Theoretical Bounds on Control-Plane Self-Monitoring in Routing Protocols

Raj Kumar Rajendran
Dept. of Electrical Engineering
Columbia University
Email: raj@ee.columbia.edu

Vishal Misra
Dept. of Computer Science
Columbia University
Email: misra@cs.columbia.edu

Dan Rubenstein
Dept. of Electrical Engineering
Columbia University
Email: danr@ee.columbia.edu

***Abstract*— Routing protocols rely on the cooperation of nodes in the network to both forward packets and to select the forwarding routes. There have been several instances in which an entire network's routing collapsed simply because a seemingly insignificant set of nodes reported erroneous routing information to their neighbors. It may have been possible for other nodes to trigger an automated response and prevent the problem by analyzing received routing information for inconsistencies that revealed the errors. Our theoretical study seeks to understand when nodes can detect the existence of errors in the implementation of route selection elsewhere in the network through monitoring their own routing states for inconsistencies. We start by constructing a methodology, called Strong-Detection, that helps answer the question. We then apply Strong-Detection to three classes of routing protocols: distance-vector, path-vector, and link-state. For each class, we derive low-complexity, self-monitoring algorithms that use the routing state created by these routing protocols to identify any detectable anomalies. These algorithms are then used to compare and contrast the self-monitoring power these various classes of protocols possess. We also study the trade-off between their state-information complexity and ability to identify routing anomalies.**

I. Introduction

Routing protocols enable a distributed set of nodes to determine the flow of data over important networks such as the Internet. As a result, ensuring that nodes throughout the network properly implement the routing-protocol is of paramount importance. The routing-protocols in these networks are distributed, and nodes operate independently, but must cooperate and "play by the rules" of the routing protocol if the network is to function correctly and efficiently.

A node takes on two sets of responsibilities when it participates in a distributed routing protocol. Within the control plane, it is responsible for *route establishment*: identifying the paths through which packets flow across the network. Within the data plane, it is responsible for the *forwarding* of packets along these computed routes. Routing can suffer if either of these responsibilities are mishandled, whether intentionally or by accident. This paper focuses on the control plane.

One or more nodes that misimplement route establishment can cause widespread damage. The most infamous example is the AS7007 incident in which AS7007, running the BGP protocol, announced very short, inaccurate routes to most of the Internet [21]. For over two hours this disrupted connectivity to large tracts of the Internet. Despite the publicity this incident generated, it is clear that serious anomalous behavior continues to occur. More recently, AS3561 propagated more than 5000 improper route announcements again leading to global connectivity problems [4]. These accidents demonstrate the need to guard against disruption in a network caused by misimplementations of the route establishment portion of a protocol.

One solution is to introduce additional mechanisms that verify the correctness of the selected routes. However, it may be difficult to deploy such modifications within existing protocols. Since good design philosophies seek to utilize all available information and avoid inessential mechanisms we attempt to understand and use the inherent capabilities routing protocols possess for self-monitoring. More to the point, when one considers a routing protocol in its current form, can analysis of the information that is exchanged between nodes during route establishment detect errors elsewhere in the network? If so what are the kinds of errors that can be detected?

Our goal in this paper is to answer these questions by quantifying the intrinsic capability for self-monitoring that different routing-protocols possess and to show how this capacity for introspection can be harnessed. Toward this we contribute the following:

- We expand on the notion of Strong Detection proposed in [28] and show how it can be used to detect *all* errors detectable through self-monitoring. We show that errors not detectable through Strong Detection *cannot* be detected without additional information by any other technique.
- We construct low-complexity algorithms that implement Strong Detection for well-known classes of routing protocols.
- We develop computational measures that quantify the self-monitoring ability of protocols and use them in a simulated evaluation to compare popular classes of routing protocols.

Our contribution in this paper is theoretical: we look at general classes of routing protocols and show how they can be analyzed for their ability to monitor themselves. We use Strong Detection to reveal "bounds" on the kinds of errors that these classes of routing protocols can detect. Hence, we are identifying "complexity classes" of routing protocols in terms of their self-monitoring abilities. We also develop measures that provide insight into the relative state-complexity and self-monitoring ability of protocols. We hope our work provides a rough guide to the relative self-monitoring capacities of the different protocols and will help in the design of robust protocols and in choosing protocols that provide the correct balance between complexity of state-information and resistance to corruption.

The rest of the paper is organized as follows. We discuss related work in Sec. II and introduce the theory of Strong Detection in Sec. III. In Sec. IV we present the distance-vector, path-vector and link-state routing-protocols and provide practical implementations of Strong Detection for them. In Sec. V we provide experimental verifications of our self-monitoring techniques through simulated evaluations. We conclude our paper with Sec. VI.

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 [22] 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 [35] and use centralized public-key infrastructures or key-distribution mechanisms to address the problem [11], [13], [1], [12], [15], [30], [3]. However these works do not attempt to harness the self-monitoring capabilities that protocols possess.

The authors in [34], [33] 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 works [24], [14] 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 [23] 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 [28] 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 [18] and yet others that aim to detect misconfigurations through observance of dynamic behavior [16], [2], [31], [10], [6], [5], [22], [7].

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 [8], [27], [9], [29]. Other work is willing to take advantage of simultaneous analysis of multiple perspectives [32], where, for instance, 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 IV-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 some ground rules that we apply to keep the theoretical problem both tractable and well-specified:

- Stability: we assume that the analysis is conducted after routes have stabilized. In real networks whose underlying topology may continually change, routes are continually re-evaluated to account for changes in the underlying topology. Our assumptions rule out our ability to detect "dynamic" errors that can be induced by other types of misconfigurations.
- Detection, not identification: Our goal here is to design techniques that detect *the existence* of misconfigurations somewhere in the network. We show that sometimes, even detection cannot be achieved. Identifying the offender of the misconfiguration once such a misconfiguration is detected is a separate problem beyond the scope of this paper.
- Protocol Fixing: We evaluate the self-monitoring abilities of particular routing protocols *as they exist*. One could ask "what if the monitors somehow obtained some additional information beyond the information gathered in their respective states?". For instance, what if forwarding along a route was attempted and the packet never reached the destination? Clearly, additional information might enable identification of misconfigurations that we show cannot be identified. Our view is that this additional information obtained should be viewed as an enhancement to the routing protocol.
- Correctness of Monitor: We assume that the monitor node which does the evaluation operates correctly. Clearly, we cannot make any claims about the validity of our techniques when applied by misconfigured nodes.

