UNIVERSITÀ DI PISA
Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in
Ingegneria dell'Informazione
Informatica, Elettronica, Telecomunicazione

Tesi di Dottorato di Ricerca

Towards a Secure Cooperation Mechanism for Challenging Networks

Angelica Lo Duca

Anno 2012

Al mio caro nonno Antonio.

Sommario

Una Challenging Network (CN) è un paradigma di rete che si adatta ai problemi dell'ambiente, al fine di garantire la comunicazione tra i nodi. Uno dei problemi più importanti di una CN è quello di garantire la cooperazione sicura tra i nodi. In questo lavoro descrivo il problema della cooperazione sicura in tre tipi di CNs: le Underwater Acoustic Networks (UANs), le Delay Tolerant Networks (DTNs) e le Publish/Subscribe Networks (PSNs). Una UAN è un paradigma che permette la comunicazione tra nodi subacquei equipaggiati con modem acustici. Poichè il canale acustico per sua natura è un canale aperto, un avversario, sufficientemente equipaggiato, potrebbe intercettare i messaggi che attraversano la rete. In questo lavoro descrivo una suite crittografica, in grado di proteggere la comunicazione tra i nodi acustici. Una DTN è un paradigma di rete che garantisce la consegna dei messaggi anche in presenza di partizioni della rete. Una DTN si basa sull'assunzione implicita che i nodi cooperino per il routing dei messaggi. Comunque, questa assunzione non può essere soddisfatta quando sono presenti nella rete nodi maliziosi che agiscono da *blackholes* in modo da attrarre o cestinare volontariamente i messaggi. In questo lavoro propongo un protocollo basato sul concetto di reputazione al fine di contrastare i nodi blackholes. Una PSN e' un paradigma di rete che permette la comunicazione da nodi publishers a nodi subscribers, attraverso una infrastruttura chiamata Dispatcher. In questo lavoro presento una PSN sicura in grado di supportare la cooperazione tra le organizzazioni. Il servizio è basato sulla nozione di *gruppo di sicurezza*, composto da un insieme di brokers che rappresentano le organizzazioni e che garantiscono la confidenzialità e l'integrità nella consegna dei messaggi.

Abstract

A Challenging Network (CN) is a network paradigm adapting to the many issues of the environment in order to guarantee the communication among nodes. One of the most important issues of a CN is the problem of secure cooperation among nodes. In fact, an attacker, either internal or external, may constitute a threat for the network. In this work I investigate the problem of secure cooperation in three kinds of CNs: the Underwater Acoustic Networks (UANs), the Delay Tolerant Networks (DTNs) and the Publish/Subscribe Networks (PSNs). A UAN is a network paradigm allowing communication among underwater nodes equipped with acoustic modems. Since the acoustic channel is an open medium, an attacker conveniently equipped could intercept the messages traversing the network. In this work I describe a cryptographic suite, aimed at protecting the communication among underwater acoustic nodes. A DTN is a network paradigm guaranteeing message delivery even in presence of network partitions. A DTN relies on the implicit assumption that nodes cooperate towards message forwarding. However, this assumption cannot be satisfied when there are malicious nodes acting as *blackholes* and voluntarily attracting and dropping messages. In this work I propose a *reputation-based* protocol for contrasting blackholes. A PSN is a network paradigm allowing communication from publishers to subscribers by means of an infrastructure, called Dispatcher. In this work I present a secure PSN conceived to support cooperation between organizations. The service is based on the notion of *security group*, an overlay composed of brokers representing organizations that guarantees confidentiality and integrity in end-to-end delivery of messages and supports clients mobility.

Acknowledgments

This thesis has been prepared while I was working at the Dipartimento di Ingegneria della Informazione, of University of Pisa. I would like to thank my advisor, prof. Gianluca Dini, for his support, his high expertise and humanity and for his helpful comments during the last years. I would like to thank also prof. Andrea Caiti and his stuff for their cooperation with my work. A big thank you to my family for having encouraged me in all situations. I would like to thank also Andrea, for his loving support and his patience during the last years. Last but not least, I would like to thank God, because every day He gives me the strength to walk across this journey of the life.

Grazie di cuore a tutti!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cominciate col fare ciò che è necessario, poi ciò che è possibile. E all'improvviso vi sorprenderete a fare l'impossibile.

FRANCESCO D'ASSISI (1182–1226)

1.1 Introduction

A Challenging Network (CN) is a network which is able to adapt to the specificity of the environment. An example of CN is the Underwater Acoustic Network (UAN) [1], which is characterized by long delays, high packet loss and low bandwidth. Another example of CN can be considered the Delay Tolerant Network (DTN) [2], which is a partitioned network, where the connectivity among nodes is not guaranteed. As a further example of CN, we have the Publish/Subscribe Network (PSN) [3], which is characterized by the presence of many clients, acting one independently on the others.

Each CN presents issues depending on the specificity of the environment. However, we can classify them in three families: a) *static networks*, b) *dynamic networks*, c) *mixed networks*.

In the *static networks*, all the nodes are well known before the network deployment and usually they do not change during the network lifetime. An example of *static networks* is the UAN, which has a limited number of nodes, as they are very expensive.

In the *dynamic networks*, the nodes are not known before the network deployment, because they can add to the network or remove from it dynamically during the network lifetime. An example of *dynamic network* is the DTN, which is composed of different types of nodes (e.g. smartphones, laptops).

Finally, in the *mixed networks* there are some nodes that are known before the network deployment and some other nodes which change during the network lifetime. An example of *mixed network* is the PSN, which is composed of an infrastructure of brokers, acting as dispatcher for the messages, and many clients. The brokers usually are static, while the clients change dynamically.

Since it is not possible to discuss all the problems related to a CN in this work, we focus only on the problems related to the *secure cooperation* among nodes. In particular, this property must be guaranteed even in presence of one or many attackers, either *external* or *internal* to the network.

In this chapter we discuss the requirements of each family of CN in order to guarantee the secure cooperation among nodes. Furthermore, we analyze the effects of an external or internal attack on the network.

The chapter is organized as follows: in Section 1.2 we describe the security issues, then in Section 1.3 we give some evaluation criteria for the networks comparison. Sections 1.4, 1.5 and 1.6 describe the static networks, the dynamic networks and the mixed networks, respectively. In Section 1.7 we compare all the families. Finally in Section 1.8 we give our conclusions.

1.2 Security Issues

A challenging network may be subjected to many threats and malicious attacks [4–7]. In this work we consider only attacks acting against the cooperation among nodes. We define the *secure cooperation* among nodes as the property of the network, according to which: a) a message sent from a sender to a destination is not altered or dropped during the routing process, b) a message from the sender to a destination is not read during the routing process by unauthorized sources.

An attack against the *secure cooperation* can be performed by two types of adversaries:

- *external attacker*: the adversary does not belong to the network, but he has access to the communication channel, which generally is open.
- *internal attacker*: the adversary is a node of the network. In this case he has access to the messages which receives for routing. He can also access to all the other information shared by the nodes (e.g. shared keys).

In order to protect the network from an *external attacker*, it is sufficient to protect the communication channel for example by means of a cryptographic suite which encrypts and authenticates messages. If we want to protect the network from an *internal attacker*, instead, the problem is much more complicated, because it needs a mechanism guaranteeing that nodes behaves as expected. This mechanism may include additional software or hardware which a node must be equipped with or an external trusted entity acting as a watchdog for the network.

1.3 Evaluation Criteria

In order to compare the challenging networks, we define the following metrics, aimed at evaluating their performance.

1.3.1 Security Requirements

This metric specifies what are the external structures used by the network to guarantee the secure cooperation. The security requirements include:

- *Infrastructure.* This metric specifies whether the network needs an infrastructure or not.

- *Additional Hardware.* This metric specifies whether every node of the network needs an additional hardware, such as a Trusted Platform Module (TPM), able to perform the remote attestation of nodes [8–10].

- *Additional Software.* This metric specifies whether every node of the network needs an additional software, such as a Watchdog mechanism, able to monitor the nodes behavior and exclude from the network malicious nodes [11].

1.3.2 Attacks Effects

This metric specifies what kind of effect an attack may produce on the network. It includes:

- *External Attacks Effects.* This metric specifies what kind of effect an external attack may produce on the network.

- *Internal Attacks Effects.* This metric specifies what kind of effect an internal attack may produce on the network.

1.4 Static Networks

A *Static Network* (SN) is a network paradigm where: a) nodes are well known before the network deployment, b) the number of nodes is constant. An example of SN is the Underwater Acoustic Network (UAN), which is a network paradigm allowing communication among underwater nodes equipped with acoustic modems. A UAN can be used for many purposes such as ocean sampling, environment monitoring, undersea explorations, tactical surveillance and disaster prevention. Figure 2.1 shows an example of a UAN. The Figure shows a Base Station (BS), able to communicate both with a Command and Control Centre (C3), located in the land, and many underwater acoustic nodes.

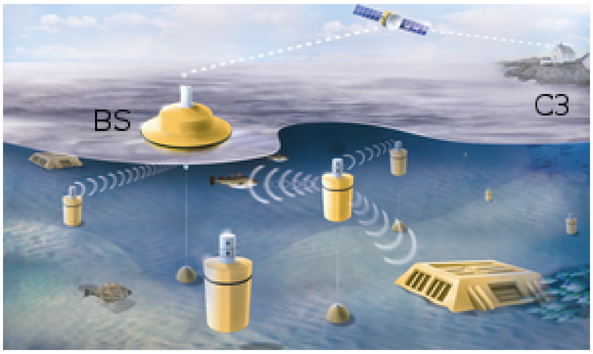


Figure 1.1: A UAN.

1.4.1 Security Requirements

Due to its nature, generally a SN is an infrastructured network. The presence of an infrastructure allows the nodes to efficiently communicate and exchange information.

In order to guarantee the secure cooperation among nodes, the infrastructure can be exploited. For example, there could be a central base station playing the role of authenticating all the nodes of the network or acting as a Watchdog for the network. This means that it is not needed that nodes are equipped with additional hardware in order to support secure cooperation, although each node could be equipped with a TPM.

In order to exchange messages with the base station (e.g. for the authentication), the software of each node must be extended with algorithms and protocols supporting that.

1.4.2 Attacks Effects

Let us assume that the network has a mechanism which authenticates all the nodes. Since the nodes are well known before the network deployment and there is a mechanism for nodes authentication, for an attacker it is difficult to add himself to the network as an additional node. This means that he can act only in the following ways: a) as an *external attacker*, b) compromise an existing node.

In the first case, he has access only to the communication, but not to the nodes. However, if the messages are protected through a cryptographic suite guaranteeing integrity and confidentiality of messages, the attacker cannot perform the external attack.

In the second case, the attacker must be sufficiently equipped in order to compromise an existing node. Once compromised a node, the attacker has full access to the information it stores (e.g. encryption and authentication keys). The effects of this attack could be very destructive, because the attacker can send bogus messages, modify the existing ones or behave into an unpredictable way. In order to protect the network from such an attack, an Intrusion Detection System (IDS) [12] can be added to the network. The IDS can be deployed as a part of the infrastructure. It monitors if the nodes behave as expected; if they do not, it excludes them from the network. However, the adversary is able to control the compromised node until the IDS does not discover it. This means that the protection of the network depends on the efficiency of the IDS to discover a compromised node.

1.5 Dynamic Networks

A *Dynamic Network* (DN) is a network paradigm where: a) nodes change during the network lifetime, b) the number of nodes change during the network lifetime, c) the position of nodes is known only at runtime, d) the nodes are not homogeneous. An example of DN is the Delay Tolerant Network (DTN), which is a network paradigm allowing communication among nodes even in presence of network partitions. Figure 1.2 shows an example of DTN. It is composed of many smartphones, laptops and buses equipped with special routers.

1.5.1 Security Requirements

Generally, a DN does not need an infrastructure, because nodes connect each other by means of the ad-hoc mechanism. However, in some cases the infrastructure is present in order to coordinate

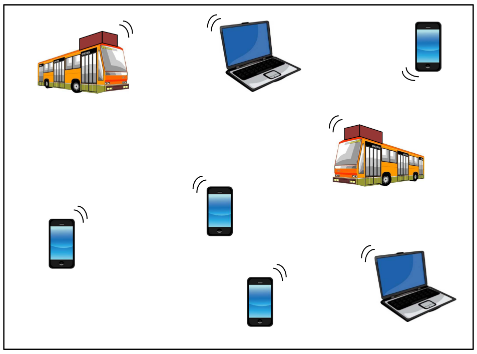


Figure 1.2: A DTN.

the nodes.

An engineer wanting to design a DN guaranteeing the secure cooperation among nodes should take into account that it does not require an infrastructure natively so that he should not introduce it to the network. A well designed DN, instead, should exploit the network features, such as upgrade the nodes capabilities. For example, each node could be equipped with additional hardware (e.g. TPM), allowing it to verify whether the software stack of the other nodes has been compromised or not. Furthermore, the software a node is equipped with could be extended with additional algorithms and protocols (e.g. a Watchdog), allowing a node to verify whether the other nodes behaves as expected or not.

1.5.2 Attacks Effects

Since in a DN the nodes change dynamically during the network lifetime, an attacker can join the network as a standard node. This means that he does not need an additional equipment for example to compromise a node, because it is sufficient that he has a node.

The attacker can launch both an *external* or *internal* attack. In the first case, the network can be protected from the attack through a cryptographic suite, guaranteeing both confidentiality and integrity of messages. In the second case, instead, the at-

	SN	**DN**	**MN**
Infrastructure	YES	NO	YES
Additional Hardware	NO	YES	YES
Additional Software	YES	YES	YES
External Attack	message confidentiality message integrity		
Internal Attack	IDS	TPM Watchdog	IDS TPM Watchdog

Table 1.1: Comparison of challenging networks requirements and attacks countermeasures.

tack may be destructive, because the malicious node can behave into an unpredictable way. The best solution to this problem is to force each node to be equipped with an additional hardware (i.e. a TPM), which allows all the other nodes to perform a remote attestation of the correctness of its software. The TPM should be able to attest that the software has not been altered by the adversary. Furthermore, it should also be able to perform the semantic remote attestation [13].

Another solution to the problem could be extend each node with an additional software (i.e. a Watchdog), which monitors the behavior of the other nodes. If a node does not behave as espected, the Watchdog of all the other nodes, should recognize it and exclude it from the network.

1.6 Mixed Networks

A *Mixed Network* (MN) is a network paradigm where: a) some nodes do not change during the network lifetime but other nodes do, b) the number of nodes is not constant, c) the position of nodes is known only at runtime, d) the nodes are not homogeneous. An example of MN is the Publish/Subscribe Network (PSN), which is a communication paradigm that supports dynamic, asynchronous, many-to-many communication in a distributed system [3]. In a

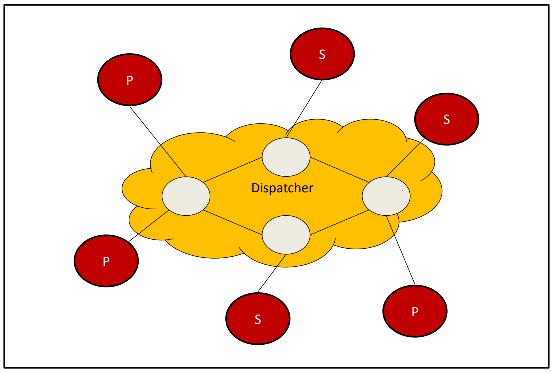


Figure 1.3: A Pub/sub network.

PSN a network of brokers is responsible for routing messages from publishers to subscribers. In practice, they act as a dispatcher for the network. Messages are routed based on their topics, an information descriptor contained in the messages themselves. Subscribers have to declare their interests in specific topics by issuing subscriptions to brokers. The network of brokers does not change during the network lifetime, while publishers and subscribers change dynamically. Figure 1.3 shows an example of PSN. The nodes inside the cloud represent the network of brokers, while the other nodes represent the publishers (P) and the subscribers (S).

1.6.1 Security Requirements

A MS is a semi-infrastructured network, in the sense that some nodes communicate through an ad-hoc mechanism, while some others are connected through an infrastructure. For example, in a PSN, the brokers are connected through the ad-hoc mechanism, but at the same time, they act as the infrastructure for the clients of the network.

In order to guarantee the secure cooperation among nodes, a combination of the techniques described for SNs and DNs can be used. In practice, for nodes supported by the infrastructure, the mechanisms described for SNs can be exploited, while for nodes not supported by the infrastructure, we can use the mechanisms described for DNs. It is important to notice that the two mechanisms must be integrated each other in order to avoid that the network is subjected to new vulnerabilities.

1.6.2 Attacks Effects

An adversary wanting to perform an attack to a MN, can act in two different ways: a) attack the infrastructured nodes, b) attack the ad-hoc nodes. In the first case, the attacker can perform one of the attacks described in Section 1.4.2, while in the second case he can perform one the attacks described in Section 1.5.2.

In the case of the PSN, if the attacker breaks the network of brokers (i.e. performs an attack against the ad-hoc nodes), then he has broken the infrastructure too.

1.7 Discussion

In this section we resume the previous described CNs according to the evaluation criteria described in Section 1.3. Table 1.1 resumes security requirements and attacks countermeasures for each family of CN.

The SNs and MNs are infrastructured, while the DN is not.

In order to protect the network from an external attack, all the families must provide message encryption and message integrity. In the case of an internal attack, the best solution for a SN is to extend the infrastructure with an IDS. For a DN the protection against an internal attack can be achieved by equipping each node with additional hardware (i.e. a TPM) or software (i.e. Watchdog). Finally, a MN can contrast an internal attack by extending the infrastructure with an IDS and by equipping each node with additional hardware (TPM) and software (Watchdog).

1.8 Conclusions

In this chapter we have presented the problem of secure cooperation in CNs. We have classified the CNs in three families: *static*, *dynamic* and *mixed*, according to the type of nodes. We have also shown that the requirements for guaranteeing the secure cooperation change according to the nature of the CN. Furthermore we have analyzed the effects of the external and internal attacks on the different families of CN.

Chapter 2

Underwater Acoustic Networks

Chi lavora con le mani è un operaio,
chi lavora con le mani e la testa è un artigiano,
chi lavora con le mani, la testa e il cuore è un artista.

Francesco d'Assisi (1182–1226)

2.1 Introduction

An Underwater Acoustic Network (UAN) is a network paradigm allowing communication among acoustic underwater nodes. A UAN can be used for many purposes such as ocean sampling, environment monitoring, undersea explorations, tactical surveillance and disaster prevention.

The communication among the underwater nodes is made possible by exploiting the acoustic modems which nodes are equipped with. Since the acoustic channel is an open medium, an attacker conveniently equipped could intercept the messages traversing the network. This could be very dangerous in a tactical surveillance network where messages must be secret. Furthermore, the attacker could also modify or inject fake messages in the network. This could compromise the correctness of data used for the environment monitoring. All these considerations show the urgency of protecting the underwater acoustic channel and establish a secure channel among the underwater nodes.

The problem of secure cooperation among underwater nodes has been investigated in [14], in which the author presents a survey of vulnerabilities and security requirements of a UAN. Akyildiz *et al.* in [15] also present a survey on research challenges for UANs, including security requirements. They think that the best solution for protecting a UAN is a cross layer approach, in which security traverses all the layers. Dong *et al.* in [16] make a taxonomy of the attacks against a UAN. They classify the attacks into three categories: a) physical attacks against nodes, b) attacks against the network, c) attacks against the protocols. They give some guidelines to contrast such attacks, but they do not propose any concrete solution. The same authors in [17, 18] propose some mechanisms for nodes re-organizations, when one or more nodes get destroyed. However, at the best of our knowledge, there is not any work proposing a concrete architecture providing a secure cooperation among underwater acoustic nodes.

