Service Assurance in Insecure Networks with Byzantine Adversaries

Rabinovich, Paul
SERVICE ASSURANCE IN INSECURE NETWORKS WITH BYZANTINE ADVERSARIES

by

Paul Rabinovich
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
the Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

Dr. Robert Simon, Dissertation Director
Dr. Hakan Aydin, Committee Member
Dr. Peter Paris, Committee Member
Dr. Sanjeev Setia, Committee Member
Dr. Hassan Gomma, Department Chair
Dr. Lloyd J. Griffiths, Dean,
The Volgenau School of Information
Technology and Engineering

Date: April 1, 2008
Spring Semester 2008
George Mason University
Fairfax, VA
Service Assurance in Insecure Networks with Byzantine Adversaries

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Paul Rabinovich
Master of Science
George Mason University, 2000
Bachelor of Science
Moscow Transportation Academy, 1987

Director: Dr. Robert Simon, Professor
Department of Computer Science

Spring Semester 2008
George Mason University
Fairfax, VA
Dedication

I dedicate this dissertation to my Father whose example inspired me to start this project and whose kind love and support gave me the tools necessary to complete it.
Acknowledgments

First and foremost, I’d like to thank my Dissertation Director Dr. Robert Simon for guidance and friendly advice he has provided to me for over eight years, first when I was working on my Master’s Thesis and now, when I am working towards my PhD degree. Without his help in maintaining the pace of this project, I believe, it would have taken significantly longer than it actually did.

I also want to thank the members of my Dissertation Committee for their comments on the Dissertation Proposal; these comments helped me to make the final product better.

My family has constantly cheered me on throughout this endeavor, and I am grateful to them for their support.
# Table of Contents

<table>
<thead>
<tr>
<th>List of Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>viii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>Abstract</td>
<td>xii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem Statement</td>
<td>4</td>
</tr>
<tr>
<td>1.1.1 Secure Delivery in Publish/Subscribe Networks</td>
<td>5</td>
</tr>
<tr>
<td>1.1.2 Secure Aggregation in Sensor Networks</td>
<td>5</td>
</tr>
<tr>
<td>1.1.3 Fraud Resistance in Massively Multiplayer Online Games</td>
<td>6</td>
</tr>
<tr>
<td>1.2 Contributions</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Organization</td>
<td>8</td>
</tr>
<tr>
<td>2 Related Work</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Publish/Subscribe Networks</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1 Subject-Based vs. Content-Based Pub-Sub</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2 Content Types and Query Languages</td>
<td>12</td>
</tr>
<tr>
<td>2.2.3 Composite Objects</td>
<td>13</td>
</tr>
<tr>
<td>2.2.4 System Architecture</td>
<td>13</td>
</tr>
<tr>
<td>2.2.5 Dynamic vs. Static Membership</td>
<td>14</td>
</tr>
<tr>
<td>2.2.6 Matching Algorithm</td>
<td>14</td>
</tr>
<tr>
<td>2.2.7 Delivery Mechanism</td>
<td>14</td>
</tr>
<tr>
<td>2.2.8 Quality of Service</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Security in Publish/Subscribe Networks</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 Service Integrity</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2 Authorized Access</td>
<td>19</td>
</tr>
<tr>
<td>2.3.3 Authenticity</td>
<td>21</td>
</tr>
<tr>
<td>2.3.4 Message and Subscription Integrity</td>
<td>22</td>
</tr>
<tr>
<td>2.3.5 Availability</td>
<td>22</td>
</tr>
<tr>
<td>2.3.6 Confidentiality</td>
<td>22</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Computing an aggregation result over a set as a function of aggregation</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>results over its constituent subsets</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Simulation configuration parameters</td>
<td>118</td>
</tr>
<tr>
<td>4.3</td>
<td>Simulation configuration profiles</td>
<td>119</td>
</tr>
<tr>
<td>5.1</td>
<td>FRAPPE’s assumptions about supported massively multiplayer online games</td>
<td>130</td>
</tr>
<tr>
<td>5.2</td>
<td>Boundary values of $\alpha$ for several confidence levels and corresponding peer groups</td>
<td>136</td>
</tr>
<tr>
<td>5.3</td>
<td>Summary of FRAPPE tokens</td>
<td>141</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Representative sensor node, Moteiv Corp.’s Tmote Sky</td>
<td>27</td>
</tr>
<tr>
<td>2.2</td>
<td>Typical sensor network</td>
<td>28</td>
</tr>
<tr>
<td>2.3</td>
<td>Layered sensor network architecture</td>
<td>31</td>
</tr>
<tr>
<td>2.4</td>
<td>Clustered sensor network architecture</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Typical query in TinyDB</td>
<td>37</td>
</tr>
<tr>
<td>2.6</td>
<td>Lifetime-based query in TinyDB</td>
<td>38</td>
</tr>
<tr>
<td>2.7</td>
<td>Basic MMOG architectures</td>
<td>59</td>
</tr>
<tr>
<td>3.1</td>
<td>Intermediate nodes and machines in a pub-sub network</td>
<td>68</td>
</tr>
<tr>
<td>3.2</td>
<td>Intermediate node composed of three services: receiving, filtering, and forwarding</td>
<td>69</td>
</tr>
<tr>
<td>3.3</td>
<td>Sample subscriptions (predicates) and sets of objects satisfying them</td>
<td>70</td>
</tr>
<tr>
<td>3.4</td>
<td>Progression from a predicate graph to a delivery tree</td>
<td>72</td>
</tr>
<tr>
<td>3.5</td>
<td>Mapping subscriptions to clusters of non-trusted nodes</td>
<td>74</td>
</tr>
<tr>
<td>3.6</td>
<td>Reducing the original problem to a graph-theoretic one</td>
<td>75</td>
</tr>
<tr>
<td>3.7</td>
<td>Algorithm MST-PT: Solve the MSTP on a predicate tree</td>
<td>77</td>
</tr>
<tr>
<td>3.8</td>
<td>Algorithm MST-PG: Solve the MSTP on a predicate graph</td>
<td>79</td>
</tr>
<tr>
<td>3.9</td>
<td>Algorithm DC: The Divide-and-Conquer (DC) heuristic</td>
<td>80</td>
</tr>
<tr>
<td>3.10</td>
<td>Cost vs. number of domains</td>
<td>89</td>
</tr>
<tr>
<td>3.11</td>
<td>Paired-( t )-confidence intervals for the cost vs. number of domains graph</td>
<td>90</td>
</tr>
<tr>
<td>3.12</td>
<td>Cost as a function of the filter ratio decrease factor</td>
<td>91</td>
</tr>
<tr>
<td>3.13</td>
<td>Paired-( t )-confidence intervals for the cost vs. filter ratio decrease graph</td>
<td>92</td>
</tr>
<tr>
<td>3.14</td>
<td>Cost vs. density of the underlying network</td>
<td>93</td>
</tr>
<tr>
<td>3.15</td>
<td>Paired-( t )-confidence intervals for the cost vs. density of the underlying network graph</td>
<td>94</td>
</tr>
<tr>
<td>4.1</td>
<td>Our model of a sensor network</td>
<td>97</td>
</tr>
<tr>
<td>4.2</td>
<td>A temperature-sensing sensor network with a linear constraint</td>
<td>99</td>
</tr>
<tr>
<td>4.3</td>
<td>Construction of the delivery graph</td>
<td>101</td>
</tr>
<tr>
<td>4.4</td>
<td>A DCP message</td>
<td>103</td>
</tr>
</tbody>
</table>
5.10 Relationship between the required number of peers and the fraction of malicious peers accepted by the soliciting entity ........................................ 162
5.11 Relationship between the fraction of dishonest potential neighbors for a seed and the fraction of malicious peers accepted by the soliciting entity .... 163
5.12 Relationship between the neighbor list update frequency and the fraction of honest peers rejected by the soliciting entity ................................. 164
5.13 Relationship between the neighbor list update frequency and the fraction of honest peers rejected by the soliciting entity with heartbeats exchanged every 3 minutes ................................................................. 166
Abstract

SERVICE ASSURANCE IN INSECURE NETWORKS WITH BYZANTINE ADVERSARIES
Paul Rabinovich, PhD
George Mason University, 2008
Dissertation Director: Dr. Robert Simon

This dissertation describes research into security threats in new communication environments and methods to counter them. More specifically, we consider networks where some nodes perform an important application function such as routing, filtering, aggregation, etc., and may become compromised while doing so.

We consider three types of networks: Internet-scale publish/subscribe networks, aggregating sensor networks, and peer-to-peer massively multiplayer online games (MMOG) and virtual environments. These environments are complementary to each other in many respects. A publish/subscribe network is responsible for disseminating data objects produced by one set of nodes (publishers) to another set of nodes (subscribers). Subscribers want to receive those and only those objects that satisfy their interests. A large-scale publish/subscribe network using many network providers, each under its own administrative control, presents numerous opportunities for mischief. In many cases the network providers will not trust one another and will not be fully trusted by the end users. A malicious network provider may insert, delete, modify, reorder, misdirect, or delay messages, and remain undetected. The first topic of our research is how to assure service integrity in such networks if some of the intermediate nodes may attack the system in an arbitrary fashion.
We solve the problem by creating filtering agents corresponding to user subscriptions, and mapping these agents to either hosts from trusted providers or to clusters of hosts taken from multiple non-trusted providers. As a whole, each cluster can be trusted since only a relatively small number of providers are assumed to be malicious.

A sensor network is a network connecting hundreds or thousands of sensors, tiny battery-powered computers equipped with units measuring some physical phenomena (e.g., light intensity, temperature, humidity, ambient chemical composition, etc.) and wireless radio transceivers. We study aggregating sensor networks that do not fully propagate raw measurements to their users but rather perform in-network aggregation with the intent of lowering the total amount of transmitted data thus preserving bandwidth and energy. As sensor networks are frequently placed in hostile environments (for instance, in military applications) it is important to devise mechanisms guaranteeing integrity of their service under attack, and their survivability. In this dissertation we discuss CoDeX, a collect-detect-exclude framework for secure aggregation in sensor networks. Our approach to solving the problem is based on the fact that many physical phenomena exhibit strong spatial correlation. Sensor nodes can take advantage of the broadcast nature of radio transmissions, receive measurements from their neighbors, and compare them with their own results. If the values significantly differ, the fact can be reported to the user (the "collect" phase). If a node is a subject of many such reports, it is, probably, compromised (the "detect" phase) and should be removed from the network (the "exclude" phase). We complement this approach with the use of randomized delivery (aggregation) trees, cryptography, and repeated aggregation of the same data in different configurations.

Peer-to-peer massively multiplayer online games and virtual environments pose challenges similar to those in the other two types of networks: all three lack centralized control and run autonomously. Like pub-sub networks, P2P-based MMOGs may span the Internet and contain thousands and, potentially, hundreds of thousands of nodes. Like sensor networks, these systems almost completely rely on end nodes for providing infrastructure services.
Unfortunately, most games do not provide sufficient safeguards against cheating and fraud perpetrated by the players. We developed FRAPPE, an architecture that significantly reduces this vulnerability by forming trusted "supernodes" out of non-trusted peer machines, employing authentication, confidentiality and pseudonymity services, using secret sharing and other secure multiparty computation techniques, and constructing anonymizing tunnels to hide identities of communicating parties. We also introduce a useful primitive, called local broadcast with verification (LBV), used to solicit services of peers in a particular neighborhood and subsequently verify that the peers properly executed the protocol.

Using a combination of analysis and experimental results we demonstrate that all three approaches provide strong service assurance guarantees for their respective types of networks.
Chapter 1: Introduction

Rudimentary computer networks appeared in the 1950s; they connected remote terminals, printers and remote job entry points to shared central computers [1]. As peripheral devices became widespread and varied, connecting each individual device to the central machine ceased to be economical: specialized networking hardware, such as concentrators (to batch traffic from and to a set of related units) and front-end processors (to offload the task of managing the network), was introduced.

Creation of the Arpanet, the first packet-switched network (1969), invention and implementation of TCP/IP (1977), adoption of TCP/IP as the Arpanet’s protocol stack (1982), and introduction of links to other parts of the world and other North American networks led to the creation of the Internet, a world-wide network of millions of computers that supports e-mail, instant messaging, file and news transfer, the Web, voice communications, and other applications [2]. Concurrent advances in design and manufacturing of microprocessors resulted in many new - ever more compact - computer-like devices (personal computers, laptops, tablet PCs, personal digital assistants, cellular phones, smart phones, sensors, etc.) capable of simultaneously processing data and communicating.

Communication networks can be categorized into many taxonomies. One categorization may be based on the communication medium, whether it is wired or wireless. Networks may also be characterized by scale: there are wide-area networks, metropolitan-area networks, short range networks, and local area networks. Networks may provide varied quality of service, for example, in terms of reliability. Most modern networks can provide both reliable and unreliable service and only differ in how such services are implemented and made
available to their users. Networks and network-based distributed systems may support several delivery models: broadcast, multicast and unicast. Communications based on these models may be one-to-many (as in broadcast and multicast), many-to-one (using point-to-point links), or many-to-many. They can also employ a static infrastructure, e.g., gateways, routers, brokers, location services, etc., distinct from the end-user devices, or be completely unstructured, as in peer-to-peer systems. Finally, challenges in networks (e.g., security, reliability, etc.) and mechanisms to mitigate those challenges are frequently concentrated in a particular layer of the OSI Reference Model. For example, reliable delivery in IP networks is addressed in the transport layer (via the TCP protocol), security may be addressed in the network layer (e.g., IPSec), in the transport layer (e.g., SSL/TLS) or in the application layer (e.g., HTTP Basic authentication).

Today’s Internet shows a remarkable variety of communication patterns, protocols and applications. One important communication model is publish/subscribe. A publish/subscribe (pub-sub or PS for short) network brings together producers of data objects, called publishers, and consumers of these objects, called subscribers. When joining, subscribers tell the network what kinds of objects they are interested in, and the network is expected to deliver to them only objects satisfying their interests [3][4]. The pub-sub model is used in distributed simulation, multiplayer online games, electronic publishing, information dissemination, industrial automation, software distribution, and other areas. An Internet-scale pub-sub network will inevitably span multiple network providers which may not be trusted by one another and/or by the end users. A malicious network provider may insert, delete, modify, reorder, misdirect, or delay messages, and remain undetected. The first topic of our research is how to assure service integrity in such networks if some of the intermediate nodes attack the system in an arbitrary fashion.

Countervailing the patchwork of communication protocols, media and models is the undercurrent of convergence. As Stallings writes in [5], it is clear that:
• There is no fundamental difference between data processing and data communications.

• There are no fundamental differences between data, voice and video communications.

• The distinction between single- and multi-processor computers, local area networks, metropolitan networks, and long-haul networks has blurred.

Recent strong interest in sensor networks (SNs) is a manifestation of this trend. Sensors are usually small, battery-powered devices, equipped with one or more measurement units for sensing the environment around them. In most cases sensors use inexpensive radio-based wireless links to communicate with one another and with the base station. The base station is a more powerful computer, for example, a laptop, that manages the sensor network, disseminates and collects information, and provides services to the network (e.g., time and location information). Possible applications of sensor networks include highway traffic analysis, monitoring of industrial processes, surveillance, structural health analysis, environment monitoring, etc. As sensor networks are frequently placed in hostile environments (e.g., in military applications) it is important to design systemic approaches to their protection. Our second topic of research is service assurance in aggregating sensor networks. Protection of aggregating SNs is especially challenging as much of raw information is never delivered to the base station to preserve the bandwidth and limit energy consumption by the sensors.

Wireless sensor networks are a subclass of peer-to-peer (P2P) networks. In such networks no central authority controls the participants or provides infrastructure services. Rather, the participants themselves become part of the infrastructure. P2P networks are blossoming on the Internet in the form of file swapping services (such as Napster, Kazaa, Gnutella and BitTorrent), document publishing and storage networks (e.g., Freenet [6]) and collaborative virtual environments (e.g., Croquet [7]). Popular client/server massively multiplayer online games (such as World of Warcraft, Ultima Online, Lineage, EverQuest II
and others [8]) are quickly reaching their scalability limits, and many researchers are looking into providing such games with a robust peer-to-peer foundation. As an example, the most popular MMOG, World of Warcraft from Blizzard Entertainment, grew from about 500,000 active subscribers in January 2005 to over 6 million in July 2006 [8]. Client/server-based architectures cannot support this kind of phenomenal growth indefinitely. Using a peer-to-peer architecture provides obvious scalability benefits but exacerbates challenges related to cheating and fraud perpetrated by the participants: not only the domain-specific elements of the network are vulnerable, as in more traditional architectures, but also its basic infrastructure - since it is provided by the participants themselves. In the third part of this dissertation we present FRAPPE, a fraud-resistant architecture for peer-to-peer environments that attempts to prevent cheating by combining non-trusted peers into trusted clusters, engaging peers in secure multiparty computations, anonymizing some types of communications between peers, and guarding against conflicts of interest.

1.1 Problem Statement

The topic of our research is service assurance in insecure networks with Byzantine adversaries. By an insecure network we understand a multi-actor multi-node communication system where some actors (Byzantine adversaries) may intentionally cause the system to arbitrarily deviate from its behavior specification. Service assurance is a set of measures designed to allow an insecure network to continue to provide service to its users even when under attack. We consider three types of systems that take advantage of the next generation of networking technologies, and demonstrate that (a) security issues they face are similar in many cases and (b) some of the solutions to their security problems are similar as well. All three systems under consideration are based on the peer-to-peer architecture with non-trusted peers potentially exhibiting Byzantine malicious behavior.
1.1.1 Secure Delivery in Publish/Subscribe Networks

Many emerging applications require publication of events to a large number of subscribers scattered across the Internet. Network layer multicast is commonly considered to be the optimal approach for supporting large-scale publish/subscribe systems. However, the availability of multicasting in the present day Internet is limited at best. Because of this there has been increased interest in using end-host multicast, where end systems are organized into a logical overlay whose topology is built on top of the Internet. All participating hosts implement multicast-related functionality, including membership management and packet replication. It is likely that messages and subscriptions will travel through multiple network providers, each under its own administrative control, jurisdiction, policies and regulations. We investigate how service assurance can be provided if infrastructure components cannot be fully trusted.

Our work assumes the use of topology-aware overlay multicast, whereby shortest path information calculated by IP routers is available to all nodes.

1.1.2 Secure Aggregation in Sensor Networks

Sensor networks (SN) emerged in recent years as a viable platform for monitoring applications [9][10][11][12][13]. They are used for military applications, real-time highway traffic analysis, machinery and process monitoring, surveillance, structural health prognostication, disaster recovery, air and water quality control, etc. Some of these application domains are naturally adversarial: there is a party with an incentive to disrupt the normal functioning of a deployed sensor network. Disruptions may vary from denial of service via jamming of radio links or manipulation of the network’s routing infrastructure, to selective forwarding and injection of false messages aimed at misleading recipients of data produced by the network.

We focus on application-level attacks. The difficulty of dealing with them is exacerbated
by the fact that many SNs perform in-network processing of data - such as filtering and aggregation - in an attempt to conserve energy and bandwidth. Thus, raw data may not be available outside the sensor network, and mechanisms must be devised to validate the network’s results without them.

Our goal is to provide a mechanism to detect and mitigate stealthy attacks where an attacker attempts to subvert the network without being noticed by the server and/or the base stations.

1.1.3 Fraud Resistance in Massively Multiplayer Online Games

Massively multiplayer online games (MMOG) and virtual worlds are fast becoming a prominent feature on the Internet. The most successful MMOG thus far, World of Warcraft from Blizzard Entertainment, had over 6 million active subscribers in July 2006 [8]. New MMOGs appear all the time. Unfortunately, most games do not provide sufficient safeguards against cheating players. Yan and Choi quote a poll putting the number of cheaters in a particular online game as high as 35% of all players [14]. In addition, since traditional client/server-based architectures do not scale well to support large numbers of simultaneous players required by MMOGs, there is much interest in peer-to-peer (P2P) architectures where players’ machines become part of the game’s infrastructure. This makes games even more vulnerable to cheating.

In the third part of our work we set out to design a fraud-resistant architecture that provides assurance to the game participants that (under reasonable assumptions) no cheating can occur and the game acts according to its security specifications.

1.2 Contributions

The most important contributions of this dissertation are listed below. In the area of Internet-scale content-based publish/subscribe networks they include:
• Development of mechanisms ensuring service integrity in Internet-scale content-based publish/subscribe networks with non-trusted intermediaries using overlay multicast.

• Design of efficient algorithms for countering "omission" attacks by malicious intermediaries in Internet-scale content-based publish/subscribe networks.

• Development of an analytical model for estimating cost of delivery trees in Internet-scale content-based publish/subscribe networks based on subscription predicate relationships and network topology information.

Our contributions to research on secure data delivery in aggregating sensor networks are:

• Development of CoDeX, a collect-detect-exclude framework for secure aggregation in sensor networks.

• Development of the "neighborhood watch" (distributed constraint validation) mechanism for secure aggregation in sensor networks.

• Development of the accompanying protocols for secure aggregation and network maintenance in sensors networks.

• Development of a server-based reputation subsystem and a probabilistically approximately correct (PAC) learning model-based subsystem for detection and removal of compromised sensors.

The main contributions of this dissertation to protection of massively multiplayer online games from cheating by the players include:

• Development of a fraud-resistant architecture for peer-to-peer environments (FRAPPE) based on the use of clustering, secure multiparty computations, threshold cryptography, and anonymity services.
• Development of a verification mechanism for local broadcast in P2P environments allowing network entities to solicit service participation via local broadcast at a remote location and verify that the remote peers honestly executed the protocol.

The final contribution reflects the common approach we took to validation of our solutions in all three areas:

• Development of simulation models to study effectiveness and efficiency of the proposed mechanisms improving service assurance in the classes of insecure networks under consideration.

1.3 Organization

This dissertation is organized as follows. Chapter 2 provides related work. Chapter 3 discusses our approach to providing service integrity in publish/subscribe networks. Chapter 4 describes security issues in aggregating sensor networks and countermeasures to stealthy attacks against them. In Chapter 5, a comprehensive fraud-resistant architecture for peer-to-peer massively multiplayer online games is presented. Finally, Chapter 6 presents our conclusions and discusses future work.
Chapter 2: Related Work

2.1 Introduction

In this chapter we discuss work contributed by other authors to the topic relevant to our research. To establish the context and validate usefulness of our solutions we first provide a brief overview of real-life applications using the technologies from each of our research areas. We then provide a taxonomy of network types in each area and the common elements (such as routing, time synchronization, localization, scheduling, etc.) our solutions rely on to provide service assurance. Finally, a separate section discusses the security issues we address in this dissertation and solutions to them proposed elsewhere.

Following this pattern, the chapter is organized as follows. Section 2.2 describes applications and examples of publish/subscribe networks. It also includes a taxonomy of PSNs. In Section 2.3 we discuss security in pub-sub networks. Sensor networks are introduced in Section 2.4. Section 2.5 discusses common security issues in sensor networks. General discussion of massively multiplayer online games (MMOG) is presented in Section 2.6. Section 2.7 covers cheating prevention in MMOGs.

2.2 Publish/Subscribe Networks

A publish/subscribe (pub-sub) system has two categories of participants: those that produce information and those that consume it. More specifically, one or more users may periodically publish data objects to the pub-sub infrastructure, and zero or more users may consume those objects. Data objects come in many forms: they may be documents (for example, articles from a news wire service), events (e.g., in a network management system), messages,
etc. Usually in a pub-sub system not every user is interested in every published object. Therefore, to conserve bandwidth a pub-sub system must provide facilities for consumers to express their interests, on the one hand, and a delivery mechanism that takes these interests into account, on the other. The former is achieved through subscriptions, advertisements submitted by consumers to the pub-sub infrastructure that contain acceptance conditions for data objects.

Perhaps, the oldest widespread electronic pub-sub system in the world is USENET [15]. USENET is organized as a collection of newsgroups. A newsgroup is a forum for discussion on a specific topic. Newsgroups form several hierarchies similar to domains in the Internet Domain Name System (DNS). Users express their interest by subscribing to a newsgroup at their ”home” USENET server with the expectation of receiving articles from this newsgroup and all its subgroups. When somebody posts a message to a newsgroup, it travels to his ”home” server and then propagates through a mesh of point-to-point links between servers until it reaches all subscribers in the newsgroup.

Pub-sub techniques are also used in distributed simulation. The High Level Architecture (HLA) - the standard framework used to conduct simulations by the U.S. Department of Defense - defines federations of simulations composed from modular components with well-defined interfaces [16]. A federation is a combination of a common object model, a set of federates, and a set of run-time services. Federates use HLA’s Declaration Management services to subscribe to specific object and interaction types, Object Management services to send and receive updates, and Data Distribution Management services to efficiently route messages and minimize transmission of unnecessary information. Subscriptions are expressed in terms of routing spaces, geographic or logical areas of interest for the subscriber. More specifically, a federate can treat objects as points in a vector space: if objects under consideration have the attribute set \( \{a_i\}_{i=1}^n \) and attribute \( a_i \)'s range is \( R_i \), every object is just a vector \( \{v_i\}_{i=1}^n \) with \( v_i \in R_i \). Then the federate can describe its interest as a vector
of ranges \( \{r_i\}_{i=1}^{\alpha} \) where \( \emptyset \neq r_i \subseteq R_i \) for all \( i \) [17].

Pub-sub architectures play an important role in multiplayer computer games. To maintain consistency nodes must exchange information about changes in their environment. Frequently only a small number of other nodes are affected by a node change. To account for this, multiplayer gaming infrastructures use interest management techniques to optimize delivery of updates based on the nodes’ interests. In some systems interest is expressed as an aura, a space that may affect - and may be affected by - an object; in others, two types of expressions are used: one to designate an observer’s area of perception, and another an observed object’s area of perceptivity [18].

In recent years software industry has shown significant interest in the pub-sub paradigm. For example, Java Message Service (JMS) lets nodes define labeled data channels called topics [19]. When a node has an object to share it publishes the object to a topic. Consumers of data express their interest by subscribing to topics. In addition, JMS messages can carry meta-attributes, and subscribers can further restrict the set of messages they receive by providing a selector, an acceptance predicate over these attributes. Another example is the Web Service Notification (WS-Notification) standards promulgated by the OASIS Consortium [20]. The standards define pub-sub communications in the context of the Web Services Architecture, the standard for business-to-business (B2B) interactions on the Web. Data producers publish notifications to topics similar to JMS and USENET. Topics may be organized into a forest. To subscribe, a consumer must provide a topic expression (over topic names and other attributes) that may resolve into zero or more topics. It can also specify: (a) a precondition, a logical expression over the data producer’s attributes, (b) a selector, a logical expression over the notification’s attributes, and (c) a subscription policy defining, for example, message rate, required delivery guarantees, etc. Pub-sub technology is at the core of many enterprise application integration products, TIBCO Rendezvous® [3] and Vitria BusinessWare™ [21] among them. It is also popular in the grass-roots
world of Web publishing: many sites create news feeds through a mechanism called Really Simple Syndication, or RSS, and subscribers can have them delivered to their desktops [22]. Popular Web portals such as Google, Yahoo! and others even allow users to create sinks for RSS feeds from other sites and present the news as part of the users’ personalized Web pages hosted by them.

In the subsections below we will use the convenient pub-sub classification system proposed in [4].

2.2.1 Subject-Based vs. Content-Based Pub-Sub

In subject-based pub-sub systems data producers label their messages as belonging to a particular category (or topic, or group, etc.), and data consumers subscribe to them by listing one or more categories they are interested in. In a content-based system the decision whom to send a message to is based on predicates advertised by subscribers. JMS [19], Scribe [23] and Bayeux [24] are examples of subject-based systems. The numerous IP multicast protocols can be said to use this approach as well: publishers send packets to a multicast group identified by a unique Class D address, and consumers of data subscribe to these packets by joining the appropriate group [25][26]. On the other hand, Siena [27] and Gryphon [28] support the content-based model.

