Universität Karlsruhe (TH)

Design and Evaluation of a Secure Self-Organizing Routing Protocol

Student Thesis
System Architecture Group
Department of Computer Science
Universität Karlsruhe (TH)

Author
cand. inform.
Christian Wallenta

Advisors:

Prof. Dr. Frank Bellosa
Dipl.-Ing. Kendy Kutzner

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 19. August 2005

Contents

1. Introduction

Routing security is an important subject. An excellent routing protocol with strong performance and low demands for resources is useless if very few malicious nodes can easily and successfully attack the network. As a result it is very important to analyze an existing protocol towards possible attacks. Furthermore when designing new routing protocols, security should be one of the main design goals since it is easier to design a secure protocol from scratch than to customize an existing protocol.

1.1 Goals of this Study

A new routing protocol for self-organizing networks [Fuh04] is examined. Messages in the protocol are not authenticated and it is not possible to check their integrity.

The goal of this study is to enhance the routing protocol to make authentication and integrity checks possible. Asymmetric cryptography and digital signatures shall be used to achieve this goal

The current implementation of the unsecured protocol is modified with regard to generation and verification of digital signatures. The network, then including nodes that compute asymmetric cryptographic algorithms, should be analyzed. Estimations about the scalability of the modified network are supposed to be made.

1.2 Organization of this Study

This study begins with a section containing basic terms about cryptography, network security and a brief introduction to the current protocol.

Afterwards the protocol is analyzed with regard to general attacks and protocol specific problems. Some countermeasures are proposed with their advantages and disadvantages.

The design chapter introduces and motivates the modifications and enhancements of the protocol.

Implementation and evaluation is the subject of chapter 4 and 5. The evaluation analyzes the effects of the protocol modifications with regard to routing performance.

The study closes with a short conclusion and an outlook on future work.

2. Analysis

The chapter starts with a brief overview about basic cryptographic terms like asymmetric cryptography, digital signatures and hash functions. Afterwards the considered protocol [Fuh04], the most important protocol messages and functions are introduced.

The following section describes general attacks on peer-to-peer and ad-hoc networks. The protocol is analyzed with regard to these general attacks.

The chapter is completed by showing possible solutions concerning their advantages and disadvantages. A full design of the enhanced protocol is the subject of chapter 3.

2.1 Basics Terms

2.1.1 Secret Key Cryptography

Secret key cryptography, also called symmetric cryptography, is a form of cryptography which uses a single key to encrypt and decrypt messages. Besides encryption secret key cryptography can also be used for authentication. One of such authentication techniques is called *message authentication code (MAC)*.

Assume Alice wants to send an encrypted message to Bob using a secret key K_{sym}. The main problem is to exchange the key since Bob needs K_{sym} to decrypt Alice's message. This requires that Alice and Bob initially communicate over a secure channel to exchange the secret key. If both parties exchanged the key, Alice is able to send Bob an encrypted message

$$C = Encrypt(Message, K_{sym}).$$

Bob who is aware of the secret key K_{sym} now can decrypt the message

$$Message = Decrypt(C, K_{sym}).$$

The major advantage of secret key cryptography is that it is faster than public key cryptography in most cases. The main disadvantage is the secret key exchange.

2.1.2 Public Key Cryptography

Public key systems are primarily used for encryption and digital signatures (see 2.1.4).

The main challenge of secret key cryptography, that both parties first must agree on a secret key, is solved. Each party maintains a pair of keys including a public key $K_{Alice,pub}$ respectively $K_{Bob,pub}$ and a private key $K_{Alice,priv}$ respectively $K_{Bob,priv}$. The public key is published while the private key is kept secret. Since public key and private key are linked mathematically, it is hypothetically possible to derive the private key from the public key but in good public key system this challenge is computationally infeasible.

Now Alice is able to send a confidential message to Bob by encrypting the message M with Bob's public key

$$C = Encrypt(M, K_{Bob,pub}).$$

Bob decrypts the message with his private key

$$M = Decrypt(C, K_{Bob,priv}).$$

An example for an asymmetric cryptographic system is RSA [rsc05] which is named after the developers (Rivest, Shamir and Adleman). RSA is used for digital signatures and encrypting messages.

There is no need to exchange secret keys over a secure channel. This simplification is the major benefit of asymmetric systems. Alice may look up Bob's public key in a public directory. On the other hand the main disadvantage is that asymmetric systems are generally much slower than secret key systems. Furthermore it is not that simple to obtain a trustworthy public key. How can Alice be sure that the key $K_{Bob,pub}$ belongs to Bob's identity? It is possible that this key belongs to an attacker who claims to be Bob.

An asymmetric cryptographic system must ensure a trustworthy mapping between identity and public key. This problem is discussed again later when secure node ID assignment is considered.

2.1.3 Hash Functions

This following introduction is taken from [rsc05].

> A hash function H is a transformation that takes an input m and returns a fixed-size string, which is called the hash value h (that is, $h = H(m)$). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography, the hash functions are usually chosen to have some additional properties.
>
> The basic requirements for a cryptographic hash function are as follows.

- The input can be of any length.
- The output has a fixed length.
- H(x) is relatively easy to compute for any given x.
- H(x) is one-way.
- H(x) is collision-free.

A hash function H is said to be one-way if it is hard to invert, where 'hard to invert' means that given a hash value h, it is computationally infeasible to find some input x such that $H(x) = h$. If, given a message x, it is computationally infeasible to find a message y not equal to x such that $H(x) = H(y)$, then H is said to be a weakly collision-free hash function. A strongly collision-free hash function H is one for which it is computationally infeasible to find any two messages x and y such that $H(x) = H(y)$.

The hash value represents concisely the longer message or document from which it was computed; this value is called the message digest. One can think of a message digest as a 'digital fingerprint' of the larger document. Examples of well known hash functions are MD2, MD5 and SHA.

Perhaps the main role of a cryptographic hash function is in the provision of message integrity checks and digital signatures. Since hash functions are generally faster than encryption or digital signature algorithms, it is typical to compute the digital signature or integrity check to some document by applying cryptographic processing to the document's hash value, which is small compared to the document itself. Additionally, a digest can be made public without revealing the contents of the document from which it is derived. This is important in digital timestamping where, using hash functions, one can get a document timestamped without revealing its contents to the timestamping service.

2.1.4 Digital Signatures

Authentication is a process that proves and verifies a certain information, for example the information about the origin or the sender of a document or message. A digital signature of an electronic document or message can prove such information.

If a public key system is used to generate a digital signature, the signature is computed from the message and the signer's private key. First, a hash function is used to derive a digital fingerprint from the message called the 'message digest'

$$Digest = Hash(Message).$$

Afterwards Alice who wants to sign the message encrypts the message digest with her private key

$$Signature = Encrypt(Digest, K_{Alice,sign}).$$

The message together with the signature is sent to the receiver Bob who wants to be sure that Alice is really the sender.

Bob must know which hash function Alice has used to derive the digest. Bob takes the message and computes the digest again

$$Digest' = Hash(Message).$$

Additionally he decrypts the signature with Alice's public key and is now able to compare Alice's digest to the one he has derived. If

$$Digest' = Digest$$

then Bob can be sure that Alice is the author of the message.

Digital signatures rely on the fact that Alice is the only person that is able to derive the signature since nobody else is aware of her private key. Furthermore the hash function is required to be collision-free because when Alice signs the message digest she actually signs every message with the digest *Digest*.

The problem regarding the mapping between public key and identity (described in 2.1.2) is a problem in the range of digital signatures, too.

2.1.5 Digital Certificates

In section 2.1.2 the problem regarding the trustworthy mapping between identity and public key was introduced. This problem can be solved with digital certificates.

Bob wants to check if the key K_{Alice} really belongs to identity 'Alice'. Bob is not able to do this verification but a trusted third party can do it. Such a certification authority checks Alice's identity for instance by inspecting her identity card and ensures oneself that Alice is holding the private key that matches the public key K_{Alice}. Alice has to appear at the certification authority and make a proof about the key and her identity.

After checking Alice's identity the certification authority issues her a digital ID certificate which ensures the correct mapping between the ID 'Alice' and the public key K_{Alice}. This certificate is signed by the certification authority which is a trusted third party.

Bob trusts the certification authority and is aware of its public key. When Alice wants to prove that the public key is trustworthy she passes Bob the ID certificate. Bob is able to check the signature of the certification authority and can be sure that he holds the correct public key.

Digital certificates are also used to map other facts than identities and public keys. Authorization certificates for example map grant access permissions to public keys.

2.1.6 Challenge Response Authentication

Challenge response procedures are used to prove the identity of a person or entity towards another. Alice wants to prove her identity to Bob or the fact that she is holding a shared secret. For that purpose Bob sends a challenge to Alice who is the only one that is able to solve that challenge. The solution is sent back. This message is called the response. Bob checks if the solution is correct and is now sure of Alice's identity [Buc01].

A challenge response procedure can either use symmetric or asymmetric cryptographic systems. In the case of the symmetric alternative Alice and Bob share a secret key K_{shared}. Bob sends a random number r to Alice. For example, Alice computes the hash of r and encrypts the result using the shared key

$$c = Encrypt(Hash(r), K_{shared}).$$

The result is sent back to Bob.

Bob is aware of the hash function and computes

$$c' = Encrypt(Hash(r), K_{shared})$$

because he holds the shared key K_{shared}. Bob can now check if the response c matches c'. In that case Bob can be sure that Alice holds the secret key because she was able to solve the challenge.

If asymmetric cryptography is used Alice signs a message such as $(Alice, Hash(r)_{signed})$. This messages is sent back to Bob as the response. Bob is aware of Alice's public key and is able to verify her signature. He can be sure that his counterpart is Alice because she should be the only one who is holding the private signing key. If Alice lost her private key or has given it to somebody else, a trustworthy authentication is not longer possible.

Again we encounter the problem described above regarding the trustworthy mapping between Alice's identity and her public key. If an attacker has the ability to substitute Alice's public key with his own public key, he might be able to pretend Alice's identity [Buc01].

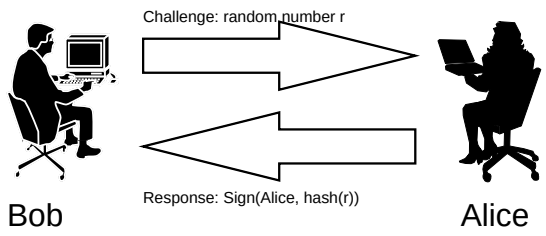


Figure 2.1: Challenge response authentication with a public key system

2.1.7 Source Routing

Source routing is a technique whereby the sender of a packet preselects the route that the packet should follow when traveling through the network. The packet contains a route towards the destination and each intermediate node should be able to forward it along the given path.

The sender of the packet must be aware of enough information about the network to choose a source route. Sometimes there is a distinction between strict source routing and loose source routing.

Strict source routing means that the sender specifies a complete route to the destination. For example:

- source: node 5
- destination: node 9
- route: node 5 $\longrightarrow$ node 7 $\longrightarrow$ node 4 $\longrightarrow$ node 9

If loose source routing is used the sender does not choose the complete route. He only selects some hops that the message should pass through. For instance:

- source: node 5
- destination: node 9
- via: node 4

2.2 The Protocol

2.2.1 Overview

The core idea of the given protocol [Fuh04] is to combine overlay routing techniques with source routing. The knowledge about the network topology is distributed over all nodes in the network. This reduces the routing table size in each node.

Each node maintains only a small routing table with few entries. The address space, the routing table and the protocol messages are introduced in the next sections.

2.2.2 The Address Space

The address space is circularly connected and builds up a ring with a defined orientation. For each node there is a successor and a predecessor in the ID space. Finding these two special nodes is the challenge of the bootstrapping phase (see 2.2.4).

Assume {2,10,22,31} to be a set of nodes. Then 2 is the successor of 31. In the address ring there is no such relation as ‘larger’ or ‘smaller’ between the nodes, but they can form a correct sequence like (22-31-2) or an incorrect sequence like (2-31-10).

The distances between nodes in the address space are asymmetric. If {0..39} is the complete ID space then $d(2, 10) = 8$ but $d(10, 2) = 32$. The protocol distinguishes between distances in the ID space and distances in the network which are defined as the minimal length of a path between two nodes measured in hops.

2.2.3 The Path Store Data Structure

Each node maintains a data structure called the 'path store' that stores the known source routes. If a packet contains a source route towards the destination, an intermediate node is able to forward it along the given route. Otherwise it looks up a route in the path store and prepends it.

If A is the intermediate node and D is the destination, the look-up in the path store for the next node B is done considering the following aspects in descending order.

1. (A,B,D) is a correct sequence
2. the source route is minimal
3. d(A,B) is maximal

If there is no node that matches criteria 1) the destination D is considered as unreachable. The detailed operations on the path store are described in [Fuh04].

2.2.4 The Bootstrapping Phase

The goal of the bootstrapping phase is a network with a consistent state where each node is able to forward a message towards the destination. For this reason each node must be aware of its correct successor in the ID space.

First the node needs to know who are its direct topological neighbors. This knowledge is gained from some kind of 'neighbor discovery' procedure. For that purpose the *Hello-Message* is sent, which is described in 2.2.4.1. After that the node chooses the best current successor node and informs it with a *SuccessorNotification-Message* (2.2.4.2). The chosen node either agrees that it is the correct successor or knows a better one. If there is a better one the sender of the *SuccessorNotification-Message* is informed about that fact by a *SuccessorUpdate-Message* (2.2.4.3). Step by step the network approaches a consistent state where each node is aware of its correct successor. The idea of this iterative bootstrapping phase is described in [CF04].

In most cases the bootstrapping phase leads to a consistent state where each node knows a source route to its successor. Inconsistencies come up with very low probability. Each node learns from the messages that are passing through and is so able to fill the path store very quickly (see [Fuh04] and [CF04] for simulation results). So the network is now able to forward all messages correctly towards the destination.

2.2.4.1 The Hello-Message

The *Hello-Message* as described above is used to detect all direct neighbors of a node. Each node announces itself to its direct neighbors by sending a *Hello-Message* containing the own address.

A node that receives such a *Hello-Message* stores the node ID and the one-hop source route into the path store. Assuming that no messages are dropped each node is aware of all direct neighbors after a certain time. The set of neighborhood nodes contains the best current successor which is informed using a *SuccessorNotification-Message* (2.2.4.2).

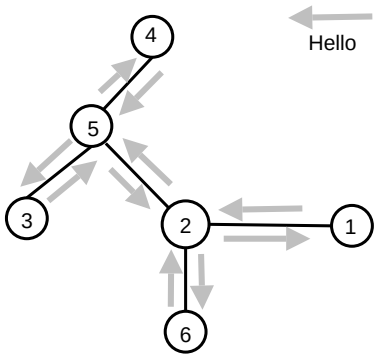


Figure 2.2: Exchanging Hello-Messages

2.2.4.2 The SuccessorNotification-Message

Having received all *Hello-Message*s a node is able to decide which neighbor node is the proper successor from its point of view. This node is informed about the choice by means of a *SuccessorNotification-Message*. This message always contains the complete route towards the destination which is a one-hop route for the first time. An intermediate node is able to forward this message along the given route.

The receiver of the *SuccessorNotification-Message* stores the route into the path store and checks if the choice is right or if there is a better successor for the sender.

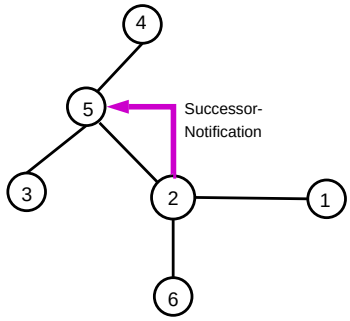


Figure 2.3: SN-Message from node 2 to node 5

In the case of a better successor the node sends a *SuccessorUpdate-Message* (2.2.4.3) back to the sender of the *SuccessorNotification-Message*. This update message contains information about the better successor like node ID and source route. Otherwise it accepts the choice and updates the predecessor pointer. Before doing so the old predecessor is informed about the new situation by means of a *SuccessorUpdate-Message*.

2.2.4.3 The SuccessorUpdate-Message

The purpose of a *SuccessorUpdate-Message* is to inform a node about a better successor. Either a choice of successor is corrected or a predecessor is informed about the existence of a better successor.

In the example of 2.2.4.2 node 2 sends a *SuccessorNotification-Message* to node 5 because 2 believes that node 5 is the best current successor. Node 5 does not agree

because it is aware of the existence of node 3. For this reason a *SuccessorUpdate-Message* is sent from node 5 to node 2 containing a source route to node 3. Thereupon node 2 sends a *SuccessorNotification-Message* to node 3. Figure 2.4 makes this message flow clear.

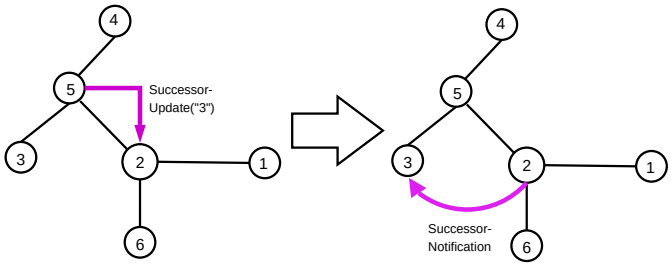


Figure 2.4: Example of a SuccessorUpdate-Message

2.2.5 The Routing Phase

After the bootstrapping phase (2.2.4) the network is in a consistent state where each node knows a source route to its successor. Additionally each node learns from messages passing through and the path store fills up with source routes very quickly.

Assume that node A wants to communicate with node B. It builds up a 'connection' where connection means to find a source route from A to B. To achieve this goal A generates a *ConnectionRequest-Message* containing a path towards B and B's address. The path can be the null-path for the first time in case that A does not know a source route towards B.

There are three cases of handling such a *ConnectionRequest-Message*. The destination node B of the message stores the complete source route into its path store and sends back an *Established* message to A. Any inner node of the path is able to forward the message along the given path. Otherwise the node looks up the best matching route towards the destination in the path store. If there is no such route the node is unreachable assuming that the network is in a globally consistent state. In this case an *Unreachable* message is sent back to A.

2.3 A Protocol Setting

This section describes a possible scenario for an application of the protocol (2.2). This protocol setting is analyzed and considered when the design for the enhanced protocol is discussed.