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 [25], it is shown that the triangle inequality can be applied within RIP [19] (a specific Distance Vector implementation) to detect certain misconfigurations.

A shortcoming of this previous method is that it identifies a specific property (in the case above, the property is the triangle inequality) 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
a	0	2
b	2	0
c	3	1
d	3	3

Fig. 1. Node N's state

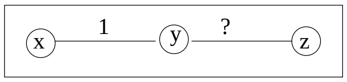


Fig. 2. Undetectable Report

Consider the example state-table in Fig. 1 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. 1 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 using 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 1. 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 using 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* [28] 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. Note however that there are some misconfigurations that are impossible to detect, even by Strong Detection techniques. Consider the simple situation pictured in Fig. 2 where node z's only edge goes to node y. Suppose node x wishes to determine whether or not node y is reporting accurate information.Since information about the edge e_{yz} has to *always* propagate through node y, node y can choose to report any distance to z. Hence if x is the monitor it cannot detect a misconfiguration at y.

A graphical depiction of Strong and Weak detection is presented in Fig. 3. The $x-y$ plane 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. Known Weak Detection techniques, such as the triangle-inequality, can be used to check for subsets of the detectable misconfigurations. There may be Weak Detection techniques, unknown as of now, that may detect other subsets of detectable misconfigurations. However, Strong Detection, by definition must detect *all* detectable misconfigurations. By definition, the region covered by Strong Detection equals the detectable region, and thus encompasses the union of what is detectable by weak detection.

[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.

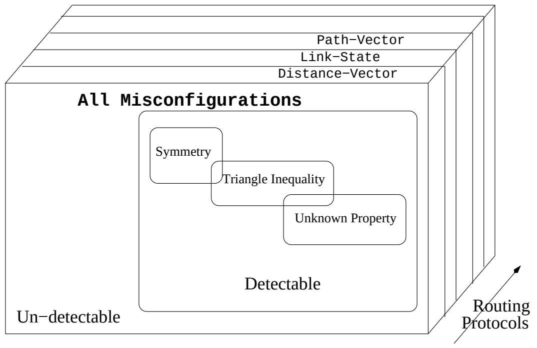


Fig. 3. The Space of Misconfigurations

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.

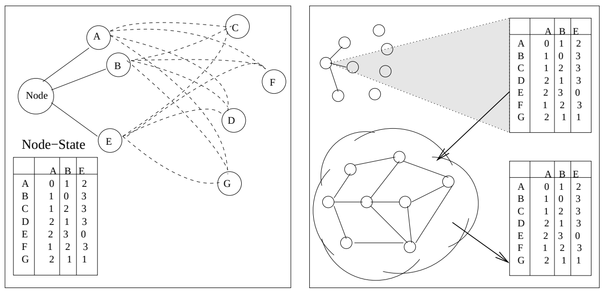
The problem with Strong Detection, as described above, is the time needed to either find a G that produces a matching state, or the (potentially infinite) time needed to demonstrate that there is no matching graph.

C. Computationally Feasible Strong Detection

We now describe in generality 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 the *canonical graph*. Node n with state d_n pictured in Fig. 4(a) runs the following procedure:

- n executes an algorithm (the details of the algorithm are specified later in the paper) 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. 4(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. 4(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 protocols we consider.



	A	B	E
A	0	1	2
B	1	0	3
C	1	2	3
D	2	1	3
E	2	3	0
F	1	2	3
G	2	1	1

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

Fig. 4. 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 describe how the canonical graph G' mentioned in Sec. III is constructed for each of the protocols listed in Sec. IV.

IV. The Protocols

In this section we study a number of routing protocols and show, for each, 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.

For each protocol, we outline its operation, describe the state-information contained in each node, present a low-complexity algorithm to detect errors and prove that the algorithm does indeed detect all detectable misconfigurations. We study the following protocols, illustrating the error-detection process for each with the example graph of Fig. 5(a). In all figures node n in the example graph is the self-monitoring node.

- Distance-Vector
- Path-Vector with hop-by-hop distance
- Path-Vector with total distance
- Path-Vector with Incomplete Information
- Link-State

A. Distance Vector

The popular RIP protocol was [19] 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 because of its simplicity and minimal footprint. AODV [26] is a widely known on-demand version of the distance-vector protocol that is used by mobile nodes in ad-hoc networks.

1) 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].

2) 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. 5(a) is shown in Table 5(b). The graph G' that results in running the canonical-constructor algorithm on the state table of Table 5(b) is shown in Fig. 5(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 5(b).

Theorem 4.1: 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, $G' \in \mathcal{G}$.

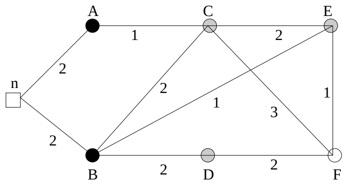
The proofs can be found in the Appendix.

3) 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. 5(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. 5(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. 5(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. 5(c) constructed from the correct state. As the final step in the process, node N executes the distance-vector protocol on the canonical-graph G' of Fig. 5(e). This will result in a state d'_N as pictured in Fig. 5(f). Node N will compare this state to its state d_N of Fig. 5(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. 1 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

[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.

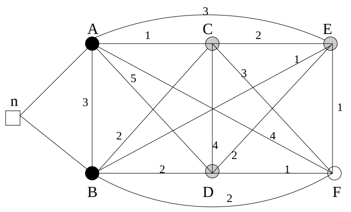
[3]Note that it is this requirement that we choose a value no less than the stated value that forces us to impose the requirement that each interval is closed from below.



