Simulation of Self-Organized Public-Key Management for Ad Hoc Networks

Arifa S Tamboli, Sunita S. Shinde and Shabanam S.Tamboli

Annasaheb Dange College of Engineering & Technology,

Ashta, MS (India)

ABSTRACT

In this paper we addressed the difficult problem of key management in mobile Ad Hoc networks. A self-organized public-key management scheme is proposed that does not rely on any trusted third party, not even in the network initialization phase. To distribute keys to the neighboring nodes we have taken help of inherent properties of DSDV routing algorithm. To exchange the keys of each node with other node in network, by checking the chain appended with entry & based on trust relationship. We also implemented fault handling strategy. New DSDV results are shown with help of X-graph, NS-2.

General Terms: Ad-hoc Network Security, Public Management system for Ad Hoc network

Key-Words: Ad-hoc Network, DSDV, key-management, simulation Ns-2, self-organized.

1. INTRODUCTION

Wireless networking is an emerging technology that allows users to access information and services electronically, regardless of their geographic position.

Wireless networks can be classified in two types: -

1.1 Infrastructure networks

Infrastructure network consists of a network with fixed and wired gateways. A mobile host communicates with a bridge in the network (called base station) within its communication radius. The mobile unit can move geographically while it is communicating. When it goes out of range of one base station, it connects with new base station and starts communicating through it. This is called handoff. In this approach the base stations are fixed.

1.2 Infrastructureless (Adhoc) networks

In ad hoc networks all nodes are mobile and can be connected dynamically in an arbitrary manner. All nodes of these networks behave as routers and take part in discovery and maintenance of routes to other nodes in the network. Ad hoc networks are very useful in emergency search-and-rescue operations, meetings or conventions in which persons wish to quickly share information, and data acquisition operations in inhospitable terrain.

2. DSDV (DESTINATION SEQUENCE DISTANCE VECTOR)

The Destination-Sequenced Distance-Vector (DSDV) Routing Algorithm is based on the idea of the classical Bellman-Ford Routing Algorithm with certain improvements.

DSDV routing is an enhancement to distance vector routing for ad-hoc networks. DSDV is a table-driven algorithm. The improvements made include freedom from loops in routing tables. Every mobile node in the network maintains a routing table for all possible destinations within the network and the number of hops to each destination. Each entry is marked with a sequence number assigned by the destination node. The sequence numbers enable the mobile nodes to distinguish stale routes from new ones, thereby avoiding the formation of routing loops. Routing table updates are periodically transmitted throughout the network in order to maintain table consistency.

To help alleviate potentially large amount of network traffic that such updates can generate, route updates can employ two possible types of packets. The first is known as a full dump. This type of packet carries all available routing information and can require multiple network protocol data units (NPDUs). During period of occasional movement, these packets are transmitted infrequently. Smaller incremental packets are used to relay only the information that has changed since the last dump. Each of these broadcasts should fit into a standard-size NPDU, thereby decreasing the amount of traffic generated. The mobile nodes maintain an additional table, where they store the data sent in the incremental routing information packets. The new route broadcast contains the address of the destination, the number of hops to reach the destination, the sequence number of the information received regarding the destination, as well as a new sequence number unique to the broadcast. The route labeled with the most recent sequence number is always used. In the event that two updates have the same sequence number, the route with small metric is used in order to optimize (shorten) the path. Mobile nodes also keep track of the settling time of the routes, or the weighted average time that routes to a destination will fluctuate before the route with the best metric is received. By delaying the broadcast of a routing update by the length of the settling time, mobiles can reduce network traffic and optimize routes by eliminating those broadcasts that would occur if a better route could be discovered in the very near future.

