

# An Efficient Algorithm for Malicious Update Detection & Recovery in Distance Vector Protocols

Anirban Chakrabarti and G. Manimaran  
 Dept. of Electrical and Computer Engineering  
 Iowa State University, Ames, IA 50011.  
 {anirban,gmani}@iastate.edu

**Abstract**— The Internet infrastructure security has been gaining importance in recent years due to growing concerns for cyber-warfare. Among different network threats, the routing table poisoning attack is the most devastating and least researched topic which needs immediate research attention. In this paper, we develop a scalable algorithm for detecting and recovering from router attacks in distance vector routing protocols. The algorithm is able to detect and recover from malicious updates under certain well-defined conditions. We carry out extensive simulation studies to evaluate the proposed Pivot Based Algorithm for Inconsistency Recovery (PAIR) for three performance metrics, viz. detection probability, recovery probability and malicious distance under different network and attack scenarios. Our studies show that the PAIR is extremely scalable and offers high detection and recovery capability.

## I. INTRODUCTION

The importance of securing the Internet has grown rapidly due to a series of attacks that shut down some of the world’s most high profile Web sites, including Amazon and Yahoo [1] and due to the growing concerns of cyber-terrorism. Internet attacks can be mainly categorized into four types [2]: (i) *Routing Table “Poisoning” Attacks* (ii) *Packet “Mistreating” Attacks* (iii) *Denial-of-Service Attacks* and (iv) *DNS “Hacking” Attacks*. Routing is among the most critical functions of the Internet. “Poisoning” of routing tables thus forms one of the most potent attacks possible on the Internet [3] which can result in: (i) Sub-optimal routing, (ii) network partitioning, (iii) congestion, (iv) looping and (v) illegal access to data.

The majority of work on routing protocols for the Internet has proceeded in two main directions: distance vector protocols (e.g. RIP [4]) and link state protocols (e.g. OSPF [5]). In distance vector protocols, each node sends the distance (in hops) to its neighbor nodes. In case of link state protocols, each node periodically floods the state of its links to all the nodes in the network. Distance vector protocols are less robust than the link vector protocols. This is because each router computes the routes based on the computation done in the other routers in the network. Distance vector protocols can be subjected to: (i) *Link Attacks*, where an adversary gets access to a link in the network and changes the distance vector update passing through the link and (ii) *Router Attacks*, where a malicious router (source or intermediate) changes the distance vector information. It is to be noted that such attacks are also possible in case of link state protocols. However, the attacks are much more lethal in case of distance vector because of the

implicit trust relationship among the routers. In this paper, we concentrate on router attacks in distance vector protocols.

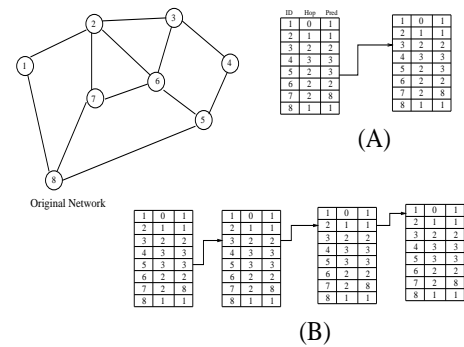


Fig. 1. Example showing the working of CC Algorithm

## A. Related Work

The solutions proposed for detecting distance vector attacks can be broadly classified into three categories:

**Routing Information Techniques:** In this type of techniques [6], [7] digital signatures are used to detect malicious distance vector updates in case of link attacks. However, these schemes are unable to detect router attacks.

**Intrusion Detection Techniques:** These techniques [8] are used to detect the anomalous behavior in the routers, assuming that intrusion detection devices are available in the network.

**Routing Protocol Techniques:** In this type of techniques, detection capability is built into the routing protocol itself. In Cisco White Papers [9], several techniques have been mentioned to detect bad/malicious routers. However, though the techniques are able to prevent looping, malicious distance vector updates cannot be detected using these techniques. One method of validating the integrity of the distance vector update, in presence of router attacks, is by using a technique called the “Consistency Check” (CC) [10]. In this technique, each router, in addition to the hoplength information, also sends the predecessor information to its neighbors.