In this paper we focus on the problem of secure cooperation of a team of underwater acoustic nodes within surveillance applications and we propose an architecture guaranteeing the secure cooperation among the nodes. The main mission goal is that of protecting an asset (e.g. a critical infrastructure such as a power plant placed on the shore or directly in the water) using detection sonars mounted on each node. We assume that nodes cooperate according to the cooperation algorithm described in [19].

In order to provide a secure communication among nodes, we propose a cryptographic suite which takes into account the peculiarities of the underwater channel. With respect to a terrestrial network, in a UAN communication bandwidth is limited, propagation delays are very long, and underwater nodes have limited energy resources as they are battery operated. It follows that standard cryptographic protocols adopted for terrestrial networks must be revisited for the underwater environment. Traditional techniques such as ciphers, digests and digital signatures make message expand so introducing an amount of overhead that is often comparable to or, even, larger than the payload itself. For these reasons, we propose the use of a cryptographic suite that protects the communication among nodes while keeping at minimum message expansion. The cryptographic suite has been tested during real experiments made in the sea.

We propose two architectures employing the cryptographic suite: a) a *short-medium scale* system, in which all the nodes are into a unique broadcast domain, b) a *large scale* system, in which many broadcast domains exist. In the first case, the cryptographic suite maps directly to the data link layer, while in the second case a routing protocol is needed in order to provide multihop. As routing protocol, we use FLOOD [20], a routing protocol for UANs. FLOOD consists in two phases: the *network discovery* and the *network reconfiguration.* During the *network discovery* phase each vehicle, fixed or mobile, discovers its neighbors and routing tables are built, while during the *network reconfiguration* phase a mobile vehicle moves. Before starting to move, a mobile node temporarily leaves the system. While moving, a mobile node is not reachable by the other node. When the mobile node reaches its new destination, it joins again the system. The *network discovery* phase is executed only once, when the system is deployed, while the *network reconfiguration* phase is executed whenever a mobile node moves.

The *network discovery* phase of FLOOD is subjected to the spoofing-based Denial of Service (DoS) attack, in the sense that an attacker can continuously trigger the discovery phase of FLOOD and never terminate it. In this paper we extend FLOOD with the cryptographic suite and with a mechanism acting against the spoofing-based DoS attack. The resulting protocol is called SeFLOOD. In SeFLOOD, every node discovers and authenticates its neighbors and oganizes them in a *cluster* by distributing them a *cluster key.* Once clusters have been built, routes can be securely established as in FLOOD. Furthermore, mobile nodes are authenticated before adding to the network.

The chapter is organized as follows. Section 2.2 describes the system model, while Section 2.3 the threat model. In Sections 2.4 we describe our cryptographic suite. Section 2.5 describes the short-medium scale. In Section 2.6 we describe the large scale system with reference to FLOOD (Section 2.6.1) and SeFLOOD

(Section 2.6.2). Section 2.7 describes the security analysis. Finally in Section 2.8 we give our conclusions and future work.

2.2 System Model

We consider an underwater acoustic surveillance network aimed at protecting a critical asset by means of a set of underwater nodes. The main task of the network is to detect the presence of physical underwater intruders. The research presented has been carried out in the framework of the European project “Underwater Acoustic Network” (UAN) [1].

From a logical point of view, we consider a system composed of a networked set of underwater nodes, each equipped with sensing, computing and acoustic communication facilities. Every underwater node has computational resources comparable to those of a low-budget personal computer—e.g.,. an Intel Atom processor (also for power saving) or a PC104 embedded standard computer—and it is able to run a commodity operating system such as Windows XP or Linux. However, underwater nodes are battery operated and thus have limited energy resources.

Every node is equipped with a number of sensors that allows it to sense both the state of surrounding sea and the presence of a possible target in the neighbourhood. Of course, a node may only achieve a local partial view of the system. For this reason, a number of underwater nodes are deployed around the critical asset and each of them reports the sensed data to a *Command and Control Center* (C3) station. Such a station collects sensed data from underwater nodes, achieves a global view of the system, and performs an intrusion detection algorithm.

Nodes may be both fixed and mobile. Mobile underwater nodes are unmanned and for this reason we call them *autonomous underwater vehicles* (AUVs). AUVs are crucial components in the surveillance system. The surveillance system strives to cover the largest connected area. Due to the changing sea conditions, fixed nodes alone would be hardly able to guarantee this requirement all the time. For this reason, the system employs AUVs that dynam-

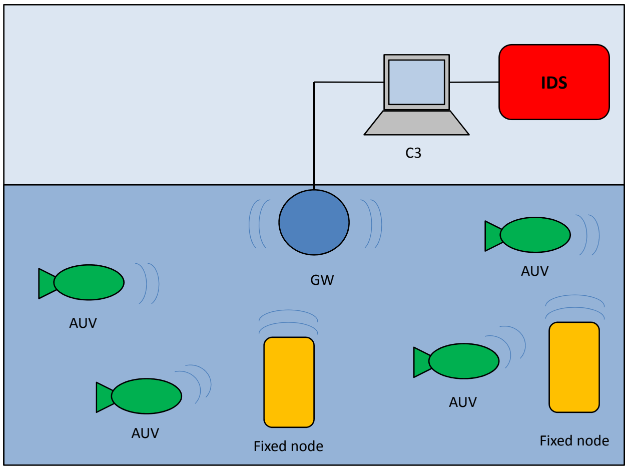


Figure 2.1: The surveillance network.

ically change their position in order to fulfil the requirement [19]. AUVs move to locations in response to specific commands from the C3 station. Upon reaching its target location, an AUV dynamically adjusts its position according to the changed sea conditions in order to both maintain communication connectivity with the other AUVs and guarantee the largest connected detection area.

From the logical model, we can derive two different communication models that the surveillance system has to support. One model is a point-to-point communication model between and underwater node and the C3. The other is a one-to-many communication model between AUVs. Given the peculiarities of an underwater communication system, we support these models by means of the network architecture depicted in Figure 2.1. The *gateway* (GW) is a powerful underwater acoustic node, located under the asset, equipped both with a radio antenna, emerging from the water and an acoustic modem. It communicates with the C3 through the radio antenna and with the underwater nodes through the acoustic modem. In practice, the GW acts as a gateway between the C3 and the underwater nodes. Although the GW is located in the sea, it is not energy-constrained because it is connected to the asset through a power cable.

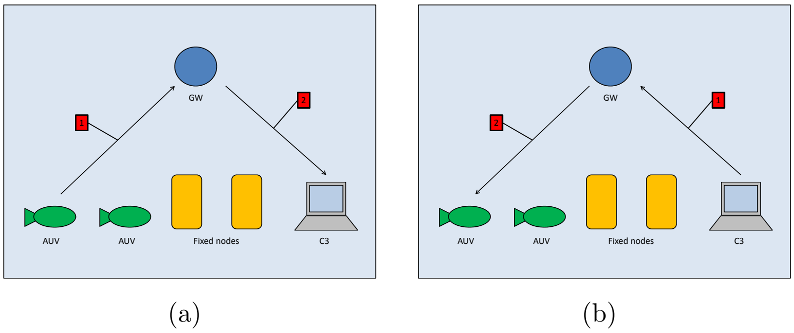


Figure 2.2: The point-to-point communication model.

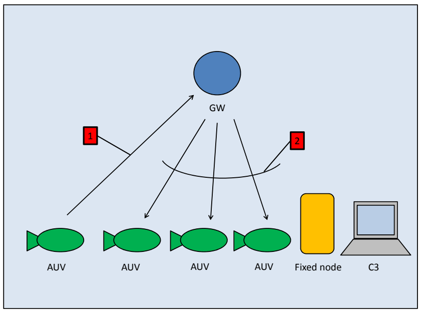


Figure 2.3: The one-to-many communication model.

Figure 2.2 shows how the point-to-point communication model is supported by the UAN network architecture. The point-to-point communication can be of two types: a) from the node to the C3, b) from the C3 to an AUV. In the first case (Figure 2.2 a), a node reports sensed information or other useful information (e.g. its position) to the C3. In order to achieve that, it sends the message to the GW (1), which relays it to the C3 (2). In the second case (Figure 2.2 b), the C3 sends a command to an AUV (e.g. move from a specific location to another of the area). In order to achieve that, the C3 sends the message to the GW (1), which relays it to the AUV (2).

Figure 2.3 shows how the one-to-many communication model is supported by the UAN network architecture. In order to send a message to the other AUVs, AUVs could exploit the broadcast underwater acoustic medium, an thus broadcast a message with sufficient energy to reach all the AUVs, as needed. However, the necessary power would be very large and thus AUVs would drain their battery quickly. As the power needed to send a message decays with powers greater than two of the distance [15, 21], direct link communication between AUVs may not be the most energy efficient solution. For this reason, we assume that when an AUV wants to broadcast a message, it sends the message to the GW (message 1 in the Figure 2.3), which acts as a relay for the message and broadcasts it to the other AUVs (message 2 in the same Figure). The GW can broadcast the message using all the power it is necessary, because it is not resource constrained.

The above communication models can be implemented in underwater environments having different geographical sizes. We consider *small-medium scale* a system composed of a single underwater acoustic broadcast domain, and *large scale* one which instead requires a multi-hop network. In the former case, the logical model of communication maps directly to the real structure of the network and there is no need for a routing protocol, while in the latter case a routing protocol is needed. In this paper we focus on FLOOD [20], a routing protocol for underwater acoustic networks. FLOOD builds a tree according to the Dijkstra algorithm with the GW as root. Then, unicast and multicast messages are routed along the tree.

2.3 Threat Model

The described application scenario may be subjected to several threats and attacks [5]. We assume that the adversary, equipped with an acoustic modem, has full access to communication. Furthermore we assume that when the network is deployed, it is not compromised. However, after the network deploying, the adversary can compromise a node.

In particular, we assume that the adversary can perform the following attacks:

- *spoofing-based* attack against integrity. The adversary impersonates an existing node by injecting wrong measures, statistics or wrong routing information. This could lead to the production of wrong routing tables or wrong results, which potentially can be dangerous,

- *spoofing-based* Denial of Service (DoS) attacks [22]. The adversary can impersonate an existing node and compromize the network integrity with the aim of reducing the network availability. This attack is particularly dangerous because it is difficult to distinguish it from the standard protocol execution,

- *node compromision.* The adversary can compromise a node, accessing to all the information contained in it.

2.4 The Cryptographic Suite

In order to protect the system from the *spoofing-based* attack against integrity, we organize the GW and the nodes in order to form a *secure group* $\mathcal{G}$. Each node n_i belonging to the group shares a link key K_{ig} with the GW. This key is used to secure the communication between the GW and each node. Furthermore, all the nodes and the GW share a group key K_g which is used to secure the broadcast messages sent by the GW to all the nodes.

In the remainder of the section, we describe a cryptographic suite providing confidentiality, authenticity of messages and a key management protocol to distribute a new key whenever needed. Implementing these services in a UAN is challenging due to the severe limitations of the networking environment in terms of very high message propagation delay, very low bandwidth, and high energy consumption for communication. Limitation in the message size is hence of paramount importance in order to reduce battery consumption in AUVs.

2.4.1 Confidentiality

Confidentiality of messages is achieved by encrypting messages. In this paper we use the symmetric encryption technique. Encryption is achieved by splitting cleartext in blocks of fixed, predefined bit-length and encrypting each single block. In the most general case, cleartext length is not multiple of the block length. Thus padding is necessary. However, padding has the negative effect that the ciphertext may result up to one block longer than the corresponding cleartext. This effect is called ciphertext expansion. While ciphertext expansion is negligible in a traditional network, it becomes relevant in wireless sensor networks and, in particular, underwater acoustic networks. In these networks, communication and energy limitations require to keep a message size small and ciphertext expansion may introduce an overhead that is not negligible anymore. In order to completely avoid the ciphertext expansion problem, we use the CipherText Stealing (CTS) technique that alters the processing of the last two blocks of plaintext, resulting in a reordered transmission of the last two blocks of ciphertext and no ciphertext expansion [23].

2.4.2 Authenticity

Encryption without authentication is insecure [24]. For example, an adversary may flip bits in unauthenticated ciphertext and cause predictable changes in the plaintext that receivers are not able to detect. To address this vulnerability, the system always authenticates messages. Security of hash functions is directly related to the length of the digest. However, as a digest is appended to the message, it becomes another source of message expansion and consequent communication overhead. UAN features a trade-off between security and performance by using 4 bytes digests resulting from truncating the real hash function value. Using such a short hash function value is not detrimental to security [25]. An adversary has 1 in 2^{32} chances to blindly forge a digest. If an adversary repeatedly tries to forge it, he/she needs at maximum 2^{31} trials. However, the adversary cannot perform trials off-line. This means

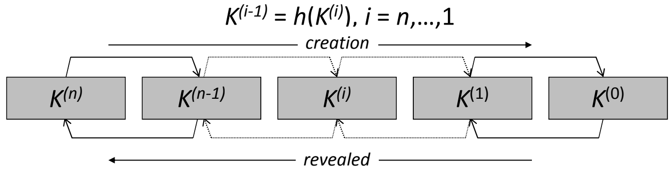


Figure 2.4: Key Chain.

that the adversary has to validate a given forgery only by sending it to an authorized receiver. This implies that the adversary has to send 2^{31} messages in order to successfully forge a single malicious message. In a conventional network this number of trials is not large enough. However, in a underwater acoustic network this may provide an adequate level of security. An adversary can try to flood the network with forgeries, but on a 500 bps channel with 184-bit messages, he/she can only send about 2.71 attempts for second. Thus, sending 2^{31} messages requires around 306 months, i.e., about 25 years. Battery-operated vehicles have not enough energy to receive that many messages. Furthermore, the integrity attack would translate into a denial of service attack since the adversary needs to occupy the acoustic channel for a long time. Fortunately, it is feasible to detect when such a attack is underway. UAN uses a simple heuristic: vehicles could signal the base station when the rate of digest/MAC failures exceeds some predetermined threshold.

2.4.3 Group Key Management

Each time a node leaves the system, the GW generates and distributes a new group key K_g. This is done to avoid that an old node is able to read new messages. As to rekeying protocol, we choose S2RP, a secure and scalable rekeying protocol for resource-constrained devices [26]. S2RP is particularly suitable for UAN for two reasons. First of all, S2RP provides a very efficient proof of key authenticity. Actually, S2RP verifies the authenticity of a key by computing a hash function. So, verification is very computing efficient and does not require any additional information, e.g., MACs or digital signatures, which would cause message expansion. Secondly, S2RP requires a number of rekeying messages that is logarithmic in the number of nodes so making the key distribution phase highly scalable. In short, the key authentication mechanism levers on key-chain, a technique based on the Lamport one-time passwords. A key-chain is a set of symmetric keys so that each key is the hash pre-image of the previous one (see Figure 2.4). Hence, given a key $K^{(i)}$ in the key-chain, anybody can compute all the previous keys $K^{(j)}$, $j \leq i$, however nobody, but the key-chain creator, can compute any of the next keys $K^{(j)}$, $j > i$. Keys are revealed in the reversed order with respect to creation order. Given an authenticated key in the key-chain, anybody can authenticate the next revealed keys by simply applying an hash function. For example, if $K^{(i)}$ is an authenticated key, than anyone can verify the authenticity of $K^{(i+1)}$ by verifying that $K^{(i)} = h(K^{(i+1)})$. To reduce the communication overhead, the GW maintains a logical key tree (see Figure 2.5) each internal node contains a key-chain, whereas each leaf is associated with a node and contains the node-key, i.e., the secret key that the node shares with GW. We call current key of a key-chain the last revealed key of the key-chain and next key the hash pre-image of that key. Furthermore, we denote by K_i and K_i^+ respectively the current and next key of the key-chain associated to tree node i. Notice that $K_i = h(K_i^+)$. Each node maintains a *key-ring* that contains every key K_i such that the sub-tree rooted at node i contains the leaf associated with the node-key. Hence, with reference to Figure 2.5, the *key-ring* of node v_4 is K_1, K_2, K_5. As it turns out, key K_1, associated to the key tree root, is shared by all nodes and acts as the group-key. Let us now suppose that node v_4 leaves the group. All keys in its key ring are considered compromised and KMS has to broadcast the respective next keys K_1^+, K_2^+, K_5^+ by means of the following rekeying messages:

1. GW $\rightarrow$ n_3: $E_{K_{v_3}}(K_5^+)$
2. GW $\rightarrow$ n_3: $E_{K_5^+}(K_2^+)$
3. GW $\rightarrow$ n_1, n_2: $E_{K_4}(K_2^+)$

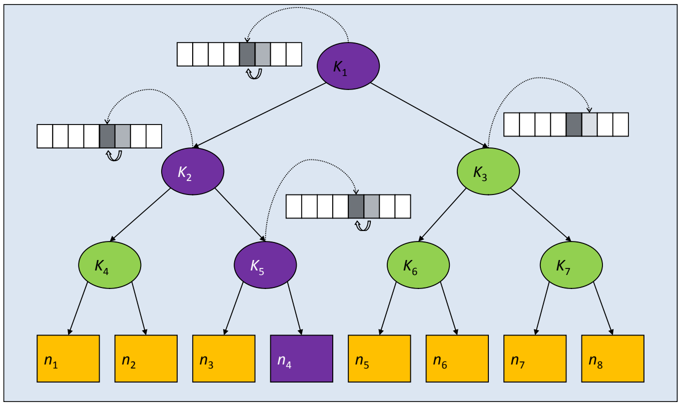


Figure 2.5: Key Tree.

4. GW $\rightarrow n_1, n_2, n_3$: $E_{K_{2+}}(K_1^+)$
5. GW $\rightarrow n_5, n_6, n_7, n_8$: $E_{K_3}(K_1^+)$

Upon receiving a rekeying message, after it has been properly decrypted, the authenticity of the next key therein contained is verified by computing its hash and comparing the result to the corresponding current key. For instance, upon receiving rekeying message 5, n_6 decrypts the message by means of K_3 and verifies the authenticity of K_1^+ by ascertaining that $K_1 = h(K_1^+)$. As it turns out the rekeying protocol requires $O(logN_n)$ rekeying messages, where N_n is the number of nodes. Furthermore, given the key-chain authentication mechanism, every rekeying message needs to carry only the next key (in its encrypted format). No additional information proving key authenticity is thus required. Notice that this is a great advantage in terms of communication overhead with respect to using digital signatures, for example. Let us suppose that group keys are 128-bit long and we use ECC-180 digital signature to authenticate them. ECC-180 is nowadays considered as secure as RSA-1024. In ECC-180 a digital signature is 360 bits and thus a rekeying message would be 488bits, i.e., 3.8125 times longer than in the approach proposed here.

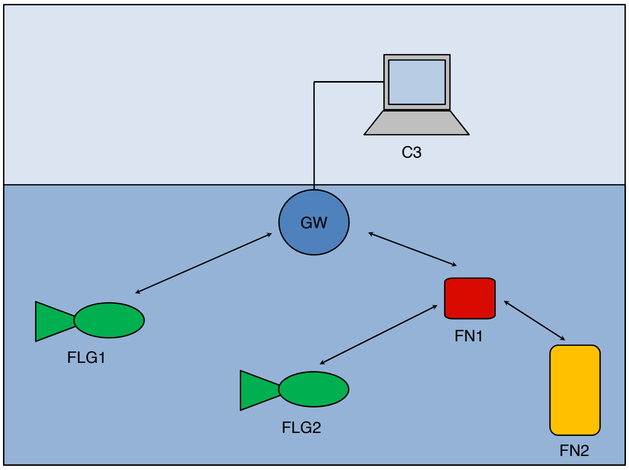


Figure 2.6: The tested scenario.

Node	**Distance from GW (m)**	**Depth(m)**
GW	-	90.3
FN1	163.5	96
FN2	860	39
FLG1	677.8	20
FLG2	687.1	15

Table 2.1: Configuration parameters for Scenario L.