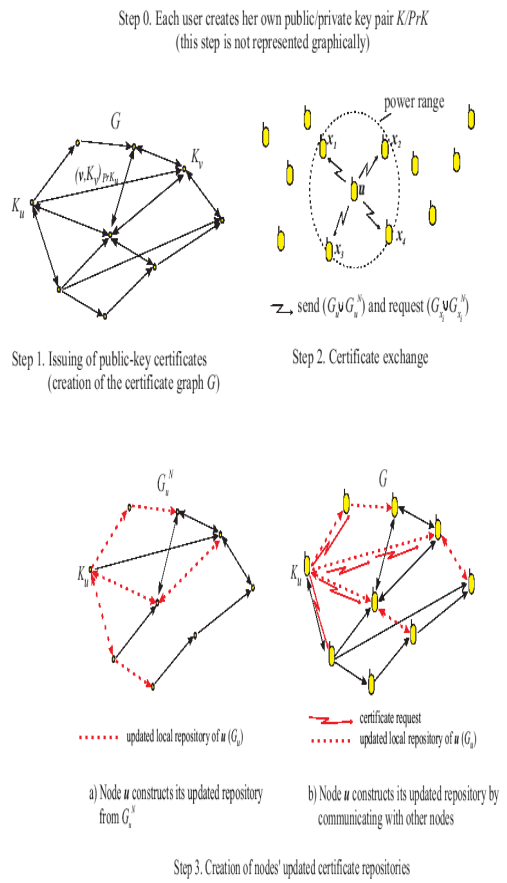


Figure 1. The creation of nodes' updated and non-updated repositories.

The routing table updates can be sent in two ways: - a "full dump" or an incremental update. A full dump sends the full routing table to the neighbors and could span many packets whereas in an incremental update only those entries from the routing table are sent that has a metric change since the last update and it must fit in a packet. If there is space in the incremental update packet then those entries may be included whose sequence number has changed. When the network is relatively stable, incremental updates are sent to avoid extra traffic and full dump are relatively infrequent. In a fast-changing network, incremental packets can grow big so full dumps will be more frequent.

2.1 Different Security mechanism in Adhoc network:

-common server

-threshold cryptography

-hash keying

-self organized

2.2 Self-organizing public-key management system

Self-organizing public-key management system that allows users to create, store, distribute, and revoke their public keys without the help of any trusted authority or fixed server. Moreover, in our solution, we do not assign specific missions to a subset of nodes (i.e., all the nodes have the same role).

Our main motivation for taking this approach comes from the self-organized nature of mobile ad hoc networks, and from the need to allow users to fully control the security settings of the system. As such, our approach is developed mainly for "open" networks, in which users can join and leave the network without any centralized control.

The main problem of any public-key based security system is to make each user's public key available to others in such a way that its authenticity is verifiable.

In mobile ad hoc networks, this problem becomes even more difficult to solve because of the absence of centralized services and possible network partitions.

More precisely, two users willing to authenticate each other are likely to have access only to a subset of nodes of the network (possibly those in their geographic neighborhood). The best known approach to the public-key management problem is based on public-key certificates. A public-key certificate is a data structure in which a public key is bound to an identity (and possibly to some other attributes) by the digital signature of the issuer of the certificate. In our system, like in PGP, users' public and private keys are created by the users themselves.

For simplicity, we assume that each honest user owns a single mobile node. Hence, we will use the same identifier for the user and her node (i.e., both user u and her node will be denoted by u). Unlike in PGP, where certificates are mainly stored in centralized certificate repositories, certificates in our system are stored and distributed by the nodes in a fully self-organized manner.

Each certificate is issued with a limited validity period and therefore contains its issuing and expiration times. Before a certificate expires, its issuer issues an updated version of the same certificate, which contains an extended expiration time. We call this updated version the certificate update. Each node periodically issues certificate updates, as long as its owner considers that the user-key bindings contained in these certificates are correct.

3. PROPOSED SYSTEM

In our system, key authentication is performed via chains of public-key certificates in the following way. When a user u wants to obtain the public key of another user v, she acquires a chain of valid public-key certificates such that:

The first certificate of the chain can be directly verified by u, by using a public key that u holds and trusts (e.g., her own public key).The last certificate contains the public key of the target user v.

To correctly perform authentication via a certificate chain, a node needs to check that: all the certificates on the chain are valid (i.e., have not been revoked), and All the certificates on the chain are correct (i.e., not false; the certificates contain correct user-key bindings).

To find appropriate certificate chains to other users, each node maintains two local certificate repositories: the non-updated certificate repository and the updated certificate repository. The non-updated certificate repository of a node contains expired certificates that the node does not keep updated. The reason for collecting and not updating expired certificates is that most of the certificates will permanently

be renewed by their issuers, and only a few will be revoked. Therefore, the non-updated repositories provide the nodes with a very good estimate of the certificate graph.