In this paper, we adopt the principle of routing protocol techniques. Before going into details, we illustrate the working of CC algorithm through an example in Figure 1(A). The figure shows the distance vector update having three columns: node id, shortest distance and the predecessor for each shortest path. Whenever a node receives the distance vector from its neighbor, it carries out consistency check by tracing the path

from each destination to the source. An example is illustrated in Figure 1A. Let *node1* send its distance vector to all its neighbors. Each neighbor node checks the distance vector update by tracing out a path from each node to the source (*node1*). The distance vector update from *node1* claims that the predecessor of *node5* is not *node8*, but *node3*. This inconsistency in the update can be identified by the CC algorithm. Figure 1(B) shows an example when the CC fails. The check fails when an intermediate router sends a wrong update keeping the topology in mind. Let the routing update sent by *node1* has distance entry to *node5* as 3 instead of 2, and the predecessor as *node3* instead of *node8*. Tracing to source *node1* indicates that the update is consistent. Therefore, in this case, CC is unable to detect a wrong update.

Any algorithm, which is used to validate the router data in distance vector routing protocol should be able to maximize the detection and recovery of malicious updates. In addition, the protocol should be lightweight *i.e.*, the packet header overhead and packet processing overheads should be minimized to achieve scalability. The CC algorithm does not provide any guarantees in terms of its detection capability. As shown in the Figure 1(B), even a single change a pair of distance vector entries (hoplength and predecessor) can result in detection avoidance by CC. Also CC is not able to recover from any kind of malicious update. In this paper, we propose a scalable algorithm for inconsistency error recovery called ‘‘Pivot Based Algorithm for Inconsistency Recovery’’ (PAIR) which achieves the following: (i) PAIR is able to provide guarantees in terms of detection, (ii) PAIR is able to recover from certain malicious updates, (iii) PAIR is lightweight as it does not introduce any extra overhead (compared to CC) and has a low running time.

## II. PIVOT-BASED ALGORITHM FOR INCONSISTENCY RECOVERY (PAIR)

In PAIR, *transmitting node* is the node transmitting the distance vector update and *receiving node* is the node receiving the distance vector update. PAIR has the following steps:

**The transmitting node** (i) Computes the shortest path tree (distance vector tree), (ii) Computes the predecessor and path sum metrics for each destination based on the distance vector tree. Path sum metrics is defined as the sum of hoplengths of all paths that passes through or terminates in the destination node, (iii) Sends the predecessor and pathsum metrics for each destination, to all its neighbors.

**The receiving node** (i) Constructs a tree based on the predecessor information available in the update, (ii) Calculates the hop length and the path sum metrics, based on the constructed tree, (iii) Compares the calculated result with the received information, and concludes that the packet is valid if they match, (iv) Identifies whether the error is recoverable, if the received information is not valid. This identification is based on seven properties which have been explained in the subsequent sections. If the recoverable properties are satisfied, the algorithm recovers from the error.

An example of the tree creation in PAIR is shown in the Figure 2. The table shown in the figure refers to the distance vector information received from *node1* of the network

mentioned in Figure 1. The tree shown in the figure, is the distance vector tree constructed from the update. The tuple mentioned beside each node in the Figure 2 refers to the pair  $\langle \text{hoplength}, \text{pathsum} \rangle$ . It is to be noted that the calculated values match the received information, so the update is valid and no recovery procedure is required.

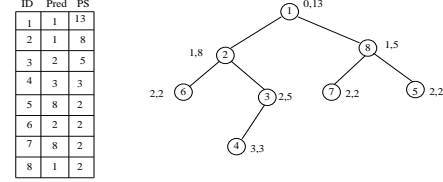


Fig. 2. Distance Vector Tree Construction using predecessor information

### A. Some Definitions

Some terms are defined in this section which will be used extensively during the description of PAIR.

**Distance Vector Tree ( $\tau$ ):**  $\tau$  is the distance vector tree constructed at the receiving node, based on the received predecessor information. The transmitting node forms the root of the distance vector tree.

**Pivot:** Pivot of two nodes  $i$  and  $j$ , of a distance vector tree  $\tau$  having root  $r$  is defined as the lowest common ancestor in the path  $(i, r)$  and  $(j, r)$ . In Figure 2, the pivot of nodes 5 and 6 is node 1, pivot of nodes 4 and 6 is node 2 and so on.