(a) Example Graph

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

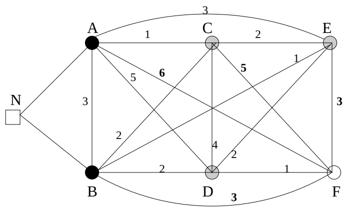
(b) State-Table



(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. 5. Distance Vector

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. 1 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. 1. We conclude, without the complicated logic needed earlier, that this state is the result of a misconfiguration.

4) 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-A 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-A 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 $O(|V|^2)$.

B. Path Vector with hop-distances

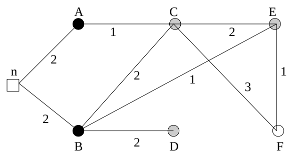
In the path-vector routing algorithm, each node maintains a path (a list of nodes to be traversed) to each node in the network. Periodically each node exchanges *some* paths with each neighbor and recomputes its most-desirable path to each node in the network based on its individual *policies*. Therefore each node's state or view consists of the most-desirable paths each of its neighbors claims to other nodes in the network. We first consider a version of the path-vector routing algorithm where neighbors exchange complete paths to destinations and additionally the hop-by-hop distance of these paths.

	A		B	
	d	p	d	p
A	0	-	2,1	C,A
B	1,2	C,B	0	-
C	1	C	2	C
D	1,2,2	C,B,D	2	D
E	1,2	C,E	1	E
F	1,3	C,F	1,1	E,F

(a) Path Vector with hop-distances

	A		B	
	d	p	d	p
A	0	-	3	C,A
B	3	C,B	0	-
C	1	C	2	C
D	5	C,B,D	2	D
E	3	C,E	1	E
F	4	C,F	2	E,F

(b) Path Vector



(c) Canonical Graph

Fig. 6. Path Vector

1) Model: Consider the same example network used earlier and pictured in Fig. 5(a). The state of node n is as shown in Fig. 6(a) for this network. Note that in this case, each node n's state consists, as before, of 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 n_i. However each $d_n(i,j)$ now consists of two vectors of length k and $k+1$, $h_0, h_1, \ldots, h_k$ and $v_1, \ldots, v_k$ where $h_i \in V$ and $v_i \in W$. The first vector is the ordered set of nodes the shortest path from i to j takes while the second vector represents the weights of the associated edges.

2) Canonical Graph Construction: Constructing the canonical graph for such a state table is straightforward. For each path reported by any neighbor, edges with the appropriate weights are added if they don't exist. Note that all edges in the original graph that do not participate in a path will be excluded from the canonical graph G'. A contradiction can arise during the construction if multiple edges of

```
• Initialize $G'$ with $V$
• For each $d_n(i,j)$ do
    – For each $v_i$ in $d_n(i,j)$ do
        ∗ Add edge $e'_{h_{i-1},h_i}$ with weight $v_i$ if it does not already exist.
        ∗ If edge $e'_{h_{i-1},h_i}$ exists with weight not equal to $v_i$ return error.
    – done (for each $v_i$)
• done (for each $d_n$)
```

Fig. 7. Canonical Construction Algorithm for Path Vector

varying lengths need to be constructed between two nodes. Such a contradiction implies that the state table is erroneous. The algorithm is presented in Fig. 7.

The canonical graph constructed from the state-table pictured in Fig. 6(a) is pictured in Fig. 6(c).

Theorem 4.2: In the path-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 path-vector canonical graph, $G' \in \mathcal{G}$.
The proof is given in the Appendix.

3) Space and Time-Complexity: We consider the space and time complexity of the self-monitoring process here. The canonical graph constructed for path-vector in Sec. IV-B.2 has approximately $|V|$ nodes and $l \times |V|$ edges where l is the average path-length and $l \approx |V|^{1/d}$. Therefore the graph requires space of the order of $|V| \times |V|^{1/d}$. Executing the routing-protocol on the canonical-graph requires space of the order of $|V|^2 * |V|^{1/d}$. Therefore the overall space-complexity is $O(|V|^2 \times |V|^{1/d})$.

Constructing the canonical graph requires looking at each edge in all the paths, so has a time complexity equal to the number of edges in all paths provide so is $l \times |V|$. Running the path-vector routing-protocol on the canonical-graph has time complexity of $O(d|V|^2)$ since at each of $|V|$ nodes, routes need be computed to $|V|$ nodes by looking at information provided by d neighbors. Therefore the overall time-complexity will be $O(|V|^2)$.

C. Path Vector

BGP is an example of a protocol that uses path-vector routing. In BGP a node (known as an AS) computes 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 other nodes in the network. We consider a simplified path-vector routing algorithm where the shortest-paths are the most desirable paths.

1) Model: Here, neighbors exchange the total distances and complete-paths to destinations but do *not* provide the hop-by-hop distance. That is is, they exchange the sequential list of nodes than must be traversed to reach a destination and the total distance of the complete path but not the individual hop weights. Such a state is pictured in Fig. 6(b).

Even though, at first glance, it may seem that there is less information in this state than in the state of a node where the hop-by-hop distance is also provided, it can be seen that the two states provide identical information. This is because the individual hop weights can be inferred from the total distance as follows. If neighbor N_i reports a distance $d(N_i, X)$ to node X with path $N_i, \ldots, Y, X$ then it must be the case that the path to Y must be $N_i, \ldots, Y$, since if a shorter path existed to Y then the path to X through Y would also include that path. Therefore w_{XY} can be computed as $d(N_i, X) - d(N_i, Y)$ for all X, Y, in any reported path.

2) Canonical Graph Construction: Therefore the canonical graph is the same as for the case of path-vector with hop-by-hop distance and is also pictured in Fig. 6(c).

3) Space and Time-Complexity: Since individual edge weights have to be computed from total path lengths, an additional $l|V|$ operations need to be performed relative to the algorithm for path-vector with hop-by-hop distance. This is small compared to the overall time-complexity.