2.2.2 Content Types and Query Languages

Most pub-sub systems impose restrictions on the types of messages they transmit. For example, USENET supports exchange of 7-bit ASCII documents [15]. Most systems discussed in this section (Siena [27], Gryphon [28], Scribe [23], etc.) model messages as structured documents. While JMS messages carry serialized Java objects as their payload, publishers can annotate them with attribute lists thereby adding structure to opaque sequences of bytes they produce. Content types are relevant here since they heavily influence design of
subscription languages used in content-based pub-sub systems. In general a subscription
can be described as \(\{f_1(t), \ldots, f_k(t) | P(t)\}\) where \(f_i(t)\) are some facets of a published object \(t\) and \(P(t)\) is a predicate over it. A facet may be an attribute in a structured object, a region
in an image, or a page, paragraph or sentence in a text document. All pub-sub systems we
know of - except HLA - allow only whole objects to be delivered to subscribers. HLA [17]
lets subscribers specify object attributes they want sent their way thus reducing traffic but
potentially increasing complexity of the system.

2.2.3 Composite Objects

Some pub-sub frameworks allow users to subscribe to composite objects, i.e., objects assembled
on the fly from atomic objects actually generated by publishers. This may be useful
in an event management system performing event correlation: some events become relevant
only if other related events do or do not occur. Siena [27] is one example of a pub-sub
system that supports this functionality to some extent.

2.2.4 System Architecture

A pub-sub system with client/server architecture has two distinct classes of nodes: clients
and servers. Servers are part of the pub-sub infrastructure; they are responsible for re-
ceiving objects from publishers and making them available to subscribers. Publishers and
subscribers are clients in this model. By contrast, the peer-to-peer (P2P) model doesn’t en-
vision any static infrastructure: end users (i.e., publishers and subscribers) themselves are
responsible for routing and delivery of published objects. JMS [19], USENET [15], Gryphon
[28] and Siena [27] use the client/server model. JMS is a centralized system with one server
acting as a mediator between publishers and subscribers. USENET servers are organized
into a mesh of point-to-point links; each client communicates only with its "home" server.
Siena supports hierarchical and mesh server configurations as well as hybrid configurations
where some parts of the network are set up as hierarchies and others, as meshes. Bayeux [24] and Scribe [23] are examples of P2P systems.

### 2.2.5 Dynamic vs. Static Membership

Pub-sub frameworks like Bayeux, Scribe, Siena and others allow clients to join and leave at will, and their delivery mechanisms are optimized to support dynamic membership. In some systems, however, membership is static and known beforehand, or is dynamic but changes slowly relative to the rate at which objects are published. For instance, in a distributed simulation (e.g., in HLA) participating federates are known upfront although simulated objects "interested" in changes in other objects may or may not exist at all times.

### 2.2.6 Matching Algorithm

A matching algorithm describes how published messages are matched against active subscriptions. It should be rather straightforward for subject-based systems although sophisticated algorithms also exist [24]. The widely used algorithms in content-based pub-sub systems are based on matching trees [28] and binary decision diagrams [27][29].

### 2.2.7 Delivery Mechanism

Both system architecture and delivery mechanism in pub-sub systems are heavily influenced by their communication style or direction. In many environments new messages are pushed to subscribers while in others subscribers must pull them. Some systems support both delivery styles. For example, a USENET server can ask its news feed server for a list of new postings by using the NNTP command NEWNEWS [15]. Alternatively, the news feed server may issue the IHAVE command to let the receiving server know what postings it has.

Most modern pub-sub systems are push-based [4]. As we mentioned earlier, there are many similarities between subject-based pub-sub systems and IP multicast. In both cases
users subscribe to a named channel (subject or address), and data objects are explicitly published to these channels. Topics in systems with flat and even hierarchical namespaces can be easily mapped to multicast groups. On the other hand, lack of universal support for IP multicast and difficulties in integrating it with content-based routing make it less attractive as a delivery mechanism for pub-sub systems [30]. Bayeux [24], Scribe [23], Siena [27] and many others use application-level (overlay) multicast protocols. Overlay multicast has the following advantages [31][32]:

- It does not require network layer support and thus can be deployed universally.
- It relieves routers from keeping state information and places heavier load on end hosts which - in many cases - only need to deal with a small number of messages.
- Approaches to reliability, flow control and security can be decoupled from packet delivery and customized in an application-specific manner.

In our model delivery trees are publisher-rooted. Bayeux and Scribe, being subject-based pub-sub systems, deterministically map each group’s name to the address of a rendezvous point, a server that becomes the root of that group’s delivery tree. Publishers send their documents towards the rendezvous point of the group they are publishing into; potential subscribers augment the group’s multicast tree by sending a join request towards the rendezvous point and receiving a confirmation along the newly created path.

### 2.2.8 Quality of Service

End-to-end delay is an important dimension of quality of service. Different applications may have different delay tolerance thresholds. Subscribers to a news delivery service may accept delays of many hours and, perhaps, even several days, while those using a trading system want information to be delivered within minutes or seconds after it’s published. Most applications have a requirement (possibly implicit) to bound the delay; most pub-sub
systems cannot guarantee such a bound, in part because they don’t want to impose any restrictions on the underlying network or transport protocol. One notable exception is the Data-Centric Publish/Subscribe (DCPS) model adopted by the Object Management Group [33]. DCPS-compliant software (e.g., NDDS from Real-Time Innovations, Inc. [34]) uses UDP over switched Ethernet LANs to provide (soft) real-time guarantees, for example, in industrial automation and sensor networks.

Applications that use the pub-sub communication paradigm vary in their QoS requirements in other areas as well. Some of them may tolerate loss of published messages; others may require guaranteed delivery. For instance, a distributed computer game could compensate some loss of position information through the use of dead reckoning, an approximation technique that predicts the future position of an object based on its recent history [18]. Other applications are more stringent. Bayeux [24], Scribe [23] and Siena [27] offer best effort service. JMS [19] and Gryphon [28] support guaranteed delivery. Both let clients set a durability flag in their subscriptions. If a ”durable” subscriber fails and later recovers, it will receive all undelivered messages matching its subscription.

All ”concrete” pub-sub systems we use for comparison are fault-tolerant to some degree. Gryphon organizes its servers (brokers) into multiple cells [28]. Servers within a cell are fully connected. Cells themselves form a mesh network with redundant links that allow the system to route messages around failed servers. Scribe achieves fault tolerance through the use of replicated delivery trees. It also requires nodes to send periodic heartbeats to their children. If a heartbeat is missing, the child starts a tree repair algorithm to reconnect its subtree to one of the roots.

2.3 Security in Publish/Subscribe Networks

In this section we discuss security issues in pub-sub networks. Our work focuses on content-based systems and, given the advantages outlined in Section 2.2.7, on application-level,
or overlay, multicast. We consider pub-sub networks spanning the Internet and assume they will use communication and security protocols available on the Internet (e.g., the IP protocol, TCP, SSL/TLS, etc.) Many security issues are not specific to pub-sub networks; we will discuss them only in passing. Instead, we will focus on application-layer issues unique to public/subscribe infrastructures. There are four families of security concerns in a pub-sub system:

- Service integrity and availability which include all aspects of proper functioning of the infrastructure of a pub-sub network.
- Authorized access to published documents.
- Authenticity of publishers and subscribers in support of authorization and accountability.
- User, message and subscription confidentiality.

Their detailed discussion is provided below.

2.3.1 Service Integrity

In this section we focus on service integrity in pub-sub networks. Since service assurance is the topic of our research, we’d like to discuss solutions proposed in the networking and security literature. This section describes deficiencies of the current approaches and contrasts them with our ideas fleshed out in Chapter 3.

When a pub-sub network accepts a subscription from a user it enters into a contract promising that a published object will be delivered to it if and only if it satisfies the subscription. A compromised intermediate node may cause the system to violate this contract by inserting, deleting, modifying, reordering, misdirecting, and delaying messages. A large-scale pub-sub infrastructure is likely to include components from multiple network providers, not necessarily trusting one another or trusted by the end users.
Feige et al. divide their pub-sub network called Rebeca into overlapping scopes, essentially communities with common security requirements [35]. Rebeca allows hosts from different administrative domains (e.g., network providers) to join the network. Members of a given administrative domain trust one another. Permissions are granted based on attribute certificates [36] issued by trusted attribute authorities (AA). Attribute authorities form a trust forest; two parties are in a trust relationship if they hold attribute certificates issued by AAs with a common ancestor. Imagine that a broker (intermediate node), say, A, participating in some Rebeca scope S, receives a scope join request from another broker, B. Three situations are possible:

- A and B belong to the same administrative domain.
- A and B are from different domains but are in a trust relationship (i.e., have a common ancestor attribute authority).
- A and B are not in a trust relationship.

In the first two cases B will be admitted into S subject to its attributes’ satisfying security requirements of the scope. (Conceivably, in the first case B may be allowed to join unconditionally.) In the third case Rebeca will let B join the scope but will not entrust message content to it: the system will build an encrypted tunnel through B to either the subscriber that caused B to attempt to join the scope, or to another trusted broker. Rebeca’s is a sensible approach to service assurance for pub-sub networks with pre-existing trust relationships. Such relationships incur a significant setup cost; they also require significant expenditures on accreditation and auditing. For a pub-sub service to be viable it must impose as few entry barriers for new network providers as possible.

Wang et al. [37] suggest multi-path routing as one possible means to guarantee service integrity in pub-sub systems. This is consistent with the many proposals in networking literature dealing with fault tolerance, including tolerance of security faults, in other types
of networks. [37] does not offer any specific solutions, however. Castro et al. [38] consider secure routing in Pastry, the P2P messaging infrastructure used in Scribe. Their work seeks to ensure that each message reaches - with high probability - the rendezvous points for the group it was published into. Although offering many useful ideas, Castro et al.’s approach does not solve the problem of secure end-to-end routing, all the way from publishers to subscribers.

Much work has been done on providing security through multi-path routing in other environments, for example, in ad hoc and sensor networks. Ganesan et al. [39] also use multi-path routing but with the goal of rapidly discovering faulty nodes and creating routing paths that navigate around them. Discovering faults in a pub-sub network is difficult since subscribers cannot detect undelivered messages they were supposed to receive. Thus, Ganesan et al.’s approach is not applicable to our environment. Other approaches such as dispersity routing suffer from the same deficiency.

Srivatsa and Liu [40] provide service integrity in their EventGuard pub-sub network by constructing an n-ary delivery tree rooted at the publisher and augmenting it with additional links connecting nodes to \( k < n \) of their ”uncle” nodes in the tree. This construction creates exactly \( k \) independent paths between the publisher and any subscriber. If the number of malicious intermediate nodes is small, chances are every message will reach its destination at least once. This solution does not address the threat of collusion between malicious nodes. The nodes may be geographically dispersed but not independent, so if they collude (or under control of a single adversary) effectiveness of Srivatsa and Liu’s algorithm is greatly diminished.

2.3.2 Authorized Access

A pub-sub network incorporates three types of entities: publishers, subscribers, and the infrastructure. Each type may impose access control requirements on the other two. For
example, a commercial news distribution service will want to deliver articles only to members and among them only to those in good standing. On the other hand, publishers in a syndicated network don’t care about the subscribers but the network itself (the infrastructure) does. Publishers of sensitive information will want only sufficiently cleared users to read their dispatches. Equally, subscribers will want to receive information only from reputable sources.

In general, authorization policies grant or deny access based on the attributes of the resource being requested and the attributes of the requestor. For this to work, both types of attributes must be authentic. This requires pub-sub networks with access control to implement an authentication service.

Many access policy models have been proposed over the years, some of which are applicable to pub-sub networks. Belokosztolszki et al. propose to use role-based access control (RBAC) for authorization in pub-sub [41]. Some authors advocate the rule-based approach. Opyrchal et al. [42] use the KeyNote trust management system [43] to define access policies for the location tracking service prototyped at the University of Michigan. The service is capable of tracking individuals on campus using location sensors and distributing location information to users in the form of events through a publish/subscribe infrastructure. A policy entry includes the name of the granting party, the user name, zero or more actions (such as subscribe, publish, receive, change a policy, etc.), and a set of conditions. Each condition is expressed as a SQL-like logical expression over event attributes.

As we discussed in the subsection on service integrity, Feige et al. grant access to Rebeca scopes based on attribute certificates [35]. An attribute in Rebeca expresses constraints on content a subscriber may receive or a publisher forward.

When should the access control policies be applied? It depends on the application. Opyrchal et al.’s system [42] provides means to indicate when an event access policy must be evaluated: at the time of subscription, at the time of reception, or both. Rebeca evaluates
applicable policies when the user joins a scope in the pub-sub network. Many commercial pub-sub services practice usage-based access control: access policies are checked every time a message is used. For example, Qmags, a subscription service publishing electronic copies of many popular magazines, uses digital rights management (DRM) features in the Adobe® Acrobat Reader to ensure that only authorized users can access the content after it’s downloaded to their personal computers [44]. (In DRM-enabled systems information objects are usually encrypted, and decryption keys are given only to authorized users.) Policy enforcement is performed every time the file with the magazine is opened.

2.3.3 Authenticity

There are three issues related to authenticity in pub-sub networks:

- Message authenticity: publisher Alice may impersonate publisher Bob and publish messages on his behalf. This will enable Alice to publish false information and mislead her subscribers taking advantage of the trust they put in Bob.

- Subscriber authenticity: subscriber Alice may impersonate subscriber Bob and receive messages that she is not allowed to see but Bob is. If she succeeds, Alice will gain unauthorized access to published information.

- Subscription authenticity: subscriber Alice may impersonate subscriber Bob and induce the pub-sub network to deliver messages, satisfying her subscription, to Bob. If the subscription’s query expression is NOT Elvis¹, Age >= 0, or TRUE, Bob, in all likelihood, will be inundated with worthless messages which will result in a denial of service to him.

Authenticity of a message can be proved through the use of public key cryptography: if a publisher digitally signs messages using his private key, subscribers can easily verify them.

¹Invented by David Holtzman, former Chief Scientist of IBM’s infoMarket Project
2.3.4 Message and Subscription Integrity

Integrity can be guaranteed by the same means as authenticity, i.e., through the use of digital signatures or message authentication codes. See the subsection above for more details.

2.3.5 Availability

A large-scale pub-sub network with dynamic membership and no admission control for published messages may be vulnerable to denial-of-service attacks, especially by bogus publishers. Even if messages are authenticated, a malicious publisher may successfully overwhelm the network: all its resources will be spent verifying signatures or message authentication codes and rejecting non-matching messages.

One simple approach employed by Waldman et al. in their Publius system is limiting the size of published messages [45]. (Publius’s messages are at most 100 kilobytes long.) Likewise, frequency of publication by each publisher can be restricted. Yet another approach involves client puzzles, computationally intensive tasks systems must perform before initiating interactions with other systems [46]. Each request, connection or message, depending on the application, must be accompanied by an easy to verify unforgeable solution to a puzzle; if no solution is presented, the request is denied, the connection is not established, the message is dropped. Wang et al. [37] recommend Hashcash, an early client puzzle implementation by Back [47], for use in pub-sub networks.

2.3.6 Confidentiality

Publishers in a pub-sub system may require that messages be delivered only to authorized parties, for instance, paying customers, registered users, principals with sufficient clearance,
etc. If the pub-sub infrastructure itself is trusted, access control mechanisms at the end points should suffice to satisfy this requirement; if not, messages can be encrypted to provide confidentiality. If the infrastructure elements are trusted but need to use insecure networks for communication, they can take advantage of transport (e.g., the SSL/TLS protocol [48]) or network layer (e.g., IPSec [49]) mechanisms to achieve hop-by-hop confidentiality.

In a topic-based system publishers can encrypt their messages and subscribers can decrypt them at the other end. For example, EventGuard [40] sets up a per-topic private key and distributes it out of band to eligible subscribers when they log in. Public key cryptography may be used as well.

In a content-based system end-to-end confidentiality clashes with the need for intermediate nodes to inspect messages and route them based on content. One simple solution is to annotate encrypted objects and transmit the annotations in the clear along with the objects themselves. The Java Message Service [19], for example, allows publishers to create message envelopes with attributes, and subscribers can express their interest to the system using SQL-like logical expressions over those attributes. One interesting approach involves computing with encrypted data proposed by Feigenbaum [50] and expanded by Abadi et al. [51] and by others. The problem consists of two parties $A$ and $B$: $A$ possesses some data item $x$, $B$ has the ability to compute some function $f(y)$ for any input $y$, and $A$ wants $B$ to compute $f(x)$ without revealing $x$. Such a computation is possible if there exist two efficiently computable functions $E$ and $D$ such that $D(f(x')) = f(x)$ if $E(x) = x'$.

Some applications may require confidentiality of subscriptions [37]. Again, this requirement is at cross-purposes with content-based message filtering and routing. If the real need is only to de-link users from their subscriptions (but not to make subscriptions secret) this can be achieved if the pub-sub network provides anonymity. Otherwise, users can encrypt their subscriptions. Secure circuit evaluation can be used to match messages against encrypted subscriptions. The problem of secure circuit evaluation, studied by Abadi and
Feigenbaum [52], consists of two parties A and B: A possesses some data item x, B has the ability to compute some function \( f(y) \) for any input \( y \), A wants B to compute \( f(x) \) without revealing \( x \), and B is willing to do so but wants to keep the function \( f \) secret. Abadi and Feigenbaum proposed an interactive protocol to solve this problem.

More recently, Raiciu and Rosenblum [53] studied practical methods for achieving confidentiality of notifications and subscriptions in pub-sub networks. In their setting intermediate nodes are honest but curious: they follow the pub-sub system’s rules but at the same time want to learn notifications, subscriptions, and/or their relationships. One of their protocols called LSCP assumes that the publisher and the subscribers share a secret key used, on the one hand, to encrypt published objects and, on the other, to ”garble” Boolean circuits representing subscriptions. A non-trusted intermediate node feeds an encrypted notification into the encrypted circuit and receives a clear-text bit indicating whether they match each other. Other methods referenced or proposed by [53] include equality-based filtering (achieved through the comparison of delivered and desired encrypted values), range-based matching (achieved through the reduction of range matches to sets of prefix matches and the use of a prefix-matching-preserving encryption method) and keyword-based matching (achieved through the use of encrypted Bloom filters by both the publisher (to represent all keywords in a notification) and the subscribers (to represent the keywords of interest)).

### 2.3.7 User Anonymity

Publish/subscribe networks separate publishers and subscribers in time and space, and thus provide an excellent means for subscriber anonymity vis-a-vis publishers. (In some applications publishers do want to know who receives their information, e.g., for billing purposes. A pub-sub infrastructure, trusted by publishers, can perform billing on their behalf and thus still support subscriber anonymity.) It is more difficult to provide subscriber anonymity vis-a-vis the network itself. Encrypted subscriptions discussed in the subsection
on confidentiality provide subscription anonymity but cannot hide the fact that some user received (or did not receive) a particular message. Wang et al. [37] point out that only the last intermediate node on the shortest path from a publisher to a subscriber needs to know who the subscriber is. If we impose a requirement that all subscribers connect to the network through a trusted entry point, we can guarantee end-user anonymity at the cost of introducing an extra hop in each delivery path.

Multiple systems were also proposed to provide publisher anonymity to combat censorship on the Web.

### 2.4 Aggregating Sensor Networks

Sensor networks recently emerged as a promising technology for monitoring applications. A node in a sensor network (SN) is a small device equipped with sensors to monitor physical or environmental conditions, a microprocessor, a radio transceiver and an energy source (e.g., a battery). Figure 2.1 shows a representative sensor node manufactured by Moteiv Corp. [54]. The mote's size (not including the USB connector and the two AA-battery compartment) is $0.256 \times 2.580 \times 1.267$ in. It contains components typical of such devices:

- An 8MHz microprocessor (part of the Texas Instruments-made microcontroller unit in Figure 1b).

- 10 Kb RAM (part of the TI microcontroller, Figure 2.1(b)).

- 48 Kb Flash memory (part of the TI microcontroller, Figure 2.1(b)).

- A 250 Kbps wireless radio transceiver (Chipcon’s CC2420 in Figure 2.1(a)).

- An antenna with 50 m indoor and 125 m outdoor range (Figure 2.1(a)).

- Humidity, temperature and light sensors (Figure 2.1(a)).
• A USB interface for connecting to a host PC (Figure 2.1(a)).

A typical sensor network is shown in Figure 2.2. The system consists of three parts:

• A server, the ultimate consumer of information from the sensor network.

• A network of inexpensive, computationally-constrained sensors communicating via low-power radio.

• One or more base stations, more powerful computers communicating with the sensor network via radio and with the server via a wired network. All communications between the server and the network pass through the base station(s). Note that in some models sensors communicate directly with the server thus obviating the need for base stations.

Sensor networks pose challenges not seen in other types of networks. The reasons are manifold and have to do with their size, cost, communication medium, and deployment characteristics [55]:

• Sensors are usually deployed at random and thus do not form networks with well-defined topologies. In most scenarios they operate unattended. This requires sensor network setup and maintenance to be completely autonomous.

• Sensor networks do not own any identifiable infrastructure: routing and maintenance algorithms must be distributed and supported by the sensors themselves.

• Sensors usually rely only on their battery for power which may be difficult or undesirable to recharge or replace. Thus, preference must be given to energy-efficient algorithms and protocols.

• Sensor nodes should be able to synchronize with each other, so that transmission schedules can be constructed and temporal ordering of events can be performed.
Figure 2.1: Representative sensor node, Moteiv Corp.’s Tmote Sky [54]
A sensor network must be able to adapt to node failures. The routing protocols should be able to construct new routes based on node availability and latency information.

Real-time communication in sensor networks must be supported to provide maximum delay, minimum bandwidth, or other quality of service (QoS) guarantees.

Sensor networks must be secure against denial-of-service, impersonation, eavesdropping and other threats, especially in adversarial environments.

Sensor networks are used in real-time highway traffic analysis, machinery and process monitoring, surveillance, structural health prognostication, disaster recovery, air and water quality control, wildlife management, and other areas. For example, Sensys Networks, Inc. produces motes with built-in magneto-resistant sensors that get installed in the surface of measured roadways [56]. Sensors measure changes in the Earth’s magnetic field induced by passing cars and communicate their measurements to roadside access points using low-power radio transmitters. The access points, in turn, aggregate data and forward them to a traffic control center. Sensys motes allow measurement of occupancy, vehicle speed, length, and other parameters.

Rockwell Scientific and Boeing are jointly working on a sensor network-based system for health monitoring of space vehicles [57]. A reconfigurable sensor network will perform different functions at different points in the spaceship’s flight. For example, during liftoff
the sensors will measure vibration levels while in flight and during re-entry they will monitor structural changes due to significant temperature fluctuations.

Rockwell is also working on a sensor network-based system for area monitoring and military surveillance. The system will implement vibration, acoustic and magnetic signal classification algorithms, and use both on-board and in-network data fusion mechanisms for threat identification and localization.

Cooperative sensor networks can be used for health monitoring of natural and man-made structures. Werner-Allen et al. [10] report on a study of the Ecuadorian volcano Reventador they performed using a network of motes equipped with seismic and acoustic sensors. The motes communicated via wireless radio links, passing information to several radio modems which maintained long-distance, reliable radio connections with a base station in the observatory. Since in volcanic monitoring it is important to accurately timestamp sampled data, a GPS receiver was installed to provide a time synchronization signal to the network. Each node sampled seismoacoustic data provided by its sensors; having detected an interesting event, a node sent a message to the base station. If a sufficient number of nodes reported the event, the base station queried the network for extra information. The nodes recorded the sensors’ measurements for 60 seconds and forwarded them back to the base station.

Delaney et al. [12] describe NEPTUNE, a large-scale multi-purpose system for ocean and earth exploration. The system features multiple sensor networks above, on and below the sea floor. Glaser discusses the use of sensor networks in monitoring of structural changes in buildings, caused by earthquakes and explosions [13]. Zhang [58] is implementing a wireless sensor network on a cable-stayed bridge in eastern China to monitor its structural health.

Sensor networks provide an excellent tool for wildlife habitat monitoring. Mainwaring et al. [11] describe an SN-based system used to study habits of Leach’s petrels on Great Duck Island in Maine. Leach’s petrel is a relatively small seabird that spends most of its
life on the open ocean; it breeds on remote islands in the Atlantic and the Pacific. Petrels lay and incubate eggs in burrows dug into soil or sand. Because of their reclusive habits petrels are hard to study in the wild, and this is where sensor networks may be of great help. Mainwaring et al.’s system consists of several clusters of sensors placed in burrows and on the surface, a transit network connecting remote clusters to the base station, and the base station itself interacting with the transit network, on the one hand, and the Internet, on the other. The sensors are outfitted with light-, temperature-, humidity- and barometric pressure-sensing units; these measurements provide rich raw data that can be used to infer behavioral patterns of the birds under study. In one experiment the authors could remotely detect the presence or absence of a parent by studying temperature measurements in a burrow (it is approximately 6° higher when an adult is present). A robust data gathering and archiving approach used by the system allows selective propagation and storage of data from in situ sensor clusters.

2.4.1 Sensor Network Architectures

The two basic types of sensor network architecture are: (a) layered architecture and (b) clustered architecture. The simplest layered network has a base station (BS) surrounded by circular bands of sensors, called layers. By construction, sensors located within the transmission ranges from the BS are combined into the $i$th layer as shown in Figure 2.3.

A clustered architecture groups sensors into clusters that can be thought of as sensor sub-networks. Each cluster has a specially designated node, called a cluster head, which serves as a router between the cluster and the rest of the network. Figure 2.4 presents an example of a clustered network.
Figure 2.3: Layered sensor network architecture [55]

Figure 2.4: Clustered sensor network architecture [55]
2.4.2 Network Routing Protocols

Routing in sensor networks is especially challenging because it has to rely on fault-prone nodes, use an unreliable transmission medium, and be energy-efficient. Fortunately, most applications require a sensor network that supports only a limited set of communication models. The most frequently used models include [9]:

- Dissemination. Information flows from a single source (a base station or a sensor) to multiple sensor nodes. This model may be used for network maintenance, time synchronization, reprogramming of the nodes, and query broadcasting.

- Data gathering. Some nodes collect information from their neighbors, clusters or other types of ”basins”. Relationship between sources (original or intermediate) and sinks (intermediate or final) in this model is many-to-many or many-to-one. The data gathering model is especially suitable for in-network data processing, filtering, fusion and aggregation.

- Any-to-any communication. This model supports complex interactions among sensors, for example, to enable load balancing and implementation of distributed algorithms and multi-party protocols.

Many routing protocols were proposed for sensor networks. We will briefly review several of them. The simplest approach to routing is flooding: broadcast your message to all neighbors with the expectation that they re-broadcast every message they receive; eventually the ”ripple” will reach the target node. There are many well-known problems with flooding (see, for example, [1]). In a sensor network they are more acute. A message straying far from an optimal route between the source and the target or delivered multiple times to the same node wastes energy, which, in the end, will reduce the lifespan of the sensor network. Flooding may be less burdensome for messages intended for all nodes (i.e., in network-wide
However, its blindness to energy availability at individual nodes may lead either to partial message delivery (if low-battery nodes stop forwarding) or premature node failures (if they do not) [59]. *Gossip routing*, a variant of flooding, involves sending messages to a subset of neighbors selected at random. Gossip routing may be suitable for data gathering applications, but, if used for data dissemination, it takes significant time to reach all nodes in the network [55].

Braginsky and Estrin [60] proposed a mechanism called *rumor routing*. It is suitable for sensor networks where a sensor (or a group of sensors) detects an event, and other sensors or the base station are interested in receiving additional information about the event. Rumor routing is based on two premises: (a) in many applications communication paths do not need to be optimal and (b) two random straight lines in a plane are very likely to intersect, so if one builds a random (almost straight) path from the event’s source and another random (almost straight) path from the event’s sink, these paths will intersect with high probability. Every node in the network maintains a table of entries each containing an event name, the distance to the source, and the best next hop towards it. When a node detects an event it appends the event to the table and launches a long-lived packet (called an *agent*) containing the event name and the distance (initially set to 0) to a random neighbor.

