Secure Last Acknowledged Packet Routing Algorithm (SLAPRA) for MANETs

P. Cooper, L. Chen, and C. Sun

Abstract—**Mobile Ad Hoc Networks (MANETs) are becoming more pervasive thanks to their advantages of being infrastructureless and portable. However such networks suffer from a variety of attacks due to several inherent factors such as mobility, signal attenuation and interference, and limited computation power and battery resources on mobile devices. Secure light-weight routing for a reliable path consisting of trustable nodes has not been an easy task. In this research, we propose the Secure Last Acknowledged Packet Routing Algorithm (SLAPRA),capable ofefficient and prompt attack detection and malicious node isolation, and demonstrate the validity of the algorithm through simulation.**

Index Terms—**MANETs, security, secure routing, malicious nodes, Black Hole attacks, packet crafting, trust evaluation.**

I. Introduction

In contrast to wired networks with stable infrastructure, MANETs consist of mobile devices communicating over relatively less robust wireless links in a possibly ever-changing topology. The unreliability is mainly due to two reasons: device mobility and signal interference.

Random node mobility, hardly possible to predict, allows movement in arbitrary direction and speed leading to a higher incidence of broken links, and consequently can greatly reduce the performance of MANETs. Exceptions are cases such like Vehicular Ad Hoc Networks where the movement of vehicles and mobile devices may be confined within certain areas such as roads, and therefore can be predicted at a certain degree. In most MANETs, a path performing well at one moment may become unavailable in the next time interval. Therefore, an adaptive and dynamic routing algorithm is demanded to achieve better performance.

In previous research, group mobility models were developed according to the movement patterns of mobile nodes in certain applications. For example, the Reference Point Group Mobility (RPGM) model and a number of extensions were introduced [2]-[8]. In RPGM, mobile hosts are organized in groups according to their logical relationship, and each group has a logical center whose motion defines the entire group's motion behavior (location, speed, direction, and acceleration) where the motion of a single node is the vector sum of the motion of the reference point and the independent random movement. Reference Velocity Group Mobility (RVGM) model extends RPGM by adding velocity vector of the group and each node, resulting a more accurate depict of how nodes actually move. Ravikiran and Singh [5] carried out simulation and performance analysis on the above models over three major MANET routing algorithms: Dynamic Source Routing (DSR), Ad Hoc On Demand Distance Vector (AODV), and Destination Sequenced Distance Vector (DSDV). Cano and Manzoni [1] also reported on the simulation and performance of several different group mobility models. Although these mobility models may work well in predicting the location of mobile nodes, they fail to show the logical relationship (e.g. trust level) among neighbor nodes, which is critical in secure routing and routing optimization.

Noticing information exchange being important among neighbor nodes, Joe and Batsell introduced a Multipoint Relaying (MPR) based hybrid routing algorithm which makes use of the multipoint relaying based on the information exchange among neighbor nodes [3]. Nonetheless, this routing algorithm still does not explore logical relationship among neighbor nodes.

In an effort to achieve better reliability in MANETs, Ye and collaborators introduced a reliable routing framework, in which a number of reliable R-nodes are purposely placed in the network and play the role of supporting the network as backbones [9]. While this idea may be practical in certain metropolitan areas, it would not be suitable for impromptu scenarios such as disaster rescues and recovery.

It has been suggested in earlier research that routing can be optimized by collecting and aggregating relative information among neighbor nodes. For example, Chai-KeongToh and co-workers introduced the Associativity-Based Routing (ABR) which makes use of the Associativity Ticks among neighbor nodes [6], [7]. The Associativity Ticks indeed show a mobile node's dormant time, in which the node is in a stable status, yet, they are not able to illustrate the long term cumulative relationship among neighbors. For example, a node will treat its old, stable neighbor nodes same as newcomer neighbor nodes, which may be just passing through the neighborhood. Also, in ABR only the source node maintains the routing information, meaning when a node is temporarily unavailable (e.g. when a device is restarting), ABR will redo routing and may choose a worse node and not be able to switch back.