D. Path Vector with Incomplete-Information

We now consider the path-vector algorithm, but in the case where paths and total-distances are known to only a subset of all the nodes in the network. This is the case in some on-demand path-vector algorithms, where paths to a destination node are computed only when a packet needs to be delivered to the destination. In dynamic and mobile environments paths to a destination that are older than preset limit are considered stale and are discarded and are only regenerated when a new packet arrives for that destination. Such strategies lead to a situation where a node possesses valid paths to some subset of the nodes in its network.

1) Model: In this version of the path-vector algorithm, each node knows the paths and the total distance to a subset $M \subset N$ of the nodes in the network. Such a state is pictured in Fig. 8(b). Even though there are five non-neighbor nodes B, C, D, E, F, distances and paths to only the two nodes B and D are part of the state.

Unlike the "Path-vector with total-distance to all nodes" algorithm, in this case the individual link-weights cannot always be computed. Consider the example network of Fig. 8(a) and the resulting state of Table 8(b) for a network where all links e_{ij} have $S_{i,j}$, the set of allowable weights, to be the real values in the range from 2 to 3 ($2 \leq w_{ij} \leq 3$). From the state table we know that

$$w_{ac} + w_{ce} + w_{eb} = 8 \tag{1}$$

and

$$w_{ac} + w_{ce} + w_{ef} + w_{fd} = 10 \tag{2}$$

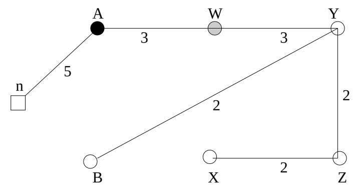
and from the $S_{i,j}$ we further know that $2 \leq w_{ac} \leq 3$, $2 \leq w_{ce} \leq 3$, $2 \leq w_{eb} \leq 3$, $2 \leq w_{ef} \leq 3$, $2 \leq w_{fd} \leq 3$. We have five unknowns, two equations and five ranges for the five unknowns. The value of the five variables cannot be uniquely determined from the two equations and therefore the correct weights for the links cannot be determined.

2) Canonical Graph Construction: The canonical graph in this situation with incomplete information can be constructed if we can determine allowable values for each edge-weight ($w_{ij} \in S_{i,j}$), such that they satisfy the total distance requirement for each known path. So to determine the canonical path we need to find a *feasible solution* for the linear set of equations given by the state-table with the requirement that each edge-weight also satisfies its bound $S_{i,j}$.

For the case where $S_{i,j}$ is an interval on the real-line $x < w_{ij} < y, x, w_{ij}, y \in \Re$ (i.e. each edge weight is known to belong to an interval on the real-line) the problem reduces to finding an initial feasible solution to a linear-programming problem. [4] The first-phase of the Simplex algorithm produces such a solution [17] and so can be used to generate the canonical graph G'. Since the Simplex algorithm is well known, we omit the details of computation and just show one feasible canonical graph for the state of Fig. 8(b) in Fig. 8(c).

Theorem 4.3: In the path-vector protocol with incomplete-information where each edge weight is constrained to an interval on the real-line ($x < w_{ij} < y$ where $x, w_{ij}, y \in \Re$), d_n is a valid state table for some graph $G \in \mathcal{G}$ if and only if it is valid for the canonical-graph for Path-Vector with Incomplete-Information, $G' \in \mathcal{G}$.
The proof is simple and is omitted.

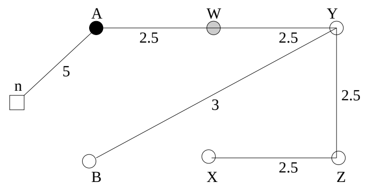
[4]In general our canonical-graph constructions are valid for a more general $S_{i,j}$.



(a) Example Graph

	A_d	A_n
A	0	-
B	8	C,E,B
D	10	C,E,F,D

(b) State-Table



(c) Feasible Graph

Fig. 8. Path-Vector with Incomplete Information

3) Space and Time-Complexity: The space and time complexities are the same as that for the path-vector case except that $|V|$ in this case is the number of nodes to which paths are known (rather than all nodes in the network).

E. 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 links (or list of neighbor nodes and weights) with each neighbor. The process stops when each node has the complete list of neighbors (and weights) for each node in the network. From this information each node can recreate the entire network. Therefore in the link-state protocol each node's state is a snapshot of the entire graph. This property trivializes the analysis of detecting misconfigurations for link-state protocols. In fact, we have that $G' = G$. 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 its state S_i. The only graph in this set would be G. In fact, if $n_1, n_2, \cdots, n_k$ are neighbors of i and all neighbor's states match i's state (i.e, $S_{n_j} = S_i$ 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, $S_x \neq S_y$ then i will detect a misconfiguration.

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

Theorem 4.4: Let $G = (N, E, W)$ be a graph running the link-state protocol where a subset of nodes, $N' \subset N$ are properly configured. Then a misconfiguration is detectable in the 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 $1 \leq j \leq k$.

F. Asymmetric Links

We have considered networks with symmetric edges until now. In cases where the underlying network has asymmetric links, the canonical graphs for the path-vector protocol proceed as for the symmetric case except that the edges are directed. The canonical graph for distance-vector proceeds similar to the case for symmetric case with a couple of differences: the monitor's neighbor nodes are connected to it with directed edges of the appropriate direction. Additionally every pair of non-neighbor nodes i, j in the graph is connected with a directed edge in each direction whose weight $w'(i, j)$ is the smallest value in $S_{i,j}$ that is no less than $\max_{k \in N(n)} d_n(k, i) - d_n(k, j)$ (in undirected graphs we took the absolute value of the difference between $d_n(k, i)$ and $d_n(k, j)$). The proof that this method indeed produces the canonical graphs for asymmetric distance-vector is similar to the proof for symmetric distance-vector.