2.4.4 Experimental tests

In this section we analyze performance of our system through experimental tests. We have performed experimental tests during the UAN experiments performed on May 2011 in Norway. For major details about water conditions, winds, temperatures and so on, please see [1]. Figure 2.6 shows the topology of the network which has been tested during the experiments. There are the GW, two fixed nodes (FN1 and FN2), but the FN1 is only a relay node. The network is composed also of two mobile nodes (FLG1 and FLG2). As mobile nodes, we have used the Folaga, deployed by GraalTech [27].

Tables 2.1 shows the configuration parameters. The GW is

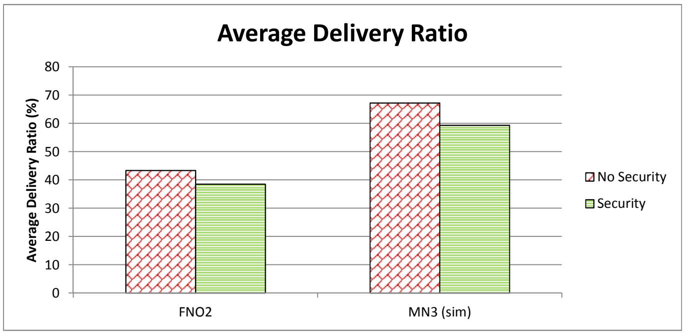


Figure 2.7: Average Delivery Ratio with security and without security.

placed at a depth of 96m, while FN1 and FN2 at a depth of 96m and 39m, respectively, with a distance from the GW of 163.5m and 860m respectively. The nodes FLG1 and FLG2 are placed at a depth of 20m and 15m respectively, with an initial position of 677.8m and 687.1m far from the GW, respectively.

We have measured the Average Delivery Ratio (ADR), in order to verify whether the nodes are still able to communicate even in presence of the overhead added by security. The ADR is the average ratio between the number of received messages and the number of sent messages, both calculated at the application layer. In practice, it measures the fraction of messages the network is able to deliver. Ideally the delivery ratio should be equal to 1. The ADR has been tested with the cryptographic suite enabled (C-ON) and disabled (C-OFF).

Figure 2.7 shows the ADR of nodes FN2 and FLG2. The other nodes are not shown because of lack of data. We note that the ADR without security is greater than that with security in both nodes. We can explain this in the following way. Each message is associated a CRC, which is calculated at the data link layer. When the CRC verification fails, the message is dropped. Furthermore, when the message size increases, also the packet loss for that message increases so that the probability that the message is corrupted and dropped increases. When the security is enabled, the message size is increased of 4 bytes (due to the digest appended to the message) with respect to the message without security. This causes the ADR increase when the security is enabled.

2.5 Small-medium scale system

The *small-medium* scale system is given when all the nodes fall into a unique broadcast domain. In this case a routing layer is not needed so that the logical models described in section 2.2 can be mapped directly to the data link layer. Both the GW and the nodes can send their messages by exploiting the broadcast acoustic channel. Since the messages sent from a node to the GW should be unicast, but the medium is broadcast, we protect such messages through the cryptographic suite described in section 2.4.

2.6 Large Scale System

The large scale system is given when all the nodes do not fall into a unique broadcast domain so that a multihop network is needed. As routing protocol we use FLOOD [20]. FLOOD has the advantage that has been developed and tested in a real UAN. However, it does not provide any mechanism to protect messages from the attacks described in section 2.3. For this reason, we extend it with the cryptographic suite described in section 2.4. In practice, each message exchanged to build routing tables or containing data is encrypted and authenticated through the cryptographic suite. The resulting protocol is called SeFLOOD.

In its original version, the FLOOD protocol does not support the broadcast communication. In order to support the key distribution, we map the S2RP protocol to FLOOD by using the *relay mechanism*: each message is sent only once on a given communication link. Assume that we have a network with the topology shown in Figure 2.8 and the key three for this topology is that shown in Figure 2.5. Furthermore, assume that the node n_4 leaves the network. In this case the keys K_1, K_2 and K_5 must be renewed. The key K_1 must be sent to all the nodes. The GW sends it to n_1

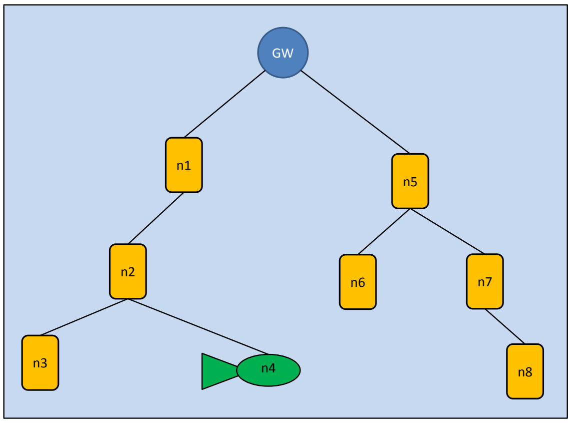


Figure 2.8: An example of topology.

and n_5. Then n_1 and n_5 relay it to n_2 and n_6 and n_7 respectively. Finally, n_2 and n_7 relay it to n_3 and n_8, respectively. The key K_2 must be sent to n_1, n_2 and n_3, while the key K_5 must be sent only to n_3. As the messages containing the keys K_2 and K_5 must follow the same path, they can be aggregated so that the GW sends the aggregated message to n_1, which relays it to n_2, which relays it to n_3.

The *relay mechanism* can be used also when a node wants to broadcast a message to the other nodes (one-to-many communication model). It firstly sends it to the GW, which relays it to the other nodes. The point-to-point communication model, instead, can exploit the sandard unicast routing mechanism supported by FLOOD.

In the remainder of the section we describe firstly FLOOD, then SeFLOOD and finally we compare them through simulation performance analysis.

2.6.1 FLOOD

During the *network discovery* phase, the FLOOD protocol is based on the *flooding* principle. The protocol relies on the idea that a node broadcasts a received message until it receives the same message it has sent. Figure 2.9 shows an execution instance of the FLOOD protocol. Suppose that the network is composed of three nodes. Furthermore, suppose that the transmission range of the GW (n_1) covers only the node n_2, but the node n_2 is able to reach both n_1 and n_3. Node n_3 is able to reach only n_2. The protocol is initiated by the GW which broadcasts a `HELLO`(n_1), including its identifier and the time t_i at which the message is sent. Upon receiving the hello message, the node n_2 adds to the message its identifier and other information, including the quality of the link (e.g. signal attenuation). Then n_2 relays the messasge `HELLO`(n_1,n_2) by broadcasting it after a randomized delay. When n_1 receives the message, it re-broadcasts it. When n_2 receives the new message, it sends to n_1 a `REPORT`(n_1,n_2) unicast message. The GW acknowledges n_2 by sending it a `ACK`(n_1,n_2) unicast message. With the ack message, the protocol between n_1 and n_2 is completed. However, also the node n_3 receives the `HELLO`(n_1,n_2) message. Suppose that n_3 receives it after that n_2 receives `ACK`(n_1,n_2). The node n_3 adds to the message its identifier and information about link quality. Then it broadcasts `HELLO`(n_1,n_2,n_3). Upon receiving this message, the node n_2 re-broadcasts it. The message is received both by the GW n_1 and the node n_3. Upon receiving the message, the node n_3 builds a report message `REPORT`(n_1,n_2,n_3). When n_2 receives the report message from n_3, it sends it the `ACK`(n_1,n_2,n_3) unicast message so that the protocol between n_2 and n_3 is complete. When the GW receives the message `HELLO`(n_1,n_2,n_3) it re-broadcasts it. Upon receiving the message the node n_2 builds a report message `REPORT`(n_1,n_2,n_3). When n_1 receives the report message from n_2, it sends it the `ACK`(n_1,n_2,n_3) unicast message so that the protocol between n_1 and n_2 is complete.

Network Reconfiguration

When the network discovery phase is completed, FLOOD allows the network to dynamically reconfigure. In order to support the *network reconfiguration phase*, it makes the following assumptions: a) each mobile node knows when it is going to move and when it has reached its new location; b) each mobile node does not send/receive

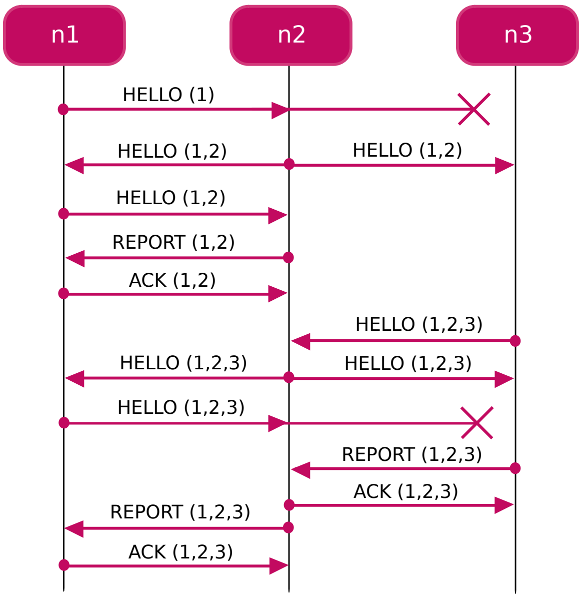


Figure 2.9: The FLOOD protocol.

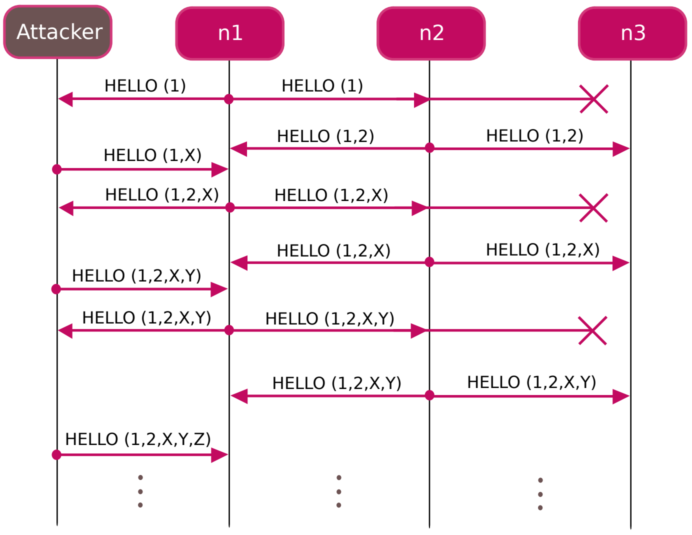


Figure 2.10: A DoS attack.

messages while moving.

Before starting to move, a mobile node n_m sends a DEL(n_m) message to the GW in order to be removed from the routing tables. Upon receiving the message, the GW updates routing tables by considering only the remaining nodes. Then the GW sends a BEST message to each node for which the routing table is changed. The BEST message is unicast. The message includes also information about the new next hop to reach the GW.

Assume now that the mobile node has reached its new location so that it broadcasts an ADD(n_m) message. Upon receiving the ADD(n_m) message, each node sends back it a ADD-ACK message containing information about link quality. The mobile node acknowledges all the received ADD-ACK messages by sending back unicast ADD-ACK messages, containing information about link quality. Finally, the mobile node sends an ADD-REPORT message to the GW so that the GW can update the routing table and distribute to all the interested nodes the new table by sending a ADD-BEST message.

The spoofing-based DoS attack

The FLOOD protocol is subjected to the spoofing-based DoS attack. Figure 2.10 shows an example of how an attacker can perform a spoofing-based DoS attack in a network composed of three nodes. Whenever the attacker receives a `HELLO` message, it can add a *new fake node* and broadcast the message to the other vechicles. In this way the protocol never completes. In order to avoid this kind of attack, each node could be deployed with the list of all the nodes identifiers belonging to the system. Thus, if the attacker tries to add a new fake node to the network, each node rejects it, because the fake node is not contained in that list. However, due to the underwater conditions and to the high distances among nodes, it may happen that a node n_1 is not able to directly contact nodes n_2 and n_3 belonging to the system. The attacker could exploit this situation in order to use n_2 and n_3 identifier to perform a DoS attack.

2.6.2 SeFLOOD

In order to protect the FLOOD protocol from the threats described in section 3.3.1, we have extended it with the cryptographic suite described in section 2.4. The resulting protocol, SeFLOOD, encrypts and authenticates all the messages by using the cryptographic suite. Furthermore, SeFLOOD extends FLOOD with a mechanism contrasting the DoS attack. This mechanism is based on the *Cluster Key Distribution Protocol* (CKDP).

In order to improve security, we organize nodes in *clusters*, such that if an adversary compromises a node, he is able to compromize only the cluster which the node belongs to and not all the network. Each cluster is associated to a Cluster Key, which is used to secure communication among nodes belonging to the same cluster. As the number of nodes in the network is low, all the nodes could be pre-deployed with all the possible Cluster Keys. This will be discussed later in the paper. In the reminder of the section, we firstly describe the CKDP, which dynamically distributes a Cluster Key only to nodes belonging to that cluster. Then we describe a particular application scenario where the Cluster Keys can be predeployed in the nodes.

The Cluster Key Distribution Protocol

We consider the network as connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertexes $\mathcal{V}$ representing nodes and edges $\mathcal{E}$ their connections. Each connection has a cost $\mathcal{C}$, given by the channel quality (e.g. signal attenuation). We assume that each pair of nodes (n_i, n_j), $i \neq j$ in the network shares a key, called *Link Key* K_{ij}, which is used to protect unicast messages between n_i and n_j. In order to distribute the link keys, a well known key agreement protocol can be used such as Elliptic curve Diffie-Hellman [28] or the Blundo scheme [29]. However, for simplicity, each node can be deployed with a table, called *Link Key Table* (LKT), containing all its *Link Keys*. This solution is indicated for a UAN because the number of nodes is low and each node has enough memory to store all the LKT. For example, with a Link Key of 128 bits and a network of 1024 nodes, the LKT is 16 Kbytes. If we consider that a node is equipped with a memory of 2 GBytes, for a network of 1024 nodes, the LKT wastes only 0.78% of memory.

We split the graph $\mathcal{G}$ into N subgraphs $\mathcal{G}_1, \mathcal{G}_2, ..\mathcal{G}_N$ such that each subgraph $\mathcal{G}_n$, $0 \leq n \leq N$ constitutes a *cluster*. A *cluster* is made of a subset of nodes contained into a unique broadcast domain.

In order to securely and efficiently broadcast messages into a cluster, each node n_i generates a *Cluster Key* K_i and distributes it to all the members of its cluster. Thus each node maintains a Cluster Key for each node of the cluster. The *Cluster Key Table* (CKT) of node n_i stores all the Cluster Keys of the members of the cluster which n_i belongs to. In general, the number of entries of the CKT is less than the number of entries of the LKT. However if all the nodes fall into a unique broadcast domain, the number of entries in CKT is equal to that of LKT. This means that also the memory wasted by CKT is negligible.

In order to distribute all the Cluster Keys, the following *Cluster Key Distribution protocol* is executed:

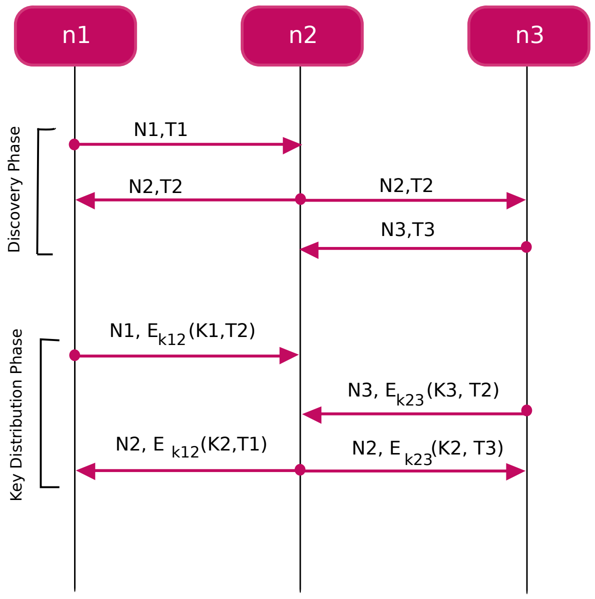


Figure 2.11: The CKDP applied to FLOOD.

1. $n_i \rightarrow * : n_i, T_i$
2. $n_j \rightarrow n_i : n_j, E_{K_{ij}}(K_j, T_i)$

The protocol is composed of two phases, the *discovery phase* (1) and the *key distribution phase* (2). During the *discovery phase* each node n_i generates a fresh quantity T_i and broadcasts it together with its identifier n_i. Upon receiving the message (1), each node n_j sends n_i its cluster key K_j, its identifier n_j and the fresh quantity n_j, all encrypted with the key K_{ij} (2). Upon receiving the message from n_j, the node n_i verifies its freshness and stores the received cluster key K_j.

The CKDP is executed before FLOOD, in order to allow only authorized nodes to trigger the FLOOD protocol. In fact, an adversary performing the DoS attack can only replay the first message of the protocol, but he cannot trigger all the FLOOD protocol.

Figure 2.11 shows an instance of CKDP. Assume that the network is composed of three nodes, the GW n_1 and the nodes n_2 and n_3. Furthermore, suppose that the transmission range of n_1 covers only the node n_2, but the node n_2 is able to reach both n_1 and n_3. node n_3 is able to reach only n_2. Each node broadcasts its identifier and a fresh quantity. As the medium access is CSMA like, a single node per time can transmit. Assume that at the beginning the node n_1 broadcasts the message $[n_1, T_1]$. Upon receiving the message, the node n_2 encrypts its cluster key K_2 and the received T_1 with the link key K_{12}. Then n_2 appends to the message its identifier n_2 and sends the message back to n_1. As the medium is not busy, the node n_2 now can broadcast the message $[n_2, T_2]$. Both n_1 and n_3 receive it so that they can encrypt their cluster keys, K_1 and K_3 respectively and the fresh quantity T_2, received from n_2 with the link ley K_12 and K_13 respectively. Finally they send the message back to n_2, specifying also their identifier. Eventually, n_3 broadcasts the message $[n_3, T_3]$. Upon receiving it, n_2 encrypts the fresh quantity T_3 and cluster key K_2, with the link key K_{23}, and sends the message back to n_3. At the end of the protocol three clusters are built, $\mathcal{C}_1 = (n_1, n_2)$, $\mathcal{C}_2 = (n_1, n_2, n_3)$, $\mathcal{C}_3 = (n_2, n_3)$ with cluster keys K_1, K_2 and K_3, respectively.

Secure Network Reconfiguration

SeFLOOD supports also mobile nodes, without adding overhead in terms of number of messages. In fact it exploits FLOOD messages sent by a mobile node in order to distribute the cluster keys. We assume that nodes have loosely synchronized clocks. This hypotesis is reasonable because the system is centralized so that the master can periodically send a synchronization messages to all the nodes of the network.

Assume that n_m is a mobile node. If n_m wants to delete from the network, it triggers the following protocol:

1. $n_m \rightarrow GW$: `DEL`, n_m, T_m, $E_{gm}(T_m, n_m)$

2. $GW \rightarrow n_i$: `BEST`, $E_{gi}(T_g, n_m)$

Firstly, n_m generates a fresh quantity T_m and encrypts it and its identifier n_m with the Link Key K_{gm}. Then it sends the GW the message (1), containing the previous authenticated T_m, its identifier n_m and the request to be deleted from the network (`DEL`). Upon receiving the message (1), the GW , deletes n_m from the routing tables, updates the routing tables, generates a fresh quantity T_g and informs every node n_i which the routing table must be updated for, by sending it the unicast message (2). The message (2) contains the FLOOD `BEST` message, the identifier of node n_m and the fresh quantity T_g, both encrypted with the link key K_{gi}, shared by the GW and n_i. Upon receiving the message (2), each node n_i updates its routing table.