**Descendants ( $\Delta$ ):** Descendants ( $\Delta_i$ ) of node  $i$  in distance vector tree  $\tau$ , is defined as the children of  $i$ . In Figure 2,  $\Delta_1 = \{2, 8\}$ ,  $\Delta_2 = \{6, 3\}$  and so on.

**Predecessor( $\rho$ ):** Predecessor  $\rho_i$  of a node  $i$  in  $\tau$  is  $j$ , if  $i$  is a descendant of  $j$ . Therefore,

$$\rho_i = j \Leftrightarrow i \in \Delta_j \quad (1)$$

**Hop Length( $\eta$ ):**  $\eta_i$  of a node  $i$  is defined as the number of hops in the shortest path from the root node to  $i$  in the distance vector tree  $\tau$ .

**Path Sum ( $\sigma$ ):**  $\sigma_i$  of node  $i$  in  $\tau$  is defined as the sum of all path lengths passing through and terminating in  $i$ . Mathematically,

$$\sigma_i = \eta_i + \sum_{\forall j \in \Delta_i} \sigma_j \quad (2)$$

**Net Path Sum( $\theta$ ):**  $\theta_i$  of a node  $i$  in  $\tau$  is defined as the difference between the calculated and received  $\sigma$ . If received  $\sigma$  for node  $i$  is  $\sigma_i^r$  then,

$$\theta_i = \sigma_i - \sigma_i^r \quad (3)$$

### B. Different Steps of PAIR

PAIR has the following steps:

**Tree Construction:** A tree is constructed based on the predecessor information from the received update.

**Updation of Metrics:**  $\eta$ ,  $\rho$  and  $\sigma$  of each node is updated based on the constructed tree.

**Detection Procedure:** Detection of malicious update is done based on the calculated metrics.

**Recovery Procedure:** An inconsistency recovery is done based on the calculated metrics.

### C. Tree Construction & Updation of Metrics

In tree construction step, for each node in the distance vector update, the node is added to the list of descendant nodes of the predecessor node. It is to be noted that the root of the tree thus formed is actually the node from which the distance vector update is received. This step runs in  $O(n)$  time.

After the construction of the distance vector tree,  $\eta$ ,  $\sigma$  and  $\theta$  need to be updated for each node in the distance vector tree.  $\eta$  *Updation*:  $\eta$  is updated by recursively traversing the tree starting from the root node. At each recursion, the  $\eta$  of a descendant of a node  $x$  is incremented.

$$\eta_y = \eta_x + 1, \quad \forall y \in \Delta_x \quad (4)$$

$\sigma$  *Updation*:  $\sigma$  is updated by recursively calculating  $\sigma$  as sum of all the descendant  $\sigma$  values according to Equation 2.

$\theta$  *Updation*:  $\theta$  is updated using Equation 3.

$\eta$ ,  $\sigma$  and  $\theta$  updations can be performed in  $O(n)$  time.

### D. Detection Procedure

PAIR identifies the update as malicious if  $\theta_i \neq 0$  for some  $i \in \{1, 2, \dots, n\}$ . This detection technique guarantees that all malicious updates will be detected if at most one pair of entries ( $\sigma$  and  $\rho$ ) are changed for any node in the distance vector update (See [11]). This guarantee cannot be extended if more than one pair of entries are changed. In Section III, we have studied the detection ability of PAIR, when more than one pair of entries are changed.

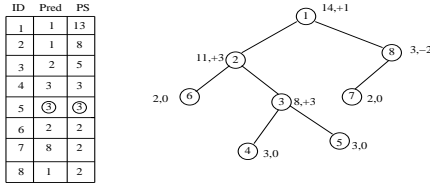


Fig. 3. Detection of Consistency Attack

**An Example of Detection:** The received update and the calculated tree (based on the predecessor information in the update) are shown in Figure 3. The tuple shown beside each node in the distance vector tree indicates the  $\langle \sigma, \theta \rangle$  pair, calculated based on the distance vector tree. The actual  $\sigma_5 = 2$  and  $\rho_5 = 8$ , the received update contains the values as 3 and 3 respectively. It can be seen that nodes 1, 2, 3 and 8 have non-zero  $\theta$  values. Thus, an attack can be identified.