The agent performs a random walk of the network depositing or, if required, merging its information at each visited node. Similarly, if a node becomes interested in an event it sends a query towards the event’s source (if it has a route to it) or at random (if it doesn’t). If the query reaches a node previously visited by the event’s agent, it retraces the agent’s path towards the source; if not, the interested node can retransmit the random request, give up or flood the network. The *directed diffusion* routing technique due to Intanagonwiwat et al. [61] serves a similar purpose: an interested node sends a query to a region where an event should be monitored, setting soft state at intermediate nodes. Queries usually indicate the type of event as well as the required duration and frequency of subsequent reports from
the source. (The initial required frequency is lower than the true frequency desired by the node.) Once the event takes place, the source uses the distributed soft state to propagate its reports retracing the multiple paths traversed by the query. When an event sink receives the first reports, it periodically reissues the query setting the frequency parameter to a higher value. This has the effect of reinforcing some of the reverse paths and weakening others bringing down the number of paths but still providing the necessary tolerance to node failures.

Another approach to routing in sensor networks is geographic routing. Karp and Kung [62] proposed a variant of geographic routing, called greedy perimeter stateless routing, or GPSR. In this approach a node greedily forwards packets to a neighbor situated (a) closer to the destination than itself and (b) nearest to the destination of all its neighbors. To support cases when a node is nearer to the destination than any of its neighbors GPSR constructs a specialized planar sub-graph of the original topology graph and routes packets along perimeters of the faces of the new sub-graph with the ultimate goal of escaping the local minimum and returning to the use of the greedy algorithm. GPSR can be used to support any-to-any communications in distributed data storage applications [9].

2.4.3 Media Access Control

Media access control (MAC) protocols in sensor networks must ensure fair and efficient sharing of communication resources between nodes. We are convinced that understanding of different options for media access control is important when designing protocols for secure aggregation in sensor networks. In an SN environment MAC protocols need to be power-efficient and quickly adapt to topology changes caused by node failures. There are three basic types of MAC protocols in sensor networks [55]:

- Fixed-allocation.
• Demand-based.

• Contention-based.

Fixed-allocation protocols control access to the common medium through a static assignment, usually generated during the network’s initialization and setup. They are well suited for sensor networks with deterministic traffic patterns such as those continuously monitoring and reporting on some phenomenon. Fixed-allocation protocols guarantee a bound delay for each node. In an environment with bursty traffic, however, they may result in inefficient utilization of the network and wasted bandwidth.

A demand-based protocol seeks to fairly accommodate dynamic bandwidth requirements of the network nodes. Although they require the overhead of an additional reservation protocol, demand-based MAC protocols are well suited for applications with variable rate traffic.

Contention-based MAC protocols involve random access-based contention for the communication medium among multiple network nodes. Because of their stochastic nature, these protocols must deal with collisions and retransmissions. They are quite suitable for bursty traffic, but cannot guarantee bounded delays and as such are not a good fit for delay-sensitive or real-time applications.

2.4.4 Reliable Communications

When discussing reliable transport protocols in sensor networks, one must distinguish between the different traffic patterns (see the subsection on routing above). The data gathering model is characterized by inherent correlation in the data flows produced by sensor nodes. Because of this, traditional approaches to reliable transport, based on TCP’s end-to-end reliability model, are superfluous and energy-draining [63]. Besides, they have significant buffering requirements, and most sensors have very limited memory, about 4 to 16 Kb.
Sankarasubramaniam et al. proposed ESRT, an event-to-sink reliable transport protocol [64], suitable for monitoring applications where multiple sensors may detect and report the same event. The protocol exploits this temporal and spatial correlation between the many flows towards the sink to achieve reliable and energy-efficient data gathering.

On the other hand, sink-to-sensors traffic, characterizing the dissemination communication model, mostly requires 100% reliability [63]. Some amount of retransmission and acknowledgement is, therefore, necessary. To maintain optimal energy expenditures preference must be given to local retransmissions and negative acknowledgement (NACK) mechanisms over end-to-end retransmissions and acknowledgements. One solution is provided by the GARUDA framework proposed by Park et al. [65]. When the sink is ready to transmit a message, it notifies the network by transmitting a special sequence of pulses. Having received a pulse, each node starts pulsing as well; this wave propagates until all nodes are ready to receive the first packet. To ensure reliability of delivery of the first (and only the first) packet GARUDA uses positive acknowledgements. Negative acknowledgements are a more economical mechanism but packet loss detection by recipients is possible if only some (but not all) packets in a message are lost. Since many messages in GARUDA are short (e.g., queries) the network requires that at least one packet in each message is provably delivered. Responding to the notification, the SN constructs a core, a near-optimal network of routing nodes similar to routing infrastructures in multicast networks. When a core "router" receives an out-of-sequence packet (indicating packet loss), it initiates a two-stage NACK-based packet recovery protocol. GARUDA also supports specialized delivery modes useful in sink-to-sensors communications: (a) reliable delivery to a region in the sensor network, (b) reliable delivery to a minimal subset of sensors covering the SN’s area and (c) reliable delivery to a probabilistic subset of sensors in the SN.

Simon et al. [66] proposed a reliable protocol for dissemination of large objects in sensor networks called McTorrent. In McTorrent sensors use a common control channel to advertise
SELECT attribute-1, attribute-2, ..., attribute-n
FROM sensors
WHERE condition
SAMPLE PERIOD frequency FOR duration

Figure 2.5: Typical query in TinyDB [67]

available pages of the large object in question, and request such pages. Actual delivery of
the pages happens over separate data channels announced via the CHANNEL message on
the common control channel. Like GARUDA, McTorrent also uses selective NACKs and
hop-by-hop error recovery to achieve reliability.

2.4.5 Data Aggregation

We explained in Chapter 1 that many applications of sensor networks are centered on
monitoring physical phenomena, collecting data about them and reporting them to a central
location for processing. Many systems attempt to perform preliminary in-network or even
local processing of raw data in order to minimize network traffic and energy consumption. In
their TinyDB system Madden et al. [67] model a sensor network as a table (called sensors)
in a relational database. The base station can issue queries against its sensor network and
collect corresponding tuples. The query language used by the authors is a variant of SQL
augmented to support sampling and other sensor network-specific functionality. A typical
query is shown in Figure 2.5: the base station asks the network for a set of attributes from
sensors satisfying a condition; the query is to be active for a period given by the duration
parameter and evaluated during that period at a rate specified by the frequency parameter.

TinyDB provides facilities for spatial and temporal aggregates, and event-based queries
that start and/or stop sampling in response to an event. It also supports lifetime-based
queries. The query shown in Figure 2.6 specifies that the network is to sample light and
acceleration sensors at the maximum possible rate that does not deplete its energy resources

37
SELECT node-id, acceleration
FROM sensors
LIFETIME 30 days

Figure 2.6: Lifetime-based query in TinyDB [67]

for at least 30 days. Madden et al. provide a mechanism for nodes to estimate their lifetime
and a simple protocol for them to adjust their sampling rates based on the sampling rates
of their parents in the delivery tree.

In an earlier project, called TAG (for Tiny AGgregation), Madden et al. [68] proposed a
useful taxonomy of aggregates in sensor networks. Before describing their approach, we note
that to compute aggregates the authors use aggregate decomposition common to database
systems. Aggregation is implemented through the use of three functions:

- The merging function \( f \) for combining partial state records, intermediate results
carried up the aggregation tree in order to compute the final result. For example,
\( \langle \lambda.count, \lambda.value \rangle \) is a partial state record for the aggregate function AVG (see Table
4.1).

- The initializer \( i \) for computing the initial partial state record from a single value. The
initializer for the function AVG is \( i(t) = \langle 1, t \rangle \).

- The evaluator \( e \) takes a partial state record and computes the final result. AVG’s
evaluator is \( e(\langle \lambda.count, \lambda.value \rangle) = \lambda.value \).

The first taxonomy dimension is sensitivity to duplicates. For example, the functions
MIN and MAX are not sensitive to duplicates (i.e., return the same correct value when dupli-
cates are present) whereas COUNT, AVG, and MEDIAN are. Clearly, networks using multi-
path routing to achieve fault tolerance must cope with this: MIN- and MAX-computing
applications can forward their results to any number of parents but those dealing with
duplicate-sensitive functions need to be more subtle. In the case of the function COUNT Madden et al. demonstrate that a node reporting some value $c$ is better off sending $c/2$ to two parents than $c$ to a single one.

Second, exemplary aggregates return representative values from the set of all values while summary aggregates compute some property over all values. Exemplary aggregates behave unpredictably in the presence of failures; a summary attribute, on the other hand, in many cases can be treated as an approximation of the real value if only a subset of nodes is available for computation.

The third taxonomy dimension is monotonicity. An aggregate is monotonic if for any two partial state records $s_1$ and $s_2$ and $s' = f(s_1, s_2)$ either $e(s') \geq \max e(s_1, s_2)$ or $e(s') \leq \min e(s_1, s_2)$ holds. This is important when determining whether some predicates (such as HAVING) can be applied in-network, before the final value of the aggregate is known.

Finally, the fourth dimension relates to the amount of information in partial state records. Madden et al. distinguish the following types of aggregates:

- **Distributive**: the partial state is the aggregate itself for the portion of data it was computed over. MIN, MAX, COUNT and SUM are examples of distributive aggregator functions.

- **Algebraic**: the partial state is not an aggregate but is of constant size. AVG is an example of an algebraic aggregate.

- **Holistic**: the partial state record is proportional in size to the number of elements in the partition. An example of such an aggregate is MEDIAN for which no useful partial aggregation can be done.

- **Unique**: the partial state record is proportional in size to the number of distinct values in the partition. The COUNT DISTINCT function is a trivial example of a unique aggregate.
• **Content-sensitive**: the partial state record is proportional in size to some property of the data values in the partition. The function HISTOGRAM is one example of a content-sensitive aggregate.

Based on this analysis, Madden et al. proposed techniques for efficient query evaluation and distribution of aggregation results.

We conclude with an example of efficient non-hierarchical aggregation due to Guestrin et al. [69]. They try to find compressed representations of data based on spatial and temporal correlation of measurements common in sensor networks. In its simplest form their approach is to approximate a series of sensor readings with a low-degree polynomial and report its coefficients rather than the readings themselves. Let’s say, we need to report \( m \) readings \( \{f(t_i)\}_{i=1}^{m} \) and would like to approximate the function \( f \) with a polynomial \( \hat{f}(t) = \sum_{j=0}^{k} w_j t^j \) where \( k \ll m \). Our goal is to minimize the difference between \( f(t) \) and \( \hat{f}(t) \). Using the root mean squared error (RMS)

\[
\sqrt{\frac{\sum_{i=1}^{m} (f(t) - \hat{f}(t))^2}{m}}
\]

as the goodness-of-fit measure we can reduce the problem to a system of linear equations (by taking partial derivatives and setting them to zero).

### 2.4.6 Localization

We plan to use a location-based node ranking algorithm to construct cycle-free aggregation graphs. Therefore, an inexpensive and accurate mechanism is needed for discovery of location by all sensors, at least relative to the location of known objects or other sensors.

One approach is to use global positioning system (GPS) receivers and require them to send periodic beacon messages to other nodes in the network [55]. Sensors can estimate their location relative to the GPS receivers by measuring signal strength, angle of arrival,
and/or time difference between arrivals of beacon messages from multiple beacons. Similar techniques can be used with fixed beacon nodes, for example, in large buildings, where the use of GPS receivers is not possible.

2.4.7 Time Synchronization

Synchronization among nodes in sensor networks is required to support TDMA schemes for media access control. We expect to use TDMA-based scheduling in our security protocols; we briefly discuss time synchronization to demonstrate that TDMA support is developed well enough to be used as a building block. (In some applications time synchronization is useful for determining temporal ordering of events. Others, such as monitoring of seismic activity around volcanoes, may require accurate event time-stamping [10].)

Many synchronization algorithms rely on time information received from a global positioning system (GPS) [55]. The accuracy of time synchronization in this case depends on the number of satellites observed by the GPS receiver. Some consumer GPS receivers can synchronize nodes to a persistent-lifetime time standard to precision of 200 ns [70]. GPS receivers are costly, however. In addition, GPS-based time synchronization cannot be used in environments unfriendly to GPS such as in large buildings, underwater, underground, and so on. Many approaches to time synchronization without GPS were put forward. For example, Elson and Estrin [70] proposed a post-facto synchronization scheme that assumes that sensors’ clocks are not synchronized most of the time, but need synchronization when certain events occur. When such an event occurs, each node within reach records the event time with respect to its own local clock. Immediately afterwards, a "third party" node - acting as a beacon - broadcasts a synchronization pulse to all nodes in the area which, in turn, use it as an instantaneous time reference and can normalize their event timestamps with respect to that reference.
2.5 Security Issues in Sensor Networks

Sensor networks create unique security challenges because of their limited computational resources, low power, low-bandwidth communication technologies (radio or acoustic waves), and deployment environments that afford little direct control to their operators. Threats to sensor networks may be defined as either *internal* or *external* [71]. Internal threats are posed by some of the sensors (e.g., compromised or under control of malicious insiders) in the network; external threats are posed by outside sources. If an attacker compromises a sensor and steals all its data including its key material an external threat turns into an internal one. There are three families of security threats to sensor networks:

- Denial of service (DoS). DoS attacks seek to disrupt the operation of a sensor network. Methods for such attacks vary from physical destruction of sensors and base stations to jamming of radio links to manipulation of the routing infrastructure. DoS attacks may be mounted by both insiders and outsiders.

- Deception. An adversary may attempt to impair the decision making ability of the users of a sensor network by injecting or dropping messages, manipulating aggregator nodes, physically distorting the sensed environment, and so on. Both insiders and outsiders may attempt to deceive a network.

- Eavesdropping. If sensor network carries sensitive data, an adversary may want to compromise one or more nodes to gain access to it. He may also eavesdrop on the sensors with the goal of mounting a DoS or a deception attack, or to circumvent the network’s countermeasures against such attacks. For example, traffic analysis can help him identify base stations for subsequent physical destruction. Eavesdropping is an external attack since, presumably, insiders already have access to the network’s data or can easily gain it.\(^2\)

\(^2\)A related threat rarely discussed in literature is the threat of discovery when the adversary must not
To analyze security in a sensor network it is useful to distinguish between adversaries with different capabilities. Karlof and Wagner [71] consider two levels: mote-class and laptop-class. Mote-class attackers have at their disposal several sensors comparable with the sensors they are going to attack. Laptop-class attackers, on the other hand, have more powerful machines with long-range radio transmitters and ample power supply. Deng et al. identified the following requirements for security mechanisms in sensor networks [73]:

- Small memory footprint.
- Computational efficiency.
- Energy efficiency.
- Bandwidth efficiency.
- Tolerance to node compromise.

In the rest of this section we will discuss sensor network security in more detail following the general outline used in [73].

### 2.5.1 Cryptographic Primitives

Perrig et al. [74] proposed a simple, low-overhead communication protocol suitable for sensor networks, called SNEP (Sensor Network Encryption Protocol). In addition to encryption, it provides message authentication, integrity and freshness for point-to-point communications between nodes. SNEP uses the DES encryption algorithm as a basic building block for encryption and calculation of message authentication codes (MACs). Any pair of nodes A and B are pre-configured with a master key $K^M_{AB}$. Using a pseudo-random function $f(K^M_{AB}, n)$, they generate unidirectional encryption keys $K^e_{AB} = f(K^M_{AB}, 1)$ and $K^e_{BA} = f(K^M_{AB}, 3)$, and authentication keys and $K^a_{AB} = f(K^M_{AB}, 2)$ and $K^a_{BA} = f(K^M_{AB}, 4)$.
In the same paper Perrig et al. proposed $\mu$TESLA, a protocol for symmetric key-based authenticated broadcast with delayed key disclosure. The base station generates a random key $K_0$, computes a key chain $\{K_i\}_{i=1}^{n}$ using a one-way hash function: $K_i = h(K_{i-1})$, and sets the current key variable in all sensors to the initial value $K_n$. When the base station is ready to transmit the $k$th message, it computes its MAC using the key $K_{n-k}$, sends the message, waits until the message reaches all nodes in the network, and then sends the key $K_{n-k}$. Since each sensor knows the key $K_{n-k+1}$ from the previous transmission, it can easily verify that the key $K_{n-k}$ it receives is, indeed, $K_{n-k}$ by comparing $h(K_{n-k})$ and $K_{n-k+1}$. In an unreliable network sensors must treat each incoming message as using $K_{n-l}$ for some unknown $l \geq k$; so to validate the key they perform multiple computations of the hash function until the corresponding values match.

Most concrete implementations of security primitives in sensors rely on symmetric cryptography. For example, Karlof et al. [75] implemented a compact crypto-library called TinySec that supports the Skipjack and RC5 encryption algorithms. TinySec provides authentication and encryption operations based on these algorithms. The authors report that energy consumption (on a Mica2 mote with an 8 MHz 8-bit Atmel ATMEGA128L CPU and 4 kilobytes of RAM) increased by less than 10% compared to communications not using cryptography. Some sensor vendors implement symmetric cryptographic primitives in hardware. For instance, the EM260 network processor from Ember Corp. has a built-in AES-128 hardware engine for link encryption [76].

Although early implementations of public key cryptography in sensors were not successful [73], many researchers continue to make gradual progress in this area. Watro et al. [77] take advantage of the fact that public-key operations (encryption and signature verification) in the RSA algorithm are relatively inexpensive compared to the operations with private keys. They proposed a system where a base station sends signed messages to a
sensor network, and the network sends encrypted messages back. Using a small public exponent \((e = 3)\), they implemented an efficient public-key operations library for Mica motes written in the NesC programming language. One exponentiation, a crucial operation in the RSA algorithm, using a 512-bit key took 3.8 seconds and using a 1024-bit key, 14.5 seconds. Gura et al. improved on these results by tightly optimizing their code and writing it in assembly language [78]. They also implemented private-key operations for the RSA algorithm and multiplication of points on a curve used as an important building block in elliptic curve cryptography (ECC). On an Atmel ATmega128 at 8 MHz the authors measured 0.81 seconds for 160-bit ECC point multiplication and 0.43 seconds for an RSA-1024 operation with the exponent \(e = 65535\). They also report that the relative performance advantage of ECC point multiplication over RSA modular exponentiation increases with the decrease in processor word size and the increase in key size. Since ECC requires smaller key sizes and allows faster computation, it may be a better alternative for sensor networks than RSA.

### 2.5.2 Key Management

The SNEP protocol described in the subsection on cryptographic primitives relies on pre-distribution of pair-wise master keys to generate session encryption and authentication keys. In a large network this is not feasible as each node will need to store keys for every other node in the network. To address this problem several random key pre-distribution schemes were proposed. Eschenauer and Gligor [79] published a pre-distribution method where each sensor receives \(k\) symmetric keys out of a large pool of \(n \gg k\) keys. When deployed, the sensors attempt to discover neighbors (within the radio transmission range) they share at least one key with. In case a direct link cannot be constructed, the nodes engage in a discovery protocol to find a multi-hop key path. If such a path is found, the nodes form a symmetric pair-wise session key for all future communication needs. By carefully selecting the pool size, the number of keys per node, and the number of sensors one can construct
a fully key-connected graph with high probability. In the Eschenauer-Gligor scheme all cryptographic operations are done hop-by-hop.

Liu and Ning [4] proposed a similar scheme but instead of numbers they pre-distribute polynomials. In the setup phase they generate a set $\Phi$ of bivariate polynomials $f(x, y) = f(y, x) = \sum_{i=0}^{t} \sum_{j=0}^{t} a_{ij} x^i y^j$ over a finite field $F_q$, where $q$ is a large prime. For each sensor $s$ the system selects a subset of polynomials $\Phi_s \subseteq \Phi$ and assigns to the sensor appropriate polynomial shares. $s$’s share of $f(x, y)$ is the univariate polynomial $f(s, y)$. Once deployed, the sensors engage in a key path discovery protocol similar to Eschenauer and Gligor’s. Two sensors $u$ and $v$ can compute the pair-wise key $f(u, v) = f(v, u)$ if they know they were given shares of the same polynomial $f$. If not, they need to construct a key path from one to the other and run a key agreement protocol along the key path. Liu and Ning showed that their scheme performs very well when a relatively small number of sensors are compromised.

The last protocol, called LEAP, is due to Zhu et al. [80] who observed that a typical sensor network has a need for four types of shared keys:

- An **individual key** each sensor shares with the base station.
- A **group key** used by the base station for authenticated broadcast to the network.
- A **cluster key** a sensor shares with its neighbors; this key is used to secure local broadcasts.
- A **pair-wise** key shared by any two communicating sensors.

Before deployment the central controller assigns each sensor its individual key. Given a family of pseudo-random functions $\{f_k\}$, a master key $K^m$ known only to itself, and a sensor ID $u$, the controller computes the individual key $K^m_u = f_k^m(u)$. It also picks a secret $K_{init}$, called the initial key, and assigns to each sensor its individual master key
\( K_u = f_{K_{init}}(u) \). The keys \( K_{init}^m \), \( K_{init} \) and \( K_u \) are loaded into the sensor’s memory, and the sensor is deployed. To establish pair-wise keys with its neighbors, each sensor \( u \) broadcasts its ID and receives acknowledgements from its neighbors \( v \), authenticated with their master keys \( K_v \). To establish the key, each independently computes \( K_{uv} = f_{K_v}(u) \). Zhu et al. make an assumption that there is a lower bound on the amount of time it takes to compromise a sensor, and that the time it takes a sensor to establish pair-wise keys with (some of) its neighbors is lower than that bound. Once the bound is reached, the key \( K_{init} \) is erased.

To establish a cluster key with its neighbors \( \{v_i\} \), the node \( u \) generates a random key \( K_c^u \) and transmits it encrypted to each of the neighbors in turn: \( u \rightarrow v : (K_{uv}^c)^{K_{uv}} \). The group key can be distributed using the \( \mu \)TESLA protocol we discussed in the subsection on cryptographic primitives.

### 2.5.3 Secure Routing

Security threats against routing protocols in sensor networks are fairly well understood. Karlof and Wagner provide an exhaustive overview of this topic in [71]. They divide the attacks into several categories:

- Spoofed, altered or replayed routing information.

- Selective forwarding. An extreme form of this attack occurs when a node drops all messages it receives. This may lead to its neighbors’ deciding to look for other routes to the required destinations. Discarding only some packets and faithfully forwarding others may mask the attack and let the node operate with impunity.

- Sinkhole attacks. This attack lures traffic from a particular area to a compromised node (the ”sinkhole”) in order to mount some other type of attack, e.g., selective forwarding. The adversary tries to convince the network that the sinkhole node lies on many or all high quality routes from the area of interest.
• Sybil attacks. Here a single node presents multiple identities to its neighbors. This attack poses a threat to some fault tolerance mechanisms, such as multi-path routing, since packets believed to travel along multiple disjoint paths may, in fact, travel through the same compromised node.

• Wormholes. A wormhole attack occurs when an adversary tunnels messages received in one part of the network over a low-delay link and replays them in a different part.

• HELLO flood attacks. Many network protocols require nodes to broadcast HELLO-like messages to advertise themselves to their neighbors. The neighbors usually assume that the source of the HELLO message is within their radio transmission range. A laptop-class computer with a powerful transmitter could, therefore, mislead many sensors into believing that they are only one hop away from it. Packets sent to this computer will be lost resulting in a denial-of-service attack.

• Acknowledgement spoofing. Because of radio’s broadcast nature many nodes may overhear transmissions in the network. Some protocols require positive acknowledgments from recipients to function properly. An attacker may send such acknowledgments on behalf of other nodes in order to convince the sender that a weak link is strong or that a dead or disabled node is alive. Since packets sent along weak or dead links are lost, the adversary obtains an opportunity to mount a selective forwarding attack.

Most attacks by outsiders can be thwarted by using link-layer encryption and authentication mechanisms. Sybil and selective forwarding attacks will not be possible since the adversary cannot join the network. Acknowledgements are authenticated and cannot be spoofed. These measures do not work against attacks by insiders or compromised nodes, however.
An insider cannot be prevented from participating in the network, but it should only be able to do so using the identities of the nodes it compromised. To prevent a node from claiming multiple identities (as it would in a Sybil attack), each node could share a unique symmetric key with the (trusted) base station. To establish a pair-wise key for encryption and authentication, any two nodes could engage in a Needham-Schroeder-like protocol through the base station. (The Needham-Schroeder protocol establishes identities of the parties before creating a pair-wise key for them.) To prevent roaming insiders from doing so with all nodes in the network the base station could set a reasonable limit on the number of neighbors each node is allowed to have.

Selective forwarding attacks can be countered through the use of multiple braided (having common vertices but no common edges) routing paths. Allowing nodes to dynamically choose a packet’s next hop probabilistically from a set of possible candidates can further reduce the chances of an adversary’s gaining complete control of a data flow.

There are protocols impervious to wormhole and sinkhole attacks, for example, geographic routing. Because traffic is naturally routed towards the physical location of a base station, it is difficult to attract it elsewhere to create a sinkhole. Artificial links or tunnels created by wormhole attacks are also easy to detect since the ”neighboring” nodes will notice that distance between them far exceeds the normal range of radio transmissions.

Recently, Deng et al. introduced INSENS, an intrusion-tolerant routing infrastructure for sensor networks [81]. In their approach the base station constructs the routing table for the entire network; the sensors themselves are only responsible for supplying proximity information and propagation of requests from, and responses to, the base station. Security of their scheme is achieved through the use of $\mu$TESLA-based authenticated broadcast [74], on the one hand, and of hop-by-hop authenticated traces, on the other.
2.5.4 Secure Aggregation

In this section we provide critical analysis of approaches to secure aggregation in sensor networks proposed by other authors. We demonstrate their shortcomings in some important areas and thus provide context and rationale for our own efforts discussed in this dissertation.

Przydatek et al. [82] consider what they call a stealthy attack where an adversary wants to mislead the server into accepting false results. In their model one or more aggregators collect data from trusted sensors and propagate results to a server. All aggregators are located within one hop from the server. We believe that this approach will not scale to large networks with hundreds or thousands of nodes. To be scalable, sensor networks must support aggregation in the network.

Zhu et al. [83] consider military surveillance applications in cluster-based sensor networks. Their interleaved hop-by-hop authentication scheme detects injection of false reports if no more than \( t \) sensors in the network are compromised. False injection in a surveillance cluster is prevented because the cluster head will only forward a report if at least \( t \) sensors in the cluster endorse it. (The endorsing sensors contribute a message authentication code (MAC) based on the key they share with the base station.) To detect false report injection en route, Zhu et al.’s mechanism creates associations between nodes \( t + 1 \) hops apart on the delivery path and requires each node to append a keyed MAC using a pair-wise key established with its downstream associate. The latter node then can verify that the message contains appropriate MACs for the previous \( t \) hops and that a valid MAC was produced by the former node. Final verification is done by the base station which computes the keyed MACs using the keys shared with the endorsing sensors. Ultimately, Zhu et al.’s approach allows dropping of injected reports en route, without reaching the base station (which does the final verification anyway) thus saving energy and bandwidth in the network. Zhang et
al. extended this work to braided multipath routes [84]. In the context of secure aggregation both approaches are of limited use since they support aggregation only within a single cluster and cannot be easily extended to hierarchical aggregation over a sensor network as a whole. Similarly, Cam et al.’s solution [85] supports aggregation within a single cluster. Interestingly, one of their goals is guaranteeing confidentiality of information, even within clusters. They achieve it using pattern codes: the full range of a particular physical parameter is divided into subranges, and each subrange is mapped to a positive integer (pattern code). Only pattern codes are reported to the cluster head. Secret permutations of pattern codes act as cluster keys; they are periodically updated by the cluster head.

Hu and Evans [86] rely on cryptographic techniques to prevent attacks by outsiders. We agree with [87] that cryptography alone is not sufficient for securing sensor networks. If a sensor is compromised its key material is fully disclosed to the attacker; cryptographic techniques cannot distinguish the new, compromised "self" of the sensor from its old, honest "self". In our research we complement cryptography with data redundancy, randomized routing, and application-level constraint validation to achieve security.

Du et al. [88] use witnesses to corroborate results produced by a network aggregator. A set of honest sensors feed their measurements to the aggregator which is responsible for detecting events based on them; the aggregator sets its value to true if the event took place, and false otherwise. Simultaneously, the measurements are sent to a set of witnesses that use the same procedure to calculate the result. The computed result and its message authentication code are forwarded to the aggregator. Finally, the aggregator combines all this information, computes its own MAC, and sends the result to the base station. The base station receives enough information to verify the MACs and vote on the "correct" result.