The updated certificate repository of a node contains a subset of certificates that the node keeps updated. This means that the node requests the updates for the certificates contained in its updated repository from their issuers, when or before they expire. The selection of certificates into the node's updated repository is performed according to an appropriate algorithm. When a user u wants to authenticate a public key Kv of another user v, both nodes merge their updated certificate repositories and u tries to find a certificate chain to v in the merged repository. If found, this chain contains only updated certificates because it is constructed in the updated repositories. To authenticate Kv, u then further checks whether the certificates on the chain have been revoked (since the last update) and the user-key bindings in the certificates are correct .u performs both validity and correctness checks locally. We use an algorithm for the construction of users' updated repositories that we call Maximum Degree. *Through simulations, with this algorithm, there is a high probability of finding certificate chains between the users in their merged updated repositories even if the size of the users' updated repositories is small.

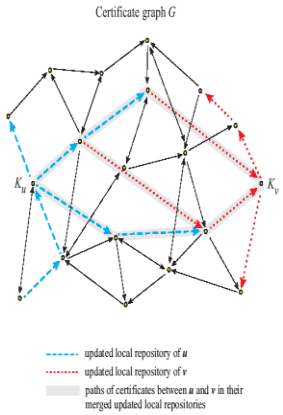


Figure 2. A certificate graph, and paths of certificates between users u and v in their merged updated local repositories.

If the authentication of Kv through the updated certificate repositories fails, node u tries to find certificate chains to v in its (u's) joint updated and non-updated repositories. If u finds a chain to v, this chain will likely contain some expired certificates, because it is constructed in the updated and non-updated repositories. To complete the authentication, u requests, from their issuers, the updates of the expired certificates that lay on the chain and checks their correctness. If the certificates are both valid and correct, u authenticates Kv. Here again, u performs the certificate correctness check locally. If node u cannot find any certificate chain to Kv, it aborts the authentication.

In our system, certificate revocation is an important mechanism. We enable two types of certificate revocation: explicit and implicit. The issuer explicitly revokes a certificate by issuing a revocation statement and by sending it to the nodes who stored the certificate in question. The implicit revocation relies on the expiration time contained in the certificates. Every certificate whose expiration time passes is implicitly revoked; this second mechanism is straightforward, but requires some loose time synchronization of the nodes.

3.1 Coping with misbehaving users

A dishonest user may try to trick other users into believing in a false user-key binding by issuing false certificates. She may issue several types of false certificates. First, she may issue a certificate that binds a key Kv to a user f instead of to user v. In this way, a dishonest user may trick other users to believe that Kv is the public key of user f, when it is really the public key of user v.

Second, she may issue a certificate that binds user v to a false key K_v, which may then cause other users to believe that K_v is indeed the key of user v.

Third, a malicious user can invent a number of user names and public keys and bind them by appropriate certificates. The malicious user can then use these public keys to issue false certificates and try to convince a given user that the certificates are correct, as they were signed by many other users. we will prevents these attacks by allowing nodes to detect inconsistent certificates and to determine which user-key bindings are correct. The certificate exchange mechanism allows nodes to gather virtually all certificates from G. This enables nodes to cross-check user-key bindings in certificates that they hold and to detect any inconsistencies (i.e. conflicting certificates).Two certificates are considered to be conflicting if they contain inconsistent user-key bindings (i.e. if both certificates contain the same username but different public-keys, or if they contain the same public-key, but are bound to different usernames).If a certificate received by a node u contains a user-key binding (v,Kv) not contained in any certificate in the updated and non-updated certificate repositories of u, then (v,Kv) and the certificates that certify it are labeled by u as un-specified. A certificate labelled un-specified means that the node does not have enough information to assess whether the user-key binding in the certificate is correct. From the moment that (v,Kv) is received, u waits for a predefined period TP . If within this period u does not receive any conflicting certificates regarding (v,Kv), the status, of this binding and of the certificate that certifies it, changes to non-conflicting. Here, we note that TP needs to be longer than the expected certificate exchange convergence time TCE. If indeed TP > TCE, nodes will detect inconsistent certificates for all users that exist in the network.

For this, each node initially issues a self-signed certificate and exchanges it with other nodes by the certificate exchange mechanism. Thus, the waiting period TP is actually the expected time for any self-signed certificate to reach all the nodes in the network.