G. Multiple Perspectives

We have emphasized, until now, that our detection technique can be applied to the protocol without modification or additional information. We now consider how the same technique can be applied when a network has several monitor nodes and the monitor nodes decide collaborate with each other in detecting misconfigurations. Sharing state information with other trusted monitors can help perform a more thorough and complete inspection.

We consider a situation where a subset of trusted monitor nodes S in the network have a trusted out-of-band communication channel available to them to communicate with each other. They can use this trusted channel to provide each other their state information. Therefore each node has a state $d_S = \cup_{n \in S} d_n$ or a view of the network that is the union of the states of the individual nodes. With this additional information each node can perform better detection that with just its own perspective. The procedure to check for misconfigurations proceeds as before except that d_S is used to construct the canonical graph rather than the d_n earlier.

V. Simulated Evaluation

In this section we present the results of simulation experiments that study and distinguish the kinds of implementation errors that can and cannot be detected through self-monitoring. We device a series of experiments and apply them to each class of protocols in turn. Our experiments show that there is a clear increase in the self-monitoring ability of path-vector protocols over distance-vector. Additionally there is a clear correlation between the detectability of an anomaly and the distance between the monitor and the liar. It is also clear that erroneous information about distances to multiple nodes are more easily detectable than isolated errors.

A. Experimental Setup

In our experiments we used five different networks with 50, 100, 200, 400 and 800 nodes each. The networks were generated by the BRITE topology generator [20] which attempts to create synthetic topologies that accurately reflect the actual Internet topology with respect to aspects such as hierarchical structure and degree distribution. All networks are *undirected* graphs with flat AS-Level topologies constructed using the Barabasi model. The nodes were placed on the X-Y plane according to a heavy-tailed distribution (details can be found at [20]). The edge-weight of a link between two nodes is set to be the Euclidean distance between the two nodes. The characteristics of the generated networks are given in Table I.

In all experiments one node, referred to as the monitor, is the self-monitoring node and another node, referred to as the liar [5] publishes erroneous information to its neighbors. The corrupt node (the liar) can propagate incorrect information in one of two ways: it can *understate* or *overstate* its distance to some node. In both these instances we

[5]The words liar used here is not entirely accurate as a node may misstate distances intentionally or inadvertently.

Nodes	Edges	Degree(%)					
		Avg.	1	2	3	4	5
50	97	3.9	0	58	14	6	6
100	197	3.9	0	57	11	8	7
200	397	4.0	0	50	21	8	6
400	797	4.0	0	49	20	8	8
800	1597	4.0	0	50	19	9	5

TABLE I
NETWORK CHARACTERISTICS

attempt to determine how large the misstatement needs to be before it is detected by the monitor. We call these the negative and positive detection-thresholds T^- and T^+.

To determine these thresholds we conduct a series of experiments where we decrease the magnitude of the misstatement until we reach the point at which it is clearly not detected. We first test a 100% misstatement and if this is not detected we set the threshold to be 100%. If the 100% misstatement is detected we make the misstatements smaller in increments of approximately 10% and note the point at which it is no longer detected. We set this point to the the detection threshold. Note that if none of the misstatements are detected T^- is set to -100% and T^+ is set to +100%.

In addition to detection-thresholds, we recorded what we call *route-change detection*. Misstatements by a node can change node's states without changing their routing (i.e. the change are not significant enough to warrant routing through a different node). We therefore recorded how often such a significant misstatement by the liar was detected at the monitor; i.e. the fraction of times that a misstatement that changed the monitor's routing was detected.

We conducted two series of experiments for each protocol. In our first series of experiments the corrupt node misstates its distance to a *single* node. In these experiments we studied how the detection thresholds were affected by three different distances: the distance between monitor and liar, the distance between liar and lied-about, and the distance between monitor and lied-about. In the second series of experiments, the corrupt node misstated its distance to *all nodes* in the network simultaneously by a constant factor. We varied this factor and determined the detection-thresholds as earlier. We again charted the detection thresholds T^- and T^+ as a function of the distance between the monitor and liar. In all experiments node 0 was arbitrarily chosen to be the monitor, but the node that played the role of the liar varied. In both experiments the liar was chosen at random so that the distance between the monitor and liar was uniformly distributed between 0% and 100% of the maximum distance between the monitor and the node furthest from it in the network. In each of our experiments we computed the detection thresholds by setting different nodes to be the liar and repeating the experiment between 100 and 500 times depending on the size of the network. The node that was lied-about was always chosen randomly in a uniform fashion from all the nodes in the network. In all cases we conducted the experiments on more than one of the networks with 50,100,200,400 and 800 nodes mentioned above, to account for the effect of network-size and characteristics on the ability of protocols to detect corrupt nodes.

B. Distance Vector (DV)

Our first experiment for the distance-vector protocol charts the understatement and overstatement detection thresholds as a function of the distance between monitor and liar when the liar misstates its distance to a single node. The understatement and overstatement thresholds are charted as a function of the distance between monitor and liar, measured in hops, for the four of size 100,200,400 and 800 nodes in Fig. 9. The results of the same experiment where the distance between monitor and liar is measured as a percent of the maximum distance to any node are shown in Fig. 10 for networks of size 50, 100 and 200 nodes.

In both these figures, the curves at the top half of the charts indicate T^+ the overstatement-threshold, while the plots at the bottom half indicate T^- the understatement-threshold. The area above and below the two respective sets of curves indicate detected misstatements while the area between the two sets of curves indicate undetected misstatements.

An effect that is seen is that, as the distance between the liar and monitor increases, it is less likely that the lie will be detected. This is because a lie is less likely to reach the monitor if the liar is further away. As the distance between two nodes increase, it is less and less likely that their paths to other nodes go through each other. Interestingly the plots in Fig. 9 also indicate that the detectability of understated lies falls off less quickly in larger networks than in smaller networks. The relationship between the parameters of the network (connectivity, out-degree distribution), network size and detectability is beyond the scope of this paper but is an interesting area for future investigation.

