated state d'_n for node n as pictured in Fig. 5(b).
- If $d'_n = d_n$, then we have identified a valid graph G', and hence there is either no misconfiguration or it is impossible to detect, since G' may accurately describe the network and would cause n to obtain state d_n within a correct implementation.
- If G' is not valid ($G' \notin \mathcal{G}$), or if ($G' \in \mathcal{G}$) but $d'_n \neq d_n$, then there is no graph $G \in \mathcal{G}$ that would produce state d_n when the protocol is run on it. This is a rather strong claim and we have proofs for the distance-vector protocol we consider.



(a) A Node's State (b) The Canonical Graph

Fig. 5. Strong Detection

This procedure applies to the very broad class of graphs where each pair of nodes i, j has a different set $S_{i,j}$ of allowable values for $w(i, j)$, making a graph G valid if and only if $w(i, j) \in S_{i,j}$. The $S_{i,j}$ are assumed to be known a priori (i.e., node n would know these values, and can be used as input to the algorithm that constructs the canonical graph). Each set $S_{i,j}$ can be distinct, and can be any arbitrarily chosen collection of intervals whose lower boundary is closed (i.e., the intervals forming $S_{i,j}$ can be of the form $[x, y]$ or $[x, y)$). For example $S_{A,C} = \{1, 3, [4.2 - 5.6]\}$ and $S_{B,C} = \{2, 4, [5.1 - 7.6)\}$ are valid allowable values for a single graph. Our procedures apply to this broad class of graphs unless otherwise stated.

We next describe how the canonical graph G' mentioned in Sec. III is constructed for the Distance-Vector protocol.

IV. The Distance-Vector Protocol

In this section we describe the application of the Strong Detection technique to the popular Distance-Vector routing protocol which is still widely used in various forms. The popular RIP protocol was [35] an early implementation of the Distance-Vector protocol. While RIP is currently not in wide use, variants of distance-vector remains popular in resource-constrained settings such as ad-hoc networks and sensor-nets due to its minimal footprint and simplicity. AODV [36], and TORA [37] are widely known examples of on-demand, ad-hoc routing protocols used by mobile networks that are based on the distance-vector protocol.

We show here, how the Strong Detection technique introduced in Sec. III can be applied by a self-monitoring node to check for detectable errors introduced by other nodes. We first briefly outline the operation of the Distance-Vector protocol, then we describe the state-information contained in each node when the protocol is executed. Then we develop a low-complexity algorithm to detect errors and prove that the algorithm does indeed detect all detectable misconfigurations. We illustrate the algorithm by applying it to a small example network that is misconfigured. We conclude with an analysis of the time and space complexity of the algorithm.

Throughout, we illustrate the error-detection process using the example graph of Fig. 6(a). In the figure node n in the example graph is the self-monitoring node.

A. Model

A node n's state in distance-vector is a table of size $|V|$ x $|N(n)|$, whose entry, $d_n(i, j)$ in the ith row and jth column equals the shortest path distance that neighbor n_j claims exists from itself to node i [2].

B. Canonical Graph Construction

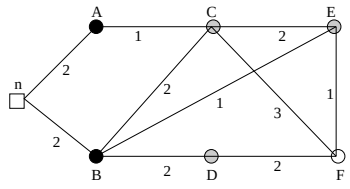
The canonical graph G' is first initialized with all the nodes in the network. Then every pair of nodes i, j in the graph is connected with an edge whose weight $w'(i, j)$ is the smallest value in $S_{i,j}$ that is no less than $\max_{k \in N(n)} |d_n(k, i) - d_n(k, j)|$. [3] If no such value exists (in the case where this maximum is larger than any value in $S_{i,j}$), then the edge is omitted (or, equivalently, set to ∞).

After all edges are constructed, if $G' \in \mathcal{G}$, the distance vector algorithm is simulated on G' producing a state table d'_n for node n. Then d'_n is compared to the original state table d_n within which we are attempting to identify a misconfiguration.

The state of of node n of the example graph of Fig. 6(a) is shown in Table 6(b). The graph G' that results in running the canonical-constructor algorithm on the state table of Table 6(b) is shown in Fig. 6(c). Note that while the original graph G is not a complete graph, the reconstructed graph G' is a complete graph. It can also be verified that running the distance-vector algorithm on either of these graphs produce the same state table of Table 6(b).