Assume now that the mobile node n_m wants to add again the network. The node n_m triggers the following protocol:

1. $n_m \rightarrow *$: `ADD`, n_m, T_m
2. $n_i \rightarrow n_m$: `ADD-ACK`, $E_{im}(T_m, T_i, K_i)$
3. $n_m \rightarrow n_i$: `ADD-ACK`, $E_{im}(T_i, K_m)$
4. $n_m \rightarrow GW$: `ADD-REPORT`, $\{E_{1m}(|| < n_i, T_i > \forall n_i \in \mathcal{C}_m)\}$
5. $GW \rightarrow n_i$: `ADD-BEST` $E_{gi}(T_i)$

At the beginning, the mobile node n_m generates a fresh quantity T_m and broadcasts the `ADD` message appending the fresh quantity T_m to it (1). Upon receiving the message (1), every node n_i generates a fresh quantity T_i and appends to the `ADD-ACK` message its cluster key K_i, the fresh quantity T_i and the received T_m, all encrypted with the link key K_{im}. Finally it sends the obtaneid message to n_m (2). When n_m receives the message (2) from the node n_i, it appends to the `ADD-ACK` message its cluster key K_m and T_i, all encryptedwith the link key K_{im}. Finally it sends the message to the node n_i (3). The node n_m sends the message (3) to every node n_i which it received the message (2) from. Eventually n_m appends to the `ADD-REPORT` message all the received fresh quantities T_i. The symbol $||$ indicates the append operation. In practice, n_m sends to the GW the list of all the fresh quantities T_i

and the nodes n_i which it received the message (2), i.e. the nodes belonging to its cluster $\mathcal{C}_m$. Before sending the message (4), n_m encrypts all the couples $< n_i, T_i >$ with the link key K_{gm}.

Key Refreshing

Each node n_i shares with the GW a key, called *Private Key P_i*, which is used by the GW to authenticate specific messages sent from the GW to n_i. In particular, whenever the GW detects a compromised node n_c, it informs all the nodes having n_c in their cluster to refresh their Cluster Key and distribute it to all the nodes of the cluster, except for n_c. The GW sends to the interested nodes the following unicast `REFRESH` message:

- GW $\rightarrow n_i$: E_{P_j}(`REFRESH`, n_c), `REFRESH`, n_c

where n_c indicates the node to be excluded from the next key distribution. Once received the `REFRESH` message, each node n_i triggers again the key distribution phase of the CKDP, excluding the node n_c.

This Key Refreshing mechanism allows a compromised node to perform attacks only until a new cluster key is distributed. The time it can remain active in the network depends on the capability of the IDS to detect quickly an adversary. However, once detected, the adversary can be easily excluded from the cluster and more in general from the network.

Key Predeployment

In the Section 2.6.2 we have described the CKDP, which allows each node to distribute its Cluster Key to the other nodes belonging to its cluster. However, sometimes, it may happen that when the network is deployed, each node is initially located into a well known position so that it knows in advance its *potential* neighbors. A *potential* neighbor i for node j is a node which is located in the transmission range of j, but it could be not acoustically reachable by j because of the sea conditions. This means that i and j become *effective* neighbors only if the conditions of the sea (e.g. temperature, salinity) allow the acoustic communication between the two nodes.

If a node knows in advance its *potential* neighbors, the first phase of the CKDP (i.e. the discovery phase) can be completely avoided, because a node can assume as neighbors its p*potential* neighbors. However, during the key distribution phase, the node is able to recognize its *effective* neighbors.

Note that the assumption that each node can know in advance its *potential* neighbors cannot be applied into Wireless Sensor Networks (WSNs), because, due to the high number of nodes, nodes are randomly deployed in the area. In a UAN, instead, the number of the nodes is low and they can be deployed into a well known position.

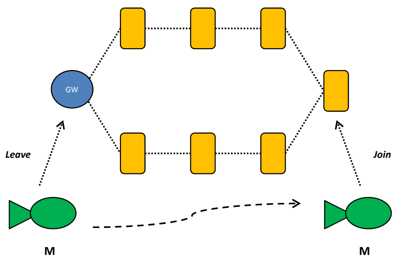
2.6.3 Performance evaluation

In order to test how much overhead the security adds to FLOOD, we have compared SeFLOOD with FLOOD by using the SimPy simulator [30]. We have used realistic results obtained during some experimental tests done in sea [1]. They showed that the average Round Trip Time of the network was about 19 sec., while the maximum transmission rate of the acoustic modems was 500 bits/sec.

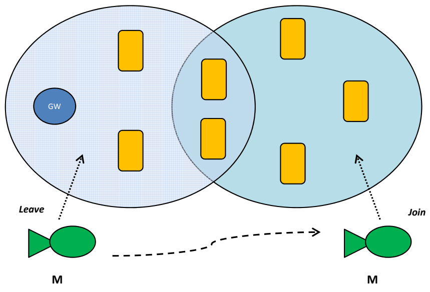
We have considered a network composed of 7 fixed nodes, the GW and 1 mobile node. We have split the simulation duration in two phases: the network discovery phase, which is executed when the simulation starts, and the reconfiguration phase, during which the mobile node moves from a point to another of the network. In particular, firstly it deletes from the network and then adds again to it.

We have simulated two scenarios aimed at studying the performance with different topologies: a *Sparse Scenario* and a *Dense Scenario*. Figure 2.12 shows Sparse Scenario and Dense Scenario respectively. In the Sparse Scenario each fixed node is able to communicate with 2 neighbors, while in the Dense Scenario each node is able to communicate with 5 neighbors.

For each simulation, we have made 15 runs, each lasting 5000



(a) Sparse Scenario.



(b) Dense Scenario.

Figure 2.12:

Parameter	**Value**	
Duration	5000 sec.	
Number of runs	15	
Average transmission range	2 Km	
Maximum transmission power	189 dB re μPa	
Depth	200 m	
Number of nodes	9	
	Sparse Scenario	**Dense Scenario**
Average number of Neighbors	2	5

Table 2.2: Simulation parameters.

sec. We have fixed the transmission range of each node to 2 Km and each node works at a depth of 200 m in the water. Each node is able to transmit a bit with a maximum transmission power of 189 dB re μPa. Table 2.2 resumes the simulation parameters.

We have evaluated the performance of SeFLOOD with respect to two metrics: a) number of messages b) number of bits. Each metric $\%M$ is calculated as the percentage increase of SeFLOOD with respect to FLOOD, by using the following formula:

$$\%M = \frac{M_S - M_F}{M_F} \times 100 \qquad (2.1)$$

where M_S represents the metric calculated using the SeFLOOD protocol, while M_F represents the same metric calculated using the FLOOD protocol.

Number of Messages and Bits Metrics measure how much overhead SeFLOOD adds to FLOOD in terms of total number of sent messages and bits, respectively. Number of Messages metrics measure the number of messages sent in the network. Number of Bits metrics instead measure the effective number of bits sent in the network. Thus even though the number of bits is higher in SeFLOOD than in FLOOD, it may happen that the total number of messages sent in the network is quite similar in SeFLOOD and FLOOD.

Number of Messages metrics include the *Discovery Messages*, the *Add Messages* and the *Delete Messages*. They measure the

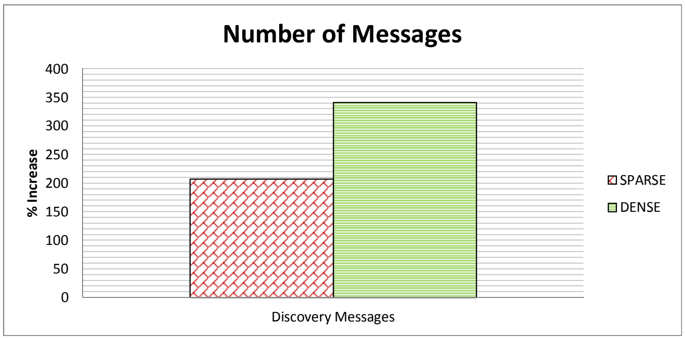


Figure 2.13: Discovery Messages in Sparse and Dense Scenarios.

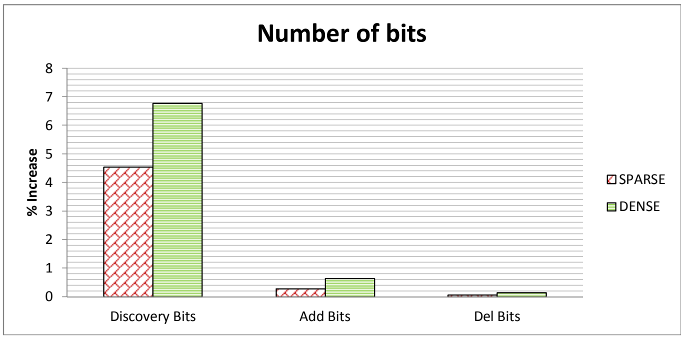


Figure 2.14: Discovery Bits, Add Bits and Del Bits in Sparse and Dense Scenarios.

percentage increase in terms of number of messages introduced by SeFLOOD with respect to FLOOD during the discovery phase, the add phase and the delete phase, respectively.

Bits metrics include the *Discovery Bits*, the *Add Bits* and the *Delete Bits*. They measure the percentage increase in terms of bits introduced by SeFLOOD with respect to FLOOD during the discovery phase, the add phase and the delete phase, respectively.

Figure 2.13 shows the Discovery Messages in Dense and Sparse scenarios. The Figure does not show the Add Messages and Delete Messages, as the number of sent messages by SeFLOOD during the add and delete phase is equal to that of FLOOD during the same phases. Figure 2.14 shows the Discovery Bits, the Add Bits and the Delete Bits in Dense and Sparse scenarios.

With respect to the Discovery Messages and Bits, we note that the gap between SeFLOOD and FLOOD is given by the messages sent by the Cluster Key Distribution Protocol. The percentage increase introduced by SeFLOOD in terms of Discovery Messages is very high (about 200% in Sparse Scenario and about 300% in Dense Scenario). However, each message is very short so that the overhead added by SeFLOOD in terms of number of bits, during the discovery phase is about 4% in Sparse Scenario and about 6% in Dense Scenario. This is due to the fact that FLOOD messages include information about link quality so that they are very big. SeFLOOD, instead, sends only few information, i.e. node identifiers, timestamps and keys.

With respect to the Add and Delete Messages, as already said, SeFLOOD does not add any overhead, because it exploits FLOOD messages. However it adds a little overhead in terms of Add Bits and Delete Bits (1% in the worst case), as Figure 2.14 shows.

2.7 Security Analysis

In this section we analyze the security of our system in terms of the attacks described in the Section 2.3.

2.7.1 Spoofing-based attack against integrity

The spoofing-based attack against integrity is an external attack in which the adversary does not compromise a node. The cryptographic suite, described in Section 2.4, allows the system to protect the communications from such attacks. In fact, the messages exchanged by nodes are encrypted through symmetric cryptography and authenticated through a MAC. Therefore, if the adversary wants to read the message, he must know the keys used for encryption/decryption or he must perform an attack against the encryption algorithm. Furthermore, if he alters the message, the MAC becomes inconsistent so that the message is discarded.

2.7.2 Spoofing-based Denial of Service (DoS) attack

We analyze this attack only with respect to SeFLOOD. SeFLOOD provides a mechanism to reduce the spoofing-based DoS attack which an adversary can make to FLOOD. As the CKDP is executed before FLOOD, only authenticated nodes can partecipate to the FLOOD discovery phase. This means that the adversary can send only the first message of the CKDP, but he cannot perform the attack described in Section 2.6.1.

2.7.3 Node compromision

Assume that an adversary captures a node n_i and accesses all the information which it stores. In particular, the adversary can: a) access the keys, b) send messages in the network, c) receive messages from other nodes.

However, as soon as the IDS detects the node n_i compromision, it informs the GW. In the case of the *small-medium scale system*, the GW simply triggers the S2RP protocol, in order to distribute a new group key K_g. In the case of the *large scale system*, instead, the GW performs the following two operations: a) firstly it triggers the key refreshing mechanism described in Section 2.6.2 in order to distribute the new cluster keys, b) then it triggers the S2RP protocol in order to distribute a new group key K_g.

Anyway, the system remains subjected to an adversary attack during the time employed by the IDS to detect it plus the time needed to distribute the new keys.

2.8 Conclusions

In this chapter we have described a cryptographic suite aimed at protecting the communication among a set of underwater nodes. We have employed this cryptographic suite in two architectures: the *small-medium scale* system and the *large scale* system. We have shown through real experiments made in the sea that the cryptographic suite adds a minimum overhead to the system in both the architectures. At the state of the art, this is the first work describing a solution to the problem of secure cooperation among underwater nodes.

Chapter 3

Delay Tolerant Networks

Quello che si spera si deve credere che possa essere ottenuto; è quanto aggiunge la speranza al puro desiderio.

Tommaso d'Aquino (1225–1274)

3.1 Introduction

A Delay Tolerant Network (DTN) is a network paradigm where connectivity among mobile nodes is not always guaranteed. In order to guarantee messages delivery even in presence of network partitions, a DTN defines specific routing mechanisms to forward messages. The most common are *epidemic-based* [31] and *probability-based* [32, 33]. In the *epidemic-based* mechanisms, many replicas of the same message are transmitted in the hope that at least one reaches the receiver. This mechanism is very expensive in terms of employed resources and it is not applicable when the nodes have limited resources (e.g., mobile nodes when they are battery operated). In the *probability-based* mechanism, the sender forwards the message to the node having the highest probability of successful message delivery. This mechanism relies on the implicit assumption that all the nodes cooperate to message forwarding. Unfortunately, malicious nodes may misbehave and act against the routing mechanism in different ways [34,35]. In this chapter we deal with a specific malicious misbehaviour according to which malicious nodes called *sinkholes* send wrong routing information to attract the as largest number of messages as possible. This *sinkhole attack* may be preparatory to other kinds of attacks, such as dropping all the attracted messages or dropping only some of them. In the former case the sinkhole acts as a *blackhole*, whereas in the latter as a *selective forwarder* [36].

In this chapter we face with the problem of reducing the impact of the presence of blackholes in a DTN and propose an approach based on the concept of *reputation*. In short, upon selecting the next forwarding node, i.e., the node to forward a message to, a node estimates how well a candidate forwarding node has behaved on the basis of past interactions with that possible forwarding node. We call reputation such an estimation. In practice the reputation measures the trustworthiness of a node. The lower the reputation of a node, the higher the chance that the node is blackhole. The chances of selecting a node as a forwarding node are weighted by means of its reputation. Therefore a node having a low reputation is unlikely chosen as a forwarding node.

In our approach every node maintains a *local* notion of reputation. We make this choice in order to avoid the overhead and the technical complications related to maintaining a global shared notion of reputation [37]. Intuitively, reputation is maintained by means of three mechanisms, namely *acknowledgments*, *nodes list* and *aging*. Every message carries the list of forwarding nodes the message has traversed. Upon receiving the message, every node can update the reputation of the forwarding nodes specified in the list. Furthermore, upon handing a message to a forwarding node, the sender starts waiting for an acknowledgment from the ultimate destination. If the acknowledgement arrives, the sender increases the reputation of that forwarding node. Finally, reputations are periodically aged, i.e, decreased. The challenge here is to *adapt* the aging period to the highly changing delays of a DTN. We succeed in this by using a properly designed Kalman filter [38].

The reputation management mechanism employs both data messages and acknowledgment messages. In particular, the format of data messages has been extended to accommodate additional information concerning reputation management while an acknowledgment message is a unicast short message. It follows that, differently from other systems [39,40], the proposed reputation management protocol does not need to resort to communication expensive mechanisms for reputation updates dissemination based on broadcast/multicast communication.

In this chapter we apply the reputation-based approach to the Context Aware Routing (CAR) protocol, a probability-based routing protocol previously defined in [32]. CAR has no protection mechanisms against blackholes. We show that by augmenting CAR by means of the proposed reputation mechanism, the delivery ratio increases from 20% up to 60% according to the considered scenario.

We call *Reputation-based CAR* (RCAR) the resulting protocol. An early version of RCAR has been already proposed in [41]. With respect to that version, here we propose the following enhancements: a) we protect the integrity of the reputation management information carried by messages; b) we add a mechanism to dynamically determine the reputation aging period; c) we compare the performance of RCAR with that of T-ProPHET from several viewpoints including delivery ratio, attraction ratio, and delivery delay. T-ProPHET is another reputation-based protocol for DTN [39] that we have chosen for comparison with RCAR because it is both quite recent and the most similar to RCAR. Performance evaluation tests show that in the best case RCAR is able to reach a delivery ratio of about 80% while T-ProPHET in the best case reaches a delivery ratio of about 70%. Furthermore RCAR works better than T-ProPHET when the number of blackholes in the network is low. In any case, RCAR is *more effective* than than T-ProPHET, that is, the improvement that RCAR makes to CAR always outperforms the improvement that T-ProPHET makes to ProPHET.

The chapter is organized as follows. Section 3.2 describes related work. Section 3.3 describes the network assumptions whereas Section 3.4 describes RCAR. Section 3.5 reports performance evaluation by means of simulation. Finally, Section 3.6 describes our conclusions and future work.

3.2 Related Work

The issue of secure cooperation in DTNs is a recent challenge. Notwithstanding a lot of work exists on this field, addressing different aspects of secure cooperation [34, 35]. The most common threats to DTNs are: *selfish nodes* and *blackhole nodes*. A selfish node is a node which is reluctant to cooperate in message forwarding, in order to save its own energy. At the state of art, the most important protocols acting against selfish nodes try to reduce the selfishness by incentiving the selfish nodes to cooperate [42–44].

A blackhole is a node which drops all the received messages. At the state of art, the most important protocols acting against blackholes can be classified as : a) *reputation-based*, b) *reference-based*, and, finally, c) *replication-based*. In *reputation-based systems* [37, 39], each node observes the behavior of other nodes and assigns each of them a reputation, which measures how much a node is well behaving. The routing of messages is done on the basis of the reputation: the lower the reputation the lower the probability that a node is chosen as next hop (forwarding node) for a message.

In *reference-based systems* [45–47], each node wanting to forward a packet gives its *references* to its neighbors. A reference is a piece of evidence specifying that a node has cooperated to message forwarding. On the basis of references of a given node j, a node i can decide whether to forward its messages to j or not. In *replication-based systems* [40,48], secure cooperation is achieved by sending many replicas of the same message.

In the Secure Reputation-based Dynamic Window Scheme (SReD) [37], messages are forwarded to nodes having the highest reputation. The basic idea of SReD is to provide recommendations to each node based on the opinions of the other nodes. In practice, if a node a wants to calculate the reputation of a node b, it asks its adjacent nodes to give it their opinion on b, except node b. Then, a calculates the reputation for b as the sum of its local opinion on b and that given by other nodes. In SReD reputations are shared among all nodes. The SReD protocol relies on the strong assumption that every node has a trusted hardware. In fact, if there is not a trusted hardware, a node could send to the other nodes bogus recommendations. With respect to SReD, we propose a protocol in which the reputation is a local notion so that there is not the need of a trusted hardware.

In [39], the authors describe T-ProPHET, a reputation assisted data forwarding mechanism for opportunistic networks. Each node sends its messages to the node having the higher reputation. Upon receiving a message, a destination builds a *Positive Feedback Message* (PFM), which contains information useful to update the reputation of the carriers. The PFM is sent through epidemic routing, in order to reach more quickly the sender. When the sender receives the PFM, it can update the reputations of the carriers contained in it. The epidemic routing mechanism allows a sender to update reputations quickly, but it also adds traffic overhead. An extensive comparison with T-ProPHET is reported in Section 3.5.

Y. Ren *et al.* propose a mechanism to detect blackhole nodes based on *packet exchange recording* [46]. When two nodes meet, they exchange their respective history records, containing the list of all node they have encountered. All the history records are authenticated through digital signatures. Comparing the other node's history with its own local history, each node is able to determine whether the other node has forwarded all the messages or not. This mechanism is called *sanity check*. If a node has not forwarded all the messages, the other nodes can detect it and classify that node as a blackhole. This mechanism is able to recognize a high number of blackhole nodes. However, this technique is not indicated when contact time between two nodes is short, because there may be no time to exchange long histories.