In a simple setting all nodes have bidirectional point-to-point links to other nodes. The author of the protocol assumes static, reliable and bidirectional point-to-point links [Fuh04]. Imagine a home or an office where miscellaneous electronical devices are connected by point-to-point links. Sensors, control systems and switches might be such devices which build up a network all over the house. There is no need for a central control unit to manage the network since the routing is self-organizing. The nodes build up and maintain all routing information by themselves.

When the network is established the topology does not change very often. The nodes normally do not change their position. The topology changes when a node shuts down, fails or a new node joins the network.

Each node in the network has as much neighbors as active interfaces. The neighbor discovery is much easier than for example in wireless networks. If there is a node behind an interface it is a neighbor. Problems with that assumption will be discussed later.

This setting is very general and an infrastructure can not be assumed. So problems such as secure node ID assignment must be solved.

2.4 Problems and Possible Attacks

This section introduces some general attacks towards ad-hoc and peer-to-peer networks. These attacks are analyzed with regard to the current protocol and the introduced protocol setting. Furthermore the current protocol messages and effects of fake protocol messages are examined.

2.4.1 Node ID Assignment

The node ID assignment is an important subject in peer-to-peer systems since many peer-to-peer systems achieve high availability by redundancy mechanisms such as replicated keys. This mechanism ensures that a key region is available even if the node which is responsible for that region fails. This is possible because the key region is replicated. This redundancy must be distributed independently over the network.

An attacker does not need to control a big fraction of the network to start an effective attack towards such a redundancy mechanism. Controlling all nodes which are responsible for all replicated keys is sufficient. This can be very simple if the attacker is able to choose node IDs without any constraints. If he can choose special node IDs he may be in the position to control all replicated keys. That means that he is able to manipulate and delete data or just control and deny access to it [CDG+02].

Another attack aims at the routing table of a victim. The attacker tries to control nearly any node ID in the routing table of the victim. If all entries in the table point towards a hostile node, the victim's access to the network is controlled by the attacker.

The two described attacks (see [CDG+02]) are always possible if node IDs are not assigned randomly, but instead each node is able to choose an ID for itself. In reality not all nodes behave friendly and generate a random number as ID. As a result a secure protocol has to take measures about node ID generation and assignment. Solutions for this problem will be introduced and discussed in section 2.5.1.

The described attacks are simplified if an attacker can easily obtain many node IDs. Then he can simulate multiple nodes which have legitimate IDs with only one physical node. This attack is called the 'Sybil attack' and is characterized in 2.4.2.

2.4.2 The Sybil Attack

The Sybil attack is characterized in [Dou02], [NSSP04] and [KW03] for example. A single node simulates multiple identities. The goal of this attack is in most cases to attack the redundancy mechanisms of peer-to-peer systems as described in 2.4.1.

A redundancy mechanism is senseless if an attacker can obtain many node IDs. By simulating a large number of node IDs the attacker might be able to gain control of a

complete region of the network. The consequences are almost the same as choosing specific IDs. The attack towards the routing table of a victim is also possible.

A combination of the Sybil attack and selectively choosing node IDs may increase the chances of success.

The attacker may also want to control as much network traffic as possible [KW03]. To achieve this goal the network traffic must pass through the attacker's node. If a single node simulates multiple identities, many packets use routes that contain these virtual hostile nodes. In fact, the packets pass through only one single or several real nodes. Without securing node ID assignment this attack is very simple.

In the current protocol the attacker might confuse a neighbor node by sending multiple *Hello-Message*s from a single node. If the victim does not recognize that all messages are received from the same interface which is very hard in the wireless network case, one node might simulate multiple neighbors.

There is a high probability that one of these virtual neighbors is selected as the successor since the chance increases with the number of simulated neighbors. It is possible to disturb the following bootstrapping phase by sending *SuccessorNotification-* or *SuccessorUpdate-Message*s in the name of virtual nodes as well.

The attack gets more difficult if there is a secure link-layer. An attacker can not fake the link-layer address and the victim might recognize that there are many nodes behind one interface. Suggestions for countermeasures against the Sybil attack are the subjects of the sections 2.5.1 and 2.5.2.

2.4.3 The Wormhole Attack

When performing a wormhole attack ([HPJ03b],[HPJ02],[WB04],[KW03]) an attacker receives and records packets or parts of packets at one point in the network and 'tunnels' them to another point in the network [HPJ02]. These packets are replayed from that point in the network. The attacker may possibly tunnel only a few bits through his wormhole. For the receiver of these bits or the complete packet it seems as if the sender is only a few hops away, but in reality the sender is not that close. This attack is very hard to detect because the attacker in most cases does not use the network but private links outside the network like wireless links.

An attack towards the current protocol may look like this. The attacker records the *Hello-Message* at one point from his neighbor, tunnels them using his wormhole and replays the packet at another point. Figure 2.5 shows such an attack. Node A and node B are not neighbors in this set. Nevertheless after the wormhole attack they believe that they are direct neighbors. The attacker tunnels the *Hello-Message* from node A towards node B and does the same with the message from node B. Both nodes are now aware of each other and believe that they are direct neighbors. This 'knowledge' may lead to routing inconsistencies and problems when A and B try to identify the correct successor.

Of course an attacker is able to fake a *Hello-Message* and there is no need to record and tunnel a packet. The problem with the wormhole attack is that it may deal with security mechanisms. Assume that it is not possible anymore to fake messages, because a packet needs a correct signature. Performing a wormhole attack still makes the described attack on the neighbor discovery possible. It is not necessary

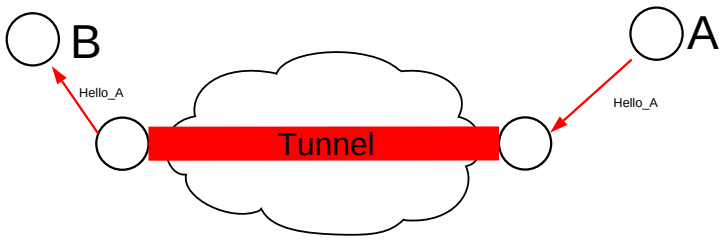


Figure 2.5: Tunneling a Hello-Message

to fake the signature. Detecting and preventing wormhole attacks is very hard to achieve. In 2.5.3 'packet leashes' will be introduced. This mechanism is able to detect a wormhole attack.

2.4.4 The Sinkhole Attack

The goal of the sinkhole attack ([KW03],[Bur03]) is to obtain as much network traffic as possible to get access to data and information. The attack may help to do selective forwarding (see 2.4.5) or to do eavesdropping on the communication. In this case selective forwarding for instance is very simple because the node obtains nearly the complete traffic towards or from its neighbors. So it is possible to listen to, to modify or to suppress packets [KW03].

The goal is not achieved by simulating as many nodes as possible (Sybil attack 2.4.2) or by choosing node IDs carefully. The node ID assignment may be secure and randomized. Nevertheless the attacker is able to perform this sinkhole attack. The attack works by making a compromised node look very attractive to other nodes with respect to the routing algorithm [KW03]. For that purpose the attacker communicates very fast, cheap or high quality routes to all his neighbors. Many nodes will try to use routes that pass through the attacker's node because it is very attractive with respect to the routing metric. Automatically much traffic will pass the compromised node and the attacker is able to perform other attacks.

To achieve this goal in the given protocol there are a couple of ways. At first the attacker can fake *Hello-Message*s because these packets are not checked for integrity respectively authenticity. As a result the attacker is stored in the neighborhood set of some nodes.

Faking *Hello-* and *SuccessorUpdate-Message*s is possible in the current protocol because there is no integrity check. A receiver can not be sure which node has sent a message and whether the message has been modified or not. Besides there is no verification of the suggested route.

A misbehavior during the bootstrapping phase such as the following may lead to a state in which a node has more than one predecessor. This can be an advantage. During the bootstrapping phase a node receives more than one *SuccessorNotification-Message* with high probability. Behaving correct the node informs the sender about the better successor with a *SuccessorUpdate-Message*. The sender will then try to contact the suggested node.

What happens if the receiver of the *SuccessorNotification-Message* does not inform the sender about the better successor but always accepts the choice? The sender

will stop searching for the correct successor because there is no information about a better one. As a result the compromised node has more than one predecessor and there will be more traffic passing through this node.

2.4.5 Selective Forwarding

This attack [KW03] is performed by forwarding packets selectively towards their destination. An attacker may suppress or disturb the traffic towards his victim. A simple way to realize this attack is to drop all packets. But operating as a 'black hole' [KW03] might lead surrounding nodes to the assumption that the node is down. As a result they will seek for another route. A better way is to forward packets selectively. An attacker can forward the main traffic but refuse to forward the traffic towards or from his victims.

The attack is very hard to detect if the attacker forwards the main traffic because the surrounding nodes will not conclude that the link is down. Detecting this attack means detecting the malicious node. The attacker himself may detect such detecting mechanisms and deal with them. On the other hand this attack is hard to perform. Each packet passing through the node has to be analyzed and forwarded with respect to the structure, sender and receiver. This requires a powerful attacker.

Selective forwarding is simplified by performing attacks like the Sybil attack or the sinkhole attack. If much traffic passes through the malicious node there are more possibilities for the attacker to influence the network traffic.

Selective forwarding during the bootstrapping phase may lead to an inconsistent state or at least disturb or delay building up the correct routing tables or path stores respectively. Dropped *SuccessorNotification-* or *SuccessorUpdate-Message*s may lead to trouble. Deleting bootstrapping messages is discussed in 2.4.8 and 2.4.9 where the messages are analyzed with respect to possible attacks.

2.4.6 Tampering with Packets

Modifying a packet is always possible when a packet passes through a node. Each node is able to modify, cut or drop a packet instead of just forwarding it. Most of the attacks described in 2.4 do modify or fake packets to achieve their goal. So this subsection is about tampering packets for no purpose.

A node cuts or deletes a part of the message for either no reason or because of being a faulty node. Some attackers might want to do damage without embarking a strategy.

Preventing such an attack is nearly impossible. It is much easier to detect a faulty packet. Solutions and proposals to protect packets are discussed in section 2.5.5.

2.4.7 The Hello-Message

The attacker may generate a *Hello-Message* and fake the sender address. Currently no node is able to verify that a received message was really sent by the claimed sender.

Furthermore a compromised node can refuse to send *Hello-Message*s to all or several neighbors. Keeping this information back may cause problems when the neighbors try to find the correct successor.

In addition an attacker could just forward the *Hello-Message*s. Normally the messages should not pass more than one hop. If a malicious node forwards a received message the next node might presume that the origin is a direct neighbor. This is only possible if a node can not detect this misbehavior on the link layer.

If there are point-to-point links the victim may notice that there is more than one node behind an interface. Maybe the node can determine that the message passes more than one hop.

2.4.8 The SuccessorNotification-Message

1. Faking the sender

   If a malicious node generates a *SuccessorNotification-Message* containing a bogus sender address this will cause problems because currently there are no acknowledgments. The receiver will change its predecessor and inform the old predecessor about the new situation with a *SuccessorUpdate-Message* (see 2.2.4.3). This action may lead to an inconsistent state because it will probably initiate new bootstrapping messages. Anyway there will be more network traffic than necessary.

   Figure 2.6 shows the described attack.

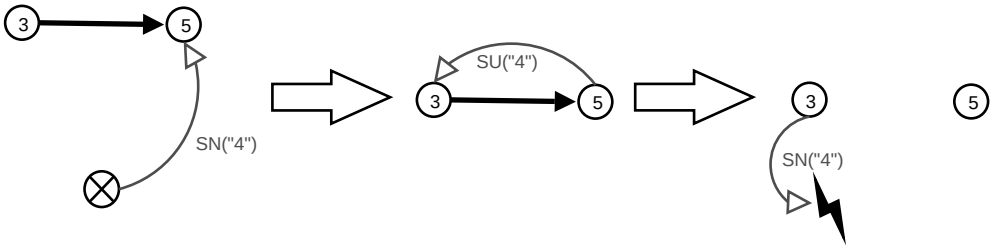


Figure 2.6: Faking a sender of a SN-Message

2. Deleting / Dropping the message

   An intermediate node may just drop or delete the message respectively. Currently the protocol does not use any acknowledges. As a result deleting messages may lead to an inconsistent state.

   Assume that node 5 sends a *SuccessorNotification-Message* to node 8. If an inner node drops this message, node 5 will assume that node 8 agrees with the choice because node 5 does not receive a *SuccessorUpdate-Message*.

   Figure 2.7 illustrates this attack.

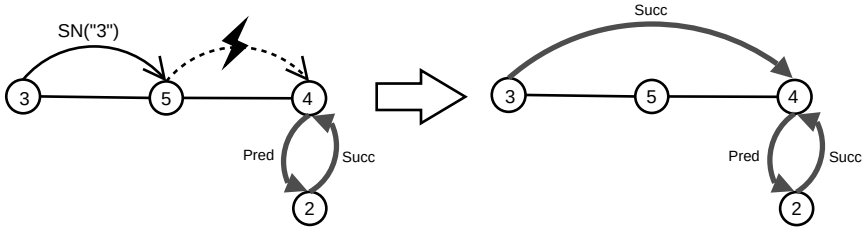


Figure 2.7: Dropping the message causes problems.

3. Modifying the message

   An inner node may change the sender, the receiver or any part of the message. This is a general attack. Each node is able to modify or cut a message.

4. Replay attack

   A malicious node may record a message that passes through the node and replay the message later. If the state of the network has changed in the meantime the attacker may cause confusion.

   The current protocol does not use signatures to protect packets against tampering. If signatures are introduced it is important to keep replay attacks in mind. The signature is useless if an attacker is able to record and replay the packet because that attack does not change the signature and a receiver does not realize that the packet is out-of-date.

2.4.9 The SuccessorUpdate-Message

1. Faking the sender

   A *SuccessorUpdate-Message* containing a fake sender may cause confusion as well. In figure 2.8 node 5 is the current successor of node 3. Now an attacker sends a *SuccessorUpdate-Message* to node 3 containing the information that node 4 is a better successor. The message is sent claiming node 5 to be the sender. It seems that node 4 has sent a message to node 5 and now node 5 informs node 3 about the new situation that the better successor is node 4.

   Node 3 will send a *SuccessorNotification-Message* to node 4 which does not exist and therefore node 4 will not answer this message with a *SuccessorUpdate-Message*. Because this response is missing, node 3 assumes that node 4 agrees with the choice and will change the successor pointer to node 4.

   At the end node 3 has a non-existing successor and node 5 a non-existing predecessor and both nodes have lost their correct successor/predecessor relationship.

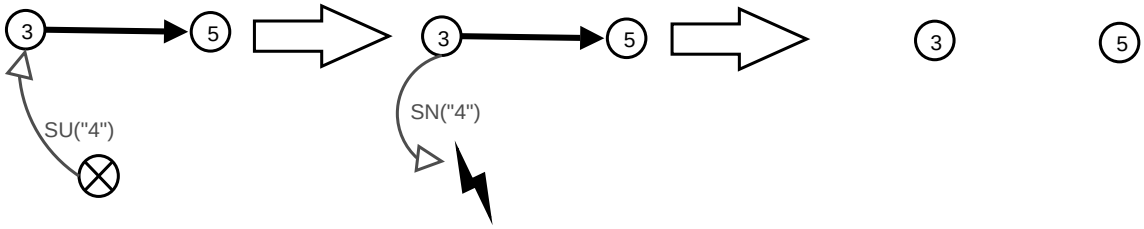


Figure 2.8: SuccessorUpdate-Message containing a fake sender

A second possibility is generating a fake update message in response to a *SuccessorNotification-Message*. Figure 2.9 shows this attack. Node 3 sends a *SuccessorNotification-Message* to node 5 which is the chosen successor. Node 5 agrees and does not answer with a *SuccessorUpdate-Message* but instead changes the predecessor pointer to node 3. Now the malicious node 6 generates a *SuccessorUpdate-Message* containing the sender address node 5 and the information about a non-existing node 4 being a better successor.

Node 3 will accept the message and send a *SuccessorNotification-Message* to the better successor node 4 and will certainly get no response. Node 3 consequently chooses node 4 as successor and changes the pointer.

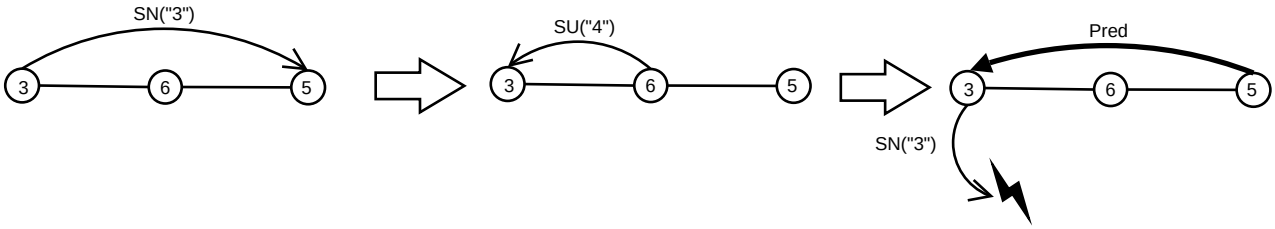


Figure 2.9: Fake SuccessorUpdate-Message

2. Deleting / Dropping the message

   Deleting a messages causes problems illustrated in figure 2.10. At first node 3 sends a *SuccessorNotification-Message* to node 5 because it identifies node 5 as the legitimate successor. Node 5 does not agree because it is aware of node 4 which would be the right choice. Therefore node 5 sends a *SuccessorUpdate-Message* back to node 3 containing the information about node 4. Node 6, the attacker, does not forward this message towards node 3 but drops it. As a result node 3 assumes that node 5 agrees with the successor choice.

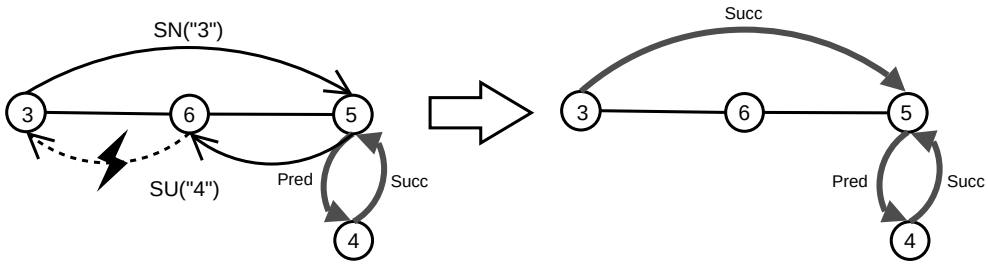


Figure 2.10: Deleting a SuccessorUpdate-Message

2.4.10 Flooding and Denial of Service

A denial of service attack (DoS) can be very harmful if the device is low on resources. Flooding is a simple way to keep a small device busy if it tries to handle each message. This study does not work on DoS attacks.

