Intrusion tolerance in Publish/Subscribe based MANET

Erland Kolstad
erlandkolstad@gmail.com
+47 907 25 635

Master thesis
Master of Information Security
Department of Computer Science and Media Technology
Gjøvik University College, 2009

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

Public/Subscribe (PubSub) paradigm is a powerful abstraction for building distributed applications and message distribution networks, and seems to be well suited model for the type of communication which takes place at a tactical/mobile level in military operations. Because of the hostile environment such networks have to operate in, they have to have good information security properties, including intrusion tolerance. A successful intrusion in tactical command and control networks can have a substantial damage on the ongoing military operation, making it a highly valuable enemy target.

Most previous work on information distribution in MANETs focuses on how to distribute the information and minimize the amount of data to be sent on the network, and not on security issues introduced by an intruder.

During this master project we have performed an analysis of the intrusion tolerance of the PubSub based MANET, i.e. the capability of a system to fulfill its mission in a timely manner, even when the network is under different types of attacks. The analysis shows that the PubSub protocol is vulnerable to some attacks performed by an intruder in the network.

To deal with those vulnerabilities we have proposed enhancements to the protocol to make it more robust and immune to these attacks. By implementing the proposed enhancements we show that the PubSub protocol can be made very robust against the different types of attacks studied in this thesis. These properties make the protocol suited for usage in communication networks which operate in hostile environments.

Acknowledgement

I would like to thank my supervisor, Professor Slobodan Petrovic, for feedback and guidance throughout the work on my master thesis. I would also thank PhD Anders Fongen for letting me use his protocol sketch, for a CBR Publish/Subscribe Ad-hoc protocol, as a basis for the thesis. I would also give a great thank to my fellow student Liubov Kokorina for great feedback. My wife Pia Kolstad also deserve a great thank for supporting me and pushing me future when the work has gone slow.

- Erland Kolstad, 15 June 2009.

Table of content

1. Introduction

1.1. Topic

The advance in wireless communication enables users to access information systems from anywhere at any time via various mobile devices. In mobile ad hoc networks (MANETs) all nodes are capable of moving actively and dynamically connected. Due to the lack of infrastructure, each node in a MANET also acts as a router which discovers and maintains routes, and forwards packets to other nodes. In a collaborative work scenario, several mobile nodes, with a small storage to store data items, may work together and access data items stored in other nodes. Because of the dynamics in MANET the topology and connectivity of the network often changes. This dynamic nature of MANETs makes information distribution in MANETs more challenging than in more static networks.

In many military applications information distribution is a key function. At a tactical level communication is radio based and all nodes are capable of moving actively and dynamically connect. This dynamic behavior gives them similar characteristic to MANETs.

Publish Subscribe (PubSub) paradigm is a powerful abstraction for building distributed applications and message distribution, and seems to be well suited model for the type of communication which takes place at a tactical/mobile level. PubSub networks seem to have some good qualities for use in MANETs [4]:

- Simplified protocols
- Reduced traffic used for router and mobility control
- High intrusion tolerance

In addition to these properties, the nature of how radios send messages makes them suitable to be used in PubSub networks; radio uses broadcast to send information, where PubSub uses multicast/broadcast.

As in any communication network an intruder in a PubSub based MANET might represent a number of threats to the network, including breach of confidentiality, integrity or availability.

In this master thesis we provide an analysis of the intrusion tolerance in a PubSub based

MANET. We will do so by providing answers to the following research questions:

1. How intrusions tolerant are PubSub based MANETs?
2. Are more densely populated networks more intrusion tolerant than less densely populated networks?
3. How will different structures of the network affect the intrusion tolerance?

1.2. Keywords

Information Security, Mobile data service, data accessibility, mobile computing, ad hoc networks, Survivability, Intrusion tolerance.

1.3. Importance of functional communication networks and message distribution systems in (military-) operations

1.3.1. The mission

In many situations we often refer to the mission, when we describe the process of reaching a desired goal. The mission is not limited to military settings, but also refers to other scenarios, where the vision of their objectives is expressed implicitly or as a formal mission statement. The decisions on whether or not a system has fulfilled its mission are typically made in the context of external conditions that my affect the achievement of the mission. [17]

In many military scenarios, efficient information distribution is a key factor of conducting successful operations (missions). It is also essential that this information is correct and kept secret, which requires high intrusion tolerance in such networks.

1.3.2. Information supremacy

When leaders have to decide the next move to achieve the objectives, the decision is normally the result of a decision making process. The decision will be made based on the amount and quality of the information available and the experience of the decision maker. The quality and amount of information available for the decision maker will thereby influence the quality of the decision; better information gives better decisions.

Accurate information and efficient distribution of relevant information is of the utmost importance in many operations to achieve information supremacy, and thereby gain the tactical mission by enabling:

- More efficient targeting and rapid operations
- Minimizing civilian losses, through more accurate targeting
- Minimize own losses, through rapid targeting the enemy and reducing chance of getting hit by enemy and friendly fire.

In many military applications, information distribution is a key function. But other services can also gain improvement when the amount and quality of available information increase.

- Emergency services (e.g. quicker situation awareness in disaster areas, patient information updates when patient is in transit, etc.)
- Other applications (e.g. traffic information updates to decrease traffic jam, etc.)

1.3.3. Information distribution

In most communication networks the main mission of the network is to support transport and exchange of information between participants in the network securely in a timely manner. In traditional communication networks the information is routed between nodes based on the addresses given to the nodes. At a tactical level where communication is radio based and all nodes are capable of moving actively and dynamically connect, the communication network has similar characteristic to MANETs. Address based transport of information requires an underlying routing protocol in order to transport the information through the network. Traditional routing protocols used in more static networks are not very well suited for the dynamic nature of MANETs, which require rapid routing updates which will increase the traffic load in the network.

Different publish-subscribe based message distribution models have been developed for use in mobile ad-hoc environments to achieve more efficient information distributions in such dynamic environments.

1.4. Introduction to Public Subscribe Ad-hoc networks

1.4.1. Mobile ad-hoc networks

Mobile ad-hoc networks (MANET) have some advantages over traditional networks as they don't need any infrastructure for operation. Due to the lack of infrastructure, each node in a MANET also acts as a router, which discovers and maintains routes, and forward packets to other nodes. The networks are created spontaneously as peer-to-peer networks when two participants are within range of each other. When more participants join, leave or change geographical position in the network, the network seamlessly reconfigures allowing the participants to continue to communicate with each other.

In a collaborative work scenario, several mobile nodes, with a small storage for data items, may work together and access data items stored in other nodes. Because MANETs are highly flexible they tend to be very dynamic; network topology can change at a high frequency because of users joining, leaving or changing geographical position in the network. The topology changes are also often results of failing network links, limited bandwidth etc. Mobile nodes are normally low power, battery operated devices with limited bandwidth compared to stationary nodes. The constrained resources and highly dynamic environments are challenging for tightly coupled distributed applications, where communication failures are the norm rather than the exception. [13]. This dynamic nature of MANETs makes information distribution in MANETs more challenging than in more static networks.

1.4.2. Content-based routing

Data-centric networking protocols use content addressing instead of host addressing for participating nodes, thus decoupling application communication from underlying network topology. Equally important for MANET is the possibility of routing protocols that can provide highly selective information dissemination between peers, thus making group communication more efficient: the routing protocol performs timely filtering of data, thus minimizing the amount of data that needs to be communicated between peers.

Many applications can benefit from selective and decoupled dissemination including reporters at a news conference sending real-time photos and news to interested readers, teammates communicating with each other, information about arriving patients in a hospital etc.

In content based routing all messages within a given theme will be assigned a pathname, and all themes are organized in different paths. Figure 1 shows an example of a hierarchy of themes that can be subscribed. Any subscription covers a subtree of themes, e.g. a node which subscribes to **AB** will receive messages with theme **AB** and **ABA**, but not **A** or **AA**.

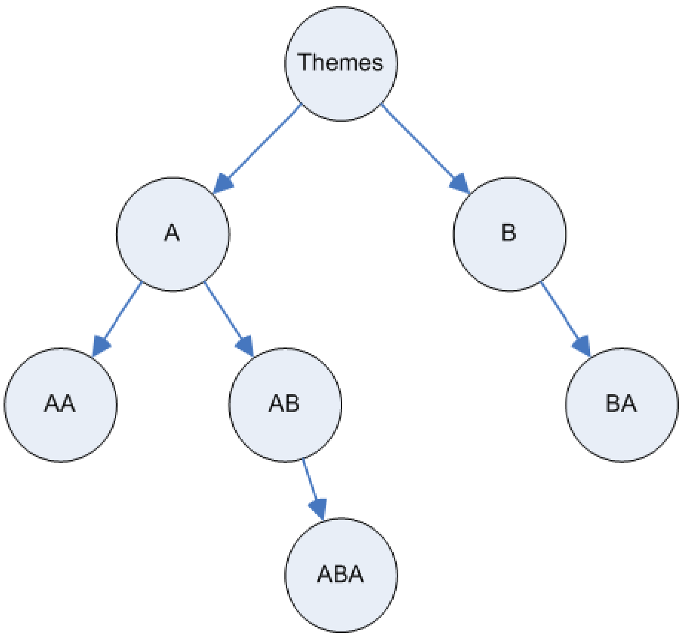


Figure 1 – An example of a theme hierarchy

1.4.3. Information distribution in PubSub based MANETs

In PubSub networks the information producers are called publisher and the users of the information are called subscribers. When a subscriber wants to get information regarding a given topic he/she initiate a subscription of this topic from the publisher. When other subscribers want the same information they also initiate subscriptions from the publisher. The different subscription requests result in building multicast trees. Figure 2 shows the principle of building one such multicast tree.

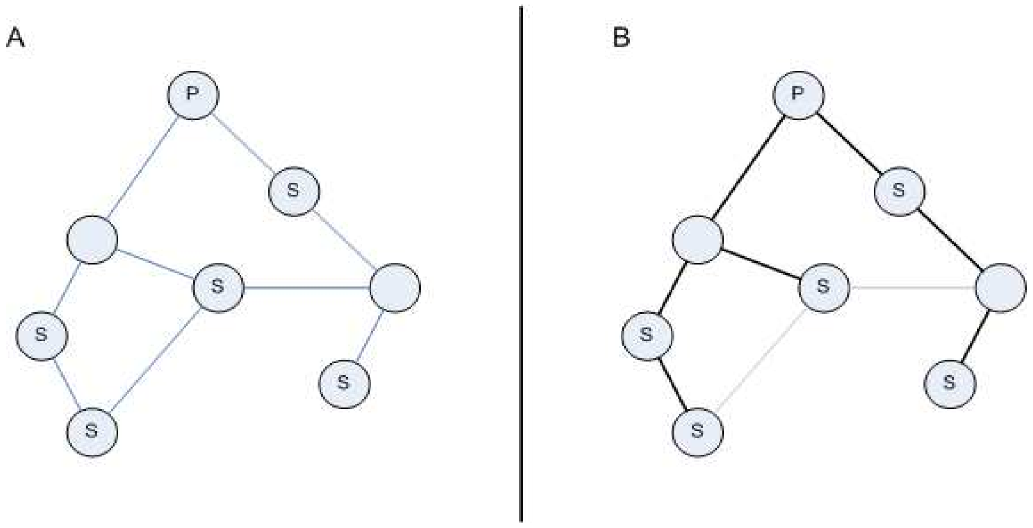


Figure 2 – A shows an example network topology with possible communication links between nodes. The P-node denotes the publisher, the S-nodes denote subscriber nodes, while other nodes are intermediate nodes that neither publish nor subscribe to information themselves. B show the resulting message distribution multicast tree in this example, where the thick lines indicate communication links used to distribute the information.

A PubSub communication model has been shown well suited for wired data-centric content-based routing (CBR). PubSub networks seem also to have some good qualities for use in MANETs [4]:

- Simplified protocols
- Reduced traffic used for router and mobility control
- High intrusion tolerance

In addition to these properties PubSub paradigm has similar characteristics to the type of communication which takes place at a tactical/mobile level. Radio uses broadcast to send information, where PubSub uses multicast/broadcast.

[13] has shown that PubSub communication models are applicable for highly dynamic network environments like MANETs. In their article they show three different algorithms specifically targeted for ad hoc networks; CBR, FT-CBR and RAFT-CBR. In addition to the protocol developed by [13], [4] has developed a sketch for another pubsub communication protocol suitable for CBR in MANETs.

1.5. Intrusion tolerance – what is it?

1.5.1. How intrusion tolerance networks differ from failing nodes.

Intrusion tolerance focuses on survivability of a network when it is under attack by an intelligent adversary. According to [16] there are a number of definitions of survivability. In this thesis we will use the same definition as [16] which deduced from [15] and [3]:

Survivability is the capability of a system to fulfill its mission in a timely manner, even in the presence of attacks or failures. Survivability goes beyond security and fault tolerance to focus on delivery of essential services, even when systems are penetrated or experience failures, and rapid recovery of full services when conditions improve. Unlike traditional security measures that require central control and administration, survivability addresses highly distributed, unbounded network environments that lack central control and unified security policies.

The Tree Rs: Resistance, Recognition, and Recovery *The focus of survivability is on delivery of essential services and preservation of essential assets. Essential services and assets are those system capabilities that are critical to fulfilling mission objectives. Survivability depends on the three key capabilities: resistance, recognition, and recovery. Resistance is the capability of a system to repel attacks. Recognition is the capability to detect attacks as they occur and to evaluate the extent of damage and compromise. Recovery, a hallmark of survivability, is the capability to maintain essential services and assets during attack, limit the extent of damage, and restore full services following attack.*

[17] argues the traditional view of computer security as a binary term that suggests that at any moment in time a system is either safe or compromised, ignores states where the system is recovering from a compromised state and aspects of maintaining services during and after an intrusion. A survivable system shall on the other hand

collectively accomplish its mission even under attack and despite active intrusions that effectively damage a significant portion of the system.

Fault tolerance relates to the statistical probability of an accidental fault or combination of faults. This definition will however not cover failures orchestrated by an intelligent adversary;

> *e.g. an analysis may determine that a simultaneous occurrence of three independent faults (f1, f2, and f3) will cause the system to fail. The probability that all three independent faults occur simultaneously is however extremely low. An intelligent adversary with knowledge of the system's internals can however orchestrate the simultaneous occurrence of these three faults and bring down the system. The system is fault tolerant, but it is not survivable because it will not continue to fulfill its mission during (and after) the attack.*

Most traditional research and practice in computer-system survivability uses a perilously narrow, security-based view of defense against computer intrusion. This narrow view is dangerously incomplete because it focuses almost exclusively on hardening a system to prevent a break-in or other malicious attacks, and very little on how to detect an intrusion or what to do once an intrusion has occurred or is under way. [3].

1.5.2. Types of intrusions

An intruder can do a number of different tasks to achieve its mission depending on the primary goal with the intrusion. These intrusions can be, but not limited to, jamming, introducing false information, impersonating other nodes, disruption of the underlying delivery system, stopping message forwarding, wiretapping and overloading the network.

1.5.2.1. Jamming

Jamming can be performed either by an intruder using the equipment available in the network or an extruder to continuously broadcast on the same radio spectrum used by the nodes in the network when they communicate with each other. This continuous broadcast makes the communication channel useless and thereby stopping the message flow in that area.

1.5.2.2. Introducing false information

An attacker which can become a part of a communication network can introduce false information to the message flow if there are no other mechanisms to validate the information. Such false information can have significant impact on how the users in the network behave if the information is being used in decision making processes.

1.5.2.3. Impersonating other nodes

By impersonating other nodes an attacker makes it harder for the other nodes to detect which information is false and which information is true. If an attacker has not been able to impersonate other nodes, he/she would have much more difficulties to successfully introduce false information, because other nodes would more easily detect the information from that source would not be trustworthy.

1.5.2.4. Distribution of the underlying delivery system

By impersonating other nodes the attacker might also try to disrupt the underlying message delivery system (the routing protocol) by manipulating the routing information used by the protocol. If these types of attacks are successful they can be catastrophic for any communication network as the network often will not forward information between nodes as expected.

1.5.2.5. Stop message forwarding

If an attacker stops message forwarding in the node he/she controls, the node will stop working as a router in the network making routes across this node unusable. If there are no other paths between the sender and the receiver using this node as an intermediate routing node, the attacker will succeed distrupting the message flow in the network.

1.5.2.6. Wiretapping

If there are no end to end security between the sender and the receiver of the information, the attacker will be able to gain access to all the information passing though the controlled node.

1.5.2.7. Overloading the network

In a network with limited resources an attacker can try overloading the network. To overload a network the attacker can e.g. start sending a large number of messages and thereby make the network using a lot of resources to transport the bogus messages. Another potential attack to overload the network is by broadcasting a lot of subscriptions and thereby trying overload the other nodes when they have to process the new subscription requests in addition to forwarding DATA messages corresponding to these themes.

1.5.3. Intrusion tolerance in PubSub based MANET

There exist only a few relevant articles on security in PubSub based MANETs which looks into intrusion tolerance of such networks. Most of the existing research on tolerance topologies for wireless ad-hoc networks is concentrated on finding multiple paths in the network with minimum energy usage. [17] argues that the tolerance ability of topologies is no equivalent to connectivity of multiply connected graph. [7] and [17] have conducted an analysis of intrusion tolerance in wireless sensor networks (WSN) which has some similarities to PubSub based MANETs. Other articles focus on security in routing protocols in MANETs used to create routing trees for information distribution based on traditional message distribution, which is not relevant for this study.

Work conducted regarding pubsub communication in MANET are primary concerned of how well the network can support communication in normal situations and to some extend how the network will react on failing nodes. We do not know about any work where the PubSub communication protocols have been exposed for an intelligent intruder, in a real world scenario this will present a significant threat to the survivability of any communication network if not appropriate countermeasures are in place.

1.5.4. Intrusion tolerance in WSN

Network topologies in WSN can be divided into two main categories; flat structure and hierarchical structure. In a hierarchical structure some nodes are selected as cluster heads, which collect and forward packets, while the others are ordinary nodes, which only collect data. In a flat structure all nodes are selected as heads. [17] describes a model for fault-tolerance and intrusion-tolerance in WSN with Bernoulli nodes. The study has however a weakness as it only focuses at intrusion tolerance in scenarios where an intruder makes the attacked node unavailable, e.g. creating failure situations.

In addition to introducing failing nodes, an intruder with sufficient resources might be able to perform many other types of attacks, including introducing false information in a network used for information distribution.

1.5.5. Intrusion tolerant PubSub Ad-hoc networks

In this thesis we show how the PubSub Ad-hoc protocol is vulnerable against some types of attacks produced by an intruder. All the attacks used in the analysis of the protocol are all attacks which are well doable in a real life scenario, and not just theoretical attacks.

At the end of the thesis we propose a solution to make the protocol more robust and immune to the different attacks studied in the thesis. By implementing those enhancements, the protocol can be made very robust against intrusion attacks, making it well suited for usage in hostile environments where the network is under attack by an intelligent adversary.

2. Detailed description of PubSub Ad-hoc networks

2.1. Description of principles in PubSub Ad-hoc networks

In PubSub Ad-hoc networks the information consumers (subscribers) subscribe to different information themes of interests from the information producers (publishers). Each subscriber will then inform the publisher of its interest, and request a subscription for the information. Using different subscription requests, the information is sent to different subscribers using multi-cast.

Information of available information themes, and who publishes those themes, is often made available to the subscribers in a directory service within the network. The subscribers can then use this directory service to look up from which it shall request the information of interest. In PubSub Ad-hoc networks the available information themes and the associated publishers can also be distributed to the subscribers using other methods than using a directory service; e.g. the publishers can broadcast the message themes they provide to all the other nodes in the network at a regular basis.

2.2. Different types of Publish/Subscribe Ad-hoc networks

[13] show in their article three different algorithms specifically targeted for ad hoc networks; CBR, FT-CBR and RAFT-CBR. In addition [4] has developed a sketch for another pubsub communication protocol suitable for CBR in MANETs.

2.2.1. CBR

In CBR every node periodically broadcast the contents of the advertisement and subscription tables to their neighbors. Entries in these tables are expired if they have not been refreshed for a specified time. This gives an efficient way of learning new routes and forgets old ones without having to explicitly delete routes that are no longer valid.

The CBR algorithm provided by [13] is based on existing distributed publish/subscribe algorithms in wired networks, but is extended to use soft state and use a new metric appropriate for ad hoc networks. In this protocol every publisher sends an advertisement message indicating the types of events it will publish. The advertisement message which is flooded and stored in an advertisement table (AdsTable) in all nodes contains the number of hops it has propagated. When a node wants to subscribe to an event type it will send a subscription message expressing its interest. Based on the information in the AdsTable the subscription is forwarded to the publisher. As subscription messages are routed toward the publisher each node stores the subscription in their subscription table (SubsTable). Events published by the publisher can then be sent along the reverse path of subscriptions to interested subscribers.

The CBR algorithm proposed by [13] enables subscribers to be anonymous in the system.

2.2.2. FT-CBR

In the CBR protocol it is possible that a node's set changes during a beaconing interval when a node moves, e.g. a node *n* sends event *e* to a neighboring subscriber *c*, if *c* has moved out of transmission range of *n*, then *n* has no way of finding *c*. In fault-tolerant CBR (FT-CBR) it is assumed that *c* is not moved far from *n* during the beaconing interval. Node *n* will then flood the message over a pre-determined number of hops to try to find *c*. In order to allow *n* to know whether or not *c* has properly received *e*, and thereby has to flood *e*, FT-CBR requires all events to be acknowledged hop by hop. [13]

2.2.3. RAFT-CBR

Even FT-CBR cannot guarantee that an event is distributed to all interested subscribers in a MANET because information about the set of subscribers is distributed over several nodes. Failure in one of these nodes can get information loss that cannot be reliably recovered. To guarantee reliable delivery of events RAFT-CBR requires the publisher to know all subscribers interested in event it publishes and that the publisher will not fail. It is also assumed that there are no prolonged partitions between the publisher and the subscriber; i.e. there is or will eventually be a routing path from publisher to subscriber.

In RAFT-CBR, publishers flood advertisements throughout the network as in CBR and FT-CBR. Subscribers unicast subscriptions to those publishers whose advertisements intersect the subscription interest. Unlike the CBR and FT-CBR subscriptions are not used to maintain the multicast tree from publisher to subscribers. Instead the publisher uses reliable xcast to disseminate events.

When a publisher multicasts an event e, e is tagged with a locally unique sequence number, allowing the (publisher id, sequence number) pair to form a globally unique identifier for e, and the addresses to the set of subscribers C, which subscribe to the event. When a node receives an event it queries the underlying routing protocol to determine the next hop to reach every $c \in C$. The node then forwards e to each of these next hops, modifying the set C in e to include only those subscribers that a particular next hop is responsible for forwarding to. This process is repeated until an event reaches all interested subscribers. When a subscriber receives an event it responds with an ACK if the event is an in-order event or else it responds with an NACK specifying the missing events. To achieve the reliability of the event distribution the publisher maintains a table that stores the sequence number of the last acknowledgement event for each subscriber.

If the publisher does not receive an ACK or NACK from a subscriber during a specified time it multicasts a PING to these subscribers in order to force the subscribers to respond with an ACK or NACK. [13]

2.2.4. CBR protocol described by Fongen

The different CBR protocols proposed by [13] are all based on the assumption that each theme originates from a simple source[1] and that the other participants are only subscribers. This limitation will however inflict great limitations in many scenarios. In scenarios where a publish/subscribe scheme is used for information exchange between nodes where several nodes can generate and hence publish information within the same topic, e.g. in a tactical operation one or more nodes are interested in opponent movements, in such a scenario several nodes can detect movements and hence publish this information to the other nodes in the network.

The new content based PubSub information exchange scheme described by [4] permit any node to publish information regarding any topic, even if another node already published information covering this topic. This property makes the protocol interesting for usage in communication networks to support decision making processes and (military-) operations.

[1] The CBR protocols proposed by [13] will support more than one source, but they require generation of dedicated distribution trees for each source. This requirement gives this approach clear scaling problems and limitations when the number of sources that publish information within the same theme increases.

2.3. CBR PubSub scheme described by Fongen

When a message is transmitted all nodes within radio coverage of the transmitting node will receive the message and retransmit this message if necessary. In this scheme each node describes which theme it wants to subscribe. This information, marked with a random number, is periodically transmitted to all neighboring nodes together with subscription information received from other neighbors. To prevent loops all subscription information is only valid for a limited time.

When a node receives a message the node transmits an acknowledge (ACK) message if it has any subscription to this message. In addition the node has to decide if the message shall be retransmitted to other nodes; there are other nodes, which subscribe to this message and have not yet received it. Given the fact that the node knows its neighbors' subscriptions and that all nodes transmit an ACK-message on reception of subscribed messages, the node makes the decision by waiting a given time and counting ACK-messages. If at least one neighboring node with subscription to this message does not send an ACK-message, the message shall be retransmitted. Chapter 2.3.2 shows an example on how messages will be distributed in the example network with subscriptions which will be described in detail in chapter 2.3.1.

2.3.1. Message subscription

In PubSub based message distribution protocols, the message subscription element plays a key role, since this feature is used to build the multicast trees, which are used to route the messages themselves from publisher(s) to subscriber(s).

An example network of 6 nodes, {N1, N2, … , N6}, where each node has a radio cover indicated by the dashed line surrounding each node, will be used to show how this protocol works.

In the following example all nodes routing tables are initially empty.