The same authors propose an improvement of the previous technique [47]. Instead of having every node to perform the sanity check, there is a single *ferry node* that performs it. Periodically, the ferry node visits all the nodes and asks them their history records. The ferry node classifies a node as a blackhole if the history of that node contains messages that the other nodes' histories do not contain. While this technique is very efficient, the ferry node features a single point of failure. If an adversary is able to compromise of knock down the ferry node, all the system is compromised.

M. Chuah *et al.* propose a dynamic replication mechanism, where the number of generated message copies, called *redundant factor*, is calculated dynamically, on the basis of the current delivery ratio [40]. In particular, given a traffic flow originating in a sender and ending in a destination, this latter node measures the delivery ratio for the flow and sends it to the sender. The sender dynamically adjusts the redundancy factor according to the received delivery ratio. The main drawback of this protocol consists in the large number of messages caused by replication.

3.3 System Model

We consider a Delay Tolerant Network (DTN) composed of several wireless nodes moving inside a fixed area. For instance, a node may be a device held by a human or embedded in a bus. In the network, messages are forwarded according to the following strategy. If a route already exists between the sender s and the receiver r, the message is forwarded using a standard routing protocol for ad hoc networks (e.g. DSDV). We call *synchronous routing* this mechanism. If the route does not exist, the sender uses an *asynchronous routing mechanism* according to which the message is forwarded to the forwarding node c having the highest chance of successful message delivery. The node c stores the message in its local buffer until it either establishes a route with the receiver r or encounters another forwarding node c' having a higher chance of message delivery to the destination. In the former case, c delivers the message to r by means of the synchronous routing. In the latter case, c applies again the the asynchronous routing and forwards the message to c'. This routing process continues until the message eventually reaches its final destination r. Notice that a buffer has a limited size, therefore, when it gets full, the arrival of a new message causes a message loss. DTN mechanism inherently loses messages, unless the buffer size is unlimited. DTN management systems differ on buffer management and message replacement policies [49].

A crucial issue in asynchronous routing is how to select forwarding nodes [2]. Many solutions have been proposed [33, 50, 51].

In this work we refer to the selection algorithm proposed by CAR (Context Aware Routing) [32]. In CAR, each node calculates its own *delivery probability*, i.e., the chance of successful message delivery on the basis of its own *context* information. A node context is defined as the set of *attributes* that describe the aspects of the system that can be used to drive the process of message delivery (e.g., mobility of node or battery level). Each node estimates its own delivery probability by means of a *Utility Function* $U(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} w_i U_i(x_i)$ where x_i represents an attribute, $U_i(x_i)$ the utility function calculated over x_i, and w_i is the significance weight reflecting the relative importance of each attribute.

A node periodically computes an estimation of its own delivery probability and then broadcasts it, together with the routing information necessary to build routing tables (for synchronous routing), to the other nodes which it is able to reach through the synchronous routing. CAR employs DSDV as synchronous routing protocol. In CAR a node selects the forwarding node c among those reachable through the synchronous routing as that having the highest chance of message delivery, i.e., the node having the greatest value of U_c.

3.3.1 Adversary Model

A DTN could be affected by many threats [4, 48]. In this chapter we assume that a node under the adversary control may behave as a *blackhole* [36]. This means that the malicious node strives to appear attractive for the other nodes as far as the routing algorithm is concerned. Then, upon receiving messages to forward—both synchronously or asynchronously—the blackhole drops them. In order to surreptitiously increase its own attractiveness, a blackhole may act in two ways. Either by modifying in transit routing information. Alternatively, a blackhole b may exploit the CAR mechanism for forwarding node selection and surreptitiously disseminate a falsely high value of the utility function, i.e., $U \to 1$, so inducing other nodes to select it with high probability. We assume that blackholes do not collude.

Incidentally, we would like to point out that the behaviour of a selfish node [36]—a node which uses the routing service but which does not want to spend its own resources to cooperate towards that service—is largely opposite to a blackhole's one. Actually, in contrast to blackholes, selfish nodes strive to appear unattractive for the other nodes in order to be not selected as a forwarding nodes. However, if selected, they will forward messages. In the CAR context, a selfish node will likely broadcast small fake values of its utility functions, i.e., $U \to 0$. It follows that countermeasures against selfish nodes tend to be very different from those of blackholes. In this chapter we will not address selfish nodes and do not mention them any further. Countermeasures against them will be the subject of future work.

3.4 RCAR

In this section we describe RCAR (Reputation-based CAR), an extension of CAR that is based on the concept of *reputation* and able to limit the effect of the presence of blackholes.

In Section 3.4.1 we introduce the concept of reputation in CAR. Then in Sections 3.4.2 and 3.4.3, we give some intuitions about its implementation and how nodes reputation gets updated.

3.4.1 The concept of Reputation

Every node estimates how well another nodes behaves regarding the forwarding of messages. We call *reputation* (R) such an estimation. The range of R is $[0, 1]$. The lower R, the higher the probability that the node is a blackhole. R is a local notion, because it is calculated by each node on the basis of its own network experience. In other words, there is no global consensus on the reputation of a given node. This is in order to save the node energy and avoid both the traffic overhead, and the technical complications related to maintain such a consensus. By R_{ij} we denote the *reputation* of node i calculated by node j. Every node j calculates the *reputation* R_{ij} of every node i it meets, as described below.

A node uses *reputation* to contrast blackholes as follows.

Definition 1 (Local Utility Function) *Let U_i be the Utility Function of i and R_{ij} be the reputation of node i at node j, then the* Local Utility Function*, L_{ij}, is given by:*

$$L_{ij} = R_{ij} \times U_i \tag{3.1}$$

Intuitively L_{ij} represents how much node j considers node i able to forward messages. Node j uses the local utility function to choose a node. In practice, it chooses as the forwarder of a message the node i having the highest value of L_{ij}. The rational basis of this choice is the following. Assume that a node i is a blackhole. Thus node j assigns a low reputation value to node i, i.e., $R_{ij} \to 0$. It follows that the value of $L_{ij} \to 0$ and thus j does not select i as a forwarding node.

More formally, let D be the event "node i delivers a message" and B the event "node i is not blackhole". The probability of successful message delivery $P(D)$ is given by the Bayes theorem:

$$P(D) = \frac{P(B)P(D|B)}{P(B|D)} \tag{3.2}$$

where $P(D|B) = U_i$, where U_i is the Utility Function of node i. This is because if a node is not blackhole (event $P(B) = 1$), the event D happens with a probability given by the chances of the node to forward a message (i.e. U_i). Furthermore, $P(B) = R_{ij}$, where R_{ij} is the reputation given by the node j to the node i. This is because a node is not blackhole with a probability equal to its reputation. Thus we have:

$$P(D) = \frac{R_{ij}U_i}{P(B|D)} \tag{3.3}$$

but $P(B|D) = 1$ because if i forwards messages, i is not blackhole. Therefore,

$$P(D) = U_i \times R_{ij} \tag{3.4}$$

where $P(D) = L_{ij}$.

Figure 3.1 shows the flow diagram of the algorithm used by RCAR to select a forwarding node. Assume that a node s, the

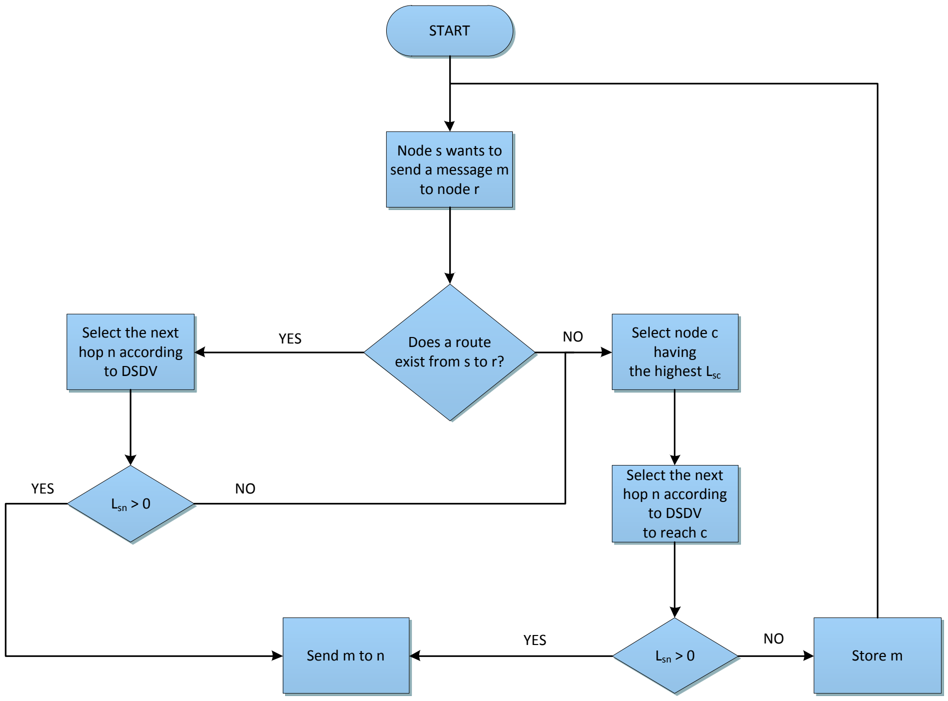

START
Node s wants to send a message m to node r
Does a route exist from s to r?
YES
NO
Select the next hop n according to DSDV
Select node c having the highest L_{sc}
YES
$L_{sn} > 0$
NO
Select the next hop n according to DSDV to reach c
Send m to n
YES
$L_{sn} > 0$
NO
Store m

Figure 3.1: Flow Diagram of the RCAR algorithm.

sender, sends a message m to a node r, the receiver. If a route exists between s and r, the node s exploits the synchronous routing, otherwise it exploits the asynchronous routing as described in Section 3.3. When the synchronous routing is available, the node s selects the next hop n to reach r according to the DSDV protocol, as in CAR. Once selected the next hop n, the node s checks whether $L_{sn} > 0$. If this is the case, it means that the node n is not blackhole, so that the node s sends it the message m. Otherwise, if $L_{sn} = 0$, the node s tries to exploit the asynchronous routing. As already said, the asynchronous routing is used also when a route from the sender to the receiver does not exist. In the case of asynchronous routing, the sender s selects the node c having the highest L_{sc}. In order to reach the node c, the node s exploits the synchronous routing, that is, it selects the next hop n

according to the DSDV protocol, in order to reach c. Once received the message m, the node c stores it in its local buffer. As before, node s sends the message m to n only if $L_{sn} > 0$, otherwise it stores m in its local buffer. Periodically both the nodes s and c try to send the messages contained in their local buffer by repeating the previously described mechanism.

We have made the choice that node j forwards a message to a given next hop n if and only if $L_{nj} > 0$ because at the same time we want both to contrast blackholes and preserve the advantages introduced by the use of synchronous routing.

Note that when the synchronous routing is used, a node s sends the message to a node n only if $L_{sn} > 0$, while when the asynchronous routing is used, the node s selects the node n having the largest value of L_{sn}. This is due to the fact that in the synchronous routing the Utility Function of the node n does not influence the routing so that we can assume that $U_{sn} = 1$. This means that the value of L_{sn} depends only on the reputation. In this case, the message must not be sent to a node if it is a blackhole. The node n is blackhole if $R_{sn} \leq \theta$, where θ is a threshold to consider a node blackhole. Without loosing in generality, we can assume $\theta = 0$, because if $R_{sn} = 0$ the node n is certainly blackhole. In contrast, in the asynchronous routing, $0 \leq U_{sn} \leq 1$ and thus the value of L_{sn} depends also on it. This means that it is not sufficient to check whether $L_{sn} > 0$ to select the forwarding node, because the protocol must also take into account the node having the largest Utility Function.

3.4.2 The Reputation Update Protocol

The *Reputation Update Protocol* (RUP) is the mechanism used by RCAR to update reputations of nodes. Integrity and authenticity of RUP information is protected by means of digital signatures. In particular, we assume that every node has a pair (public key, private key) and a certificate binding its identifier to its public key signed by a Certification Authority (CA) trusted by all the nodes. By $\langle x \rangle_a$ we indicate the digital signature of node a on quantity x. Deploying a Public Key Infrastructure (PKI) is in general a problematic task due to the problems connected to certificate revocation and, more in general, the need of an online CA. Solutions have been proposed in [52–55]. However, at the state of art, PKI is the common solution used in DTNs to authenticate messages and provide their integrity [39, 44, 45].

The basic idea behind the RUP is based on the following observation. If a node d receives a message, then all nodes the message passed through are well behaving or, otherwise, d would have not received the message. This means that, upon receiving a message, node d can increase the reputation of all nodes the message passed through, provided we can keep track of them.

We keep track of nodes a message has passed through as follows. Every message m carries the the list of identifiers of nodes the message has passed through. We call *node list* (*nlist*) such a list. Upon receiving message m, a node adds itself to the *nlist*. A node adds itself only once, even though a message passes through that node many times. A malicious node could modify the *nlist* and add identifiers of nodes the message has not passed through in order to increase the reputation of other malicious nodes. In order to avoid such modifications, the message carries also a list of digital signatures $\sigma_1, \sigma_2, ...\sigma_{i-1}$ that prove that the message has actually passed through the nodes specified in the node list. We call *slist* such a list. The digital signature σ_i establishes an unbreakable link between the node c_i receiving m and the node c_{i-1} from which it has received m. In practice through σ_i we are sure that the message m has gone from c_{i-1} to c_i without passing throughout any other intermediate node.

The length of the *slist* influences both the message length and thus the RCAR communication overhead as well as the message processing time and thus the RCAR processing overhead. However, as we will show in Section 3.5, the average number of nodes a message passes through is small, namely about 1.5 on average. This means that the overhead added by the digital signatures is practically negligible.

Let us suppose that a sender s sends a message:

$$m = (mid, p, d, t_s, nlist, slist) \tag{3.5}$$

where *mid* is the unique identifier associated to each message, p is the message payload, d is the destination node and t_s is the time at which the message is sent. Initially *nlist* and *slist* are empty. They are iteratively constructed as follows. Suppose that the forwarding node c_1 receives the message from s. It updates the *nlist* and the *slist* as follows:

$$nlist = c_1 \tag{3.6}$$

$$slist = \sigma_1 \tag{3.7}$$

In the most general case, node c_1 forwards the message m. When m passes through the i-th forwarding node c_i, $i > 0$, let us assume that $nlist = \{c_1, c_2, .., c_{i-1}\}$ and $slist = \{\sigma_1, \sigma_2, ...\sigma_{i-1}\}$. Then, c_i updates *nlist* and *slist* as follows:

$$nlist \leftarrow nlist || c_i \tag{3.8}$$

$$slist \leftarrow slist || \langle \sigma_{i-1}, c_i \rangle_{c_i} \tag{3.9}$$

where $||$ indicates the append operation and $\langle x \rangle_{c_i}$ indicates the digital signature made by node c_i on content x.

Upon receiving message m, the receiver d verifies i) the digital signatures contained in the *slist* and ii) if the list of nodes contained in the *nlist* corresponds to that contained in the *slist*. If the check is successful, it extracts the list of nodes from the *nlist* and increases the reputation of all those nodes.

The described protocol allows only the receiver d to update the reputation of the nodes. This basic mechanism can be improved, using two additional mechanisms: ack-based and step-by-step. With the *ack-based*, the destination node d builds an *acknowledgment message* $ack = (mid, t_s, clist, slist)$, and sends it back to the sender s. The *nlist* and *slist* of the acknowledgment are initialized with the *nlist* and the *slist* of the original message. Furthermore, during the ack forwarding process, the *nlist* and the *slist* are updated as described before for standard messages. The *ack* behaves as a standard message except it is not acknowledged in its turn. The *ack* message may follow a different route than the original message m. In practice, each node forwarding the *ack* adds itself in the *nlist* and *slist*, only if it is not already contained in them. Upon receiving the *ack* message, the original sender s verifies the digital signatures contained in the *slist* and the correspondence between the nodes contained in the *nlist* and the *slist*, and if this check is successful, it increases the reputation of the nodes contained in the *nlist*.

The *steb-by-step* mechanism is an improvement of the previous one. All the nodes traversed by the message and the corresponding *ack* extract the corresponding *slist* and *nlist*, verify the digital signatures contained in the *slist* and the mutual consistency of *nlist* and *slist*. Then they update the reputation of the nodes contained in the *nlist*. As it turns out, the reputation update process involves only certain nodes, namely those receiving the original message and the *ack*. We have made this choice to keep low the DTN management traffic. In the performance evaluation tests described in Section 3.5, we have used the *step-by-step* technique.

3.4.3 The aging mechanism

The just described mechanism allows every node to increase the reputation of all the nodes contained in the *nlist* of a message that passes through the node. However, if a message gets lost, a node has no means to know whether a blackhole has dropped it or another situation has occurred (e.g. message dropped for buffer overflow or TTL elapsed). Furthermore, even though the node could know that a message has not been delivered, it could not know which node along the path has misbehaved. Thus, in order to cope with blackholes, a mechanism based on *aging* is used to decrease the reputation of all the nodes. In practice, periodically a node decreases the reputations of all the nodes. This choice is done to have a conservative policy, because a node does not know which node has dropped the message.

We assume that the reputation increases and decreases linearly. That is, reputation R may be increased by a positive non-zero quantity X, i.e. $R \leftarrow \max(1, R + X)$, or decreased by a positive non-zero quantity Y, i.e., $R \leftarrow \max(0, R - Y)$. Quantities X and

Y may be different if we wish that the reputation increases and decreases at different speeds. In an optimistic policy, node i may initially consider j trusted, i.e., initially $R_{ji} = 1$. In contrast, in a conservative policy, node i may not initially trust j at all, i.e., intially $R_{ji} = 0$. Intermediate policies can be defined.

In the aging mechanism, it is important the choice of the value of the *decrease period T*. On one hand, a too large value of T generates a high number of false negatives, because reputation of blackholes is decreased too slowly. On the other hand, a too small value of T, instead, produces a high number of false positives, i.e. well-behaving nodes are classified as blackholes because their reputation decreases too quickly before acknowledgments come back to the sender.

In order to fulfil the requirements of a DTN, the decrease period T cannot be fixed, because DTN conditions change. The decrease period T could be expressed as a function of the Round Trip Time (RTT). Upon sending a message, the sender attaches the message a timestamp specifying the moment in which the message was originated. So doing, upon receiving the corresponding acknowledgment, the sender node is able to calculate the RTT of the message. This means that whenever a node receives an acknowledgment, it can update the decrease period. In this way the decrease period follows the RTT and is updated dynamically according to network conditions. However, due to the nature of DTN, an acknowledgment may get lost so that the decrease period is not updated correctly. In order to dynamically update the decrease period even when the RTT is not (always) available, we employ a mechanism to predict the future values of RTT on the basis of the past history. We us Kalman filters [38]. Kalman filters were originally thought for automatic control systems theory. They are able to estimate the next state of a dynamic system on the basis of some observations. The advantage of using Kalman filters is based on the fact that they are able to predict the next current state even when the current observation is not available. Furthermore, they do not require to store the whole past history of the system so that they can be used also by resource constrained vehicles.

3.5 Performance Evaluation

In this section we evaluate performance of RCAR via simulation. RCAR has been compared to T-ProPHET [39], a reputation-based protocol for DTN that is both very recent and the most similar to RCAR. Furthermore, we also compare RCAR to Epidemic Routing (ER) [31]. In brief, ER relies on the idea that both the sender and any forwarding node forwards a message to all the nodes it meets until the message arrives at destination. However, in order to avoid overwhelming the network, a forwarding node never forwards a message twice.