2.5 Countermeasures

This section introduces possible solutions for the described attacks and problems (2.4). In addition the advantages and disadvantages are discussed and compared with respect to the given protocol [Fuh04]. The choice of the concrete solutions takes place in chapter 3.

2.5.1 Node ID Assignment

According to [CDG+02] it is very important to ensure secure and randomized node ID assignment. Otherwise redundancy mechanism in peer-to-peer networks are useless. Furthermore the described Sybil attack (see 2.4.2) is always possible.

Each node that joins the network obtains a random node ID. In addition the network must ensure that the node IDs are uniformly distributed over the ID space. If all nodes stick to the rules they may choose their node IDs for themselves. The nodes would pick a node ID at random. In reality this will not work because there are always nodes which do not stick to the rules. Malicious nodes will pick special IDs to perform attacks and achieve their goals.

Simulating multiple identities which was introduced as the so-called Sybil attack (2.4.2) is possible if node ID assignment is not secure. [Dou02] shows that this attack is always possible without a central authority which is responsible for secure node ID assignment. Actual solutions against the Sybil attack will be discussed in section 2.5.2.

2.5.1.1 Certified Node IDs

[CDG+02] recommends building up a central certification authority. One task of this authority is the supervision of the randomized, uniformly distributed ID assignment. Furthermore the certification authority issues certificates for correct node IDs. Each node of the network would have the ability to verify a node ID of another node.

For this reason the public key of the certification authority must be installed in each node. If central nodes are part of this infrastructure that would constrain the self-organizing behavior of the network because fixed and central nodes would be essential.

An advantage of this solution is the fact that it simplifies building up of a public key infrastructure. The central authority is able to certificate the mapping between a node ID and the matching public key and each node in the network has the ability to check such an ID certificate.

In the introduced protocol setting (2.3) the user or the operator of the network respectively can act as a certification authority. When the network is built up or a new device is integrated, the operator installs the certificate on the device. This installation occurs once only and does not affect the self-organizing behavior of the network that much. Moreover the nodes only need to maintain one public key. Other people's malicious devices are not able to join the network. So the user is able to personalize his own devices. For example the power socket of user A will not communicate with the lamp of user B and deny access.

Alternatively the producer of the device can certify the node ID. This requires worldwide unique node IDs. Furthermore each node has to maintain the public keys of all producers of nodes in the network. Otherwise the node is not able to check all node IDs.

2.5.1.2 Constraints towards the Node IDs

The current protocol uses node IDs that represent bit strings from an ID space such as $[0 \dots 2^{m-1}]$. Each message is forwarded to its destination along these addresses.

One option is using the public key of the node as the node ID at the same time. The main advantage is that a mapping between node ID and public key is unnecessary and no central authority has to certify anything. The given protocol does not require a layer 3 protocol such as the Internet Protocol. For this reason a mapping between node ID and layer-3 addresses also becomes redundant.

Public keys are very long (order of 1024 bits) so it may be a good idea to use the hash value of the public key instead. Each node is able to check if the node ID is the hash of the public key. A further advantage of using a hash function is making attacks towards special ID spaces more difficult because choosing IDs selectively means finding an input for the hash function that is mapped to the desired node ID or generating public keys until the hash of one key matches the ID, which both is nearly impossible.

Nevertheless it is still possible to obtain more than one node ID. According to [Dou02] it is impossible to prevent the Sybil attack if there is no central authority which is responsible for secure node ID assignment. Even though it is possible to make this attack more difficult by defining constraints towards the node IDs.

For example hash values are only accepted as node IDs if they contain a fixed number n of zero bits or if the first n bits are on par with the last n bits. For example $n = 8$.

$$\underbrace{\underbrace{100110111}_{n-Bits}10100100....0011011\underbrace{100110111}_{n-Bits}}_{m-Bits}$$

The generation of a node ID would process the following algorithm.

1. LOOP
2. GEN_KEY(K_{pub}, K_{priv});
3. ID = $HASH_m(K_{pub})$;
4. WHILE ($first_n(ID) \neq last_n(ID)$);

The function $first_n(x)$ provides the first n bits and $last_n(x)$ the last n bits of the bit string x.

The generation of a correct node ID is 2^{n-1} times more difficult on average ($n \geq 0,\ n > \frac{m}{2}$) then. This requires the assumption that each node has got the same processing power. For a strong attacker with large processing power this is not a constraint.

Who defines the factor n? If n depends on a network state or local nodes there would a problem for any other node to know about the current value of n. Therefore it is the better alternative to define n as a global factor. So each node knows n and is able to check a node ID.

Another question is the order of n. If n is too small then it is easy to generate more than one ID and then simulate multiple identities. If n is too big then weak

devices have difficulties generating a correct node ID. The order of n depends on the application. It has to be defined application dependent.

Verification of a node ID is very simple because a node only has to check if $first_n(ID) = last_n(ID)$. The check of the mapping between the public key and the node ID is easy as well. Public key and node ID belong together if $Hash_m(K_{pub,\ ID}) = ID$.

It is important to point out that the described constraints do not prevent a Sybil attack since this is not possible without a central authority. The attack is more difficult but not impossible.

A disadvantage is that the node ID space is not used completely because there are now many node IDs that do not conform to the constraints.

2.5.1.3 Web of Trust

The web of trust is an alternative to a hierarchical public key infrastructure. Each user generates his own pair of keys. The mapping of the user and the corresponding key is signed by other users. So each user acts as certification authority. This approach is completely decentralized.

There can be many independent webs of trust which may be connected by users that are part of more than one web. The web of trust is very flexible and leaves the decision making in the hands of the users.

The problem with the web of trust in this case is the initial trust in another user. Normally users that sign other certificates know each other and trust each other. In the case of a random network this initial trust can not be assumed. If no node trusts another node a web of trust will not work since initial trust is required. A node will not sign a certificate if it does not trust the other node.

A very popular program that uses the web of trust concept is Open PGP (**P**retty **G**ood **P**rivacy, author Phil Zimmermann)[Zim05] which is mostly noted for email encryption and email signatures.

2.5.2 Countermeasures against the Sybil Attack

[CDG+02] proposes two countermeasures against the Sybil attack. One solution is to require an user to pay money for node ID certificates. The costs of a Sybil attack grows with the size of the network if the attacker wants to control a fixed percentage of the network.

This alternative requires a central authority which has to take control of the assignment and the settlement of the node IDs. The complexity of this infrastructure affects the self-organizing structure of the protocol.

The second option is to bind node IDs to real world identities. This is not possible in this case since the protocol is used in all-purpose and often the nodes do not correspond to a real world identity.

A constraint regarding the number of allowed neighbors can make the Sybil attack more difficult. If each node is only allowed to have n neighbors or if the devices have only n interfaces it might be possible for a node to detect that one node claims

to have more than n neighbors. That fact might indicate a Sybil attack. Perhaps the information is found in the path store. But this constraint would not allow networks like small world networks in which a few nodes maintain connections to many neighbors.

A challenge may indicate that a node tries to simulate multiple identities [Dou02]. The suspicious nodes have to solve a task that a single node is not able to solve. This requires the assumption that the resources of any two nodes differ by at most a constant factor. The challenge might be a crypto puzzle. But which entity does generate these challenges? There might be a central authority like a 'watchdog' which is responsible for spotting a Sybil attack. If there is no such entity all nodes may generate challenges. But how does a node move on if it suspects a node? Challenges may also lead to a complex infrastructure.

2.5.3 Countermeasures against the Wormhole Attack

Wormhole attacks are very difficult to defend against since in most cases the attacker uses a private out-of-band channel [KW03]. This channel is invisible to the network. The following proposals make the attack more difficult.

2.5.3.1 Packet Leashes

Packet Leashes are introduced in [HPJ03b] as a measure to detect and prevent wormhole attacks in wireless networks. A leash is an information which is added to the packet to prevent it from traveling more hops than intended. Therefore this information is called leash. Two versions of packet leashes are suggested: *geographical leashes* and *temporal leashes*.

Concerning the first version the node must be aware of its geographical position. Furthermore all nodes must have loosely synchronized clocks. The sender of a packet appends a timestamp t_s and its own position p_s. It might be a good idea to sign this information to prevent fake leashes. The clocks in the network are synchronized within Δ. ν is an upper bound for the velocity of any node, if the nodes change their position.

The receiver knows its own position p_r and the point in time t_r at which the packet is received. From these values and a position error δ the receiver derives the upper bound on the distance between the sender and itself:

$$d_{sr} \leq \| p_s - p_r \| + 2\nu(t_r - t_s + \Delta) + \delta \text{ ([HPJ03b])}.$$

A maximum allowed distance between sender and receiver has to be defined.

The second version of packet leashes requires tightly synchronized clocks. The error Δ between any two nodes is in order of a few microseconds or even a few hundred nanoseconds.

The sender adds a timestamp t_s and the receiver compares this value with the time at which it received the packet t_r. By means of the speed of light and the claimed transmission time the receiver is able to detect if the packet traveled too far.

Signing the timestamps would be a good idea for temporal leashes as well. The clocks must be tightly synchronized because small errors in time lead to big errors in measuring the distance. Therefore expensive hardware is required.

Packet leashes are introduced for wireless networks. The transfer to a physical link layer is not that trivial, because the length of a route does not correspond to the distance in hops in a network. Assuming equal transmitting power for any two nodes in a wireless network there is a maximum distance between two nodes that are able to communicate directly. In a wired network two nodes may have a distance of a few meters without being direct neighbors.

Other disadvantages are the necessity of synchronized clocks and the knowledge about the geographical position. This requires very expensive hardware. In most cases the nodes are not equipped with such hardware so the packet leashes are not practicable.

2.5.3.2 Signed Timestamps

A measure to detect wormhole attacks in wired networks might be to compute the round trip time. For this purpose node A generates and signs a *Hello-Message* including a timestamp T_{send}. This message is sent to each neighbor. A neighbor node B which receives this packet signs it with its signing key and then sends the packet back to node A.

Node A receives this 'Hello-Reply' and is now able to compare the receiving time with the timestamp T_{send}. The signature of A ensures that the packet was originally sent by A. Node A accepts node B as a direct neighbor if the time difference

$$T_{diff} = T_{receive} - T_{send}$$

is smaller than a bound t_{accept}.

Of course A must check the node ID of B and the signature of the packet. In addition A has to check whether it is the origin of the *Hello-Message* by checking the own signature.

At the same time node A has to prove that it is a direct neighbor of B by means of the same procedure. Both nodes prove to each other that they are direct neighbors and that there is a link between them.

Figure 2.11 shows a path-time diagram. In the left diagram B's response arrives in time and the verification of the node ID and the signatures is successful. In the right diagram B's response arrives too late ($T_{diff} > t_{accept}$) and A does not accept it.

Concerning this method the value t_{accept} must be analyzed and evaluated. It must be kept in mind that a node has to check and generate signatures within this time. At the beginning the node IDs and the public keys have to be checked.

In the design chapter flooding a node with *HelloReply-Message*s must be examined. This can lead to a denial of service attack. First of all the timestamp should be checked before verifying the signature. Packets with $T_{diff} > t_{accept}$ can be dropped without checking the signature. This order saves computation power and energy.

It should be mentioned that this method does not require tightly or loosely synchronized clocks such as packet leashes (2.5.3.1) because a node that generates a timestamp compares that timestamp later with its own clock.

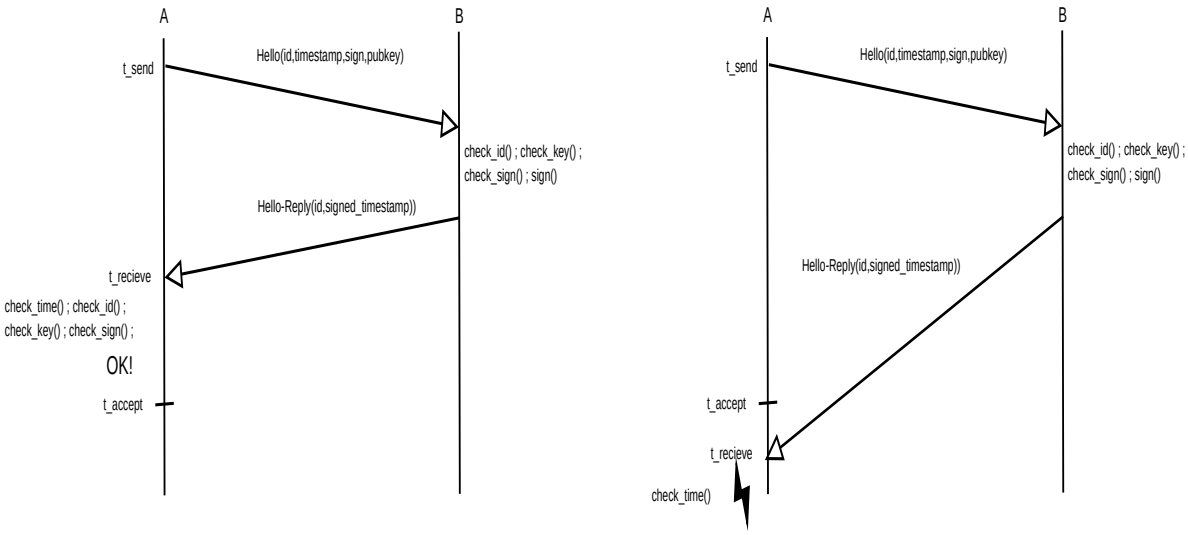


Figure 2.11: Neighbor discovery with signed timestamps

2.5.4 Countermeasures against the Sinkhole Attack

To prevent a sinkhole attack it is important to detect wrong routing messages. The malicious node must not be able to propagate that it is an attractive node for all routes.

Any node must be able to verify that a claimed route really exists. Furthermore a Sybil attack and a wormhole attack must be prevented.

If no node can claim that it is aware of non-existing routes and fake routing messages are detected, a sinkhole attack is hard to perform. The section 3.2 introduces link-certificates to prove the existence of a physical link between two nodes.

2.5.5 Countermeasures against Tampering with Packets

Preventing attacks that cut, modify or delete packets without any reason is very hard. Taking measures to detect the faulty packets is much easier. Integrity information can help to discover such attacks.

Each node maintains an asymmetric pair of keys that allows to sign messages before sending them. If an inner node modifies a packet the receiver of the packet will notice it because the verification of the signature fails. Thus the receiver takes notice of the attack and is able to ask for the packet again.

3. Design

This chapter introduces the concrete solutions for the problems and attacks that are described in chapter 2. After discussing advantages and disadvantages of the solutions the suitable ones are introduced. In addition the reasons for picking these solutions are mentioned.

3.1 Node ID Assignment

The node ID assignment is managed by a trusted third party. Therefore it is ensured that the IDs are chosen at random and uniformly distributed. As a result a selective choice of a node ID by an attacker is not possible.

This trusted third party also generates the public and private keys for all nodes. The mapping between the public key and the node ID is secured by means of an ID certificate. This certificate is installed on a node together with the public key of the trusted authority. With its certificate each node is able to prove that it maintains a correct ID and the matching public key.

To allow any node to verify all ID certificates it is important that each node possesses the public key of the trusted third party. The simplest way is operating with only one trusted party. In the protocol setting described in 2.3 this might be the operator of the network. Before a device joins the network for the first time, the ID certificate, the public key of the trusted party and the public/private key pair are installed. The nodes may be programmed by physical contact. Initializing is done only once. Except for the initialization the nodes are operating autonomously.

This procedure prevents a Sybil attack. The countermeasure described in 2.5.1.2, which is about constraints towards node IDs, makes the attack more difficult but does not prevent it. Furthermore the assumption that any two nodes have got nearly equal computation power might not be realistic. It is hard to find a good parameter n that satisfies a heterogeneous network. Some nodes might have problems to generate a correct ID whereas for other nodes it is not a constraint at all.

3.2 Link-Certificates

Important information in routing protocols are information about existing links and routes. To ensure a correct bootstrapping and routing phase it is fundamental to prevent routing messages that claim non-existent links between nodes. Routing information must not be propagated if it is not correct. For that reason a mechanism is required that allows nodes to verify a claimed route.

Two connected nodes must prove the existence of their physical link to any other node. For this reason a digital certificate is introduced. This certificate proves the existence of a link between two nodes. Since the links are bidirectional two certificates are needed to prove the existence.

Assume node A and node B to be neighbors. If A and B want to prove to node C that they are direct neighbors, node A has to certify the link between A and B. In addition node B has to certify the link between B and A. Two certificates are necessary to prove the existence of a bidirectional link.

A certificate that proves the link in one direction is called 'one-way certificate'. To prove the existence of a complete link two one-way certificates are necessary. Two one-way certificates together are called 'link-certificate'.

Before issuing a one-way certificate both nodes have to convince each other that they are direct neighbors. Afterwards both nodes sign their one-way certificate with their private signing key. A physical link is considered as existent if both one-way certificates can be verified. After proving their neighborhood to each other both nodes send their one-way certificate to the neighbor.

$$A \longrightarrow B : Cert(A \rightarrow B)_{signed(A)}$$
$$B \longrightarrow A : Cert(B \rightarrow A)_{signed(B)}$$

3.2.1 Certificate Format

The link-certificate includes two one-way certificates which are signed by different nodes. The format of the two one-way certificates is identical. In the following the parts of the certificate are introduced. In addition the purpose of each field is explained.

The size of the certificate should be as small as possible to save bandwidth and memory in the nodes.

3.2.1.1 Unique Identifier

A unique identifier is not absolutely necessary since the certificate can be identified by means of the timestamp and the two unique node IDs. Nevertheless a unique ID might be useful to recognize a certificate quickly or to speed up the access to it. Duplicated certificates are recognized faster.

The identifier might be a random number or the two IDs of the connected nodes added by a random number. The implementation and evaluation shows if an identifier is really needed. Maybe the described node IDs and the timestamp are sufficient. Without an additional identifier the certificate is smaller. This would save bandwidth.

3.2.1.2 Node ID

The certificate must include the two IDs of the connected nodes to identify the link. To save an additional link ID it is reasonable to stick to the correct order of the node IDs.