### E. Recovery Procedure

Along with detection, the ability of PAIR to recover makes the algorithm very powerful. PAIR is able to recover from malicious updates under the following conditions: (i)  $\sigma_i > 0 \forall i \in \tau$  i.e., there is no loop in the received predecessor information. (ii) If  $x$  is the root node, then  $\theta_x \neq 0$  i.e., the hoplength of the actual and the changed predecessor are not equal. (iii) There is difference of only one entry between the actual and the received distance vector information i.e. either  $\sigma$  or  $\rho$  for only one node in the received distance vector information can be different from the actual one.

In addition to the general definitions mentioned earlier, we mention some added definitions which will be used for the

description of the predecessor recovery. An example shown in Figure 4 is used to illustrate the definitions given below. Figure 4(a) is the distance vector tree corresponding to the network shown in Figure 1. Figures 4(b), (c) and (d) show the constructed distance vector trees based on the received updates at different instances of time. In Figures 4(b), (c) and (d) one predecessor of a node is different from that in the actual distance vector tree shown in Figure 4(a) e.g., in Figure 4(b), the predecessor of *node5* has been changed from *node8* to *node3*.

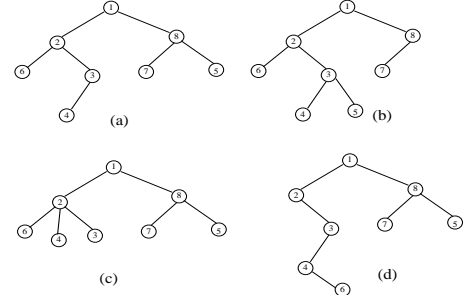


Fig. 4. Examples of different types of pivots

**Inconsistent Node:** The node whose predecessor has been changed by the malicious adversary, is called the inconsistent node. In Figures 4(a), (b) and (c), nodes 5, 4 and 6 respectively are the inconsistent nodes.

**Detaching & Attaching Nodes:** The actual predecessor and the calculated predecessors of the inconsistent node are called the *detaching node* and *attaching node* respectively. In Figure 4(b), *node5* is the inconsistent node. It has been detached from *node8* and attached to *node3*. Therefore, the detaching and attaching nodes for inconsistent node (*node5*) are *node8* and *node3* respectively.

**Passive, Detaching & Attaching Pivots:** The pivot of the detaching and attaching nodes of an inconsistent node, which is an attaching node by itself is called an *attaching pivot*. Similarly, a pivot which is a detaching node by itself is called a *detaching pivot*. A pivot which is neither an attaching or a detaching node by itself is called a *passive pivot*. An example of attaching node pivot is shown in Figure 4(c). *Node4* is the inconsistent node, while nodes 2 and 3 are attaching and detaching nodes respectively. Pivot of *node2* and *node3* is *node2*, which is an attaching node. Therefore, *node2* is an attaching pivot. Examples of detaching and passive pivots are shown in Figures 4(d) and (b) respectively.

**Common Nodes:** All nodes excepting the pivot, attaching or detaching nodes are called *common nodes*.

**Algorithm for Inconsistency Recovery:** To recover from inconsistency, several properties of pivot need to be used listed in Figure 5 (For detailed proofs, please refer to [11]). The algorithm proceeds by checking whether the update is recoverable. In that case, the type of inconsistency ( $\sigma$  or  $\rho$ ) is identified. If the  $\theta$  value of only one node in the constructed distance vector tree is non-zero, the update is considered to possess  $\sigma$  inconsistency. This is because, inconsistency in  $\rho$  results in multiple nodes having non-zero  $\theta$  in the distance vector tree. Recovery in case of  $\sigma$  inconsistency is carried out by accepting the calculated  $\sigma$  value as actual for the node

<b>Property1:</b> If $x$ is a common node and $\theta_x \neq 0$ , then there is only one node $x_i$ such that, $\theta_{x_i} = \theta_x$ and $\theta_{x_j} = 0 \forall j \in \Delta_x, i \neq j$
<b>Property2:</b> If $p$ is the pivot, and $\lambda$ is the inconsistent node, then $\theta_p = \theta_\lambda$ .
<b>Property3:</b> If a node $p$ is a passive pivot, then there are exactly two nodes among the descendant nodes of $p$ ( $p^+$ and $p^-$ ) such that, $\theta_p = \theta_{p^+} + \theta_{p^-}$ , where $\theta_{p^+} > 0, \theta_{p^-} < 0$
<b>Property4:</b> If node $p$ is a detaching pivot, then there is exactly one descendant of $p$ ( $x$ ) such that $\theta_x > \theta_p > 0$ .
<b>Property5:</b> If node $p$ is an attaching pivot, then there are exactly two descendant of $p$ ( $x$ and $y$ ) such that $\theta_x < \theta_y = \theta_p < 0$
<b>Property6:</b> If $x$ is a non-pivot detaching node, then $\theta_x < 0$ and $\theta_y = 0, \forall y \in \Delta_x$ .
<b>Property7:</b> If $x$ is a non-pivot attaching node, then there exists exactly one descendant $y$ of $x$ , such that $\theta_y = \theta_p$ , where $p$ is the pivot.