In our evaluation we also compare the improvement introduced by RCAR on CAR [32] to the improvement introduced by T-ProPHET on ProPHET [33]. CAR and ProPHET have been already compared in [32]. In the absence of blackholes CAR outperforms ProPHET from several standpoints. In particular, CAR achieves a higher delivery ratio, a smaller number of sent messages and a smaller propagation delay than ProPHET. This is due to the fact that CAR has a more efficient mechanism to update delivery probabilities than ProPHET. However, as we are going to describe below, this update mechanism allows blackholes to attract a larger number of messages in CAR than in ProPHET. It follows that CAR is more susceptible to blackholes than ProPHET.

In order to make performance evaluations and comparisons, we have simulated RCAR, CAR and ER using the OMNet++ simulator, while for ProPHET and T-ProPHET we have taken results from [39]. In order to compare CAR, RCAR, ER, ProPHET and T-ProPHET in the same conditions, we have considered the same simulation scenarios as described in [39]. In particular, we have created blackholes and well-behaving nodes moving around a fixed area. Only well-behaving nodes send/receive messages while blackholes have been implemented as nodes distributing a delivery probability $U_b = 1$ for all the destinations. Once attracted a message, a blackhole drops it. In the case of ER, it does not employs delivery probability, so that a blackhole is a node which drops received messages.

We have set the number of well-behaving nodes to 20. We have

Parameter	**Value**
Network area	$1km^2, 2km^2, 3km^2$
Number of well behaving nodes	20
Number of blackholes	2,6,10,14
Communication range	25 m
Node speed	0–5 m/sec
Data generation rate	1 message/second
Simulation time	8900sec.
Warmup period	300 sec.
RTTI	3 sec
Number of runs	20
Buffer Size	1000 messages
Reputation increment	0.1
Reputation decrement	0.05
Initial reputation	1

Table 3.1: Configuration parameters.

performed two sets of simulations: a) we have varied the network area size (1000m $\times$ 1000m, 2000m $\times$ 2000m and 3000m $\times$ 3000m), while keeping fixed the number of blackholes (6 blackholes), b) we have varied the number of blackholes (2,6,10,14) while keeping fixed the network area size (1000m $\times$ 1000m). Each node has a communication range equal to 25 m, with a variable speed from 0 to 5 m/sec. Each node generates a message with a rate of 1 message per second and has a buffer size of 1000 messages. For each simulation we have performed 20 runs, each lasting 8900 sec. In each run we have set the warm up period to 300 sec. The warm up period is the initial time needed to build the routing tables for synchronous routing. We have set the routing table transmission interval (RTTI) to 3 sec. The RTTI represents the interval of retransmission of the routing tables. In practice, each node retransmits its routing table to the other nodes every RTTI seconds. We have set the initial reputation to 1, the reputation increment X to 0.1, while the reputation decrement Y to the half of the reputation increment (i.e. 0.05). Table 3.1 resumes configuration parameters.

We have measured the following metrics:

- *Node List Length.* It is the number of nodes contained into an acknowledgment, when it arrives to the original sender of the message.

- *Delivery Ratio.* It is the ratio between the number of received messages and the number of sent messages. Acknowledgements do not contribute to the delivery ratio. In practice, it measures the fraction of messages the network is able to deliver in the presence of blackholes. Ideally the delivery ratio should be equal to 1.

- *Attraction Ratio.* It is the ratio between the number of attracted data message by the blackholes and the number of sent data messages. Acknowledgments do not contribute to the attraction ratio, although blackholes attract acknowledgments too. In practice, the attraction ratio measures the fraction of messages the blackholes are able to attract. Ideally the attraction ratio should be equal to 0.

- *Average Delay.* It is the time between the generation of a message and its delivery to the final receiver. It is calculated as the average of all the messages received, included acknowledgments. Ideally the average delay should be equal to 0.

- *Total Number of Sent Messages.* It is the total number of messages sent in the network, including routing messages and acknowledgments.

3.5.1 Node List Length

Figure 3.2 shows the distribution of the Node List Length versus the network area sizes. Each tick on the x-axis represents the node list length, that is, the number of nodes contained in the acknowledgment. We note that for about the 50% of messages the node list contains 1.5 nodes and only sporadically the node list length is two or longer.

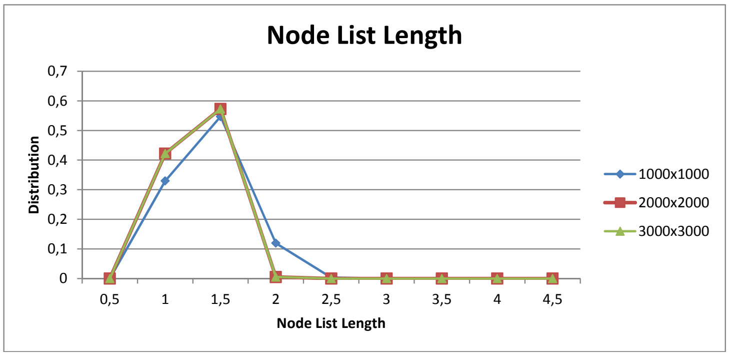


Figure 3.2: Nodes List Length Distribution.

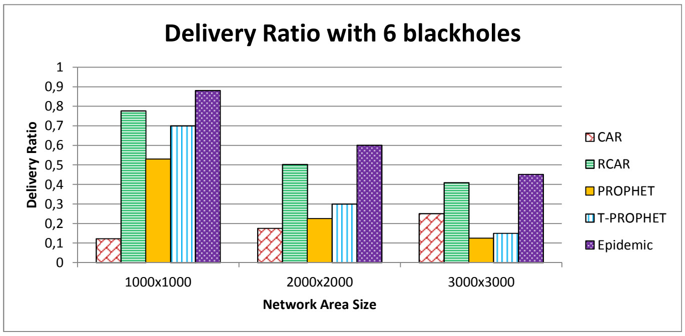


Figure 3.3: Delivery Ratio vs Network Area Size.

3.5.2 Delivery Ratio

Figure 3.3 shows the delivery ratio of RCAR, CAR, ER, ProPHET and T-ProPHET with respect to different network area sizes. In all the cases ER outperforms all the other protocols, paying the cost of a very high number of sent messages, as explained in Section 3.5.5. However, its delivery ratio decreases while increasing the network area size. This is due to the fact, that when the network area size increases, the probability that two nodes meets is low so that their buffers are not emptied. This causes buffer overflow and many messages get lost.

In all the cases RCAR outperforms T-ProPHET. It is interesting to note that when the network area size is 1000 m $\times$ 1000 m

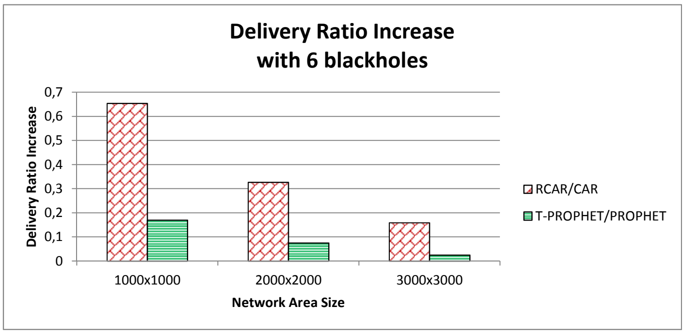


Figure 3.4: Delivery Ratio Increase vs Network Area Size.

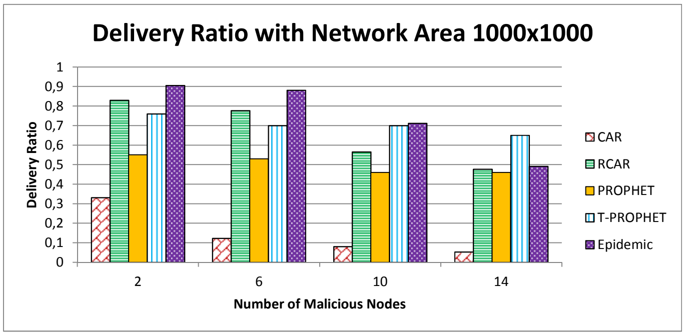


Figure 3.5: Delivery Ratio vs Number of malicious nodes.

and 2000 m × 2000 m, CAR reaches the lowest delivery ratio. This is due to the fact that when the network is small, in CAR the synchronous routing works, while ProPHET has not a synchronous routing. If the synchronous routing is very efficient when there are not blackholes [32], it does not work as expected in presence of blackholes, because as the network is small, the probability to meet a blackhole is high so that many messages are sent to them and they are dropped. When the network area size increases (3000 m × 3000 m), CAR outperforms both T-ProPHET and ProPHET. This is due to the fact that when the network area is large, CAR uses the asynchronous routing. As the network area is large, the probability to meet a blackhole is smaller so that the asynchronous

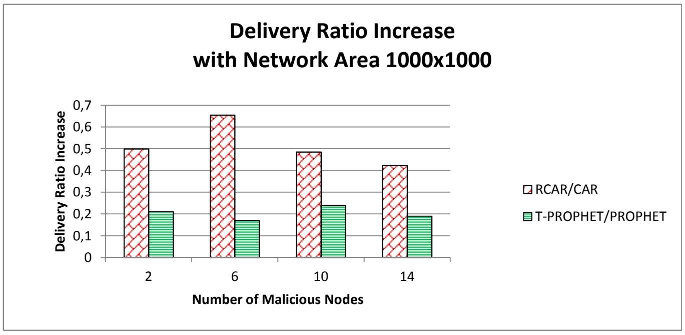


Figure 3.6: Delivery Ratio Increase vs Number of malicious nodes.

routing works quite efficiently.

Figure 3.4 shows the performance increase in delivery ratio of RCAR (T-ProPHET) on CAR (ProPHET) with respect to differenct network area sizes. The performance increase in delivery ratio is calculated as the difference between the delivery ratio of RCAR (T-ProPHET) and that of CAR (ProPHET). The greater the performance increase the greater the increase introduced by RCAR (T-ProPHET) on CAR (ProPHET). We note that in all the cases the performance increase in delivery ratio introduced by RCAR on CAR is higher than that introduced by T-ProPHET on ProPHET.

Figure 3.5 shows the delivery ratio of RCAR, CAR, ER, ProPHET and T-ProPHET with respect to an increasing number of blackholes. ER outperforms all the other protocols, except in the scenario with 14 blackholes where it reaches a delivery ratio comparable to that of RCAR and CAR and smaller than that of T-ProPHET. This is due to the fact that when the number of blackhole increases, the probability that a node meets a blackhole is larger and consequently is larger the probability that a node forwards replicas of messages to balckholes. As a results many messages are dropped and the delivery ratio decreases.

When the number of blackholes is lower (2 and 6) RCAR outperforms all the other protocols whereas when the number of blackholes is high (10 and 14) T-ProPHET outperforms all the other protocols. However, in all the cases, the performance increase in

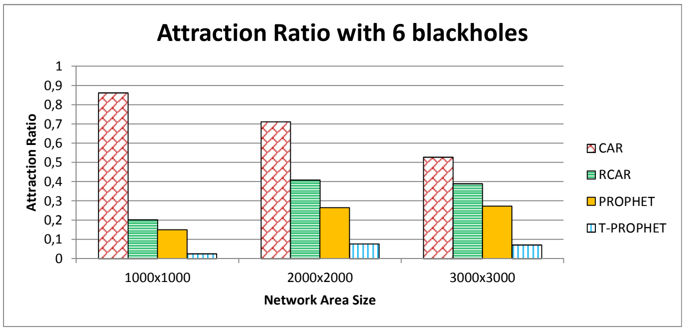


Figure 3.7: Attraction Ratio vs Network Area.

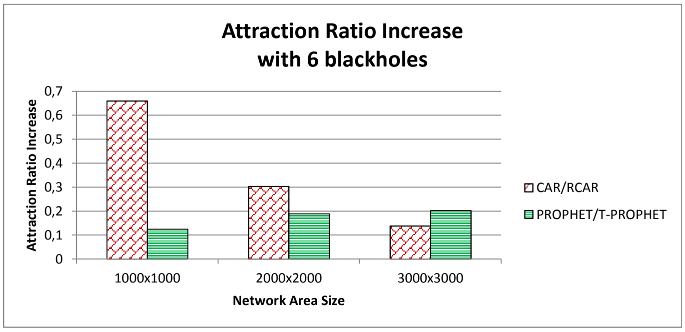


Figure 3.8: Attraction Ratio Increase vs Network Area.

terms of delivery ratio introduced by RCAR on CAR is higher than that introduced by T-ProPHET on ProPHET (Figure 3.6).

3.5.3 Attraction Ratio

Figures 3.7 and 3.9 show the attraction ratio of RCAR, CAR, ProPHET and T-ProPHET with respect to an increasing network area size and an increasing number of blackholes respectively. We do not show the values for ER, because in ER a blackhole simply drops the messages it receives without attracting them. In ER, in fact, a blackhole has no means to attract a message, as the routing mechanism sends messages to all the nodes.

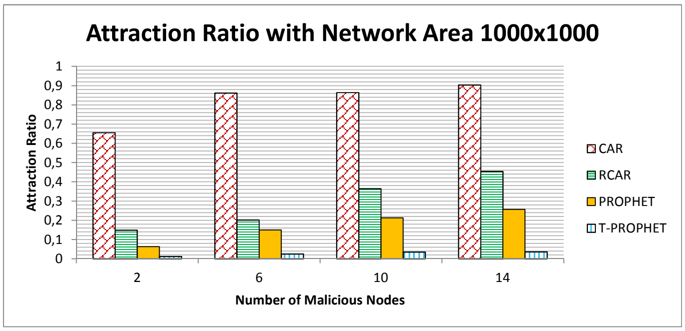


Figure 3.9: Attraction Ratio vs Number of malicious nodes.

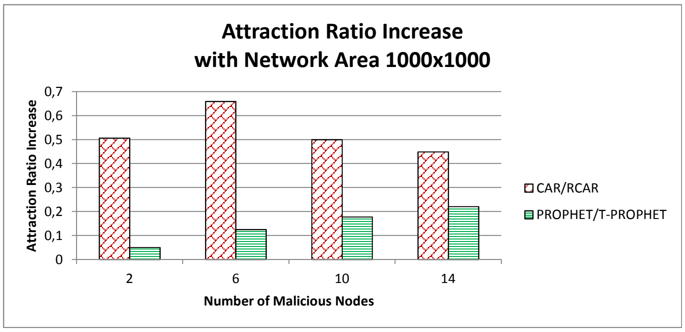


Figure 3.10: Attraction Ratio Increase vs Number of malicious nodes.

We note that the attraction ratio of CAR is higher than that of ProPHET in all the scenarios so that also RCAR attracts more messages than T-ProPHET. However this is due to the different mechanisms adopted by CAR and ProPHET to update delivery probabilities. In fact in CAR a node calculates another node delivery probability exactly as the delivery probability sent by that node. In particular, if a blackhole sends a delivery probability $U_b = 1$, every node i receiving that delivery probability, assumes that the blackhole has a delivery probability $U_{ib} = U_b = 1$. In ProPHET, instead, a node i calculates the delivery probability of a node j as the combination of three factors: a) the previous delivery probability assigned by i to j, b) the probability for i to meet j, which is calculated locally by i, c) the delivery probability sent by j to i. This means that if a blackhole sends a delivery probability $U_b = 1$ to a node i, the node i calculates the delivery probability U_{ib} for that blackhole keeping into account the described factors so that in general $U_{ib} \leq U_b$. It follows that in CAR the probability to choose a blackhole is higher than in ProPHET.

It is interesting to note that when the network area size increases, the attraction ratio of CAR decreases, while that of ProPHET increases. This is due to the different mechanisms used by CAR and ProPHET to update the delivery probabilities. When the network area size increases, the probability that a node meets a blackhole is lower. However, in ProPHET the lower the probability to meet a carrier, the higher the influence of the delivery probability sent by that carrier. In other words, if a blackhole sends a delivery probability $U_b = 1$ to a node i, if the network area size is small, node i calculates $U_{ib} \leq U_b$, while if the network area size is large, node i calculates $U_{ib} = U_b$ [33]. In CAR, instead, the less the probability to meet a carrier, the less the probability that the carrier is chosen as forwarder.

Figures 3.8 and 3.10 show the attraction ratio increase of RCAR over CAR and T-ProPHET over ProPHET. The attraction ratio increase is calculated as the difference between the attraction ratio of CAR (ProPHET) and RCAR (T-ProPHET). The greater the attraction ratio increase the greater the improvement introduced by the RCAR (T-ProPHET) over CAR (ProPHET). RCAR outperforms T-ProPHET in the scenario with a small network area, while when the network area size increases T-ProPHET outperforms RCAR.

3.5.4 Average Delay

Figure 3.11 and 3.12 show the average delay of RCAR, CAR, ER, ProPHET and T-ProPHET with respect to an increasing network area size and an increasing number of blackholes, respectively. We note that ER reaches a smaller average delay than T-ProPHET and ProPHET and a greater average delay than RCAR and CAR.

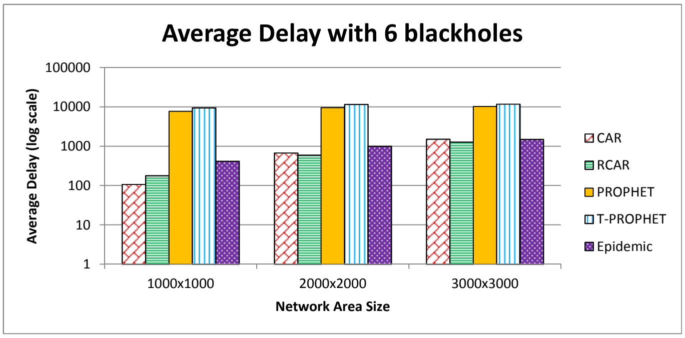


Figure 3.11: Average Delay vs Network Area.

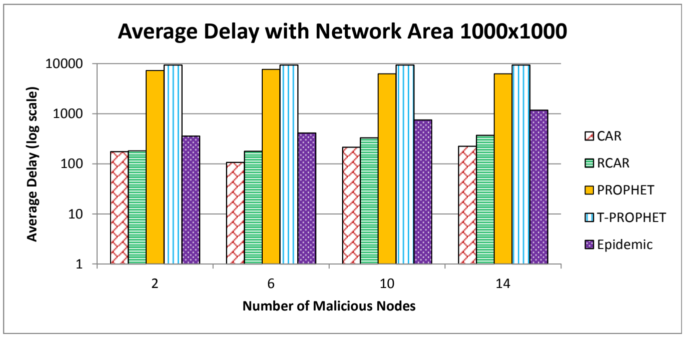


Figure 3.12: AverageDelay vs Number of Blackholes.

Furthermore, the average delay of ProPHET is higher than that of CAR in all the scenarios so that also the average delay of T-ProPHET is higher than that of RCAR. The fact that the average delay of CAR is lower than that of ProPHET has been already discussed in [32]. In practice, CAR employs also synchronous routing while ProPHET does not.

3.5.5 Total number of sent messages

Figures 3.13 and 3.14 show the total number of sent messages in the network by RCAR, CAR and ER. We do not show the values for T-ProPHET and ProPHET, because the authors in [39] do

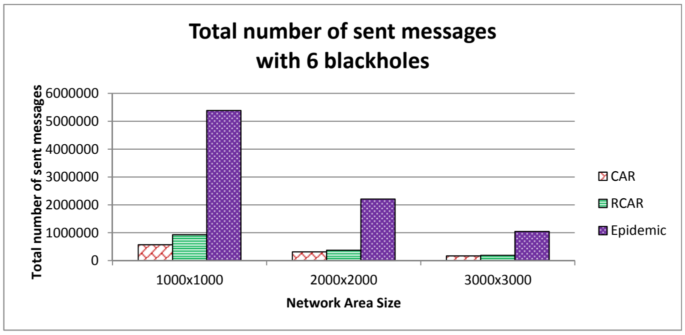


Figure 3.13: Total number of sent messages vs Network Area Size.