[2] An alternate view has $d_n(i, j)$ to be the shortest path distance of node n to node i through neighbor j. The two views can easily be computed from one another.

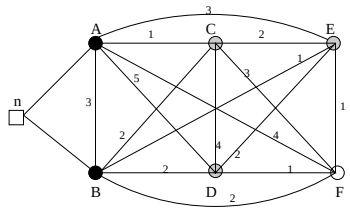
[3] Note that it is this requirement that we choose a value no less than the stated value that forces us to impose the requirement that each interval is closed from below.



(a) Example Graph

	A	B
A	0	3
B	3	0
C	1	2
D	5	2
E	3	1
F	4	2

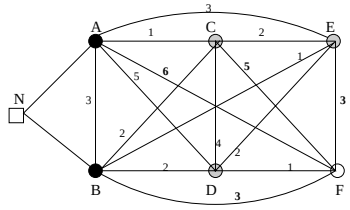
(b) State-Table



(c) Canonical Construction

	A	B
A	0	3
B	3	0
C	1	2
D	5	2
E	3	1
F	**6**	2

(d) State-Table with Error



(e) Canonical Graph with error

	A	B
A	0	3
B	3	0
C	1	2
D	5	2
E	3	1
F	6	**3**

(f) Computed State

Fig. 6. Distance-Vector

C. Proof

Here we provide the theorem and proof that the canonical-graph G constructed in Sec. IV for the distance-vector protocol is indeed the only graph that needs to be considered among all the graphs in $\mathcal{G}$. We present some preliminary results as lemmas before we proceed to proving the main theorem. The reader may skip the proof on first reading without loss of continuity.

Lemma 4.1: If there exists a graph $G \in \mathcal{G}$ with edges of length $w(i,j)$ between nodes i and j so that running the routing protocol correctly on G produces table d_n, then running the routing protocol correctly on the canonical graph G' produces a table d'_n where $d'_n(k,i) \leq d_n(k,i)$ for each $k \in N(n)$ and every node i.

Proof: The proof is by contradiction. Assume there is a graph G that would produce state table d_n, but that a neighbor k exists for which there is a node i where $d'_n(k,i) > d_n(k,i)$ (where a non-existent edge in G' has length ∞). WOLG, select i for which $d_n(k,i)$ is minimized, i.e., i is chosen so that $d'_n(k,i) > d_n(k,i)$ and for any other node j where $d'_n(k,j) > d_n(k,j)$, we have that $d_n(k,i) \leq d_n(k,j)$. Let x be the node on a shortest path from k to i in G that immediately precedes i on this shortest path (x may in fact be node n itself). Since x is closer (no edges of length 0), by our choice of i, we have that $d'_n(k,x) \leq d_n(k,x)$.

Since (x,i) is an edge in graph G, in the accurate state table the shortest path distance from any neighbor to node x can differ from the shortest path distance to node i by no more than $w(x,i)$, i.e., $|d_n(m,x) - d_n(m,i)| \leq w(x,i)$ for every $m \in N(n)$. Thus, $w'(x,i) \leq w(x,i)$ [4].

Utilizing $d'_n(k,x) \leq d_n(k,x)$, $w'(x,i) \leq w(x,i)$, and the fact that the shortest path from k to i in G' is no longer than the path from k to x in G' plus $w'(x,i)$, we have that $d'_n(k,i) \leq d'_n(k,x) + w'(x,i) \leq d_n(k,x) + w(x,i) = d_n(k,i)$, contradicting $d'_n(k,i) > d_n(k,i)$. ■

Lemma 4.2: Let there exist a graph $G \in \mathcal{G}$ so that running the protocol correctly on G produces table d_n and let G' be the canonical graph. Then $d'_n(k,i) \geq d_n(k,i)$ for all neighbors $k \in$ and all nodes i.