Security of networks has always been a critical issue in many network applications, especially in wireless environments. But unfortunately most current on-demand routing algorithms in MANETs, such as AODV, are not competent in detecting malicious nodes or malevolent behavior. Therefore a node with a piece of malicious code may advertise itself being on the shortest path yet in fact it is not [4]. Although Secure AODV (SAODV), proposed by

Manuscript was received September 4, 2012; revised October 10, 2012. This work was supported in part by the Enhanced Research Grant (ERG)funded by Sam Houston State University, Huntsville, TX 77341USA.

The authors are with the Department of Computer Science, Sam Houston State University, Huntsville, TX 77341 USA (e-mail:cooper@shsu.edu, chen@shsu.edu, cxs017@shsu.edu)

Zapata, protects route discovery mechanism which provides security features like integrity, authentication, and non-repudiation[10], it requires a signature key pair from a suitable asymmetric cryptosystem and each node being able to verify the association between the address of a given node and its public key. Such computation load and key management scheme may not be suitable in current MANET applications.

In this research, we propose the Secure Last Acknowledged Packet Routing Algorithm (SLAPRA), an innovative trust level-based routing algorithm to improve the security level of MANETs. More specifically, by making use of last acknowledged packets, SLAPRA detects and isolates misbehaving nodes that may be involved in attacks such as Black Hole Attacks, and packet crafting. The fundamental idea of SLAPRA is that, every node keeps the record of their Last Received Packet from the nodes and that of Last Sent Packet to other nodes. If a node detects change of route, in order to verify abnormality in the network, it compares the content of its recorded Last Received Packet to that of its last hop neighbor's Last Sent Packet.

This paper is organized as follows. In Section II, details of SLAPRA are introduced in four subsections: Tables, Packet Types, Algorithm, and Pseudo Code. Section III presents simulation and analysis. We draw conclusion and propose future work in Section IV.

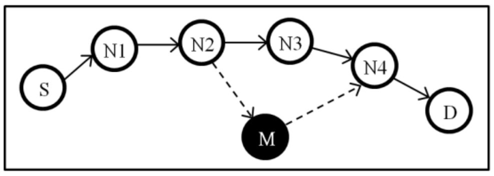

Fig. 1. A route example with six good nodes (*S*, *D*, *N*1, *N*2, *N*3, and *N*4) and one malicious node (M).

II. Secure Last Acknowledged Packet Routing

In the following subsection, we introduce the tables that are utilized by mobile nodes to store critical information about routing and trust relationship among neighbors. Subsection B describes the structures and functions of four different types of packets facilitating the secure routing algorithm. SLAPRA is discussed in detail in subsection *C* followed by the pseudo code in subsection *D*.

TABLE I: Node *N*4's Table T1

Source	Last Hop	SHA-1 Hash of LAP
S	*N*3	2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
*N*1	*N*3	de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b3
*N*2	*N*3	da39a3ee5e6b4b0d3255bfef95601890afd80709
*N*3	*N*3	aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d

A. Tables

Each node maintains and updates five tables to implement the algorithm. T1 (refer to Table I) is used for storing the hash (SHA-1 in this case) values of each of the Last Acknowledged Packets (LAPs) and the corresponding source (which initiated the current data packet) and the last hop node (which forwarded the current data packet to the current node) IDs. Source works as the key in Table I. Since hash value changes dramatically (at least half of bits) even with small change in original packet content, when original LAP is changed or modified, it helps detect changes in a fast and efficient manner.

In the route example shown in Fig. 1, *N*4's last hop is *N*3, and the hash of LAP is updated every time after it receives a packet from S. this information is stored in T1 of *N*4 as shown next.

Table T2 is used to store and manage the trust levels of the neighbor nodes from which the local node received packets. Every time when the local node receives a packet from one of its neighbor nodes for the first time, it appends a new record for its neighbor in its T2 with a default trust level, e.g. 30. Neighbor ID is the key for T2, and trust level can be dynamically updated. Table II below shows the content of *N*4's T2 in the same example.