1. ID 1: start node (certifies)
2. ID 2: end node

This order means that node 1 certifies the existence of a link from node 1 to node 2. The certificate is signed with the signing key of node 1. Any other node needs the public key of node 1 to verify this certificate.

3.2.1.3 Link ID

An additional link ID is not necessary if the certificate format sticks to the introduced order of the node IDs (3.2.1.2). Then a link is identified by a start node and an end node. Each node is able to identify the link and the direction of the certificate by means of the start and the end node.

As a result the node ID can be omitted to save bandwidth and memory.

3.2.1.4 Timestamp

A timestamp is useful for several reasons. First of all it identifies a certificate together with the start and the end node. Additionally it provides protection against replay attacks. If an attacker records a package that contains a correct validated certificate and replays it later the certified link might be down already.

The certificates have limited validity. As a result the replayed package would not be accepted because the timestamp is out-of-date. That is why a timestamp provides good protection against replay attacks if the clocks of all nodes are loosely synchronized.

An important value is the valid time of a certificate. This is discussed later in section 3.2.2.

3.2.1.5 Public Key

The node that verifies a one-way certificate needs the public key to check the signature. An ID certificate which includes the public key contains much data with regard to the size of a message. The size of a protocol message is limited by the maximum size of the underlying layer 2 message. There are no protocol messages to fragment and reassemble messages.

For that purpose the ID certificate is not part of the one-way certificate. Protocol messages to exchange the ID certificates will be introduced in section 3.3. These messages lead to extra traffic but ID certificates have a very long validity period and the public keys can be maintained in a cache structure to speed up the access. In addition fragmentation means overhead and complexity too.

3.2.2 Validity of a Certificate

The certificates can be realized soft-state or hard-state. Soft-state means that a certificate expires and must be generated and exchanged again. Hard-state means that it does not expire or has got a very long valid time. This requires a revocation of certificates since it is possible that links break and then the certificates would certify invalid links.

In the protocol setting 2.3 a revocation of link-certificates is not possible since there is no central infrastructure. Central nodes which store the revocation lists would be necessary to provide the access to these lists to all nodes. If the topology of the network changes quickly the revocation lists are often out-of-date.

Link-certificates with limited validity make more sense. A certificate is valid if both one-way certificates are valid. A one-way certificate expires if the valid time t_Δ added to the current time is older then the timestamp of the certificate,

$$(t_{certificate} + t_\Delta) < t_{now}.$$

An important design parameter is the value t_Δ. It must be kept in mind that the clocks in the network are only loosely synchronized. If t_Δ is too small then the certificates expire very often. This means that links are not accepted by other nodes and that certificates have to be renewed which increases the costs for each node and costs bandwidth.

If t_Δ is too long replay attacks are simplified. Furthermore there might be certificates that ensure links that are down already. The dimension of t_{valid} and its effect will be examined in chapter 5.

3.2.3 Example of a One-Way Certificate

As described above a complete link-certificate includes two one-way certificates. Each one-way certificate ensures the existence of the link in one direction.

The signature is computed over the bold fields.

unique ID
start node ID
end node ID
timestamp
signature

3.2.4 Verification of a Certificate

The verification of a certificate sticks to an exact algorithm. It makes sense to check the expiration first because if the certificate is out-of-date it can be dropped. Verifying the signature is needless if the certificate has expired. This correct order saves computing time.

Afterwards the public key of the start node must be looked up in the public key cache. If the look-up fails the ID certificate of the start node is requested from the node by means of a *GetIdCertificate-Message* (see 3.3.4).

Finally the signature of the one-way certificate that ensures the existence of the link in one direction is verified. The public key of the start node can be taken from the ID certificate or from the data structure that contains the known nodes.

3.2.5 Management of ID Certificates

ID certificates have a very long validity period. It is not necessary to apply the certificates to each one-way certificate since this would lead to very big messages and network traffic. An exception is the ID certificate of the sender. This might be applied to the message to allow the receiver the verification of the message without demanding the ID certificate with an extra message.

If a node is not aware of a public key it sends a *GetIdCertificate-Message* (see 3.3.4) and the receiver replies with a *SendIdCertificate-Message* (see 3.3.5).

After verifying a certificate the public key together with the node ID is stored in a public key cache data structure. This cache speeds up the access to the public keys. The ID certificate does not have to be verified before checking a one-way certificate or a message signature.

Since the cache is not big enough to store all obtained public keys and node IDs a replacement strategy is needed. First of all the direct neighbors have a special status as they are protected from being replaced. Furthermore all public keys of node IDs that are part of the source route to the successor are protected too.

All other public keys are replaced by means of a last recently used (LRU) strategy.

The size of the cache data structure is discussed in chapter 5.

3.3 Protocol Messages

This section describes the modifications of the protocol messages and introduces new messages.

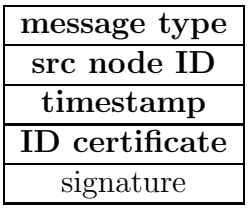
3.3.1 The Hello-Message

The current *Hello-Message* is enhanced by three new fields to come up to the new requirements. The message must contain the ID certificate of the sender to allow the neighbor node to check the node ID and to obtain the public key.

The next field is a timestamp. By means of the timestamp the round trip time is computed. This procedure was introduced in section 2.5.3.2. Measuring the round trip time makes tunneling *Hello-Message*s and replay attacks more difficult.

Finally the certificate now contains a signature of the complete message. This signature allows the receiver to determine the sender and to detect fake or modified messages.

The enhanced *Hello-Message* might look like this (signature is computed over the bold fields):



3.3.2 The HelloReply-Message

This message is not part of the current protocol. The message is a reply and acknowledge to the *Hello-Message* and its second purpose is to estimate the round trip time. Assume that node A sends a *Hello-Message* to node B containing the timestamp $t_{send-hello}$. Node B receives this message and responds with a *HelloReply-Message*. This reply is signed by node B.

If A receives the acknowledgment it clocks the timestamp $t_{received-reply}$. The round trip time is computed as

$$t_{RTT} = t_{received-reply} - t_{send-hello}.$$

To make the wormhole attack (see 2.4.3) that tunnels *Hello-Message*s more difficult node A accepts node B as a neighbor only if the round trip time is smaller than a threshold. If the reply message arrives too late the round trip time is bigger than the threshold. As a result node A assumes that the *Hello-Message* traveled more than one hop and node B is not a real neighbor.

If node A has verified the signature of B it accepts node B as direct neighbor and the existence of the link between them.

The *HelloReply-Message* must contain a proof that node B has received a *Hello-Message* from A. This could be the complete *Hello-Message* or the signed hash of the message.

The complete message is much bigger than the hash and this increases the bandwidth costs. But if a hash is used, node A has to store the hash and the timestamp to recognize the message and to compute the round trip time. The complete message contains the timestamp and a signature of A and A can easily check the own signature.

It is not absolutely necessary to add the ID certificate to the reply message because node B sends a *Hello-Message* to node A, too. On the other hand if the message gets lost, node A is not able to verify the signature and this might cause a delay. Therefore adding the ID certificate is worthwhile.

message type
src node ID
dst node ID
timestamp
ID certificate
hello hash
signature

3.3.3 The SendLinkCertificate-Message

After receiving a *HelloReply-Message* and verifying that the sender is a direct neighbor, the node generates the one-way certificate. Afterwards a *SendLinkCertificate-Message* including this certificate is sent to the neighbor. The neighbor is now able

to prove the existence of the direct link towards other nodes since it obtained both one-way certificates.

Additionally the message contains a timestamp to prevent replay attacks and the signature of the sender.

message type
src node ID
dst node ID
timestamp
one-way certificate
signature

3.3.4 The GetIdCertificate-Message

Since an ID certificate has a very long validity period it does not make sense to apply it to every one-way certificate. The certificate is verified once and then the public key is maintained in a cache (see 3.2.5).

If a node is not aware of the public key, it generates a *GetIdCertificate-Message* to obtain the missing key. Since the source route towards the node can not be verified until the public key is known, this message must be sent over an untrusted path.

Many *GetIdCertificate-Message* are generated in the bootstrapping phase since many paths are exchanged and signed and the nodes are not aware of all the public keys.

To send a *GetIdCertificate-Message* the sender node copies the path from the received message and inverts it. Clearly this is an untrusted path. A destination node which gets such a request replies with a *SendIdCertificate-Message* (3.3.5).

The sender applies its own ID certificate to the get-message to speed up the propagation of the public keys. Each inner node that forwards the message can learn from the message and can insert the public key into the public key cache after verifying the certificate.

Normally there is more than one ID certificate missing since missing certificates are the result of path-inserting operations. To speed up the process of obtaining certificates, more than one certificate can be requested with one *GetIdCertificate-Message*. The message is sent to the last node of the source route that could not be inserted and the nodes in the path for which the ID certificates are missing are marked.

The signature of the *GetIdCertificate-Message* is very important. No node should answer a request without verifying the signature. Otherwise a *GetIdCertificate-Message* could be used for a denial of service attack. The attacker may send a lot of request messages containing the victim's address as sender. Many nodes will fulfill the request and then the victim has a lot of work with verifying, inserting and processing the reply-messages.

It is also important that this messages are not enhanced by inner nodes since inserting and changing the path will lead to more *GetIdCertificate-Message* and if one of this messages causes n other this will result in a huge number of messages. Any inner node behaves the following way:

1. Check the message's signature by using the ID certificate from the message
2. Check the request field and answer the requests with *SendIdCertificate-Message*s
3. Mark answered requests as fulfilled
4. Forward the original message if there is still an open request

There is a high probability that an inner node is aware of one or more requested certificates. As a result a *GetIdCertificate-Message* will normally not travel the complete path since there are inner nodes of the path that can fulfill the requests. Together with the accumulative request this decreases the number of messages.

message type
src node ID
dst node ID
timestamp
ID certificate
length
source route
request field
flag field
signature

3.3.5 The SendIdCertificate-Message

The *SendIdCertificate-Message* is the response to a *GetIdCertificate-Message*. The receiver of the get message inverts the source route and applies it to the generated *SendIdCertificate-Message*. The payload contains the own ID certificate.

Each inner node forwards the message towards the destination and in addition inserts the public key into its key cache after verifying the certificate.

To decrease the number of messages more than one requested certificate could be allowed per *SendIdCertificate-Message*. On the other hand the message size will increase since an ID certificate is not that small. The number of certificates per message will be examined in chapter 5.

message type
src node ID
dst node ID
timestamp
ID certificate
length
source route
ID certificate 1
ID certificate ...
ID certificate n
signature

3.3.6 The SuccessorNotification-Message

The *SuccessorNotification-Message* is enhanced by the link-certificates to prevent propagating fake routes. For this purpose the sender adds all relevant link-certificates. This means that for each hop between sender and receiver there has to be a valid link-certificate.

The first message is sent to a direct neighbor. The required link-certificate is made up of the two exchanged one-way certificates. Step by step the nodes obtain the required link-certificates from *SuccessorUpdate-Message*s for instance.

The message contains a timestamp to prevent replay attacks. Since the message content can change the signature is computed over the constant field such as source and destination node and timestamp.

message type
src node ID
dst node ID
timestamp
ID certificate
length
source route
link-certificate 1
...
link-certificate n
signature

It is very important to maintain the connection between the predecessor and the successor node. The link-certificates of this connection have to be up-to-date.

For this purpose the *SuccessorNotification-Message* is sent periodically. The successor must reply with a *SuccessorNotificationAck-Message* to keep the connection between the nodes alive. The period time is a subject of the evaluation in chapter 5.

It is possible that the route to the successor changes because nodes are down or new nodes join the network. Any inner node that forwards one of these messages looks up its path store for a better route and substitute the source route and the paths. Furthermore any inner node refreshes the own one-way certificate in the message to update the link-certificates.

If the successor is unreachable or does not respond after m messages with an acknowledgment message the node loses its successor. If this happens a *SuccessorNotification-Message* is sent to the node which is the next best successor.

3.3.7 The SuccessorNotificationAck-Message

The *SuccessorNotificationAck-Message* is a response to the *SuccessorNotification-Message* to make sure that the potential successor accepts the choice. Furthermore it is a reply to the periodic notification to maintain the connection between predecessor and successor as a kind of keep-alive message.

The acknowledge message is important since a lost or dropped *SuccessorUpdate-Message* leads to an inconsistent state. In the current protocol the sender of a *SuccessorNotification-Message* will assume that the receiver has accepted its choice if an update message gets lost.

The node does not change its successor until it receives an acknowledge message. If the acknowledge message gets lost the notification message is repeated.

The node checks the signature everytime a notification is received. Furthermore it checks if the source route towards the predecessor has changed. If so, the link-certificates are checked and the source route is updated.

Since the link-certificates of the predecessor route have to be up-to-date, old certificates are refreshed. The notification contains the newest certificates.

Any inner node forwards the message and refreshes the own one-way certificates. If the source route does not exist any longer the route is adapted and the required link-certificates are substituted. For that reason the sender of a *SuccessorNotificationAck-Message* only signs the constant fields of the message.

message type
src node ID
dst node ID
timestamp
ID certificate
length
source route
link-certificate 1
...
link-certificate n
signature

3.3.8 The SuccessorUpdate-Message

The sender of a *SuccessorUpdate-Message* has to apply all relevant link-certificates to the message because the inner nodes learn paths from the update messages and have to check the link-certificates before inserting the routes into the path store. The signature can be computed over the constant fields only such as source and destination node and timestamp.

The source route from the sender to the receiver of a *SuccessorUpdate-Message* changes with high probability because inner nodes of the path try to find a better route and modify the source route if they find one. Any inner node that substitutes the source route has to substitute the link-certificates too since the receiver has to check the complete source route.

Furthermore the *SuccessorUpdate-Message* contains the update route to inform the receiver about the better successor. All link-certificates that certify this update path have to be appended too.

message type
src node ID
dst node ID
timestamp
ID certificate
length
source route
update route
link-certificate 1
...
link-certificate n
link-certificate (update route) 1
...
link-certificate (update route) n
signature

3.3.9 The ConnectionRequest-Message

If a node wants to communicate with another node it tries to send a *ConnectionRequest-Message* to this node. An inner node just forwards this message while a node, that is neither an inner node nor the destination node, tries to append a path towards the destination.

The sender signs the hash over the constant fields such as source node, destination node and timestamp.

3.3.10 The Established-Message

The *Established-Message* is sent back to the sender of a *ConnectionRequest-Message*. The current protocol recommends to treat an *Established-Message* like a *SuccessorUpdate-Message*. This means an inner node learns from the path and tries to find a better one.

This learning would require messages with link-certificates of the complete route. The evaluation has to show whether this is too much work for the inner node. If so, the inner node may just forward the messages and the link-certificates are not necessary.

3.3.11 The Unreachable-Message

The *Unreachable-Message* is sent back to the sender of a *ConnectionRequest-Message* if the destination node is unreachable. To prevent fake messages a hash is computed over the message and signed by the node that is not able to forward the *ConnectionRequest-Message* because the route is broken.

message type
src node ID
dst node ID
unreachable ID
timestamp
ID certificate
length
source route
signature

3.4 Management of Neighbors

It is of great benefit to maintain a data structure that contains information about the direct neighbors such as node IDs, public keys and link-certificates.

This data structure simplifies access to the public keys. The *Hello-* and the *HelloReply-Message*s are sent periodically and it is not necessary to verify the ID certificate each time if the node ID and the public key can be found in the data structure.

A clean-up procedure that deletes old entries helps to keep the neighbor information up-to-date. This means that the entries are soft-state and there is a need to refresh the information. The soft-state concept ensures always the recent neighbor information.

After receiving a *Hello-Message* the entry is refreshed. If the entry was deleted meanwhile the ID certificate has to be verified again.

4. Implementation

This chapter describes the implementation of the new secure protocol.

4.1 The OMNet++ Simulator

To simulate the protocol behavior **OMNet++**[Var05] is used. **OMNet++** is a discrete event simulation environment and provides a component architecture for models. These components are programmed in C++ which are then put together to larger components. **OMNet++** models use NED (NEtwork Description) as a high-level language to describe the topology of the components.

4.2 The Network Structure

The simulated network has one Watchdog module and several Node modules. Node modules can be nodes, secure nodes and several malicious nodes. All modules are declared in the NED language in the file mynetwork.ned. In this file the parameters of these modules are declared. **OMNet++** modules declare parameters which can be changed in the omnetpp.ini file. These parameters are parsed in the C++ code. This allows the user to simulate several so called *runs* with different parameters without recompiling the source code between the runs.

4.2.1 The Network Topology

The watchdog module is responsible for the generation of a network. In this network each new node is connected to n existing nodes at random. This network topology is called an *Erdős-Rényi-Graph*. First of all an initial network of size *initialNodes* is created. Each node will request a correct node address from the watchdog and the watchdog will generate a pair of keys (public/private) and set up the ID certificate for each node. After that the nodes join the network by sending a *JoinMsg* to the watchdog. When a new node joins the network the watchdog chooses a random number (*initialConnections*) of nodes and connects the new node to these random nodes. The new node and the random new neighbors are informed about the new existing link with a *LinkUpMsg*. This message is not part of the protocol but rather

simulates a layer 2 message about a new physical link. If the watchdog kills a node, the neighbors of the leaving node are informed with a *LinkDownMsg* about the fact that the physical link is down.

4.2.1.1 Problems with this Topology

In this network topology there are a few nodes that have many neighbors and connections. In the enhanced protocol these nodes can be a bottleneck since many messages pass these nodes and they have a lot of signing and verification work to do. As a result the passing messages are delayed. To come along with this problem nodes with many neighbors must have more computational power. In a real network this will be the situation but it is not part of this implementation.

4.2.2 The CPU Concept

To keep the simulation of the secure protocol realistic it is very important to consider the costs of signing and verifying messages. Every sign and verify operation costs CPU time. To simulate this fact in **OMNet++** each secure node holds a CPU variable. If a message or certificate is signed or verified the computation time is added to the CPU variable. Then the next message that is sent to another node is sent with delay. This delay is computed in the following way:

$$delay = Time_{current} - Time_{CPUfree} + link\ \ delay$$

4.2.3 The Modules

The following subsection describes the several modules that are used within the simulation and their parameters.