To resolve the conflict, u tries to find chains of non-conflicting and valid certificates to public-keys Kv and K_v. Based on the characteristics of the certificate paths (i.e., their number and length), two confidence values that show the user's confidence in the correctness of the two bindings are computed. The two values are then compared and one user-key binding is labelled non-conflicting and the other is labelled false.

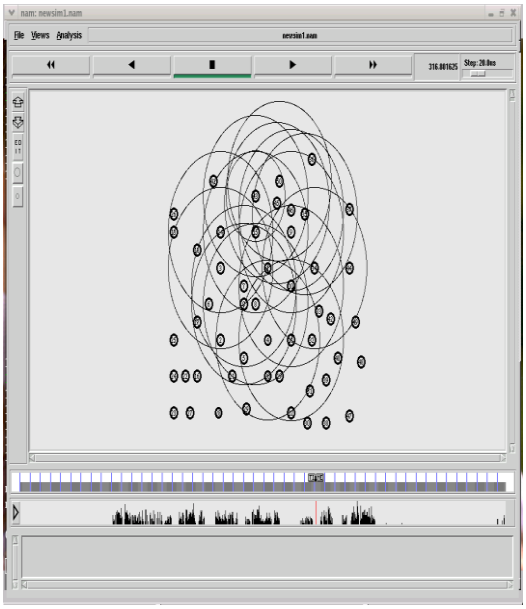
3.2 Topology and platform chosen:

Hardware:- p4 512mb ram
Software :- redhat linux ws3 (2.4 kernel with gcc --)
Ns -2.28 network simulator
Topology :-plane size 700x700m

a) 50 nodes, stationary for 500 seconds
b) 50 nodes, 7 moving for 500 seconds
c) 50 nodes, 14 moving for 500 seconds

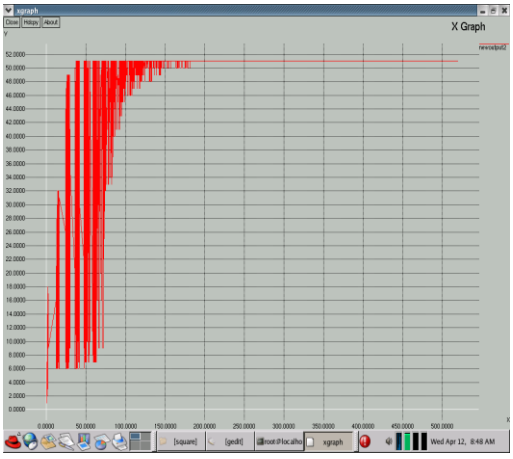
for the comparison between older and newer dsdv and to track the effect of motion on repository building.