Fig. 5. Properties of Pivot under recoverable updates

whose  $\sigma$  value has been changed. This is done because, the calculated  $\sigma$  depends on the received  $\rho$  information, not on the  $\sigma$  information. Recovery from  $\rho$  inconsistency is carried out by identifying the pivot, the inconsistent node, the attaching and the detaching nodes. To locate the pivot, help of Property1 is taken. Search is carried out from the root node and the first node which does not have  $\theta$  equal to that of the root node is the pivot. Once the pivot is located, identification of the pivot is done based on properties 2,3 and 4. Then inconsistent node, attaching and detaching nodes are identified based on Properties 2, 6 and 7. After the identification of different nodes, the detaching node is made the predecessor of the inconsistent node. After the recovery process, the  $\theta$  values of all nodes are recalculated and the recovery is accepted if the update becomes consistent after recovery.

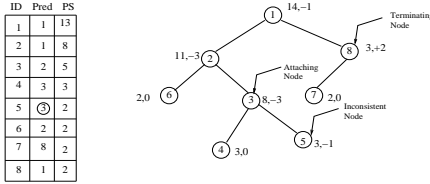


Fig. 6. An Example of Recovery - Presence of Passive Pivot

**Example of Recovery:** In Figure 6, a simple example of inconsistency recovery is illustrated. The scenario refers to the network shown in Figure 2. Node1 sends the distance vector update shown in Figure 6. Node1 is malicious, and has changed the entry for node5, marking the predecessor as node3, instead of node8. The neighbor nodes of node1, on receiving the distance vector update, identify that there is an inconsistency as all  $\theta$  values are not equal to 0, and  $\theta_1 = -1$ . It is to be noted that node1 is a passive pivot, as we can find two descendants of 1, such that they satisfy Property3. Iterating along the positive descendant of the pivot, node8 is the detaching node as all its descendants (node7 in this case) have  $\theta$  value of 0. Iterating along the positive descendant, node3 is identified as the attaching node and node5 as the inconsistent node since  $\theta_5 = \theta_1 = -1$ . Thus, the algorithm is able to recover by attaching node5 to node8.

### III. SIMULATION STUDIES

In order to evaluate the performance of PAIR, extensive simulation studies have been performed using NS-2 [12]. The performance metrics used to evaluate PAIR are:

**Detection Probability ( $p_d$ ):** It is the probability that a malicious update can be detected. It is to be noted that  $p_d = 0$  for traditional distance vector protocols.

**Recovery Probability ( $p_r$ ):** It is the probability that a malicious update will be recovered. It is to be noted that  $p_r = 0$  for CC algorithm and traditional distance vector protocols.

**Malicious Distance ( $\mu$ ):** It is defined as the average number of entries required to be changed in the distance vector update, to make the update consistent. Higher the value of  $\mu$ , more difficult it is to make an update consistent and hence lower is the probability of getting a consistent update.

The following simulations setup was used: (i) Network topologies were generated randomly. (ii) Two types of experiments were conducted by employing two different methods of malicious update generation: random and to minimize the detection probability respectively. (iii) The default values of the network parameters are: #Nodes = 40, Average node degree = 4.0, Link cost = 1.

#### A. Variation of Detection Probability ( $p_d$ )

In Figure 7(a), the detection probability ( $p_d$ ) of PAIR vis-a-vis CC is varied with node degree. In these set of experiments, 1 pair of entries ( $\rho$  &  $\eta$  in case of CC and  $\rho$  &  $\sigma$  in case of PAIR) are randomly changed in the distance vector update. This set of experiments illustrate the detection capability of PAIR, as nearly 99% of updates are detected in PAIR, while the  $p_d$  of CC reduces as the degree of the nodes increases. The reason behind this is because, when the nodes form the leaves of the distance vector tree even a change in one pair of entries is sufficient to make the update undetectable by CC. PAIR, however, does not suffer from this deficiency and thus is able to detect nearly 99% of the malicious updates.