Proof: The proof is also by contradiction. Let k be any neighbor for which the claim does not hold and choose i where $d'_n(k,i) < d_n(k,i)$ and for any j where $d'_n(k,j) < d'_n(k,i)$ implies that $d'_n(k,j) > d_n(k,j)$. Let x be the node on the shortest path in the canonical graph G' that precedes i on a shortest path from k to i. By choice of i, we have that $d'_n(k,x) \geq d_n(k,x)$. By construction of G', we have that $w'(x,i) \geq d_n(k,i) - d_n(k,x)$, so $d_n(k,i) \leq d_n(k,x) + w'(x,i) \leq d'_n(k,x) + w'(x,i)$, which, since x lies one hop before i on the shortest path to i, equals to $d'_n(k,i)$, hence $d_n(k,i) \leq d'_n(k,i)$, contradicting our choice of i. ■

Theorem 4.3: In the distance-vector protocol, d_n is a valid state table for some graph $G \in \mathcal{G}$ if and only if it is valid for the distance-vector canonical graph, $G' \in \mathcal{G}$.

Proof: If the state table is valid for no graph $G \in \mathcal{G}$, then clearly it cannot be valid for $G' \in \mathcal{G}$. If it is valid for some graph G, then by Lemmas 4.1 and 4.2, the state table d'_n of G' will match the state table d_n of G. ■

D. Misconfiguration Examples

Now let us consider a situation where there is a misconfiguration and show how it is detected by node N. Suppose that node C erroneously reports to A that its distance to node F is 5 instead of the correct distance 3. Node N's state will change and be as pictured in Fig. 6(d) (A's distances to F has changed because of C's error). Next node N constructs the canonical graph G' from its new state shown in Fig. 6(d) that contains the changes resulting from C's error. This canonical-graph G' constructed according to the algorithm outlined earlier will be as pictured in Fig. 6(e). Note that the weights on edges e_{AF}, e_{CF}, e_{EF} and e_{BF} have changed relative to the canonical-graph of Fig. 6(c) constructed from the correct state. As the final step in the process, node N executes the distance-vector protocol on the canonical-graph

[4]Note that because the Claim assumes the existence of valid graph G and G contains edge $w(x,i)$, it must be the case that $w(x,i) \in S_{x,i}$, and hence there is some value in $S_{x,i}$ larger or equal to $|d_n(m,x) - d_n(m,i)|$ for all m.

G' of Fig. 6(e). This will result in a state d'_N as pictured in Fig. 6(f). Node N will compare this state to its state d_N of Fig. 6(d) and will notice that $d_N(F, B) = 2$ while $d'_N(F, B) = 3$. Since $d_N \neq d'_N$ node N will correctly conclude that there is a misconfiguration!

As a second example let us apply this methodology to the sample state of Fig. 3 we considered earlier where symmetry and the triangle-inequality properties held and the only allowable edge-weight was 1 (i.e. $S_{x,y} = \{1\}$ for all x, y). In that case we concluded, through a complex logical argument, that there could be no valid graph that produced that particular state-table. We illustrate here how we could arrive at the same conclusion using Strong Detection. Applying the canonical-constructor described above to the state-table of Fig. 3 would produce a graph with just one edge between b and c of weight 1. This graph obviously has no path between a and d and therefore if we ran the distance-vector protocol on this graph it would definitely not produce the state table of Fig. 3. We conclude, without the complicated logic needed earlier, that this state is the result of a misconfiguration.

E. Space and Time-Complexity

In performing Strong Detection, the self-monitoring node executes two new procedures: one to create the canonical-graph from its state, and the second to run the routing-protocol on the canonical graph. The canonical graph constructed for distance-vector in Sec. IV is fully-connected so has $|V|$ nodes and $|V| \times |V-1|/2$ edges requiring space of the order of $|V|^2$. Executing the routing-protocol on the canonical-graph requires the state for each node to be stored. This requires space of the order of $k|V|^2$ where k is approximately $3 + 2|V|^{1/d}$. Therefore the overall space-complexity is $O(|V|^{2+1/d})$ with a small proportionality constant.