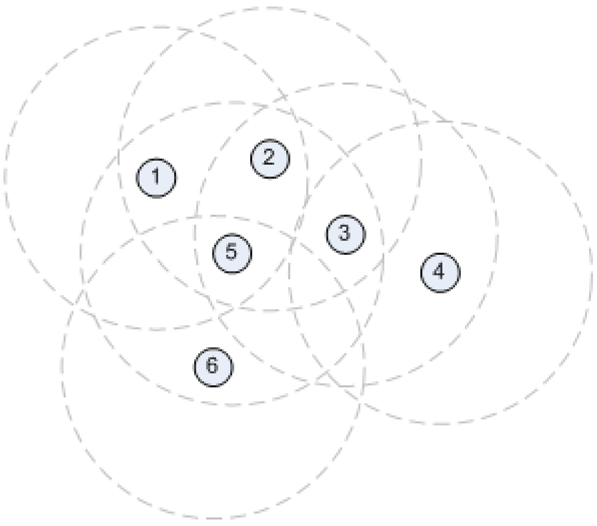


Figure 3 - Example network

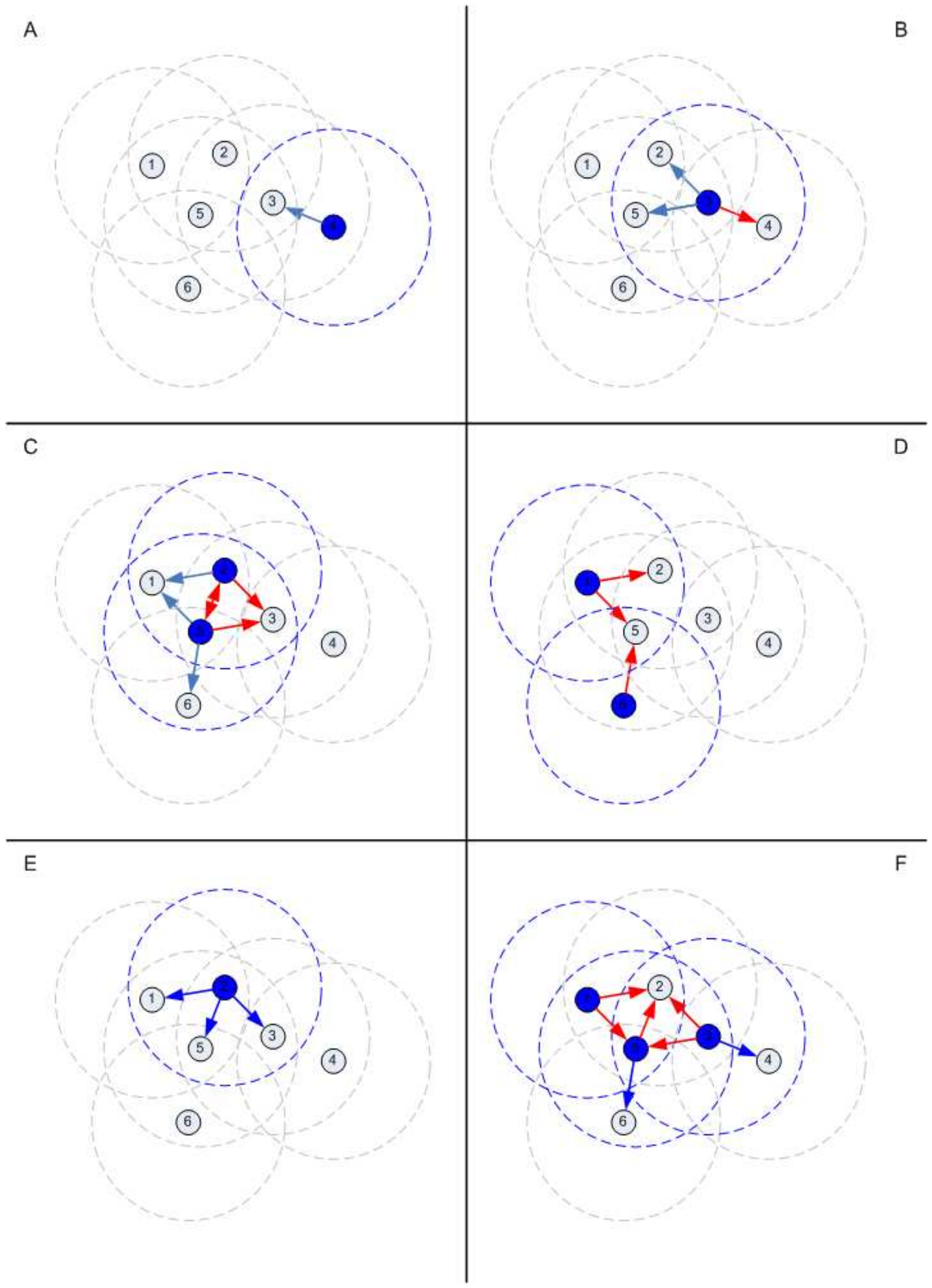


Figure 4 - Subscription in example network

Figure 4A: If N4 wants to subscribe to a theme, it will announce the subscription to all its neighbors, in this case N3. N3 confronts its routing table and check if it has seen this subscription before, meaning there exist a better route to the subscriber than via node 4. In this case this is a new subscription, so the routing table gets updated.

Figure 4B: Node 3 will now broadcast the subscription to all its neighbors, in this case N2, N4 and N5. N2, N4 and N5 will now confront their routing tables. N2 and N5 will update, while N4 already has this routing information in its own table and thereby does not do anything about the information.

Figure 4C: N2 will now broadcast the subscription to all its neighbors, in this case N1, N3 and N5. N1, N3 and N5 will now confront their routing table and update their routing table if this is a fresh subscription. N5 will also broadcast to all its neighbors N1, N2, N3 and N6, which in turn confront and eventually update their routing tables. N1 will receive the update from both N2 and N5 within the time interval, meaning there exist two routes to the subscriber which has initiated the subscription.

Figure 4D: N1 and N6 broadcast the subscription to their neighbors, but all receiving nodes (N2 and N5) treat it as an old subscription and will therefore not broadcast it further.

Figure 4E: Suppose N2 will also subscribe to the same message theme as the earlier subscription. N2 broadcast its subscription to all its neighbors, which confront and update their routing table since this is a new subscription despite they already know of subscription covering the theme (subscription ID is different)

Figure 4F: The new subscription from N2 will propagate through the network in the same manner as the subscription form N4.

Detailed subscription tables for each of the steps are provided in appendix A

2.3.2. Message distribution

Given the routing table established in the previous example.

Figure 5A: N6 broadcasts a message within the theme subscribed by N2 and N4.

Figure 5B: N5 checks its subscription table and sees it has subscriptions for this message theme. N5 broadcasts ACK-message for the message. N6 ignores the ACK-message because it has already sent the message. N1, N2 and N3 ignore the ACK-message because they have no subscriptions learned from N5 for the message theme.

Figure 5C: N5 broadcasts the message because it has not received ACK-messages from all the nodes which N5 has received subscriptions for the message theme.

Figure 5D: N6 ignores the message because it has already sent the message, and the message is thereby an old message. N1, N2 and N3 check their subscriptions table and see they all have subscriptions for the message. They all send ACK-messages for the message. N5 will ignore all the ACK-messages because it has already sent the message. N2 will ignore the ACK-message from N1 because it does not have any subscription from N1 for the theme. N1 receive the ACK-message from N2 and update its ACK-count for the message. N2 receive the ACK-message from N3 and update its ACK-count for the message. N3 receives the ACK-message from N2 and updates its ACK-count.

Figure 5E: N3 broadcasts the message, because the ACK-count has not reached the number of subscriptions for this theme. N1 and N2 will not forward the message because they have ACK-count the number of subscriptions for this theme.

Figure 5F: N4 broadcast ACK-message for the message received.

Detailed message distribution tables for each of the steps are provided in appendix B

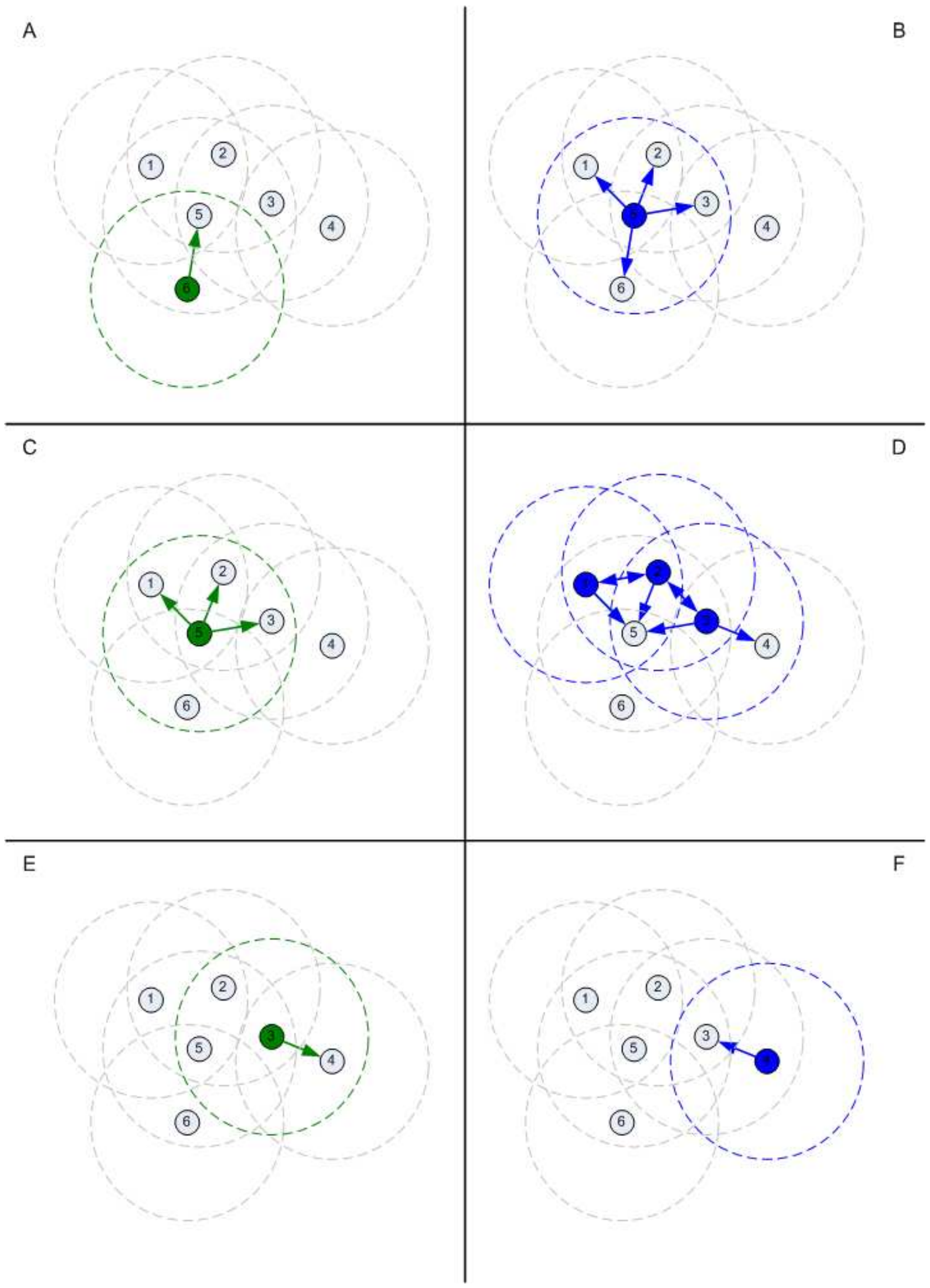


Figure 5

3. Intrusion tolerance in PubSub networks

If the protocol shall be intrusion tolerance, it has to achieve similar message delivery rates under attacks, as under normal situations with no attacking nodes. In the thesis we will therefore compare the delivery rate observed in the different attack scenarios with the baseline delivery rate for the protocol.

By analyzing the protocol we have found a number of different types of attacks an intruder can try to use in the network to achieve its goal;

- Jamming
- Stop forwarding subscriptions and DATA-messages
- Stop forwarding only DATA-messages
- Randomly introduce false ACK-messages
- Introduce false ACK-messages based on knowledge of the network
- Sending many ACK-messages to manipulate the ACK-count on other nodes
- Introduce false DATA-messages
- Wiretapping
- Overloading the network

Below we discuss how vulnerable the protocol is to the different types of attacks.

3.1. Jamming

In radio based communication networks, jamming is normally some sort of disturbing of the communication channel. It can be achieved by a powerful radio transmitter which transmits on the same frequency used by the communication network, and thereby introduces enough disturbances in the frequency to make it useless as a communication channel. In order to make the communication network robust against this type of attacks, the communication network has to relay on robust radio communication, which is handled by an underlying protocol and hence outside the scope of this analysis, or the protocol has to be able to reroute the information around the affected area. Because ad hoc protocols are designed to find the path between two nodes in the network, if there exist any, we assume the publish/subscribe ad hoc protocol will be able to find the alternative path if it exists.

In ad-hoc radio networks the effect of the jamming attack will therefore most likely be connected to how well the underlying radio equipment can handle jamming and not that much on how the routing protocol performs. The publish/subscribe ad hoc protocol will see the part of the network, which is under jamming attack, as if there was no path between the nodes on either side across the jammed part.

3.2. Stop forwarding subscriptions and DATA-messages

The network will basically see this attack identical to situations with radio silent nodes, because the network will not receive any subscriptions from this node, and hence this node will not be part of the communication path between the publisher and the subscriber.

3.3. Stop forwarding only DATA-messages

This attack is a bit different from the previous attack and radio silent nodes, because the node will forward subscription messages, and thereby try to become a transit node on the communication path between the publisher and the subscriber. If the attacker is on the only path between the publisher and the subscriber the attack will be successful. If there however exists another path between the publisher and the subscriber the attack is not likely to succeed, as the message will be forwarded along this path. In a real life communication network, formed by nodes acting as a group, we expect to see more than one communication path between nodes in the network. We therefore expect this attack to have little impact on the message delivery rate in the protocol.

3.4. False ACK-messages

Nodes using the PubSub protocol use the ACK-messages to decide if they shall forward messages or not. By sending false ACK-messages to fool the neighboring nodes to believe that all their neighbors have received the DATA-message, the attacker can stop them from forwarding the DATA-message to other nodes.

3.4.1. Randomly introduce false ACK-messages

To produce false ACK-messages the attacker has to produce ACK-messages which correspond to actual messages with correct theme and message id. In addition the attacker has to set the sender address to the address of the node the forwarding node has learned the subscription from. If we should choose all the parameters, message theme, message id and sender address randomly we get the following complexity of the task:

Given a optimal situation where the message theme is described in binary using only
16-bit, 16-bit message id number and 32-bit address we get:

$$complexity = NumberOfThemes * numberOfIds * numberOfAddressesInAddressSpace$$

$$complexity = 2^{16} * 2^{16} * 2^{32} = 2^{64}$$

This complexity is all too high to be doable in a real life network. We therefore tries to reduce the complexity by only produce false ACK-messages when the attacker receives a DATA-message or an ACK-message from another node. This way the attacker will know both the message theme and message id, and hence the complexity will be reduced to:

$complexity = numberOfAddressesInAddressSpace$ *e.g. in a 32-bit address space the complexity will be* 2^{32}

This complexity is still very high, so the attacker will not be able to cover all possible addresses in the address space in a real life scenario:

Given the same ideal situation as above where the message theme is described in binary using only 16-bits, 16-bits message id number and 32-bits address space.

Using a 2 Mbit/s communication link we get:

$$n = \left(\frac{\text{bandwith}}{\text{ACKpacketSize}}\right) * \text{timeBeforeForward} =$$

$$\left(\frac{2 * 2^{20}\,\text{bit/s}}{64\text{bit/packet}}\right) * 0.5\text{s} = 2^{15}\,\text{packets}$$

$$\text{cov erage} = \frac{n}{\text{numberOfAddressesInAddressSpace}} = \frac{2^{15}}{2^{32}} = \frac{1}{2^{17}} = 0{,}00076\%$$

As shown above the complexity involved in this attack is very high making it very little likely doable in a real life situation.

3.4.2. Introduce false ACK-messages based on knowledge of the network

This attack is identical to the previous attack, but instead of randomly generating false ACK-messages, the attack uses knowledge of the network to produce false ACK-messages only from nodes that are active in the network and thereby potential subscribers. The attacker builds the database of node addresses by listening to all the traffic passing through the node and collecting sender addresses.

In a network with *n* nodes the attacker will at most learn *n* addresses using this approach, which is satisfactory within what is doable in a real life scenario.

The attack is performed by letting the attacking node produce false ACK-messages for all the learned node addresses. Because the neighbor nodes are not able to distinct the false ACK-messages from the real ACK-messages received from other nodes, the attacker can make all nodes within his/her radio coverage believe all their neighbors have received the message, and they will thereby not forward the message.

3.4.3. Sending many ACK-messages to manipulate the ACK-count on other nodes

This attack is feasible because the nodes only count ACK-messages from subscribers, and do not keep track of which nodes they have already received an ACK-message from. This enables a node to send more than one ACK-message and making the receiving node count all of them.

The attack can be performed by letting the attacking node subscribe to all the top themes, and thereby subscribe to all message themes. The attacker will now be able to send ACK-messages for all messages generated in the network using his own source address.

3.5. Introduce false DATA-messages

By introducing false information into the network the attacker can manipulate the information used by other nodes to make decisions. If an attacker in addition can impersonate other nodes it will be harder for the other nodes to detect which information is false and which information is true because it will be difficult to detect which sources are trustworthy and which are not.

The attack is feasible because the protocol does not give the nodes in the network a method to verify the source of a message, and thereby distinct the false messages from real messages.

3.6. Wiretapping

If there is no end to end security between the sender and the receiver of the information, an attacker will be able to gain access to all the information passing though the controlled node. The PubSub ad hoc protocol is designed to be a protocol to efficiently distribute messages between nodes, and is not designed to provide any end to end security for the data sent over the protocol so we have therefore not done any analysis to cover this topic.

3.7. Overloading the network

In attacks created to overload the network, the attacker will try to use as much resources as possible in the network for his activity, and thereby limit the network ability to perform its mission. These attacks will mostly focus on other parts of the communication system (i.e. computation power, available memory, available bandwidth) and not that much on the protocol itself. We have therefore not done any analysis to cover this topic in the thesis.

4. Description of the experiment

Building a full scale real life testing facility would be very difficult within this project boundary because the cost and complexity involved in such facilities would be too high. We will therefore use simulations of the problem, which are more easily accessible and require fewer resources than a real world experiment, to provide us with the required data. In this thesis we used the NS-2 simulator [12] to perform the required simulations.

In simulated environments we can keep control over all variables that may influence with the result, even variables we are not even aware of which may be important to the performance of the network in a real world environment. In our project we are primary interested in how the protocol used for message distribution in PubSub MANET is performing when it is under different types of attacks from an intelligent intruder in the network, and not that much how the protocol performs under influence by other physical phenomena that would be a natural focus when simulating lower level protocols.

4.1. Description of how the experiment will be conducted

4.1.1. Equipment used

The experiments conducted in this thesis are conducted using the open source NS2 network simulator running on a standard PC. Since there is no known implementation of the protocol in NS2 we had to implement the protocol as an extension to the NS2 simulator as described in 4.2.

4.1.2. Performing the simulation

In our experiment we will create several simulations to simulate different scenarios in order to give answers to the research questions in section 1.1. The different simulated scenarios will be exposed to different types of attacks from an attacker with different levels of resources available, ranging from simple jamming attacks to attacks where the attacker has full control over one or more nodes in the network. The attacker might also be able to manipulate the software for nodes he has overtaken. The different simulations will give us required data to answer research question 2 and 3. By analyzing the results from research question 2 and 3 we will be able to give an answer to the research question 1.

4.2. Implementation of the protocol

4.2.1. Establish a detailed protocol design based on the sketch

Because this project is based on analysis of a sketch for a protocol and not an already implemented and working protocol, we first have to develop a detailed protocol design for a working protocol based on the sketch.

4.2.1.1. Subscription handling

When a node wants to initiate a subscription for a new message theme it will inform its neighbors by broadcasting a subscription message containing the message theme and a subscription id. Because subscriptions will age and be removed from the routing tables after a period of time, the subscribing node has to refresh the subscription by sending a new subscription message for the theme with the same id before the subscription ages out if it will continue to subscribe to the theme.

When a node receives a subscription message, it will check if it is an old subscription (subscription recently sent by the node), if it is a new subscription or an update of an already received subscription. The node has to ignore subscriptions it receives which it has recently sent because this will most likely be a neighbor node forwarding the subscription. If the nodes do not ignore these messages the subscriptions will never expire and thereby stay in the network forever, even after the subscribing node has stopped updating subscriptions for this theme. When a node receives a new subscription it will update its routing table before it will broadcast the subscription to all its neighbor nodes. If the received subscription is an update of an old subscription the node will refresh the timer before it broadcasts the subscription.

When a subscription is aged out, it will simply be removed from the node's routing table.

4.2.1.2. Data forwarding

In the protocol two message types are used to forward data, DATA and ACK. The DATA messages contain the actual data which are sent, while the ACK message is used to control the message flow and decide if it is necessary to forward the message as described in 2.3. In the protocol each node waits a defined period of time before the message is forwarded in order to give neighbor nodes time to send ACK-messages if they have received the message. Because the ACK-message can arrive before the actual DATA message the nodes also have to handle this situation by creating an ACK-count for the message based on the information in the ACK-message.

4.2.1.3. Problem with the protocol described in the sketch

The sketch [4] describes three different messages (packets) used by the protocol, SUBSCRIPTION, DATA and ACK.

In the sketch [4] the SUBSCRIPTION messages contain a message theme and an id to identify the subscription in the network. Because this message does not contain any information on who sends the packet, on a hop by hop basis receiving nodes will not know from where it has learned the subscription. This lack of knowledge makes the protocol not working as intended, because there are situations where the messages will not be forwarded to the subscribing node, as illustrated in Figure 6.

To solve this problem we introduce a sender id, which is updated on a hop to hop basis. This information is sufficient to give receiving nodes enough information to solve this problem, as shown in Figure 7.

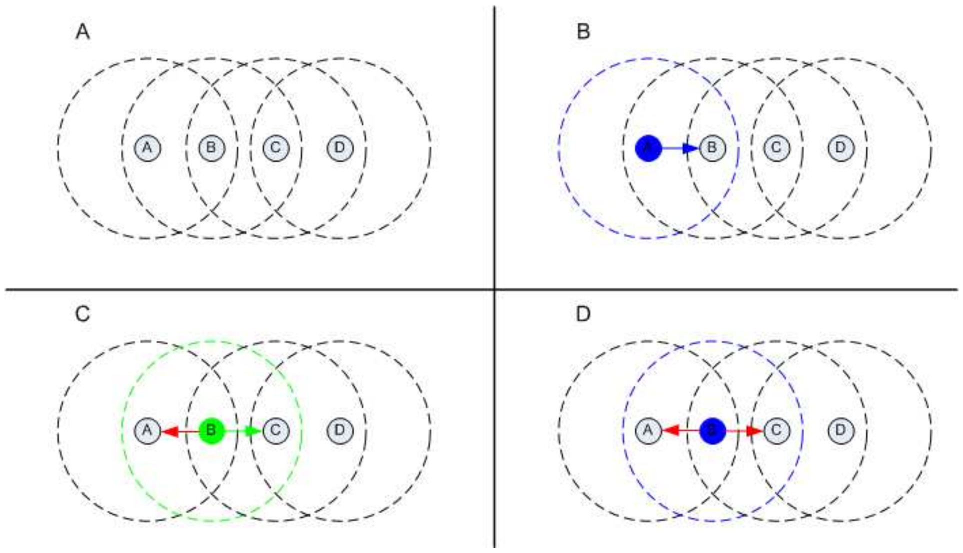


Figure 6
A – Given nodes A, B, C and D where D subscribes to a message theme
B – Node A generates a message in the message theme subscribed by D, A broadcasts the messages to all neighbouring nodes
C – Node B see it has subscriptions for the theme, and thereby broadcasts ACK for the message to A and C. A ignores the ACK because it has already sent the message, C stores the ACK in its

ACK-count for this message
D – Node B forward the message because it has not received ACK-messages making the ACK-count for this message equal or higher than number of subscriptions for this theme. Node A will again ignore the message because it has already forwarded the message. Node C will also ignore the message because its ACK-count is equal to the number of subscriptions for this theme, based on the ACK-message received from B. D which is the node that subscribes to this theme will thereby never receive the message

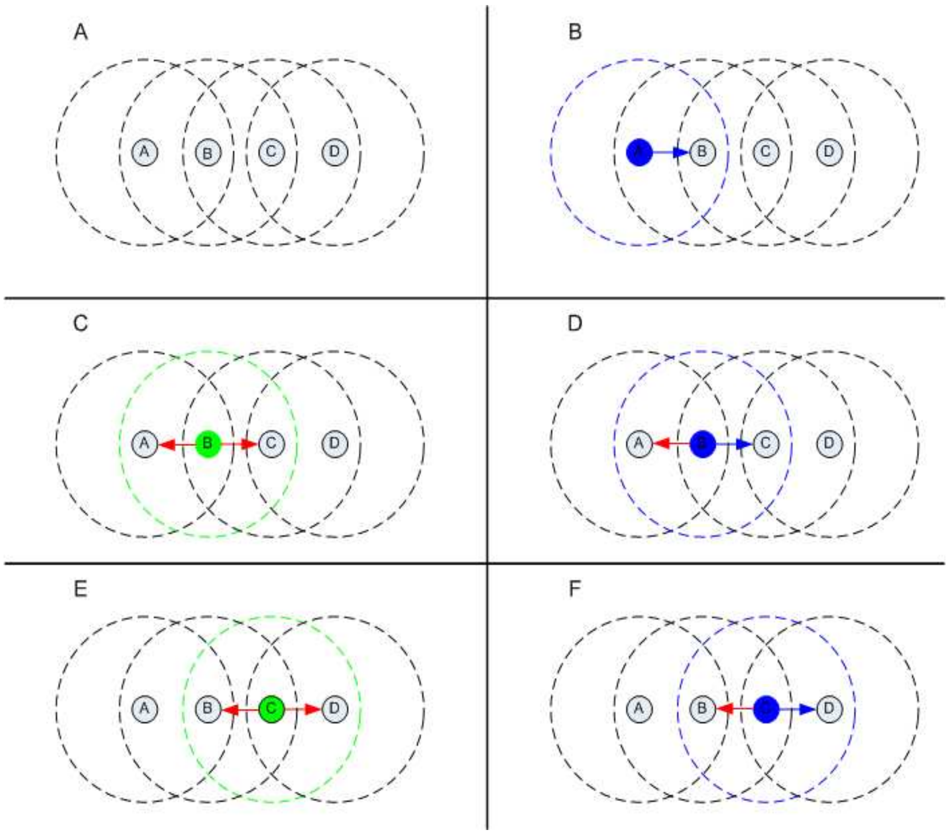


Figure 7
A – Given nodes A, B, C and D where D subscribes to a message theme
B – Node A generates a message in the message theme subscribed by D, A broadcasts the messages to all neighbouring nodes
C – Node B sees it has subscriptions for the theme, and thereby broadcasts ACK for the message to A and C. A ignores the ACK because it has already sent the message, C ignores the ACK because it has not any subscriptions learned from B for this message theme
D – Node B forwards the message because it has not received ACK-messages making the ACK-count for this message equal or higher than the number of subscriptions for this theme. Node A will again ignore the message because it has already forwarded the message. Node C will store the message because it has subscriptions for the message theme and the ACK-count for this message is not higher or equal to the number of subscriptions for the theme
E – Node C broadcasts ACK message to B and D. B ignores the ACK because it has already sent the message, D ignores the ACK because it dos not have any subscriptions learned from B for this message theme
F – Node C broadcasts the message because it has not received ACK-messages making the ACK-count for this message equal or higher than number of subscriptions for this theme. D receives the message, and the protocol has successfully delivered the message to the recipient

4.2.1.4. Description of the packets used by the protocol

The protocol uses three different packets to handle subscriptions and forward data. Bellow we will describe what each packet contains and what the information is used for.