4.2.3.1 The Watchdog Module

The watchdog is the master and observer of the network and implements several managerial functions. With respect to the real world scenario the watchdog can be seen as the user or administrator of the network. One task is to generate a pair of keys for each node address and to certify this mapping within an ID certificate. These ID certificates are distributed to each node once at the time the node joins the network. All other tasks do not correspond to a real world activity. The watchdog creates and kills nodes and records statistical data. The Watchdog module provides the following parameters.

maxNodes If nodes are created dynamically in a simulation this is the upper bound of nodes in the network (*maxNodes* > *initialNodes* in a simulation run).

minNodes If nodes are killed dynamically in a simulation this is the lower bound of nodes in the network.

nodeType The node type of the normal nodes in the network. This parameter is 1 for a standard node or 2 for a secure node.

sinkholeNodes This value sets the number of malicious sinkhole nodes in the initial network with the current protocol.

secureSinkholeNodes This value sets the number of malicious sinkhole nodes in the initial network with the enhanced protocol.

maxAddress The addresses that the watchdog generates lie in the range [1..*maxAddress*).

createIntervall The time between the creation of two new nodes measured in seconds. 0 means that no new nodes are created within a simulation run.

initialNodes The number of initial nodes/secure nodes in the network.

startSimulation The time the first node is created and the simulation begins.

killIntervall The time between the 'death' of two different nodes.

measureIntervall The frequency of doing statistics. Every *measureIntervall* the watchdog gets a message to record the statistical data.

initialConnections The number of links for each node. When a node joins the network, the watchdog tries to connect it to a number(*initialConnections*) of nodes.

4.2.3.2 The Node Module

The Node module implements the behavior of a node in the standard protocol. The code can be found in the files node.h and node.cc.

pathstoreSize The maximum size of the node's path store. The path store provides a clean-up function. Periodically the least-recently used nodes are deleted from the path store if the current size of the path store is bigger than *pathstoreSize*.

connectionIntervall The frequency of connection requests per node. If a node sends a connection requests it waits for that time until the next request is sent. If a node should send a request with an interval of n seconds, it makes sense to let **OMNet++** compute a value in the range $[n-\frac{n}{2}, n+\frac{n}{2}]$. Otherwise each node in the network will send a connection request at the same time and then the complete network does not send any request until the next interval.

cleanUpInterval With this frequency the clean-up function is called and the path store is cut down to the maximum size again.

linkDelay The link delay to a neighbor in seconds.

helloTimer A *Hello-Message* is sent periodically with this frequency.

4.2.3.3 The SecureNode Module

The SecureNode module implements a node that sticks to the enhanced protocol. It is derived from the Node module but many functions have changed since no message is handled in the same way. Parameters that are taken from the Node module are not listed again.

pathstoreSize The same parameter as in the Node module. In the clean-up function all nodes with expired link-certificates are removed too.

maxMsgId The maximum message identifier.

successorNotificationTimeout When a node sends a *SuccessorNotification-Message* to another node and does not get an acknowledgment, it waits for this time until the message is retransmitted.

successorNotificationTimeoutRetries When a *SuccessorNotification-Message* timed out the message is retransmitted and the retries counter is decremented. Furthermore the timeout period is increased. If retries is 0 the successor is deleted from the path store and the next better successor is informed.

successorKeepAliveTimer If a node receives a *SuccessorNotificationAck-Message* the timeout is canceled and the keep-alive timeout is started. After this time the node sends a *SuccessorNotification-Message* again.

timestampAcceptance A one-way certificate expires if

$$t_{current} > timestamp_{Certificate} + timestampAcceptance.$$

signTime The CPU time that is necessary for a sign operation.

verifyTime The CPU time that is necessary for a verify operation.

idCertificateStoreLimit The maximum size of the ID certificate store.

4.2.3.4 The SinkholeNode Module

The SinkholeNode module is derived from the Node module and can be found in the files sinkholenode.h/cc. It implements a malicious node that performs the attack described in 5.2. Since the module does not send any *Connect-Message*s and does not maintain a path store, the only parameter is *linkDelay* described in 4.2.3.2.

4.2.3.5 The SecureSinkholeNode Module

The SecureSinkholeNode module is derived from the Node module and can be found in the files securesinkholenode.h/cc. It implements a malicious node that performs the attack which is described in 5.3. All parameters are the same as for the secure Node module.

4.3 The Classes and Files

4.3.1 Certificates

The code for the certificates is declared in the file certificate.h and implemented in certificate.cc. A certificate can be an ID certificate or a one-way certificate. So it makes sense to use a base class *Certificate* that provides all basic methods like *sign()*, *verify()* and timestamps. Since this is only a simulation, signatures are not computed with a real signing algorithm. A signature in this simulation is just a boolean flag that is set to *true* if the signature is correct.

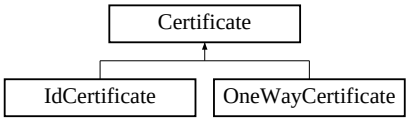


Figure 4.1: The class diagram for the certificate classes

4.3.2 Messages

All protocol messages are declared in the file message.h and implemented in message.cc. In addition to these files there is a file message_id.h where unique integers are defined for all messages. These integers are used for the 'message kind' field in **OMNet++**. Messages are handled by means of the 'message kind'.

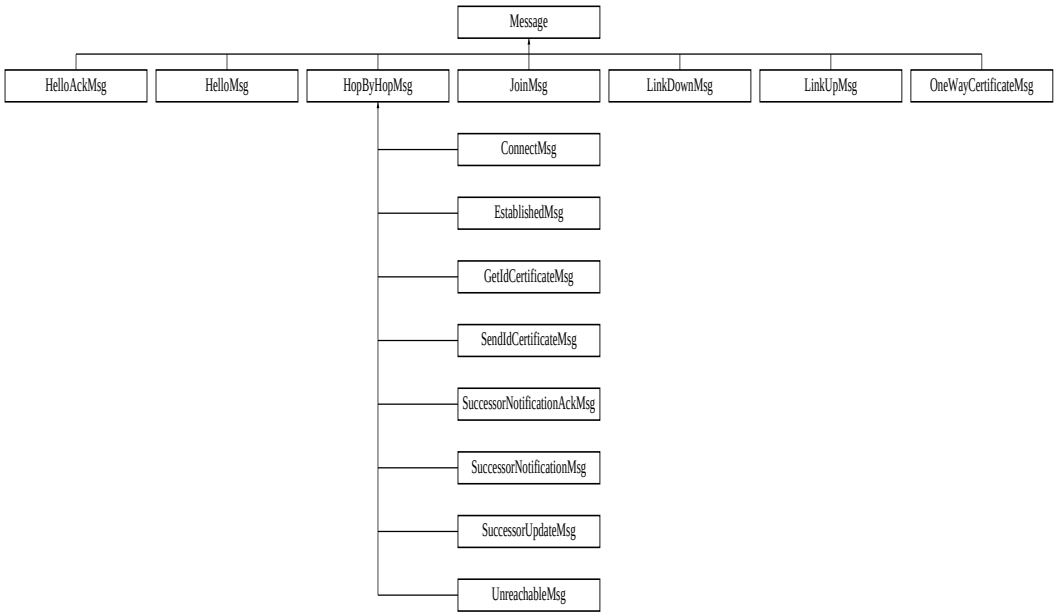


Figure 4.2: The class diagram for the message classes

4.3.3 Paths and Path Store

The path data structure and methods to handle and manipulate paths are defined in path.h/cc. A path is an array of address elements together with an array of forward certificates and an array of backward certificates. The maximum length of the array is fixed to the `#define MAXPATHLENGTH`.

The path store structure is implemented in the files pathstore.h/cc. The path store is organized as a tree. The root address is the address of the node that maintains the path store. Each tree node has a parent pointer, a parent certificate, a list of children and a list of child certificates. In addition to the tree all nodes in the path store are stored in a double-linked list which is sorted with respect to the nodes' addresses. To load a path to a node X from the path store, the destination node X is looked up in the tree and then the demanded path is the route from X upwards to the root node.

4.3.4 Modules

The different modules are implemented in the files node.h/cc, securenode.h/cc, sinkholenode.h/cc, securesinkholenode.h/cc and watchdog.h/cc.

4.4 Statistics

4.4.1 Vector and Scalar Files

OMNet++ stores the statistical data in vector files and scalar files. A vector file is a text file that contains (vector number,simtime,value) tuples. A scalar file is recorded at the end of each run and contains values like *Total Number Of Messages*. The Watchdog module is responsible for recording the statistical data. Vector data is collected periodically and activated by a self-message. The scalar data is collected when the Watchdog module finishes in the method *finish()*.

4.4.2 Scripts

4.4.2.1 Start Scripts

The source code contains some bash scripts that generate **OMNet++** ini files for different parameters. These ini files contain the parameters for several runs. Each run creates a scalar file 'scalar_parameter_parametervalue_run_X' and a vector file 'vec_parameter_parametervalue_run_X'. After executing all runs, the vector and scalar files together with the ini file are zipped.

4.4.2.2 Perl Scripts

There are two perl scripts to analyze the vector and scalar files. The perl script *derive_mean_vec.pl* needs the two arguments *parameter* and *parametervalue*. It derives a file called 'mean_parameter_parametervalue' which is a vector file containing the average vectors. It is important that all vector files contain the same number of vectors since this simple script is not fault-tolerant.

The perl script *derive_mean_scalar.pl* with the same arguments does the same for the scalar files. Again it is important that all scalar files contain the same number of scalars.

The script *buildgnu.pl* needs the arguments *parameter* and *search string*. It searches for files like 'mean_scalar_parameter_*' and extracts the *search string* scalar into a text file. Furthermore it creates a *gnuplot* [TW04] input file and a postscript file containing a bar plot for the different parameter values. The user is asked if he wants to delete the temporary .gnu and .dat file. Sometimes a *gnuplot* file needs some modifications and the user does not want to delete it directly. The *boxfill.pl* [Wid00] script needs an input and an output postscript file and paints the bar plots which are created by *gnuplot*. *buildgnu.pl* calls *boxfill.pl* automatically.

4.4.3 Creating Figures

All plots are created with *gnuplot*, a command-line driven interactive data and function plotting utility [TW04]. Mean vector files are first split with the **OMNet++** program *splitvec* and then several files are plotted with gnuplot. The bar plots for the scalar data are plotted with *gnuplot* too and painted with the perl script *boxfill.pl* [Wid00].

5. Evaluation

The evaluation chapter is split up in three parts. The first part describes the attempts to find good parameters for the enhanced protocol and tries to figure out the costs of security. The second part shows an attack against the old protocol and the the last part examines the resistance of the new protocol against this attack.

5.1 Finding Good Parameters

The enhanced protocol has many parameters which influence each other. As a result the search space for the right parameter is very large and can not be completely searched in this study. Furthermore each run in **OMNet++** starts with a run-specific seed value for the random number generator. This means that the network topology of run 1 is completely different from the topology of run 2. It can happen that a parameter value shows great results for run 1 but bad results for run 2. To get a balanced result to some degree each tuple (parameter, value) is tested for n runs and then a perl-script derives a mean result. Because of the time constraints of this study, we have chosen $n = 5$. It is totally clear that with increasing n the results are more reliable. The time constraints limit the network size too, because running n simulations for each choice of parameters is not that fast. Most of the simulations run with 256 nodes. This value allows to test more parameters. The table shows the fixed parameters for this section.

Parameter	Value
Nodes	256
Sign Time	100ms
Verify Time	10ms
Link Delay	1ms

The evaluation tries to find good parameters sequentially. For example the first parameter is the successor notification timeout. The best value is chosen and then this value is fix for the next parameter test. This will apparently not find the global optimum but it may find good values. For each subsection a table illustrates the fixed values in addition to the table above and the range of values for the test parameter.

5.1.1 Successor Notification Timeout

This timeout is started when a *SuccessorNotification-Message* is sent. The possible successor must reply with a *SuccessorNotificationAck-Message* within this time. After a node receives an acknowledge from its successor, the timeout is replaced by a keep-alive timer to keep the virtual connection between successor and predecessor alive and to ensure that the source route's one-way certificates are always up-to-date.

It is not that easy to set the timeout parameter correct since it depends on the number of nodes, the mean path length, the message delay that is caused by signing and verifying and so on.

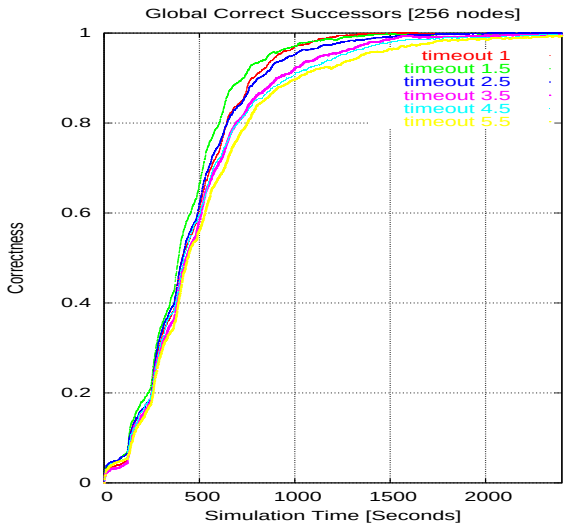


Figure 5.1: Successor notification timeout: global correctness

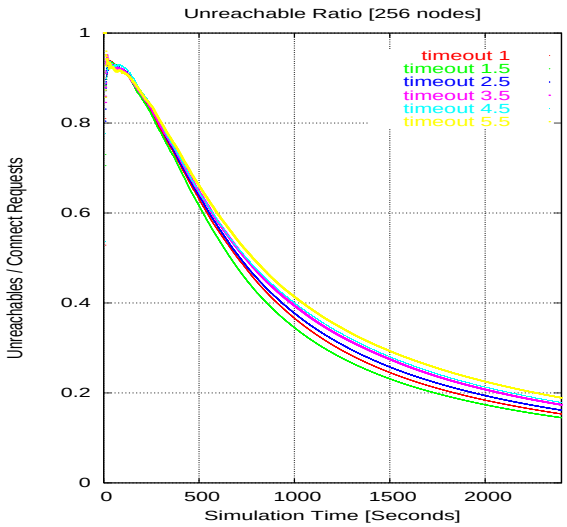


Figure 5.2: Successor notification timeout: unreachable ratio

Figure 5.1 and 5.2 show the successor notification for the range [1..5.5] seconds in steps of $0.5s$. The left graph shows the global correctness, in other words the ratio

$$\frac{\#GlobalCorrectSuccessors}{\#Nodes}.$$

The right figure shows the unreachable ratio. For each connection request a global *ConnectRequest-Counter* is increased. For each *Unreachable-Message* and for each connection that failed from the beginning an *Unreachable-Counter* is increased. The ratio

$$\frac{Unreachable-Counter}{ConnectRequest-Counter}$$

is the unreachable ratio. The unreachable ratio will not reach the value 0 since the complete time is considered and not a value for the last n seconds. So the *Unreachable-Messages* from the beginning of the simulation affect the ratio until the end. We expect the ratio to approach 0 asymptotically.

For the network with the given parameters the best successor notification timeout lies between 1.5 and 2 seconds where $t_{Timeout} = 2s$ shows better results with regard to the correctness and $t_{Timeout} = 1.5$ has a better unreachable performance.

When the network gets up the different timeouts show nearly equal results. The several graphs do not differ at first. The reason may be that very few nodes find their correct successor in the beginning. During a long timeout period the assumed incorrect successor is replaced by a better node and the timeout is canceled. Later in the bootstrapping phase when each node has learned more about the network, the different timeouts matter. A very long timeout causes a long waiting time if the successor is correct but was not able to verify the path and dropped the message. The predecessor will wait for the complete timeout since there is no better successor and therefore no *SuccessorUpdate-Message* that causes the abort of the timeout.

If a timeout is too short a node may have problems to send a *SuccessorNotification-Message* during this short timeout period. In the beginning of the bootstrapping phase each node executes many signing and verification operations. These operations increase the message delay and a reply message may arrive too late.

Parameter	Value
Path Store Size	50
ID Certificate Store Size	50
Path Store CleanUp Interval	40
Timestamp Acceptance	1000
Hello Timer	500
Certificates per SendIdMsg	2
Successor Notification Timeout	[1..5.5]
Successor Notification Timeout Retries	10
Successor Keep-alive Timer	120
Connection Interval	uniform(10,30)

5.1.2 Successor Keep-alive Timer

After a node has established the virtual connection to its successor, in other words a *SuccessorNotification-Message* has been replied with a *SuccessorNotificationAck-Message*, the successor notification timer is canceled and the keep-alive timer replaces the previous timeout. The keep-alive timer should be longer than the successor notification timeout to prevent message overhead. The idea is to keep the path to the successor and above all the link-certificates of this path up-to-date. If the timeout is too long the certificates may expire before the path is updated. If it is too short, there will be many messages sent that cost signing and verifying operations and cause message delay. Furthermore more messages make the nodes spend more energy. The table illustrates the used parameter values. For the successor notification timeout the best value $t_{Timeout} = 1.5s$ from the previous test is taken.

Figure 5.3 and 5.4 show that the keep-alive timer with $t_{Keepalive} = 20s$ has the best results. Another important point is the number of total messages. A short timeout leads to more messages. On the one hand there are more messages to learn paths from, on the other hand, more messages mean more signing and verifying operations and more energy consumption. Figure 5.5 shows the mean number of total messages for each test value. The better results in the unreachable ratio costs more messages.