Ye et al.’s work [89] is an improvement over [88]; the authors want multiple parties to confirm that an event took place. They take advantage of the fact that in a dense SN several sensors are likely to observe a phenomenon and be able to report on it. Their system
generates \( nm \) symmetric keys, divides them into \( n \) partitions of \( m \) keys each, and pre-assigns to each sensor a random set of \( k < m \) keys from a random partition; the server (sink) knows all keys. When an event is observed, all observing sensors (a) generate a MAC using one of their keys, (b) elect a leader to submit the event to the sink and (c) forward their MACs to the leader. The leader prunes the list to make sure that it includes at most one MAC from any given partition and that at least \( t < n \) partitions are represented, and sends a message to the sink. Any intermediate node can verify that at least \( t \) MACs are present and that no two MACs correspond to the same partition; in addition, if the node knows any keys referenced in the message, it can verify the corresponding MACs. If any of the checks fails, the message is discarded. Thus, the network probabilistically removes spurious messages before they reach the sink; if it doesn’t happen, the sink itself can do the final check and discard the message if validation fails. Since they only seek to compute a Boolean value for a section of the network, [88]’s and [89]’s approaches work well for algebraic aggregates (see Section 2.4.5) but not for other aggregate types. A more general-purpose solution to the problem is required.

Yang et al. [90] study stealthy attacks as well. In each delivery cycle they probabilistically partition the SN into groups (subtrees) of roughly equal size; each group’s aggregation results are cryptographically committed and reported to the server. The server compares the data, identifies outlier values, and requests the groups that supplied them to attest their results. Each suspect group then selectively forwards raw measurements to the server. Yang et al.’s approach is similar to ours in the use of randomization to make it difficult for the attacker to achieve consistency. We believe that their assumption that subtrees of similar size produce similar aggregation results may be unrealistic in some sensor applications.

Several authors proposed outlier detection as a technique for detecting false readings and aggregation results. Wagner [91] showed that popular aggregation functions (MIN, MAX, COUNT, etc.) are not resilient to spoofed inputs and suggested truncation (elimination
of readings outside an acceptable range) and trimming (elimination of some percentage of highest and lowest readings). Guestrin et al. [69] label measurements as outliers if they significantly deviate from predicted values. Treatment of outliers as unequivocally false and disposable may not be acceptable in many applications. As a protection mechanism it can mitigate attacks in which only a small number of sensors have been compromised; it will not be effective against a cunning adversary that stealthily compromises a significant portion of the network, remains passive while doing so, and then becomes active feeding false information from many sources simultaneously.

[69], [92] and others propose to take advantage of spatial correlation of measurements to reduce the amount of data reported to the server. In our model spatial correlation of data is expressed in the form of network constraints; each sensor periodically intercepts its neighbors’ messages in order to verify that their measurements respect these constraints.

2.5.5 Reputation Systems

One possible approach to modeling trust in distributed networks is through the use of reputation systems. Reputation can be defined as the probability of a certain set of actions on the part of an individual (device) based on his (its) prior history. One reputation system, called the beta reputation system [93], is especially suitable for sensor networks. Ganeriwal and Srivastava [87] proposed to use it to maintain distributed trust information in sensor nodes.

The beta reputation system models a peer-to-peer network with two-party transactions; for every transaction it executes a peer is expected to report its satisfaction level as a number in the interval $[-1, 1]$. The system collects these reports for each subject peer and attempts to predict its future behavior using the beta distribution. It assigns different weights to reports based on the reporters’ own reputation (this is called reputation discounting). The system also supports forgetting, i.e., it can assign lower significance to older reports and
emphasize the newer ones.

2.5.6 Physical Protection

The most obvious option in physical protection of sensors is the use of tamper-resistant hardware. Many technologies have been developed for other types of devices [Anderson01]. They may not be suitable for sensors, however, because (a) they are expensive and (b) there are many ways to compromise sensors without physically attacking them. For example, Hartung et al. [94] showed how a Mica2 mote can be compromised in less than one minute without physical tampering. They placed their mote in a programming board and used its serial interface to dump both its Flash memory and data RAM. Having disassembled the retrieved binary images, they eventually extracted the mote’s TinySec key. The attack was particularly easy because Mica2 motes provide a wired interface (for reprogramming) and on-chip debugging (OCD) capabilities. One possible countermeasure is to erase pre-distributed master keys after derivative key material has been generated. Another proposal is to implement the OCD switch in software rather than hardware thus allowing the mote’s OS to apply access control rules to the operation. Alternatively, the OCD switch could be enabled to generate an interrupt when it is being turned on; the interrupt handler then could remove all sensitive information from the mote’s memory.

One way to compromise a sensor is to modify its programming logic by installing malicious ”patches”. Many sensor platforms provide functionality for their reprogramming and/or reconfiguration on the fly. Seshadri et al. proposed a mechanism, called SWATT, to quickly verify integrity of a sensor’s memory [95]. Their approach assumes the presence of a memory analysis agent in the sensor’s memory. An external verifier knows the expected memory layout and has a copy of the agent as well. The agent is responsible for pseudo-randomly walking the sensor’s memory and accumulating a checksum. If the checksums
computed by the agent and by the external verifier are not equal, the sensor has been compromised. SWATT also assumes that a dishonest agent should take more time to traverse the sensor’s memory than the real agent (because of the need to execute comparisons and jumps). Assume that the real agent takes $t_r$ units to complete, a dishonest agent could be developed that takes $t_d + t_{comp} \leq t_r$ units where $t_d$ is the time to traverse the memory and $t_{comp}$ is the time to perform comparisons and jumps. If we tightly optimize the agent to take $t^*_r = \min t_r$ units, it will be that $t_d \geq t^*_r$ and $t_d + t_{comp} \geq t^*_r$ even more so. Thus, the external verifier could also time the internal agent and declare a compromise if it took more time than it should.

Another approach to secure software distribution in sensor networks was proposed by Dutta et al. [96]. They use authenticated broadcast to advertise the newly available binaries to the sensors. Since binaries are usually relatively large and wireless packets are small (several dozen bytes), each binary is transmitted as a sequence of packets, each packet containing a MAC computed over it and all its predecessors. The full binaries themselves are also digitally signed using the RSA algorithm with a well-known 512-bit key. Sensors can verify the signature once the full binary is received. Since public-key operations in RSA are inexpensive (compared to private-key operations with comparable data and keys), this mechanism does not unduly deplete energy resources of the upgraded sensors.

2.6 Massively Multiplayer Online Games and Virtual Environments

Massively multiplayer online games (MMOG) and virtual worlds are fast becoming a prominent feature on the Internet. The most successful MMOG thus far, World of Warcraft from Blizzard Entertainment, had over 6 million active subscribers in July 2006 [8]. New MMOGs appear all the time. According to the market analysis firm DFC Intelligence the online game
industry will grow from $3.4 billion in 2005 to $13 billion in 2011 [97]. It is supported by many auxiliary industries: application service providers, manufacturers of hardware devices and general-purpose computer equipment, broadband providers, etc. Although the social consequences of mass online gaming are still unclear, millions of people of different walks of life, ages and incomes participate in it, and the trend is accelerating. Nick Yee, a Stanford University researcher, writes: "The connotation in the word 'game' is heavy with triviality and the minimal (or negative) impact it has on 'real' life. And because these environments are marketed as games, it is easy to assume those implications of the trivial nature of games are also true of MMORPGs\(^3\); however the finding that users experience both positive and negative experiences in these environments that are comparable and sometimes more salient than their day-to-day real-life experiences reveals how misleading it is to label and think of these environments as trivial games, and it is also a denial of the rich complexity of these environments and the experiences that users are deriving from them" [98]. Whatever the social aspects of MMOGs, they pose significant technical and security challenges. They are briefly discussed in this section.

### 2.6.1 Classification of Online Games

Levy distinguishes four types of online games on the Internet [99]:

- **First-person shooter (FPS):** as the name suggests, these are games of action requiring quick action, for example, shooting of a moving target.

- **Massively multiplayer online games (MMOG):** in these games players embark on an epic adventure in a large online world with numerous interactions with the world and with other players. Many MMOGs fall into the category of role-playing games (RPG).

- **Real-time strategy games (RTS):** the player usually controls a large number of units;

\(^3\)Massively multiplayer online role-playing games
his goal is to destroy his adversary’s base.

- Sports and racing games: here the players focus mostly on their individual performance.

A more abstract taxonomy of online games was provided by Rollings and Adams [100]. They identify three game models:

- Omnipresent
- Avatar-first person
- Avatar-third person

In the omnipresent game model a player has the ability to view and control a significant portion of the game. A player’s perspective is often variable, giving him, for example, a bird’s eye view of the game world but allowing him to zoom in on specific resources and artifacts of the game. Real-time strategy, and construction and simulation games are examples of the omnipresent model. In the avatar models the player interacts with the game through a single representative character. An avatar-first person player perceives the game world through the eyes and other sensory inputs of his avatar while an avatar-third person player is given a third-person perspective. Some games allow players to switch their point of view in some situations. First-person shooter games fall into the avatar-first person category. RPGs are frequently avatar-third person games.

2.6.2 MMOG Architectures

The three basic MMOG architectures are shown in Figure 2.7. In the client/server model (Figure 2.7(a)) a single server manages the state of the game played out on multiple clients. The server is responsible for all operations: validation of events, synchronization, dead
reckoning, etc. It periodically receives snapshots from all clients, reconciles them, and
sends results and, possibly, predicted future state of the game back to them.

Figure 2.7(b) illustrates the peer-to-peer (P2P) model. Clients may be connected to all
other clients as the figure demonstrates; however, more sophisticated P2P architectures take
advantage of the locality of interest prevalent in online games and connect clients only to
those clients that affect them or are affected by them. The P2P model promises to provide
almost unlimited scalability to MMOGs.

The distributed model is shown in Figure 2.7(c). Several distributed architectures have
been proposed [99]. In the session server distributed architecture each server manages a
game world completely independently of all other worlds. A client connected to such a server
can only play with other clients connected to the same server. In the zone server distributed
architecture servers divide the game world into zones. Each server handles a single zone;
when a character wants to leave one zone and enter another, the servers synchronize with
each other. This architecture is especially suitable for large-scale games since the locality
of interest allows each server to act independently of other servers most of the time. The
last architecture, mirrored server distributed architecture, has multiple servers sharing in
the management of the game world: a client connected to one server can play with clients
connected to any other server.

2.6.3 Event Synchronization

Event synchronization in online games is a challenging and active research topic. Many
authors proposed to take advantage of traditional group communication and event ordering
mechanisms. However, several mechanisms were recently introduced to specifically address
this issue in the online gaming context. Trailing state synchronization due to Cronin et al.
[101] attempts to solve this problem for the mirrored game architecture (see Section 2.6.2).
Each server in their system maintains a fixed number of copies of the game state, each of
Figure 2.7: Basic MMOG architectures: (a) client/server; (b) peer-to-peer; (c) distributed [99]
which is kept at a different simulation time. Inconsistencies are identified by comparing the leading state (that optimistically processes game events without any additional delay) with the game states of the delayed executions (that reorder and then process the received game events). If an inconsistency is found, the server performs a rollback by copying the game state from the delayed execution view to the leading execution view; after that rolled back events are reprocessed in correct order.

The New-Event Ordering (NEO) protocol proposed by GauthierDickey et al. [102] addresses the synchronization issues for P2P games. In GauthierDickey et al.’s model each peer communicates with every other peer to exchange events. The game time is broken into rounds. In each round each player sends new events and a bit vector of events received in the previous round to the other players. Players always discard late events, i.e., events for round \( r \) must arrive at all other players in the same round. Having received bit vectors from all other participants a player can identify on-time events by voting among the bit vectors. Events not garnering majority are discarded; the on-time events are applied to the local state of the game world.

### 2.6.4 Delay Tolerance and Fairness

In a recent study of user motivation in massively multiplayer online games by Yee [98] immersion/escapism had the highest factor score for both men and women. Network delays play a very important role in users’ perception of the game and their ability to immerse. Claypool and Claypool [103] studied the impact of latency on player behavior in the omnipresent and the avatar game models. They investigated how precision required by the game and game-imposed deadlines together with network latency impacted player performance. They concluded that for a given game action (a) the higher the required precision, the greater the impact of latency on performance and (b) the tighter the deadline, the
greater the impact of latency on performance. Experimenting with real users and measuring round-trip times (RTT) in response to the users’ actions they concluded that avatar first-person games are the most sensitive to latency: when the RTT exceeds 100 ms the players’ performance significantly deteriorates. Avatar third-person games have medium sensitivity to latency: the players’ performance dropped at about 300 ms RTT, and at 700 ms the game became unplayable. Omnipresent games are not sensitive to delays. RTTs of up to 1 second did not produce any noticeable worsening in the players’ results.

Another aspect of latency is network fairness. Due to their geographic location, capabilities of their hardware, or bandwidth supplied by their Internet service providers different players may experience different round-trip times in the game. If so, some players will have a better shot at victory than others. One solution to this problem proposed by Zander et al. [104] introduces a local lag, an artificial delay for faster clients used to bring them on a par with the slower ones. Ferretti et al. [105] augment the local lag approach by dropping obsolete events. Their system measures network traversal latency of each event, creates a representative sample of recent events, and maintains a running average of the latency. The system is also configured with two thresholds: $t_L$ and $t_G$ ($t_L < t_G$). Once the average reaches $t_L$ it drops them with probability $p < 1$; if it rises above $t_G$, all contributing events are dropped. Because events are frequently superceded by other events, they can be safely dropped without processing. Ferretti et al.’s model assumes the mirrored server distributed architecture (see Section 2.6.2) which allows them to further equalize response times between different players. They propose that each mirror serve clients within 150-180 ms latency diameter.

2.6.5 Security

Online games must be engineered with special attention to security because they represent complex distributed systems, and complex systems are vulnerable to attack [106].
Client/server and mirrored distributed game architectures are susceptible to denial of service attacks by both players and outsiders. Such attacks may cause disaffection among, and flight of, the (honest) subscribers to the game and thus damage the game-hosting company’s business.

Another concern for the game-hosting company is access control to the game: only paying customers should be able to join the game. All game companies implement an authentication, authorization and accountability (AAA) system in some form. Authentication is usually based on user IDs and passwords.

Since cheating in online games is an important part of research discussed in this dissertation, we cover it separately in Section 2.7.

2.7 Cheating in Massively Multiplayer Online Games

Many solutions have been put forward to mitigate inherent untrustworthiness of game clients. Baughman et al. [107] consider specific types of cheating in MMOGs and propose protocols to protect against them. For example, they describe the look-ahead cheat where a player, instead of sending his events to other players as they occur, collects some or all of their events for a given game frame and then, based on information thus collected, generates new events that give him unfair advantage. The authors offer several protocols to combat this problem: the Lockstep Protocol where all players in the game first commit to an update and - after all of them made a commitment - actually execute it, the AS Protocol where commitment is required only between players with intersecting spheres of influence, and two other protocols that divide the game world into cells and have only players with avatars located in the same cell or in two adjacent cells exchange commitments and final updates. In our view, of these four the last protocol, called SCHP, is the most scalable and can be used as a building block for securing multiplayer online games. Even SCHP, however, has the drawback of allowing a single client to be the source of authority for some data elements,
albeit with future validation and rollback if necessary. Rolling back to a consistent state for all players is difficult. In addition, it will satisfy one type of attackers, griefers, that merely lying about their avatars’ coordinates causes the game to lose its verisimilitude to the real world for all other players because of a rollback.

Pritchard proposed for the "game" to be able to verify that all players use the same client software [108]. In addition, his clients periodically compare their game states and attempt to identify outliers, for example, by voting. Neither approach fully solves the problem as nothing prevents maliciously modified clients from reporting correct checksums or maintaining a "shadow" view of the world used only for comparison with others, but not for the actual playing of the game. Another approach suggested by Pritchard is to keep critical part of a player’s state on the server thus preventing client software from direct manipulation of the state to the player’s advantage. This is similar to our mechanism of maintaining the full state of a game world cell on the virtual server responsible for the cell.

The mutual checking proposal by Kabus et al. [109] is also akin to ours: it divides the world in a P2P game into cells, assigns a group of peers to maintain the state of each region, and uses voting to ascertain a region’s correct state. Our solution is different from that of Kabus et al. in that (a) no Cell Manager member holds full information about any object, (b) Cell Manager members are expelled if a conflict of interest is likely to arise, and (c) Cell Managers are constantly reshuffled to minimize consistent historical information that may be retained by a member.

Kabus et al. [109] also consider the use of the trusted computing (TC) model which allows external parties to verify hardware and software components running on a machine and, based on this verification, trust data produced by these components. In the context of multiplayer online games, the "game" (e.g., the server) could use TC to verify that the game client software is authentic before admitting its user into the game. Neumann et al. [110] also mention tamper-proof devices as a means of ensuring integrity of the game.
clients. We believe that TC can play an important role in securing online games. However, because TC is not in widespread use and because there are many privacy, anti-trust and usability concerns about this technology that may hamper its acceptance for home use [111], MMOGs must use other mechanisms to detect - or work around - malicious clients, at least in the foreseeable future.

Some attacks described in the gaming literature are only possible because no reliable event ordering infrastructure is available in the game (coupled with misplaced trust in the game clients). This is the case for time cheats that allow cheaters to see into the future [112]. GauthierDickey et al. proposed an event-ordering substrate for P2P online games in [102]. Their low-latency protocol, NEO, maintains total ordering of events in the game. Total ordering of all events may be excessive in many applications. Some systems support other types of ordering (e.g., causal) or even several types used for different classes of events. Linebarger and Kessler [113] provide an overview of event ordering subsystems in peer-to-peer virtual environments. Our model assumes the existence of such a subsystem in the game.

Many authors advocate cheating detection rather than prevention. Pritchard [108] suggests that honest players should be able to detect commands they receive from others that violate the rules of the game, and evict the player that submits them. Yan and Choi [14] point out that some game-hosting providers have system administrators police the game in order to identify suspicious activity, and suggest - without providing additional details, however - that this functionality can be supported by a cheating detection engine. They also propose to maintain a (protected) logging and audit trail to detect cheating. Similarly, Kabus et al. [109] propose to use logs for cheating detection. In their approach a client sends signed commands to a region controller that modifies the state of its region based on them and keeps an audit log of all processed commands. Each region controller periodically dumps its log and the state of its region to the server which can replay the commands and
validate the region’s state against what was presented by the region controller. We believe that participation of peers in audit logging exposes the detection process itself to additional attacks and increases complexity of the requisite protocols. Thus, our model relies on (a limited number of) trusted peers for detection and exclusion of cheaters.

Numerous researchers suggested that communications between various parties in online games must support sender authentication and message confidentiality. Gauthier-Dickey et al. [102] use digital signatures and encryption in their NEO protocol for secure event ordering in P2P games. Baughman et al. [107] also use message authentication and encryption in their protocols. Our model provides message authenticity and integrity through the use of digital signatures in all communications. Message and data encryption are also supported when necessary. Our model also provides a high degree of information hiding because only minimal subsets of data are disclosed between communicating parties, and binding of avatars to their owners’ client machines cannot be ascertained with confidence.

Dead reckoning is frequently used in distributed games to predict the future position of an object based on its recent history to compensate for network and processing delays [18]. Cronin et al. [112] proposed several protocols to mitigate cheating in dead-reckoned games. On the other hand, Baughman et al. [107] show that cheaters and honest players are indistinguishable under dead reckoning. Oliveira and Henderson [114] state that players will notice that dead reckoning is used if the perceptual threshold is exceeded. For these reasons our model does not directly support dead reckoning; neither does it preclude it from being used (a) remotely if constraint validation is equally applied to dead-reckoned and "real" data, or (b) locally.

Yan and Choi also propose for the game-hosting company to raise the players’ awareness of security issues in the game and to provide a channel for them to file reports on cheating they encounter [14]. This is a sensible approach; it can be implemented in addition to the technical mechanisms described in this and other papers.
Chapter 3: Publish/Subscribe Networks

3.1 Introduction

This chapter is dedicated to our research in providing service assurance in Internet-scale content-based publish/subscribe networks. Overlay (or application-level) multicast is a fundamental element of our approach. As we mentioned in Section 2.2.7 overlay multicast has the several advantages over network-level multicast: (a) it does not require network layer support and thus can be deployed universally, (b) it relieves routers from keeping state information and places heavier load on end hosts which, in many cases, only need to handle a small number of messages and (c) approaches to reliability, flow control and security can be decoupled from packet delivery and customized in an application-specific manner [31][32]. In addition, recent attempts at simulating content-based message delivery in IP multicast-based environments proved unsatisfactory (see, for example, [30]).

In this section we will show how the original problem can be reduced to a graph-theoretic one (the Minimum Steiner Tree Problem, or MSTP) and how the graph-theoretic problem can be efficiently solved for the types of graphs our reduction procedure produces. We will also present an analytical model supporting quick probabilistic estimation of the (approximate) optimal solution without actually constructing pub-sub delivery trees. Lastly, we will present results of our simulation experiments. This chapter is based on our papers [115] and [116].

The chapter is organized as follows. Section 3.2 explains how pub-sub delivery nodes are composed in our system. In Section 3.3 we describe our model and the reduction procedure we mentioned above. Our basic algorithm is presented in Section 3.4 while Section 3.5
discusses enhancement heuristics aimed at improving its performance. In Section 3.6 we present our analytical model for estimating delivery cost in a pub-sub network. Section 3.7 covers our simulation experiments. Conclusions are discussed in Section 3.8.

### 3.2 Local Service Composition

Most pub-sub systems: Siena [27], Scribe [23], Bayeux [24], etc. - use a static set of intermediate nodes allocated before the network becomes operational. Such an approach is not feasible for an Internet-scale pub-sub infrastructure. First, the system will inevitably use nodes from multiple network providers; they will require a fair amount of autonomy in how to administer their intermediate nodes. Second, the system must be tolerant of faults at intermediate nodes, and traffic will need to be routed around the failed nodes. Finally, the system should be adaptive to the changing workload, fluctuating membership and geographic distribution of its users.

In our work we assume the following model. A set of machines is available in a worldwide pool, maintained by multiple network providers. Based on the load, the pub-sub network may check out some machines from the pool and check in others. Each machine has the software, necessary for its participation in the system, installed. It also has a running configuration agent that interacts with the rest of the system to configure the machine as required. Figure 3.1 shows several participating machines with intermediate nodes running on them.

Each intermediate node is composed of three types of services that may be implemented as separate processes, threads or libraries:

- Receiving service.
- Filtering service.
- Forwarding service.
A sample intermediate node is shown in Figure 3.2. An instance of the receiving service gets messages from the previous hop on the delivery tree. An instance of the filtering service is configured with a single predicate it must evaluate against every passing message. An instance of the forwarding service sends the message to the downward hops on the delivery tree if the message satisfies the filtering service.

### 3.3 Our Model

We study PSNs rooted at a single publisher. Each subscriber advertises its interest by publishing a predicate in some query language. At the core of our approach is the observation that user interests are not completely random but follow certain patterns. For example, the popularity of content on the Internet is governed by Zipf’s law which roughly states - in this context - that a large number of users are interested in a relatively small number of items [117]. It is not clear if the same holds for real-life publish/subscribe systems, however. A
Figure 3.2: Intermediate node composed of three services: receiving, filtering, and forwarding

more realistic assumption is that users’ interests form a pattern of inclusion where items satisfying one subscription also satisfy a number of others even though the subscriptions are not identical.

Objects satisfying each predicate constitute a subset of the total object set that may be sent by the publisher. These sets induce an inclusion relation on the predicates. As an example, imagine a hypothetical company buying and selling rubber balls that wants to publish information about its transactions. Each lot, sold or bought, has balls of one color. A record of transaction has two attributes: a count, $x$ (integer), and a color, $y$ (enumerated). A positive count indicates a purchase, a negative one, a sale. As illustrated in Figure 3.3(a), the set of all lots that are either red or green is a subset of all non-yellow lots. Likewise, the set of green lots with 5 balls is a subset of all lots with 6 or fewer balls of any color and a subset of lots that are either green or red and contain any number of balls. The sets - and, equivalently, the predicates - form a partially ordered set that may be represented as a directed acyclic graph (DAG) (Figure 3.3(b)).

Every object in the universe satisfies the predicate $\text{TRUE}$. Assuming we can estimate the number of objects satisfying a query, we could assign each predicate $P$ a *filter ratio* $0 < \alpha_P \leq 1$ equal to the fraction of objects in the universe satisfied by it. Note that if
Figure 3.3: Sample subscriptions (predicates) and sets of objects satisfying them: (a) a Venn diagram showing some of the sets; (b) the equivalent partially ordered set of subscriptions.

Legend: ■ - subscriber; ○ - subscription
$P \subseteq Q$ then $\alpha_P \leq \alpha_Q$.

Our threat model can be described as follows:

- There are $m + 1$ administrative domains, one of which is trusted and the rest are not.
- Publishers and subscribers belong to the trusted domain.
- Intermediate nodes may belong to the trusted domain or to a non-trusted domain.
- At most $\lceil (m - 1)/2 \rceil$ domains are malicious.
- Malicious hosts may exhibit Byzantine behavior. Note that an non-trusted host may or may not be malicious.

We cannot rely on an non-trusted node to faithfully deliver published messages. A node can be trusted, however, if it’s mapped to a cluster of hosts, each chosen from a different non-trusted domain. Our approach is illustrated in Figure 3.4. We first start with the partially ordered set of subscriptions (predicates) and construct a predicate graph showing the publisher, all predicates, and all subscribers (Figure 3.4(a)). Then we attempt to map each predicate to a trusted machine in the spare host pool described in Section 3.2. An instance of the filtering service will be distributed to each allocated machine. This is illustrated in Figure 3.4(b). It is possible that some predicates will be mapped to a single host; it is also possible that some predicates will be split among two or more hosts. This is because the mapping process is expected to seek to minimize some cost function (for example, network utilization), and no restrictions are imposed on how it’s done. Since some of the machines in the spare host pool belong to non-trusted domains, the mapping process may allocate some predicates to non-trusted machines (e.g., predicates labeled 2, 5 and 6 in the figure). We would like to avoid it. To this end, each ”unlucky” predicate is mapped to a cluster of non-trusted machines rather than to a single machine as Figure 3.4(c) illustrates.
Figure 3.4: Progression from (a) a predicate graph, to (b) a delivery tree with all predicates that could be mapped to trusted hosts mapped to them, to (c) a tree with remaining predicates mapped to clusters of non-trusted hosts. Legend: • - root; ○ - predicate; ⊙ - predicate mapped to a cluster; △ - forwarding service; ⋯ - cluster; ▽ - receiving service; ■ - subscriber
Figure 3.5 provides an alternative view of the same process: predicates initially mapped to non-trusted hosts (Figure 3.5(a)) are ultimately mapped to trusted clusters (Figure 3.5(b)).

The secure delivery protocol operates as follows. If its predicate evaluates to true, the filtering service hands over the message to the forwarding service associated with the next hop on the delivery tree. Each receiving service collects all "versions" of the message from the previous hop, selects the true (majority) version by voting, and hands it over to its filtering service.

To simplify analysis and simulation we assume throughout this section that only non-trusted hosts are used as intermediaries in the delivery process. Our problem can be described thus: given a predicate graph and the topology of an underlying network construct a trusted overlay network that minimizes the cost of delivery of a single message. We will further transform the problem using the following procedure. First, we build transitive closures of the predicate graph and of the network topology graph. Second, we generate a list of all possible clusters and a matrix of distances between them (called the Extended Distance Matrix, or EDM). The publisher and subscribers are also included in the EDM. Let $D$ be the EDM; $C_1$ and $C_2$, clusters containing hosts $\{h_{ik}\}_{k=1}^m$ and $\{h_{il}\}_{l=1}^m$, respectively; $p$, the publisher; and $s$, a subscriber. Then $D_{C_1C_2} = \sum_{k=1}^m \sum_{l=1}^m \|h_{ik}h_{il}\|$, $D_{pC_2} = \sum_{l=1}^m \|ph_{il}\|$, $D_{C_2s} = \sum_{l=1}^m \|h_{il}s\|$, and $D_{ps} = \|ps\|$ ($\|xy\|$ is the distance between hosts $x$ and $y$). Third, we add subscribers to the predicate graph closure (PDC). Finally, we replace each internal node of the PDC with the full set of clusters and replace each edge of the PDC with an all-to-all set of edges. An edge, connecting cluster $C_1$ in predicate node $P$ to cluster $C_2$ in another predicate node, is assigned the weight $\alpha_P D_{C_1C_2}$; root-to-cluster, root-to-subscriber, and cluster-to-subscriber distances are assigned similarly. The result of this procedure is a DAG with a publisher, subscribers, and candidate mappings between predicates and clusters. The resulting structure is shown in Figure 3.6.
Figure 3.5: Mapping subscriptions to clusters of non-trusted nodes
Figure 3.6: Reducing the original problem to a graph-theoretic one

Note that a single predicate may be split among several clusters, and a single cluster may be responsible for multiple predicates. Our task is to find a minimum cost tree connecting the publisher and a mandatory set of subscriber nodes. This is the definition of the Minimum Steiner Tree Problem (MSTP) [118].