SUBSCRIPTION

- Message theme – Used to describe the theme of the data the node wants to subscribe
- Subscription ID – Used to identify the subscription
- Sender address – The address of the node sending the packet, when a node forwards a subscription it will update this address with its own address.

DATA

- Message theme – Used to describe the theme of the data contained in the message
- Message ID – Used to identify the message
- Data – The actual data sent
- Sender address – The address of the node originating the message

ACK

- Message theme – Used to describe the theme of the data contained in the corresponding DATA message
- Message ID – Used to identify the corresponding DATA message
- Sender address – The address of the node sending the packet

4.2.2. Implementing the PubSub protocol in the simulator

Before we can do any simulation of the Fongen PubSub message distribution protocol it has to be implemented in the simulator. Since there are no known implementations of the Fongen PubSub message distribution protocol used in this project for the NS-2 simulator we had to develop an extension to the simulator ourselves. The simulator requires new protocol to be written in C++ followed by a recompilation of the entire simulator packet to make them available for use in simulations.

Full source code for the Fongen PubSub protocol is available in Appendix C

4.2.3. Creating scenarios

To create realistic simulations we have to create scenarios to reflect the real world environment we are simulating. In our simulations however, we will use a basic environment to keep the simulations as clean as possible to be able to study the desired properties of the protocol without being influenced by other components.

Manipulating the density of nodes (the average distance between two neighbor nodes) in the network we simulate will give us data to answer research question 2. By performing different simulation where we change the topology of the network we will achieve data to answer research question 3.

4.3. Description of how we will run the simulations

Each of the scenarios will be repeated a number of times with different input parameters to generate representative datasets. The different input data are achieved by changing the location of each node within the simulated area, or by changing the node acting as an intruder.

Some of the simulator parameters however will be kept the same for all the simulations. These are:

- Radio range is set to 200 meters

- We will be using a flat grid as the simulation environment. We are interested in how the protocol will react on different attacks from an intruder, and not how the protocol will be affected by the environment
- The nodes will not move around, since this will make it harder to determinate if changed delivery rate is caused by the attacker or temporary formation of communication islands
- When we simulate attacks, we are using only one node as attacker node in each simulation run

The NS-2 simulator uses a TCL-script to control the behavior of the simulator. To perform the simulations we therefore have to create the TCL-scripts for each of the simulations we will run. To create the necessary scripts we have developed a program which creates TCL-scripts with random node positions within the simulation grid. The program will also create random message subscriptions and creation of new messages in the network. The source code for the program can be found in Appendix D.

To create the other simulation scenarios where we introduce different types of attacks we are using modified variants of the baseline scripts created with the simulation creation program.

All the scripts used to run the simulations can be found in Appendix E

4.4. Establishing a baseline

Because we are analyzing a new protocol with no known studies to show how well the protocol distributes messages in the network, we first have to establish a baseline for the protocol. The baseline we establish here will be used as reference when we introduce different intrusion attacks. The design criteria for the protocol require the protocol to handle nodes that enter radio silence mode, e.g. they will not forward any subscriptions, data messages or respond with ACK messages.

4.4.1. Normal situation

To establish a baseline for normal situations, we run several simulations where we change the number of nodes, how they are distributed in the simulation area and using different size and shape of the simulation area. In the simulations we used a random generator to randomly distribute the nodes in the area, and the same random generator to randomly generate message subscriptions and message generation. This data was then fed into the simulator as input data.

Area size	***Number of nodes***	***Average***	***Mean***	***Standard derivation***
500 x 500 m	10	0.8633	0.9085	0.1240
500 x 500 m	25	0.9978	0.9980	0.0011
250 x 1000 m	25	0.9978	0.9986	0.0018
750 x 750 m	25	0.9509	0.9783	0.0630
750 x 750 m	50	0.9666	0.9654	0.0102

Table 1 – Delivery rate under normal situations. The datasets and calculations are given in Appendix F

As we can see from the data in Table 1, some of the simulations have significantly worse delivery rate and higher standard derivation than the others. If we look closer to the data in those situations we see that node density is too low to prevent communication islands to form, Figure 8. If we choose to use those formations in simulations where we introduce different types of attacks, we will get a lot of noise in the experiment data, because it will be very difficult to decide if the lower delivery rate is because of the attacker or because of the formation of communication islands. In the following experiments we will therefore not use those simulation configurations.

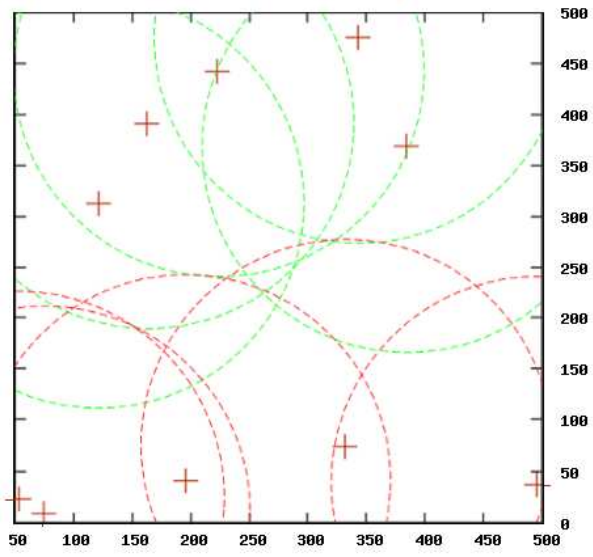


Figure 8 – Example of formation of communication islands. In this simulation the nodes location of each node are plotted in a 500 by 500 meter grid, with the radio coverage of each node indicated by the dotted lines. The formation of communications islands occurs because none of the nodes in the "green island" have radio coverage, which includes any node in the "red island" and vice versa

4.4.2. Radio silence nodes

To analyze how radio silence nodes affected the delivery rate of the protocol, we modified the same input data as used in the normal situations by randomly choosing nodes that entered radio silence mode in each simulation. In the experiment we used simulations with one radio silent node and simulations where approximately 10 % of the nodes were radio silent. The results of those simulations are shown in Table 2.

Area size	*Number of nodes*	*Number of silent nodes*	*Average*	*Mean*	*Standard derivation*
500 x 500 m	25	1	0.9974	0.9977	0.0009
500 x 500 m	25	3	0.9975	0.9977	0.0009
250 x 1000 m	25	1	0.9965	0.9973	0.0021
250 x 1000 m	25	3	0.9867	0.9971	0.0592
750 x 750 m	50	1	0.9642	0.9670	0.0138
750 x 750 m	50	5	0.9655	0.9691	0.0139

Table 2 - Delivery rate radio silent nodes

Using the observed results from the simulations and comparing theme with the observed results from the baseline simulations (Table 3), we can clearly see that there are almost no changes in the delivery rate between the two simulated datasets.

Area size	*Number of nodes*	*Number of silent nodes*	*Baseline delivery rate*	*Delivery rate with radio silent nodes*	*Delivery rate change*
500 x 500 m	25	1	0.9978	0.9977	-0.0001
500 x 500 m	25	3	0.9978	0.9977	-0.0001
250 x 1000 m	25	1	0.9978	0.9973	-0.0005
250 x 1000 m	25	3	0.9978	0.9971	-0.0007
750 x 750 m	50	1	0.9666	0.9642	-0.0024
750 x 750 m	50	5	0.9666	0.9655	-0.0011

Table 3 - Observed change in delivery rate with introduction of radio silent nodes

4.5. Simulate different types of intrusion attacks

4.5.1. Creating attacker nodes

In order to simulate scenarios where the network is under attack by an intruder, we have to implement those functions in the simulator. In our experiment we have achieved this by extending the node design so that we could instruct any given node in the network to start behave as an intruder with different types of attacks.

4.5.2. Different types of intrusion in the protocol

By analyzing the protocol we have found a number of different types of attacks an intruder can launch against the protocol to disrupt the message delivery in the network;

- Stop forwarding subscriptions and DATA-messages
- Stop forwarding only DATA-messages

- Introduce false ACK-messages based on knowledge of the network
- Sending many ACK-messages to manipulate the ACK-count on other nodes
- Introduce false DATA-messages

Below we describe how the different attacks work and how they are implemented in the experiment.

4.5.2.1. Stop forwarding subscriptions and DATA-messages

The network will basically see this attack identical to situations with radio silent nodes, because the network will not receive any subscriptions from this node, and hence this node will not be part of the communication path between the publisher and the subscriber.

We will therefore not do any simulations using this type of attack.

4.5.2.2. Stop forwarding only DATA-messages

This attack is a bit different from the previous attack and radio silent nodes, because the node will forward subscription messages, and thereby try to become a transit node on the communication path between the publisher and the subscriber.

In order to simulate this type of attack we instruct the node controlled by the intruder to not forward any DATA-messages. In the simulations we choose the node controlled by the attacker randomly among the nodes in the network for each of the simulations. We will run each simulation several times where we choose different nodes to be controlled by the attacker to get a statistical significant result.

4.5.2.3. Introduce false ACK-messages based on knowledge of the network

The idea behind this attack is identical to the previous attack, but instead of randomly generating false ACK-messages, it uses knowledge of the network to produce false ACK-messages only from nodes that are active in the network and thereby potential subscribers. The attacker builds the database of node addresses by listening to all the traffic passing through the node and collecting sender addresses.

In a network with n nodes the attacker will at most learn n addresses using this approach, which is satisfactory within what is doable in a real life scenario.

In the simulations we choose the node controlled by the attacker randomly among the nodes in the network for each of the simulations. We will run each simulation scenario a number of times where we choose different nodes to be controlled by the attacker to get a statistically significant result.

4.5.2.4. Sending many ACK-messages to manipulate the ACK-count on other nodes

This attack is feasible because the nodes only count ACK-messages from subscribers, and do not keep track of which nodes they have already received an ACK-message from. This enables a node to send more than one ACK-message and making the receiving node count all of them.

The attack will be performed by letting the attacking node subscribe to all the top themes, and thereby subscribe to all message themes. The attacker will now be able to send ACK-messages for all messages generated in the network using his own source address. Again the attacker is chosen randomly among the nodes in the network for each simulation.

4.5.2.5. Introduce false DATA-messages

By introducing false information into the network the attacker can manipulate the information used by other nodes to make decisions. If an attacker in addition can impersonate other nodes it will be harder for the other nodes to detect which information is false and which information is true because it will be difficult to detect which sources are trustworthy and which are not.

In the simulations we will randomly introduce false messages for themes with subscriptions, and analyze how many of those are successfully delivered to the subscribers. We will also randomly change source address for the information sent during this attack to hide the identity of the attacker. The attacking node will again be chosen randomly among the nodes in the network for each simulation.

5. Analysis of the results observed during the simulations

5.1. Methodology used in the analysis

To analyze how well the protocol handle the different types of attacks described above we use statistical methods on the data collected from the simulations described in 4.5. In the analysis we especially focus on how well the network can deliver messages to all nodes subscribing to the corresponding message theme when the network is under attack by an intruder.

In the following paragraphs we present the results from the analysis as well as give an explanation of the observed behavior. The corresponding datasets and tables used to create the graphs are presented in appendix F.

For each of the attack types we will describe the message delivery rate observed when the network is under attack and compare it with the delivery rate observed in the baseline simulations in 4.4. In situations where we observe significant changes in the delivery rate we will in addition analyze how the network topology and location of the attacker influence the observed message delivery rate.

5.2. Stop forwarding DATA-messages

5.2.1. Observed results

In these simulations we used a subset of the simulation scenarios used in the baseline simulations, and introduced the stop forwarding DATA-messages attack described in 4.5.2.2, by letting a random node act as the intruder. The results from the simulations are presented in Table 4.

Area size	*Number of nodes*	*Average*	*Mean*	*Standard derivation*
500 x 500 m	25	0.9973	0.9977	0.0011
250 x 1000 m	25	0.9900	0.9969	0.0352
750 x 750 m	50	0.9629	0.9647	0.0120

Table 4 – Observed delivery rate when the network is under nodata forwarding attack

5.2.2. Change in observed delivery rate compared with baseline

To be able to give an estimate of how successful the attack was, we used the observed results from the simulations and compare them with the observed results during the baseline simulations in 4.4 (Table 5). This comparison shows that there are almost no changes in the delivery rates between the two simulated datasets giving this attack a very low success rate, and hence the protocol is very little affected by the attack.

Area size	*Number of nodes*	*Baseline delivery rate*	*Delivery rate observed under attack*	*Delivery rate change*
500 x 500 m	25	0.9978	0.9973	-0.0005
250 x 1000 m	25	0.9978	0.9900	-0.0078
750 x 750 m	50	0.9666	0.9629	-0.0037

Table 5 - Change in observed average delivery rate when the network is under nodata forwarding attack

5.3. Introduce false ACK-messages based on knowledge of the network

5.3.1. Observed results

In these simulations we used a subset of the simulation scenarios used in the baseline simulations, and introduced the false ACK-messages attack described in 4.5.2.3, by letting a random node act as the intruder. The results from the simulations are presented in Table 6.

Area size	*Number of nodes*	*Average*	*Mean*	*Standard derivation*
500 x 500 m	25	0.8782	0.90815	0.1031
250 x 1000 m	25	0.7815	0.78775	0.1347
750 x 750 m	50	0.9268	0.9324	0.0306

Table 6 – Observed delivery rate when under false ack attack

5.3.2. Change in observed delivery rate compared with baseline

To be able to give an estimate of how successful the attack was, we again use the observed results from the simulations and compare them with the observed results from the baseline simulations (Table 7). This comparison shows that there are some changes in the delivery rates between the two simulated datasets making this type of attack a partial success, and hence the protocol are vulnerable to this attack.

Area size	***Number of nodes***	***Baseline delivery rate***	***Delivery rate observed under attack***	***Delivery rate change***
500 x 500 m	25	0.9978	0.8782	-0.1196
250 x 1000 m	25	0.9978	0.7815	-0.2163
750 x 750 m	50	0.9666	0.9268	-0.0398

Table 7 - Change in observed average delivery rate when the network is under false ack attack

5.3.3. Differences in impact between attacking nodes

The observed delivery rates in Table 6 shows a high standard derivation in the results, indicating there are differences in the impact of the attack between the attacking nodes. By analyzing the histograms of the simulation data sets, the assumption of differences of delivery rate impact between the attacking nodes is verified, Figure 9 - Figure 11.

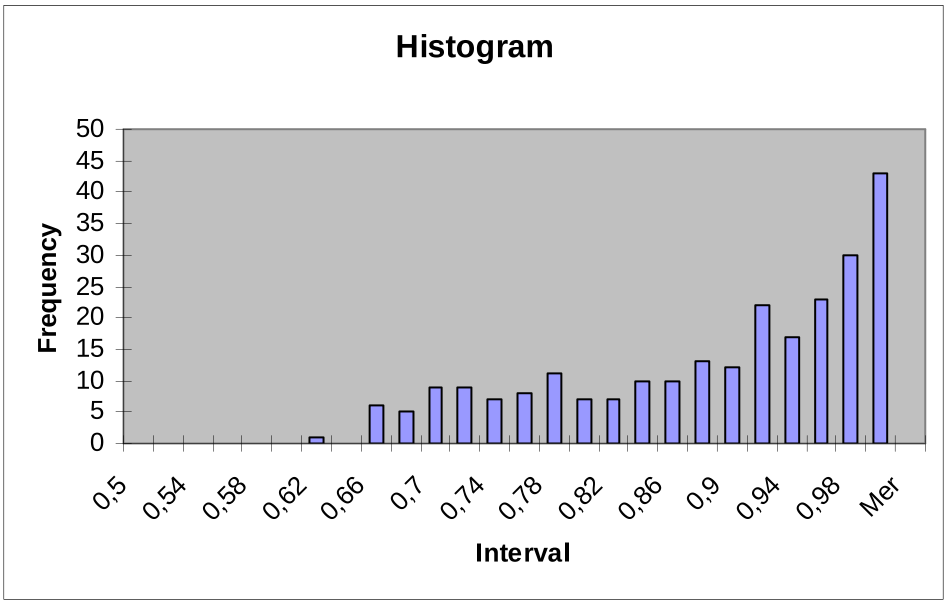


Figure 9 - False ACK-message attack using 25 nodes in a 500 x 500 m area

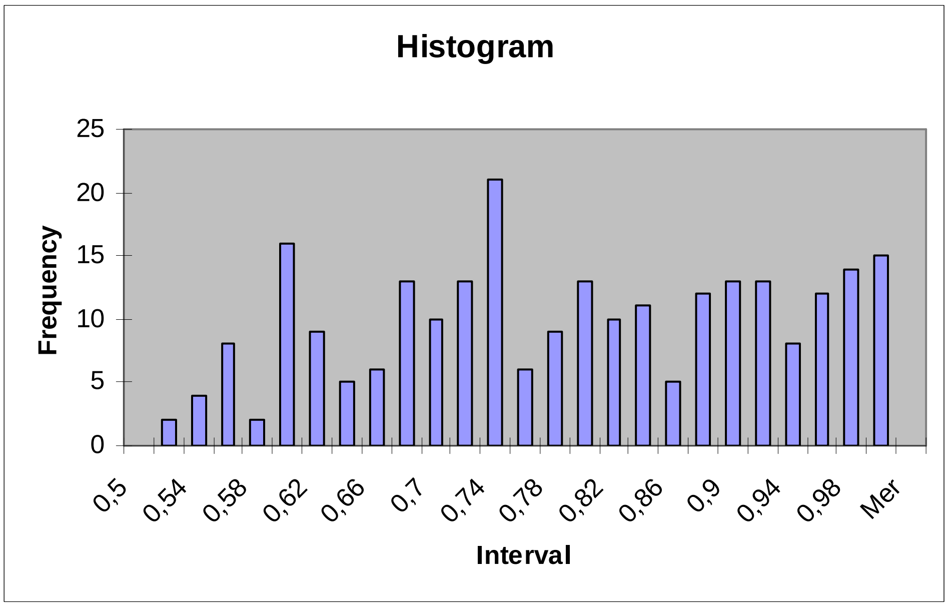


Figure 10 - False ACK-message attack using 25 nodes in a 250 x 1000 m area

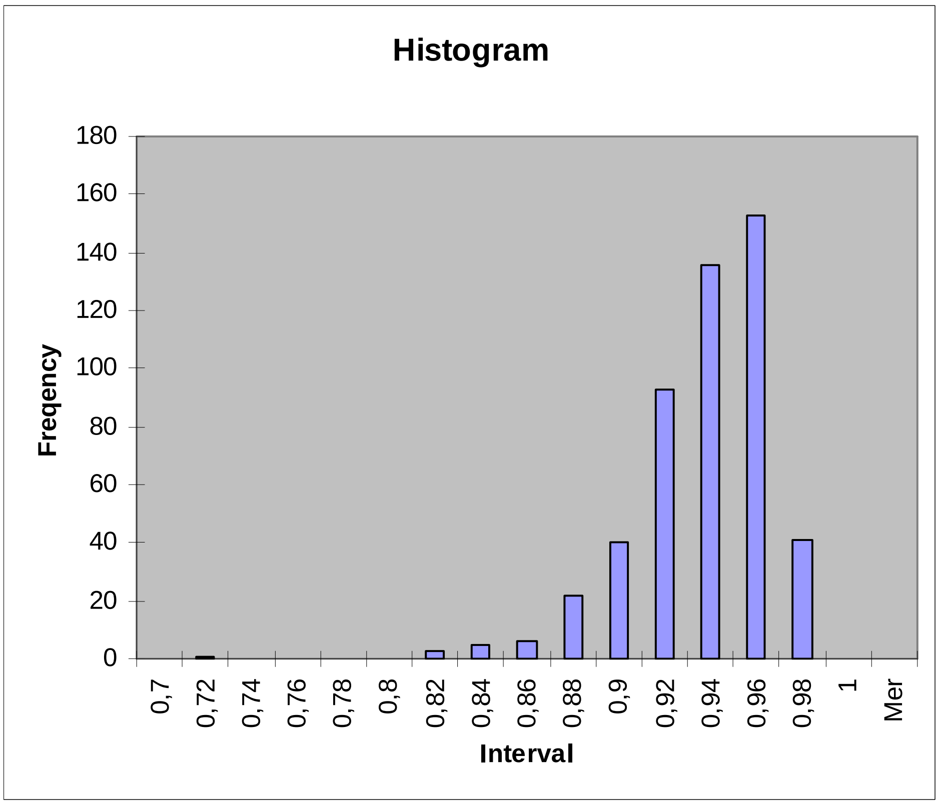


Figure 11 - False ACK-message attack using 50 nodes in a 750 x 750 m area

5.3.4. Attacker location analysis

In this analysis we take a closer look at why some attacker nodes have a much higher impact of the delivery rate than other nodes. We performed this analysis by looking at how the location of the attacker within the network influences how successful the attack becomes. To perform this analysis we have plotted the location of the attacker and give each node color based on how much that node was able to change the delivery rate of the network when it was acting as the attacker node, for each of the simulation scenarios above. Each set of simulation scenarios is plotted in the same plot to visualize any connection.

To analyze where the attacker should place himself within the simulation grid area to get the highest success rate, we simply plot each simulation within a scenario set using the same coordinates, as when we run the simulation, [Figure 12 - Figure 14].

The network center might however not be located at the center of the simulation grid. To analyze how the attacker location within the network influences the success rate, we place the center of the network in each simulation set as origin of the plot. Figure 15 - Figure 17 show the resulting plots from this analysis.

From the plots in Figure 12 and Figure 13 we observe that the attacker is more likely to have a higher impact on the delivery rate if he is located close to the center of the simulation area, than when he is located at the edges. For the larger simulation area, Figure 14, however, there is no such clear indication.

Figure 15 and Figure 16 shows even more clearly that the attacker is more likely to have a higher impact of the delivery rate if he is located close to the center of the network, than when he is located at the edges. Using this knowledge we assume the location within the network is more important than the location in the simulation area. In a defined area where the nodes are randomly placed within that area, the center of the network will most likely be close to the center of the simulation area.

For the larger simulation area however, Figure 17, there is again no such clear indication. To provide an explanation of this behavior we have analyzed these properties further in 5.3.5.