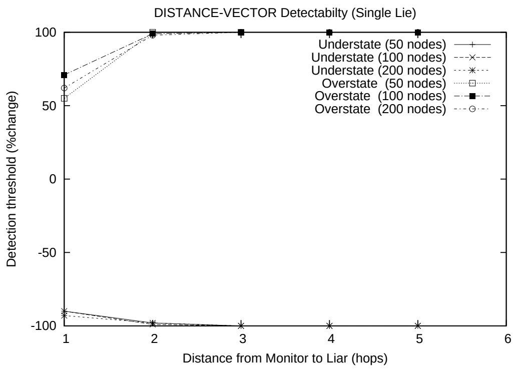


Fig. 9. DV: Misstatement to Single Node (hops)

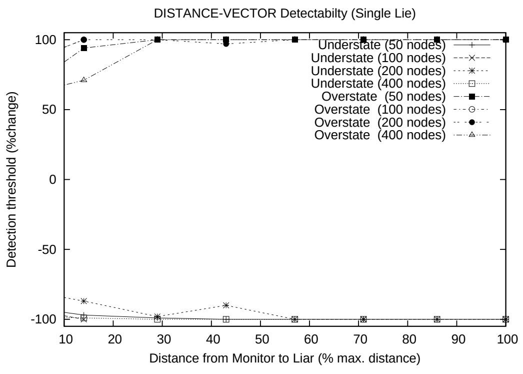


Fig. 10. DV: Misstatement to Single Node (relative distance)

In the next chart (Fig. 11) we plot the detection-thresholds as a function of three distances: distance between monitor and liar, liar and lied-about, and monitor and lied-about for the network with 400 nodes. We notice that detection-thresholds monotonically increase only when the distance is between the monitor and the liar. The distance between the monitor and lied-about and the distance between liar and lied-about are only weakly correlated to detection. These results indicate that in large networks, it may be necessary to have multiple nodes applying Strong Detection techniques to ensure that there is at least one monitor within a short distance of a potentially corrupt node.

In our second series of experiments we chart detection-thresholds as a function of the hop-distance between liar and monitor when the liar misstates distances to all nodes. The liar in this case simultaneously misstates its distance to all nodes in the network by a constant factor. We again determine the detection-thresholds at which these simultaneous lies become perceptible by the monitor. The results are charted in Fig. 12. By comparing it to Fig. 9 where the liar misstated its distance to just a single node, it is clear that such lies are more detectable. The area above and below the overstatement and understatement curves is relatively smaller and the area between between the curves is relatively larger. While the general trend in Fig. 12 is clear, there is a kink in the detectability of the 200 node graph. We also noticed this kink in other experiments on the 200 node graph. Increasing the number of sampling points did not get smoothed out this kink. This leads us to believe that local topological details of a network can affect detectability in that area.

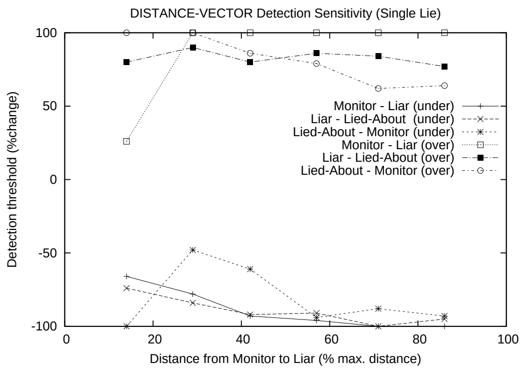


Fig. 11. DV: Effect of distances on detection

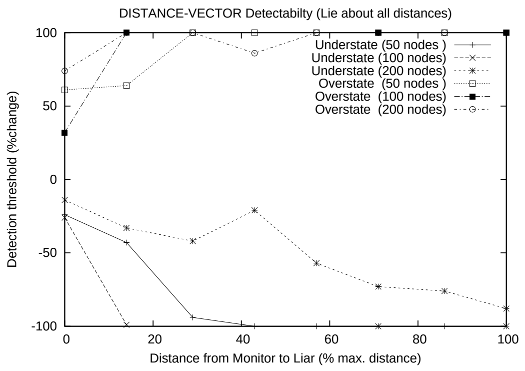


Fig. 12. DV: Misstatement to All Nodes

Since misstatements by nodes sometimes change the distances to target nodes at the monitor but do not change its routing, we charted the efficiency of self-monitoring in detecting such significant misstatements. The fraction of lies that changed routing and were detected is charted in Fig. 13. It should be noted that this measures only the fraction of misstatements that changed routing at the monitor node and were detected. It is possible that some misstatements did not change routing at the monitor but did change routing at other nodes.

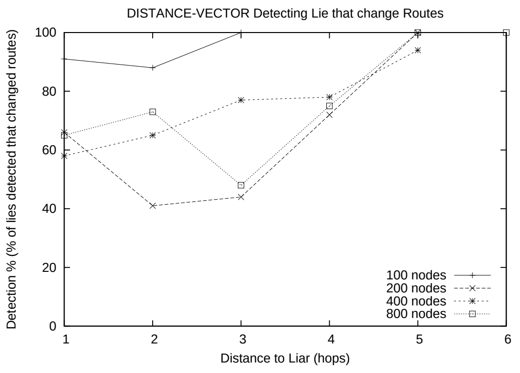


Fig. 13. DV: Detecting route-changing errors

C. Path Vector (PV)

We next show the results of our experiments for the Path Vector protocol. The detection-thresholds when a corrupt node misstates distances to a single node is charted in Figs. 14 and 15 as a function of the distance between monitor and liar. The distances are measured in hops in Fig. 14 and as a percent of distance to the furthest node in Fig. 15. It is clear from these graphs that anomalies are more easily detected in path-vector than distance-vector and that the distance between the monitor and liar clearly affects the monitor's ability to detect anomalies.

It can be seen that overstatements are detected less often. One reason for this is that overstatements do not propagate as much as understatements. If a node overstates its distance to a node, other nodes will ignore the path through the liar and use other paths. However if a liar understates its distance, the path through the liar becomes attractive and attracts routes.