As mentioned in Section 1.1.2, we assume that non-trusted nodes may exhibit Byzantine behavior. Thus far we addressed the issue of their incorrect evaluation of associated predicates: a receiving service will accept a message from its predecessor if and only if the latter’s predicate holds for that message. A public key infrastructure (PKI) should help prevent the insertion of spurious messages or the modification of legal ones: the (known, trusted) publisher can sign every message it produces.

We use the following mechanism to deal with malicious delays. The publisher assigns a digitally signed ID to every message it publishes. Each node maintains a table of recently
received messages. When a message arrives, the node verifies its ID and checks for this ID in the table. If it’s not there (i.e., the first replica of the message has been received), the node inserts it into the table, sets a counter associated with it to 1, and sets a timer; if the message is already in the table, the node just increments the counter. Once the counter reaches \(\lceil (m+1)/2 \rceil\) the message is processed, forwarded along the delivery tree (if applicable), and removed from the table. Since honest hosts never delay it is the earliest moment this can be done. If a late replica arrives, it will be treated as a replica of a new message since all record of the message has been removed. Since only a minority may delay messages, the newly started timer will expire without achieving a quorum, and the message will be removed without further processing. (Note that the timer may be set to an arbitrarily high value; its purpose is to help maintain the table’s size bounded.)

Message reordering can be prevented if the publisher uses sequence numbers as message IDs. Let \(I\) be the last ID some receiving service received from a given forwarding service. A new message \(M\) from that forwarding service must have ID \(I_M > I\); if not, the message was sent out of order (or replayed) and must be dropped.

Another type of "contract" violation is misdirection, i.e., forwarding of messages to nodes other than the successor node. One way to deal with it is to use authentication chains proposed in [119].

### 3.4 Basic Algorithm

In Section 3.3 we reduced our original problem to an instance of the MSTP on a DAG. Unfortunately, the MSTP is NP-complete [118]. Here we discuss an approximate solution suitable for the types of DAGs constructed in Section 3.3. We will require that each predicate map to a single cluster. Before solving the actual problem let’s consider a special case when the predicate graph is, in fact, a predicate tree.
Algorithm MST-PT \((T^*_P, u, D)\)

1. \(t^*_u = \emptyset\)

2. for \(v\) in \(L_P\)

3. \(t^*_u = t^*_u \cup \{u \xrightarrow{\alpha_P D_{uv}} v\}\)

4. for \(Q\) in \(C_P\)

5. for \(v\) in \(Q\)

6. \(t^*_v = \text{Algorithm MST-PT} \((T^*_Q, v, D)\)\)

7. \(v^* = \arg\min_{v \in Q} (\alpha_P D_{uv} + w(t^*_v))\)

8. \(t^*_u = t^*_u \cup t^*_v \cup \{u \xrightarrow{\alpha_P D_{uv^*}} v^*\}\)

9. return \(t^*_u\)

Figure 3.7: Algorithm MST-PT: Solve the MSTP on a predicate tree. Legend: \(T^*_P\) - subtree on the expanded predicate tree rooted at predicate \(P\). (Each internal node is a set of cluster instances; the root node is a singleton set containing only the publisher. These set members are called \textit{member vertices}.); \(u\) - member vertex in \(P\); \(D\) - extended distance matrix; \(\alpha_P\) - predicate \(P\)'s filter ratio; \(C_P\) - set of predicate \(P\)'s children in \(T_P\); \(Q\) - a predicate; \(L_P\) - set of predicate \(P\)'s leaves (subscribers) in \(T_P\); \(v\) - a member vertex; \(u \xrightarrow{w} v\) - edge from \(u\) to \(v\) with weight \(w\); \(t^*_v\) - minimum Steiner tree rooted at member vertex \(v\) whose leaf set contains all leaves in \(T^*_P\) reachable from \(v\); \(w(t)\) - weight of tree \(t\)
Algorithm MST-PT shown in Figure 3.7 works recursively starting at the root (publisher). At each step it finds the best member vertex corresponding to the current predicate. The best member vertex is the one whose total distance to all subscribers of the current predicate (loop in lines 2-3) and to the best member vertices of the child predicates (loop in lines 4-8) is the smallest.

**Theorem 3.1.** Algorithm MST-PT runs in polynomial time.

**Proof.** Let \( n \) be the number of clusters in the system, \( N_P \), the number of predicates (excluding the root) and \( N_L \), the number of subscribers. The algorithm is called \( nN_P + 1 \) times. If invoked for some predicate \( P \), line 3 is executed as many times as there are subscribers that had advertised \( P \). Over all invocations this line will be executed \( nN_L \) times. There are \( n \) expressions under the min sign in line 7; they can be computed in time \( O(n) \). Computing the minimum will require only \( O(1) \) operations in each iteration if we keep track of the current minimum as we go along. Let \( d_P \) be the out-degree for a predicate \( P \), and \( d_r \), the out-degree for the root. The first summation is done \( n \) times for each predicate, and there are \( d_P \) members in it. Likewise, this summation for the root has \( d_r \) members but is executed only once:

\[
\left( n \sum_P d_P \right) n + d_r \leq n \left( \sum_P d_P + d_r \right) n.
\]

(3.1)

The second \( n \) is there because it takes \( O(n) \) operations to compute the sums \( \alpha_P D_{uv} + w(t_v^*) \) and find the smallest among them. The expression inside the parentheses is the sum of the out-degrees of all nodes in the predicate graph (edges connecting leafs are not counted). It is equal to the number of edges, which - since the graph is a tree - is equal to the number of internal nodes. Thus, we have

\[
n(N_P + 1)n = n^2(N_P + 1) \leq n^2(N_L + 1).
\]

(3.2)
Algorithm MST-PG \((G_P, D)\)

1. repeat until the best tree is close to the optimum
2. construct a random spanning tree on the predicate graph
3. call Algorithm MST-PT on the constructed spanning tree
4. return the best tree

Figure 3.8: Algorithm MST-PG: Solve the MSTP on a predicate graph. Legend: \(G_P\) - predicate graph, \(D\) - extended distance matrix

Summing up this expression with the expression for the leafs we get \(n^2(N_L + 1) + nN_L = O(n^2N_L)\). \textbf{QED}

Now let’s relax the requirement that the predicate graph be a tree. Algorithm MST-PG solves this problem using Algorithm MST-PT as a building block (Figure 3.8).

Each spanning tree on the predicate graph contains a number of Steiner trees when the graph is expanded. Let this number be \(m\). Let the number of spanning trees on the predicate graph be \(n\). Thus we have \(n\) buckets full of Steiner trees with each bucket containing \(m\) elements. Consider random variables \(W_T\), \(W_B\) and \(W_G\), where \(W_T\) is the weight of some Steiner tree \(T\) in one of the buckets, \(W_B\) is the minimum weight of any Steiner tree in some bucket \(B\), and \(W_G\) is the minimum weight of any of the Steiner trees. \(W_G\) is the minimum of \(mn\) random variables. Assuming \(n\) and \(m\) to be large \(W_G\) is distributed according to the Gumbel distribution with mean \(\mu_{mn}\) [120]. Local minima are also Gumbel-distributed but with parameter \(m\) (we’ll use \(G_m\) to designate its CDF). How many times should the loop in lines 1-3 be repeated before we hit upon a value less than \(\mu_{mn} + \epsilon\) for some small \(\epsilon > 0\)? From the definition of a CDF it follows that \(p = \Pr [W_B < \mu_{mn} + \epsilon] = G_m(\mu_{mn} + \epsilon)\). Lines 1-3 conduct a binomial experiment with the probability of success \(p\). The number of trials till the first success is Pascal-distributed with the mean \(1/p\) [121]. Thus, on average the loop will need to execute \(1/p = 1/G_m(\mu_{mn} + \epsilon)\) times.
Algorithm DC \((G_P,G_T,\Delta)\)

1 if \(|\Delta| = 1\)

2 \(T = \text{Algorithm MST-PG} \((G_P,D(C(\Delta)))\)\)

3 else

4 divide \(\Delta\) at random into equal subsets \(\Delta_1\) and \(\Delta_2\)

5 \((,C_1) = \text{Algorithm DC} \((G_P,G_T,\Delta_1)\)\)

6 \((,C_2) = \text{Algorithm DC} \((G_P,G_T,\Delta_2)\)\)

7 \(T = \text{Algorithm MST-PG} \((G_P,D(C_1 \times C_2))\)\)

8 return \((T,C_T)\)

Figure 3.9: Algorithm DC: the Divide-and-Conquer (DC) heuristic. Legend: \(G_P\) - predicate graph; \(G_T\) - network topology graph; \(\Delta\) - set of allowed non-trusted domains; \(\Delta_i\) - set of allowed domains for the \(i\)th sub-problem; \(C_i\) - set of clusters used in the solution of the \(i\)th sub-problem; \(D(X)\) - the extended distance matrix over a cluster set \(X\); \(C(\Delta)\) - complete cluster set over the domain set \(\Delta\)

### 3.5 Enhancement Heuristics

It can be shown that the total number \(N\) of candidate clusters grows exponentially with the total number of non-trusted hosts. Hence, there is a need to find a way to work with smaller sets of clusters input into the model in Section 3.3. To this end we introduce two heuristics. The first one (Random Selection, or RS) is based on the idea that a limited subset of candidate clusters may provide a good approximation for the solution of the overall problem. We select \(\log N\) clusters at random. The second heuristic (Divide-and-Conquer, or DC) solves the \(m\)-domain problem by dividing it into two \(m/2\)-domain problems and solving them recursively. Clusters involved in the solutions for each sub-problem are combined to form candidate clusters used to solve the full-size problem.

Now we will show analytically that the heuristics allow us to reduce the original problem
to a manageable-size instance of the MSTP. Let \( n_T \) be the number of terminals in the MSTP, \( n_N \) the number of all other nodes, and \( A(n_N, n_T) \) the complexity of an MSTP algorithm used as a building block in the simulations. For simplicity we'll assume that all domains contain an equal number of hosts (this corresponds to the worst case for a constant total number of hosts). The following theorems provide time complexity formulas for the basic algorithm and the DC and RS heuristics.

**Theorem 3.2.** Let \( c \) be the total number of subscribers, \( p \), the total number of predicates including the root, \( m \), the total number of non-trusted domains, \( n \), the total number of non-trusted hosts. Then the problem of creating a secure overlay structure in our publish/subscribe system can be solved in time \( A((p - 1)(n/m)^m + 1, c) \).

**Proof.** Follows from the fact that we have \( p - 1 \) internal predicates each containing a full set of candidate clusters in the expanded predicate graph plus the root predicate. There are \( (n/m)^m \) clusters altogether which yields the value of the first parameter. The second parameter’s value is \( c \) because the set of subscribers is the set of terminals in the Steiner tree. QED

**Theorem 3.3.** The Random Selection (RS) heuristic solves the original problem in time \( A((p - 1)m \log(n/m) + 1, c) \).

**Proof.** Trivially follows from the fact that we randomly select \( \log M \) clusters out of the total set of \( M \). QED

To prove the time complexity formula for the Divide-and-Conquer heuristic we will need an intermediate result given by the lemma below.

**Lemma 3.1.** The total number of clusters used in a minimum Steiner tree on the expanded predicate graph has the upper bound \( c^2/4 + c \) where \( c \) is the number of subscribers.

**Proof.** The total number of used clusters cannot exceed the number of internal nodes in the minimum Steiner tree. This is because each internal node contains a cluster, and some clusters may be used more than once. Let us consider a Steiner tree with the largest number
of edges. In this tree any two terminals $u$ and $v$ are connected to the root by two paths $\pi_u$ and $\pi_v$ such that $\pi_u \cup \pi_v = \emptyset$. If a terminal (subscriber) is connected to (published) a predicate located at depth $i$ in the predicate graph, there are $i$ internal nodes on the path between the root and that terminal. Thus, if the expanded predicate graph has $c^{(i)}$ leafs at depth $i$ and is $k$-deep the worst-case Steiner tree must have

$$\sum_{i=0}^{k} ic^{(i)}$$

(3.3)

internal nodes, and

$$\sum_{i=0}^{k} c^{(i)} = c$$

(3.4)

Now let’s consider two sums (we omit the 0th element since it’s always 0): $1 \cdot c^{(1)} + \ldots + g \cdot c^{(g)} + \ldots + h \cdot c^{(h)} + \ldots + k \cdot c^{(k)}$ and $1 \cdot c^{(1)} + \ldots + g \cdot (c^{(g)} - 1) + \ldots + h \cdot (c^{(h)} + 1) + \ldots + k \cdot c^{(k)}$. Subtracting the first expression from the second, we get $h - g$ which is positive as $h > g$. Thus, expression (3.3) reaches its maximum when the $k$th element is the largest possible and all the other elements are the smallest. The smallest legal value is 1; this leaves $c^{(k)} = c - k + 1$. Hence, the maximal value of (3.3) is $f(k) = (k - 1) + k \cdot (c - k + 1)$. This function reaches its maximum when $k = (c + 2)/2$. Indeed, since $f'(k) = 1 + 1 \cdot (c - k + 1) + k \cdot (-1) = c + 2 - 2k$, we get the result by solving the equation $f'(k) = 0$ and noticing that $f''(k) = -2 < 0$. Thus, $\max_k f(k) = f((c + 2)/2) = c^2/4 + c$. QED

**Theorem 3.4.** Let $\bar{A}(t) = A((p - 1)t + 1, c)$. The Divide-and-Conquer heuristic solves the original problem in time $O(m\bar{A}(n/m))$.

**Proof.** We will prove this formula for the case $m = 2^r$ for some $r \in N$. When solving the problem for $m$ non-trusted domains Algorithm DC solves two sub-problems for $m/2$ domains each and the full problem for $m$ domains restricting the candidate cluster set only to
the Cartesian product of those clusters actually used in the solutions for the sub-problems. If \( D \) is the upper bound on the number of clusters used in the sub-problem (such a bound exists due to Lemma 3.1) and \( T(k) \) is the time complexity of the heuristic on a \( k \)-domain problem, we get

\[
T(m) \leq 2T(m/2) + \bar{A}(D^2),
\]

\[
(3.5)
\]

\[
T(1) = \bar{A}(n/m).
\]

(3.6)

The inequality (3.5) can be rewritten as

\[
T(m) \leq 2^rT(1) + \bar{A}(D^2) \sum_{i=1}^{r-1} 2^i < 2^rT(1) + 2^r\bar{A}(D^2).
\]

(3.7)

In other words, \( T(m) < m\bar{A}(n/m) + m\bar{A}(D^2) \). Since \( D \) is independent of the number of non-trusted domains, it follows that \( T(m) = O(m\bar{A}(n/m)) \). QED

### 3.6 Cost Estimate

In some situations it may be too expensive to construct the exact predicate graph. Consider the case of "flash crowds", i.e., rapidly forming and changeable PS groups, in a resource-constrained environment. Our goal should be to admit as many subscribers as possible without violating the constraint. To solve this problem we could use one of the well-known search heuristics such as hill climbing. Each point in the search space will represent a subset of subscribers, and at each step the search heuristic will need to calculate the cost of the delivery tree for that subset. A heuristic can use the estimation method in this section to quickly reject those subsets that don’t qualify.

Throughout this section we assume that the original predicate graph is a tree. Assume there are \( n+1 \) predicates (the 0th predicate is the root), the \( i \)th predicate has the filter ratio \( \alpha_i \), has \( c_i \) leafs directly attached to it, and can reach \( C_i \) other leafs (through its children), and
$W$ is a random variable representing distances in the EDM $D$. $W$ is approximately normally distributed since it’s a sum of a large number of independent random distances in the original network topology graph. Without loss of generality we’ll assume that $W \sim N(0, 1)$. Let $K_i$ be the number of edges emanating from the $i$th predicate in a Steiner tree ($0 \leq K_i \leq C_i$).

The weight of a Steiner tree with these characteristics is

$$W_T(D) = \sum_{i=0}^{n} \alpha_i \left( \sum_{j=1}^{K_i} W + \sum_{j=1}^{c_i} W \right).$$

(3.8)

The minimum Steiner tree has the smallest $W_T(D)$ among all the Steiner trees on the graph and, moreover, among all the Steiner sub-graphs of the graph. The second sum in (3.8) is a sum of $c_i \ N(0, 1)$-distributed variables. Thus, it is $N(0, c_i^2)$ [121]. To calculate the first sum, let’s construct a random Steiner sub-graph by selecting a uniformly random path from the root to each terminal. Then the following lemma holds.

**Lemma 3.2.** $K_i$ have the probability density function

$$\Pr[K_i = k] = \binom{C_i}{k} \left( \frac{1}{2} \right)^{C_i}$$

(3.9)

**Proof.** Consider a terminal (subscriber) $s$ that advertised some predicate $L$ located at depth $d$ in the predicate graph and an intermediate predicate $I$ on the path between the root $R$ and $L$. Since all nodes on any path between $R$ and $L$ are directly connected to one another, each such path can be represented as a bit sequence of length $d-1$ with the $j$th bit set to 1 if the predicate at depth $j$ is included in the path, and to 0 otherwise. Thus, there are $2^{d-1}$ paths between $R$ and $L$, and half of them include $I$. Therefore, the probability that a cluster in the member set $I$ on the expanded predicate graph is included in some path is $1/2$. To construct a random Steiner sub-graph we need to repeat this step $c$ times (where $c$ is the total number of subscribers). Only $C_i$ repetitions, however, affect the $i$th
predicate because only this many terminals are reachable from it. From this it follows that
the total number \( N_i \) of paths passing through the \( i \)th predicate is binomially distributed
with the base probability \( 1/2 \) and the number of experiments equal to \( C_i \); hence \([121]\)

\[
\Pr[N_i = k] = \binom{C_i}{k} \left(\frac{1}{2}\right)^{C_i}.
\] (3.10)

Every time one of these paths passes through a member set on the extended predicate graph
we need to choose a cluster in that member set to ”represent” it on this path. Let \( n \) be the
number of clusters in the set. This procedure independently selects \( C_i \) times one cluster
out of \( n \) with replacement, and if a cluster is selected there is exactly one edge on the path
emanating from the cluster. Let \( p_s(t) \) be the probability of selecting exactly \( t \) clusters for
\( s \) paths. Naturally, \( p_1(1) = 1 \), and \( p_j(j) = p_{j-1}(j-1)(n-j+1)/n \) for \( j > 1 \). It follows
that \( p_j(j) = \prod_{l=1}^{j-1} (1 - l/n) \) and \( p = p_{N_i}(N_i) = \prod_{l=1}^{N_i-1} (1 - l/n) \geq 1 - \sum_{l=1}^{N_i-1} l/n \) (due to
the Weierstrass inequality). Thus, \( p \geq 1 - N_i(N_i - 1)/n > 1 - C_i^2/n \). Assuming \( C_i^2 \) is an
order of magnitude smaller than \( n \) we get \( p \approx 1 \). From this it follows that the probability of
selecting \( k \) clusters (and, therefore, \( k \) outgoing edges) is equal to the probability of \( k \) paths
passing through the predicate, and it is given by (3.9). QED

If \( W \sim N(0, 1) \) and \( K_i \) is a random variable distributed according to (3.9), then the
sum of \( K_i \) instances of \( W \) is a random variable with mean 0 and standard deviation \( \sqrt{C_i}/2 \).
Indeed, a random sum of \( L \) independent identically-distributed random variables \( X \) with
expectation \( \mu_X \) and standard deviation \( \sigma_X \) has the mean \( \mu_X E[L] \) and the standard deviation
\( \sqrt{\sigma_X^2 E[L] + \mu_X^2 var[X]} \) [121]. Since in our case \( \mu_X = 0 \) and \( \sigma_X = 1 \) we get, respectively,
0 and \( \sqrt{E[L]} \). As \( L = K_i \) is binomially distributed, \( E[L] = C_i/2 \). The CDF of the sum’s
distribution has a somewhat shorter tail than the normal distribution but for simplicity we
will assume henceforth that it is normal.

85
Lemma 3.3. $W_T(D) \sim N(0, \sum_{i=1}^{n} \alpha_i^2 (c_i^2 + C_i/2))$.

Proof. Follows from the discussion above and Lemma 3.1. QED

Now let’s consider another random variable: $W^*_T = \min W_T$. Since we know $W_T$’s distribution, we can determine $W^*_T$’s distribution as well. Let $G$ be the total number of all Steiner trees on the extended predicate graph. Then,

$$\Pr[W^*_T \leq w] = 1 - \Pr[W_T^{(1)} > w, \ldots, W_T^{(G)} > w],$$

where $W_T^{(i)}$ is the $i$th instance of the random variable $W_T$. Since all $W_T$ are mutually independent, we can rewrite (3.11) as

$$\Pr[W^*_T \leq w] = 1 - \Pr[W_T > w]^G = 1 - (1 - \Pr[W_T \leq w])^G.$$

Let $G_k$ be the total number of Steiner trees with $k$ edges and $P_k$ the probability that a Steiner tree has $k$ edges. By definition, $G = \sum_{k=0}^{\infty} G_k$, $\sum_{k=0}^{\infty} P_k = 1$, and $G_k = P_k G$. Therefore,

$$G = G_k/P_k.$$

Lemma 3.4. The probability of a Steiner tree on the extended predicate graph having $k$ edges is defined by the formula

$$P_k = \binom{C}{k-c} \left(\frac{1}{2}\right)^C,$$

where $C = \sum_{i=0}^{n} C_i$ and $c = \sum_{i=0}^{n} c_i$.

Proof. The random variable $K'$ equal to the number of edges in a Steiner tree can be defined as $K' = \sum_{i=0}^{n} K'_i$ where $K'_i$ is the total number of edges emanating from the $k$th predicate. By definition, $K'_i = K_i + c_i$. Therefore, $K' = c + \sum_{i=1}^{n} K_i$. Equation (3.9) gave the PDF for $K_i$. They are binomially distributed independent random variables with the
base probability 1/2. Therefore, the random variable $K = K' - c = \sum_{i=0}^{n} K_i$ has the PDF

$$\Pr[K = k] = \binom{C}{k} \left(\frac{1}{2}\right)^C.$$

(3.15)

In other words,

$$\Pr[K' = k + c] = \binom{C}{k} \left(\frac{1}{2}\right)^C,$$

(3.16)

or, equivalently,

$$\Pr[K' = k] = \binom{C}{k - c} \left(\frac{1}{2}\right)^C.$$

(3.17)

QED

The smallest possible number of nodes in a Steiner tree is equal to the number of leafs plus the number of predicates in the original predicate graph. This is because the root and the leafs must be there by the definition of a Steiner tree, and there must be at least one cluster allocated per each internal predicate since each subscriber must be ”attached” to the predicate it advertised. The number of edges in this tree is $n + c - 1$, i.e., one less than the number of nodes. If we use only one cluster per predicate node, a Steiner tree in the extended predicate graph must be a spanning tree in the predicate graph. If $m$ is the number of available clusters per predicate node, we can choose the single cluster for the Steiner tree in $m$ ways. Since this is done independently for each internal predicate node there are $m^{n-1}$ Steiner trees corresponding to a single spanning tree in the predicate graph. Let $d_1, d_2, \ldots, d_n$ be the in-degrees of the internal nodes in the predicate graph. Since this graph is directed acyclic we can build a spanning tree by selecting a uniformly random in-edge for each of the nodes. Since edges for different nodes are selected independently, there are $\prod_{i=1}^{n} d_i$ spanning trees in the predicate graph. Thus, the total number of 1-cluster-per-predicate-node Steiner trees in the extended predicate graph is $G_{n+c-1} = m^{n-1} \prod_{i=1}^{n} d_i$. 87
From this and (3.13) we get \( G = G_{n+c-1}/P_{n+c-1} = m^{n-1} \prod_{i=1}^{n} d_i/P_{n+c-1} \). Combining all the ingredients, we can compute the value of (3.8).

### 3.7 Evaluation

For our simulation experiments we selected the MSTP algorithm due to Charikar et al. [118]. It performs reasonably well: on a directed acyclic graph with \( n \) vertices and \( k \) terminals its running time is \( O(n^{3\log k}) \). It also provides a good \( O(\log^2 k) \) approximation of the optimal solution, and is relatively easy to implement.

To create topologies we used the GT-ITM package [122] in a hierarchical configuration with three levels of hierarchy: domains, subdomains, and hosts. GT-ITM provides several algorithms for constructing a network topology; we chose the Doar-Leslie algorithm since it realistically reflects the power laws governing the Internet. Input parameters were selected to produce distances (delays) consistent with those reported for the Internet, for example, by the Internet Health Report [123]. To create predicate graphs we used the BNGenerator tool [124]. Each experiment consisted of running the two heuristics from Section 3.5 and the Benchmark (BM) Algorithm, and recording the cost of the generated delivery tree. BM’s purpose was to construct a network-agnostic delivery tree taking into account only the predicate graph. More specifically, the BM Algorithm first solves the problem using an artificial underlying network with the following parameters:

- The trusted domain remains the same.

- The number of non-trusted domains remains the same, but only one host is left in each domain. (It means that only one cluster may be created in this system.)

- The hosts form a complete graph with edge costs equal to 1.

The tree produced by this step is transformed by replacing each vertex with a random cluster from the real underlying network. Finally, each edge is assigned new cost from the
Figure 3.10: Cost vs. number of domains. Number of replications: 30, total number of hosts: 1024

extended distance matrix (EDM) using the procedure described in Section 3.3. This allowed us to study absolute as well as relative usefulness of the proposed heuristics. Results are presented in terms of cost ratios between the heuristic in question and the BM algorithm.

In the first experiment we studied the cost of the minimum Steiner tree as a function of the number of non-trusted domains (Figure 3.10). The figure seems to indicate that the Divide-and-Conquer heuristic performs better than both the Random Selection heuristic and the Benchmark Algorithm. Paired-\(t\) confidence intervals (see Figure 3.11) confirmed that DC outperforms BM for all values of the input parameter \(m\), and RS performs better than BM when \(m < 16\). Similarly, the DC algorithm performs better than the RS algorithm for \(m < 16\).

The second group of experiments measured the cost as a function of the filter ratio decrease factor. By construction of the predicate graph described in Section 3.3 filter ratios described in Section 3.3 filter ratios must decrease along all paths from the root (publisher) to a leaf (subscriber). We believe that filter ratios will significantly vary from application to application. In this experiment
Figure 3.11: Paired-t confidence intervals for the cost vs. number of domains graph in Figure 3.10: (a) for the Divide-and-Conquer heuristic and Benchmark Algorithm; (b) for the Random Selection and Divide-and-Conquer heuristics

Filter ratios were assigned as follows. The root, as in all other cases, had the ratio of 1. All predicates were topologically sorted, and the ratio was assigned to each predicate in turn starting at the second element in the list (the first element was the root). Let $j$ be the name of the predicate we need to set at some point in the process, $\{j_k\}_{k=1}^n$ its predecessors in the graph, and $0 < \phi < 1$ the decrease factor. Then $\alpha_j = \phi \min_k \alpha_{j_k}$. The predicate graph post-processor, responsible for assigning these values, was given a range $0 < \phi_{\text{min}} \leq \phi_{\text{max}} < 1$, and the actual value of the decrease factor $\phi$ was uniformly drawn from it.

Figure 3.12 compares our heuristics with the BM Algorithm. Apparently, the Divide-and-Conquer heuristic outperforms the Random Selection heuristic at all values of the filter ratio decrease factor when there are 8 non-trusted domains in the network. This is consistent with Figure 3.10 and suggests that relative performance of the heuristics is independent of the distribution of filter ratio values. Having constructed confidence intervals (Figure 3.13) we confirmed that DC was better than RS, which in turn was better than BM for all values of the input parameter.