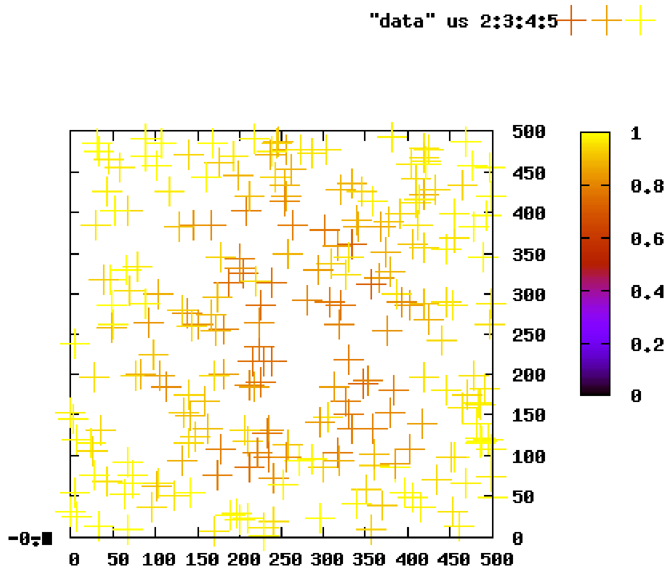


Figure 12 - Delivery rate impact false ACK-message attack based on location in the simulation scenario area, 25 nodes in 500 x 500 m

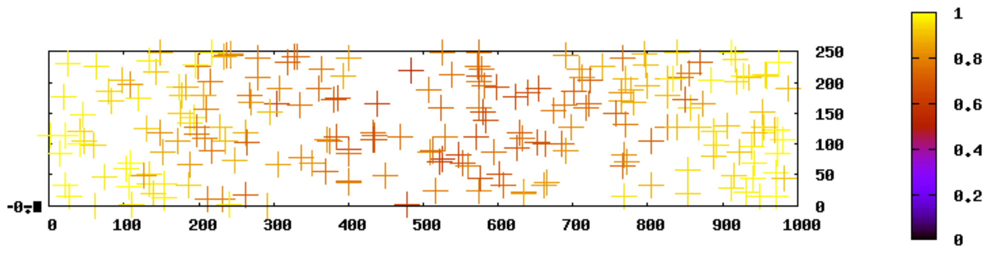


Figure 13 - Delivery rate impact false ACK-message attack based on location in the simulation scenario area, 25 nodes in 250 x 1000 m

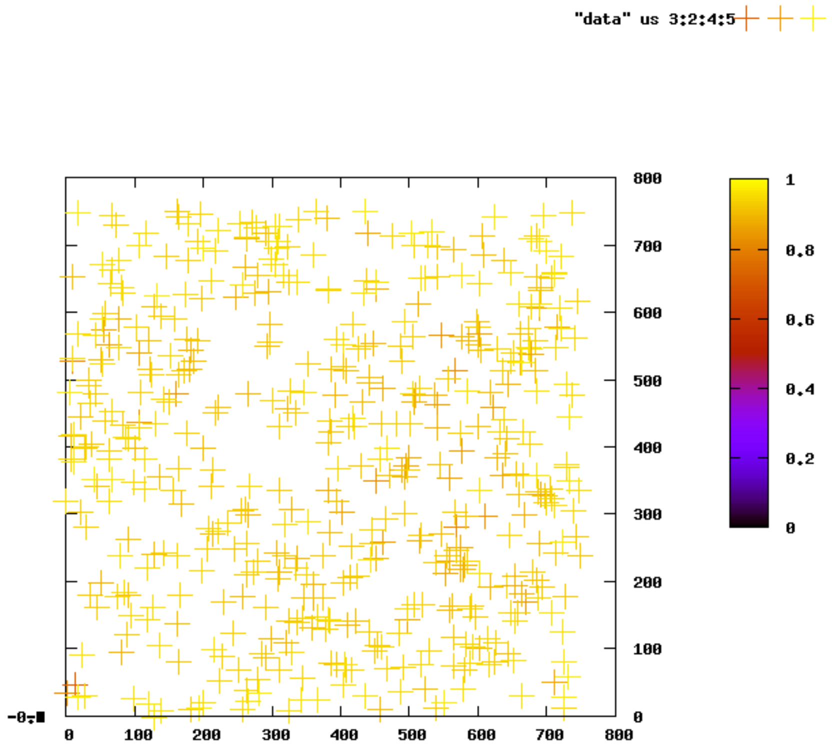


Figure 14 - Delivery rate impact false ACK-message attack based on location in the simulation scenario area, 50 nodes in 750 x 750 m

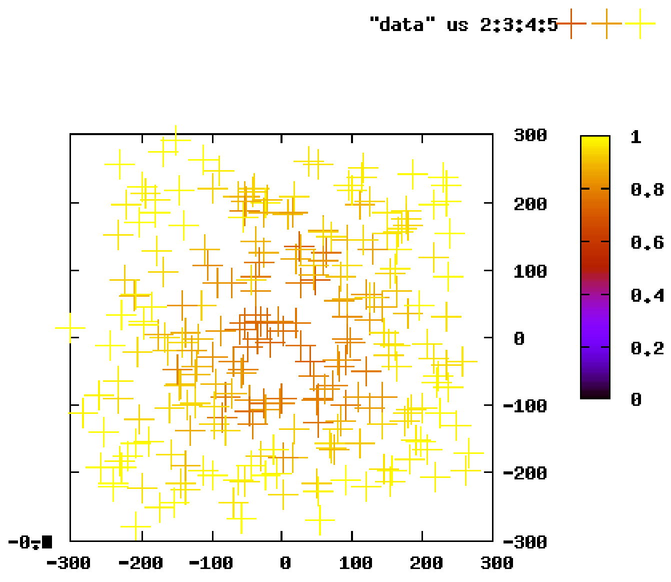


Figure 15 - Delivery rate when false ACK-message attack is introduced in the network for attacker location in the network, 25 nodes in 500 x 500 m

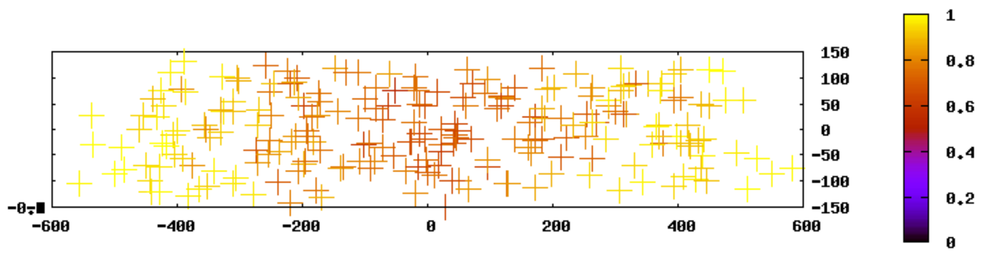


Figure 16 - Delivery rate when false ACK-message attack is introduced in the network for attacker location in the network, 25 nodes in 250 x 1000 m

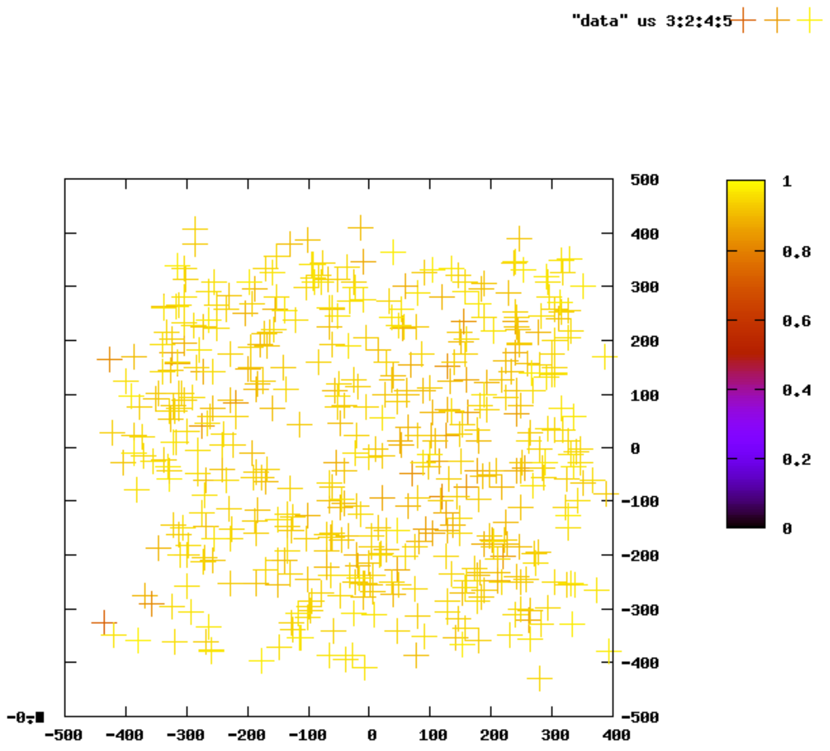


Figure 17 - Delivery rate when false ACK-message attack is introduced in the network for attacker location in the network, 50 nodes in 750 x 750 m

5.3.5. Impact of different network size and shape

As we have seen by the results in Table 7 there are quite large differences in how much the attack change the delivery rate in the network. The analysis in 5.3.4 gives a clear indication that the size and shape of the network area have significant influence in how well the network resists the attack.

Before we can give an analysis of the differences in delivery rate impact between the different scenarios we need to have an understanding of the radio coverage of the nodes in the network. Figure 18 - Figure 20, which show how the nodes are distributed in the simulation area and their radio coverage, will be used as an example to explain the observed differences. The data used to create those examples are taken from one of the simulations in each of the data sets.

In the smaller simulation areas, the attacker can affect all the communication paths between the two ends of the network when he is located close to the center of the network, and thereby cut off the communication between them, Figure 18. In the larger simulation area however the attacker is not able to efficiently cut off these communication paths (the nodes which are located in the radio range of the attacker will not forward messages because they believe all their subscribing neighbors have received the message) when he is located in the center of the network, as there are different communication paths around the affected nodes, Figure 19. The attacker might however be able to affect parts of the network as illustrated in Figure 20 where the red attacker cuts off the communication path between the blue nodes and the rest of the network.

Based on these observations, and the results from the simulations, we can assume the shape and size of the network will have significant influence on the success rate of the attacker using the false ACK-message attack. We can clearly see that smaller networks where the attacker can cover a large portion of the communication paths in the network are more vulnerable than larger network where there exist communication paths around areas which are affected by the attack. The attacker might however be able to disturb the message delivery rate in parts of the networks and even create communication islands also in larger network as shown in Figure 20.

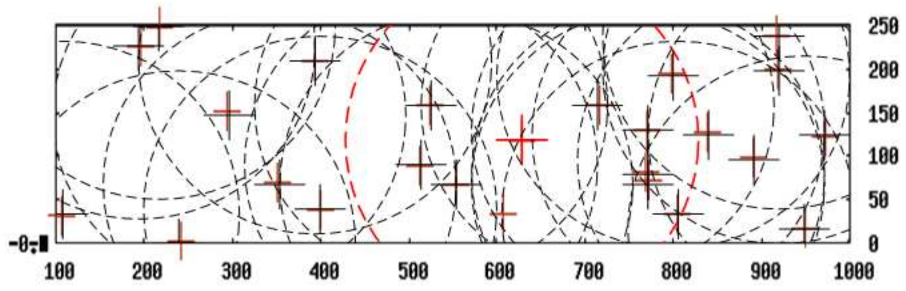


Figure 18 - Radio coverage for 25 nodes in a 1000 x 250 m area. In the figure the radio coverage of each node is represented with the dashed lines. The red dashed lines indicate the radio coverage of the attacker.

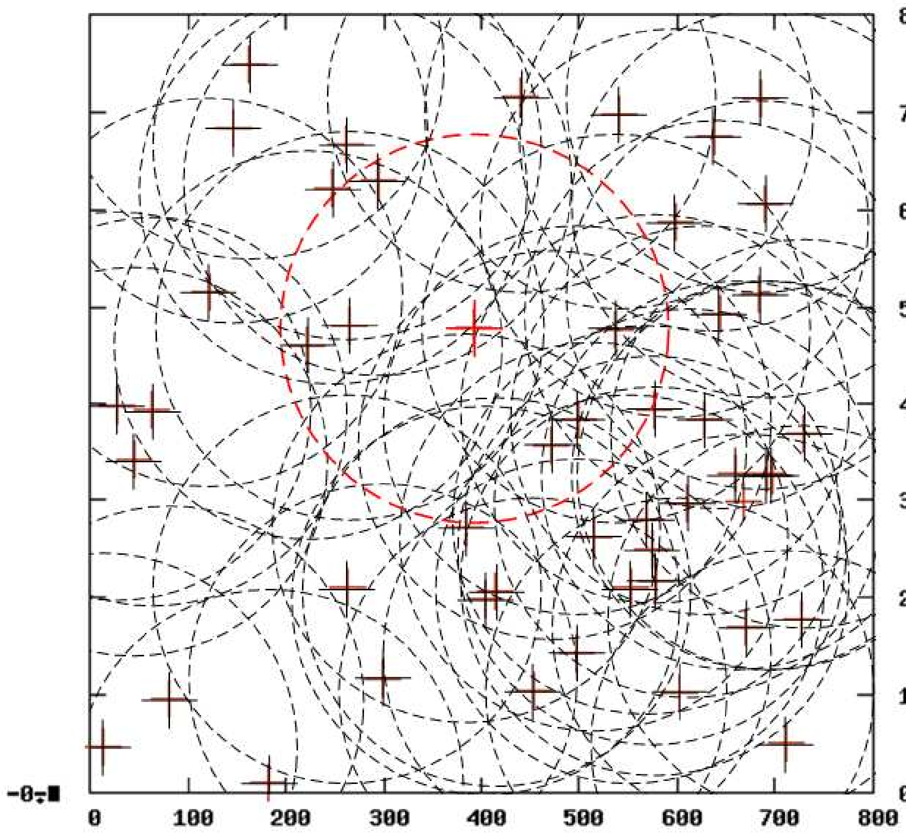


Figure 19 - Radio coverage for 50 nodes in a 750 x 750 m area. In the figure the radio coverage of each node is represented with the dashed lines. The red dashed lines indicate the radio coverage of the attacker.

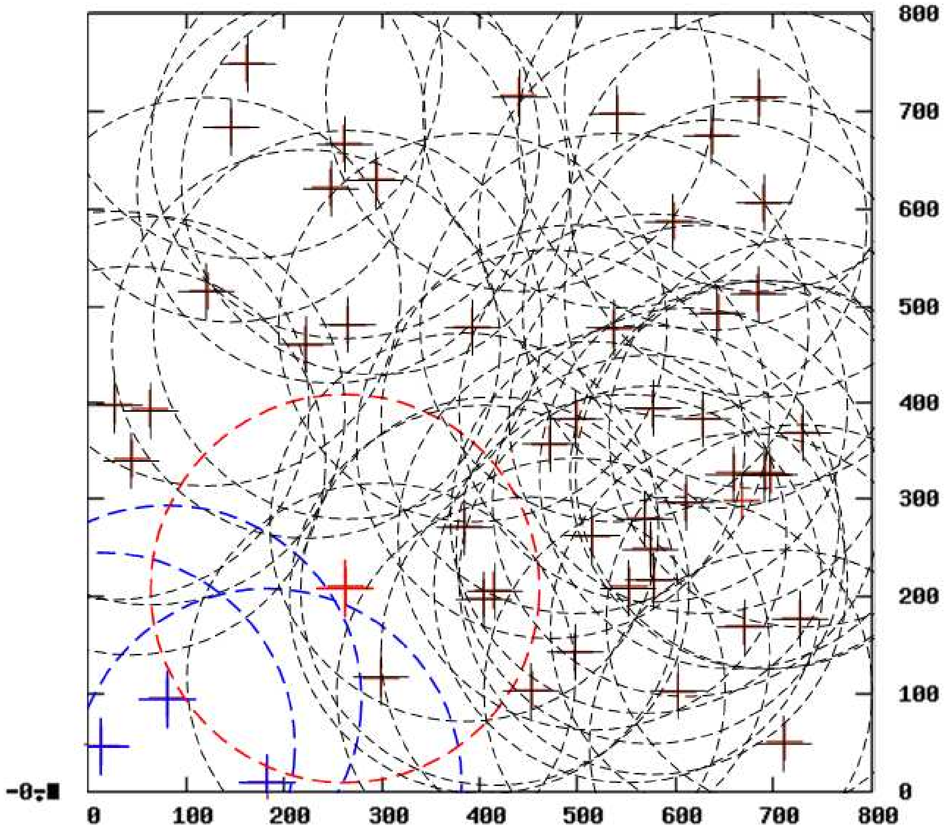


Figure 20 - Radio coverage for 50 nodes in a 750 x 750 m area. In the figure the radio coverage of each node is represented with the dashed lines. The red dashed lines indicate the radio coverage of the attacker. The blue nodes (and dashed lines) indicate the nodes, which are cut off from the rest of the network during this attack.

5.4. Sending many ACK-messages to manipulate the ACK-count on other nodes

5.4.1. Observed results

Area size	Number of nodes	Average	Mean	Standard derivation
500 x 500 m	25	0.8474	0.88135	0.1068
250 x 1000 m	25	0.7659	0.77375	0.1351

Table 8 - Observed delivery rate when the network is under attack where the attacker sends many ACK messages

5.4.2. Change in observed delivery rate compared with baseline and other attacks

To be able to give an estimate of how successful the attack was, we use the observed results from the simulations and compare them with the observed results from the baseline simulations [Table 7]. This comparison Table 9 shows the attack can change the delivery rates making this type of attack a partial success, and hence the protocol is vulnerable to this attack.

Area size	***Number of nodes***	***Baseline delivery rate***	***Delivery rate observed under attack***	***Delivery rate change***
500 x 500 m	25	0.9978	0.8474	-0.1504
250 x 1000 m	25	0.9978	0.7738	-0.2250

Table 9 - Change in observed average delivery rate when the network is under many ACK attack

When we analyze how successful the attack is reducing the delivery rate compared with the results from the false ACK-message, impersonating other nodes, attack in 5.3, we observe this attack is even more successful. To give an explanation of these observations we have to take a closer look at how the attacker gains sufficient knowledge required launching the attack.

In the false ACK-message, impersonating other nodes, attack the attacker need to learn the identity of the other nodes before he/she can start producing false ACK-messages. This gives a possible timeframe between when a node starts subscribing to a theme and when the attacker learns the identity of this node. Messages send during this timeframe will not be affected by the attack, because the attacker has no knowledge of the subscribing or intermediate node's identity. In the other attack, the attacker does not need this information as he/she already has all the required information available to launch the attack, when he/she receives the first ACK or DATA-message containing the correct message theme and message ID.

5.5. Introduce false DATA-messages

5.5.1. Ability to introduce false DATA-messages

In the simulations the attacker created false DATA-messages when he received a new subscription. The created DATA-message was given the same message theme as the subscription. To analyze the success of this attack we analyze the delivery rate of the false messages to subscribers which subscribe to the corresponding message theme. Table 10 show the observed delivery rate of the false messages.

Area size	***Number of nodes***	***Average***	***Mean***	***Standard derivation***
500 x 500 m	25	0.9956	0.9979	0.0075
250 x 1000 m	25	0.9939	0.99665	0.0092

Table 10 - Delivered false data messages to node subscribing to the corresponding theme

As shown by the observed results in Table 10 the attacker has a very high success rate to introduce false messages in the network, and the delivery rates are similar to the delivery rate we observed in the baseline simulations in Table 1.

5.6. Summary of findings

During the analysis of the behavior of the Fongen PubSub Ad Hoc protocol we have shown the protocol is vulnerable against some types of attacks from insiders in the network. All the observed weaknesses regarding DATA-message delivery rate are related to how the protocol handles and verifies ACK-messages, as the attacker can significantly influence the delivery rate of the protocol by impersonating other nodes or sending many ACK-messages himself when he subscribes to all the top-level themes.

The analysis has shown there are some significant differences between the different shapes and sizes of the network. Small networks, or other networks where the attacker can cover a large portion of the communication paths between different parts of the network is more vulnerable than larger networks where there will normally be other communication paths around the affected area.

During the analysis of the attacker location within the simulation grid area and the location within the network we have also shown differences between small and larger network regarding how the location of the attacker influence the success rate of the attack. In smaller network the attacker is more likely to become successful if he is located close to the center of the network, where he covers a large portion of the communication paths in the network. In larger network however there are no such clear indications, as there are often other communication paths around the area covered by the attacker.

We have also shown how an attacker can introduce false DATA-messages in the network, impersonating other nodes, making it hard for the other nodes in the network to distinct the false information from the correct information.

In the next chapter we propose enhancements to the protocol, which make the protocol more robust against these attacks, and thereby give the protocol high intrusion tolerance for attackers within the network.

6. Enhancement proposals for the protocol to make it more intrusion tolerant

As we have seen in the previous chapter, there is especially one type of attacks that is devastating for the message delivery rate in the protocol, as only one node which introduces false ACK-messages can stop a significant part of the messages sent in the network. In addition, the protocol is vulnerable to false DATA-messages since the attacker can introduce a lot of false information, and still hide his identity using other nodes' source addresses.

In this chapter we will try to introduce enhancements to the protocol to make those attacks less successful and hopefully stop the attack completely.

6.1. Using public key cryptology to sign messages

Since the attacker forges the sender address in both attacks, the primary focus is on how to prevent the attacker from being able to produce false messages, which appears valid for the other nodes in the network.

6.1.1. How public key cryptology can be used in the protocol to sign the messages

To prevent the attacker from being able to produce false messages where he impersonates other nodes we need a mechanism to validate the originator (sender) of the messages. This verification can e.g. be achieved by adding a digital signature scheme. Public key cryptology is a well documented method used to achieve such validation where the sender uses his private key to sign the message. The receivers can verify the sender and integrity of the message using the corresponding public key. Figure 21 shows an example of a signature scheme using a public key cryptosystem.

This signing and verification of messages makes it harder, it not impossible, for the attacker to create false ACK-messages and thereby introduce false DATA-messages impersonating other nodes. To ensure the verification of the messages, the nodes are required to be able to verify the owner of the key-pair used in the signing process.

In public key cryptology system it is common to use certificates to provide proof of ownership for a given public key. In order to verify the authenticity of the certificate, the certificates are signed by a trusted third party, using the trusted third party's private key. The nodes can now use the public key belonging to the trusted third party, which is preloaded into the nodes in the network, to verify the ownership of a given key pair.

In smaller networks it is also possible to preload the identity and corresponding public key for the nodes in the network offline into each of the node before they start communicating.

Given node *A* and *B*, message *M* containing both the data and the source address of the sender, *A*'s private key A_{priv}, *A*'s public key A_{pub}, a hash function *H* and a random number *nonce*, we get the following signature and verification scheme:

If *A* wants to send a verifiable message *M*, we get:

1. *A* generate a random number *nonce* to ensure freshness.
2. *A* generates *hashvalue* = $H(M,nonce)$
3. *A* encrypt the *hashvalue* using his/hers private key A_{priv} and send this and the *nonce* as the signature of the message:
   $signature = \left(hashvalue^{A_{priv}}, nonce\right)$
4. *A* send (*M,signature*)

Node *B* can now use *A*'s public key to verify the message:

1. *B* generates $hashvalue_B = H(M,nonce)$
2. *B* decrypt the encrypted *hashvalue* received in the signature using *A*'s public key: $hashvalue_{recv} = \left(hashvalue^{A_{priv}}\right)^{A_{pub}}$
3. The message is verified if and only if $hashvalue_B = hashvalue_{recv}$ since only *A* is able to create the encrypted version of the *hashvalue* using the corresponding private key to A_{pub}.

If an attacker tires to generate a false message impersonating *A*, the signature verification will fail as the attacker is not able to create the correct encrypted version of the hashvalue.

Figure 21 - Example of signing and verification of messages using public key cryptosystem

6.1.2. Weaknesses in this enhancement

The asymmetric cryptology scheme will give good protection against false data being introduced into the network [Figure 21], as it provides good protection against attacks where the attacker impersonates other nodes. A digital signature scheme will however not stop the attack in 4.5.2.4 where the attacker is using his own source address, as the signature verification will validate all the messages send by the attacker.

The enhancement does not come for free as the resources required to handle each message increase. In Ad Hoc networks the nodes are normally battery operated and equipped with limited computational and memory resources to keep the power consummation low, this increase can influence the overall performance of the communication system.

The signing of messages, especially subscription and ACK-messages, will also significantly increase the message size [Figure 22], requiring more bandwidth or time to send the messages. Increased bandwidth or sending time will again increase the power consumption of the nodes, which can reduce the overall performance of the network using battery operated nodes.

The distribution of the certificates used to verify the public key belonging to a node will in addition increase the complexity of the network design and increase the administrative workload required to operate the network. This is especially the truth where different organization units are going to cooperate in the same network, and there are no or little trust established between them before they start cooperating in the same network.

If we use the same binary message description method as used in 3.4.1, and a 160-bit message hash with a 32-bit random number used to provide freshness, we get:

$$new\_ACKmessage = ACKmessage + signature + random =$$

$$64bit + 160bit + 32bit = 256bit$$

$$Increased\_ACKsize = \frac{new\_ACKmessage}{ACKmessage} = \frac{256bit}{64bit} = 4 = 400\%$$

Figure 22 - Increae in message size with digital signatures, compared with orginal message size

6.2. Keeping track of ACK-messages

6.2.1. Concept of how to keep track of ACK-messages

To make the protocol more robust against attacks where the attacker sends many ACK-messages, we need to change how the nodes in the network count the number of subscriptions and received ACK-messages and keep track of which node has received the message and which has not. We can achieve this enhancement by changing the message count from a simple number of ACK-messages to a list of nodes. The list contains an entry for all the nodes, the node are missing ACK-messages for a given DATA-message.