TABLE II: Node *N*4's table T2

Neighbor ID	Trust Level for receiving
*N*3	30

Table T3 is used to store the information of Last Sent Packets (LSPs) to the next hops. Each record has two attributes: the neighbor node ID, and the Hash value of LSP to the corresponding neighbor node. Neighbor ID works as a key for this table. Table III below shows the content of *N*4's T3 in the same example.

TABLE III: Node *N*4's table T3

Neighbor ID	SHA-1 Hash of LSP
D	efen5c7fd25e1b3afad3e85a0bd17eft100db5b3

Table T4 is the Bad Node List, which stores all the IDs of malicious nodes detected in the neighborhood. The local node will place its neighbor node IDs into this table if their trust levels decreased to below a threshold, such as zero. As a consequence, a node will refuse to process any packet from the neighbor nodes listed in its T4. Table IV below shows the content of *N*4's T4 in the same example, where *N*8 is a previous bad node not shown in current route.

TABLE IV: Node *N*4's Table T4

Bad Node
N8

TABLE V: Node *N*4's table T5

Verification_Sequence_Number	Suspicious_Node
123	M
124	M

Table T5 maintains a list of suspicious nodes and the associated Verification Sequence Numbers (VSNs). VSNs work as a key for this table, and every time when the local node receives a suspicious (e.g. route change) packet, it creates a new VSN associated with the suspicious node in T5. Each suspicious node may be associated with multiple VSNs in T5, since it may send multiple suspicious packets to the local node. For example, when node *N*4 receives one suspicious packet from *M*, it creates a VSN in T5, and another suspicious packet from *M* later will result in a new VSN associated with *M* in T5.

B. Packet Types

There are four types of packets in this algorithm: data packet, verification packet, warning-type-1 packet, and warning-type-2 packet. We limit all packet size to 1KB. Packet header is set to 16 bytes, and the rest of packet is the payload. While the payload structure remains constant, the length of the actual content in the payload may vary.

A data packet is simply a regular packet with user data content. Its header consists of four parts: Packet Type, Source ID, Last Hop ID, and Payload Length, each of which is 4 bytes in size. To keep our simulation simple, we labeled data packets as type 1. The Payload Length field in the header dictates the length of the payload's content that will be read. The format of the data packets goes as Fig. 2 below.

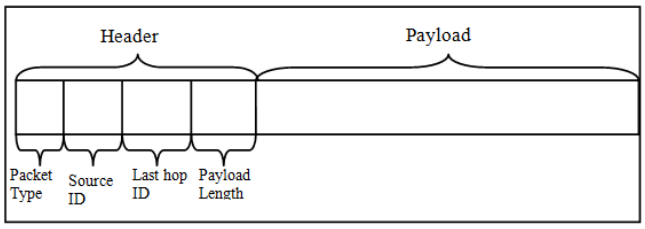

Fig. 2. Format of data type packets.

A verification packet is sent to the last hop in the path by a local node when the node finds certain data packet suspicious (e.g. when route change is detected). The header of verification packets includes four parts: Packet Type, Source ID, Last hop ID, and Verification Sequence Number (VSN). Each of these parts is 4 bytes. In the simulation, the packet type of verification packets is set to 0. The Payload part of verification packets consists of two parts: the 20-byte-long hash, and the Timestamp, which is 12 bytes in length. The structure of verification packets is shown in Fig. 3.

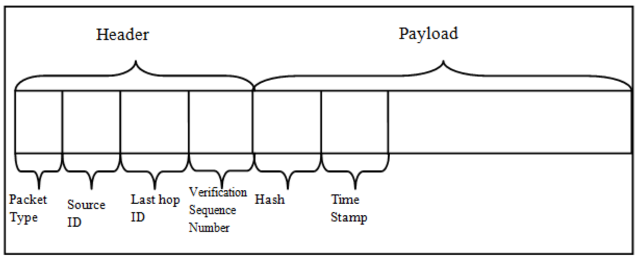

Fig. 3. Format of verification packets.