In the third experiment we studied how the cost of the minimum Steiner tree depends on the density of the underlying network. In this experiment we created edges between hosts by using the random method [122]: for each pair of nodes an edge is inserted with some
probability $p$. Intuitively as connectivity in a network increases (with an unchanged weight distribution) the cost of the minimum Steiner tree should not increase since we are getting more candidate Steiner trees to choose from. Our experiments confirm this. Figure 3.14 shows that the DC heuristic outperforms the RS heuristic for all but very sparse networks. Both DC and RS perform better than BM for medium- to high-density networks (with the edge probability $p > 0.4$). In sparse networks - when $p \leq 0.4$ - we could not determine the winner since the corresponding confidence intervals (Figure 3.15) included zero.

### 3.8 Conclusions

In this chapter we used the dynamic service composition approach to construct overlay multicast trees in publish/subscribe networks. Although our primary goal was to ensure service integrity in the presence of non-trusted and, possibly, malicious nodes, it can be used to build delivery trees even in secure environments. By taking advantage of the implicative
Figure 3.13: Paired-\(t\) confidence intervals for the cost vs. filter ratio decrease graph in Figure 3.12: (a) for the Divide-and-Conquer heuristic and Benchmark Algorithm; (b) for the Random Selection and Divide-and-Conquer heuristics
relationship between predicates in subscriptions our method constructs a multicast tree
simultaneously laid over the network topology graph and the predicate graph. It eliminates
the need for complex matching algorithms and data structures since only a single predicate
is assigned to each filtering/routing service.

The most important contribution in this chapter is in securing the "natural" flow of
published messages even when some intermediate nodes maliciously violate their contract
with the PS system. By introducing redundancy and clustering we were able to provide
service integrity guarantees in a network with multiple security domains up to half of which
may be malicious. Our model is also applicable to the case when nodes and links fail for
other reasons.

We reduced the original problem to the Minimum Steiner Tree Problem (MSTP) and
showed that a randomized polynomial algorithm can construct a good approximation of the
optimal delivery tree for the types of graphs we’re interested in. We developed a technique to

Figure 3.14: Cost vs. density of the underlying network. Number of replications: 20,
number of non-trusted domains: 8, total number of hosts: 1024
Figure 3.15: Paired-\(t\) confidence intervals for the cost vs. density of the underlying network graph in Figure 3.14: (a) for the Divide-and-Conquer heuristic and Benchmark Algorithm; (b) for the Random Selection and Divide-and-Conquer heuristics
quickly estimate the cost of an optimal delivery tree; it can be used in resource-constrained networks to find the maximal subset of subscribers that can be supported without violating the network’s limitations. We also proposed two heuristics aimed at reducing the size of the input into the MSTP algorithm. Results of our simulation experiments show that the Divide-and-Conquer heuristic outperforms both the Random Selection algorithm and the network topology-agnostic Benchmark Algorithm. The Divide-and-Conquer heuristic has time complexity $O(m\bar{A}(n/m))$; this is a significant improvement over the exhaustive cluster enumeration approach, which runs in time $O(\bar{A}((n/m)^m))$ ($m$ is the number of non-trusted domains, $n$, the total number of non-trusted hosts, and $\bar{A}$, the time complexity of the algorithm on a 1-domain problem).
Chapter 4: Aggregating Sensor Networks

4.1 Introduction

In this chapter we discuss our approach to securing aggregating sensor networks against stealthy attacks. As we explained in Section 1.1.2 a stealthy attacker wants to (a) mislead the server into believing that the sensor network produced results advantageous to the attacker and (b) remain undetected. This is an application-layer attack. We solve the problem by introducing the concept of “neighborhood watch”, generating randomized delivery trees, and defining protocols and mechanisms for detection and removal of compromised sensors. This chapter is based on our paper [125]. We call our solution CoDeX, because it provides a framework for data Collection, compromise Detection, and eXclusion of compromised sensors from aggregating sensor networks.

The chapter is organized as follows. In Section 4.2 we describe components of our trustworthy sensor network. Section 4.3 discusses the Data Collection, Sensor Exclusion and Network Abandonment Protocols necessary for secure aggregation in, and maintenance of, the network. The server and its operations are described in Section 4.4. Setup and supporting services required by a secure sensor network are discussed in Section 4.5. Section 4.6 provides an evaluation of our approach based on extensive simulations. Finally, Section 4.7 discusses our conclusions.

4.2 Trustworthy Sensor Network

We use the basic model shown in Figure 4.1. As we discussed in Chapter 2 the system consists of three parts: (a) a server, (b) a network of sensors, and (c) a set of base stations
communicating with the sensor network via radio and with the server via a wired network.

The whole sensor network is divided into logical groups \( \{z_i\}_{i=1}^{m} \) called zones. Each sensor belongs to one and only one zone. The SN is responsible for calculating some vector function \( f : z \times t \rightarrow \Omega \) where \( t \) is time and \( \Omega \) is an application-specific range (e.g., temperature, probability, radiation level, etc.) Sensors may be assigned to zones statically or dynamically.

Static assignment based, for example, on the sensors’ locations is done once, at the time the SN is set up. Dynamic assignment changes the sensors’ affiliation when the SN is running; it may be explicit, achieved by assigning new zone IDs to individual sensors, or implicit, induced by a GROUP-BY-like clause in the queries sent by the server.

The setting can be described as an adversarial game between the server that wants the sensor network to (a) report a good approximation of the true values and (b) detect results outside its tolerance interval, and the attacker that wants to (a) mislead the server by manipulating the sensor network into reporting values \( f' \neq f \) and (b) remain undetected.

We make the following assumptions about the sensor network (they are reproduced here from Section 1.1.2 for convenience):

- Time is divided into epochs. In every epoch the SN calculates a vector function, with one dimension per sensor zone. The function may be multi-valued (e.g., AVG in Table 4.1)

- The aggregation function \( \lambda \) is such that \( \lambda(S) = \mu(\lambda(S_1), \ldots, \lambda(S_n)) \) for any partition
\( \{S_i\}_{i=1}^n \) of a sensor set \( S \). Table 4.1 illustrates validity of this assumption for some frequently-used aggregation functions.

- Only part of the sensor network has been compromised. The attacker can, however, continue compromising new sensors even as it manipulates the network.

- The attacker has complete control over a compromised sensor and may manipulate measurements, aggregation results, and/or security protocols.

- The base stations and the server are secure.

- The SN’s setup phase is secure, i.e., an attack may be started only after setup has been completed.

- There exists a constraint \( \phi(m_i(t), m_j(t)) \) that must hold for measurements \( m_i(t) \) and \( m_j(t) \) made at time \( t \) by any spatially close sensors \( i \) and \( j \). As an example, Ishida [126] describes a set of temperature-measuring sensors in a condenser-type heat exchanger. Let \( T_{is}^i \) be the temperature at the inlet of the exchanger’s shell side, \( T_{os}^i \) be the temperature at the outlet of the shell side, \( T_{it}^i \) be the temperature at the inlet of the tube side, and \( T_{ot}^i \) be the temperature at the outlet of the tube side. If all sensors function properly, the following constraints must hold: \( T_{is}^i > T_{os}^i, T_{it}^i > T_{ot}^i, T_{is}^i > T_{it}^i, \) and \( T_{os}^i > T_{ot}^i \). If an agent running in one of the sensors observes that a particular constraint does not hold, it can conclude that the other sensor is faulty. On the other hand, many sensor networks take advantage of spatial correlation between measurements in small neighborhoods to extrapolate reported values for locations with no sensors and, thus, reduce network density as well as bandwidth and energy requirements [127]. ”Close” here means within the radio transmission range of the sensors. Figure 4.2 shows an example of a temperature-sensing sensor network with a linear constraint.
Figure 4.2: A temperature-sensing sensor network with a linear constraint. Legend: ° - a sensor’s sensing range; ⋯ - a sensor’s transmission range; ΔT - temperature difference; k - a constant

Table 4.1: Computing an aggregation result over a set as a function of aggregation results over its constituent subsets

<table>
<thead>
<tr>
<th>Function</th>
<th>Computation</th>
</tr>
</thead>
</table>
| AVG      | \[
\lambda.value(S) = \frac{\sum_{\lambda.count(S_i)} \lambda.value(S_i)}{\sum_{\lambda.count(S_i)}} \\
\lambda.count(S) = \sum_{\lambda.count(S_i)}
\]  |
| COUNT, SUM | \[
\lambda(S) = \sum \lambda(S_i)
\] |
| MAX, MIN  | \[
\lambda(S) = \lambda(\lambda(S_1),\ldots,\lambda(S_n))
\] |

- There exists a constraint \(\varphi(a_i(t))\) that must hold for aggregation results produced by any sensor \(i\) at time \(t\). The constraint may take the form \(\varphi(a_i(t), a_j(t))\) where \(i\) and \(j\) are two spatially close sensors. For example, a monotonic aggregate, by its definition, requires that either \(\lambda(\text{parent}) \geq \lambda(\text{child})\) for all nodes or \(\lambda(\text{parent}) \leq \lambda(\text{child})\) for all nodes [68].

**Trustworthy measurements.** A secure aggregation protocol won’t provide much value if the raw measurements it relies on are faulty. We take advantage of the fact that the constraint \(\phi(m_i(t), m_j(t))\) holds for any two sensors \(i\) and \(j\) located in each other’s vicinity. Since the transmission range of a wireless sensor is small (60-300 feet), we can assume that

99
the constraint holds for any two neighbors. When a sensor transmits (broadcasts) its measurement to an aggregator all its neighbors can hear the transmission and compare it with their measurements. If the constraint is not satisfied, the neighbors report the misbehaving sensor in their next transmission. Note that to support constraint validation aggregators do not incorporate their own measurements in the aggregation results but transmit them separately (in the same message).

**Trustworthy aggregation.** We assume that the adversary possesses limited resources, and therefore, will be able to attack only a small number of sensors at a time but will continue the attack indefinitely. To secure the aggregation process we employ a two-pronged solution:

- Use multiple delivery trees.
- Aggregate the same measurements several times each time using a different delivery tree.

Assuming for the moment that (a) sensors never fail and (b) the sensor network provides a reliable communication service, an uninfected network will produce the same aggregation results for the same set of raw measurements, regardless of the delivery/aggregation tree. Not so in a partially compromised SN: a compromised node aggregating over different sub-trees may not be able to adjust its results to guarantee consistent outcomes produced by the network as a whole.

Our sensor network constructs a random delivery tree in each epoch. Each sensor is aware of its immediate neighbors and, among them, of those closer to the server in some metric. When it needs to transmit a message it picks a random neighbor from this set and sends the message to it. This process repeats at each sensor, closer and closer to the server, until the message reaches one of the base stations. We call the union of all possible delivery trees the *delivery graph*. It is constructed using the algorithm \texttt{Construct-Delivery-Graph}. 

\texttt{Construct-Delivery-Graph

100}
Construct-Delivery-Graph \((G_T)\)

1. \(V_D = V_T\)
2. \(E_D = \emptyset\)
3. \(r = \text{Rank-Nodes} \ (G_T)\)
4. for \(e \in E_T\)
5. if \(r[e.nodex_1] < r[e.nodex_2]\)
6. \(E_D = E_D \cup \{(r[e.nodex_2], r[e.nodex_1])\}\)
7. else
8. \(E_D = E_D \cup \{(r[e.nodex_1], r[e.nodex_2])\}\)
9. return \(G_D\)

Figure 4.3: Construction of the delivery graph. Input: topology graph \(G_T(V_T, E_T)\); output: delivery graph \(G_D(V_D, E_D)\); local variables: associative array of ranks \(r : V_T \rightarrow Z^*\), edge iterator \(e\) of type edge with two fields representing an edge’s incident nodes. Algorithm \text{Rank-Nodes} (not shown) assigns ranks to nodes (Figure 4.3).

In addition, each leaf sensor transmits its measurements for \(s\) recent epochs including the current one. Each intermediate sensor performs aggregation separately for each epoch and includes results for all \(s\) epochs in the output message.

### 4.3 Protocols and Messages

The lifecycle of a sensor network consists of three phases: (a) discovery and setup, (b) steady state, and (c) abandonment. We assume that the setup phase is secure, i.e., it will take some time for the attacker to discover the SN and start the attack. A steady-state SN sends to the server its aggregation results. Even a compromised network may provide useful information to the server. Therefore, in many cases the server will not immediately abandon the SN but will continue using it separating "signal" from "noise" produced by
the attacker. As the attacker compromises more sensors, the SN becomes less reliable. At some point the server will need to abandon the network thereby starting the last phase.

Our model uses three protocols to provide data and achieve resilience against attack:

- The Data Collection Protocol (DCP), used to propagate measurements and aggregation results through base stations to the server.
- The Sensor Exclusion Protocol (SEP), used by the server to notify the SN of untrustworthy, failed and isolated sensors.
- The Network Abandonment Protocol (NAP), used by the server to request the SN to cease operation.

**Data Collection Protocol.** DCP messages flow from sensors farther from the server to those closer to it until they reach a base station and then the server itself. The SN generates these messages in every epoch except maintenance and key disclosure epochs (see the SEP subsection). In each data epoch each sensor randomly selects an aggregator from the list of its lower-ranked neighbors and sends a DCP message to it. This enables the sensor network to construct a random delivery/aggregation tree, an important element of protection against stealthy attacks (see Section 4.2). The format of a DCP message is shown in Figure 4.4(a). Each message contains information on $s$ most recent epochs. An epoch’s data consists of three sections: aggregation, measurement, and complaint.

The aggregation section (Figure 4.4(c)) is a set of entries indexed by zone. Each entry contains a partial aggregation result. Non-aggregators leave this section empty. The measurement section contains a single entry with the ID of the sensor reporting the measurement, the ID of the zone it belongs to, and the measurement itself. Recall that an aggregator does not include its own measurement in the aggregation result but sends it along for aggregation by its parent.
Figure 4.4: A DCP message: (a) the message proper; (b) data for a single epoch; (c) an aggregation section example; (d) a measurement section example; (e) a complaint section example

**Theorem 4.1.** In each epoch the DCP counts every measurement from every sensor exactly once.

**Proof.** In some epoch \( f \) each sensor sends measurement and aggregation results for the epochs \( f, f-1, \cdots, f-s+1 \) indexed by epoch ID to which they belong. Therefore, without loss of generality we can consider the case when only the current epoch’s data are sent.

By construction, in each epoch each leaf node transmits its measurement to exactly one parent node. An aggregator node does the same and does not include its own measurement in the aggregation result it produces. Thus, each measurement enters the data stream exactly once. Then, double accounting could happen only upstream from the reporting node. A value from some node \( n_1 \) is counted twice by some node \( n_2 \) if and only if there exist more than one delivery/aggregation path from \( n_1 \) to \( n_2 \). This would imply that the delivery/aggregation structure used by the DCP is a DAG, not a tree. This, in turn, would imply that there is at least one node in the delivery/aggregation structure that has more than one outgoing edge. By construction, however, every node in every epoch selects a single random neighbor (closer to the server). Thus, multiple outgoing edges - and double
accounting of measurements - cannot happen. \textbf{QED}

The complaint section holds information about sensors whose measurements violated the network constraints $\phi$ and/or $\varphi$. Each entry contains two IDs: one of the suspected sensor, the other of the reporting one (Figure 4.4(e)). If a sensor transmits a DCP message and then detects a violation, it files the requisite complaint in the next data epoch. Figure 4.4(e) provides a conceptual view of the complaint section. Although we expect only a small number of complaints in each epoch, the number of entries in this section is bounded only by the total number of sensors. Each complaint represents a directed edge in the network topology graph, and the complaint section represents a subset of all edges. To solve the problem we use Bloom filters [128] for compact representation of complaint sections in DCP messages. Recall that an $m$-bit Bloom filter $b$ encodes a subset $S$ of some $n$-element universal set $U$ ($m < n$) by applying a family of hash functions \( \{h_i : U \rightarrow Z_m\} \) to every $s \in S$ and setting bits $b_{h_i(s)}$ to 1. A Bloom filter may produce false positives, i.e., indicate that some element is present in the set while it actually isn’t. Values $m$ and $l$ and hash functions $h_i$ must be chosen to minimize communication and computation cost while providing an acceptable rate of false positives.

Note that any given sensor may include in the complaint section only a small subset of edges, namely, only edges to its neighbors. This set is known to the sensor at the end of the setup phase, and it never grows. We propose to pre-compute the hash values $h_i$ and store them in a table indexed by neighbor ID. A sensor with $d$ neighbors will require a table $ld \log_2 m$ bits long.

The last section of a DCP message, the trace, helps to hold sensors accountable for the aggregation results they produce. (The server can use traces to identify dead sensors as well.) It records the delivery tree (i.e., traversed edges in the SN’s topology graph) as it is being built during data submission. Recall from Section 4.2 that in each data epoch the SN constructs a new delivery tree representing a random spanning tree in the delivery graph.
Every sensor is aware of its surroundings. Among other things it has a list of lower-ranked neighbors each of which can serve as the sensor’s parent in a delivery tree. When ready to transmit a DCP message in some epoch, the sensor picks a random lower-ranked neighbor using the uniform distribution. This neighbor becomes the aggregator of the sensor’s data for that epoch. If sensor $v$ receives DCP messages from sensors $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$ in some epoch, the trace it produces is $T = \bigcup_{k=1}^{n} T_{i_k} \cup \bigcup_{k=1}^{n} \{(v_{i_k}, v)\}$. Leaf sensors produce empty traces. By the time a DCP message arrives at the server, its trace section contains as many edges as there are sensors in the network; this number may be very large. We use Bloom filters for trace representation.

**Securing DCP.** To prevent impersonation attacks against honest sensors DCP messages must be authenticated. The simplest approach is for sensors to use authenticated local broadcast. For instance, the Localized Encryption and Authentication Protocol (LEAP), due to Zhu et al. [129], requires each sensor to establish a *cluster* key shared with all its immediate neighbors. The sensor then uses this key to compute the HMAC-MD5 [130] message authentication code over the contents of the message and append the code to it. One benefit of secure local broadcast is that it supports encryption without violating our requirement for neighborhood watch.

**Sensor Exclusion Protocol.** Based on the information supplied by the network the server determines which sensors are: (a) live and trustworthy, (b) live but not trustworthy, (c) dead, and (d) isolated. Every $k$ epochs the network interrupts its normal operation and spends two epochs, called respectively the *maintenance epoch* and the *key disclosure epoch*, on excluding sensors in groups (b), (c), and (d). (The key disclosure epoch is discussed at the end of this section.) Section 4.4 describes how the server assigns sensors to one of these groups. Once the decision to exclude is made, the server must notify the SN about it. It does so by flooding.

The transmission schedule for the SEP is constructed as follows. If there are $m$ slots in
Figure 4.5: SEP and NAP: (a) a SEP message; (b) a single sensor section example; (c) a NAP message. The SEP reason field takes one of three values: U (untrustworthy), D (dead), or I (isolated).

an epoch and a sensor uses some slot \( i \) for data submissions, it must use the slot \( m - i \) to transmit sensor exclusion messages. The conceptual structure of an SEP message is shown in Figure 4.5(a). The message contains zero or more entries, shown in Figure 4.5(b), each holding a sensor ID (MAC address) and an optional reason for exclusion. To keep the size of messages tightly bounded SEP uses Bloom filters [128]. Currently we are not using the reason field, so a single Bloom filter (sensor set) should suffice. Since Bloom filters may generate false positives, an SEP message constructed by the server may contain more sensors than intended. The server is responsible for retrofitting the false positives into its list of isolated sensors and recomputing the Bloom filter until the set represented by the filter and the actual set of newly excluded sensors become equal.

Note that each SEP message acts as a heartbeat confirming to each recipient that it’s still part of a functioning SN. If a sensor does not hear from the server for \( r \) consecutive maintenance epochs it assumes it must have missed a NAP message and needs to shut down.

**Network Abandonment Protocol.** The format of a NAP message is given in Figure 4.5(c). The server sends a single NAP message in a maintenance epoch once it deems the SN too compromised to be useful. The message propagates through the SN by flooding. Having received a NAP message each sensor rebroadcasts it to all its neighbors.

**Securing SEP and NAP.** The ability to forge both SEP and NAP messages gives
the attacker much power to manipulate the SN. To guard against it we have the server authenticate all messages sent to the SN. As public key cryptography is too expensive for sensor networks [86][131] we adopt $\mu$TESLA, a popular secret key-based broadcast method with delayed key disclosure [131]. The initial key is preset when sensors are configured and readied for deployment. In each maintenance epoch the server uses the next available key from the chain to sign its message. In the key disclosure epoch that follows the server discloses the current key. Server messages are authenticated using an HMAC-MD5-based MAC.

One important drawback in $\mu$TESLA is that it requires protocol re-initialization once the end of the key chain is reached. This is not a problem in our setting. For instance, in an SN running for 30 years with maintenance epochs occurring every minute the server will require only $\approx 128.4M$ of storage (for 256-bit keys).

Authentication of server messages in and of itself does not completely solve the problem of accurate sensor exclusion. This is because dropping a message that fails to validate will only help the attacker. Since SEP messages propagate by flooding, each sensor may receive multiple copies of each message. Normally, the attacker will not be able to modify all the copies. Therefore, the sensor can store all distinct copies and identify the authentic one when the key is disclosed. (Note that each sensor must forward all collected copies of the message.) NAP messages are secured in the same manner.

4.4 The Server and Its Operation

There are three types of nodes in the system. Ordered by decreasing complexity they are: (a) a server, (b) sensors, and (c) base stations. Sensors are described in detail throughout the rest of the paper. The base stations act as proxies between the sensor network and the server; their role in the network is quite passive. In this section we discuss the behavior of the server.
First and foremost, the server is responsible for reconciling multiple contradictory results received for each epoch. As discussed in Sections 4.2 and 4.3 for some epoch $f$ the server collects results in the epochs $f$, $f + 1$, ..., $f + s - 1$. The attacker cannot ensure consistency of these results, and the server should receive several distinct values when under attack. To arrive at the true outcome for the epoch $f$ the server chooses the value garnering plurality of the votes. Figure 4.6 shows the result accumulation matrix maintained by the server.

**Reputation network.** Another task the server must perform is to identify dead, untrustworthy and isolated sensors, and to provide information about them to the SN. Recall that a compromised sensor may lie about its measurements, aggregation results, or both. Complaints lodged by sensors in DCP messages aimed at discovering incorrect measurement results turn our SN into a reputation network. A suitable reputation system needs to accommodate the following facets of our problem:

- The server performs reputation analysis in each epoch.
- Only negative reputation reports are sent to the server.
- Honest sensors always provide trustworthy reputation reports, compromised sensors may lie.
- Compromised aggregators may drop reputation reports produced by honest sensors.
One suitable approach is that of the beta reputation system due to Jøsang and Ismail [93]. Their system models a peer-to-peer network with two-party transactions; for every transaction a peer executes it is expected to report its satisfaction level as a number in \([-1, 1]\). The system collects these reports for each subject peer and attempts to predict its future behavior using the beta distribution. It assigns different weights to reports based on the reporters’ own reputation (this is called reputation discounting). Having received a DCP message, our server eliminates duplicates and assigns the satisfaction level of \(-1\) to all complaints. Then it excludes sensors with posteriori probabilities of ”bad” behavior greater than the system parameter \(p_0\).

**PAC learning model.** When in some epoch \(f\) the server decides on the true result for the epoch \(f - s + 1\), it has \(s\) DCP aggregation sections and \(s\) traces corresponding to them. The server can construct a worksheet similar to the one shown in Figure 4.7. The table captures the role each sensor played in each epoch (leaf vs. aggregator) and truthfulness of the result conveyed by the SN in that epoch. Intuitively, an incorrect result produced by the SN in an epoch where a sensor acted as an aggregator provides evidence against this sensor.

Since we want the server to base its conclusion on correlation between incorrect (correct) results and membership in the set of aggregators (leaf sensors) it is preferable that the probability of a given sensor acting as an aggregator in a given epoch be neither too small nor too large. The following theorem shows that this is, indeed, the case.

**Theorem 4.2.** Let \(SN\) be a sensor network of infinite size where each sensor has at least \(m\) lower-ranked and at least \(m\) higher-ranked neighbors \((m > 3)\). Then on average fewer than \(2/3\) of all sensors in \(SN\) act as aggregators in a delivery tree.

**Proof.** Consider an arbitrary sensor in the network. Let \(l\) be the number of its higher-ranked neighbors, and \(\{k_i\}_{i=1}^{l}\) be their respective counts of lower-ranked neighbors. Then the probability that the sensor’s ”children” don’t select it as an aggregator
Figure 4.7: Worksheet constructed by the server to help detect compromised aggregators.

The column A indicates whether a sensor was an aggregator in a particular epoch, the column C, whether the SN "cheated" in that epoch. (The server considers the network as cheating in epoch $f$ if the aggregation result for any epoch $e$ reported in $f$ was incorrect.)

is $p = (1 - 1/k_1) \ldots (1 - 1/k_l)$. Since all $k_i \geq m$, $p \geq (1 - 1/m)^l \geq (1 - 1/m)^m$. When $m > 3$ $(1 - 1/m)^m \approx 1/e$; thus, $p \geq 1/e$. Let $N$ be the number of leaves and $n$ the total number of sensors under consideration. $N$ is a binomially-distributed random variable with probability of success $p$. Then, $E[N] = np$, and $E[N/n] = p \geq 1/e$. Then the expected fraction of aggregators is $1 - p \leq 1 - 1/e \approx 0.632$. QED

The server’s problem is akin to the problems of identification of faulty components in industrial quality control, software bug isolation, and fault localization in communication networks. At a more fundamental level it can be cast in terms of the generic problem of learning of a Boolean formula. Each sensor $i$ can be represented by a Boolean variable $x_i$ equal to 1 when the sensor is an aggregator, and to 0 when it’s a leaf; the Boolean function $f(x)$ is equal to 1 if the aggregation section for some epoch (in some DCP message) agrees with the "true" result for that epoch, and to 0 otherwise. The server can use techniques from the field of computational learning theory, specifically, the probabilistically approximately correct (PAC) learning model [132]. In this model an adversary chooses a concept class (the "form" of the Boolean formula) $C$, an unknown target concept (a Boolean formula) $c \in C$, and a sample of inputs drawn from some arbitrary distribution $D$. Given $C$, a sample of inputs and outcomes, an accuracy metric, an accuracy bound $\epsilon$ and a confidence level $\delta$ the learner must formulate a hypothesis about the target concept (i.e., construct a formula
approximating the target formula) that differs from it by more than $\epsilon$ with probability at most $1 - \delta$. Let $S_c = \{s_{ik}\}_{k=1}^m$ be the set of currently compromised sensors in the network, and $S_{c1}, \ldots, S_{ct}$ be subsets of $S_c$ such that sensors in each $S_i$ acted as aggregators in a recent epoch and as a group managed to propagate an incorrect value to the server. We say "as a group" because it’s likely that they had to coordinate their effort to achieve the desired outcome. The total number of non-empty subsets on $S_c$ is, of course, $2^{|S_c|} - 1$ but we believe the adversary can engage only a small number of them because:

- It’s better to have a group of sensors cheating a little than a single sensor cheating a lot. Thus, single-sensor "teams" (subsets) are unlikely which makes $t \leq m/2$.

- Coordinating a set of compromised sensors requires time.

- The server is constantly watching the network and prunes suspicious sensors.

Thus, although $m$ may be fairly large, the number of "teams" of compromised sensors should remain small. If the server assumes that not more than $k$ teams may be formed before it finds them out, the problem can be cast as the $k$-term-DNF learning problem [132]. For example, let’s assume that $S_c = \{s_1, s_2, s_3, s_5, s_8, s_{13}, s_{21}\}$ and two teams have been formed: $\{s_1, s_5, s_{21}\}$ and $\{s_3, s_{13}\}$. If we also assume that each team is always successful (i.e., if all team members are aggregators the SN always reports an incorrect result) the target concept can be expressed as $c = x_1x_5x_{21} \lor x_3x_{13}$. Although a $k$-term-DNF concept is not PAC-learnable by $k$-term-DNF (i.e., using $k$-term-DNF hypotheses) it is learnable in polynomial time by $k$-CNF [132]. Therefore, the server can efficiently identify compromised aggregators and exclude them from the SN.