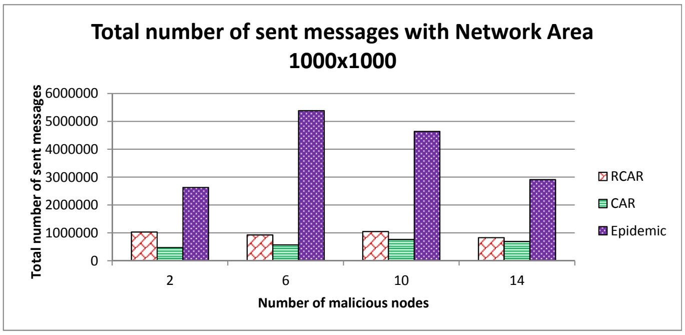


Figure 3.14: Total number of sent messages vs Number of malicious nodes.

not analyze this metric. We note that ER sends about 5.38×10^6 messages in the worst case (scenario with 6 blackholes and network area of 1000 m $\times$ 1000 m) and 1.046×10^6 messages in the best case (scenario with 6 blackholes and network area of 3000 m $\times$ 3000 m). RCAR instead sends about 1.01×10^6 messages in the worst case (scenario with 10 blackholes and network area of 1000 m $\times$ 1000 m) and 0.2×10^6 messages in the best case (scenario with 6 blackholes and network area of 3000 m $\times$ 3000 m).

We note that ER transmits a number of messages that is about five times greater than RCAR. This large overhead allows ER to outperform the other protocols in terms of delivery ratio. However, such a gain is very limited. For instance, in the case of 6 blackholes and a network size equal to 3000 $m \times 3000$ m the delivery ratio of ER is 0.88 whereas that of RCAR is 0.77 (Figure 3.13). It follows that from a practical point of view such an improvement of delivery ratio is not sufficient to justify the corresponding large communication overhead. For this reason, ER is not advisable in a DTN.

RCAR generates a greater number of messages than CAR because of the acknowledgment mechanism. The overhead in number of messages introduced by RCAR to CAR is 53.95% in the worst case (scenario with 2 blackholes and network area of 1000 m $\times$ 1000 m) and 12.36% in the best case (scenario with 6 blackholes and network area of 3000 m $\times$ 3000 m). The high overhead introduced by RCAR to CAR in the worst case is because in this scenario CAR has also a high Attraction Ratio (Figure 3.9). This means that CAR delivers a low number of messages, because many of them are dropped by blackholes.

3.6 Conclusions

In this chapter we have presented a reputation-based protocol to contrast blackholes in a DTN. The protocol has been integrated in CAR and the resulting RCAR protocol has been compared with T-ProPHET, a state-of-the-art reputation-based routing protocol deriving from ProPHET, from several viewpoints. As it turns out RCAR outperforms T-ProPHET in terms of delivery ration and average delay. Furthermore, the improvement that RCAR makes to CAR always outperforms the improvement that T-ProPHET makes to ProPHET. In addition to this, we believe that RCAR has the following merits.

- RCAR proves that an effective and efficient reputation-based routing protocol for DTN can be based on a *local* notion of reputation. As it turns out from Section 3.5, such a local notion allows us to effectively contrast blackholes without incurring in the overhead and the technical complications inherent to maintaining a global notion of reputation.

- Furthermore, RCAR has a reduced overhead. Actually, in RCAR the information for reputation management is properly integrated in data and acknowledgement messages. So, RCAR does not need to resort to expensive broadcast/multicast communication for reputation management. In addition, simulations show that the node list is short so making sustainable the related communication and computation overhead.

- Finally, RCAR shows that it is able to adapt to the highly changing conditions of a DTN. In particular RCAR is able to adapt its aging period so reducing the chance of false positives and false negatives.

Chapter 4

Publish/Subscribe Systems

Quello che facciamo è soltanto una goccia
nell'oceano. Ma se non ci fosse quella goccia
all'oceano mancherebbe.

Teresa di Calcutta (1910–1997)

4.1 Introduction

Publish-subscribe is a communication paradigm that supports dynamic, asynchronous, many-to-many communication in a distributed system [3]. In a publish-subscribe (pub-sub) system a network of brokers is responsible for routing messages from publishers to subscribers. Messages are routed based on their topics, an information descriptor contained in the messages themselves. Subscribers have to declare their interests in specific topics by issuing subscriptions to brokers.

When deployed over a wide, large scale distributed environment, a pub-sub service must distribute messages to a dynamic population of publishers through a network of brokers belonging to distinct possibly untrusted authoritative domains. Such an environment raises serious security concerns [7, 56]. Actually, an adversary who succeeds in impersonating a client or a broker can eavesdrop messages, modify them or inject fake ones. An adversary pretending to be a broker can even drop and divert messages.

Security of publish-subscribe systems has become an emerging research issue. Most of proposed works have investigated how to efficiently support secure services, enclosing access control and secure end-to-end delivery of messages, over a network of untrusted brokers [6, 57–62]. In this chapter we face with a different problem, namely how to support an overlay of brokers that unite to provide confidentiality and integrity in end-to-end message delivery. This approach starts from the key observation that the need for mechanisms to allow organizations to collaborate securely is recognized in many environments, from military to industrial and academic [63]. Thus our proposal revolves around the notion of *security group*, a coalition of brokers, each representing a given organization, interconnected by secure connections. A security group guarantees confidentiality and integrity in end-to-end message delivery. Furthermore, it allows clients belonging to any organization to connect to any broker in the group to publish and subscribe. So doing, security groups support mobility of clients. Security groups are *dynamic* in that brokers may join and leave over the lifetime of the collaboration. Brokers are *trusted* in that a broker can join a security group if and only if it fulfills a predefined *group admission policy*. Trust between brokers is generally established through some extra-technological means, such as contracts, treaties, or memoranda of agreement [63]. However, after the trust relationships between brokers have been established the remaining process by which a broker joins a security group is automatic. Finally, security groups are *closed* in that no message published in a security group can be notified outside the group. Communication between security groups needs is supported at the application level by the *gateway*, a special client that can publish and subscribe in both groups.

The chapter is organized as follows. In Section 4.2 we discuss the security requirements the publish-subscribe system must fulfill. In Section 4.3 we introduce the notions of security groups and group admission policy, and present a *group admission protocol*. Furthermore, we describe client management and inter-group communication. In Section 4.4 we briefly discuss an early prototype. In Section 4.5 we compare our system with related works, and, finally, we finally draw our conclusions in Section 4.6.

4.2 Security Requirements

In a distributed setting, a pub-sub service is implemented by a network of brokers that is called the *Dispatching Network*. In such a network, two directly connected brokers are called *peers*. Brokers provide clients the interface to the pub-sub system and cooperate in order to provide content-based routing and filtering based on topics, and reliable delivery of messages. We assume that brokers are fixed whereas clients are mobile. This means that at different times a client can be connected to different brokers.

When deployed over a large scale, wide area setting, a pub-sub service must handle information dissemination across distinct authoritative domains, heterogeneous platforms and a large, dynamic population of publishers and subscribers. Such an environment raises serious security concerns. Actually, an adversary who succeeds in passing herself as a client or a broker can eavesdrop messages, modify them or inject fake ones. An adversary pretending to be a broker can drop messages or divert them to bogus clients.

A secure publish-subscribe system must meet the *confidentiality* and *integrity* security requirements [6,7]. In general, integrity refers to the trustworthiness of information and system and it is usually phrased in terms of preventing improper or unauthorized modifications. In a publish-subscribe system, integrity can be stated in terms of preventing improper or unauthorized modifications of messages, but also of subscriptions, and service [56]. Actually, subscription integrity is crucial as subscriptions form the basis for routing and forwarding. Furthermore, the integrity of the pub-sub service is crucial because if an adversary managed to join the dispatching network, the adversary could insert bogus subscriptions and act as a bogus subscriber to neighboring brokers, ignore the routing algorithm and route messages to arbitrary destinations or

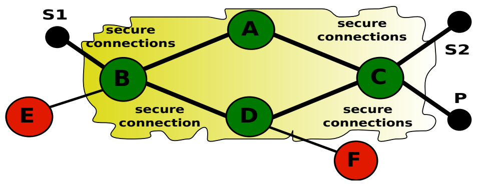


Figure 4.1: A Security Group.

drop them altogether. For this reason, in a secure publish-subscribe system, routing must be performed by trusted brokers only.

A publish-subscribe system has also to satisfy the secrecy requirement. In general, secrecy refers to the ability to keep the content of information from all but those authorized to have it. In a publish-subscribe system, secrecy can be phrased in terms of preventing any non-authorized entity to see messages and subscriptions. When messages and subscriptions contain sensitive content, publishers and subscribers may wish to keep them secret. This is especially important in a large pub-sub system where information may travel through network segments that are not necessarily trusted. For this reason, in a secure pub-sub system, messages should travel along secure channels and be handled in the clear by trusted brokers.

4.3 Secure Pub-Sub System

In order to fulfill the security requirements, we organize brokers in *security groups*. A security group is composed of a connected subset of brokers that provide a *closed* and *secure* routing of messages. Closed routing means that a message published in the group is routed within the group and never leaves it. In other words, a message is forwarded only to brokers belonging to the group. Secure routing means that a security group guarantees confidentiality and integrity of routed messages by requiring that any pair of peers in the group are interconnected with a secure connection. Figure 4.1 shows a security group composed of four brokers, namely A, B, C, and D interconnected by secure connections.

Only *trusted* brokers can join a security group. The trust of one broker with respect to another is generally established through some extra-technological means, such as contracts, treaties, or memoranda of agreement. After the trust relationships between brokers have been established the remaining process by which a broker joins a group is automatic.

A broker willing to join a security group must comply with the Group Admission Policiy (GAP) of that group. The GAP specifies the trust relationship that the broker must have with the security group in order to be admitted to the group itself. Possible policies are: a) the *One Broker Policy* (OBP), according to which at least one member of the group must trust the broker willing to join; b) the *Majority Brokers Policy* (MBP), according to which a majority of members of the group must trust the broker willing to join, and, finally, c) the *All Brokers Policy* (ABP), according to which all members of the group must trust the broker willing to join. Of course other policies can be defined as needed. A *group admission protocol* ascertains whether the broker requesting to join is indeed allowed to join according to the given *Group Admission Policy* (GAP).

Clients wanting to use the security services provided by a security group must establish a secure connection to the group. Only trusted clients can connect to the security group. A client is trusted by the group if and only if it is trusted by at least one broker in the group. Figure 4.1 shows a publisher P and two subscribers, S_1 and S_2 connected to the group. Any path leading from P to S_1, or S_2, is composed of secure connections.

4.3.1 Keys and Certificates

We assume that each principal has a pair of private and public keys. In order to guarantee the authenticity and validity of public keys, we use certificates. A certificate is a data structure composed of two parts: a data part and a signature part. The data part contains cleartext data including, as a minimum, a public key, a string identifying the principal to be associated there with, and a validity period. The signature part contains the digital signature of a Certification Authority (CA) over the data part, thereby binding the principal's identity to the specified public key. The Certification Authority is a trusted third party whose signature on the certificate vouches for the authenticity of the public key bound to the principal identity. Anyone with the public key of this authority can verify this assertion and, providing he trusts the authority, he uses the indicated key to authenticate the indicated principal. For simplicity, but without lack of generality, we assume a single certification authority CA, trusted by all principals. The public key of the certification authority must be distributed in an authenticated manner to every principal. We assume this is done on-line by means of well-known methods.

Finally, we assume that a couple of principals (brokers and clients) can use certificates to establish a *secure connection* between each other. A secure connection implies mutual authentication of principals and prevents eavesdropping, tampering and message forgery. Secure Socket Layer (SSL) and Transpor Layer Security (TLS) can be used to establish secure connections [64, 65].

4.3.2 The group admission protocol

Every broker records the brokers it trusts in the Trusted Broker List (TBL) that has one element for each trusted broker. It follows that a trust relationship between two brokers is established by reserving an element for the one in the TBL of the other and vice versa. Furthermore, every broker keeps track of the membership of the security group by means of the membership variable *members*. In addition, a broker keeps track of the brokers it is connected to by means of the Trusted Peers Table (TPT), which has one entry for each connected broker. Finally, the Group Admission Policy is implemented by a logical *policy evaluation function*, *pev*, that takes a set of brokers $\mathcal{S}$ as input and returns TRUE if the set is consistent with the GAP; FALSE otherwise.

In the most general case, a broker may be connected to one or more other brokers in the security group. For instance, in Figure 4.2, broker B is connected to three other brokers: A, E, and D. A broker joins a security group upon establishing the first connection with any broker in the group. When a broker leaves the

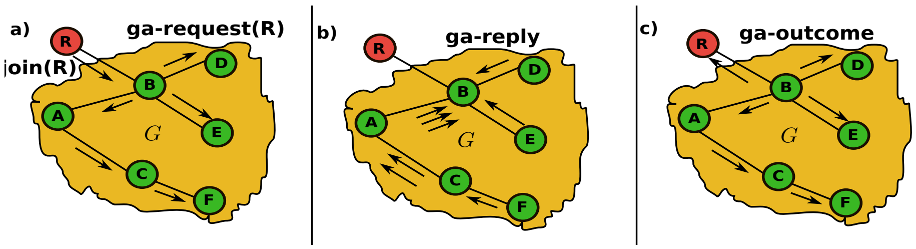


Figure 4.2: Group admission protocol.

security group all existing connections to the brokers of the group has are severed.

With reference to Figure 4.2, in order to join a security group $\mathcal{G}$, a broker R establishes a secure connection to any broker in $\mathcal{G}$, say B, and sends this broker a *join* request specifying its identifier R as a parameter (Figure 4.2.a). Upon receiving a join request, broker B checks whether broker R is in $members_B$ to ascertain whether R is already member of $\mathcal{G}$. If this not the case, broker B starts an execution instance of the *group admission protocol* (Figure 4.2.a–c), to check whether the requesting broker R fulfills the group admission policy (Figure 4.2.c) If R does, the broker is admitted to the group. Otherwise, R is not admitted and the secure connection between R and B may be either closed or degraded to an insecure one.

More in detail, the group admission protocol consists in the following actions. Initially, the broker B publishes in the group $\mathcal{G}$ a *group admission request* message, *ga-request*, that specifies the identifier R of the requesting broker (Figure 4.2.a).

Upon receiving a ga-request message, every broker B_i in $\mathcal{G}$ searches its TBL to ascertain whether it has a trust relationship with the requesting broker R specified in the message, and returns the result to B in a *group admission reply* message, *ga-reply* (Figure 4.2.b).

Upon receiving a ga-reply message from every broker in B_i in $\mathcal{G}$, the broker B determines the subset $\mathcal{T}(R) \subseteq \mathcal{G}$ of brokers in $\mathcal{G}$ that have a trust relationship with R. Then, B computes $outcome \leftarrow pev(\mathcal{T}(R))$. If the outcome is TRUE, B reserves an entry for R into its Trusted Peers Table, updates the membership

$members_B \leftarrow members_B \cup \{R\}$, and returns R a successful join reply message that contains $members_B$, the current membership of the group, and *pev*, the policy evaluation function. Otherwise, if the outcome is FALSE, B returns an usuccessful join reply message to R and severes the secure connection with R, or downgrades it to an insecure one. Finally, B publishes in $\mathcal{G}$ a *group admission outcome* message, *ga-outcome*, that carries the value of *outcome*.

Finally, upon receiving a ga-outcome message from B, every broker B_i in $\mathcal{G}$ checks whether the message contains the TRUE value. If this is the case, B_i inserts R into its membership, $members_i \leftarrow members_i \cup \{R\}$. Otherwise, the identifier R is dropped.

For performance reasons, the group admission protocol can be optimized. One possible optimization consists in the summarization of the protocol as follows. With reference to Figure 4.2 b), instead of propagating all the receiving *ga-replies*, a broker of the group could send a unique *ga-reply*, containing a *summary* of the received ones. For example, if the *ga-reply* coming from node F is affermative and that coming from node C is negative, the *ga-reply* sent by node A contains a summary: (one negative, two positives) (supposing that A's TBL contains R) or (two negatives, one positive) (if A's TBL does not contain R).

A further optimization depends on the chosen admission policy. If for example, the chosen policy is the OBP and if B is the is the broker with which the requesting broker R has a trust relationship, then the execution of the protocol is not necessary and B can send soon the ga-outcome message. It is worthwhile to notice that typically, the broker that is topologically closer with which a secure connection is established it is also the broker with which a trust relationship exists.

When a member B_i of the security group wants to leave the group $\mathcal{G}$, B_i publishes in the security group a *group leave* message. Upon receiving a group leave message, every broker B_j removes the leaving broker from the secure group membership, $members_j \leftarrow members_j \setminus \{R\}$. Furthermore, if B_j has any connection with B_i, then B_j severes it and removes the entry reserved for B_i in its Trusted Peer Table TPT_j.

As a final comment, we point out that, by default, every broker in the secure group subscribes to group admission and group leave events and messages.

4.3.3 Management of clients

Similar to the case of inter-broker trust, the trust of one broker in a client is generally established through some extra-technological means. Every broker records the clients it trusts in the *Trusted Clients Table* (TCT).

A client C can connect to a security group G if and only if the client is trusted by at least one broker in $\mathcal{G}$. In order to connect to $\mathcal{G}$, a client C establishes a secure connection with any broker B in $\mathcal{G}$ and sends it a *connect request* message. Upon receiving a connect request message, a broker starts an execution instance of the *client admission protocol*, to check whether the requesting client can connect to the security group (if the requesting client cannot connect to the security group, the secure connection between C and B may be either closed or degraded to an insecure one).

More in detail, the client admission protocol consists in the following actions. The broker B publishes a *client connection request* specifying the identifier C of the client. Upon receiving a client connect request, any broker B_i in $\mathcal{G}$ checks whether C is contained in its *Trusted Clients Table* (TCT), and returns B the results of the check in the *client connection reply*. Upon receiving a client connection reply from every broeker in the group, broker B determines whether the client C can connect to $\mathcal{G}$ by ascertaining whether at least one broker in the group trusts C.

4.3.4 Communication between Security Groups

In this section we analyze communication between security groups. In particular, we require that two or more security groups can communicate with each other, that is, messages published into a group can be notified to subscribers belonging to another group. For instance, with reference to Figure 4.3, messages published by

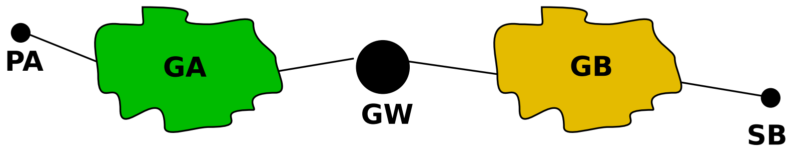


Figure 4.3: The *gateway*.

P_A, belonging to the group $\mathcal{G}_\mathcal{A}$, have to be notified to S_B, belonging to group $\mathcal{G}_\mathcal{B}$.

As security groups are closed, we solve the problem at the application level through the abstraction of *gateway*, i.e., a client that acts as publisher in one group and subscriber in the other. With reference to Figure 4.3, the gateway GW is connected to both security groups $\mathcal{G}_\mathcal{A}$ and $\mathcal{G}_\mathcal{B}$, receives publications from the former group and re-publishes them in the latter. Thus, thee gateway GW must be both trusted by both groups and compliant with any access control policies enforced by both security groups [58, 62].