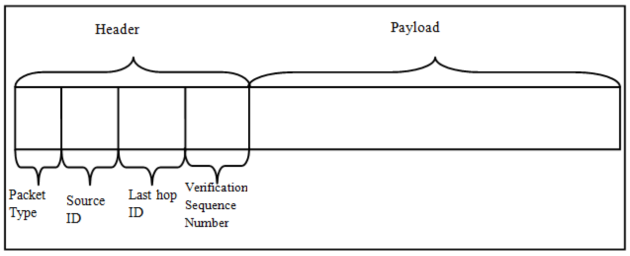

Fig. 4. Format of warning-type-1 and warning-type-2 packets.

Warning-type-1 and warning-type-2 packets share the same header format with verification packets, except for their payload being empty (refer to Fig. 4). Warning-type-1 packets are labeled as type 3 and warning-type-2 packets as type 4. The function of Verification Sequence Number (VSN) here is to inform the next hop node to decrease the trust level of the specific node corresponding to this VSN in the T5. The difference between these two warning-type packets is, Warning-type-1 informs the next hop to decrease the trust level with a constant β (due to black hole node claiming shortest path), and type 2 triggers trust level to decrease by a constant Ω (due to intentional packet crafting), where *β* is smaller than Ω. Details of the use of these two types of packets are discussed in following subsections.

C. Algorithm

For the following algorithm description, also refer to pseudo code in the next subsection. Every node, upon receiving a packet, will identify the packet type before proceed. If it is a data type packet, the node will traverse its T4 locally to check whether the last hop node of the packet can be found in table. Packet will be rejected if last hope is found in bad node list T4. Otherwise, packet's last hop will be compared to the Last Hop attribute value in T1. If there is a match on both source and last hop, this is considered a normal and regular operations are executed. Otherwise new record will be added to T1 if no match is found. Regular operations include updating record in T1, increasing trust level of last hop in T2by a small adjustable preset constant α, updating last hop information in the packet and record in T3, and finally sending the packet to next hop. In the example shown in Fig. 1, node N4 constantly receives packets from N3. Every time N4 updates LAP information with corresponding Source ID in its T1, and increases the trust level of N3 in T2. N4 also updates the hash of Last Sent Packet in T3, and then forwards the packet to destination node *D*.

Route change is detected when packet's last hop did not match the records in T1. In such case a verification packet needs to be sent to the corresponding last hop node. In the same example above, once *N*4 receives a packet from malicious node M, it finds out, after checking its T1, that route has been changed. Consequently *N*4 sends a verification packet to *N*3.

When a node receives a verification packet, it appends timestamp of the packet to the hash of LSP, and hashes the new combined bytes. Then the new hash result is compared with the hash content in the verification packet. In the above example, *N*3 receives a verification packet, which contains the hash and timestamp. *N*3 looks up the hash of LSP record from *S* in its T3, then calculates checksum = Hash(Hash of LSP + timestamp). If the checksum equals to the hash content in the packet, then M is considered to be a regular node just relaying packets. However in this case, *M* is still not the shortest path to N4, so it is considered ablack hole attacker if it claims being on the shorted path. When this is detected, *N*3 will send *N*4 a warning-type-1 packet to inform that *M* is a black hole. The worst case happens when the above checksum comparison turns out to be not equal. It indicates that *M* intentionallycraftedpackets apparent from source *S* (assuming all nodes on path are informed in regular rerouting). In this situation, *N*3 will alert *N*4 by sending it a warning-type-2 packet.

Both warning type packets carry VSN to be used in search within table T5. Upon any match, the trust level of suspicious node will be decreased in T3, by either β or Ω depending on the actual warning type.