The exclusion of compromised and dead sensors leaves fewer sensors in the SN to take measurements and perform aggregation. It also makes the SN more vulnerable in the face of an attack since there are fewer sensors to watch over one another and fewer candidates for transmission of DCP messages. If for some sensor the number of neighbors falls below
or the number of lower-ranked neighbors falls below \( h \), the server considers the sensor isolated and excludes it from the SN. This may cause other sensors to become isolated, and then others still, and so on.

4.5 Setup and Building Blocks

In this dissertation we consider two-dimensional sensor networks. Sensors are distributed randomly, for example, from aircraft. For simplicity we assume that the deployment area is a square. Once deployed, the sensors discover their neighborhood and start the setup phase.

How many sensors are required to cover the area of deployment? This number is driven by the actual phenomenon measured by the network as well as by the physical characteristics of the environment. In addition, our approach to resilience requires redundancy; we’d like each sensor to have at least \( H \) neighbors. On the other hand, adding more sensors increases the epoch length, and at some point the delay may become unacceptable to the server. We’ll assume that \( L > H \) is the target maximum number of neighbors for each sensor. The following theorem provides guidance on how many sensors to deploy in the network to meet these requirements.

**Theorem 4.3.** Let the SN deployment area be a square \( a \times a \), and the sensors, whose transmitting range is \( r \ll a \), be uniformly distributed over the deployment area. Then, assuming we want each sensor to have \( k \in [H, L) \) neighbors with probability at least \( 1 - \epsilon \) we must deploy \( n \in \left[ \frac{H + \ln(1/\epsilon) + \sqrt{2H \ln(1/\epsilon) + \ln^2(1/\epsilon)}}{p} \right], \frac{L \epsilon}{p} \) sensors where \( p = \pi r^2 / a^2 \).

**Proof.** Consider an arbitrary point \( A \) in the deployment area. A sensor falls within \( A \)’s \( r \)-range with probability \( p = \pi r^2 / a^2 \) (the ratio of the areas involved). Placing a sensor at random is a Bernoulli experiment with probability of success \( p \). If \( n \) is the total number of sensors, the expected number of sensors in \( A \)’s \( r \)-range is \( \mu = np \). Using the Chernoff bound
Pr\[k \geq (1 - \delta)\mu \geq 1 - e^{-\mu \delta^2/2} \] (0 < \delta \leq 1), and setting \( \delta = \sqrt{2 \ln(1/\epsilon)/\mu} \) [133], on the one hand, and \( H = (1 - \delta)\mu \), on the other, we arrive at the quadratic equation \((1 - H/\mu)^2 = 2 \ln(1/\epsilon)/\mu \). This equation has two solutions: \( \mu = (H + \ln(1/\epsilon)) \pm \sqrt{2H \ln(1/\epsilon) + \ln^2(1/\epsilon)} \). The smaller of the two must be rejected since \( \sqrt{2H \ln(1/\epsilon) + \ln^2(1/\epsilon)} > \ln(1/\epsilon) \) and \( \mu = (H + \ln(1/\epsilon)) - \sqrt{2H \ln(1/\epsilon) + \ln^2(1/\epsilon)} < H \) which contradicts the equality \( H = (1 - \delta)\mu \) with 0 < \( \delta \leq 1 \). Therefore, if \( n \geq H + \ln(1/\epsilon) + \frac{\sqrt{2H \ln(1/\epsilon) + \ln^2(1/\epsilon)}}{\mu} \) with probability at least 1 - \( \epsilon \) the point \( A \) will have \( H \) or more sensors in its \( r \)-range.

Now let’s consider Markov’s inequality \( \Pr[k < L] > 1 - \mu/L \). To make sure \( \Pr[k < L] > 1 - \epsilon \) we can insist that \( 1 - \mu/L \geq 1 - \epsilon \) or, equivalently, that \( \mu \leq L \epsilon \). Thus, if \( n \leq \frac{L \epsilon}{\mu} \) with probability at least 1 - \( \epsilon \) the point \( A \) will have \( L - 1 \) or fewer sensors in its \( r \)-range. QED

We use a variant of TDMA for scheduling of DCP transmissions. Each epoch is divided into slots; every sensor is assigned a slot it must use for transmissions. The transmission schedule has to guarantee that aggregators send DCP messages after the sensors that may supply input data to them. To ensure this, we impose a static ranking of nodes in the SN. A DCP message can only travel from a higher-ranked sensor to a lower-ranked one. We use a distributed ranking algorithm called Directional-Nearest-Neighbor-Tree (Directional-NNT) from [134]. To decide which of two sensors should have a lower rank they use a simple procedure taking into account the sensors’ positions and MAC addresses (Figure 4.8). The schedule must also guarantee that simultaneous radio transmissions do not interfere with one another. Any distributed algorithm satisfying these requirements may be used. See, for example, literature on contention-free protocols using distance-two graph coloring.

In order to operate properly each sensor must discover its neighbors in the deployed network. In addition, as the Data Collection Protocol relies on TDMA for scheduling of
Compare-Nodes \((E_T, v_1, v_2)\)

1 if not \((v_1, v_2)\) in \(E_T\) then

2 return 0

3 if \(v_1.x < v_2.x\) then

4 return -1

5 if \(v_1.x > v_2.x\) then

6 return 1

7 if \(v_1.y < v_2.y\) then

8 return -1

9 if \(v_1.y > v_2.y\) then

10 return 1

11 return \(v_1.id < v_2.id \) ? -1 : 1

Figure 4.8: Deciding which sensor has lower rank. Input: set of edges \(E_T\) in the topology graph, two vertices \(v_1\) and \(v_2\); output: 0 if the vertices are not neighbors, -1 if \(v_1\)'s rank should be lower, 1 if \(v_1\)'s rank should be higher. The variables \(v_1\) and \(v_2\) are of type \texttt{vertex} containing three fields: the \(x\) coordinate, the \(y\) coordinate, and the sensor ID.
Figure 4.9: Time to detect a compromised sensor as a function of $\alpha$ for a closest sensor attacker for different profiles in Table 4.3

transmissions, sensors in the network must have synchronized clocks. Many approaches are available in this area.

Khan et al.’s algorithm Directional-NNT [134] we use to rank sensors in the network requires that the sensors know their position, at least relative to one another. Many approaches were proposed to address this localization problem.

4.6 Evaluation

To evaluate our approach we need to answer two questions:

- How effective is it? In other words, can it prevent or delay a successful attack and/or increase the cost of mounting it?

- How efficient is it, i.e., what is its overhead in terms of message and time complexity and energy consumption?
Figure 4.10: Time to detect a compromised sensor as a function of $\alpha$ for a random sensor attacker for different profiles in Table 4.3

Figure 4.11: Time to detect a compromised sensor as a function of $\alpha$ for a swath attacker for different profiles in Table 4.3
In our simulation experiments we studied the effects of hardening the basic data delivery protocol by randomizing delivery trees and providing mutual constraint validation. Our simulated server contains a beta reputation engine. For simplicity, the whole network constitutes a single sensor zone. Each sensor "measures" a Boolean environment variable (a constant); each aggregator computes a Boolean function of its inputs. Some time after the setup phase the adversary initiates an attack. It employs mobile actuators that move from sensor to sensor and install malicious software on them. A compromised sensor always reports an incorrect value (the reverse of the value reported by honest sensors). We simulated three types of actuators:

- **Closest sensor.** Wherever it finds itself, this type of actuator locates the closest sensor and attacks it.

- **Random sensor.** This actuator picks a random position in the sensor network and compromises the closest sensor found there, then repeats the cycle again.

- **Swath.** The swath actuator is similar to the random actuator in that it picks a random position in the network and moves there. Unlike the random actuator, it finds all sensors within certain distance from its route and attacks them.

Table 4.2 lists all input parameters accepted by our simulator. (Threshold parameters in rows 10, 14, and 19 are explained in our discussion of Figures 4.16, 4.17, 4.18, and 4.19.) To make the number of possible simulation experiments manageable we hypothesized on how each parameter in isolation may affect the average time to detect a compromised sensor. Values expected to yield the best time were grouped into a *fast* profile, the worst time - into a *slow* profile, and some intermediate time - into a *medium* one. These profiles are given in Table 4.3. Note that in several experiments (Figures 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.16, 4.17, and 4.18) we vary a profile parameter listed in Table 4.3. When we use the term "profile" in such cases it refers to the remaining three parameters. For example, Figures
Table 4.2: Simulation configuration parameters

<table>
<thead>
<tr>
<th>Ref</th>
<th>Input parameter</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Actuator speed</td>
<td>Real, m/s</td>
<td>0.6</td>
</tr>
<tr>
<td>2</td>
<td>Actuator type</td>
<td>Enumerated</td>
<td>See above</td>
</tr>
<tr>
<td>3</td>
<td>Aggregation constraint</td>
<td>Enumerated</td>
<td>Always-TRUE</td>
</tr>
<tr>
<td>4</td>
<td>Aggregation function</td>
<td>Enumerated</td>
<td>OR</td>
</tr>
<tr>
<td>5</td>
<td>$\alpha$</td>
<td>Probability</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Area size</td>
<td>Real, m</td>
<td>$\frac{P_k}{57.5}$</td>
</tr>
<tr>
<td>7</td>
<td>Base stations</td>
<td>Count</td>
<td>$P_k$</td>
</tr>
<tr>
<td>8</td>
<td>Compromised sensor measurement</td>
<td>Enumerated</td>
<td>Always-FALSE</td>
</tr>
<tr>
<td>9</td>
<td>$k$</td>
<td>Count</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>Lower ranked neighbors threshold</td>
<td>Count</td>
<td>$2P_{13}$</td>
</tr>
<tr>
<td>11</td>
<td>Maximum neighbors</td>
<td>Count</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>Measurement constraint</td>
<td>Enumerated</td>
<td>EQUALS</td>
</tr>
<tr>
<td>13</td>
<td>Minimum neighbors</td>
<td>Count</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Reputation threshold</td>
<td>Probability</td>
<td>0.4</td>
</tr>
<tr>
<td>15</td>
<td>Physical environment</td>
<td>Enumerated</td>
<td>Always-TRUE</td>
</tr>
<tr>
<td>16</td>
<td>Radio range</td>
<td>Real, m</td>
<td>75</td>
</tr>
<tr>
<td>17</td>
<td>Ranking algorithm</td>
<td>Enumerated</td>
<td>Directional-NNT</td>
</tr>
<tr>
<td>18</td>
<td>$s$</td>
<td>Count</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>Total neighbors threshold</td>
<td>Count</td>
<td>4</td>
</tr>
</tbody>
</table>
Table 4.3: Simulation configuration profiles

<table>
<thead>
<tr>
<th>Input parameter</th>
<th>Profile</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fast</td>
<td>Medium</td>
<td>Slow</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1.0</td>
<td>0.25</td>
<td>0.03125</td>
<td></td>
</tr>
<tr>
<td>Area size</td>
<td>150</td>
<td>300</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>$k$</td>
<td>2</td>
<td>8</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Minimum neighbors</td>
<td>64</td>
<td>16</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total number of sensors</td>
<td>200</td>
<td>300</td>
<td>600</td>
<td></td>
</tr>
</tbody>
</table>

4.9, 4.10 and 4.11 demonstrate a dependency on $\alpha$, the intercepting frequency. A profile in these experiments consists of the values for area size, maintenance epoch frequency $k$, and minimum required number of neighbors.

In the first experiment we measured the time it takes the server to establish that a sensor is compromised versus $\alpha$, the probability of a sensor intercepting another sensor’s messages in a given epoch. The results are presented in Figures 4.9, 4.10 and 4.11. These experiments confirmed our expectation that as $\alpha$ increases, the time to detect and exclude a compromised or isolated sensor decreases. Indeed, with a larger $\alpha$ more neighbors per time unit are watching each sensor and, thus, more complaints are sent to the server if the sensor is compromised.

In our simulation experiments the server excludes sensors if they are (a) compromised, (b) isolated because too few of their neighbors remain in the network, or (c) isolated because too few neighboring aggregators remain in the network. In the experiments whose results are shown in Figures 4.12, 4.13 and 4.14 we ran our simulator until all sensors became compromised and measured the fraction of sensors excluded by the server. (The actuator was unaware of sensor exclusion and had to compromise all sensors.) As can be seen in the figures, the server successfully excluded sensors as required in the ”medium-” and ”slow-profile” environments for all types of attackers. Its detection rate was over 95%. In the ”fast-profile” case it performed well when the network was attacked by closest-sensor and
Figure 4.12: Fraction of sensors compromised by a closest sensor attacker and detected by the server under different simulation profiles in Table 4.3.

random-sensor actuators and the probability of intercepting was above 0.25. When the adversary deployed a swath attacker in a "fast-profile" environment, the server’s success rate was 75% or less.

Our safeguards introduce overhead that consumes bandwidth and energy. Figure 4.15 illustrates this overhead measured in bytes transmitted in DCP messages as a function of the parameter $s$. As may be expected, the DCP message volume measured in bytes grows linearly with the parameter. In the experiments reported in this section the adversary attacked only the "data" part of DCP but not the "security" part. We believe higher values of $s$ will make the SN more reliable in the latter case. As in many other systems, there is a trade-off between service integrity, on the one hand, and network utilization (and energy consumption), on the other. We plan to study this in our future work.

Intuitively, the more neighbors a compromised sensor has, the more complaints will be sent to the server and the sooner it will decide to exclude that sensor. By "sooner" we mean
Figure 4.13: Fraction of sensors compromised by a random sensor attacker and detected by the server under different simulation profiles in Table 4.3

Figure 4.14: Fraction of sensors compromised by a swath attacker and detected by the server under different simulation profiles in Table 4.3
Figure 4.15: Overhead introduced by the proposed safeguards under different simulation profiles in Table 4.3 (both axes are in the log scale)

the time in epochs, not in seconds, since duration of an epoch increases with higher sensor density if all other parameters remain the same. Figures 4.16, 4.17 and 4.18 illustrate this dependency. Note that the time to detect starts to decrease when the minimum number of neighbors for sensors in the network reaches about 16. When this number is between 4 and 16, the time to detect actually increases. Our conjecture is that the time to detect is affected by the type of attacker, on the one hand, and the exclusion thresholds, on the other. In our model the server uses three thresholds:

- **Reputation**: the sensor reputation below which a sensor is considered compromised.

- **Total neighbors**: the minimum number of neighbors below which a sensor is considered isolated (to ensure a sufficient number of interceptors).

- **Lower rank neighbors**: the minimum number of lower ranked neighbors below which a sensor is considered isolated (to ensure a sufficient number of candidate aggregators).
Figure 4.16: Time to detect a compromised sensor as a function of the minimum number of neighbors for sensors in the network for different types of attackers (fast profile)

Figure 4.19 suggests that at least for some combinations of the threshold values the average time to detect a compromised sensor actually decreases as the initial minimum number of neighbors increases. Although the functions do not appear monotonically decreasing, their trendlines certainly are. For example, when the lower-ranked neighbor threshold is 1, the minimum acceptable number of neighbors is 8, and the reputation threshold is 0.4, the time it takes to detect a compromise drops from 4.459 epochs when the minimum initial number of neighbors is 8 to 1.743 epochs when it is 14. Similarly, for the 1-8-0.5 combination the time to detect drops from 2.604 to 1.319 epochs. Additional research is required in this area.

4.7 Conclusions

In this chapter we discussed stealth attacks against sensor networks. The first contribution of this chapter is the observation that many physical phenomena exhibit strong spatial
Figure 4.17: Time to detect a compromised sensor as a function of the minimum number of neighbors for sensors in the network for different types of attackers (medium profile)

Figure 4.18: Time to detect a compromised sensor as a function of the minimum number of neighbors for sensors in the network for different types of attackers (slow profile)
Figure 4.19: Time to detect a compromised sensor as a function of the minimum number of neighbors for sensors in the network. The first number indicates the lower rank neighbors threshold; the second, the total neighbors threshold; the third, the reputation threshold correlation between point measurements. This correlation is expressed in our model in the form of measurement constraints. Sensors observe their neighbors during transmission and file "complaints" if their neighbors’ data violate the constraint when compared with their own.

The second contribution is the collect-detect-exclude framework which allows the server to collect information from the network, detect compromised sensors, and exclude them from the network by notifying other sensors about the compromise. This framework takes advantage of the Data Collection, Sensor Exclusion, and Network Abandonment Protocols.

Third, our system lets sensors select the next hop during transmission thus forming a randomized delivery (and aggregation) tree. Coupled with this is the requirement in each epoch to transmit information about \( s > 1 \) recent epochs. This allows the server to reconcile contradictory results (by voting) and to detect an attack when a fairly small
fraction of sensors have been compromised.

Fourth, we proposed that the server use a beta reputation system [93] to identify compromised sensors based on complaints from the network. We also propose the use of the probabilistically approximately correct (PAC) learning model [132] to help the server discern the adversary’s strategy and identify coordinated attacks involving multiple compromised sensors.

Finally, we simulated sensor networks under several attack models. Our results show that increasing the frequency of sensors’ intercepting one another’s messages decreases the average time to detect a compromise. It also increases the chances of catching all compromised sensors. We also studied dependency between the density of the sensor network and the time to detect a compromise. When the number of neighbors was 16 or higher, the time to detect fell with the increase in density. Preliminary results indicate this may be true for sparser networks as well.
Chapter 5: Peer-to-Peer Massively Multiplayer Online Games

5.1 Introduction

In this chapter we discuss FRAPPE\textsuperscript{1}, a fraud-resistant architecture for peer-to-peer environments. At the core of our approach is an observation that under reasonable assumptions it is possible to construct trusted groups out of non-trusted peers, and use these groups as units of computation in the game. FRAPPE provides general-purpose infrastructure services as well as security features aimed at hiding information from peers (while making it available to trusted groups), reducing traceability through the use of anonymizing tunnels, and avoidance of conflicts of interest. More precisely, FRAPPE achieves its security objectives because:

- It constructs a distributed trustworthy game engine.
- It provides a hybrid peer-to-peer server-assisted architecture.
- It protects the system against active attacks by forming trusted "clusters" from groups of non-trusted peers.
- It protects the system against passive attacks by using secret sharing among non-trusted peers.

FRAPPE also introduces a useful primitive, called \textit{local broadcast with verification} (LBV), used to solicit services of peers in a particular neighborhood and subsequently verify that the peers properly executed the protocol.

\textsuperscript{1}The French noun "frappe" /frap/ means "strike" but may also mean "scoundrel", "bad lot" [135].
This chapter is organized as follows. Section 5.2 describes our game model. The fundamental elements of the architecture such as trusted peer groups, infrastructure services and basic security services are discussed in Section 5.3. FRAPPE’s network components are covered in Section 5.4. Section 5.5 explains how the components described in Sections 5.3 and 5.4 work together to provide a secure and efficient game-playing environment. In Section 5.6 we present results of our simulation experiments focused primarily on local broadcast with verification. Conclusions are given in Section 5.7.

5.2 Game Model

We explained in Section 2.6.2 that MMOG architectures are divided into three main categories: (a) client/server, (b) peer-to-peer, and (c) distributed. It should become clear from Section 5.2.1 below that ours is a hybrid architecture: peers actively participate in the management of the game’s infrastructure but each participating peer’s scope is limited to a particular zone (called a cell in FRAPPE). This certainly makes FRAPPE a peer-to-peer architecture. However, a central server initially manages the game alone; it also serves as the ultimate source of trust in the game and acts as a last resort for providing services when nothing else is available. This adds a client/server aspect to FRAPPE. Finally, multiple infrastructure services exist in the game at any point in time, each service consisting of one or more instances. Each service instance, although composed of multiple peers in many cases, acts as a single unit and communicates with other instances as in the distributed server architecture.

The rest of this section preamble discusses assumptions we made when designing our system. Since FRAPPE is a complete architecture, these assumptions apply to several different aspects of the system. We divide them into three categories:

- System and network: those pertaining to the infrastructure of the game as a complex
peer-to-peer distributed system.

- Game: those pertaining to the “application” (i.e., game-playing) elements of the overall system.
- Security: those pertaining to the security aspects of the game and/or the game’s infrastructure.

Table 5.1 explains all assumptions and provides their categorization.

5.2.1 Server-Assisted Peer-to-Peer Architecture

Most existing massive multiplayer online games are based on the client/server architecture [99]. Many use server replication and partitioning of the game world to achieve scalability. Some (for example, World of Warcraft) run multiple independent game worlds each managed by a single server or a group of servers and scale up by creating new worlds and bringing more servers online. We propose a hybrid P2P-based architecture shown in Figure 5.1. Peers in FRAPPE act in two capacities: (a) as clients to the game engine and (b) as elements of the game engine’s infrastructure. Several infrastructure services exist in the game, and each service may consist of one or more instances. As we explain in more detail later in this chapter, some services are implemented as fixed singletons in the Server, some as peer group-based distributed components, and some (more precisely, one, the Network Beacon Service) as collections of trusted peers run by the game-hosting company. Some services require dual implementations, e.g., as a singleton and as a peer group-based component, to allow FRAPPE to be scalable, highly available and fault-tolerant. Figure 5.1 emphasizes the peer group-based approach to service implementation.

Under our assumptions (Table 5.1) the game-hosting company runs a fixed highly available and trusted Server. Physically it may be implemented as a farm, a cluster or a distributed federation of servers; it may (and will) also host multiple services. Functionally,
Table 5.1: FRAPPE’s assumptions about supported massively multiplayer online games

<table>
<thead>
<tr>
<th>#</th>
<th>Assumption</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The game world is a three-dimensional cube. Support for three dimensions is crucial to enable players to &quot;immerse and escape&quot;. As pointed out in Section 2.6.4 immersion/escapism is the highest-important motivational factor for online game players [98]</td>
<td>Game</td>
</tr>
<tr>
<td>2</td>
<td>Every logged-in player has a single avatar, his/her persona in the game</td>
<td>Game</td>
</tr>
<tr>
<td>3</td>
<td>When a player logs out his/her avatar’s state is saved, and it &quot;hibernates&quot; without affecting, or being affected by, the game</td>
<td>Game</td>
</tr>
<tr>
<td>4</td>
<td>Each avatar is mostly interested in objects (or avatars) in its proximity</td>
<td>Game</td>
</tr>
<tr>
<td>5</td>
<td>Client and network failures are rare</td>
<td>System</td>
</tr>
<tr>
<td>6</td>
<td>The server is trusted and available 100% of the time</td>
<td>System</td>
</tr>
<tr>
<td>7</td>
<td>The game network is never partitioned</td>
<td>System</td>
</tr>
<tr>
<td>8</td>
<td>One or more reliable communication protocols (e.g., TCP, reliable UDP, etc.) are available to the game</td>
<td>System</td>
</tr>
<tr>
<td>9</td>
<td>Overlay multicast is used for some communications: machines of some or all logged-in players may be asked to route messages to other machines. More specifically, peer-to-group communications through anonymizing tunnels (see Section 5.4.3) require overlay multicast as a primitive</td>
<td>System</td>
</tr>
<tr>
<td>10</td>
<td>No more than $\alpha &lt; 1/2$ of players are cheaters, and no insider cheating is possible</td>
<td>Security</td>
</tr>
<tr>
<td>11</td>
<td>Out-of-band attacks are not possible, all attacks happen in and through the game. (Among other things, it means that the clients’ IP addresses cannot be forged)</td>
<td>Security</td>
</tr>
<tr>
<td>12</td>
<td>No side or hidden channels can be established by the players</td>
<td>Security</td>
</tr>
<tr>
<td>13</td>
<td>Dishonest game clients are Byzantine</td>
<td>Security</td>
</tr>
<tr>
<td>14</td>
<td>Each client is fully controlled by its player: no external attacks are possible against game clients</td>
<td>Security</td>
</tr>
<tr>
<td>15</td>
<td>A player’s account may have at most one open session with the game at any given time</td>
<td>Security</td>
</tr>
</tbody>
</table>
Figure 5.1: FRAPPE’s hybrid architecture. The shaded elements together constitute the trusted game engine. The Service on the left is implemented as a trusted group of non-trusted peers just like the Service on the right whose cross-section is shown.
however, we consider it a single Server. The Server performs three main roles in our model:

- The common source of trust in the game.

- The last resort for service invocation in those cases when distributed services fail or cannot be located.

- The sole manager of the game when the number of simultaneous concurrent users is small. We expect the Server to manage the game through the introduction stage.

The Server is also responsible for keeping the game’s persistent state and for recovering the game in case of failure.

5.2.2 Management of the Game World

We model the game world as a three-dimensional cube with objects and avatars. Objects may be inanimate or animated; an avatar is a special kind of object representing its player’s persona in the game. Objects are just collections of attributes of various types; they may also contain other objects. Avatars (and, perhaps, other objects) have senses (or sensors) that allow them to detect phenomena in the game world. In this chapter we make a common assumption that only avatars have senses and at that only a single one: vision. We also assume that visibility of an object to an avatar depends solely on the distance between them and does not take intervening objects into account. Based on our analysis of the source code for *WoWWoW* [136], a server simulator for the very popular game *World of Warcraft* [137], this is a reasonable and practical assumption: subject to other conditions (such as visibility levels or playing on the same map), visibility is evaluated only based on distance. For simplicity we do not discuss other (also distance-based) interactions in *World of Warcraft*, such as spells.

To manage the game world in a scalable fashion the Server divides it into cells. Each cell is managed by a single logical *Cell Manager* (CM). The fixed Server is the CM for
the topmost cell representing the whole game world. A cell may be further divided into sub-cells based on its CM’s load. Division is always done into equal cubes as illustrated in Figure 5.2. After multiple divisions we get a tree similar to the one in Figure 5.3. A parent Cell Manager continuously exchanges information with its children. If it deems their load sufficiently light it may dissolve its child CMs, merge the child cells into a single cell and manage it directly.

Note that the Cell Manager has complete control over its cell; it acts as the authoritative source for all changes to the state of the cell. Players whose avatars are located in a particular cell (each avatar may exist in one and only one cell) receive authoritative updates to their view of the game from the cell’s CM. When a player plays the game his client program may provide "play-ahead", i.e., it may react to the player’s actions locally, without consulting the CM. These changes, however, remain tentative until approved by the CM.
Honest players (and clients) have nothing to worry about: the client wouldn’t let the player violate the rules of the game.

The game time is divided into frames. In each frame clients send tentative updates to the respective Cell Managers. The Cell Managers verify, order and apply the updates, query their neighbors as needed to update visibility information, and perform required housekeeping. Then they forward to each client the necessary updates for its view of the game world.

5.3 FRAPPE Elements

5.3.1 Trusted Whole out of Non-Trusted Parts

Since we cannot trust individual peers, depending on them for vital services in the game may lead to rampant cheating and denial of service to honest players. We posit, however, that a sufficiently large group of peers can be trusted even if each individual peer is not. FRAPPE uses quorum-based protocols working under the assumption that a properly constructed, sufficiently large group of peers can be trusted with high probability. How many participating peers are required to make a group trusted? The following theorem provides an estimate for this number.

**Theorem 5.1.** Let $1 - \beta$ be the lower bound on the probability that a group of peers is trusted ($0 < \beta < 1$) and $\alpha < 1/2$ be the probability of a given peer being dishonest. Then for the group to be trusted it must contain at least $n = (2 \ln \beta) / (\ln(2\alpha) + 1 - 2\alpha)$ peers.