In this scheme, the node will attach a list of all the nodes from which the node has learned subscriptions, which correspond to the message theme when it receives a DATA-message, or an ACK-message, with a previously unseen message ID. When the node receives an ACK-message for a given message ID, the sending node is removed from the list. The message will now be forwarded if there are still nodes in the list, when the message “wait timer” expires.

The necessary changes to the simulator implementation are described in appendix G.

6.2.2. Simulation using this enhancements

To analyze how successful the enhancement is, we have performed simulations with the enhanced protocol using the same input data as for the simulations in 4.5.2.4. The observed results from the simulations are shown in Table 11.

Area size	*Number of nodes*	*Average*	*Mean*	*Standard derivation*
500 x 500 m	25	0.9260	0.9259	0.0200
250 x 1000 m	25	0.9175	0.9222	0.0357

Table 11 - Observed delivery rate in enhanced protocol when the network is under attack where the attacker sends many ACK messages

6.2.3. Analysis of the results

To give an analysis of the enhancement we have compared the observed results in Table 11 with the baseline and the observed results for the attack in the original protocol. As we have shown in Table 12 the improvement reduces the effect of the attack and makes the protocol more tolerant against this type of attack.

Area size	*Number of nodes*	*Baseline delivery rate*	*Delivery rate when under attack with original protocol*	*Delivery rate when under attack with improved protocol*
500 x 500 m	25	0.9978	0.8474	0.9260
250 x 1000 m	25	0.9978	0.7738	0.9175

Table 12 - Analyzing effect of improvements in the enhanced protocol

6.3. Discussion of the new enhancements

Introducing public key cryptology to sign the messages will stop an attacker from introducing false messages in the system impersonating other nodes. Public key cryptology alone will however not stop all the attacks, as the attacker can launch the many ACK-message attack described in 4.5.2.4. To make the protocol more resistant against this type of attack we have proposed a solution where the nodes keep track of which node has send ACK-message for the different DATA-messages. As shown in this chapter, we can make the protocol very robust against the different attacks we have analyzed, by introducing a digital signature scheme and the enhancement to keep track of ACK-messages in 6.2.

The improvements will however introduce additional resource requirements as the computation complexity involved in sending and receiving the messages will increase. In addition, the introduction of digital signatures will increase the message size, especially for the subscription and ACK messages. The increased message size will require more bandwidth or time to send the messages, which will increase the power consumption of the nodes. The increase in power consumption can reduce the overall performance of the network where the nodes are battery operated.

The nodes will also require more memory to keep track of the ACK-message it has received in the enhanced protocol than in the original protocol where the nodes only has to keep track of a counter. The increase in memory requirements can however be kept small, and have little or no significant impact on modern communication devices since the increase will require only a few bytes of memory to keep track of each subscription per message.

Introduction of public key cryptology will require distribution and administration of certificates to prove the relation between the public key and the owner of this key. Any communication network used to communicate sensitive information will however need some sort of crypto mechanism to protect the data sent in the network, which will also require distribution and administration of crypto keys and/or certificates. We will therefore assume distribution and administration of certificates will not significantly increase the complexity and workload to administer the network

7. Conclusions

In many scenarios efficient information distribution is a key factor to conduct successful missions. It is also essential that this information is correct and kept secret, which requires high intrusion tolerance in networks used to distribute information used in decision making processes.

Publish/Subscribe (PubSub) Ad-hoc networks have been introduced as an effective mechanism to distribute information, where several nodes are interested in the same information. The information in PubSub networks is organized in information trees, where a given information theme covers its own theme as well as all sub themes within the theme. To get access to the information in the network consumers (subscribers) subscribe to different information themes of interests from the information producers (publishers), i.e. the subscribers inform the publisher(s) of their interest, and request a subscription for the information. The network uses the different subscription requests to send the information to different subscribers using multi-cast.

[13] show in their article three different algorithms specifically targeted for Ad Hoc networks; CBR, FT-CBR and RAFT-CBR. All those algorithms are primarily focused on scenarios where there are a few publishers, which publish information within a given theme. This assumption will however not hold in many scenarios, including tactical military scenarios, where all the nodes can be subscribers and publishers for all the different themes at the same time. To enable such communication [4] has developed a sketch for another PubSub communication protocol suitable for CBR in MANETs.

In this master thesis we have implemented the protocol described by [4] in the NS-2 simulator to analyze the protocol behavior when it is under different types of attacks. The analysis has shown some weaknesses in the protocol for some attack types where the attacker can impersonate other nodes or manipulate how the protocol keeps track on from which nodes it has received the messages and from which it has not. These attacks seems however to be most effective for small networks or other network topologies where the attacker can cover a large portion of the communication paths between different parts of the network.

To make the protocol more intrusion tolerant against these types of attacks we have provided proposals to solve the weaknesses we observed in the protocol. By introducing a digital signature scheme to verify the integrity and authenticity of the messages the attacker will not be able to impersonate other nodes and thereby effectively stop those types of attacks. In addition, we have changed how the nodes keep track of which nodes have received the DATA-messages, and thereby send ACK-messages, to stop the attack where the attacker can send many ACK-messages to manipulate the ACK-counters in the neighbor nodes.

By introducing those improvements to the protocol, we have shown that the protocol can be very robust against all the different attacks we have analyzed, making it very intrusion tolerant. The intrusion tolerant behavior makes the protocol a good candidate for a information distribution protocol in tactical military scenarios or other scenarios with many to many communication in hostile environments.

The protocol is however still in an early stage of development, so additional work has to be done before the protocol is ready to be implemented in real life scenarios. This additional work includes, but is not limited to, performance analysis, protocol optimization and finding a more effective message signature scheme for small messages.

8. Future work to be done

As shown in the thesis, the protocol described in [4] is a good candidate for an intrusion tolerant protocol used to distribute data/messages in networks where several nodes publish and subscribe to the same themes. There is however still a lot of work that has to be done before the protocol can be used in a live network where critical operations depend on it.

All the work done in this thesis is based on simulations of the protocol in a flat environment with relatively few stationary nodes. It will therefore be necessary to perform some real life experiments to verify the simulation results. It will also be necessary to perform analysis of how the network will operate in a more dynamic environment, where the different nodes move around.

The simulations in the thesis only analyze the basic behavior of the protocol as a baseline, and when it is under different types of attacks from an insider. Before the protocol can be implemented and used in a live network, it will be necessary to optimize the protocol to minimize the delay introduced by the network and how the protocol will handle subscriptions and forward messages.

As we have seen in the simulations, the protocol is vulnerable against certain types of attacks if we do not introduce message authentication to verify the sender of different messages. These enhancements will however significantly increase the message size for the control messages used in the network, and thereby most likely reduce the network throughput in busy networks. Because of this increased messages size, it will be necessary to perform analysis on how the enhancements will influence the network performance and eventually design a more effective signature scheme.

Bibliography

[1] **Altman, E. and Jiménez, T.** *NS Simulator for beginners*, Lecture notes 2003-2004, University de Los Andes, Mérida, Venezuela and ESSI, Sophia-Antipolis, France, http://www-sop.inria.fr/mistral/personnel/Eitan.Altman/COURS-NS/n3.pdf, last visited 2009-05-07.

[2] **R.Baldoni, R.Beraldi, G.Cugalo, M.Migliavacca and L.Querzoni.** *Structure-less Content-Based Routing in Mobile Ad Hoc Networks*, 2005 IEEE

[3] **Ellison, R. J., et al.** *Survivable Network Systems: An Emerging Discipline.* Software Engineering Institute, Carnegie Mellon University. Pittsburgh, PA 15213, USA : Software Engineering Institute, Carnegie Mellon University, 1997. s. 48, Technical report. CMU/SEI-97-TR-013.

[4] **Fongen, Anders.** Pubsub distribusjon i manet - skisse. Oslo : Norwegian Defence Rescearch Establishment, 2007.

[5] **Gollmann, Dieter.** *Computer Security*. John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2004, ISBN 0 471 97844 2, Chapter 12

[6] **Huang, Jiun-Long og Chen, Ming-Syan**. *IEEE On the Effect of Group Mobility to Data Replication in Ad Hoc Networks*. 5, May 2006, IEEE Transactions on mobile computing, Vol. 5.

[7] **Jing Deng, Richard Han and Shivakant Mishra.** *Intrusion Tolerance and Anti-Traffic Analysis Strategies For Wireless Sensor Networks*, Porceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)

[8] **Jiun-Long Huang and Ming-Syan Chen, Fellow,** *IEEE On the Effect of Group Mobility to Data Replication in Ad Hoc Networks*, IEEE Transactions on mobile computing, vol.5, no.5, may 2006

[9] **Paul D. Leedy and Jeanne Ellis Ormrod** *Practical Research*, Pearson Prentice Hall, 2005, ISBN: 0-13-124720-4

[10] **Liang-min Wang, Jian-feng Ma, Chao Wang and Alex Chichung Kot.** Fault and Intrusion Tolerance of Wireless Sensor Networks, 2006 IEEE

[11] **Mead, Nancy R., et al**. *Survivable Network Analysis Method*. Pittsburg, PA 15213, USA : Carnegie Mellon University, Software Engineering Institute, September 2000. p. 58, Technical report. CMU/SEI-2000-TR-013.

[12] NS-2 Wiki. http://nsnam.isi.edu/nsnam/index.php/Main_Page last visited: 2009-05-07

[13] **Milenko Petrovic, Vinod Muthusamyy, Hans-Arno Jacobsen,** *Content-Based Routing in Mobile Ad Hoc Networks,* Department of Electrical

and Computer Engineering Department of Computer Science, University of Toronto

[14] **Schäfer, Günter**. *Security in fixed and wireless networks*. John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2003.

[15] **Software Engineering Institute**, *Carnegie Mellon University Survivable Network Analysis*, http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr013.pdf last visited 2009-05-07

[16] **J.P.G.Sterbenz, R.Krishnan, R.R.Hain, A.W.Jackson, D.Levin, R.Ramanathan and J.Zao** *Survivable Mobile Wireless Networks: Issues, Challanges, and Research Directions*, WiSe'02, September 28, 2002, Atlanta, Georgia, USA.

[17] **US Joint Chief of Staff**, Joint publication 3-60, Joint Targeting, US Joint Chief of Staff, 2007-04-13.

[18] **Wang, Liang-min, et al.** *Fault and Intrusion Tolerance of Wireless Sensor Networks*. 2006, IEEE.

A Subscription routing updates

A:

Node 1

Theme	SubID	FromNode

Node 4

Theme	SubID	FromNode
A	123	4

Node 2

Theme	SubID	FromNode

Node 5

Theme	SubID	FromNode

Node 3

Theme	SubID	FromNode
A	123	4

Node 6

Theme	SubID	FromNode

B:

Node 1

Theme	SubID	FromNode

Node 4

Theme	SubID	FromNode
A	123	4

Node 2

Theme	SubID	FromNode
A	123	3

Node 5

Theme	SubID	FromNode
A	123	3

Node 3

Theme	SubID	FromNode
A	123	4

Node 6

Theme	SubID	FromNode

C:

Node 1

Theme	SubID	FromNode
A	123	2
A	123	5

Node 2

Theme	SubID	FromNode
A	123	3

Node 3

Theme	SubID	FromNode
A	123	4

Node 4

Theme	SubID	FromNode
A	123	4

Node 5

Theme	SubID	FromNode
A	123	3

Node 6

Theme	SubID	FromNode
A	123	5

D:

Node 1

Theme	SubID	FromNode
A	123	2
A	123	5

Node 2

Theme	SubID	FromNode
A	123	3

Node 3

Theme	SubID	FromNode
A	123	4

Node 4

Theme	SubID	FromNode
A	123	4

Node 5

Theme	SubID	FromNode
A	123	3

Node 6

Theme	SubID	FromNode
A	123	5

E:

Node 1

Theme	SubID	FromNode
A	123	2
A	123	5
A	234	2

Node 2

Theme	SubID	FromNode
A	123	3
A	234	2

Node 3

Theme	SubID	FromNode
A	123	4
A	234	2

Node 4

Theme	SubID	FromNode
A	123	4

Node 5

Theme	SubID	FromNode
A	123	3
A	234	2

Node 6

Theme	SubID	FromNode
A	123	5

F:

Node 1

Theme	SubID	FromNode
A	123	2
A	123	5
A	234	2

Node 2

Theme	SubID	FromNode
A	123	3
A	234	2

Node 3

Theme	SubID	FromNode
A	123	4
A	234	2

Node 4

Theme	SubID	FromNode
A	123	4
A	234	3

Node 5

Theme	SubID	FromNode
A	123	3
A	234	2

Node 6

Theme	SubID	FromNode
A	123	5
A	234	5

B Message distribution routing table usage

A:

Node 1

Theme	MsgID	ACKcount

Node 2

Theme	MsgID	ACKcount

Node 3

Theme	MsgID	ACKcount

Node 4

Theme	MsgID	ACKcount

Node 5

Theme	MsgID	ACKcount
A	65	2

Node 6

Theme	MsgID	ACKcount
A	65	BC

B:

Node 1

Theme	MsgID	ACKcount
A	65	-1

Node 2

Theme	MsgID	ACKcount

Node 3

Theme	MsgID	ACKcount

Node 4

Theme	MsgID	ACKcount

Node 5

Theme	MsgID	ACKcount
A	65	2

Node 6

Theme	MsgID	ACKcount
A	65	BC

C:

Node 1

Theme	MsgID	ACKcount
A	65	1

Node 4

Theme	MsgID	ACKcount

Node 2

Theme	MsgID	ACKcount
A	65	1

Node 5

Theme	MsgID	ACKcount
A	65	BC

Node 3

Theme	MsgID	ACKcount
A	65	2

Node 6

Theme	MsgID	ACKcount
A	65	BC

D:

Node 1

Theme	MsgID	ACKcount
A	65	ARECV

Node 4

Theme	MsgID	ACKcount

Node 2

Theme	MsgID	ACKcount
A	65	ARECV

Node 5

Theme	MsgID	ACKcount
A	65	BC

Node 3

Theme	MsgID	ACKcount
A	65	1

Node 6

Theme	MsgID	ACKcount
A	65	BC

E:

Node 1

Theme	MsgID	ACKcount
A	65	ARECV

Node 2

Theme	MsgID	ACKcount
A	65	ARECV

Node 3

Theme	MsgID	ACKcount
A	65	BC

Node 4

Theme	MsgID	ACKcount
A	65	ARECV

Node 5

Theme	MsgID	ACKcount
A	65	BC

Node 6

Theme	MsgID	ACKcount
A	65	BC

C Original protocol implementation

C-1 Introductions

This appendix describes how the Fongen Pub-Sub ad-hoc routing protocol **Feil! Fant ikke referansekilden.** described in the master thesis are implemented, as well as the source code of the actual implementation.

C-2 High level design of the implementation

C-2.1 Introduction

This part defines the principles of the protocol and the necessary procedures to implement the protocol.

C-2.2 Subscription principles

1. Each message is associated with a theme
2. Each node's community of interest is defined by its theme subscriptions

C-2.3 Routing protocol principles

The Publish/Subscribe routing protocol proposed by Fongen **Feil! Fant ikke referansekilden.** is characterized by the following principles:

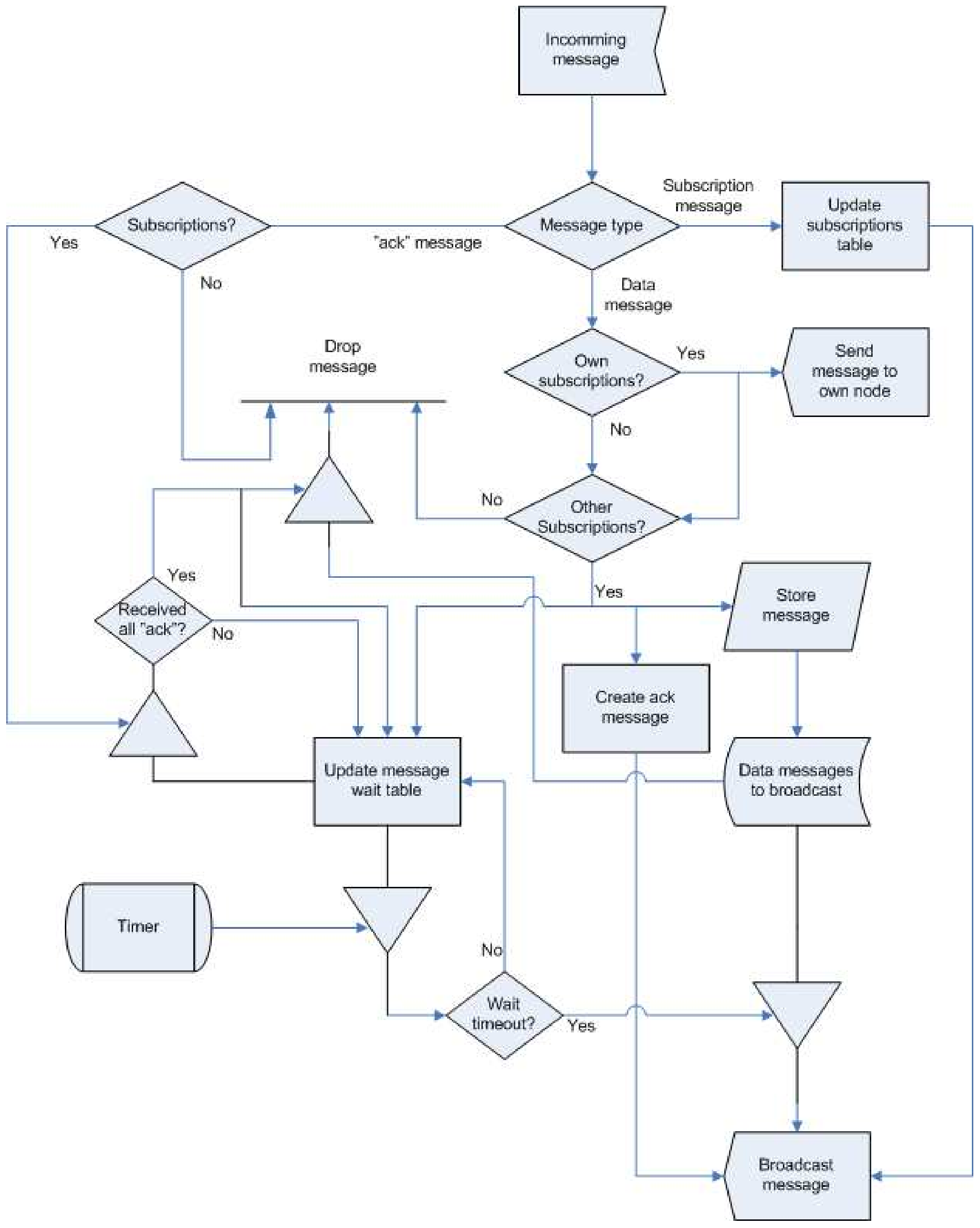
1. Each node's subscriptions describe messages of interest for this node
2. Each node broadcast their subscriptions to neighbor nodes
3. The broadcast of subscription is done on a regular bases to prevent old routes to persist in the network
4. In order to send transit messages each node has to exchange subscription information with their neighboring nodes
    a. have to prevent loops in the routing protocol
    b. subscription information have a aging time to prevent old routes to persist in the network
5. Each node broadcast
    a. own subscription, marked with a random number
    b. subscriptions received from neighbor nodes

C-2.4 Message distribution principles

Defined rules for message forwarding

1. All nodes know their neighbors and their subscriptions, and thereby know if a message has to be forwarded
2. Each node wait a period of time before the message is forwarded because the neighbor nodes might have received the same message, and thereby forwarding would waist of resources
3. During the wait period each node count received "ack"-messages from neighbor nodes
    a. The "ack"-message might arrive prior the received message
4. If one or more nodes with subscription to a message has not send a "ack"-message stating they have received the message, the message has to be forwarded

C-2.5 Protocol high level design



C-3 Implementation

C-3.1 File overview

fongenpubsub.h	-	Headerfile for the main part of the protocol
fongenpubsub.cc	-	Source code for the main part of the protocol
fongenpubsub_pkt.h	-	Implementation of the message packets used in the protocol
fongenpubsub_rtable.h	-	Headerfile for the routing table
fongenpubsub_rtable.cc	-	Source code for the routing table
fongenpubsub_msgtable.h	-	Headerfile to handle message forwarding
fongenpubsub_msgtable.cc	-	Source code to handle message forwarding

C-3.2 Fongenpubsub.h

```
#ifndef __fongenpubsub_h__
#define __fongenpubsub_h__

#include "fongenpubsub_pkt.h"
#include <agent.h>
#include <packet.h>
#include <trace.h>
#include <timer-handler.h>
#include <random.h>
#include <classifier-port.h>
#include <map>
#include "fongenpubsub_rtable.h"
#include "fongenpubsub_msgtable.h"

#define CURRENT_TIME Scheduler::instance().clock()
#define JITTER (Random::uniform()*0.5)

#define DEBUG 0
#define FONGENTRACE 1

#define NORMALFORWARD 0
#define NOFORWARD 1
#define NODATA 2

#define NORMALACK 0
#define RADIOSILENCE 1
#define FALSERANDOMACK 2
#define FALSELEARNEDACK 3
#define MANYACK 4

#define ATTACKMAXNODEAGE 3600
#define ATTACKMAXADDR 254
#define ATTACKNUMOFACK 30
#define NUMOFACK 100

#define FALSENODERANGE 25

struct nodetable {
  int id_;
  float time_;
};
```

```

typedef std::map<int, struct nodetable> nodetable_t;

class Fongenpubsub; // forward declaration

/* Timers */
class Fongenpubsub_Timer : public TimerHandler {
 public:
 Fongenpubsub_Timer(Fongenpubsub* agent) : TimerHandler() {
    agent_ = agent;
  }
 protected:
  Fongenpubsub* agent_;
  virtual void expire(Event* e);
};

class attack_nodetable {
 public:
  nodetable_t nt_;
  attack_nodetable();
  void rm_entry(int);
  void attack_add_node(int);
};

/* Agent */
class Fongenpubsub : public Agent {
  friend class Fongenpubsub_Timer;

  nsaddr_t ra_addr_;
  int forward_;
  int ack_;
  u_int8_t seq_num_;
  fongenpubsub_rtable rtable_;
  fongenpubsub_msgtable msgtable_;
  attack_nodetable nodetable_;
  int createfalsemsg_;

 protected:
  PortClassifier* dmux_; // For passing packets up to agents
  Trace* logtarget_; // For logging
  Fongenpubsub_Timer fongen_timer_; // Timer for updating queues

  inline nsaddr_t& ra_addr() { return ra_addr_; }
  inline int& forward() { return forward_; }
  inline int& ack() { return ack_; }
  inline int& createfalsemsg() { return createfalsemsg_; }

  void forward_data(Packet*);
  void recv_fongenpubsub_pkt(Packet*);
  void send_fongenpubsub_sub_pkt(char*, int);
  void send_fongenpubsub_data_pkt(char*, char*, nsaddr_t, int);
  void send_fongenpubsub_ack_pkt(char*, int);
  void fongenpubsub_update();
  void recv_sub_pkt(Packet*);
  void recv_data_pkt(Packet*);
  void recv_ack_pkt(Packet*);
  void new_sub(char*);
  void remove_sub(char*);
  void new_data(char*, char*);
  void send_to_agent(char*, char*, nsaddr_t, int);
  void reset_fongenpubsub_timer();
  void set_forwardstatus(int);
  void set_ackstatus(int);
  void send_fongenpubsub_false_ack(char*, int, nsaddr_t);
  void create_fongenpubsub_false_msg(char*);
public:
  Fongenpubsub(nsaddr_t);
  int command(int, const char*const*);
  void recv(Packet*, Handler*);

};


#endif
```

C-3.3 Fongenpubsub.cc

```
#include "fongenpubsub.h"
#include <cmu-trace.h>
#include <address.h>
#include <string.h>

int hdr_fongenpubsub_pkt::offset_;
static class FongenpubsubHeaderClass : public PacketHeaderClass {
public:
  FongenpubsubHeaderClass() : PacketHeaderClass("PacketHeader/Fongenpubsub", sizeof(hdr_fongenpubsub_pkt)) {
    bind_offset(&hdr_fongenpubsub_pkt::offset_);
  }
} class_rtProtoFongenpubsub_hdr;

static class FongenpubsubClass : public TclClass {
public:
  FongenpubsubClass() : TclClass("Agent/Fongenpubsub") {}
  TclObject* create(int argc, const char*const* argv) {
    assert(argc == 5);
    return (new Fongenpubsub((nsaddr_t)Address::instance().str2addr(argv[4])));
  }
} class_rtProtoFongenpubsub;