D. Pseudo Code

Fig. 5 below shows the pseudo code and comments of the algorithm.

```
Packet=get_packet()  // packet received
if (Packet.type==data packet)   // data packet received
{
if (Packet.last_hop not in T4)  //packet not from bad node
{
hash Packet;
if (source exists in T1) // when packet source already exists in T1
{
ifPacket.last_hop<>T1(last_hop)
        // packet from different route; needs verification
{
T5.append(Sequence_number, Packet.last_hop, hash);
        // to verify the change of last hop
        send_verifcation_packet(Sequence_number);
}
else // last hop did not change; proceed with regular operations
{
update T1(last_hop, hash);
increase T2(last_hop, trust_level, α);
update T3(next_hop, LSP);
send_data_packet(next_hop);
       }
}
else
     {
append record to T1; // new source added to T1
update T3(next_hop, LSP);
send_data_packet(next_hop);
      }
}
   else // packet is from a bad node
    reject packet;
 }
else if (packet.type == Verification packet)
// verification packet received
{
   checksum=SHA-1(T3[Verification_source].hash_LSP +
   T3[Verification_source].ID + payload.system_timestamp);
   // hashed LSP along with timestamp is hashed
packet.payload.remove_timestamp;
    // timestamp is removed for comparison of the rest of payload
If ((checksum == payload)
// when same packet forwarded by a different node
       send_warning_1_packet(Verification_source); // inform next hop
else // packet content was changed and packet was forwarded
       send_warning_2_packet(Verification_source);
}
else if (packet.type == warning_type_1)
{
SnodeID= get_suspicious _node(Sequence_Number);
// suspicious black hole node
decrease T2(nodeID, trust_level,β); // black hole
if (trust_level< = 0) append to T4;
}
else if (packet.type == warning_type_2)
{
SnodeID = get_suspicious _node(Sequence_Number);
// suspicious packet crafting node
decrease T2(nodeID, trust_level, Ω); // packet crafter
if (trust_level< = 0) append to T4;
}
send_verification_packet(Verification_Sequence_Number)
{
   payload = SHA-1(concatenate LKCP(hash, source) +
   system_timestamp);
payload.append(local_ID, system_timestamp);
send_packet(source);// last hop node (source of current hop)
 }
```

Fig. 5. Pseudo code and comments of SLAPRA algorithm

III. Simulation and Analysis

In the following subsections we present the design and implementation of simulation, and demonstrate the results of testing.

A. *Design of Implementation*

The simulation is implemented by coding a multi-threaded peer to peer UDP application using Java Sockets. From the topology point of view, each node is an object of the Node class. Every node runs two threads: receiver and sender. The thread of receiver catches the UDP packets and then pushes them into the input queue. The thread of sender on the other hand fetches the received packets from the input queue in the same order they were received and queued, then processes the packets according to SLAPRA, and finally forwards them to the next hop on route.

B. *Coding*

Our simulation coding consists of eight java files. Node.java is a class for regular nodes containing a “Linkto” function which configures the topology by connecting one node to its next hop. Its “SendInitialPacket” thread sets the data rate of sending packets, and the numbers of total packets to be sent. This file also has three variables alpha, beta, and omega which configure the values of α, β and Ω in the algorithm respectively. These preset values can be adjusted for various network scenarios in future research. MaliciousNode.java is a class of malicious nodes, of which “Attack” function configures the node it attacks. The “Sender” thread sets the data rate at which malicious packets are sent. Table2.java is the class of T2 table in every node containing a constant “InitialTrust”, which sets the initial trust value, e.g. 20.

During the execution of these programs, each node in the network produces a log file, with its node ID as file name, containing all behavioral records of the node. These records are used for analyzing the validity and performance of the proposed algorithm.

C. *Testing and Results*

Program logs from testing show that our algorithm is capable of finding black hole node(s) in the network. Also, by examining these log files, we can verify that the regular nodes have actually isolated the malicious node(s) by decreasing trust levels and finally dropping them off the neighborhood’s communication.