We again noticed the kink in the detectability of the 200 node network mentioned earlier in Fig. 15 leading us to believe that the particular topology of the 200 node graph affects the detectability of our monitor node.

Detection-thresholds as a function of the three distances: monitor to liar, liar to lied-about and monitor to lied-about are charted in Fig. 17 for the 800 node network. Again it is clear that only the distance between the monitor and the liar is clearly correlated to the detection-thresholds.

Detection thresholds when a liar misstates its distance to all nodes is charted in Fig. 16 as a function of the hop-distance between liar and detector. Again, by comparing it to Fig. 14, it is clear that this class of lies are more detectable than lies to single-nodes.

D. Path Vector with Incomplete-Information

We next show the results of our experiments for the Path Vector protocol where nodes have information to only a fraction of the nodes in the network. In this instance we wished to study how the availability of information to more nodes affected the ability to detect errors elsewhere in the network. In our experiments we varied the fraction of nodes in the network to which the monitor had routes from 25% to 100%. The effect on the detection-thresholds curves as a function of the distance between monitor and liar measured in hops to a single node is charted in Figs. 18. A clear effect is seen. As the fraction of the nodes to which paths are available, the chance of detection goes up. For example the area between the understatement and overstatement curves when routes to 100% of the nodes are available (undetected errors) is smaller than the area between the understatement and overstatement curves when routes to 25% of the

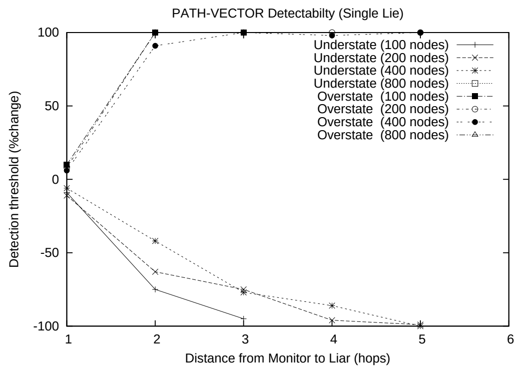


Fig. 14. PV: Misstatement to Single Node (hops)

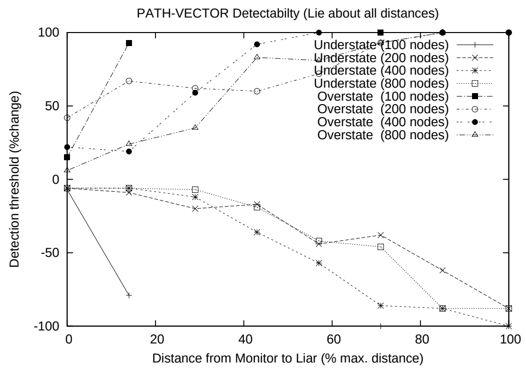


Fig. 16. PV: Misstatement to Multiple Nodes

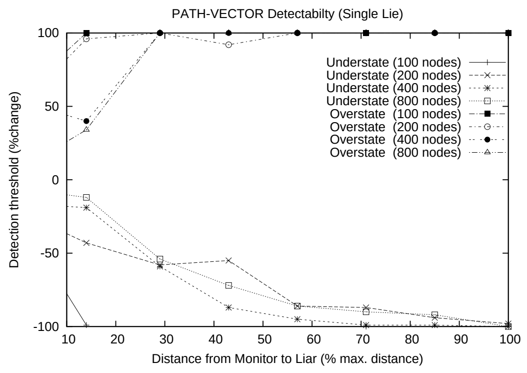


Fig. 15. PV: Misstatement to Single Node (relative distance)

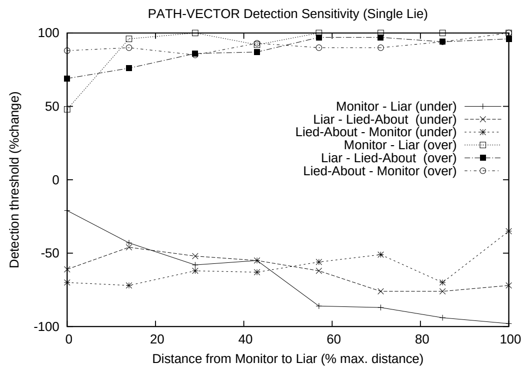


Fig. 17. PV: Effect on distance on Detection

nodes are available. This fits our intuition that as more accurate data becomes available the more likely that an error will be detected.

E. Comparing the Protocols

It is clear from the experimental results in Sec. V that detection-thresholds and therefore the self-monitoring capability varies for the different protocols. Additionally it is clear from Sec. IV that the state-complexity of the different protocol classes vary. In this section we briefly attempt to quantify the difference in the self-monitoring capabilities and state-complexities of different protocols and chart the trade-off.

A measure of the self-monitoring capacity of a protocol should tell us about the ratio of lies it can detect relative to the lies it cannot. Since the set of all lies is infinite the measure has to be over some constant set of lies. We therefore define a protocol's P's self-monitoring ability $d(P)$ for a set of lies S to be the ratio between detected lies and all lies in the set S.

$$\text{Self-Monitoring Ability} d(P) = |detected(S)|/|S|$$

where S is a set of lies, and $detected(S)$ is the set of lies in S that are detected.

In Sec. IV we computed the state-complexities of the different protocols. We tabulate it here in Fig. II for convenience. Note that $|V|$ is the number of nodes in the network, $|E|$ the number of edges, d the average out-degree and l the average path-length.

We tabulate the self-monitoring ability of the different protocols, their state-complexities and their efficiencies for the network with 200 nodes in Fig. III. The self-monitoring ability of the different protocols is measured over the over the set of understatements uniformly distributed between 0% and -100%. We compute the efficiency to be the ratio between the self-monitoring ability and state-complexity.