In Figures 7(b), detection probability ( $p_d$ ) is varied with the number of pairs of changed entries selected randomly. In this set of experiments, the average degree of the nodes are kept fixed at 4. PAIR consistently provides a very high  $p_d$ , while for CC  $p_d$  is 50% for 1 pair of entry change, 67% for two pairs of entry change and so on. For a large number of changes both PAIR and CC provide high  $p_d$ . The reason for this behavior is that, when the 1 pair of entries are changed, CC gets affected 50% of the time, as approximately half of the nodes are leaves of the distance vector tree. This limitation is not there in case of PAIR, as detection is possible even if the nodes form the leaves of the distance vector tree. However, when the number of pairs of changed entries increases, more and more non-leaf nodes gets affected resulting in detection in case of CC.

In Figure 7(c), the detection probability is again varied with the number of pairs of changed entries in the distance vector update. The entries are selected to minimize the detection probability. A change in predecessor of a node  $X$  in case of PAIR, results in change of  $\sigma$  of all nodes from  $X$  to root of the distance vector tree. On the other hand, a change in predecessor of a node  $Y$  in case of CC, results in change of  $\sigma$  of all nodes which are descendants of  $Y$ . To minimize the detection probability, the above conditions are taken into account. Inspecting the results, PAIR performs significantly better in terms of detection when the number of entries

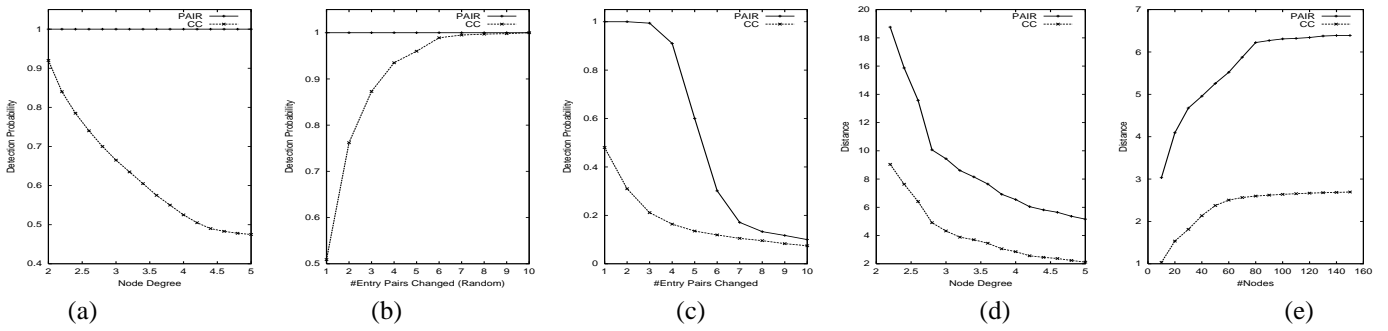


Fig. 7. Variation of Detection Probability and Distance

changed in low. Detection probability in case of PAIR, is above 90%, when the number of entries changed is 4, or less than 4. After 4, the detection probability in case of PAIR drops significantly, and finally becomes more or less equal to the detection probability in case of CC, for number of changed entries equal to 8 or more.

### B. Variation of Distance ( $\mu$ ) & Recovery Probability ( $p_r$ )

In Figure 7(d) and 7(e), the distance is varied with node degree and number of nodes respectively. Figure 7(d) shows that the distance of PAIR is more than two times the distance in case of CC for both dense and sparse networks. The distance of the detection algorithms (both PAIR and CC), decreases with increase in node degree as the height of the distance vector tree decreases as the network density increases. This is because, with the increase in density of the network, probability of two nodes being directly connected increases. As shown in Figure 7(e), the distance of PAIR is significantly higher than CC, and the ratio of distance in case of PAIR and CC increases as the network size increases. These sets of experiments clearly show that, it is more difficult to make an update consistent in case of PAIR than in case of CC.