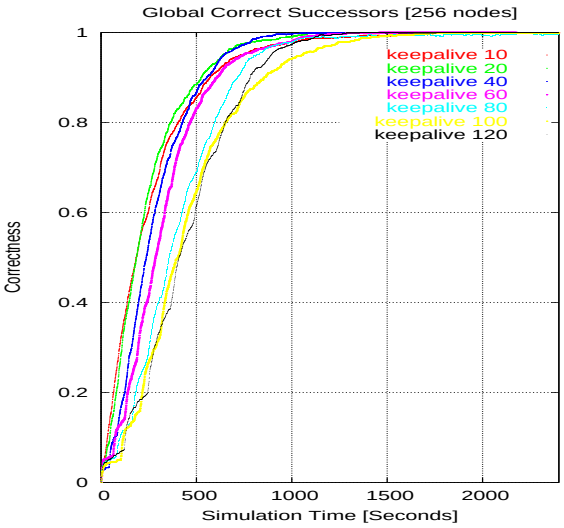


Figure 5.3: Keep-alive timeout: global correctness

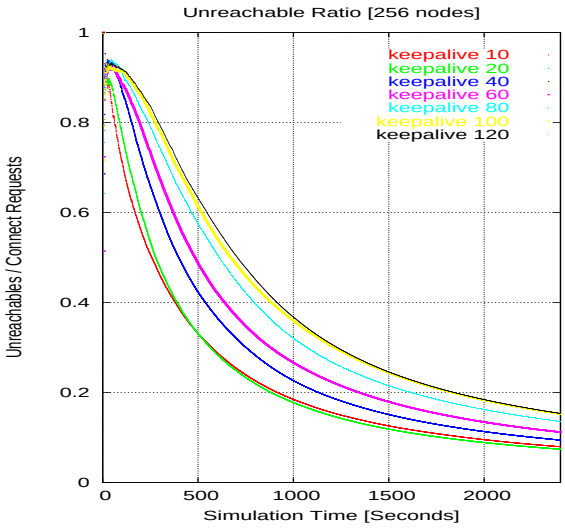


Figure 5.4: Keep-alive timeout: unreachable ratio

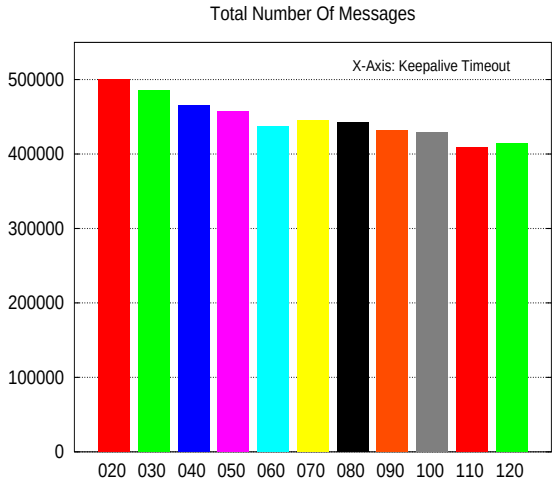


Figure 5.5: Keep-alive timeout: total messages

Parameter	Value
Path Store Size	50
ID Certificate Store Size	50
Path Store CleanUp Interval	40
Timestamp Acceptance	1000
Hello Timer	500
Certificates per SendIdMsg	2
Successor Notification Timeout	1.5
Successor Notification Timeout Retries	10
Successor Keep-alive Timeout	[10..120]
Connection Interval	uniform(10,30)

5.1.3 Retransmit Retries for the SuccessorNotification-Message

The table shows the parameters for this test. The keep-alive timer is set to 20s since this was the best result in 5.1.2 and the timeout retries lie in the range [3..16] with steps of 1 second. To keep the figure readable only every second graph is plotted. The best result is achieved for 14 retries which is a surprising large number. If the retry value is too small the estimated correct successor is dropped and a worse successor is chosen. This worse successor will probably answer with a *SuccessorUpdate-Message* since the choice is not correct. Then the former node is notified with a *SuccessorNotification-Message* again. These messages are wasting CPU time and cause message delay. If the value is too big, a node will try very long to contact the successor which is probably down or there is too much network traffic so that the messages are delayed and the node is not able to answer with a *SuccessorNotificationAck-Message* in time. Then sending one notification after another is very counterproductive since these messages cause delay too.

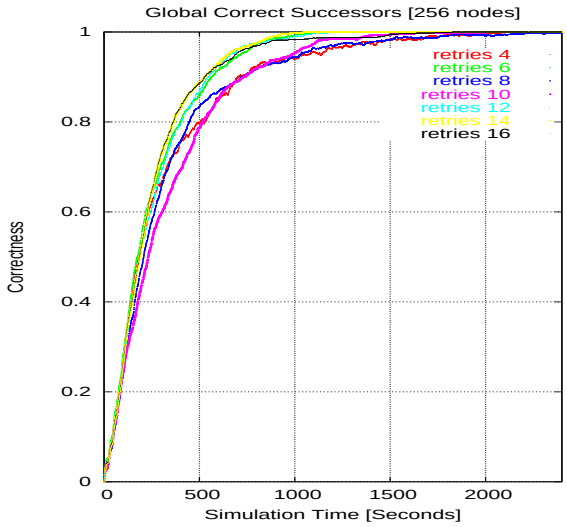


Figure 5.6: Successor notification retries: global correctness

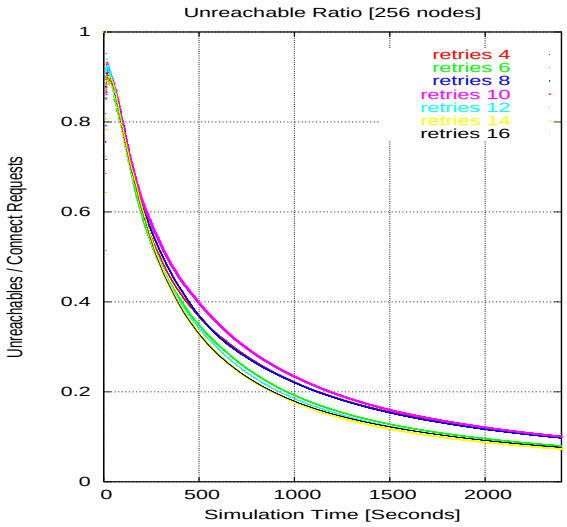


Figure 5.7: Successor notification retries: unreachable ratio

Parameter	Value
Path Store Size	50
ID Certificate Store Size	50
Path Store CleanUp Interval	40
Timestamp Acceptance	1000
Hello Timer	500
Certificates per SendIdMsg	2
Successor Notification Timeout	1.5
Successor Notification Timeout Retries	[3..16]
Successor Keep-alive Timeout	20
Connection Interval	uniform(10,30)

5.1.4 Certificates per SendIdCertificateMsg

This parameter adjusts the number of ID certificates in a *SendIdCertificate-Message*. Normally a node that tries to insert a path into the path store is not aware of all ID certificates of all nodes that are part of the path. As a result it is not possible to verify all one-way certificates of the path. In that case the node sends a *GetIdCertificate-Message* along the path to the last node whose certificate is missing. This message contains all requests. Any inner node tries to fulfill the requests by looking up the requested certificates in its ID certificate store and sending a *SendIdCertificate-Message* back to the requesting node if a certificate is found, but how many ID certificates should be allowed in a *SendIdCertificate-Message*? If only 1 certificate is allowed, many messages are generated. If too much are allowed, the message's size grows very fast since an ID certificate is large.

The table shows the parameters for this test. Allowing more than 5 certificates per *SendIdCertificate-Message* does not make sense since the message size is too big then. We expect a good result for 2 instead of 1 ID certificate per message but no big enhancement for the values 3...5 that would legitimate such big messages. In this test, besides from the global correctness and the unreachable ratio, the total number of messages and the total number of *GetIdCertificate-Message* and *SendIdCertificate-Message* are examined.

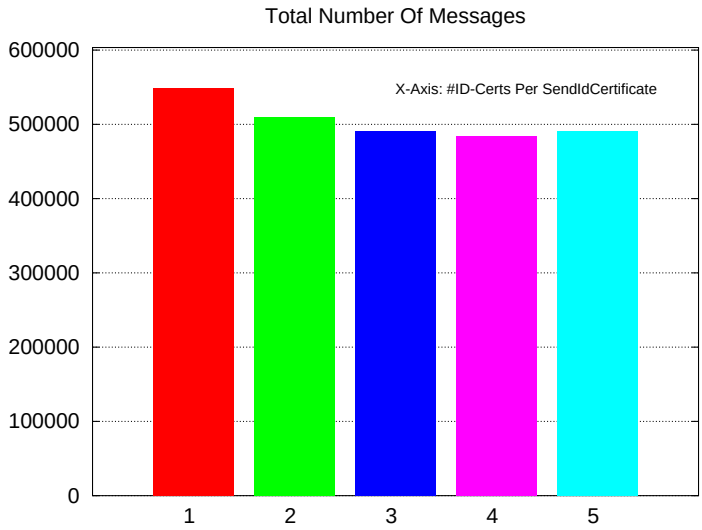


Figure 5.8: ID certificates per SendIdCertificateMsg: total messages

The unreachable ratio and the global correctness are not much affected by this test, so there are no plots. More interesting is the number of total messages and the number of *SendIdCertificate-Message*s. Figure 5.8 and 5.9 show that between 1 and 2 certificates per message there is major difference while the step from 2 to 3 or from 3 to 4 certificates is not that gainful. If the maximum message size allows to send 3 certificates per message that would be a good choice since 4 or more certificates is not feasible in most cases. For the next tests we set the parameter to 2.

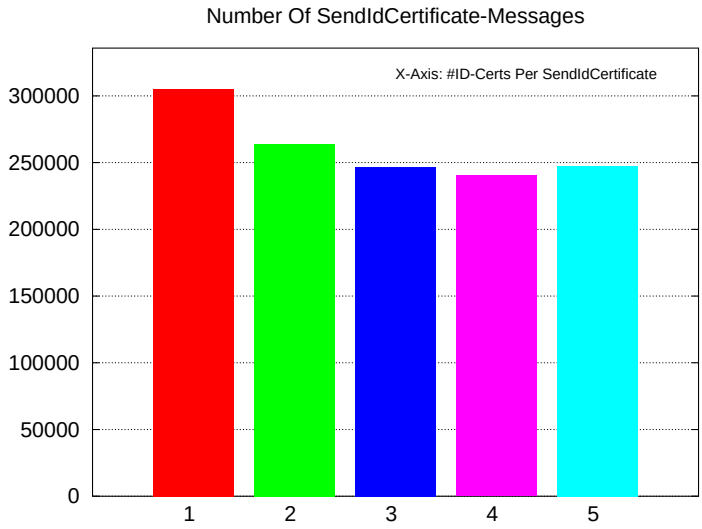


Figure 5.9: ID certificates per SendIdCertificateMsg: SendIdCertificate messages

Parameter	Value
Path Store Size	50
ID Certificate Store Size	50
Path Store CleanUp Interval	40
Timestamp Acceptance	1000
Hello Timer	500
Certificates per SendIdMsg	[1..5]
Successor Notification Timeout	1.5
Successor Notification Timeout Retries	14
Successor Keep-alive Timeout	20
Connection Interval	uniform(10,30)

5.1.5 Timestamp Acceptance

The next test tries to find two parameters: the timestamp acceptance which influences the expiration of the link-certificates and the hello timer which controls the frequency of exchanging *Hello-Messages*. A one-way certificate is valid if

$$T_{current} < T_{timestamp} + T_{acceptance}.$$

The hello timer is always half the timestamp acceptance value. The value of the hello timer must allow the nodes to exchange hello messages and one-way certificates in time to keep the certificates of the paths always up-to-date.

The best result for this network is for $T_{acceptance} = 300s = 5min$. Since this requires loosely synchronized clocks and the simulation does not implement unsynchronized clocks between the nodes, the value is perhaps too small in reality. If the clocks are not synchronized, many certificates are dropped even though they were issued recently.

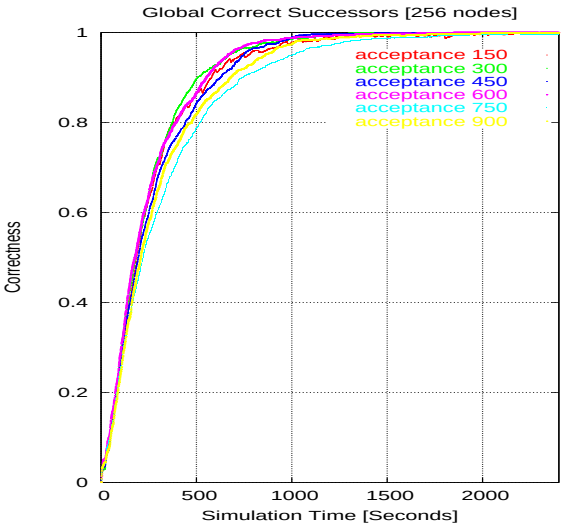


Figure 5.10: Timestamp acceptance: global correctness

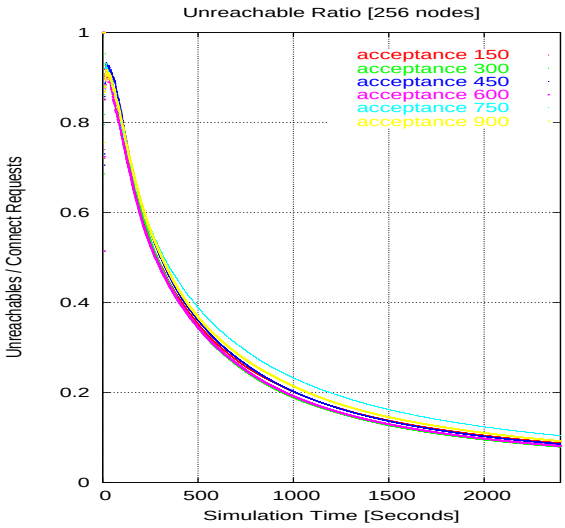


Figure 5.11: Timestamp acceptance: unreachable ratio

The acceptance value does not affect the number of messages pretty much, except for $T_{acceptance} = 150s$. This can be explained with the fraction of the *HelloMessage*s, *HelloAck-Message*s and *SendOneWayCertificate-Message*s of the total messages. This overview is shown in the figures 5.12, 5.13 and 5.14. The fraction is not very big since the lion's share are the *SendIdCertificate-Message*s. The many *SendIdCertificate-Message*s slow down the bootstrapping phase compared to the old protocol. When enhancing the new protocol this should be a starting point. If the distribution of link load is considered for each message type, the fraction of *SendIdCertificate-Message*s is still very large.

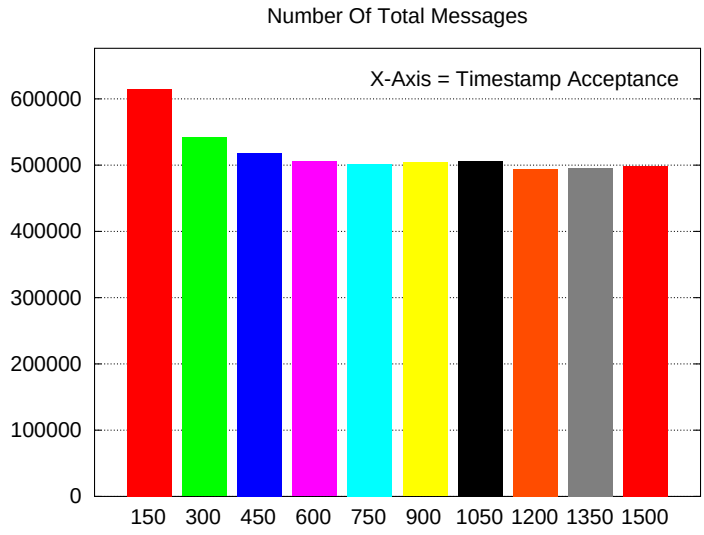


Figure 5.12: Timestamp acceptance: total messages

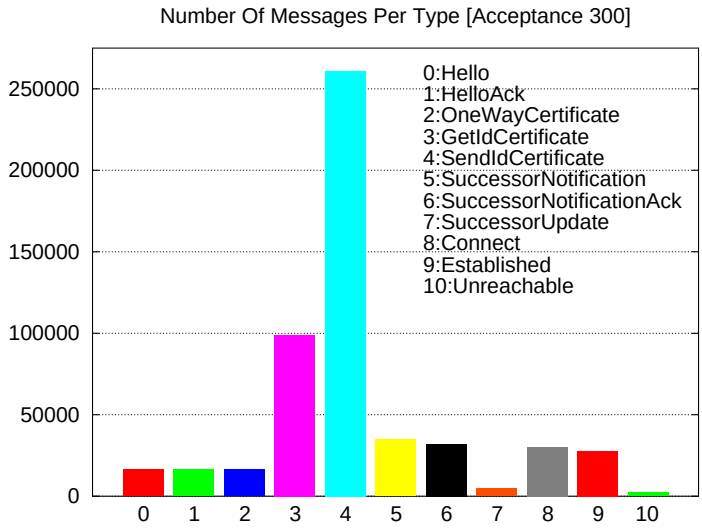


Figure 5.13: Timestamp acceptance: messages per type

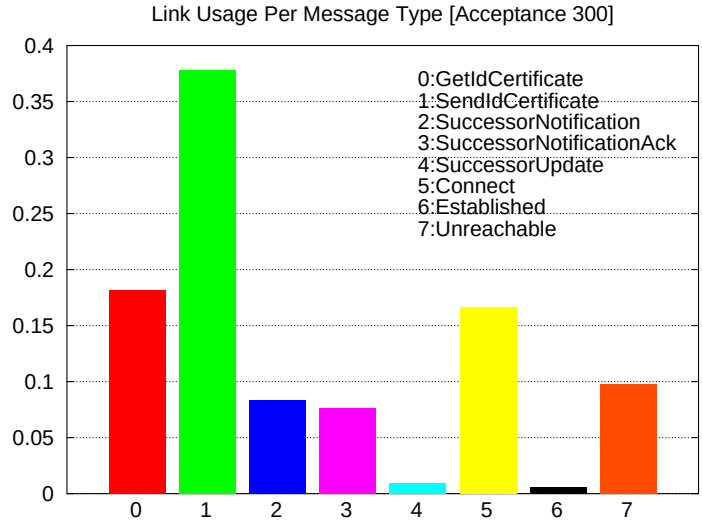


Figure 5.14: Timestamp acceptance: message fraction

Parameter	Value
Path Store Size	50
ID Certificate Store Size	50
Path Store CleanUp Interval	40
Timestamp Acceptance	[150..1500]
Hello Timer	[75..750]
Certificates per SendIdMsg	2
Successor Notification Timeout	1.5
Successor Notification Timeout Retries	14
Successor Keep-alive Timeout	20
Connection Interval	uniform(10,30)

5.1.6 Sign and Verify Operations

Consider the simulation run number from 5.1.5 with $T_{acceptance} = 300s$. Figure 5.15 shows the asymmetric cryptographic operation distribution. There are about 34

times more verify than sign operations which is quite asymmetric. The number of signing operations does not contain watchdog operations. The protocol does not propose concrete cryptographic algorithms but an algorithm with lower verification costs than signing costs is recommended.