```

```
void Fongenpubsub_Timer::expire(Event* e) {
  agent_->fongenpubsub_update();
  agent_->reset_fongenpubsub_timer();
}

Fongenpubsub::Fongenpubsub(nsaddr_t id) : Agent(PT_FONGENPUBSUB), fongen_timer_(this) {
  bind("forward_", &forward_);
  bind("ack_", &ack_);
  ra_addr_ = id;
  createfalsemsg_ = 0;
}

int Fongenpubsub::command(int argc, const char*const* argv) {
  if (argc == 2) {
    if (DEBUG) {
      fprintf(stderr, "Command: %s, argc: %d\n", argv[1], argc);
    }
    if (strcasecmp(argv[1], "start") == 0) {
      fongen_timer_.resched(0.0);
      return TCL_OK;
    }
    else if (strcasecmp(argv[1], "print-rtable") == 0) {
      if (logtarget_ != 0) {
  sprintf(logtarget_->pt_->buffer(), "P %f _%d_ Routing Table", CURRENT_TIME, ra_addr());
  logtarget_->pt_->dump();
  rtable_.print(logtarget_);
      }
      else {
  fprintf(stdout, "%f _%d_ If you want to print this routing table "
        "you must create a trace file in your tcl script",
        CURRENT_TIME,
        ra_addr());

      }
      return TCL_OK;
    }
  }
  else if (argc == 3) {
    if (DEBUG) {
      fprintf(stderr, "Command: %s, argc: %d, argv2: %s\n", argv[1], argc, argv[2]);
    }
    // Obtains corresponding dmux to carry packets to upper layers
    if (strcmp(argv[1], "port-dmux") == 0) {
      dmux_ = (PortClassifier*)TclObject::lookup(argv[2]);
      if (dmux_ == 0) {
  fprintf(stderr, "%s: %s lookup of %s failed\n",
        __FILE__,
        argv[1],
        argv[2]);
  return TCL_ERROR;
      }
      return TCL_OK;
    }
    // Obtains corresponding tracer
    else if (strcmp(argv[1], "log-target") == 0 || strcmp(argv[1], "tracetarget") == 0) {
      logtarget_ = (Trace*)TclObject::lookup(argv[2]);
      if (logtarget_ == 0)
  return TCL_ERROR;
      return TCL_OK;
    }
    else if (strcmp(argv[1], "fongen-new-sub") == 0) {
      // Add new subscription
      char* theme_;
      theme_ = (char *)malloc(sizeof(argv[2]));
      strcpy(theme_, argv[2]);
      new_sub(theme_);
      return TCL_OK;
    }
    else if (strcmp(argv[1], "fongen-remove-sub") == 0) {
      // Remove subscription
      char* theme_;
      theme_ = (char *)malloc(sizeof(argv[2]));
      strcpy(theme_, argv[2]);
      remove_sub(theme_);
      return TCL_OK;
    }
    else if (strcmp(argv[1], "fongen-set-forward") == 0) {
      forward() = atoi(argv[2]);
      return TCL_OK;
    }
    else if (strcmp(argv[1], "fongen-set-ack") == 0) {
      ack() = atoi(argv[2]);
      return TCL_OK;
    }
    else if (strcmp(argv[1], "fongen-create-falsemsg") == 0) {
      createfalsemsg() = atoi(argv[2]);
      return TCL_OK;
    }
  }
  else if ( argc == 4 ) {
    if (strcmp(argv[1], "fongen-data") == 0) {
      // Handle new message from node
      char* theme_ = (char*)malloc(sizeof(argv[2]));
      char* data_ = (char*)malloc(sizeof(argv[3]));
      strcpy(theme_, argv[2]);
      strcpy(data_, argv[3]);
      new_data(theme_, data_);
      return TCL_OK;
    }
  }
  // Pass the command to the base class
  return Agent::command(argc, argv);
}

void Fongenpubsub::recv(Packet* p, Handler* h) {
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);

  if (ih->saddr() == ra_addr()) {
    // If there exists a loop, must drop the packet
    if (ch->num_forwards() > 0) {
      drop(p, DROP_RTR_ROUTE_LOOP);
      return;
    }
    // else if this is a packet I am orginating, must
add IP header
    else if (ch->num_forwards() == 0)
      ch->size() += IP_HDR_LEN;
  }

  // If it is a Fongenpubsub packet, must process it
  if (ch->ptype() == PT_FONGENPUBSUB)
    recv_fongenpubsub_pkt(p);

  // Otherwise, must forward the packet (unless TTL
has reached zero)
  else {
    ih->ttl_--;
    if (ih->ttl_ == 0) {
      drop(p, DROP_RTR_TTL);
      return;
    }
    forward_data(p);
  }
}

void Fongenpubsub::recv_fongenpubsub_pkt(Packet* p) {
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_fongenpubsub_pkt* ph =
HDR_FONGENPUBSUB_PKT(p);

  assert(ih->sport() == RT_PORT);
  assert(ih->dport() == RT_PORT);

  if (DEBUG) {
    fprintf(stderr, "Node %d recieved packet type
%d\n", ra_addr(), ph->message_type());
  }

  /* ... processing of packet ... */
  if ( ph->message_type()== SUBSCRIPTION )
    Fongenpubsub::recv_sub_pkt(p);
  else if ( ph->message_type() == DATA )
    Fongenpubsub::recv_data_pkt(p);
  else if ( ph->message_type() == ACK )
    Fongenpubsub::recv_ack_pkt(p);
  else
    fprintf(stderr, "Unknown packet recieved.");

  // Release resources
  Packet::free(p);
}

void Fongenpubsub::fongenpubsub_update() {

  /* ... update subscriptions ... */
  float time = CURRENT_TIME;
  int id_;
  while ( (id_=rtable_.update_table(time, ra_addr())) != RESERVED_ID) {
    struct rtabledata *rtableelement = rtable_.lookup(id_);
    if (rtableelement->src_ == ra_addr()) {
      rtableelement->time_ = CURRENT_TIME;
      send_fongenpubsub_sub_pkt(rtableelement->theme_, id_);

      // Make a Log entry about the subscription update
      if (logtarget_ != 0 && FONGENTRACE) {
  sprintf(logtarget_->pt_->buffer(), "FONGEN_SUB_OWN_UPDATE %f %d %s %d", CURRENT_TIME, ra_addr(), rtableelement->theme_, id_);
  logtarget_->pt_->dump();
      }

    }
    else {
      // Make a Log entry about the removal of subscription
      if (logtarget_ != 0 && FONGENTRACE) {
  sprintf(logtarget_->pt_->buffer(), "FONGEN_SUB_REMOVE %f %d %s %d", CURRENT_TIME, ra_addr(), rtableelement->theme_, id_);
  logtarget_->pt_->dump();
      }

      rtable_.rm_entry(id_);
    }
  }

  /* ... update message queue and forward if necessary ... */
  while ( (id_=msgtable_.update_table(time)) != RESERVED_ID) {
    struct msgtabledata *msgtableelement = msgtable_.lookup(id_);
    if (msgtableelement->data_ == NULL) {
      // Make a Log entry about the removal of msg metadata
```

```
        if (logtarget_ != 0 && FONGENTRACE) {
    sprintf(logtarget_->pt_->buffer(), "FONGEN_DATA_REMOVE %f %d %s %d", CURRENT_TIME, ra_addr(), msgtableelement->theme_, id_);
    logtarget_->pt_->dump();
        }
        msgtable_.rm_entry(id_);
      }
      else {
        if ( forward() == NORMALFORWARD ) {
    if (msgtableelement->send_) {

    }
    else {
      // Make a Log entry about forwarding message
      if (logtarget_ != 0 && FONGENTRACE) {
        sprintf(logtarget_->pt_->buffer(), "FONGEN_DATA_FORWARD %f %d %s %s %d %d", CURRENT_TIME, ra_addr(), msgtableelement->theme_, msgtableelement->data_, id_, msgtableelement->src_);
        logtarget_->pt_->dump();
      }

      send_fongenpubsub_data_pkt(msgtableelement->theme_, msgtableelement->data_, msgtableelement->src_, id_);
      msgtableelement->data_ = NULL;
      msgtableelement->send_ = true;
      msgtableelement->time_ = CURRENT_TIME;
    }
        }
        else {
      if (logtarget_ != 0 && FONGENTRACE) {
        sprintf(logtarget_->pt_->buffer(), "FONGEN_ATTACK_DATA_NOTFORWARD %f %d %s %s %d %d", CURRENT_TIME, ra_addr(), msgtableelement->theme_, msgtableelement->data_, id_, msgtableelement->src_);
        logtarget_->pt_->dump();
      }

      msgtableelement->data_ = NULL;
      msgtableelement->send_ = true;
      msgtableelement->time_ = CURRENT_TIME;

        }
      }
    }

    // Update attacker learned nodes
    if ( ack() == 3 ) {
      float time_ = CURRENT_TIME;
      for (nodetable_t::iterator it = nodetable_.nt_.begin(); it != nodetable_.nt_.end(); it++) {
        if ((time_ - (*it).second.time_) > ATTACKMAXNODEAGE ) {
```

```
 nodetable_.rm_entry((*it).first);
      }
    }
  }
}


void Fongenpubsub::forward_data(Packet* p) {

  // Insert code to forward other packets

}

void Fongenpubsub::reset_fongenpubsub_timer() {
  fongen_timer_.resched((double)0.1);
}

void Fongenpubsub::recv_sub_pkt(Packet* p) {
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_fongenpubsub_pkt* ph = HDR_FONGENPUBSUB_PKT(p);
  if (DEBUG) {
    fprintf(stderr, "Node(%d) recv_sub_pkt(%d, %s)\n", ra_addr(), ph->pkt_id(), ph->pkt_theme());
  }

  // Store address of source node(s)
  if (ack() == 3) {
    nodetable_.attack_add_node(ph->pkt_src());
  }

  // Handle subscription packets
  struct rtabledata new_rdata_;
  if (DEBUG) {
    fprintf(stderr, "Node %d new_sub %s\n", ra_addr(), ph->pkt_theme());
  }
  strcpy(new_rdata_.theme_, ph->pkt_theme());
  new_rdata_.src_ = ph->pkt_src();
  new_rdata_.time_ = CURRENT_TIME;
  if (rtable_.add_entry(ph->pkt_id(), new_rdata_, CURRENT_TIME)) {
    // Make a Log entry about the new recieved sub data
    if (logtarget_ != 0) {
      sprintf(logtarget_->pt_->buffer(), "FONGEN_SUB_RECV_NEW %f %d %s %d %d", CURRENT_TIME, ra_addr(), ph->pkt_theme(), ph->pkt_id(), ph->pkt_src());
      logtarget_->pt_->dump();
    }

    // Create a false message with the theme recieved in sub msg if we are instructed to do so
    if (createfalsemsg() == 1) {
      create_fongenpubsub_false_msg(ph->pkt_theme());
    }
    if ( forward() == NORMALFORWARD || forward() == NODATA ) {
      send_fongenpubsub_sub_pkt(ph->pkt_theme(), ph->pkt_id());
    }
    else {
      sprintf(logtarget_->pt_->buffer(), "FONGEN_ATTACK_SUB_STOP %f %d %s %d %d", CURRENT_TIME, ra_addr(), ph->pkt_theme(), ph->pkt_id(), ph->pkt_src());
      logtarget_->pt_->dump();

    }
  }
}

void Fongenpubsub::recv_data_pkt(Packet* p) {
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_fongenpubsub_pkt* ph = HDR_FONGENPUBSUB_PKT(p);

  // Handle data packets
  int subs;
  if (DEBUG) {
    fprintf(stderr, "Node(%d): Recieved packet datapacket, FORWARDSTATUS: %d\n", ra_addr(), forward());
  }

  // Store address of source node(s)
  if (ack() == 3) {
    nodetable_.attack_add_node(ph->pkt_src());
  }

  if (msgtable_.exist(ph->pkt_id())) {
    if ( msgtable_.oldmsg(ph->pkt_id()) ) {
    // Make a Log entry about the new data which are dropped
      if (logtarget_ != 0 && FONGENTRACE) {
   sprintf(logtarget_->pt_->buffer(), "FONGEN_DATA_RECV_DROP %f %d %s %s %d %d OLD_PACKET", CURRENT_TIME, ra_addr(), ph->pkt_theme(), ph->pkt_data(), ph->pkt_id(), ph->pkt_src());
   logtarget_->pt_->dump();
      }

      return;
    }
    else {
      // Make a Log entry about the new data
      if (logtarget_ != 0 && FONGENTRACE) {
   sprintf(logtarget_->pt_->buffer(), "FONGEN_DATA_RECV %f %d %s %s %d %d UPDATE",
CURRENT_TIME, ra_addr(), ph->pkt_theme(), ph->pkt_data(), ph->pkt_id(), ph->pkt_src());
  logtarget_->pt_->dump();
      }

  msgtable_.add_data(ph->pkt_id(), ph->pkt_data(), ph->pkt_src(), CURRENT_TIME, rtable_.num_of_sub_from_source(ph->pkt_theme(), ra_addr()));

      if (rtable_.num_of_sub_from_source(ph->pkt_theme(), ra_addr()) > 0) {
  // Own subscription for data
  send_to_agent(ph->pkt_theme(), ph->pkt_data(), ph->pkt_src(), ph->pkt_id());
      }

    }
  }
  else if ((subs = rtable_.num_of_sub(ph->pkt_theme())) > 0) {
    struct msgtabledata new_msgdata_;
    int ack = rtable_.num_of_sub_from_source(ph->pkt_theme(), ra_addr());
    // Make a Log entry about the new data
    if (logtarget_ != 0 && FONGENTRACE) {
      sprintf(logtarget_->pt_->buffer(), "FONGEN_DATA_RECV %f %d %s %s %d %d", CURRENT_TIME, ra_addr(), ph->pkt_theme(), ph->pkt_data(), ph->pkt_id(), ph->pkt_src());
      logtarget_->pt_->dump();
    }

    if (ack > 0) {
      // Own subscription for data
      send_to_agent(ph->pkt_theme(), ph->pkt_data(), ph->pkt_src(), ph->pkt_id());
    }

    strcpy(new_msgdata_.theme_,ph->pkt_theme());
    new_msgdata_.num_sub_ = subs;
    new_msgdata_.num_ack_ = ack;
    new_msgdata_.time_ = CURRENT_TIME;
    new_msgdata_.oldmsg_ = true;
    if (ack >= subs) {
      new_msgdata_.send_ = true;
      new_msgdata_.data_ = NULL;
    }
    else {
      new_msgdata_.send_ = false;
      new_msgdata_.data_ = (char*)malloc(sizeof(ph->pkt_data()));
      strcpy(new_msgdata_.data_, ph->pkt_data());
    }
    new_msgdata_.src_ = ph->pkt_src();
    msgtable_.addmsg(ph->pkt_id(), new_msgdata_);
  }
  else {
    // Make a Log entry about the new data which are dropped
    if (logtarget_ != 0 && FONGENTRACE) {
      sprintf(logtarget_->pt_->buffer(), "FONGEN_DATA_RECV_DROP %f %d %s %s %d %d NO_SUBSCRIPTIONS", CURRENT_TIME, ra_addr(), ph->pkt_theme(), ph->pkt_data(), ph->pkt_id(), ph->pkt_src());
      logtarget_->pt_->dump();
    }
    return;
  }

  // Send ACK
  if ( ack() == NORMALACK ) {
    send_fongenpubsub_ack_pkt(ph->pkt_theme(), ph->pkt_id());
  }
  else if ( ack() == RADIOSILENCE ) {

  }
  else if ( ack() == FALSERANDOMACK ) {
    nsaddr_t src_;
    for (int i=0; i < ATTACKNUMOFACK; i++) {
      src_ = (nsaddr_t)ATTACKMAXADDR*((float)rand()/(float)RAND_MAX);
      send_fongenpubsub_false_ack(ph->pkt_theme(), ph->pkt_id(), src_);
    }
  }
  else if ( ack() == FALSELEARNEDACK ) {
    for (nodetable_t::iterator it = nodetable_.nt_.begin(); it != nodetable_.nt_.end(); it++) {
      send_fongenpubsub_false_ack(ph->pkt_theme(), ph->pkt_id(), (*it).second.id_);
    }
  }
  else if ( ack() == MANYACK ) {
    for (int i=0; i < NUMOFACK; i++) {
      send_fongenpubsub_ack_pkt(ph->pkt_theme(), ph->pkt_id());
    }
  }
  else {
    fprintf(stderr, "Error: Node(%d): Wrong acktype\n", ra_addr());
  }
}

void Fongenpubsub::recv_ack_pkt(Packet* p) {
  // Handle ack packets
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_fongenpubsub_pkt* ph =
HDR_FONGENPUBSUB_PKT(p);

  int subs;

  // Make a Log entry about the recieved ACK msg
  if (logtarget_ != 0 && FONGENTRACE) {
    sprintf(logtarget_->pt_->buffer(),
"FONGEN_ACK_RECV %f %d %s %d %d", CURRENT_TIME,
ra_addr(), ph->pkt_theme(), ph->pkt_id(), ph-
>pkt_src());
    logtarget_->pt_->dump();
  }

  if (ack() == 3) {
    nodetable_.attack_add_node(ph->pkt_src());
  }

  if (msgtable_.exist(ph->pkt_id())) {
    msgtable_.new_ack(ph->pkt_id(),
rtable_.num_of_sub_from_source(ph->pkt_theme(), ph-
>pkt_src()));
  }
  else if ((subs = rtable_.num_of_sub(ph-
>pkt_theme())) > 0) {
    struct msgtabledata new_msgdata_;
    strcpy(new_msgdata_.theme_,ph->pkt_theme());
    new_msgdata_.num_sub_ = subs;
    new_msgdata_.num_ack_ =
rtable_.num_of_sub_from_source(ph->pkt_theme(), ph-
>pkt_src());
    new_msgdata_.time_ = CURRENT_TIME;
    new_msgdata_.send_ = false;
    new_msgdata_.data_ = NULL;
    new_msgdata_.oldmsg_ = false;
    msgtable_.addmsg(ph->pkt_id(), new_msgdata_);
  }
}

void Fongenpubsub::send_fongenpubsub_data_pkt(char*
theme_, char* data_, nsaddr_t src, int id_) {
  Packet* p = allocpkt();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_fongenpubsub_pkt* ph =
HDR_FONGENPUBSUB_PKT(p);

  ph->pkt_src() = src;
  ph->pkt_len() = (8 + (int) strlen(theme_) + (int)
strlen(data_));
  ph->pkt_id() = id_;
  ph->pkt_theme() = theme_;
  ph->pkt_data() = data_;
  ph->message_type() = DATA;

  ch->ptype() = PT_FONGENPUBSUB;
  ch->direction() = hdr_cmn::DOWN;
  ch->size() = IP_HDR_LEN + ph->pkt_len();
  ch->error() = 0;
  ch->next_hop() = IP_BROADCAST;
  ch->addr_type() = NS_AF_INET;

  ih->saddr() = ra_addr();
  ih->daddr() = IP_BROADCAST;
  ih->sport() = RT_PORT;
  ih->dport() = RT_PORT;
  ih->ttl() = IP_DEF_TTL;

  Scheduler::instance().schedule(target_, p, JITTER);
}

void Fongenpubsub::send_fongenpubsub_ack_pkt(char*
theme_, int id_) {
  Packet* p = allocpkt();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_fongenpubsub_pkt* ph =
HDR_FONGENPUBSUB_PKT(p);

  ph->pkt_src() = ra_addr();
  ph->pkt_len() = (8 + (int)strlen(theme_));
  ph->pkt_id() = id_;
  ph->pkt_theme() = theme_;
  ph->message_type() = ACK;

  ch->ptype() = PT_FONGENPUBSUB;
  ch->direction() = hdr_cmn::DOWN;
  ch->size() = IP_HDR_LEN + ph->pkt_len();
  ch->error() = 0;
  ch->next_hop() = IP_BROADCAST;
  ch->addr_type() = NS_AF_INET;

  ih->saddr() = ra_addr();
  ih->daddr() = IP_BROADCAST;
  ih->sport() = RT_PORT;
  ih->dport() = RT_PORT;
  ih->ttl() = IP_DEF_TTL;

  Scheduler::instance().schedule(target_, p, JITTER);
}

void Fongenpubsub::send_fongenpubsub_sub_pkt(char*
theme_, int id_) {
  Packet* p = allocpkt();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_fongenpubsub_pkt* ph =
HDR_FONGENPUBSUB_PKT(p);

  ph->pkt_src() = ra_addr();
  ph->pkt_len() = (8 + (int) strlen(theme_));
  ph->pkt_id() = id_;
  ph->pkt_theme() = theme_;
  ph->message_type() = SUBSCRIPTION;

  ch->ptype() = PT_FONGENPUBSUB;
  ch->direction() = hdr_cmn::DOWN;
  ch->size() = IP_HDR_LEN + ph->pkt_len();
  ch->error() = 0;
  ch->next_hop() = IP_BROADCAST;
  ch->addr_type() = NS_AF_INET;

  ih->saddr() = ra_addr();
  ih->daddr() = IP_BROADCAST;
  ih->sport() = RT_PORT;
  ih->dport() = RT_PORT;
  ih->ttl() = IP_DEF_TTL;

  Scheduler::instance().schedule(target_, p, JITTER);
}

void Fongenpubsub::new_sub(char* theme_) {
  int id_ = rand();
  struct rtabledata new_rdata_;
  if (DEBUG) {
    fprintf(stderr, "Node(%d) new_sub(%s)\n",
ra_addr(), theme_);
  }
  strcpy(new_rdata_.theme_, theme_);
  new_rdata_.src_ = ra_addr();
  new_rdata_.time_ = CURRENT_TIME;
  rtable_.add_entry(id_, new_rdata_, CURRENT_TIME);

  // Make a Log entry about the new subscription
  if (logtarget_ != 0 && FONGENTRACE) {
    sprintf(logtarget_->pt_->buffer(),
"FONGEN_AGENT_SUB_NEW %f %d %s %d", CURRENT_TIME,
ra_addr(), new_rdata_.theme_, id_);
    logtarget_->pt_->dump();
  }

  send_fongenpubsub_sub_pkt(theme_, id_);
}

void Fongenpubsub::remove_sub(char* theme_) {
  int id;
  if (DEBUG) {
    fprintf(stderr, "Node(%d) remove_sub(%s)\n",
ra_addr(), theme_);
  }
  id = rtable_.find_own_entry(theme_, ra_addr());
  if (DEBUG) {
    fprintf(stderr, "Node (%d) theme_subID(%d)\n",
ra_addr(), id);
  }
  if (id != RESERVED_ID) {
    // Make a Log entry about the removale of subscription
    if (logtarget_ != 0 && FONGENTRACE) {
      sprintf(logtarget_->pt_->buffer(), "FONGEN_AGENT_SUB_REMOVE %f %d %s %d", CURRENT_TIME, ra_addr(), theme_, id);
      logtarget_->pt_->dump();
    }
    rtable_.rm_entry(id);
  }
}

void Fongenpubsub::new_data(char* theme_, char* data_) {
  int id_ = rand();

  // Store metadata in msg queue
  struct msgtabledata new_msgdata_;
  strcpy(new_msgdata_.theme_,theme_);
  new_msgdata_.num_sub_ = 0;
  new_msgdata_.num_ack_ = 0;
  new_msgdata_.time_ = CURRENT_TIME;
  new_msgdata_.send_ = true;
  new_msgdata_.oldmsg_ = true;
  new_msgdata_.data_ = (char*)malloc(sizeof(data_));
  strcpy(new_msgdata_.data_, data_);
  new_msgdata_.src_ = ra_addr();
  msgtable_.addmsg(id_, new_msgdata_);

  // Make a Log entry about the new data
  if (logtarget_ != 0 && FONGENTRACE) {
    sprintf(logtarget_->pt_->buffer(), "FONGEN_AGENT_DATA_NEW %f %d %s %s %d", CURRENT_TIME, ra_addr(), theme_, data_, id_);
    logtarget_->pt_->dump();
  }

  // Send ack msg for this data packet
  send_fongenpubsub_ack_pkt(theme_, id_);

  // Send data packet
  send_fongenpubsub_data_pkt(theme_, data_, ra_addr(), id_);

}

void Fongenpubsub::send_to_agent(char *theme_, char *data_, nsaddr_t src_, int id_) {
  if (logtarget_ != 0 && FONGENTRACE) {
    sprintf(logtarget_->pt_->buffer(), "FONGEN_AGENT_DATA_RECV %f %d %s %s %d %d", CURRENT_TIME, ra_addr(), theme_, data_, id_, src_);
    logtarget_->pt_->dump();
  }
}

```

```
void Fongenpubsub::send_fongenpubsub_false_ack(char* theme_, int id_, nsaddr_t src_) {
  Packet* p = allocpkt();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_fongenpubsub_pkt* ph = HDR_FONGENPUBSUB_PKT(p);

  ph->pkt_src() = src_;
  ph->pkt_len() = (8 + (int)strlen(theme_));
  ph->pkt_id() = id_;
  ph->pkt_theme() = theme_;
  ph->message_type() = ACK;

  ch->ptype() = PT_FONGENPUBSUB;
  ch->direction() = hdr_cmn::DOWN;
  ch->size() = IP_HDR_LEN + ph->pkt_len();
  ch->error() = 0;
  ch->next_hop() = IP_BROADCAST;
  ch->addr_type() = NS_AF_INET;

  ih->saddr() = src_;
  ih->daddr() = IP_BROADCAST;
  ih->sport() = RT_PORT;
  ih->dport() = RT_PORT;
  ih->ttl() = IP_DEF_TTL;

  Scheduler::instance().schedule(target_, p, JITTER);
}

void Fongenpubsub::create_fongenpubsub_false_msg(char* theme_) {
  char *data_ = "False";
  int id_ = rand();
  int src_ = rand()%FALSENODERANGE;

  // Store metadata in msg queue
  struct msgtabledata new_msgdata_;
  strcpy(new_msgdata_.theme_,theme_);
  new_msgdata_.num_sub_ = 0;
  new_msgdata_.num_ack_ = 0;
  new_msgdata_.time_ = CURRENT_TIME;
  new_msgdata_.send_ = true;
  new_msgdata_.oldmsg_ = true;
  new_msgdata_.data_ = (char*)malloc(sizeof(data_));
  strcpy(new_msgdata_.data_, data_);
  new_msgdata_.src_ = src_;
  msgtable_.addmsg(id_, new_msgdata_);

  // Make a Log entry about the new data
  if (logtarget_ != 0 && FONGENTRACE) {
    sprintf(logtarget_->pt_->buffer(), "FONGEN_AGENT_DATA_NEW %f %d %s %s %d", CURRENT_TIME, ra_addr(), theme_, data_, id_);
    logtarget_->pt_->dump();
  }
```

```

    // Send ack msg for this data packet
    send_fongenpubsub_ack_pkt(theme_, id_);

    // Send data packet
    send_fongenpubsub_data_pkt(theme_, data_, src_,
  id_);

  }

  attack_nodetable::attack_nodetable() { }

  void attack_nodetable::rm_entry(int id) {
    nt_.erase(id);
  }

  void attack_nodetable::attack_add_node(int id_) {
    struct nodetable curr_element;
    nodetable_t::iterator it = nt_.find(id_);
    if (it == nt_.end()) {
      curr_element.time_ = CURRENT_TIME;
      curr_element.id_ = id_;
      nt_[id_] = curr_element;
    }
    else {
      (*it).second.time_ = CURRENT_TIME;
    }
  }
```