Let us suppose that messages belonging to topic “T” published in group $\mathcal{G}_\mathcal{A}$ are to be published in $\mathcal{G}_\mathcal{B}$. First of all, the gateway GW must be allowed to subscribe to topic “T” in $\mathcal{G}_\mathcal{A}$. Then, as every security group has its own topic space, the topic of messages from $\mathcal{G}_\mathcal{A}$ must be renamed in order to publish them into $\mathcal{G}_\mathcal{B}$ without risk of topic clashing. Thus, these messages will be published in $\mathcal{G}_\mathcal{B}$ as belonging to the topic “GA::T”. Finally, the gateway GW must be allowed to publish messages belonging to topic “GA::T” in $\mathcal{G}_\mathcal{B}$. Of course, interested clients in $\mathcal{G}_\mathcal{B}$ must subscribe to this topic.

Consider again the example of Figure 4.3. Suppose that S_B in $\mathcal{G}_\mathcal{B}$ subscribes to messages of topic “GA:T”. Thus, when P_A publishes a message of topic “T”, this is notified to the gateway GW, that adds to the topic the name of the group so obtaining the new topic “GA::T”, and publishes it in the group $\mathcal{G}_\mathcal{B}$. The new message is later notified to S_B.

4.4 A prototype

The described architecture has been implemented in the Java programming language as an extension of REDS (REconfigurable Dispatching System) a pub-sub system conceived to tolerate dynamic reconfigurations of the dispatching infrastructure [66]. In the REDS context, every broker is structured in two layers, the *Overlay* and the *Routing* layer. In order to support the security issues described above, the following components are added: a) the *SSLTransport*, which allows a broker to establish a secure connection with another broker; b) the *GroupManager*, which allows to manage groups; c) the *ExtendedDispatchingService*, which is the extended client interface, allowing a client to act as a gateway.

4.4.1 The SSLTransport

The standard REDS architecture defines two Transport objects, at the Overlay level, the `TCPTransport` and `UDPTransport`. In order to support secure communications between two brokers, a new Transport is added, the `SSLTransport`. This new Transport takes advantage of the SSL protocol to guarantee confidentiality and integrity of messages.

4.4.2 The Group Manager

The *Group Manager* is mainly responsible to perform the broker and the client admission protocol. To this purpose the Group Manager maintains the local TBL and the local TCT. The TBL is implemented by the `TBL` component whose Java class is the following:

```
public class TBL{
 void addBroker(Certificate c);
 void removeBroker(Certificate c);
 boolean isContained(Certificate c);
}
```

The methods `addBroker()` and `removeBroker()` add and remove a new broker with certificate `c` in the local TBL respectively.

The method `isContained()` checks whether the certificate `c` is contained in the local TBL. A similar class `TCT` is defined to manage trusted clients which is not shown for brevity.

The group admission protocol is implemented by implementing the following Java interface:

```
public interface GAP{
 boolean execute(Certificate broker);
}
```

The operation `execute` receives the certificate `broker` of the requesting broker and triggers the execution of the group admission protocol. The operation returns a boolean value which specifies whether the requesting broker is allowed to join according to the group admission policy. In the prototype the `GAP` interface has been implemented by three classes, `OBP`, `MBP` and `ABP`, to support the policies and the optimizations described above.

The client admission protocol is implemented by a class `CAP` whose interface is similar to the `GAP` interface except it has not the OBP, MBP and ABP instantiations. `CAP` is not shown for brevity.

The Group Manager encapsulates a `TBL` object, a `TCT` object, a `GAP` object and a `CAP` object. The Group Manager is not a new layer but rather a new component that does not alter the two-layer structure of REDS. For instance, when there is a new request of connection, the Overlay layer passes the request to the Group Manager by invoking the method `gap()` of the `GAP` object. If the group admission policy is satisfied then the requesting broker is added as new member of the group.

4.4.3 The ExtendedDispatchingService

The ExtendedDispatchingService component is an extension of the standard client interface supported by REDS. It allows a client to connect to many groups and specify what groups it wants to subscribe to or publish in. In practice, it allows the inter-group communication. Its Java interface is the following:

```
public interface ExtendedDispatchingService
```

```
  extends DispatchingService{
 public void open(Group g);
 public void close(Group g);
 public void subscribe(Filter f, Group g);
 public void unsubscribe(Filter f, Group g);
 public void publish(Message msg, Group g);
 public Message getNextMessage(Group g);
}
```

The method `open()` opens a connection to group g, while the method `close()` closes the connection to group g. The methods `subscribe()` and `unsubscribe()` respectively subscribe and unsubscribe to filter f in the group g. The method `publish()` allows to publish a message in the group g. Finally, the method `getNextMessage()` notifies a message coming from group g.

4.5 Related Works

Many recent research has been made in the field of *security* for publish/subscribe networks.

Chmielewski *et al.* in [67] propose a solution at the application level, without affecting the infrastructure level. In their solution, the programmer does not care about security details. However he/she must define what kind of security guarantees he/she wants. In our solution, instead, the programmer does not care about security at all, because it is a service offered at the infrastructure level.

Brian Shand and Jem Rashbass in [57] guarantee security services through the concept of access control. Those services are applied by the first broker of the chain and then are verified by the last broker of the chain, without affecting the rest of the network. In our case, instead, security services are applied step-by-step by each broker of the group.

Hui Zhang *et al.* in [59] use the *information foiling* technique. However, due to the high number of messages passing throughout the network, this mechanism does not scale. Furthermore it wastes a lot of network resources to deliver messages.

Another solution has been proposed by Himanshu Khurana in [60]. In his article, he provides the following strategy: encrypt only a part of the messasge, that is the interesting one. In this way, confidentiality and integrity of messages is guaranteed, but he does not say anything about *service integrity*. Infact he assumes that messages are correctly forwarded throughout the whole Dispatching Network.

Paul Rabinovich and Robert Simon propose a secure message delivery, based on *messages replication and voting* [61]. This solution guarantees service integrity, but it does not specify any mechanism to provide messages confidentiality, integrity and authentication.

In their work [62], Yuanyuan Zhao and Daniel C. Sturman discuss about dynamic access control. Their solution is based onto a manual configuration of rules by the network administrator. Rules are then propagated throughout the whole Dispatching Network. Their solution, however, investigates only about access control, without specifying nothing about the other security services.

4.6 Conclusions

With reference to a set of organizations wishing to collaborate we have defined a secure pub-sub service that supports inter-organization communication. The service revolves around the notion of security group, a group of brokers, each representing a given organization, interconnected by secure connections. A security group is formed only by trusted servers, guarantees confidentiality and integrity in end-to-end delivery of messages and support clients mobility. An early prototype of the service has been implemented. Future effort will address access control in publications and subscriptions.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments. Furthermore, we would like to thank the REDS group, and in particular Prof. Gian Pietro Picco, University of Trento, Italy, who have allowed us to study and extend REDS.

Bibliography

[1] A. Caiti, P. Felisberto, T. Husoy, S.M. Jesus, I. Karasalo, R. Massimelli, T.A. Reinen, and A. Silva. Underwater acoustic network. In *OCEANS*, pages 1 –7, Santander, Spain, June 2–6 2011.

[2] Salman Qadir Ali and Adeel Junaid Baig. Routing protocols in delay tolerant networks - a survey. In *International Conference on Emerging Technologies (ICET 2010)*, pages 70–75, Islamabad, Pakistan, October 18–19 2010.

[3] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces Of Publish/Subscribe. *ACM Computing Surveys*, 35(3):114–131, June 2003.

[4] Ling-Jyh Chen, Che-Liang Chiou, and Yi-Chao Chen. An evaluation of routing reliability in non-collaborative opportunistic networks. pages 50 –57, May 2009.

[5] Yanping Cong, Guang Yang, Zhiqiang Wei, and Wei Zhou. Security in Underwater Sensor Network. In *Proceedings of the 2010 International Conference on Communications and Mobile Computing (CMC 2010)*, pages 162–168, Shenzhen, China, April 12–14 2010.

[6] Lukasz Opyrchal and Atul Prakash. Secure Distribution of Events in Content-Based Publish Subscribe Systems. In *Proceedings of the 10th USENIX Security Symposium*, pages 281–295, Washington, D.C., USA, August 13–17 2001.

[7] L. Fiege, A. Zeidler, A. Buchmann, and Tu Darmstadt. Security aspects in publish/subscribe systems. In *In Proceedings of the Third International Workshop on Distributed Event-based Systems (DEBS04*, Edinburgh, Scotland, United Kingdom, 23–28 May 2004.

[8] Trusted computing group. tcg specification architecture overview, revision 1.4. august 2007.

[9] Trusted computing group. tpm main specification, version 1.2. revision 103. july 2007.

[10] Trusted computing group. tcg software stack(tss) specification version 1.2. march 2007.

[11] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pages 255–265, Boston, Massachusetts, United States, August 6–11 2000.

[12] A. Mishra, K. Nadkarni, and A. Patcha. Intrusion detection in wireless ad hoc networks. *Wireless Communications*, 11(1):48–60, February 2004.

[13] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation: a virtual machine directed approach to trusted computing. In *Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium - Volume 3*, San Jose, CA, May 6–7 2004.

[14] M.C. Domingo. Securing underwater wireless communication networks. *Wireless Communications*, 18(1):22–28, February 2011.

[15] Ian F. Akyildiz, Pompili Dario, and Tommaso Melodia. Underwater acoustic sensor networks: research challenges. *Ad Hoc Networks*, 3(3):257–279, 2005.

[16] Yangze Dong and Pingxiang Liu. Security considerations of underwater acoustic networks. In *International Congress on Acoustics (ICA 2010)*, Sydney, Australia, August, 23–27 2010.

[17] Yangze Dong and Pingxiang Liu. A comparison of two secure routing protocols in underwater acoustic network. In *International Congress on Acoustics (ICA 2010)*, Sydney, Australia, August, 23–27 2010.

[18] Yangze Dong and Pingxiang Liu. Simulation study on self-reorganization of underwater acoustic networks. In *International Conference on Information and Automation*, pages 595–600, Shenzhen, China, June 2011.

[19] A. Caiti and A. Munafo'. Adaptive Cooperative Algorithms for AUV Networks. In *IEEE International Conference on Communications Workshops (ICC 2010)*, pages 1–5, Capetown, May 23–27 2010.

[20] H. Rustad. A Lightweight Protocol Suite for Underwater Communication. In *2009 International Conference on Advanced Information Networking and Applications Workshops (WAINA 2009)*, pages 1172–1178, Bradford, May 26–29 2009.

[21] Milica Stojanovic. On the relationship between capacity and distance in an underwater acoustic communication channel. In *Proceedings of the 1st ACM international workshop on Underwater networks (WUNet 2006)*, pages 41–47, Los Angeles, CA, USA, September, 25 2006.

[22] Anthony D. Wood and John A. Stankovic. Denial of service in sensor networks. *Computer*, 35:54–62, October 2002.

[23] Bruce Schneier. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*, pages 191–195. John Wiley & Sons, Inc., 1995.

[24] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, October 1996.

[25] C Karlof, N Sastry, and D Wagner. TinySec: a link layer security architecture for wireless sensor networks. In *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 162–175, Baltimore, MD, USA, November 3–5 2004.

[26] Gianluca Dini and Ida Maria Savino. S2RP: A secure and scalable rekeying protocol for wireless sensor networks. In *Proceedings of the 3rd IEEE International Conference on Mobile Ad-Hoc and Sensor systems (MASS06)*, pages 457–466, Vancouver, BC, October, 9–12 2006.

[27] GraalTech http://www.graaltech.it. The folaga vehicle, 2011.

[28] NIST. *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography.* Special Publication 800–56A, March 2006.

[29] Carlo Blundo, Alfredo De Santis, Amir Herzberg, Shay Kutten, Ugo Vaccaro, and Moti Yung. Perfectly-Secure Key Distribution for Dynamic Conferences. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 1993)*, pages 471–486, California, USA, August 22–26 1993.

[30] The simpy simulator http://simpy.sourceforge.net, 2011.

[31] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical report, Department of Computer Science, Duke University, 2000.

[32] M. Musolesi and C. Mascolo. Car: Context-aware adaptive routing for delay-tolerant mobile networks. *IEEE Transactions on Mobile Computing*, 8(2):246–260, February 2009.

[33] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mobile Computing Communication Review*, 7:19–20, July 2003.

[34] A. Seth and S. Keshav. Practical security for disconnected nodes. In *ICNP Workshop on Secure Network Protocols, 2005. (NPSec)*, Boston, Massachusetts, USA, November, 6 2005.

[35] N. Bhutta, G. Ansa, E. Johnson, N. Ahmad, M. Alsiyabi, and H. Cruickshank. Security analysis for delay/disruption tolerant satellite and sensor networks. In *International Workshop on Satellite and Space Communications (IWSSC 2009)*, Siena, Italy, September, 10–11 2009.

[36] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. In *IEEE Interna-tion Workshop on Sensor Network Protocols and Applications*, pages 113 – 127, Anchorage, Alaska, USA, May 11 2003.

[37] Zhong Xu, Yuan Jin, Weihuan Shu, Xue Liu, and Junhai Luo. Sred: A secure reputation-based dynamic window scheme for disruption-tolerant networks. In *Military Communications Conference. (MILCOM 2009)*, pages 1 –7, Boston, October 18–21 2009.

[38] Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering*, 1960.

[39] Na Li and Sajal K. Das. A trust-based framework for data forwarding in opportunistic networks. *Ad Hoc Networks*, In Press, Corrected Proof:–, 2011.

[40] M. Chuah and P. Yang. Impact of selective dropping attacks on network coding performance in dtns and a potential mitigation scheme. In *Computer Communications and Networks (ICCCN 2009)*, pages 1 –6, San Francisco, CA, August 3–6 2009.

[41] Gianluca Dini and Angelica Lo Duca. A reputation-based approach to tolerate misbehaving carriers in delay tolerant networks. In *IEEE International Symposium on Computers and Communications (ISCC'10)*, Riccione (IT), June 22–25 2010.

[42] L. Buttyn, L. Dra, M. Flegyhzi, and I. Vajda. Barter trade improves message delivery in opportunistic networks. *Elsevier Ad Hoc Networks, vol. 8, no. 1*, pages 1–14, January 2010.

[43] I. Koukoutsidis, E. Jaho, and I. Stavrakakis. Cooperative content retrieval in nomadic sensor networks. In *Infocom MObile Networking for Vehicular Environments Workshop (MOVE 2008)*, Phoenix, AZ, April 13–18 2008.

[44] Rongxing Lu, Xiaodong Lin, Haojin Zhu, Xuemin Shen, and B. Preiss. Pi: A practical incentive protocol for delay tolerant networks. *Wireless Communications, IEEE Transactions on*, 9(4):1483 –1493, April 2010.

[45] Feng Li, Jie Wu, and Srinivasan Avinash. Thwarting blackhole attacks in disruption-tolerant networks using encounter tickets. In *International Conference on Computer Communications (INFOCOM 2009)*, pages 2428 –2436, Rio de Janeiro, April 19–25 2009.

[46] Yanzhi Ren, Mooi Choo Chuah, Jie Yang, and Yingying Chen. Detecting blackhole attacks in disruption-tolerant networks through packet exchange recording. In *World of Wireless Mobile and Multimedia Networks (WoWMoM 2010)*, pages 1–6, Montreal, QC, Canada, June 14–17 2010.

[47] Yanzhi Ren, Mooi Choo Chuah, Jie Yang, and Yingying Chen. Muton: Detecting malicious nodes in disruption-tolerant networks. In *Wireless Communications and Networking Conference (WCNC 2010)*, pages 1 –6, Sidney, NSW, April 18–21 2010.

[48] John Burgess, George Dean Bissias, Mark D. Corner, and Brian Neil Levine. Surviving attacks on disruption-tolerant networks without authentication. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc '07)*, pages 61–70, Montreal, Quebec, Canada, September 9–14 2007.

[49] A. Krifa, C. Baraka, and T. Spyropoulos. Optimal buffer management policies for delay tolerant networks. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2008)*, pages 260 –268, San Francisco, CA, June 16–20 2008.

[50] John Burgess, Brian Gallagher, David Jensen, and Brian Neil Levine. Maxprop: Routing for vehicle-based disruption-tolerant networks. In *Conference on Computer Communications (INFOCOM 2006)*, pages 1–11, Barcelona, Spain, April 23–29 2006.

[51] Aruna Balasubramanian, Brian Neil Levine, and Arun Venkataramani. Dtn routing as a resource allocation problem. In *Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2007)*, pages 373–384, Kyoto, Japan, August 27–31 2007.

[52] Rebecca N. Wright, Patrick D. Lincoln, and Jonathan K. Millen. Dependent graphs: a method of fault-tolerant certificate distribution. *Journal of Computer Security*, 9(4):323–338, 2001.

[53] Rebecca N. Wright, Patrick D. Lincoln, and Jonathan K. Millen. Efficient fault-tolerant certificate revocation. In *Proceedings of the 7th ACM conference on Computer and Communications Security (CCS '00)*, pages 19–24, Athens, Greece, November 1–4 2000.

[54] Srdjan Capkun, Levente Butty, and Jean-Pierre Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 2:52–64, January-March 2003.

[55] Joonsang Baek, Jan Newmarch, Reihaneh Safavi-naini, and Willy Susilo. A survey of identity-based cryptography. In *Australian Unix Users Group Annual Conference (AUUG 2004)*, pages 95–102, Melbourne, Australia, September, 1–3 2004.

[56] Chenxi Wang, Antonio Carzaniga, David Evans, and Alexander L. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS2)*, pages 3940–3947, 2002.

[57] Brian Shand and Jem Rashbass. Security for middleware extensions: event meta-data for enforcing security policy. In *Proceedings of the 2008 Workshop on Middleware Security (MidSec'08)*, pages 31–33, Leuven, Belgium, December, 2 2008.

[58] Andr Belokosztolszki, David M. Eyers, Peter R. Pietzuch, Jean Bacon, and Ken Moody. Role-based access control for publish/subscribe middleware architectures. In *Procceedings of the International Workshop on Distributed Event-Based Systems (DEBS03), ACM SIGMOD*, 2003.

[59] Hui Zhang, Abhishek Sharma, Haifeng Chen, Guofei Jiang, Xiaogiao, and K. Yoshihira. Enabling information confidentiality in publish/subscribe overlay services. In *Proceedings of the International Conference on Communications (ICC'08)*, pages 5624–5628, Beijing, China, May 19–23 2008.

[60] Himanshu Khurana. Scalable security and accounting services for content-based publish/subscribe systems. In *Symposium on Applied computing (SAC 2005)*, pages 801–807, Santa Fe, New Mexico, 2005.

[61] Paul Rabinovich and Robert Simon. Secure message delivery in publish-subscribe networks using overlay multicast. *International Journal of Security and Networks*, 2:60–70, March 2007.

[62] Yuanyuan Zhao and Daniel C. Sturman. Dynamic access control in a content-based publish/subscribe system with delivery guarantees. *International Conference on Distributed Computing Systems*, 2006.

[63] D. Shands, J. Jacobs, R. Yee, and E.J. Sebes. Secure virtual enclaves: Supporting coalition use of distributed application

technologies. *ACM Transactions on Information and System Security*, 4(2):103–133, May 2001.

[64] T. Dierks and C. Allen. The TLS protocol version 1.0. The Internet Engineering Task Force (IETF), RFC 2246, January 1999.

[65] A. Freier, P. Kariton, and P. Kocher. The SSL protocol: Version 3.0. Netscape Communication, Inc., Mountain View, CA, March 1996. `http://home.netscape.com/eng/ssl3/ssl-toc.html`.

[66] Gianpaolo Cugola and Gian Pietro Picco. Reds: a reconfigurable dispatching system. In *SEM '06: Proceedings of the 6th international workshop on Software engineering and middleware*, pages 9–16, New York, NY, USA, 2006. ACM Press.

[67] Lukasz Chmielewski, Richard Brinkman, Jaap-Henk Hoepman, and Bert Bos. Using JASON to secure SOA. In *Proceedings of the 2008 Workshop on Middleware Security (MidSec'08)*, pages 13–18, Leuven, Belgium, December, 2 2008.