4. THE NAM INTERFACE

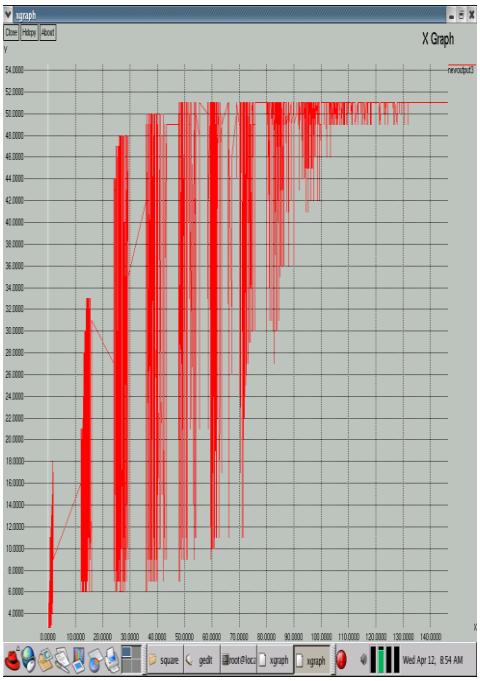


5. RESULT

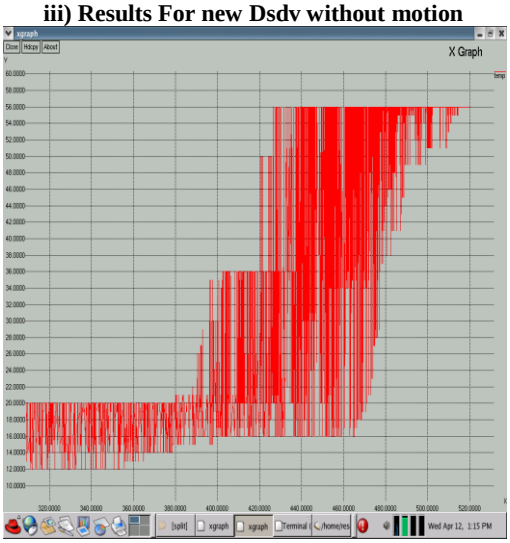
i) For new Dsdv with 7 nodes in motion



ii) Results for new Dsdv with 14 nodes in motion



iii) Results For new Dsdv without motion



iv) Comparison of all graphs of new Dsdv with motion

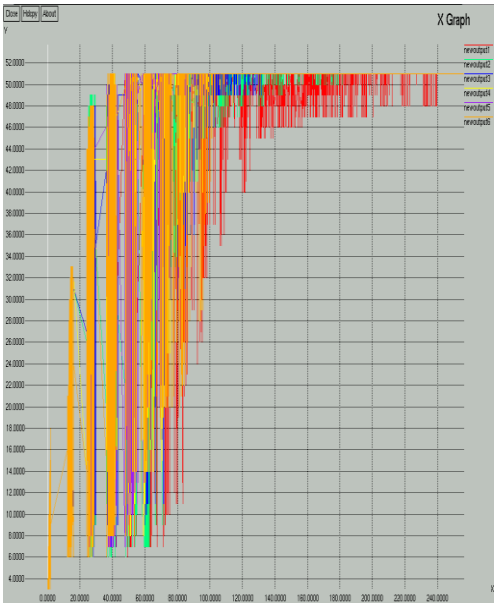


Table 1: comparison of Old Dsdv and new Dsdv with various parameters

Repository build up time		
Plane size 700*700	**Old DSDV**	**New DSDV**
Stationary	96	240
7 Nodes moving	84	182
14 Nodes moving	81	132
21 Nodes moving	86	106
28 Nodes moving	83	102
35 Nodes moving	88	118
Nodes Starts moving from 360 seconds		
Plane Size 700*1500	**Old DSDV**	**New DSDV**
Effect	370	400
Repository builds up	481	513
Faulty node		
Faulty Node	**Fault Free**	**Faulty**
	240	140

6. CONCLUSION

In this way we implemented self-organized key management in mobile ad-hoc networks. Key exchange done by checking the chain appended with entry and based on trust relationship. The DSDV protocol do not purges any record, it becomes critical to create updated and non-updated repository. There is no way to get the path to particular node until we have entry associated with that destination. When the entry is available we also have key associated with it. So we don't find need to use load balancing. We have implemented fault handling strategy. Whenever fault is found we get correct data from previously known fault free node. This mechanism some time leads to triggered update which cause the generation of quickly but increases the router data traffic. As shown in results we have taken observations for various network parameters. We found that time required to reach key to other node, in a 50 node network, varies between 250 to 5000 seconds. So while distributing the new keys care should be taken that the key is distributed at least 5000 seconds before it is used for some security operations.

7. REFERENCES

[1] Self-organized public-key management Adhoc networks by *Srdjan Capkun*, student member, IEEE, *levente Buttya'* n, student member, IEEE, and *Jean-Pierre Hubaux*, Senior Member IEEE

[2] Key management in Ad Hoc networks by *Klas Fokine* 2002-09-11

[3] The ns manual (formerly ns Nots and Documentation) The VINT project A collaboration between researchrsbat US Berkeley , LBL, USC/ISI, and Xerox PARC. *Kevin Fall hkfall@ee.lbl.govi*

[4] Implementing a New Manet Unicast Routing Protocol in NS2, *Francisco J. Ros Pedro M. Ruiz*, Dept. of Information and Communications Engineering university of Murica

[5] Locality Driven Key management Architecture for Mobile Ad-hoc Networks. *Gang xu and Liviu Iftode* Department of Computer Science, Rutgers University.

[6] Distributed Symmetric key Management for Mobile Ad hoc Networks. *Aldar C-F. Chan Edwards S. Rogers Sr.* Department of Electrical and Computer Engineering University of Toronto.

[7] Shared RSA Key Generation In A Mobile Ad hoc Network*. *B.Lehane and L.Doyle, D.O' Mahony*, Trinity College Dublin, Ireland