C-3.4 Fongenpubsub_pkt.h

```
#ifndef __fongenpubsub_pkt_h__
#define __fongenpubsub_pkt_h__

#define SUBSCRIPTION 1
#define DATA 2
#define ACK 3


#include <packet.h>

#define HDR_FONGENPUBSUB_PKT(p)
hdr_fongenpubsub_pkt::access(p)

struct hdr_fongenpubsub_pkt {
  nsaddr_t pkt_src_;
  u_int16_t pkt_len_;
  int pkt_id_;
  char* pkt_theme_;
  int message_type_;
  char* data_;

  inline nsaddr_t& pkt_src() { return pkt_src_; }
  inline u_int16_t& pkt_len() { return pkt_len_; }
  inline int& pkt_id() { return pkt_id_; }
  inline char*& pkt_theme() { return pkt_theme_; }
  inline int& message_type() { return message_type_; }
  inline char*& pkt_data() { return data_; }

  static int offset_;
  inline static int& offset() { return offset_; }
  inline static hdr_fongenpubsub_pkt* access(const
Packet* p) {
    return (hdr_fongenpubsub_pkt*)p->access(offset_);
  }
};

#endif
```

C-3.5 Fongenpubsub_rtable.h

```
#ifndef __fongenpubsub_rtable_h__
#define __fongenpubsub_rtable_h__
#define MAX_THEME_LENGTH 10
#define RESERVED_ID 0
#define PROTECT_TIME 2

#define OWNSUBAGE 30
#define SUBAGE 100

#include <trace.h>
#include <map>
//#include "fongenpubsub.h"

struct rtabledata {
  char theme_[MAX_THEME_LENGTH];
  nsaddr_t src_;
  float time_;
};

typedef std::map<int, struct rtabledata> rtable_t;

class fongenpubsub_rtable {

  rtable_t rt_;

 public:
  fongenpubsub_rtable();
  void print(Trace*);
  void clear();
  void rm_entry(int);
  bool add_entry(int, struct rtabledata, float);
  struct rtabledata *lookup(int);
  u_int32_t size();
  int find_own_entry(char*, nsaddr_t);
  int update_table(float, nsaddr_t);
  int num_of_sub(char*);
  int num_of_sub_from_source(char*, nsaddr_t);
};

```

```
#endif
```

C-3.6 Fongenpubsub_rtable.cc

```
#include "fongenpubsub_rtable.h"

fongenpubsub_rtable::fongenpubsub_rtable() { }

void fongenpubsub_rtable::print(Trace* out) {
  sprintf(out->pt_->buffer(),
"P\tID\ttheme\tsource\ttime");
  out->pt_->dump();
  for (rtable_t::iterator it = rt_.begin(); it !=
rt_.end(); it++) {
    sprintf(out->pt_->buffer(), "P\t%d\t%s\t%d\t%f",
      (*it).first,
      (*it).second.theme_,
      (*it).second.src_,
      (*it).second.time_);
    out->pt_->dump();
  }
}

void fongenpubsub_rtable::clear() {
  rt_.clear();
}

void fongenpubsub_rtable::rm_entry(int id) {
  rt_.erase(id);
}

bool fongenpubsub_rtable::add_entry(int id, struct
rtabledata data, float time) {
  rtable_t::iterator it = rt_.find(id);
  if (it == rt_.end()) {
    rt_[id] = data;
    return true;
  }
  else {
    if ( ( time - (*it).second.time_ ) > PROTECT_TIME)
{
      (*it).second = data;
      return true;
    }
    else {
      return false;
    }
  }
}

struct rtabledata *fongenpubsub_rtable::lookup(int id)
{
  rtable_t::iterator it = rt_.find(id);
  if (it == rt_.end()) {
    return NULL;
  }
  else {
    return &(*it).second;
  }
}

u_int32_t fongenpubsub_rtable::size() {
  return rt_.size();
}

int fongenpubsub_rtable::find_own_entry(char* theme, nsaddr_t own_addr) {
  for (rtable_t::iterator it = rt_.begin(); it != rt_.end(); it++) {
    if (strcmp((*it).second.theme_, theme) == 0 && (*it).second.src_ == own_addr) {
      return (*it).first;
    }
  }
  return RESERVED_ID;
}

int fongenpubsub_rtable::update_table(float time, nsaddr_t own_addr) {
  for (rtable_t::iterator it = rt_.begin(); it != rt_.end(); it++) {
    if ((*it).second.src_ == own_addr) {
      if ((time - (*it).second.time_) > OWNSUBAGE) {
  return (*it).first;
      }
    }
    else {
      if ((time - (*it).second.time_) > SUBAGE) {
  return (*it).first;
      }
    }
  }
  return RESERVED_ID;
}

int fongenpubsub_rtable::num_of_sub(char* theme) {
  int sum = 0;
  for (rtable_t::iterator it = rt_.begin(); it != rt_.end(); it++) {
    if (strncmp(theme, (*it).second.theme_, (int)strlen((*it).second.theme_)) == 0) {
      sum++;
    }
  }
  return sum;
}

int fongenpubsub_rtable::num_of_sub_from_source(char* theme, nsaddr_t src) {
  int sum = 0;
  for (rtable_t::iterator it = rt_.begin(); it != rt_.end(); it++) {
    if ((*it).second.src_ == src) {
      if (strncmp(theme, (*it).second.theme_, (int)strlen((*it).second.theme_)) == 0) {
  sum++;
      }
    }
  }
  return sum;
}
```

C-3.7 Fongenpubsub_msgtable.h

```
#ifndef __fongenpubsub_msgtable_h__
#define __fongenpubsub_msgtable_h__

#define MSGAGE 0.5
#define KEEPAGE 10

#include <trace.h>
#include <map>
#include "fongenpubsub_rtable.h"

struct msgtabledata {
  char theme_[MAX_THEME_LENGTH];
  int num_sub_;
  int num_ack_;
  float time_;
  bool send_;
  char* data_;
  nsaddr_t src_;
  bool oldmsg_;
};

typedef std::map<int, struct msgtabledata> msgtable_t;

class fongenpubsub_msgtable {

  msgtable_t mt_;

 public:
  fongenpubsub_msgtable();
  bool oldmsg(int);
  bool exist(int);
  void addmsg(int, struct msgtabledata);
  void add_data(int, char*, int, float, int);
  void new_ack(int, int);
  int update_table(float);
  struct msgtabledata *lookup(int);
  void rm_entry(int);
};
#endif
```

C-3.8 Fongenpubsub_msgtable.cc

```
#include "fongenpubsub_rtable.h"

fongenpubsub_rtable::fongenpubsub_rtable() { }

void fongenpubsub_rtable::print(Trace* out) {
  sprintf(out->pt_->buffer(),
"P\tID\ttheme\tsource\ttime");
  out->pt_->dump();
  for (rtable_t::iterator it = rt_.begin(); it !=
rt_.end(); it++) {
    sprintf(out->pt_->buffer(), "P\t%d\t%s\t%d\t%f",
      (*it).first,
      (*it).second.theme_,
      (*it).second.src_,
      (*it).second.time_);
    out->pt_->dump();
  }
}

void fongenpubsub_rtable::clear() {
  rt_.clear();
}

void fongenpubsub_rtable::rm_entry(int id) {
  rt_.erase(id);
}

bool fongenpubsub_rtable::add_entry(int id, struct
rtabledata data, float time) {
  rtable_t::iterator it = rt_.find(id);
  if (it == rt_.end()) {
    rt_[id] = data;
    return true;
  }
  else {
    if ( ( time - (*it).second.time_ ) > PROTECT_TIME)
{
      (*it).second = data;
      return true;
    }
    else {
      return false;
    }
  }
}

struct rtabledata *fongenpubsub_rtable::lookup(int id)
{
  rtable_t::iterator it = rt_.find(id);
  if (it == rt_.end()) {
    return NULL;
  }
  else {
    return &(*it).second;
  }
}

u_int32_t fongenpubsub_rtable::size() {
  return rt_.size();
}

int fongenpubsub_rtable::find_own_entry(char* theme, nsaddr_t own_addr) {
  for (rtable_t::iterator it = rt_.begin(); it != rt_.end(); it++) {
    if (strcmp((*it).second.theme_, theme) == 0 && (*it).second.src_ == own_addr) {
      return (*it).first;
    }
  }
  return RESERVED_ID;
}

int fongenpubsub_rtable::update_table(float time, nsaddr_t own_addr) {
  for (rtable_t::iterator it = rt_.begin(); it != rt_.end(); it++) {
    if ((*it).second.src_ == own_addr) {
      if ((time - (*it).second.time_) > OWNSUBAGE) {
  return (*it).first;
      }
    }
    else {
      if ((time - (*it).second.time_) > SUBAGE) {
  return (*it).first;
      }
    }
  }
  return RESERVED_ID;
}

int fongenpubsub_rtable::num_of_sub(char* theme) {
  int sum = 0;
  for (rtable_t::iterator it = rt_.begin(); it != rt_.end(); it++) {
    if (strncmp(theme, (*it).second.theme_, (int)strlen((*it).second.theme_)) == 0) {
      sum++;
    }
  }
  return sum;
}

int fongenpubsub_rtable::num_of_sub_from_source(char* theme, nsaddr_t src) {
  int sum = 0;
  for (rtable_t::iterator it = rt_.begin(); it != rt_.end(); it++) {
    if ((*it).second.src_ == src) {
```

```
   if (strncmp(theme, (*it).second.theme_, (int)strlen((*it).second.theme_)) == 0) {
 sum++;
   }
  }
 }
 return sum;
}
```

C-4 References

[1] **Fongen, Anders.** Pubsub distribusjon i manet - skisse. Oslo : Norwegian Defence Rescearch Establishment, 2007.

[2] NS-2 Wiki. http://nsnam.isi.edu/nsnam/index.php/Main_Page last visited: 2009-05-07

[3] **Ros, F.J. and Ruiz, P.M.** Implementing a New Manet Unicast Routing Protocol in NS2, Dept. of Information and Communications Engineering, University of Murcia, December 2004.

D Createsim

D-1 Introductions

This appendix describes the program used to create random simulation sets used in the simulations in the thesis.

D-1.1 Description

The program is created to automatically create different random simulation scenarios based on different user input parameters.

D-1.2 Usage

createsim <x> <y> <nodes> <time> <tracefile> [<subs> <msgs> <maxsublen> <outputfile> <numberOfFiles>]

x,y	-	size of simulation area
nodes	-	number of nodes in the simulation
time	-	
tracefile	-	name of output file from simulations
subs	-	number of total subscriptions in simulation
msgs	-	number of total messages created in simulation
maxsublen	-	maximum length of subscriptions
outputfile	-	basename of resulting simulation script
numberOfFiles	-	number of simulation files to produce

D-2 Implementation

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define SUBS 10
#define MSGS 100
#define MAXSUBLENGTH 50

const char* themes[] = {"A", "AA", "AB", "AC",
         "AAA", "AAB", "AAC",
         "ABA", "ABB", "ABC",
         "ACA", "ACB", "ACC",
         "B", "BA", "BB", "BC",
         "BAA", "BAB", "BAC",
         "BBA", "BBB", "BBC",
         "BCA", "BCB", "BCC",
         "C", "CA", "CB", "CC",
         "CAA", "CAB", "CAC",
         "CBA", "CBB", "CBC",
         "CCA", "CCB", "CCC"};

void createsim(int x, int y, int nodes, int time,
char* tracefile, int subs, int msgs, int maxsublength,
FILE* output) {
  // Create new simulator and trace file
  fprintf(output, "set ns_ [new Simulator]\nset
tracefd [open %s w]\n$ns_ trace-all $tracefd\n\n",
tracefile);

  // Create topography
  fprintf(output, "set topo [new Topography]\n$topo
load_flatgrid %d %d\n\n", x, y);

  fprintf(output, "create-god %d\n\n        $ns_ node-
config -adhocRouting Fongenpubsub \\\n
-llType LL \\\n                         -macType
Mac/802_11 \\\n                         -ifqType
Queue/DropTail/PriQueue \\\n                        -
ifqLen 50 \\\n                         -antType
Antenna/OmniAntenna \\\n                        -
propType Propagation/TwoRayGround \\\n
-phyType Phy/WirelessPhy \\\n
-channelType Channel/WirelessChannel \\\n
-topoInstance $topo \\\n                        -
agentTrace OFF \\\n                        -
routerTrace OFF \\\n                         -macTrace
OFF \\\n                         -movementTrace
OFF\n\n        for {set i 0} {$i < %d } {incr i} {\n
set node_($i) [$ns_ node]\n                $node_($i)
random-motion 1              ;# enable random motion\n
}\n\n", nodes, nodes);



```

```
  // Generate start positions
  for (int i=0;i<nodes;i++) {
    fprintf(output, "$node_(%d) set X_ %.1f\n", i, x*((float)rand()/(float)RAND_MAX));
    fprintf(output, "$node_(%d) set Y_ %.1f\n", i, y*((float)rand()/(float)RAND_MAX));
    fprintf(output, "$node_(%d) set Z_ 0.0\n\n");
  }

  // Generate subscriptions
  for (int i=0;i<subs;i++) {
    const char* theme_ = themes[(int)((38*((float)rand()/(float)RAND_MAX)))];
    int node = (int)(nodes*((float)rand()/(float)RAND_MAX));
    float start = time*((float)rand()/(float)RAND_MAX);
    float duration = maxsublength*((float)rand()/(float)RAND_MAX);
    float stop = start + duration;
    fprintf(output, "$ns_ at %.1f \"[$node_(%d) agent 255] fongen-new-sub %s\"\n", start, node, theme_);
    if (stop < time) {
      fprintf(output, "$ns_ at %.1f \"[$node_(%d) agent 255] fongen-remove-sub %s\"\n", stop, node, theme_);
    }
  }

  // Generate datamessages
  for (int i=0;i<msgs;i++) {
    const char* theme_ = themes[(int)((38*((float)rand()/(float)RAND_MAX)))];
    int node = (int)(nodes*((float)rand()/(float)RAND_MAX));
    float start = time*((float)rand()/(float)RAND_MAX);
    fprintf(output, "$ns_ at %.1f \"[$node_(%d) agent 255] fongen-data %s Valid\"\n", start, node, theme_);
  }


  // Standard data to the end of file
  fprintf(output, "for {set i 0} {$i < %d} {incr i} { $ns_ at %.1f \"$node_($i) reset\";\n}\n\n", nodes, time);

  fprintf(output, "$ns_ at %d.0 \"stop\"\n$ns_ at %d.01 \"puts \\\"NS EXITING...\\\" ; $ns_ halt\"\nproc stop {} {\n    global ns_ tracefd\n    $ns_ flush-trace\n    close $tracefd\n}\n\nputs \"Starting Simulation...\"\n$ns_ run\n", time, time);
}


int main(int argc, char* argv[]) {
  int x,y,nodes,time;
  FILE* output;

  if (argc == 6) {
    x = atoi(argv[1]);
    y = atoi(argv[2]);
    nodes = atoi(argv[3]);
    time = atoi(argv[4]);
    char* tracefile = argv[5];
    output=stdout;
    createsim(x, y, nodes, time, tracefile, SUBS, MSGS, MAXSUBLENGTH, output);
  }
  else if (argc == 10) {
    int subs, msgs, maxsublength, files;
    x = atoi(argv[1]);
    y = atoi(argv[2]);
    nodes = atoi(argv[3]);
    time = atoi(argv[4]);
    subs=atoi(argv[5]);
    msgs=atoi(argv[6]);
    maxsublength=atoi(argv[7]);
    files = atoi(argv[9]);
    for (int i=0;i<files;i++) {
      char filename[100];
      char tracefile[100];
      sprintf(filename, "%s_%d_%d_%d_%d.tcl", argv[8], nodes, x, y, i);
      sprintf(tracefile, "%s_%d_%d_%d_%d.tr", argv[8], nodes, x, y, i);
      output = fopen(filename, "w");
      createsim(x, y, nodes, time, tracefile, subs, msgs, maxsublength, output);
      fclose(output);
    }
  }
  else {
    fprintf(stderr, "Usage: createsim x y nodes time tracefile <subs msgs maxsublen outputfile numberOfFiles>\n");
  }
}
```

E Simulator control files

This appendix contains about 3000 TCL-scripts used to control the simulator in the simulations. These files are supplied as an external gzip file.

F Datasets

F-1 Baseline

F-1.1 500 x 500 meter, 10 nodes

F-1.1.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

```
0.9078 0.9085  0.7057  0.9108  0.4052
0.9111 0.9173  0.9060  0.9066  0.9103
0.9136 0.9192  0.9112  0.9041  0.7740
0.9065 0.9100  0.5811  0.9024  0.9078
0.9162 0.9114  0.9105  0.9200  0.9060
```

F-1.1.2 Analysis of dataset

	Value
Average	0.8633
Standard error	0.0248
Mean	0.9085
Standard derivation	0.1240
Minimum	0.5148
Maximum	0.9200
Number of elements in dataset	25

F-2.1 500 x 500 meter, 25 nodes

F-1.2.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