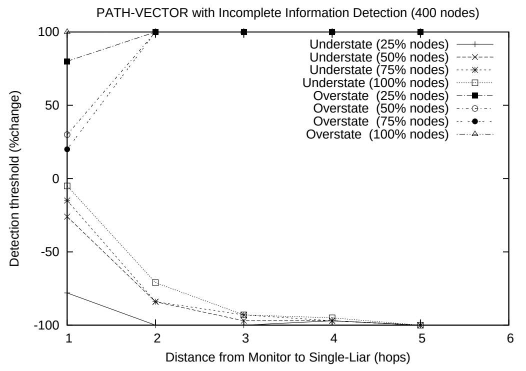


Fig. 18. PV-II: Detectability

It can be seen that the distance-vector protocol detects the smallest fraction of lies, while the path-vector protocol shows an improvement in detectability. The link-state protocol, which can detect all under-statements unless the liar partitions the network, is provided as a reference.

VI. Future work and Conclusion

Our current work involves analyzing the ability of single nodes or groups of cooperating nodes to detect errors when groups of collaborating nodes misstate distances. Additionally we are investigating the ability of a node, or a group of nodes, to pinpoint the origin of an error. We are also interested in studying how errors propagate across networks as a function of the protocol used and as a function of characteristics of the network such as connectivity, diameter and degree-distribution.

Protocol	State-Complexity
Distance Vector	$3 + d$
Path Vector	$2 + l$
Path Vector (w hop-by-hop distance)	$2(1 + l)$
Path Vector (Incomplete Information)	$2(1 + l)$
Link-State	$2(1 + \|E\|/\|V\|)$

TABLE II
STATE-COMPLEXITY

Protocol	Monitoring Ability	State-Complexity	Efficiency
Distance Vector	10	3	3.3
Path Vector	32	8.6	3.7
Path Vector (w hop-by-hop distance)	32	12	2.7
Link-State	100	8	13

TABLE III
EFFICIENCY

In this paper we presented a theory that establishes bounds on the kinds of errors that can and cannot be detected through self-monitoring in different protocols. We presented practical algorithms for well-known routing protocols that show how this theory can be applied by a node which, through a simple analysis of its state-information, can check whether other nodes are in fact operating as they are supposed to. Such policing mechanisms, we hope, will assist in identifying accidental misconfigurations and malicious attacks and further act as a deterrent to malicious attackers, since careless attacks will easily be detected.

REFERENCES

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

[2] 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.

[3] 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.*, 4(1):1–36, 2001.

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

[5] 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.

[6] N. Feamster, J. Borkenhagen, and J. Rexford. Controlling the impact of bgp policy changes on ip traffic. In *NANOG25*, 2002.

[7] 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*, pages 25–36, New York, NY, USA, 2005. ACM Press.

[8] L. Gao. On inferring autonomous system relationships in the internet. In *Proc. IEEE Global Internet Symposium, November 2000.*, 2000.

[9] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *IEEE INFOCOM 2000*, pages 1371–1380, Tel Aviv, Israel, March 2000. IEEE.

[10] T. G. Griffin and G. T. Wilfong. An analysis of BGP convergence properties. In *Proceedings of SIGCOMM*, pages 277–288, Cambridge, MA, August 1999.

[11] 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*, pages 12–23, New York, NY, USA, 2002. ACM Press.

[12] 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.

[13] 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*, pages 179–192, New York, NY, USA, 2004. ACM Press.

[14] 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*, pages 173–178, New York, NY, USA, 2005. ACM Press.

[15] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (bgp). *IEEE Journal on Selected Areas of Communication*, 18(4):582–592, April 2000.

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

[17] D. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Co.: Reading Mass, 1984.

[18] R. Mahajan, D. Wetherall, and T. Anderson. Understanding bgp misconfiguration. In *Proceedings of ACM SIGCOMM 2002.*, 2002.

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

[20] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: Universal topology generation from a user's perspective. Available at http://www.cs.bu.edu/brite/.

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

[22] 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.

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

[24] V. N. Padmanabhan and D. R. Simon. Secure traceroute to detect faulty or malicious routing. *SIGCOMM Comput. Commun. Rev.*, 33(1):77–82, 2003.

[25] 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.

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

[27] P. Radoslavov, H. Tangmunarunkit, H. Yu, R. Govindan, S. Shenker, and D. Estrin. On characterizing network topologies and analyzing their impact on protocol design. Technical Report USC-CS-TR-00-731, University of SOuthern California, Mar. 2000.

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

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

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

[31] 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.

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

[33] 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*, pages 342–351, New York, NY, USA, 2005. ACM Press.

[34] 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.

[35] 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*, pages 1–4, New York, NY, USA, 1990. ACM Press.

Appendix

Here we provide the proofs that the canonical-graphs G constructed in Sec. IV for the distance vector and path-vector protocols are indeed the only graphs that needs to be considered among all the graphs in $\mathcal{G}$. We present preliminary results as lemmas before we proceeding to proving the main theorems.

Lemma 1.1: 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, $w'(x,i) \leq w(x,i)$ [6].

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 1.2: 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 4.1: 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 1.1 and 1.2, the state table d'_n of G' will match the state table d_n of G. ■

Theorem 4.2:

In the path-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 path-vector canonical graph, $G' \in \mathcal{G}$.

[6]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.

Proof: The proof is by contradiction. Assume that there is a graph G that would produce state table d_n, but a neighbor k exists for which there is a node i where $d'_n(k,i) \neq d_(k,i)$. WLOG choose i for which $d'_n(k,i)$ is minimized. Let x be the node that immediately precedes i in the shortest-path from k to i in G'. From our choice of i, $d_n(k,x) = d'_n(k,x)$ but $d_n(k,i) \neq d'_n(k,i)$, therefore $w'(x,i) \neq w(x,i)$. Since in our construction, we draw an edges $e'(x,i)$ with weight $w'(x,i)$ if and only if it is contained in some path $d(m,n)$, it must be the case that $w'(x,i) = w(x,i)$ leading to a contradiction. ■