How fast the malicious node can be isolated depends on the values of alpha, beta, and omega plugged in the runtime. Since the good nodes drop the trust level of the suspicious node gradually, the faster the malicious nodes send malicious packets, the more quickly it will be isolated. We consider this a nice security feature. In a real test, the source node is set to send the initial packets at the rate of 800kbps, and the malicious node sends crafted packets at rate of 40kbps. The initial trust level of each node is set to 20, and alpha for increasing trust level is set to 0.5. Beta for warning-type-1 to decrease trust level is 3, and omega for warning-type-2 to decrease trust level is 5. In the network plotted in Fig. 1, malicious node M begins to attack N4 200 milliseconds after the source node S started sending initial data packets. After 20 seconds of executing the program, we obtain the following records from N4’s log file (note that Node8000 refers to the malicious node and T2 and T4 are the local tables at N4):

2012-7-16 16:13:54.337: T2: Node8000 trust level decreased to 15.0!!

2012-7-16 16:13:54.552: T2: Node8000 trust level decreased to 10.0!!

2012-7-16 16:13:54.797: T2: Node8000 trust level decreased to 5.0!!

2012-7-16 16:13:55.035: T2: Node8000 trust level decreased to 0.0!!

2012-7-16 16:13:55.035: T4: Node8000 has been added table!!

2012-7-16 16:13:55.17: Reject the packet from Node 8000!

In Node8000's log file, we can find following record, which indicates the start time Node 8000 began to send malicious packets:

2012-17-16 16:13:54.277: Sent 'This is malicious node.' to port 5600.

This log record indicates our algorithm is capable of detecting black hole nodes and making alerts.

IV. Conclusion and Future work

In this paper, we have proposed Secure Last Acknowledged Packet Routing Algorithm (SLAPRA), which is capable of detecting black hole nodes and packet crafters in a very efficient and prompt manner. Our future research includes finding appropriate values of alpha, beta, and omega for various network scenarios. We aim to improve the current algorithm so that it is also capable of detecting other types of MANET attacks.

References

[1] J. C. Cano and P. Manzoni, "Group mobility impact over TCP and CBR traffic in Mobile Ad Hoc Networks," *IEEE Infocom*, 2004.

[2] X. Hong, M. Gerla, G. Pei, and C. C. Chiang, "A Group Mobility Model for Ad Hoc Wireless Networks," in *Proc. of ACM/IEEE MSWiM'99*, Seattle, WA, Aug. 1999, pp. 53-60.

[3] I. Joe and S. G. Batsell, "MPR-based Hybrid Routing for Mobile Ad-Hoc Networks," in *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks*.

[4] D. Mishra, Y. K. Jain, and S. Agrawal, "Behavior Analysis of Malicious Node in the Different Routing Algorithms in Mobile Ad Hoc Network (MANET)," *Advances in Computing, Control and Telecommunication Technologies*, pp. 621-623, 28-29 December 2009.

[5] G. Ravikiran and S. Singh, "Influence of Mobility Models on the Performance of Routing Protocols in Ad-Hoc Wireless Networks," IEEE VTC' 04 (spring), Milan, Italy, May 17-19, 2004.

[6] G. K. Toh, "Associativity-Based Routing for Ad-Hoc Mobile Networks," *Wireless Personal Communications*, vol. 4, pp. 103-139, 1997.

[7] C. K. Toh, M. Delwar, and D. Allen, "Evaluating the Communication Performance of an Ad Hoc Wireless Network," *IEEE Transactions on Wireless Commnunications*, vol. 1, no. 3, July 2002.

[8] K. H. Wang and B. Li, "Group Mobility and Partition Prediction in Wireless Ad-Hoc Networks," *IEEE Infocom*, 2002.

[9] Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi, "A Framework for Reliable Routing in Mobile Ad Hoc Networks," *IEEE Infocom*, 2003.

[10] M. G. Zapata, "Secure ad hoc on-demand distance vector routing," *Sigmobile Mob. Comput. Commun. Rev*, vol. 6, no. 3, 2002.