```
0.9979 0.9972  0.9980  0.9981  0.9989
0.9979 0.9954  0.9984  0.9968  0.9984
0.9992 0.9969  0.9977  0.9970  0.9965
0.9987 0.9986  0.9980  0.9950  0.9982
0.9990 0.9988  0.9989  0.9975  0.9970
```

F-1.2.2 Analysis of dataset

	Value
Average	0.9978
Standard error	0.0002
Mean	0.9980
Standard derivation	0.0011
Minimum	0.9950
Maximum	0.9992
Number of elements in dataset	25

F-3.1 250 x 1000 meter, 25 nodes

F-1.3.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9986	0.9989	0.9986	0.9999	0.9939
1.0001	0.9972	0.9979	0.9990	0.9923
0.9982	0.9958	1.0000	0.9970	0.9967
0.9982	0.9986	0.9987	0.9979	0.9958
0.9990	0.9993	0.9988	0.9977	0.9978

F-1.3.2 Analysis of dataset

	Value
Average	0.9978
Standard error	0.0004
Mean	0.9982
Standard derivation	0.0018
Minimum	0.9923
Maximum	1.0001
Number of elements in dataset	25

F-4.1 750 x 750 meter, 25 nodes

F-1.4.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9827	0.9654	0.9764	0.9883	0.9797
0.8795	0.9546	0.9864	0.9110	0.8451
0.9922	0.9870	0.9857	0.9844	0.9221
0.9808	0.9862	0.9783	0.9795	0.9618
0.9773	0.8860	0.9884	0.7162	0.9767

F-1.4.2 Analysis of dataset

	Value
Average	0.9509
Standard error	0.0126
Mean	0.9783
Standard derivation	0.0630
Minimum	0.7162
Maximum	0.9922
Number of elements in dataset	25

F-5.1 750 x 750 meter, 50 nodes

F-1.5.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9475	0.9802	0.9685	0.9641	0.9688
0.9475	0.9688	0.9654	0.9802	0.9557
0.9654	0.9557	0.9479	0.9732	0.9618
0.9745	0.9618	0.9654	0.9645	0.9806
0.9641	0.9806	0.9745	0.9802	0.9685

F-1.5.2 Analysis of dataset

	Value
Average	0.9666
Standard error	0.0020
Mean	0.9654
Standard derivation	0.0102
Minimum	0.9475
Maximum	0.9806
Number of elements in dataset	25

F-2 Radio silent nodes

F-1.2 500 x 500 meter, 25 nodes, 1 radio silent node

F-2.1.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9977	0.9977	0.9972	0.9949	0.9976	0.9979	0.9982	0.9962	0.9982	0.9979
0.9976	0.9985	0.9980	0.9955	0.9976	0.9989	0.9987	0.9964	0.9975	0.9969
0.9978	0.9981	0.9981	0.9956	0.9974	0.9983	0.9978	0.9968	0.9981	0.9968
0.9979	0.9985	0.9972	0.9957	0.9975	0.9984	0.9974	0.9958	0.9982	0.9971
0.9972	0.9979	0.9980	0.9959	0.9977	0.9987	0.9985	0.9963	0.9981	0.9965

F-2.1.2 Analysis of dataset

	Value
Average	0.9974
Standard error	0.0001
Mean	0.9977
Standard derivation	0.0009
Minimum	0.9949
Maximum	0.9989
Number of elements in dataset	50

F-2.2 500 x 500 meter, 25 nodes, 3 radio silent nodes

F-2.2.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9972	0.9985	0.9979	0.9958	0.9980	0.9985	0.9981	0.9959	0.9981	0.9973
0.9974	0.9973	0.9973	0.9955	0.9980	0.9989	0.9980	0.9970	0.9982	0.9972
0.9976	0.9984	0.9975	0.9955	0.9973	0.9989	0.9981	0.9974	0.9987	0.9967
0.9977	0.9982	0.9979	0.9948	0.9979	0.9985	0.9981	0.9975	0.9973	0.9973
0.9978	0.9977	0.9976	0.9966	0.9966	0.9990	0.9982	0.9969	0.9978	0.9971

F-2.2.2 Analysis of dataset

	Value
Average	0.9975
Standard error	0.0001
Mean	0.99765
Standard derivation	0.0009

Minimum	0.9948
Maximum	0.9990
Number of elements in dataset	50

F-3.2 250 x 1000 meter, 25 nodes, 1 radio silent node

F-2.3.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9974	0.9976	0.9973	0.9979	0.9979	0.9945	0.9906	0.9971	0.9955	0.9983
0.9984	0.9979	0.9978	0.9976	0.9985	0.9944	0.9919	0.9968	0.9958	0.9982
0.9983	0.9983	0.9964	0.9984	0.9957	0.9933	0.9925	0.9972	0.9963	0.9982
0.9987	0.9981	0.9947	0.9977	0.9978	0.9950	0.9917	0.9971	0.9957	0.9983
0.9984	0.9972	0.9973	0.9979	0.9977	0.9934	0.9928	0.9971	0.9961	0.9987

F-2.3.2 Analysis of dataset

	Value
Average	0.9965
Standard error	0.0003
Mean	0.9973
Standard derivation	0.0021
Minimum	0.9906
Maximum	0.9987
Number of elements in dataset	50

F-4.2 250 x 1000 meter, 25 nodes, 3 radio silent node

F-2.4.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9972	0.9981	0.9980	0.9942	0.9977	0.9952	0.9918	0.9976	0.9831	0.9980
0.9988	0.9979	0.9976	0.9977	0.9982	0.9934	0.9918	0.9965	0.9862	0.9988
0.9980	0.9884	0.9948	0.9812	0.9948	0.9937	0.9841	0.9982	0.9838	0.9978
0.9979	0.9974	0.9975	0.9970	0.9965	0.9926	0.9929	0.9979	0.9968	0.9976
0.9982	0.9968	0.9982	0.9985	0.9969	0.9937	0.9898	0.9976	0.5776	0.9991

F-2.4.2 Analysis of dataset

	Value
Average	0.9867
Standard error	0.0084
Mean	0.9971
Standard derivation	0.0592
Minimum	0.5776
Maximum	0.9991
Number of elements in dataset	50

F-5.2 750 x 750 meter, 50 nodes, 1 radio silent node

F-2.5.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9479	0.9708	0.9737	0.9678	0.9498	0.9712	0.9556	0.9646	0.9784	0.9659
0.9420	0.9672	0.9720	0.9611	0.9813	0.9697	0.9599	0.9643	0.9829	0.9696
0.9398	0.9647	0.9688	0.9575	0.9715	0.9739	0.9547	0.9562	0.9830	0.9688
0.9236	0.9682	0.9748	0.9664	0.9693	0.9711	0.9575	0.9510	0.9829	0.9691
0.9131	0.9657	0.9655	0.9657	0.9773	0.9726	0.9573	0.9580	0.9821	0.9668

F-2.5.2 Analysis of dataset

	Value
Average	0.9643
Standard error	0.0019
Mean	0.9670
Standard derivation	0.0138
Minimum	0.9131
Maximum	0.9830
Number of elements in dataset	50

F-6.2 750 x 750 meter, 50 nodes, 5 radio silent nodes

F-2.6.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9411	0.9711	0.9761	0.9693	0.9675	0.9642	0.9587	0.9619	0.9843	0.9707
0.9515	0.9714	0.9763	0.9711	0.9757	0.9669	0.9557	0.9568	0.9818	0.9643
0.9313	0.9660	0.9772	0.9656	0.9482	0.9630	0.9577	0.9606	0.9837	0.9688
0.9608	0.9719	0.9777	0.9741	0.9741	0.9772	0.9631	0.9394	0.9771	0.9719
0.9263	0.9714	0.9760	0.9716	0.9840	0.9634	0.9520	0.9285	0.9848	0.9699

F-2.6.2 Analysis of dataset

	Value
Average	0.9655
Standard error	0.0020
Mean	0.96905
Standard derivation	0.0139
Minimum	0.9263
Maximum	0.9848
Number of elements in dataset	50

F-3 Stop forwarding DATA-messages attack

F-1.3 500 x 500 meter, 25 nodes

F-3.1.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9977	0.9980	0.9971	0.9952	0.9979	0.9982	0.9986	0.9970	0.9980	0.9966
0.9962	0.9985	0.9979	0.9957	0.9978	0.9984	0.9982	0.9973	0.9976	0.9970
0.9974	0.9983	0.9966	0.9947	0.9979	0.9986	0.9981	0.9966	0.9973	0.9965
0.9977	0.9976	0.9977	0.9963	0.9975	0.9988	0.9985	0.9970	0.9979	0.9969
0.9977	0.9970	0.9977	0.9945	0.9975	0.9988	0.9983	0.9933	0.9979	0.9970

F-3.1.2 Analysis of dataset

	Value
Average	0.9973
Standard error	0.0002
Mean	0.99765
Standard derivation	0.0011
Minimum	0.9933
Maximum	0.9988
Number of elements in dataset	50

F-2.3 250 x 1000 meter, 25 nodes

F-3.2.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9975	0.9959	0.9979	0.9872	0.9982	0.7531	0.9921	0.9964	0.9963	0.9975
0.9977	0.9969	0.9881	0.9980	0.9951	0.9951	0.9917	0.9970	0.9939	0.9969
0.9980	0.9980	0.9974	0.9985	0.9979	0.9957	0.9911	0.9973	0.9950	0.9985
0.9974	0.9973	0.9953	0.9978	0.9981	0.9950	0.9927	0.9967	0.9964	0.9986
0.9984	0.9982	0.9968	0.9981	0.9939	0.9934	0.9909	0.9966	0.9399	0.9979

F-3.2.2 Analysis of dataset

	Value
Average	0.9900
Standard error	0.0050
Mean	0.99685
Standard derivation	0.0352
Minimum	0.7531
Maximum	0.9986
Number of elements in dataset	50

F-3.3 750 x 750 meter, 50 nodes

F-3.3.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9315	0.9691	0.9647	0.9633	0.9817	0.9618	0.9440	0.9588	0.9811	0.9637
0.9451	0.9704	0.9732	0.9631	0.9704	0.9607	0.9565	0.9602	0.9827	0.9650
0.9406	0.9668	0.9745	0.9647	0.9696	0.9713	0.9554	0.9367	0.9777	0.9634
0.9341	0.9653	0.9734	0.9639	0.9729	0.9722	0.9563	0.9435	0.9602	0.9702
0.9524	0.9662	0.9745	0.9618	0.9554	0.9665	0.9553	0.9661	0.9751	0.9701

F-3.3.2 Analysis of dataset

	Value
Average	0.9629
Standard error	0.0017
Mean	0.9647
Standard derivation	0.0120
Minimum	0.9315
Maximum	0.9827
Number of elements in dataset	50

F-4 False ACK-messages attack

F-1.4 500 x 500 meter, 25 nodes

F-4.1.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9054	0.8372	0.9868	0.7599	0.9983	0.9081	0.9686	0.9961	0.8393	0.8486
0.9652	0.8900	0.8384	0.8686	0.9704	0.9705	0.9483	0.6790	0.9135	0.9104
0.9060	0.7397	0.6654	0.9538	0.9513	0.8523	0.7529	0.6828	0.9649	0.9448
0.9014	0.9529	0.8871	0.9949	0.9686	0.7556	0.9329	0.9718	0.6709	0.9552
0.9613	0.9942	0.8388	0.9965	0.9880	0.8643	0.7615	0.7394	0.9082	0.7659
0.9637	0.9018	0.9409	0.6852	0.9537	0.8472	0.7858	0.9858	0.6996	0.9512
0.8159	0.7369	0.9811	0.9643	0.9718	0.9931	0.9963	0.9959	0.9722	0.9610
0.9022	0.7477	0.8161	0.9234	0.9662	0.9758	0.7095	0.9821	0.8991	0.8178
0.7009	0.8347	0.8289	0.7984	0.9182	0.9665	0.9656	0.9361	0.9451	0.9904
0.8651	0.8003	0.9863	0.9616	0.6444	0.9660	0.9204	0.9577	0.9376	0.8562
0.7756	0.8071	0.9387	0.8609	0.8386	0.9961	0.9609	0.9079	0.9686	0.9902
0.9166	0.9196	0.8436	0.9450	0.8691	0.9417	0.7192	0.9520	0.8921	0.7576
0.6989	0.7775	0.9975	0.9893	0.9043	0.9938	0.9167	0.9825	0.9788	0.7825
0.8468	0.8920	0.9968	0.8690	0.7740	0.9018	0.8629	0.9473	0.9352	0.9862
0.8702	0.9477	0.7021	0.8960	0.7642	0.9528	0.8116	0.6781	0.9666	0.9306
0.9472	0.9927	0.8605	0.9267	0.8914	0.8442	0.7740	0.7546	0.8280	0.9811
0.9962	0.6977	0.6560	0.7062	0.6173	0.9939	0.7546	0.9181	0.9940	0.9282
0.9627	0.8979	0.9311	0.7626	0.9970	0.9981	0.6750	0.8719	0.9944	0.9099
0.7369	0.9437	0.8821	0.9239	0.7193	0.9662	0.9665	0.9860	0.8113	0.9951
0.7115	0.9630	0.9429	0.9307	0.7932	0.8928	0.8452	0.9642	0.9099	0.7672
0.6556	0.9465	0.9242	0.7362	0.9335	0.9458	0.7333	0.8385	0.8978	0.9887
0.6936	0.9173	0.7793	0.9876	0.7182	0.6901	0.8751	0.6849	0.9684	0.7678
0.9984	0.9970	0.7927	0.6978	0.9878	0.9977	0.9945	0.9754	0.9928	0.8462
0.7865	0.7511	0.6531	0.9383	0.8614	0.9056	0.7115	0.9105	0.9951	0.6467
0.8977	0.8578	0.9919	0.7961	0.6533	0.9383	0.7245	0.8792	0.8210	0.9426

F-4.1.2 Analysis of dataset

Descriptive statistical analysis

	Value
Average	0.8782
Standard error	0.0065
Mean	0.90815
Standard derivation	0.1031
Minimum	0.6173
Maximum	0.9984
Number of elements in dataset	250

Frequency analysis

Interval	Frequency	Interval	Frequency
0.62	1	0.82	7
0.64	0	0.84	10
0.66	6	0.86	10
0.68	5	0.88	13
0.70	9	0.90	12
0.72	9	0.92	22
0.74	7	0.94	17
0.76	8	0.96	23
0.78	11	0.98	30
0.80	7	1.00	43

F-2.4 250 x 1000 meter, 25 nodes

F-4.2.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.7002	0.6482	0.9193	0.5388	0.9877	0.7562	0.8247	0.9935	0.8676	0.7927
0.9547	0.8038	0.6220	0.7559	0.9641	0.9858	0.8373	0.5841	0.6395	0.9547
0.8093	0.7676	0.5597	0.9913	0.8267	0.8234	0.7053	0.7327	0.7784	0.8093
0.6872	0.6977	0.5830	0.9337	0.9682	0.7061	0.7166	0.8954	0.5926	0.6872
0.9915	0.8898	0.6655	0.9953	0.9000	0.9168	0.8407	0.6664	0.5258	0.9915
0.9421	0.8886	0.5541	0.6191	0.9668	0.7878	0.7225	0.7284	0.5927	0.9421
0.8193	0.6910	0.9933	0.9341	0.8634	0.7984	0.8397	0.9922	0.8881	0.8193
0.7942	0.6975	0.9205	0.9662	0.7059	0.8469	0.6874	0.9572	0.8676	0.7942
0.6048	0.6961	0.5585	0.7556	0.9260	0.8975	0.9522	0.8119	0.8654	0.6048
0.9737	0.7791	0.7210	0.9675	0.5348	0.7385	0.8906	0.9126	0.5954	0.9737
0.7199	0.6906	0.8870	0.5929	0.5558	0.9021	0.9910	0.8887	0.7893	0.7199
0.8225	0.8689	0.7952	0.8690	0.7309	0.9648	0.6390	0.8702	0.5144	0.8225
0.6621	0.8100	0.9602	0.6512	0.6766	0.8152	0.7635	0.6921	0.5934	0.6621
0.9130	0.8661	0.9610	0.6200	0.6301	0.8251	0.7257	0.8107	0.5853	0.9130
0.7359	0.9440	0.5881	0.8640	0.7402	0.9149	0.8390	0.6677	0.8528	0.7359
0.9094	0.9307	0.7757	0.6507	0.9600	0.7293	0.7119	0.7316	0.8580	0.9094
0.9752	0.6778	0.5916	0.5478	0.5696	0.8962	0.7138	0.7132	0.7984	0.9752
0.9412	0.7689	0.8740	0.7575	0.5462	0.9935	0.6871	0.6699	0.9942	0.9412
0.7212	0.9603	0.8034	0.9315	0.6773	0.7287	0.7365	0.8524	0.7940	0.7212
0.5983	0.7722	0.9128	0.8997	0.7137	0.7670	0.7044	0.9560	0.6041	0.5983
0.6186	0.9952	0.8699	0.6692	0.7332	0.7266	0.7230	0.7792	0.5887	0.6186
0.6520	0.9340	0.7210	0.7562	0.7327	0.7396	0.9389	0.5901	0.8836	0.6520
0.9097	0.9908	0.5569	0.5773	0.9039	0.7877	0.9741	0.9952	0.9508	0.9097
0.5964	0.6527	0.5412	0.8397	0.8976	0.7956	0.7991	0.6751	0.8661	0.5964
0.6134	0.7117	0.6698	0.5362	0.6653	0.7887	0.6240	0.8299	0.5140	0.6134

F-4.2.2 Analysis of dataset

Descriptive statistical analysis

	Value
Average	0.7815
Standard error	0.0085
Mean	0.78775
Standard derivation	0.1347
Minimum	0.5140
Maximum	0.9953
Number of elements in dataset	250

Frequency analysis

Interval	Frequency	Interval	Frequency
0.50	0	0.76	6
0.52	2	0.78	9
0.54	4	0.80	13
0.56	8	0.82	10
0.58	2	0.84	11
0.60	16	0.86	5
0.62	9	0.88	12
0.64	5	0.90	13
0.66	6	0.92	13
0.68	13	0.94	8
0.70	10	0.96	12
0.72	13	0.98	14
0.74	21	1.00	15

F-3.4 750 x 750 meter, 50 nodes

F-4.3.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.8910	0.8619	0.9593	0.9242	0.9537	0.9563	0.9252	0.9242	0.9507	0.9432
0.9263	0.8695	0.9486	0.9133	0.9081	0.9318	0.9097	0.9084	0.9558	0.9527
0.9001	0.9324	0.9241	0.9433	0.9452	0.9474	0.9052	0.9405	0.9530	0.9531
0.9032	0.8940	0.9613	0.9313	0.9150	0.9456	0.9023	0.9524	0.9640	0.9455
0.8856	0.9214	0.9521	0.8263	0.9188	0.9287	0.9452	0.9212	0.9584	0.9389
0.9233	0.8574	0.9207	0.9403	0.9346	0.9073	0.9359	0.9405	0.9777	0.9610
0.8952	0.9475	0.9541	0.8968	0.8731	0.9505	0.8976	0.9369	0.9608	0.9473
0.9047	0.8238	0.9562	0.9528	0.9413	0.9391	0.9146	0.9266	0.9703	0.9472
0.8946	0.8806	0.9320	0.9376	0.9601	0.9405	0.9411	0.9288	0.9283	0.9322
0.9334	0.9030	0.9555	0.9284	0.9490	0.9448	0.9193	0.8860	0.9557	0.9518
0.8909	0.8691	0.9515	0.8916	0.9375	0.9407	0.9358	0.9221	0.9642	0.9510
0.9194	0.8705	0.9540	0.8889	0.9194	0.9283	0.9134	0.9537	0.9659	0.9419
0.8524	0.9350	0.8958	0.9173	0.9663	0.9486	0.9089	0.9583	0.9703	0.9340
0.8881	0.8668	0.9451	0.9348	0.9308	0.9448	0.9329	0.9572	0.9737	0.9571
0.9298	0.8883	0.9527	0.9405	0.9081	0.9423	0.9102	0.9503	0.9710	0.9478
0.8726	0.9049	0.9578	0.9481	0.8902	0.9300	0.9288	0.9454	0.9693	0.9385
0.8708	0.8630	0.9557	0.9556	0.9148	0.9256	0.9011	0.9540	0.9307	0.9464
0.9250	0.8438	0.8781	0.9486	0.9010	0.9434	0.9127	0.9556	0.9452	0.9464
0.8665	0.8848	0.8963	0.9133	0.9531	0.9320	0.9311	0.9571	0.9660	0.9556
0.8945	0.9250	0.8527	0.9465	0.9036	0.9303	0.9117	0.9498	0.9554	0.9367
0.7144	0.9232	0.9416	0.9523	0.9305	0.9349	0.9336	0.8836	0.9325	0.9453
0.9161	0.8890	0.9640	0.8994	0.8970	0.9292	0.9412	0.9280	0.9644	0.9682
0.9262	0.9014	0.9217	0.9303	0.9673	0.9619	0.9308	0.9458	0.9571	0.9179
0.9174	0.9050	0.9573	0.9454	0.9208	0.9293	0.9242	0.9536	0.9648	0.9394
0.9146	0.9323	0.8864	0.9501	0.9612	0.9335	0.9364	0.9008	0.9643	0.9579
0.9172	0.8790	0.9037	0.8890	0.9521	0.9141	0.9227	0.9449	0.9344	0.9561
0.8138	0.9532	0.9693	0.8889	0.9026	0.9466	0.9186	0.9295	0.9728	0.9605
0.8751	0.9068	0.9495	0.9429	0.9293	0.9380	0.9178	0.9109	0.8999	0.9338
0.9277	0.9096	0.9494	0.8822	0.9483	0.9132	0.9391	0.9164	0.9691	0.9490
0.9174	0.9348	0.9496	0.9524	0.9085	0.9226	0.9134	0.9397	0.9701	0.9421
0.9199	0.9225	0.9694	0.8997	0.9210	0.9381	0.9347	0.9366	0.9575	0.9616
0.8603	0.9076	0.9316	0.9227	0.9199	0.9267	0.9023	0.9363	0.9560	0.9556
0.8978	0.9263	0.9500	0.8994	0.8952	0.9332	0.9289	0.9473	0.9490	0.9169
0.8710	0.9398	0.9019	0.8752	0.9599	0.9421	0.9087	0.9140	0.9584	0.9486
0.9120	0.9320	0.9175	0.9472	0.9397	0.9431	0.9286	0.9139	0.9498	0.9426
0.9109	0.9084	0.9309	0.9451	0.9349	0.9163	0.9191	0.9412	0.9709	0.9688
0.8816	0.9098	0.9065	0.9591	0.9315	0.9170	0.9073	0.9532	0.9054	0.9326
0.8447	0.8352	0.9467	0.9539	0.9355	0.9494	0.9465	0.9402	0.9633	0.9196
0.8848	0.9258	0.9431	0.9188	0.8935	0.9071	0.9345	0.9440	0.9626	0.9283
0.8706	0.9563	0.9473	0.9312	0.8768	0.9401	0.9357	0.9120	0.9500	0.9455
0.8884	0.9134	0.9125	0.9019	0.8906	0.9363	0.9365	0.9299	0.9052	0.9345

0.8787	0.9262	0.9575	0.8120	0.9241	0.9411	0.9358	0.9214	0.9611	0.9510
0.8343	0.9568	0.9615	0.9304	0.8343	0.9379	0.9174	0.8621	0.9647	0.9255
0.9069	0.9058	0.9224	0.9604	0.9179	0.9335	0.9110	0.9364	0.9683	0.9452
0.9155	0.9455	0.9090	0.9297	0.9136	0.9541	0.9377	0.9405	0.9572	0.9486
0.9172	0.9507	0.9255	0.9425	0.9335	0.9395	0.8130	0.9290	0.8856	0.9411
0.9085	0.9235	0.9465	0.9504	0.9392	0.9291	0.8918	0.9378	0.9701	0.9353
0.9308	0.8707	0.9369	0.9345	0.9276	0.9472	0.9096	0.9356	0.9587	0.9234
0.8977	0.9679	0.9547	0.8701	0.9442	0.9530	0.9323	0.9476	0.9567	0.9239
0.9186	0.8492	0.9543	0.9579	0.9139	0.9253	0.9435	0.9019	0.9457	0.9426

F-4.3.2 Analysis of dataset

	Value
Average	0.9268
Standard error	0.0014
Mean	0.93235
Standard derivation	0.0306
Minimum	0.7144
Maximum	0.9777
Number of elements in dataset	500

Frequency analysis

Interval	Frequency	Interval	Frequency
0.70	0	0.86	6
0.72	1	0.88	22
0.74	0	0.90	40
0.76	0	0.92	93
0.78	0	0.94	136
0.80	0	0.96	153
0.82	3	0.98	41
0.84	5	1.00	0

F-5 False DATA-messages attack

F-1.5 500 x 500 meter, 25 nodes

F-5.1.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9983	0.9946	0.9578	0.9987	0.9994
0.9993	0.9976	0.9982	0.9986	0.9990
0.9966	0.9990	0.9979	0.9981	0.9979
0.9988	1.0004	0.9977	0.9981	0.9996
0.9984	0.9982	0.9973	0.9982	0.9727
0.9962	0.9992	0.9955	0.9974	0.9968
0.9983	0.9847	0.9969	0.9978	0.9978
0.9972	0.9825	0.9982	0.9965	0.9974
0.9981	0.9974	0.9973	0.9882	0.9838
0.9968	0.9984	0.9991	0.9990	0.9991

F-5.1.2 Analysis of dataset

	Value
Average	0.9956
Standard error	0.0011
Mean	0.9979
Standard derivation	0.0074
Minimum	0.9578
Maximum	1.0004
Number of elements in dataset	50

F-2.5 250 x 1000 meter, 25 nodes

F-5.2.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9986	0.9935	0.9986	0.9951	0.9930
0.9968	0.9982	0.9965	0.9452	0.9950
0.9993	0.9985	0.9967	0.9952	0.9919
0.9963	0.9975	0.9970	0.9681	0.9941
0.9947	0.9958	0.9977	0.9966	0.9944
0.9954	0.9972	0.9870	0.9983	0.9835
0.9971	0.9958	0.9737	0.9984	0.9983
0.9957	0.9990	0.9969	0.9844	0.9954
0.9972	0.9980	0.9968	0.9973	0.9982
0.9978	0.9965	0.9969	0.9965	0.9971

F-5.2.2 Analysis of dataset

	Value
Average	0.9939
Standard error	0.0013
Mean	0.99665
Standard derivation	0.0092
Minimum	0.9452
Maximum	0.9993
Number of elements in dataset	50

F-6

F-7 Multiple ACK-messages attack

F-1.7 500 x 500 meter, 25 nodes

F-7.1.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.8935	0.8178	0.8656	0.9232	0.9423	0.8848	0.9533	0.6474	0.8827	0.8250
0.9376	0.9689	0.7883	0.9669	0.9357	0.8345	0.9368	0.9679	0.8800	0.8931
0.8842	0.7165	0.9679	0.9047	0.9441	0.7246	0.7437	0.9479	0.6724	0.9332
0.8781	0.7309	0.6712	0.9598	0.9010	0.9746	0.9533	0.9455	0.9465	0.9578
0.8486	0.7749	0.8386	0.8995	0.6090	0.9496	0.9485	0.6326	0.9505	0.9604
0.7508	0.9000	0.6317	0.8924	0.8446	0.9831	0.7942	0.7258	0.9601	0.7188
0.9621	0.8762	0.8610	0.7090	0.8906	0.9336	0.7367	0.8368	0.9172	0.7380
0.6333	0.6630	0.9046	0.9686	0.7333	0.8093	0.6552	0.9309	0.7709	0.9055
0.6648	0.7289	0.7474	0.6420	0.8717	0.9802	0.7145	0.8082	0.8425	0.9602
0.9674	0.8408	0.7733	0.9091	0.6911	0.9473	0.8561	0.9554	0.7906	0.8989

F-7.1.2 Analysis of dataset

	Value
Average	0.8474
Standard error	0.0107
Mean	0.88135
Standard derivation	0.1068
Minimum	0.6090
Maximum	0.9831
Number of elements in dataset	100

F-2.7 250 x 1000 meter, 25 nodes

F-7.2.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9431	0.7670	0.6311	0.7427	0.9607	0.9067	0.7817	0.7133	0.6190	0.7771
0.6805	0.7378	0.8829	0.9658	0.8979	0.8709	0.8396	0.9856	0.5720	0.9295
0.7934	0.8573	0.5305	0.5938	0.7909	0.7749	0.9833	0.9344	0.5714	0.7885
0.8926	0.6581	0.7726	0.9026	0.9437	0.8026	0.7439	0.7582	0.5667	0.7634
0.9312	0.7398	0.7387	0.9284	0.9322	0.9200	0.6997	0.8972	0.5515	0.9431
0.6819	0.6555	0.8543	0.5488	0.8086	0.7886	0.6661	0.8440	0.7421	0.6805
0.5621	0.8342	0.7794	0.5080	0.6598	0.8729	0.6977	0.6643	0.5765	0.7934
0.6128	0.9271	0.8684	0.6508	0.4749	0.7123	0.6925	0.6398	0.8211	0.8926
0.5559	0.9162	0.6839	0.7323	0.4880	0.7493	0.8983	0.9345	0.8963	0.9312
0.5838	0.9663	0.6458	0.8092	0.6889	0.7079	0.5971	0.9672	0.8215	0.5930

F-7.2.2 Analysis of dataset

	Value
Average	0.7659
Standard error	0.0135
Mean	0.77375
Standard derivation	0.1351
Minimum	0.4749
Maximum	0.9856
Number of elements in dataset	100

F-8 Multiple ACK-messages attack in the enhanced protocol

F-1.8 500 x 500 meter, 25 nodes

F-8.1.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9064	0.9430	0.9233	0.9193	0.9428	0.9500	0.9508	0.9139	0.9233	0.9350
0.9206	0.9461	0.9166	0.9444	0.9497	0.9438	0.9406	0.9534	0.9084	0.9391
0.9266	0.9192	0.9353	0.9334	0.9504	0.9232	0.9147	0.9317	0.8815	0.9442
0.9091	0.9211	0.8863	0.9197	0.9445	0.9727	0.9498	0.9310	0.9426	0.9393
0.8958	0.9119	0.9194	0.9308	0.8983	0.9670	0.9482	0.8976	0.9295	0.9404
0.9051	0.9311	0.9045	0.9243	0.9304	0.9633	0.9192	0.9275	0.9252	0.9214
0.9230	0.9317	0.9322	0.9026	0.9369	0.9551	0.9071	0.9167	0.9122	0.9177
0.8859	0.8970	0.9342	0.9400	0.9165	0.9369	0.9104	0.9245	0.9126	0.9466
0.8743	0.9232	0.8928	0.9106	0.9447	0.9596	0.9095	0.9348	0.8922	0.9458
0.9340	0.9386	0.9071	0.9178	0.9156	0.9645	0.9267	0.9376	0.8851	0.9096

F-8.1.2 Analysis of dataset

	Value
Average	0.9260
Standard error	0.0020
Mean	0.9259
Standard derivation	0.0200
Minimum	0.8743
Maximum	0.9727
Number of elements in dataset	100

F-2.8 250 x 1000 meter, 25 nodes

F-8.2.1 Dataset from simulations

From the simulations we get the following dataset of delivery rates:

0.9519	0.9360	0.8645	0.9273	0.9294	0.9104	0.8915	0.9119	0.8634	0.9265
0.9162	0.9371	0.9257	0.9620	0.9006	0.9458	0.8235	0.9702	0.9036	0.9529
0.9332	0.9521	0.8989	0.8997	0.9148	0.9194	0.9550	0.9421	0.9032	0.9281
0.9505	0.9228	0.9145	0.9343	0.9565	0.9055	0.7252	0.9301	0.8940	0.9107
0.9557	0.9092	0.8799	0.9437	0.9398	0.9439	0.8906	0.9589	0.8363	0.9519
0.9126	0.9246	0.9370	0.8979	0.9234	0.9158	0.8683	0.9569	0.9098	0.9162
0.9131	0.9430	0.9201	0.9430	0.9121	0.9494	0.8864	0.9131	0.8402	0.9332
0.8998	0.9596	0.9142	0.9486	0.9215	0.9251	0.9118	0.8943	0.9105	0.9505
0.8899	0.9401	0.8882	0.9271	0.8178	0.9317	0.8924	0.9433	0.9147	0.9557
0.9226	0.9595	0.8947	0.9445	0.9263	0.9203	0.8728	0.9507	0.9218	0.9293

F-8.2.2 Analysis of dataset

	Value
Average	0.9175
Standard error	0.0036
Mean	0.9222
Standard derivation	0.0357
Minimum	0.7252
Maximum	0.9702
Number of elements in dataset	100