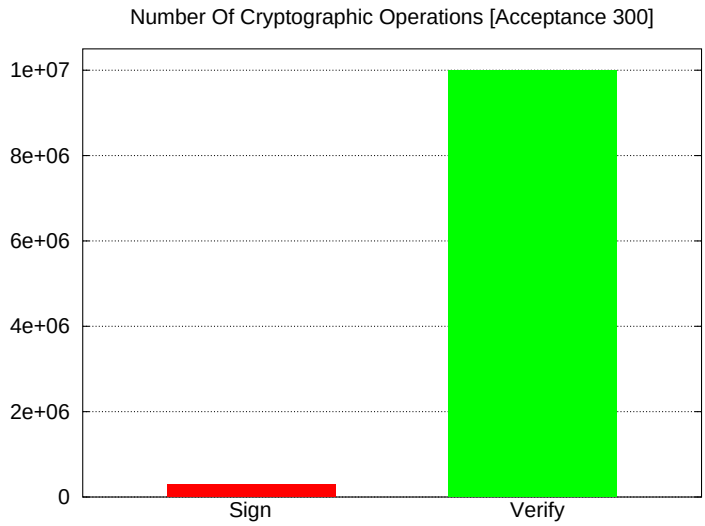


Figure 5.15: Sign and verify operation overview

Now we vary the sign and verify time and show the results. All simulations above assume verify operations to be 10 times faster than sign operations. With faster cryptographic operations the tests above must be repeated since it is not clear if for example a former value that had bad results is now bad too, since the network behaves differently with faster cryptographic operations. The messages will be forwarded faster since the message delay is decreased.

5.1.7 Signing and Verification Time

This test will vary the delay time for sign and verify operations, in other words, vary the computational power of the nodes. All tests above worked with a sign delay of 100ms and a verify delay of 10ms.

In the first test the computation time for a sign operation is brought more in line with the one for a verify operation. Figure 5.16 shows that the unreachable ratio increases very fast if a verify operation is more expensive. Furthermore $T_{verify} = T_{sign} = 55ms$ gets the worst result. So a cryptographic algorithm with symmetric costs for verify and sign operations is not a good idea. The plot in figure 5.17 shows the average message delay. This is the time a message is delayed before forwarding it because the CPU is not free.

The second test increases the sign time in steps of 10ms and decreases the verify time in steps of 1ms at the same time. The result for the average message delay is shown in figure 5.18. Quite surprisingly there is no big difference between $Time_{verify} = 10ms$ and $Time_{verify} = 9ms$ but a big step to from $Time_{verify} = 9ms$ to $Time_{verify} = 7ms$.

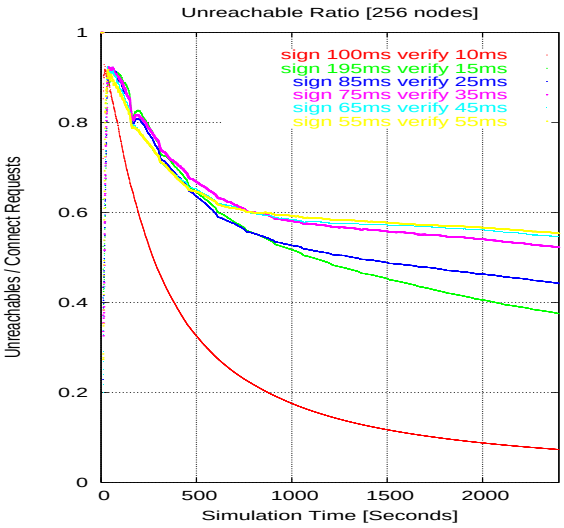


Figure 5.16: Unreachable ratio (different sign/verify costs)

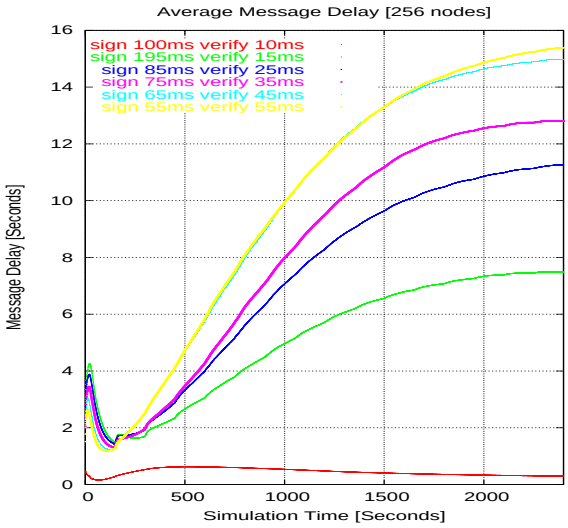


Figure 5.17: Average message delay (different sign/verify costs)

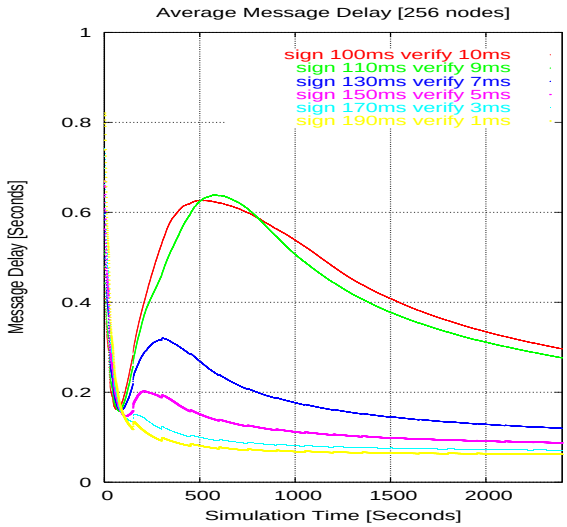


Figure 5.18: Average message delay for different sign/verify costs

Parameter	1st Test	2nd Test
Sign Time	[100..55]	[100..190]
Verify Time	[10..55]	[10..1]
Path Store Size	50	50
ID Certificate Store Size	50	50
Path Store CleanUp Interval	40	40
Timestamp Acceptance	300	300
Hello Timer	150	150
Certificates per SendIdMsg	2	2
Successor Notification Timeout	1.5	1.5
Successor Notification Timeout Retries	14	14
Successor Keep-alive Timeout	20	20
Connection Interval	uniform(10,30)	uniform(10,30)

5.1.8 Path Store and ID Certificate Store Size

A bigger path store will probably have better results with regard to the unreachable ratio and the global correctness since the nodes are able to store more paths and do not lose paths if the path store is full. But it must be kept in mind that a big path store needs more memory and more CPU time for searching, inserting and clean-up operations. The implementation of this simulation does not consider memory costs.

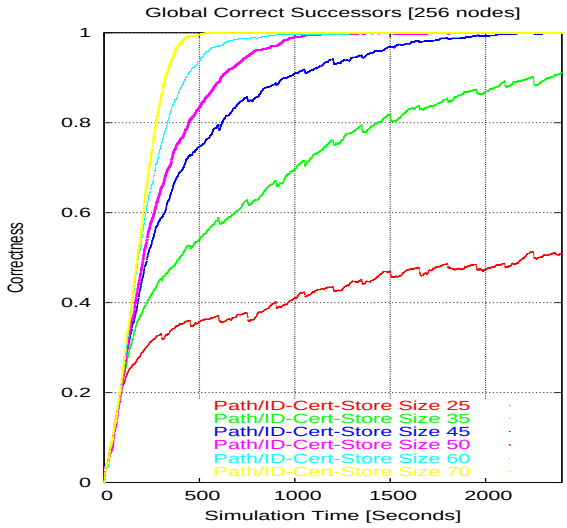


Figure 5.19: ID certificate and path store size: global correctness

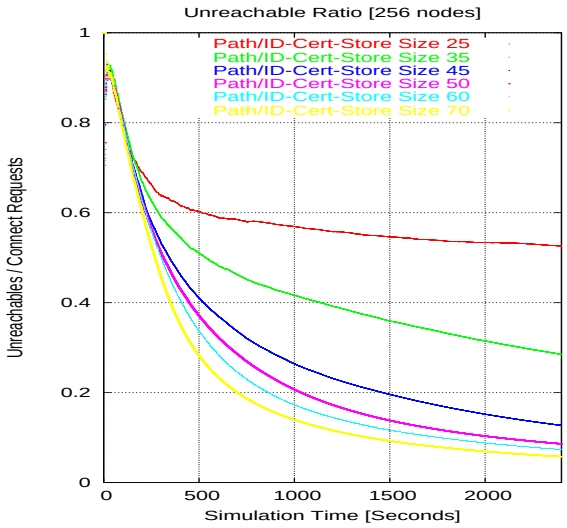


Figure 5.20: ID certificate and path store size: unreachable ratio

Figure 5.19 and 5.20 show that a path store size of 25 does not have feasible results. The unreachable ratio for example stays near 0.5 and we do not expect it to reach 0. The biggest path store of size 70 has the best results which is no big surprise. The size 50 which is the size of all tests above, has respectable results too. There is always a minimum path store size that corresponds with the diameter of the network since all nodes must be able to insert their successor. If there are successor paths with a length bigger than the path store size, the global correctness of 100% is never reached.

The second test tries to figure out if a bigger path store or a bigger ID certificate store is more important. In this test the sum

$$Size_{PathStore} + Size_{IDCertificateStore} = 100 = constant$$

and then both values are varied in steps of 5. For example if the path store size is 30, the ID certificate store size is 70.

Figure 5.21 and 5.22 show the global correctness and the total number of *SendIdCertificate-Messages*. The best results are achieved for runs with a path store size about the size of the ID certificate store. If the ID certificate store size is too small the number of *SendIdCertificate-Message* increases which results in message delay and more traffic. Comparing the size $(25, 75)$ and $(75, 25)$, where the first number is the size of the path store and the second number is the size of the ID certificate store, shows that in the beginning of the bootstrapping phase a small path

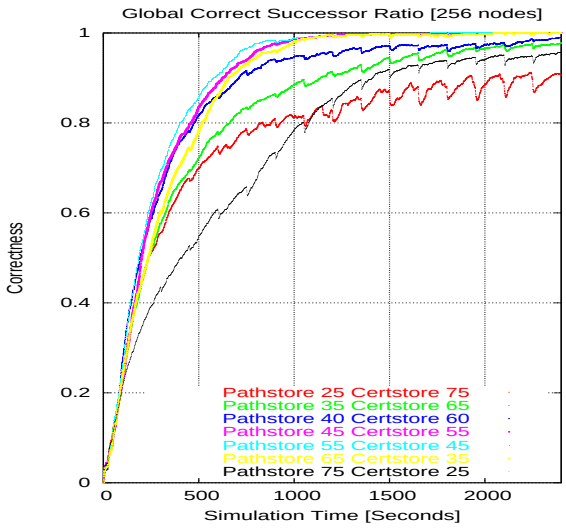


Figure 5.21: Correctness for different certificate and path store sizes

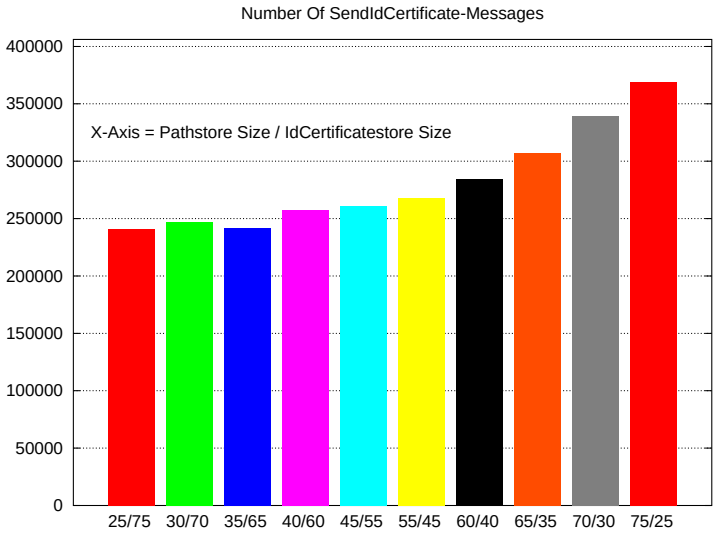


Figure 5.22: Messages for different certificate and path store sizes

store does not matter and has a better result but later nodes with long successor paths lose their successor when cleaning up the path store. This would explain the zigzag characteristics of the plot.

Parameter	1st Test	2nd Test
Path Store Size	[70..25]	[75..25]
ID Certificate Store Size	[70..25]	[25..75]
Path Store CleanUp Interval	40	40
Timestamp Acceptance	300	300
Hello Timer	150	150
Certificates per SendIdMsg	2	2
Successor Notification Timeout	1.5	1.5
Successor Notification Timeout Retries	14	14
Successor Keep-alive Timeout	20	20
Connection Interval	uniform(10,30)	uniform(10,30)

5.1.9 More Nodes

This test just increases the number of nodes without changing the parameters. The parameters from the best result of test 5.1.5 are used. The time until each node is aware of its correct successor will increase which is totally clear. But it must be kept in mind that with a larger number of nodes the evaluation steps must be repeated since the current parameters show the best result for 256 nodes.

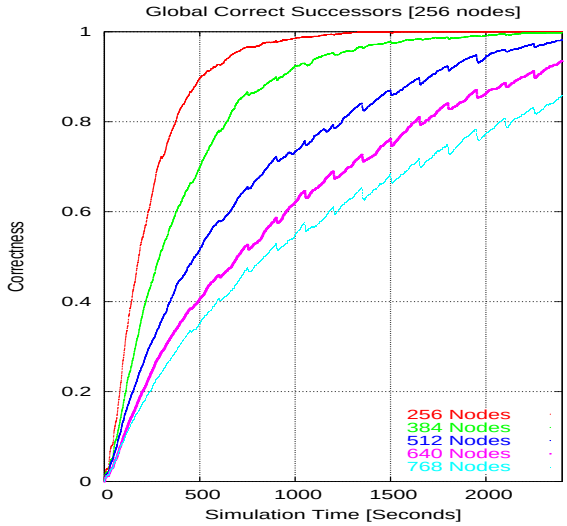


Figure 5.23: Global correctness for increasing number of nodes

Figure 5.23 shows the global correctness for network sizes from 256 to 768 nodes. The characteristic zigzag of the line for bigger networks might be a result of too short timeouts. As we expected the parameters must be adapted for bigger networks.

5.2 An Attack against the Standard Protocol

This section introduces an attack against the standard protocol. The goal of the attack is to get as many connections as possible and to disturb the global correctness of the network. After describing the behavior of the node in the first subsections, the result of the attack will be analyzed.

5.2.1 Resources

The malicious node does not maintain a path store. In other words it is able to perform the attack without any state. Any information about the network and the victims are taken from the received and forwarded messages. The problem with maintaining a path store could be that the malicious node has to store all Sybil identities and fake routes. Otherwise the malicious node will poison its own path store when the routes return and it did not store the information about the fake routes earlier. The problem is that storing all fake identities and routes is very expensive.

5.2.2 Handling *Hello-Messages*

The malicious node does not send any *Hello-Messages* on start up. If it receives a hello from a neighbor with address X, it generates a *Hello-Messages* containing

the source address $X + 1$ pretending to be the best successor of node X. As a result node X will forward all *ConnectRequest*s with an incomplete route to the malicious node, because if no path towards the destination is found, a node forwards the message to the successor. This is then the malicious node and not the real successor. Furthermore the malicious node does not use its correct address for any communication with other nodes.

5.2.3 Handling *SuccessorUpdate-Message*s

After sending *SuccessorNotification-Message*s with fake addresses to all neighbors the neighbors will respond to these messages with a *SuccessorUpdate-Message* since $X + 1$ is definitely not the correct predecessor of node X. These messages contain an update path to a better successor. Each *SuccessorUpdate-Message* is answered with a *SuccessorNotification-Message* since the malicious node does not maintain correct successors and predecessors. Furthermore each node of the update path is fooled with a fake *SuccessorNotification-Message* from the perfect successor. The procedure is described below in 5.2.5.

5.2.4 Handling *SuccessorNotification-Message*s

Since the malicious node will only get *SuccessorNotification-Message* for fake addresses, these messages are dropped. As a result the sender of the message believes that the receiver, the fake address, agreed with the choice because no *SuccesssorUpdate-Message* is sent back.

5.2.5 Forwarding Messages

Normally a message is forwarded along the path in the message. The malicious node behaves the following way before forwarding a message. For each node X in the path, it generates a *SuccessorNotification-Message* containing the path

$$X + 1 \longleftrightarrow \textit{malicious node} \longleftrightarrow \cdots \longleftrightarrow X$$

The advantage of sending a *SuccessorNotification-Message* instead of a *SuccessorUpdate-Message* is that it is inserted into the path store too and the destination node will send a *SuccessorNotification-Message* to $X+1$ anyway after checking the path store the next time since $X + 1$ is the perfect successor. But furthermore the node X will send a *SuccessorUpdate-Message* back to $X + 1$ since $X + 1$ is not the correct predecessor of X. This message contains a path and an update path and consequently more victim nodes to disturb with fake paths and successors.

5.2.6 Handling *ConnectRequest*s

For any *ConnectRequest* that is received, destined to the malicious node or not, an *Established-Message* is sent back immediately. The path is inverted and used to send the *Established-Message*. To distinguish the fake connections from real connections, a flag is set. This simplifies recording the statistical data.

5.2.7 Results

Figure 5.24 shows the mean correctness and mean unreachable ratio of this attack. 100 runs with 256 nodes and 1 sinkhole node were averaged.

The unreachable ratio differs from the unreachable ratio of the previous figures. If a node receives an *Established-Message* with the flag 'Sent By Sinkhole' set, this counts as a connect failure because the node did not establish a connection to the correct node. So the unreachable message counter is increased. This emphasizes the influence of the malicious node.