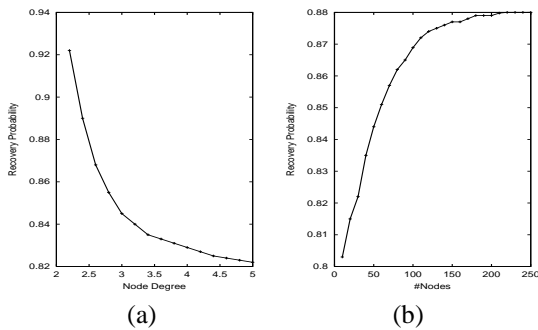


Fig. 8. Variation of Recovery Probability with Node Degree

Figures 8(a) and 8(b) illustrate the variation of recovery probability with node degree and number of nodes in the network respectively. It is to be noted that recovery probability of only PAIR is plotted as CC and traditional distance vector protocols do not recover from malicious distance vector updates. For a varied network sizes and densities, the recovery probability is very high, more than 80% in all cases. Recovery probability decreases with increasing node degree. This is because, with increasing network densities the distance vector trees typically have smaller heights. Therefore, recovery suffers. On the other hand, with increasing network size the

probability of creating a loop decreases, therefore recovery improves. It is to be noted that PAIR is very scalable as the recovery probability does not drop below 83% in any case.

## IV. CONCLUSIONS

In this paper, we presented an elegant algorithm (PAIR) to detect and recover from router attacks in distance vector routing protocols. The algorithm involves sending predecessor and pathsum metrics along with the distance vector updates, instead of the traditional hoplength information. The recovery of malicious updates is based on certain mathematical properties of pathsum metrics. We also carried out extensive simulation studies to evaluate the PAIR algorithm for three metrics *viz.* Detection Probability, Recovery Probability and Malicious Distance. Our simulation and analytical studies show that the algorithm achieves the following: (i) PAIR is lightweight, as it just requires only 4 bytes of extra information per node in the distance vector packet. (ii) PAIR has a low complexity ( $O(n)$ ), and hence easily implementable. (iii) PAIR is always able to detect and recover malicious updates under certain well-defined conditions. (iv) Detection probability of PAIR is significantly higher than that of the existing algorithms.

Future work includes: (i) Prototype implementation of PAIR, (ii) Development of an enhanced version of PAIR so as to detect/recover under relaxed conditions.

## REFERENCES

- [1] Lee Garber, "Denial-of-Service Attacks Rip the Internet," *IEEE Computers*, vol.33, no.4, pp.12-17, Apr. 2000.
- [2] Anirban Chakrabarti and G. Manimaran, "Internet Infrastructure Security: A Taxonomy," *IEEE Network*, vol.16, no.6, pp. 13-21, Nov/Dec 2002.
- [3] Sandra Murphy, "Technology Transition for Internet Infrastructure Security: Secure OSPF," *NAI Labs Advance Research*, June 2000.
- [4] C. Hendrik, "Routing Information Protocol," *RFC 1058*, June 1988.
- [5] J. Moy, "OSPF Version 2," *RFC 1583*, March 1994.
- [6] S. Murphy, M. Badger, and B. Wellington, "OSPF with Digital signatures," *RFC 2154*.
- [7] K. Zhang, "Efficient Protocols for Signing Routing Messages," in *Proc. NDSS*, 1998.
- [8] Kirk. A. Bradley, S. Cheung, B. Mukherjee, and Ronald. A. Olsson, "Detecting Disruptive Routers: A Distributed Network Monitoring Approach," in *Proc. IEEE Symp. on Security and Privacy*, 1998.
- [9] Cisco White Papers, "Strategies to Protect against Distributed Denial of Service Attacks (DDoS)," Feb. 2000.
- [10] Bradley R. Smith, Shree Murthy, and J.J. Garcia-Luna-Aceves, "Securing Distance-Vector Routing Protocols," in *Proc. SNDSS*, 1997.
- [11] Anirban Chakrabarti and G. Manimaran, "A Scalable Algorithm for Malicious Update Detection & Recovery in Distance Vector Protocols," in *DCNL Technical Report*, July 2002.
- [12] UCB/LBNL/VINT Network Simulator - ns (version 2), Available at [www.mash.cs.berkeley.edu/ns/](http://www.mash.cs.berkeley.edu/ns/).