Constructing each edge requires looking at the information provided by each neighbor. So it has a time-complexity of $d|V|^2$ where d is the number of neighbors, or out-degree. Running the distance-vector routing-protocol on the fully-connected canonical-graph has time complexity of $O(|V|^3)$. This is because at each node routes need be computed for $|V|$ nodes by looking at information provided by $|V-1|$ neighbors. However the canonical graph constructor of Sec. IV that produces a fully-connected graph is used in this paper mainly for ease of exposition. In practice, alternative canonical-constructors that produce a canonical-graph with an out-degree d that is closer to that of the original graph will be used or the fully-connected canonical graph can be pruned off redundant paths. For such graphs where $d << |V|$, executing the protocol has a time complexity of $O(|V|^2)$. Therefore, in practice, the overall time-complexity will be $O(|V|^2)$.

V. Further Work

We plan to extend our work by applying our theory of Strong Detection 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 explore if it would be possible for nodes to *identify* misconfigured nodes in addition to detecting that they exist.

A. Application to other Protocols

In this paper we have applied our theory to the Distance-Vector protocol. We next plan to apply out theory to other popular routing protocols such as Path Vector and Link State.

There has also been a growth in ad-hoc and on-demand networks. A node in such on-demand protocols may *only* have routes to a small subset of nodes in the network. We plan to investigate the application of Strong-Detection to such scenarios with incomplete information. Additionally distributed algorithms have been used in sensor-networks to estimate various parameters at each node by iteratively exchanging information with other nodes within their communication radii [38]–[41]. 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.

B. Identification

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.

VI. Conclusion

We have described our development of a theory that can be used by routers to analyze their routing-states to sense any detectable misconfigurations. We then develop an algorithm that allows this theory to be practically applied to the popular Distance-Vector routing protocol. Our theory additionally allows classes of misconfigurations in a protocol to be classified detectable or undetectable. This allows 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] J. F. Kurose and K. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[2] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding bgp misconfiguration," in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press, 2002, pp. 3–16.

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

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

[5] C. Labovitz, G. R. Malan, and F. Jahanian, "Origins of internet routing instability, Tech. Rep. CSE-TR-368-98, 17, 1998. [Online]. Available: citeseer.nj.nec.com/labovitz99origins.html

[6] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, "Fatih: Detecting and Isolating Malicious Routers," in *Proc. of the IEEE Conference on Dependable Systems and Networks (DSN)*, June 2005.

[7] P. Verissimo, "Design of fault tolerant distributed systems: the fail-controlled approach," in *EW 4: Proceedings of the 4th workshop on ACM SIGOPS European workshop*. New York, NY, USA: ACM Press, 1990, pp. 1–4.

[8] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne:: a secure on-demand routing protocol for ad hoc networks," in *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM Press, 2002, pp. 12–23.

[9] Y.-C. Hu, A. Perrig, and M. Sirbu, "Spv: secure path vector routing for securing bgp," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2004, pp. 179–192.

[10] W. Aiello, J. Ioannidis, and P. McDaniel, "Origin authentication in interdomain routing," in *Proceedings of the 10th ACM conference on Computer and communication security*. ACM Press, 2003, pp. 165–178.

[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*. ACM Press, 2003, pp. 30–40.

[12] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (bgp)," *IEEE Journal on Selected Areas of Communication*, vol. 18, no. 4, pp. 582–592, April 2000.

[13] B. Smith and J. Garcia-Luna-Aceves, "Securing the border gateway routing protocol," in *Proc. Global Internet'96*, November 1996. [Online]. Available: citeseer.ist.psu.edu/smith96securing.html

[14] H.-Y. Chang, S. F. Wu, and Y. F. Jou, "Real-time protocol analysis for detecting link-state routing protocol attacks," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 1, pp. 1–36, 2001.

[15] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz, "Listen and whisper: Security mechanisms for bgp," in *Proceedings of First Symposium on Networked System Design and Implementation (NSDI 2004)*, March 2004. [Online]. Available: citeseer.ist.psu.edu/668829.html

[16] L. Subramanian, R. H. Katz, V. Roth, S. Shenker, and I. Stoica, "Reliable broadcast in unknown fixed-identity networks," in *PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*. New York, NY, USA: ACM Press, 2005, pp. 342–351.

[17] V. N. Padmanabhan and D. R. Simon, "Secure traceroute to detect faulty or malicious routing," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 77–82, 2003.

[18] S. Kandula, D. Katabi, and J.-P. Vasseur, "Shrink: a tool for failure diagnosis in ip networks," in *MineNet '05: Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data*. New York, NY, USA: ACM Press, 2005, pp. 173–178.