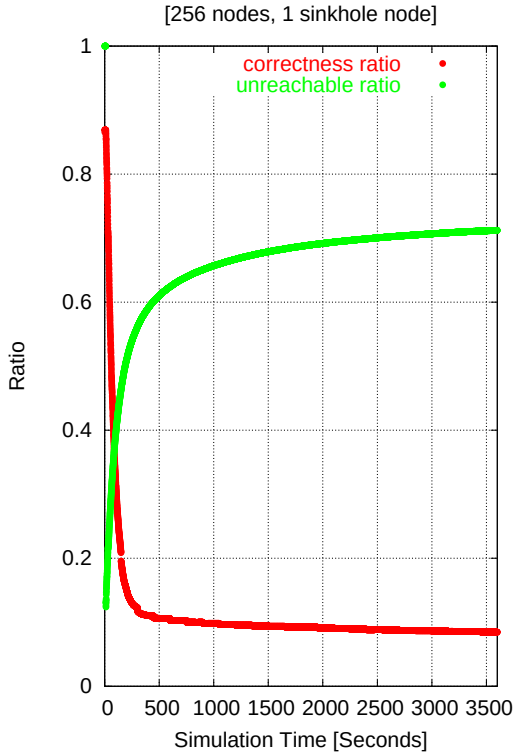


Figure 5.24: Sinkhole attack: correctness and unreachable ratio

In reality the node that sent the *ConnectRequest* may not even realize that the connection is not established to the correct node but to the malicious node. This attack can be extended to a *man-in-the-middle* attack if the malicious node forwards the original *ConnectRequest* to the correct node. This requires that the the malicious node is still part of the path when the *ConnectRequest* reaches the correct receiver. The correctness ratio never reaches 1 in this scenario and decreases very fast. Only one sinkhole node controls a large part of the network. It must be kept in mind that the sinkhole node does not use a path store to perform this attack.

Starting with fooling the physical neighbors, the first *SuccessorUpdate-Message*s are used to reach and fool more nodes. More and more nodes have the sinkhole node as successor and all *ConnectRequest*s with incomplete paths are sent to the sinkhole node. These paths contain more victims and so more and more nodes are fooled and the sinkhole node gains more and more control of many connections.

Figure 5.25 shows the correct established connections per node. The left bar shows the scenario for 256 nodes without any sinkhole node. The right bar shows a scenario

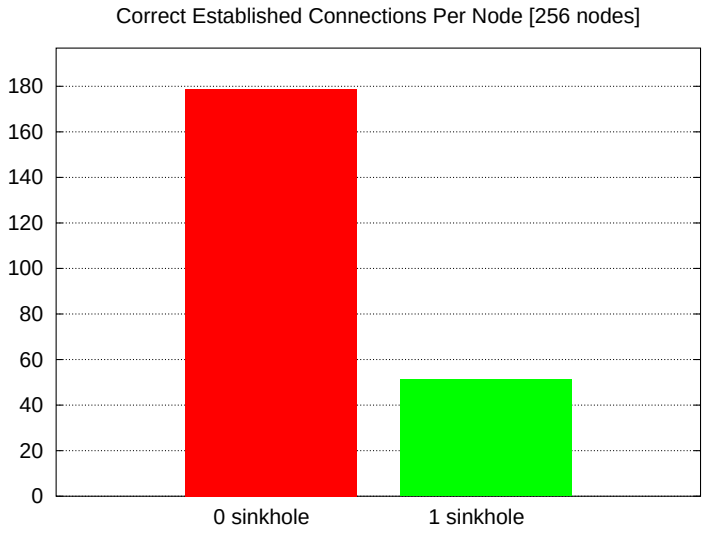


Figure 5.25: Sinkhole attack: correct established connections

with one sinkhole node attacking the network. About 180 connections per node are established if all nodes behave correct. With only one sinkhole node the value is about 51. Furthermore the correct connections are established in the beginning. Later the malicious node controls the lion's share of the connections. The mean established connection value for the sinkhole is about 30 000 which is an impressive value.

5.3 An Attack against the Secure Protocol

This section examines the effect of the same attack as described above in 5.2 against the enhanced protocol. The behavior is nearly the same as in the attack against the standard protocol except that the malicious node shows a correct behavior for its real address. The reason is that it is not able to fool its neighbors with wrong addresses. The neighbors will not accept these addresses because the ID certificate is not signed by the master key. So the only possibility to obtain node addresses of victims is to scan the paths of messages like *SuccessorNotification-Message*s.

5.3.1 Bootstrapping Behavior

The malicious node must use its correct address for the communication with its neighbors. Otherwise the neighbors will not accept the neighborhood of the sinkhole node. So *Hello-Message*s, *HelloAck-Message*s and *OneWayCertificate-Message*s are answered with respect to the protocol. In addition a *SuccessorNotification-Message* is sent to each neighbor immediately after the neighbors sent the one-way certificates. Some of the notifications will be answered with a *SuccessorUpdate-Message* which are used to get aware of more victims.

5.3.2 Handling *SuccessorNotification-Message*s

Any *SuccessorNotification-Message* is answered with a *SuccessorNotificationAck-Message* without checking whether the choice of the sender is correct, since this is not possible without a path store. In addition for each neighbor a *SuccessorUpdate-Message* is sent back to the sender containing the update path

$$sender \leftrightarrow \ldots \leftrightarrow malicious\ node \leftrightarrow neighbor.$$

This should ensure that other nodes learn about the paths to the neighbors. Otherwise the malicious sinkhole node will never be an inner node and will not have the ability to obtain a *ConnectRequest*.

5.3.3 Signing and Verifying

The malicious node does not care if a message's signature is correct or corrupted. Furthermore the node does not check the link-certificates and does not request any ID certificates. The only cryptographic operation is the signing of messages that are sent with the malicious node itself as sender. If it generates a Sybil identity with the path *'sinkhole'* $\leftrightarrow$ *'Sybil'* it sets the correct start and end node but is not able to fake the one-way certificate *'Sybil'* $\rightarrow$ *'sinkhole'* .

5.3.4 Results

The evaluation uses 50 runs and averages the results.

5.3.4.1 Bootstrapping

The neighbor nodes do not accept the Sybil identities so the malicious node is not able to choose identities and act as the perfect successor for its neighbors. This is a problem for reaching more victim nodes since the only way to obtain more addresses is to learn paths from the neighbors. As a result the sinkhole node must send *SuccessorNotification-Message*s to each of its neighbors to get information about the network and about new victims. As a second result, the malicious node will not get more *ConnectRequest*s with incomplete paths from its neighbors like in the previous attack because it is not the successor for all neighbors.

5.3.4.2 Connections

All *ConnectRequest*s with the sinkhole node as an inner node will not be established correctly since they do not reach the original destination. The enhanced protocol does not include a countermeasure against nodes that just drop or refuse to forward messages. So these connections are lost. But the malicious node is not able to fool the sender of the *ConnectRequest* because it is not possible for the sinkhole node to sign the *Established-Message* like the original receiver would do. So the node that receives the *Established-Message* will discard it because the signature is corrupted. Furthermore in the previous attack the Sybil identities are part of the network because they are stored in the path stores of other nodes. Normal nodes try to connect to the Sybil nodes because they seem to be correct addresses. In the enhanced protocol the Sybil identities are not inserted in the path stores because there is no correct link-certificate that ensures the direction

$$Sybil\ node \rightarrow sinkhole\ node.$$

As a result the complete link-certificate *Sybil node* $\leftrightarrow$ *sinkhole node* is broken.

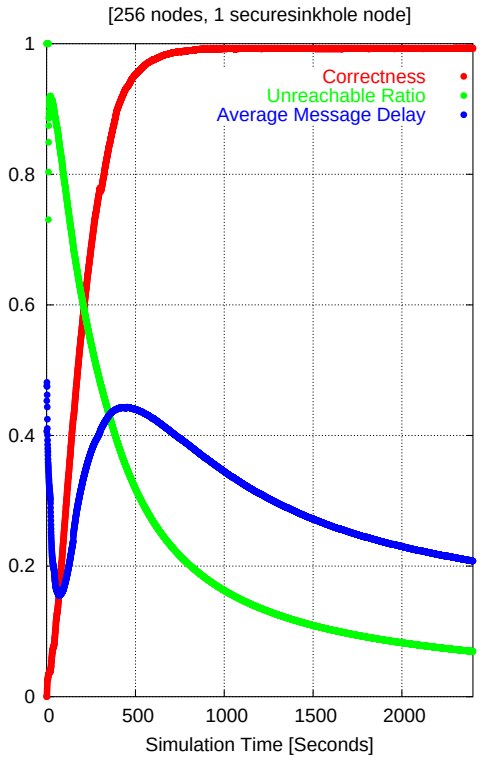


Figure 5.26: 'Secure' sinkhole attack: correctness, unreachable, delay

5.3.4.3 Message Delay and Traffic

The only effect of this attack is that there is more traffic and message delay since the receiver of fake messages and fake paths will probably send *GetIdCertificate-Message*s to verify the path *Sybil node* ↔ *sinkhole node*. But no node is able to answer this request and the message finally arrives at the sinkhole node which will just drop it.

Figure 5.26 shows that the attack does not affect the network. The global correctness is achieved and the messages delay is acceptable. On average the sinkhole node uses about 182 Sybil identities to fool the other nodes and send about 165 *Established-Message*s back to the sender. The behavior of the sinkhole node let about 10 800 signature checks fail. 0.41 connection requests per node are not accepted because the sinkhole node fakes the *Established-Message* and still about 169 are accepted. This is a good result.

6. Conclusion and Future Work

6.1 Future Work

6.1.1 Speeding up the Bootstrapping

6.1.1.1 Path-Caching

There is a problem with the link-certificates. If a node sends a *SuccessorNotification-Message* to another node, the receiving node will in most cases not be able to verify the path completely since it is not aware of all ID certificates and has to request the missing certificates with a *GetIdCertificate-Message*. In this case the receiving node does not respond with a *SuccesssorNotification-Message* and the sending node must wait until the timeout expires and then retransmit the message.

Then another problem can come up. If an inner node enhances the path with respect to the path length, the receiving node may again not be able to verify and insert the path completely because the path has changed. Then another timeout period is necessary before the message is retransmitted again. This behavior could be improved by caching the path. If the receiver is not able to insert the path completely, the path is stored in a path cache. If all ID certificates have been received the path is inserted and a *SuccessorNotificationAck-Message* is sent. This may speed up the bootstrapping-phase.

6.1.1.2 Extra Message

Another possibility is to introduce a new message. This message would have the semantic 'Your successor choice is correct, but I was not able to verify the path. Please send the message again!'. It has to be evaluated if this message really enhances the bootstrapping phase. Of course it is possible that an extra message is too much overhead. Furthermore, if the retransmit occurs too early, the successor might not have the chance to obtain all missing ID certificates.

6.1.2 Different Network Topologies

There is a need to test the protocol with different network topologies. In the current topology the nodes are connected randomly and build up an *Erdős-Rényi-Graph*.

This is a problem if all nodes have equal CPU power, because some nodes have many neighbors and some nodes have only two neighbors. Nodes with many neighbors can be a bottleneck because they have to perform more cryptographic operations. As a result these nodes delay the messages because the CPU is often at full working load. As nodes with many neighbors are forwarding more messages than other nodes anyway, this again means that more traffic is delayed. The protocol would have better results if only nodes with more CPU power are allowed to have more neighbors.

6.1.3 Implementation Enhancements

6.1.3.1 Ban List

If a timeout for the successor expires n-times there probably is a problem with the successor node. In this case it may be a good idea to keep a ban list and to insert this node into the list. Otherwise the successor node is dropped from the path store but the next better successor node will probably answer with a *SuccessorUpdate-Message* if the information about the problem with the former node does not spread fast enough. Then there are another n tries and a timeout again. A ban list would prevent a node from contacting a successor again and again.

6.1.3.2 Removing ID Certificates after Removing Nodes

In this implementation the ID certificate of a node that is removed from the path store is not deleted directly from the ID certificate store but replaced with a LRU strategy. Removing the ID certificate allows more space for new certificates. On the other hand, if a node is removed and directly re-inserted, the ID certificate is still there and no *GetIdCertificate-Message* is needed to obtain it.

6.1.3.3 Flexible Number of Certificate per SendIdCertificate-Message

To handle different nodes and network types, the number of ID certificates per *SendIdCertificate-Message* could be more flexible. The requesting node may send the parameter in the request to show how many certificates are acceptable for it.

6.1.4 Denial of Service Attacks

The denial of service attack has not been examined in this study. A node can be attacked in the enhanced protocol by flooding since it will try to verify each message and this costs CPU time. On the other hand this attack is already possible by sending a lot of messages to this node if a node has low computational power and energy .

6.1.5 Reducing *SendIdCertificateMessage*s

The evaluation showed that there are a lot of *SendIdCertificate-Message*s in comparison to all other message types. If the huge number of these messages can be reduced, the protocol would be faster since more messages mean more cryptographic operations and more traffic.

One problem in the enhanced protocol are double requests. If a node tries to insert the same path twice because it received two different messages containing the same

path, the node will request the missing certificates twice if the first request has not been fulfilled already. A data structure is needed to prevent these double requests. A request could be stored and is then not sent again during a time period, but these entries have to be removed after a certain time since a *SendIdCertificate-Message*s could get lost and then the missing ID certificate would never be requested again.

6.1.6 Node Churn

Due to the time constraints of this study, node churn has not been examined. The basics are implemented but node churn has not been evaluated. It would be interesting to analyze the reaction of the new protocol with several churn rates. How long does it take until the network learns about the new node and how long does it take until a dead node has been removed from the path store of each node. If an *Unreachable-Message* causes a node to remove the unreachable node from its path store, this could be a great attack against the standard protocol. A malicious node could send many *Unreachable-Message*s to destroy the global correctness of the network.

6.2 Conclusion

After analyzing the standard protocol with respect to possible security problems some countermeasures were examined and some of them taken over to the enhanced protocol. The evaluation of the enhanced protocol shows that introduced security mechanisms slow down the protocol, because the asymmetric cryptographic operations are very expensive, especially for small devices.

On the other hand, the attack described in 5.2 shows that the standard protocol is vulnerable. This attack even works with low resources and does not require a strong attacker. The same attack against the enhanced protocol had nearly any effect except that it causes more traffic. Denial of service (DoS) attacks are still possible in the enhanced protocol and if verifying a corrupted message is expensive, flooding could be a big problem in the new protocol.

The lion's share of the traffic in the new protocol is caused by exchanging the ID certificates. These exchanges need a large number of messages and slow down the protocol. Future work may start at this point to speed up the protocol.

Bibliography

[AMCGR04] F. Almenarez, A. Marin, C. Campo, and C. Garcia-Rubio. Ptm: A pervasive trust management model for dynamic open environments, 2004.

[AN02] Jari Arkko and Pekka Nikander. Weak authentication: How to authenticate unknown principals without trusted parties, 2002.

[BF01] D. Boneh and M. Franklin. Identity based encryption from the weil pairing, 2001.

[BGK04] Zinaida Benenson, Felix Gaertner, and Dogan Kesdogan. User authentication in sensor networks, 2004.

[BSCP03] Pedro Brandao, Susana Sargento, Sergio Crisostomo, and Rui Prior. Ad-hoc routing security report, 2003.

[BSSW02] Dirk Balfanz, D.K. Smetters, Paul Stewart, and H.Chi Wong. Talking to strangers: Authentication in ad-hoc wireless networks, 2002.

[BT03] Mathias Bohge and Wade Trappe. An authentication framework for hierarchical ad hoc sensor networks, 2003.

[Buc01] Johannes Buchmann. *Einführung in die Kryptographie.* Springer-Verlag Berlin Heidelberg New-York, 2001.

[Bur03] Adam Burg. Ad hoc network specific attacks. Seminar: Ad hoc networking, concepts, applications and security, TU München, 2003.

[Cam04] Stefano Campadello. Peer-to-peer security in mobile devices: a user perspective, 2004.

[CDG+02] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. 5th Usenix Symposium on Operating Systems Design and Implementation, Boston, MA, 2002.

[CF04] Curt Cramer and Thomas Fuhrmann. Isprp: A message-efficient protocol for initializing structured p2p networks. Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference (IPCCC), Phoenix, AZ, April 7-9, 2005, pp. 365–370, 2004.

[Dou02] John R. Douceur. The sybil attack. 1st International Workshop on Peer-to-Peer Systems, 2002.

[Fuh04] Thomas Fuhrmann. A self-organizing routing scheme for random networks. Proceedings of the 4th IFIP-TC6 Networking Conference, LNCS 3462, Waterloo, Canada, May 2-6, 2005 pp. 1366-1370, 2004.

[HBC01] J. P. Hubaux, L. Buttyan, and S. Capkun. The quest for security in mobile ad hoc networks, 2001.

[HPJ02] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Wormhole detection in wireless ad hoc networks, 2002.

[HPJ03a] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Efficient security mechanisms for routing protocols, 2003.

[HPJ03b] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Packet leashes: A defense against wormhole attacks in wireless networks. IEEE INFOCOM 2003, 2003.

[HW02] Steven Hazel and Brandon Wiley. Achord: A variant of the chord lookup service for use in censorship resistant peer-to-peer publishing systems, 2002.

[JNM+04] Petri Jokela, Pekka Nikander, Jan Melen, Jukka Ylitalo, and Jorma Wall. Host identity protocol - extended abstract, 2004.

[KHG03] Jiejun Kong, Xiaoyan Hong, and Mario Gerla. A new set of passive routing attacks in mobile ad-hoc networks, 2003.

[KW03] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: attacks and countermeasures, 2003.

[MR01] Philip MacKenzie and Michael K. Reiter. Two-party generation of dsa signatures, 2001.

[NSSP04] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig. The sybil attack in sensor networks: Analysis and defenses, 2004.

[PH02] Panagiotis Papadimitratos and Zygmunt J. Haas. Secure routing for mobile ad hoc networks, 2002.

[rsc05] rsa-security com. What is a hash function? www.rsasecurity.com RSA Laboratories' Crypto FAQ Version 4.1, 2005.

[Sch96] Bruce Schneier. *Applied Cryptography.* John Wiley and Sons, Inc. New York, 1996.

[SL04] Mudhakar Srivatsa and Ling Liu, 2004.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications, 2001.

[TW04] Colin Kelley Thomas Williams. Gnuplot. www.gnuplot.info, 2004.

[Var05] András Varga. Omnet++ discrete event simulation system user manual. www.omnetpp.org, 2005.

[Wal02] Dan S. Wallach. A survey of peer-to-peer security issues, 2002.

[WB04] Weichao Wang and Bharat Bhargava. Visualization of wormholes in sensor networks, 2004.

[Wei04] Andre Weimerskirch. Authentikation in ad-hoc und sensornetzwerken, 2004.

[Wid00] Andreas Widmann. The boxfill perl script. widmann@rz.uni-leipzig.de, 2000.

[WT01] Andre Weimerskirch and Gilles Thonet. A distributed leight-weight authentication model for ad-hoc networks, 2001.

[ZH99] Lidong Zhou and Zygmunt J. Haas. Securing ad hoc networks, 1999.

[Zim05] Phil Zimmermann. The openpgp alliance home page. www.openpgp.org, 2005.

List of Figures