[19] A. Orda, R. Rom, and N. Shimkin, "Competitive routing in multiuser communication networks," *IEEE/ACM Trans. Netw.*, vol. 1, no. 5, pp. 510–521, 1993.

[20] R. K. Rajendran, V. Misra, and D. Rubenstein, *"BRIEF ANNOUNCEMENT* strong detection of misconfigurations," *Principles of Distributed Computing (PODC)*, p. 40, July 2005.

[21] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding bgp misconfiguration," in *Proceedings of ACM SIGCOMM 2002.*, 2002. [Online]. Available: citeseer.ist.psu.edu/mahajan02understanding.html

[22] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515–528, 1998. [Online]. Available: citeseer.nj.nec.com/labovitz97internet.html

[23] 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*. ACM Press, 2002, pp. 203–208.

[24] 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*. ACM Press, 2003, pp. 113–124.

[25] T. G. Griffin and G. T. Wilfong, "An analysis of BGP convergence properties," in *Proceedings of SIGCOMM*, Cambridge, MA, August 1999, pp. 277–288. [Online]. Available: citeseer.nj.nec.com/griffin99analysis.html

[26] N. Feamster, J. Borkenhagen, and J. Rexford, "Controlling the impact of bgp policy changes on ip traffic," in *NANOG25*, 2002. [Online]. Available: citeseer.ist.psu.edu/feamster01controlling.html

[27] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek, "Measuring the effects of internet path faults on reactive routing," in *Proc. of ACM SIGMETRICS 2003, San Diego, CA*, Jun 2003. [Online]. Available: citeseer.ist.psu.edu/feamster03measuring.html

[28] N. Feamster, R. Johari, and H. Balakrishnan, "Implications of autonomy for the expressiveness of policy routing," in *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2005, pp. 25–36.

[29] L. Gao, "On inferring autonomous system relationships in the internet," in *Proc. IEEE Global Internet Symposium, November 2000.*, 2000. [Online]. Available: citeseer.ist.psu.edu/gao00inferring.html

[30] P. Radoslavov, H. Tangmunarunkit, H. Yu, R. Govindan, S. Shenker, and D. Estrin, "On characterizing network topologies and analyzing their impact on protocol design," University of SOuthern California, Tech. Rep. USC-CS-TR-00-731, Mar. 2000. [Online]. Available: citeseer.ist.psu.edu/radoslavov00characterizing.html

[31] R. Govindan and H. Tangmunarunkit, "Heuristics for internet map discovery," in *IEEE INFOCOM 2000*. Tel Aviv, Israel: IEEE, March 2000, pp. 1371–1380. [Online]. Available: citeseer.ist.psu.edu/govindan00heuristics.html

[32] G. Siganos and M. Faloutsos, "Analyzing bgp policies: Methodology and tool," in *IEEE INFOCOM 2004*. Hong Kong: IEEE, 2004.

[33] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, "Characterizing the internet hierarchy from multiple vantage points," in *IEEE INFOCOM 2002*. New York, NY: IEEE, June 2002.

[34] D. Pei, D. Massey, and L. Zhang, "Detection of invalid routing announcements in rip protocol," in *Proc. of IEEE Globecom, San Francisco, CA*, Dec 2003.

[35] G. Malkin, "Routing Information Protocol Version 2," RFC 2453, November 1998.

[36] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," United States, 2003.

[37] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *INFOCOM*, 1997, pp. 1405–1413. [Online]. Available: citeseer.ist.psu.edu/park97highly.html

[38] 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. [Online]. Available: citeseer.ist.psu.edu/romer01time.html

[39] 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. [Online]. Available: citeseer.ist.psu.edu/elson02fine grained.html

[40] 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)*, Aug. 2000, pp. 120–130. [Online]. Available: citeseer.ist.psu.edu/li00scalable.html

[41] A. Savvides, C.-C. Han, and M. B. Strivastava, "Dynamic fine-grained localization in ad-hoc networks of sensors," in *Mobile Computing and Networking*, 2001, pp. 166–179. [Online]. Available: citeseer.ist.psu.edu/savvides01dynamic.html