**Proof.** Let $n$ be the required number of members in a peer group and $X_i$ be a random variable equal to 1 if the $i$th member is dishonest and 0 otherwise. We want to establish under what conditions the number of dishonest players $X = \sum_{i=1}^{n} X_i$ does not exceed $n/2$ or, more precisely, under what conditions $\Pr[X \leq n/2] \geq 1 - \beta$. ($X$’s expectation is $\mu = \alpha n$.) This is equivalent to the inequality $\Pr[X > n/2] < \beta$. Using the Chernoff
Figure 5.4: Minimum required number of members in a peer group as a function of the probability of cheating $\alpha$ (for different values of $\beta$)

bound $\Pr[X > (1 + \delta)\mu] < F^+(\mu, \delta) = (e^\delta/(1 + \delta)^{1+\delta})^\mu$ valid for $\delta > 0$ [133] and setting $(1 + \delta)\mu = (1 + \delta)\alpha n = n/2$, we get $\delta = 1/(2\alpha) - 1$. Using the inequality $F^+(\mu, \delta) = e^{-C(\delta)\mu}$ where $C(\delta) = (1 + \delta)\ln(1 + \delta) - \delta$ [133], setting the right-hand side to $\beta$ and using the result for $\delta$, we get $n = (2 \ln \beta)/((\ln(2\alpha) + 1 - 2\alpha))$. QED

Based on this formula, Figure 5.4 shows the relationship between the probability of cheating by a given peer and the required size of a peer group. If the number of cheaters is very large, groups will grow as well, and the clustering approach may become impractical. Table 5.2 illustrates what we believe to be absolute upper bounds on the feasible size of peer groups. In practice we expect this number to remain much smaller, perhaps, between 5 and 11.

### 5.3.2 Infrastructure Services

Any large-scale distributed system requires a number of infrastructure services for object discovery, time synchronization, key management, policy enforcement, and so on. FRAPPE is no exception. Some services such as cell management are intrinsic to the application at
Table 5.2: Boundary values of $\alpha$ for several confidence levels and corresponding peer groups

<table>
<thead>
<tr>
<th>$\beta$</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>11.888</td>
<td>18.943</td>
<td>31.020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.10</td>
<td>9.138</td>
<td>14.560</td>
<td>23.843</td>
<td>41.553</td>
<td></td>
</tr>
<tr>
<td>0.15</td>
<td>7.529</td>
<td>11.996</td>
<td>19.644</td>
<td>34.236</td>
<td></td>
</tr>
<tr>
<td>0.20</td>
<td>6.387</td>
<td>10.177</td>
<td>16.665</td>
<td>29.045</td>
<td>56.795</td>
</tr>
<tr>
<td>0.25</td>
<td>5.501</td>
<td>8.766</td>
<td>14.355</td>
<td>25.018</td>
<td>48.921</td>
</tr>
</tbody>
</table>

hand, i.e., a massively multiplayer online game. Others were introduced because of some of the design decisions we made in support of this application. Still others must exist in almost any distributed system. In the rest of this section we will enumerate all infrastructure Services in FRAPPE one by one.

**Directory Service (DS).** The purpose of the Directory Service is to allow various parties to locate other Services. The Directory Service has two-tier architecture: (a) multiple dynamic replicas of the Service run on the Internet powered by peer groups and (b) the Server runs a fixed Service component used to locate the dynamic replicas. If a party finds that its list of known Directory Service replicas is obsolete it can contact the fixed Service component to refresh it.

**Time Service (TS).** To prevent replay attacks and guarantee freshness of the numerous cryptographic tokens in use FRAPPE needs to synchronize clocks of all participants in the game. The Server acts as the authoritative source of time. Potentially, there are three types of time in FRAPPE: (a) that of the players’ machines, (b) that of the players’ game clients and (c) virtual time in the game world. The Time Service in FRAPPE regulates time on the players’ clients. Replicas of the Time Service receive time updates from the Server (or from other replicas), and answer queries from any valid participant in the game. Each Time Service replica is implemented as a peer group.
**Load Tuning and Availability Service (LTAS).** This Service receives load reports from other Services and starts or shuts down their instances as required. Most Services are implemented as sets of replicas. These replicas should be uniformly distributed throughout the game network. To create a new Service replica the LTAS chooses an appropriate location using a supernode selection technique similar, for example, to the ones discussed in [138] and sends a hop-by-hop *Service Creation Request* towards that location. Once a peer in the vicinity of that location is reached, it must broadcast a *Service Participation Request* using local broadcast with verification described in Section 5.4.5. All peers reached by the broadcast request acquire a *Service Participation Authorization Token* (if they can) and connect to the LTAS; after verification of the broadcast the LTAS selects a sufficient number of peers and contacts them directly to start the new replica. If too few or no *Service Participation Responses* come within a predefined timeout period, the LTAS repeats the procedure slightly altering the desired target location. Each LTAS replica is implemented as a peer group. The LTAS must implement predictive algorithms so that it can manufacture enough Service instances in time during the growth stage in the game’s lifecycle.

**Network Beacon Service (NBS).** To support network space-based routing (see Section 5.4.2) FRAPPE needs a distributed service to establish coordinates of each peer in a trustworthy manner. The Network Beacon Service is used for this purpose. The NBS is a distributed Service with multiple geographically dispersed instances. Each peer must collect information from a sufficiently large and diverse set of NBS instances to be able to receive information about its location in the network space. The coordinates are given in the form of a *Network Location Evidence Token* bound to the recipient’s IP address. Unlike most other Services in FRAPPE the NBS is implemented as a collection of fixed trusted hosts belonging to the game-hosting company. Once a sufficient number of peers join the game, FRAPPE migrates to peer group-based instances of the NBS. Multiple mechanisms for network space-based location discovery have been proposed [139][140][141].
Network Location Service (NLS). The Network Location Service is responsible for minting *Network Location Tokens* based on evidence produced by instances of the Network Beacon Service. FRAPPE gives these tokens out only once when the player (and, equivalently, his game client) logs in. Hence, this Service doesn’t have high scalability requirements; it is implemented as a single fixed component in the Server.

Neighbor List Update Service (NLUS). This Service is responsible for reissuing *Neighbor List Tokens*. As peers log in and log out of the game, the Server makes their presence information and network coordinates available to the instances of the NLUS. Depending on the needs of local broadcast with verification (Section 5.4.5) and other requirements, peers may contact the NLUS more or less frequently to refresh their neighbor lists. Instances of the NLUS are implemented as peer groups.

Token Renewal Service (TRS). Numerous FRAPPE parties (the Server, Services, Cell Managers, etc.) issue limited-lifetime cryptographic tokens. These tokens must be renewed. Since some token types are very short-lived and since many entities may want to renew them FRAPPE outsources token renewal to the TRS. A TRS instance knows how to mint all eligible tokens; all the token possessor needs to do is present a fresh token previously signed by the original issuer or by the TRS and prove possession of the private key corresponding to the token. In the initial stage of the game each issuer is responsible for renewing tokens it grants; then the LTAS allocates peer group-based TRS instances.

Service Participation Authorization Service (SPAS). This Service keeps track of peers currently engaged in peer groups delivering services to a FRAPPE-based game. The SPAS implements a variant of the Chinese Wall access control model to avoid conflicts of interest. In its decision making it uses both current information and recent historic information. The SPAS is implemented as a peer group-based Service. The token issued by this Service, *Service Participation Authorization Token*, binds the name of the Service instance in question, the IP address of the subject and one of its public keys.
Game World Persistence Service (GWPS). GWPS is implemented as a single fixed Service running on the Server. Its role is to collect transaction logs from Cell Managers and periodically apply them to the "backup" copy of the game world maintained by the Server.

5.3.3 Authentication, Confidentiality and Pseudonymity

FRAPPE heavily relies on public key cryptography for message authentication. Each entity owns one or more limited-lifetime cryptographic tokens and uses the corresponding private key(s) for message signing. Token validation is done in a manner similar to validation of X.509 certificates: tokens must chain to a trust anchor. In many cases the trust anchor is the Server but other types of trust anchors are supported. Tokens are also used for authorization (granting and verification). For example, an instance of the Token Renewal Service will mint a new token for a caller if and only if (a) it can construct a trust path to the Server and (b) the old token was issued by the Server or (c) the old token was issued by a TRS instance authorized by the Server (i.e., tokens from parties other than instances of the TRS are not valid).

We propose to use the RSA algorithm because (a) it is suitable for both signing and encryption, (b) it is well understood, and many efficient implementations exist and (c) efficient algorithms for multiparty RSA key generation and encryption are readily available. The system must balance the cost of key generation and private key operations (which in RSA are much more expensive than public key operations) against the possibility and impact of key compromise by dishonest players. FRAPPE uses this analysis to assign required key lengths to the many tokens it uses as well as required frequencies of token renewal (issuance of newly time-stamped tokens with the same key) and key renewal (issuance of tokens with a new key).

Authenticating a security principal means proving that it is who it claims it is. The principal must do so by producing something that it is, something that it has, or something that
it knows. Unfortunately, P2P-based MMOGs pose challenges to this standard approach. First, (colluding dishonest) peers may share their private keys and thus freely impersonate one another. Second, it is desirable to minimize (or completely eliminate) traceability and linking of operations in the game; this is done through the use of pseudonyms. What this means is that finding out the true identity of a peer may be difficult or impossible in some situations (i.e., pseudonymity works too well). FRAPPE uses the following two principles to assure security against impersonation attacks:

- Logged-in colluding dishonest players may pass control over their avatars to one another. This is acceptable because our security goals are: (a) only legitimate logged-in users can play, (b) no user gets unfair advantage over other users, and (c) no user can affect other users in ways not prescribed by the game. The sharing of avatars does not violate any of these goals: our malfeasants are logged-in; regardless of which client controls the shared avatar, real control is with the (trusted) Cell Manager which prevents violations of the laws of the game by any client; the malfeasants cannot gain control over anybody else’s avatar.

- In all other transactions a peer proves possession of a private key bound to its IP address. Under our assumptions IP addresses cannot be forged (Section 5.2). Thus, a peer’s interlocutor can always ascertain its identity.

As we mentioned, FRAPPE uses many types of tokens for authentication and authorization. Detailed information about these tokens is given elsewhere in Section 5.3 as well as in Section 5.5. Table 5.3 provides a summary of all tokens in FRAPPE and their content and usage.


Table 5.3: Summary of FRAPPE tokens

<table>
<thead>
<tr>
<th>Token type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conflict of Interest Token</td>
<td>Issued to a CM’s member to prove that its avatar is (isn’t) in the Notice Region of a neighboring cell</td>
</tr>
<tr>
<td>In-Cell Avatar ID Token</td>
<td>Issued to a peer by its avatar’s CM. Contains the avatar’s pseudonym in that CM’s cell</td>
</tr>
<tr>
<td>IP Address Token</td>
<td>A short-lived token issued at login time and used to contact the Network Beacon Service</td>
</tr>
<tr>
<td>Neighbor List Token</td>
<td>Contains a list of a peer’s neighbors and their coordinates</td>
</tr>
<tr>
<td>Network Location Evidence Token</td>
<td>Contains evidence of a peer’s location furnished by an instance of the Network Beacon Service</td>
</tr>
<tr>
<td>Network Location Token</td>
<td>Contains a peer’s coordinates</td>
</tr>
<tr>
<td>Service Participation Proof Token</td>
<td>Contains proof that a peer’s game client participated in a peer group-based FRAPPE Service</td>
</tr>
<tr>
<td>Service Participation Authorization Token</td>
<td>Issued to authorize a peer’s game client to participate in a peer group-based FRAPPE Service</td>
</tr>
<tr>
<td>Session Token</td>
<td>Contains the Server-assigned session ID</td>
</tr>
</tbody>
</table>

5.3.4 Conflicts of Interest

Because the peers simultaneously participate in adversarial game playing and cooperative infrastructure management special care must be taken to avoid conflicts of interest: a peer should not be able to learn anything from its role in the infrastructure that may enable it to gain an unfair advantage in the game. In Section 5.3.2 we describe the Service Participation Authorization Service responsible for some aspects of managing the conflicts of interest arising from participation of the same peer in several infrastructure Services. Here we will discuss FRAPPE's mechanisms for avoidance of conflicts of interest based on dual roles peers play, one in the game proper, the other in the infrastructure. More specifically, we want to make sure that a peer’s participation in a Cell Manager does not give it any advantages in the game. FRAPPE achieves this by making sure that a peer does not participate in any CM for cells its avatar can "touch", for example, because the cells are within its visibility range. Granted, it can collude with another peer and pass valuable
In each frame each Cell Manager computes visibility information for each avatar in its cell. In an unlikely event that an avatar can “peer” into a neighboring cell the CM sends queries to the neighbor’s CM to discover all objects there that the avatar can “see”.

In FRAPPE a peer cannot be part of a CM managing the peer’s avatar. To avoid conflicts with neighboring CMs FRAPPE divides each cell into conflict of interest regions (Figure 5.5). The regions may be dynamic or static. (Because their goal is to manage potential future conflicts of interest they may, for example, take the avatar’s speed and direction into account.) As long as an avatar sensing area remains in the **Safe Region**, there is no concern. Once it reaches into the **Notice Region** its owner is given notification by the Cell Manager that a possible conflict may arise. If the owner is part of the neighboring CM, it must withdraw from it, and if the owner is honest it will do so. A dishonest owner may hope to take advantage of the situation and stay. If an avatar reaches the **Alert Region**, the
avatar’s CM notifies all affected neighboring CMs. It provides the avatar’s pseudonym and other information. At this point each peer-member of a notified CM must prove to the rest of the CM that the "offending" avatar is not its avatar. Since each CM member is known to other CM member only by its IP address (and will want to keep it that way), it will have to prove to the others that its avatar ID is different from that of the offending avatar without actually disclosing the former. See Section 5.5 for details.

5.4 Network Components

5.4.1 Secure Multiparty Computation

Constructing trusted peer groups out of non-trusted peers solves the problem of active attacks: whatever artifact is produced by the group, more than half of its members must have agreed to produce it (assuming correct secure multiparty computations). Passive attacks remain possible: if a group member learns a secret he can broadcast it to all his buddies or to the world at large. To prevent passive attacks we divide all attributes of all objects held by a peer group into sensitive and non-sensitive. (The game software may do it, for example, by class: a particular attribute in all objects of a given class belongs to one or the other category; other approaches are also possible.) Non-sensitive attributes are freely shared as is between the members of a group - and, thus, may be shared with others. Sensitive attributes are treated as secrets, and none of the participants in the group knows it. In a \((t, n)\)-sharing scheme each of the \(n\) participants holds a share of the secret, any subset of \(t + 1\) or more participants can recover the secret, but any subset of \(t\) or fewer participants cannot [142]. Assuming an odd number \(n = 2k + 1\) of peers in each peer group FRAPPE uses \((k, 2k + 1)\) sharing because under our assumptions at most \(k\) participants are dishonest. The original secret sharing scheme due to Shamir [143] did not protect against sabotage by dishonest parties; to address this Verifiable Secret Sharing (VSS) was
invented. Finally, to prevent powerful attackers from learning long-lived secrets *Proactive Secret Sharing* (PSS) was introduced. A PSS scheme allows periodic generation of new shares for the same original secret and share recovery from corrupted or failed participants. FRAPPE uses PSS-based secret sharing mechanisms from [142] to store avatars’ coordinates and IDs.

In many circumstances a peer group will need to cryptographically sign a data item, decrypt a message or generate a shared RSA key pair. This is an application of *threshold cryptography*. Efficient algorithms for these operations have been known for some time. See, for example, [144] for more details.

When a secret shared amongst members of a peer group is transferred to members of another group a *secret redistribution* mechanism is required to ensure full transfer without leakage. Wong et al. [145] and Chen et al. [146] proposed secret redistribution schemes for the PSSes in [142] used by FRAPPE.

Secret sharing and threshold cryptography are examples of a more general class of algorithms, called *secure multiparty computation* (SMC). In general an SMC ensures that a set of parties some of which are dishonest can compute a particular function in a trustworthy manner. FRAPPE extensively uses SMC. For example, Cell Managers periodically expel members from their midst; decision whom to expel is based on an SMC-based random selection algorithm. Another example is joint generation of random strings (e.g., IDs) required by some Services. Note that in all cases when SMC is used we assume polynomial attackers and the existence of a broadcast channel. Since FRAPPE peer groups are small, these assumptions allow us to efficiently use cryptographic, rather than information-theoretic, techniques to provide security in a multiparty setting.
5.4.2 Overlay Routing

Routing protocols in ad hoc networks (of which P2P networks are a subclass) are susceptible to a variety of attacks. This is because most of them rely on dissemination of proximity information by the routing nodes themselves, and if these nodes cheat they may be able to completely change the behavior of the network’s routing infrastructure. For an overview of security threats to routing in ad hoc networks see, for example, [71]. FRAPPE uses network space routing: each peer represents a point in a space; its coordinates are assigned (and signed) by a trusted third party. Hence, malicious manipulation of the routing infrastructure becomes more difficult. See Section 5.3.2 for information on the Network Beacon Service and Network Location Service that, together, assign network coordinates to FRAPPE peers.

There are two flavors of network space-based routing in FRAPPE. One occurs when the initiator knows the exact target with which it wants to communicate; the other, when the initiator knows the approximate location of the target and is willing to tolerate errors. In either case, it finds a neighbor closest to the target and forwards its message to it; the neighbor repeats the process, and so does its neighbor, until the message arrives at the destination. Each peer on the way checks whether the message was addressed to it (the first case) or whether it’s located sufficiently close to the target location (the second case).

The approach we just described assumes that every peer knows its neighbors and their location in the network coordinate system. When a peer logs in, the Server provides information about the initial set of neighbors for that peer. Subsequently, the peer periodically contacts the Neighbor List Update Service to refresh this list. In addition, between these updates it regularly probes its neighbors to learn about their neighborhood. To maintain a neighborhood list of constant size it may replace logged out or crashed neighbors with its neighbors’ neighbors. If new neighbors come online, the peer will exclude other neighbors located further away.
5.4.3 Anonymizing Tunnels

One of our design goals is to minimize the ability of FRAPPE entities to link the various tokens owned by a peer. Specifically, we need to hide each client’s IP address. We could adopt a general-purpose anonymity system such as Onion Routing [147] or Tarzan [148]. These systems, however, may be too powerful - and incur unnecessary cost - for our needs: we only want the destinations not to know the message originators. As we explain in Section 5.5, FRAPPE requires anonymity only in peers’ communications with peer groups, i.e., in a multicast setting. Multicast-specific anonymizing approaches [149][150] rely on trusted proxies which are not available in our environment. We settled for a simple heuristic algorithm illustrated in Figure 5.6:

- The initiator finds network locations of the peers in the group.
- The initiator computes the location of the group’s "center of mass" $C$.
- The initiator flips a biased coin with the probability of tails $p$ multiple times until it gets heads. Heads in the $m$th coin flip mean that the $m$th hop in the tunnel must connect to the target, tails - that it must connect to a random peer closer to the target.
- At each hop:
  1. Decrement $m$. If it becomes zero, connect to the target peers one by one; if not, find a point $1/m$ of the way from itself to $C$.
  2. Engage in a sub-protocol to discover the next hop: find a neighbor closest to the point found above and send a Tunnel Hop Request (THQ) to it. The neighbor will find its own neighbor still closer to the point and so on until the THQ gets close enough to that point. The first "close enough" peer responds to the initiating
hop with a *Tunnel Hop Response* (THS) establishing itself as the next hop in the tunnel.

3. Forward the *Tunnel Establishment Request* (TEQ) to the next hop.

   - The target group constructs a *Tunnel Establishment Response* (TES), signs it and sends it back to the initiator.

   - At each hop: append its *Network Location Token* (NLT) to the message, sign it and forward to the next hop.

   - The initiator receives and stores information about the tunnel. If the TES does not arrive within a predefined timeout period, the initiator uses a point close to $C$ as a target and repeats the process. Each TEQ has its own unique nonce and is signed by the initiator.

Note that anonymity is achieved because the target peer group (or any intermediate hop) does not know how far the tunnel goes after the previous hop in the tunnel. The expected length of the tunnel is $1/p$. Because tunneling imposes a performance penalty, shorter tunnels are preferable. We expect $p$ to be on the order of 0.2-0.3 to satisfy both anonymity and efficiency requirements.

### 5.4.4 Reliable Communication with Peer Groups

We make an assumption (Section 5.2) that a reliable communication protocol is available to the game. It solves the problem of point-to-point communications but, in and of itself, does not address the issue of communications with groups. This is a complex problem, and it is exacerbated by the fact that group members may cheat, even during message transmission. On the other hand, our groups are small so efficiency of multicast communications is not as important as it is in large-scale systems. We distinguish three cases of group communication in FRAPPE:
Figure 5.6: An anonymizing tunnel

- A peer to a group.
- A group to a peer.
- A group to a group.

We also need to consider two types of group messages: (a) those containing identical copies and (b) those containing shares of data. In the rest of this section we discuss these communication patterns.

**Peer-to-group messages.** FRAPPE uses reliable atomic multicast in such communications [151][152]. A message is successfully delivered to a \((k, 2k + 1)\)-group if and only if at least \(k\) distinct copies can be assumed to have been delivered. (We do not say ”acknowledged” because some reliable multicast protocols use only negative acknowledgements or a combination of positive and negative ones [151].) The two types of messages are processed in the same manner: all copies, whether distinct or identical, bind to the same message ID, and acknowledgements are matched against it.
Group-to-peer messages. An ”identical data” message is considered successfully delivered if and only if the peer receives copies from at least $k$ distinct members of the sending $(k; 2k + 1)$-group. A ”split data” message is considered successfully delivered if and only if the peer can reconstruct the data. When it receives a message with a new ID from one of the members of the group the peer starts a timer and accumulates as many copies of the message as possible while attempting to reconstruct the data. The moment such reconstruction succeeds, the timer is stopped, an acknowledgement is sent (if needed) and the message is made available to the game client’s application layer. The group must run an agreement protocol to come to an agreement whether the message has been acknowledged (or can be assumed delivered through other evidence).

Group-to-group messages. To minimize complexity, FRAPPE decomposes a group-to-group message into a series of peer-to-group messages followed by an agreement protocol between the sending group’s members.

5.4.5 Local Broadcast with Verification

In some situations, for example, when it needs to create a new instance of a Service or replenish a peer group, FRAPPE takes advantage of local broadcast: one or more peers are asked to broadcast a message in their vicinity. (Broadcast here means application-layer, not network-layer, broadcast.) The message contains a time-to-live (TTL) counter decremented at every hop; once the counter reaches zero, the message is not propagated any further. Peers reached by the message are expected to contact some entity, for example, an instance of the Load Tuning and Availability Service. How does this entity know that nobody cheated during the broadcast? It doesn’t. Instead it estimates how random and complete the broadcast was.

The protocol works as follows. The initiator (e.g., an instance of the LTAS) generates a random nonce and contacts the ”seed” peer or peers. The seed peers start the broadcast.
As the message travels hop-by-hop, a trace is constructed. The trace is empty at the start. Every receiver appends the IP address of the previous hop, its own Network Location Token (NLT) and the IP address of the next hop, and signs the message with the private key corresponding to the NLT. When the last hop contacts the initiator, the latter has full information about the message’s path. As explained in Section 5.4.2 FRAPPE peers maintain neighbor lists of approximately equal size; it means that returned paths should be fairly uniformly distributed around the seed peers if no cheating occurs. If any of the peers cheat, for example, by forwarding only to dishonest peers in collusion with them or by dropping messages altogether, some paths will be biased. The initiator then prunes such paths and selects a sufficient number of peers at random from the remaining ones. See Figure 5.8 for an example of a trace.

5.5 Game Operation

We assume that before logging in for the first time every player acquired the game client software, registered, and received a unique user ID and a password from the game-hosting company. At the time of registration the user selects the type of his avatar (if multiple types are supported by the game), the initial position of his avatar in the game world and the initial set of resources available to it. The game places the avatar in hibernation until the player logs in.

**Player login.** The player logs in to the game by communicating with the Server over a secure connection, for example, using the TLS protocol. The player presents his user ID and password to the Server, the Server presents an X.509 certificate to the player. To support better authentication to the user the game-hosting company should acquire an enhanced validation (EV) certificate [153] from a reputable source. After successful mutual authentication the Server generates a key pair, and sends to the client an IP Address Token containing the client’s IP address. It also gives the client a list of addresses of Network
Beacon Service instances to contact and receive relative network location information. The client contacts the NBS, collects location information and forwards this information back to the Server (the singleton Network Location Service). The NLS verifies evidence supplied by the client and, if it is sufficient, issues a *Network Location Token* binding the client’s location and IP address to its public key. The Server then discovers the client’s avatar’s home cell and notifies its CM that a request to join is imminent from the client. When contacting the CM the Server provides the symmetrically encrypted client IP address and a random session ID. Finally, the Server responds with:

- The client’s *Session Token* containing the session ID.
- The client’s *Network Location Token*.
- The client’s *Neighbor List Token* containing the initial list of peers (IP addresses) in the client’s neighborhood.
- The name of the player’s avatar’s home Cell Manager and its group address (a set of IP addresses of its members).
- The symmetric key used to encrypt the IP address in the message to the home CM.

To find the avatar’s home CM the Server (partially) floods the game world tree with the avatar’s ID (split into shares) and its location in the game world and receives back the Cell Manager’s name and address. The player contacts the home CM and authenticates using the newly received session ID (and proving possession of the corresponding private key). All communications between peers and Cell Managers are done via anonymizing tunnels (see Section 5.4.3) to hide avatar ID/IP address binding. The Cell Manager generates an in-cell avatar ID and a key pair, and issues an *In-Cell Avatar ID Token* to the client. (Henceforth the client will refer to the avatar by the in-cell ID.) This token also contains the encrypted IP address previously received by the CM from the Server.
Neighbor list maintenance. At login time the Server issues to every peer a Neighbor List Token. This token must be periodically refreshed. To minimize traffic caused by the neighbor list renewal process, FRAPPE uses a gossip-based mechanism to propagate neighborhood information between formal neighbor list updates: all peers regularly exchange heartbeats with their neighbors, and each heartbeat message carries the most current Neighbor List Token. If a particular neighbor is lost (due to logout or a crash), each peer replaces it with the best selection from its neighbors’ neighbor list. This resulting dynamic neighbor list slowly deviates from the game infrastructure’s view of each peer’s neighborhood, but it provides a good approximation until the neighbor list is updated from the Network List Update Service (NLUS).

Token renewal. Tokens are renewed by the instances of the Token Renewal Service discussed in Section 5.3.2.

Player logout. When a player wants to log out, he directs his client to send a Hibernate Request to his avatar’s current CM and receives a signed proof of hibernation. The client then contacts the Server, presents the proof and signs out. If a client does not refresh its player’s Network Location Token within time period $T$, the Server discovers the player’s current cell by flooding the game world tree, directs it to hibernate the player’s avatar, and invalidates his session.

Cell Manager hygiene. Each Cell Manager periodically expels a member. This is done to prevent any member from accumulating too much historic information about the cell that can be used to its player advantage. To pick a ”victim” at random the CM runs the following random selection algorithm:

- Each member generates a random number $0 \leq n_i < n$ (where $n$ is the total number of peers in the CM), commits to it (by hashing and signing it), and broadcasts it to the rest of the group. (Since we do not have a broadcast medium in our setting, we simulate authenticated broadcast using point-to-point links [154]. This type of broadcast
is different from just sending the message to all other members: the members must have assurance that each of them receives the same message.

• Each member reveals its number \( n_i \) by broadcasting it to the rest of the group.

• Each (honest) member computes the value \( m = \sum_{i=1}^{n} n_i \mod n \).

• Each (honest) member notifies the \( m \)th member it must leave.

**Service member replenishment.** When a Service loses one or more members it invites new ones to reestablish the quorum. Since Service members frequently engage in rather expensive multi-party computations their tolerance of delays is low; hence, they should be located in relative proximity of one another. In FRAPPE replenishment works as follows:

• The remaining members agree on \( d \), the number of new members to invite.

• Based on \( d \) they calculate the required time-to-live (TTL) for the solicitation broadcast.

• Each remaining member starts a local broadcast with verification (Section 5.4.5) specifying the peer group as the callback address.

• All (honest) peers reached by the broadcasts try to obtain a *Service Participation Authorization Token* from the SPAS. Those that succeed in getting the token contact the Service.

• The Service collects the responses, verifies them, selects the acceptable subset, and notifies the selected peers.

• The Service recovers the missing shares of its private key and generates a new key using a Proactive Secret Sharing protocol.
Cell splitting. If load becomes too high (e.g., too many avatars flock to its cell) a Cell Manager splits the cell into sub-cells. A cell is always divided into eight equal parts (see Figure 5.2). To solicit membership in the new Cell Managers, the CM starts eight simultaneous procedures similar to those run by the LTAS instances. Eventually, it collects enough responses, generates (shares of) private keys for the new CMs and transfers to them their respective portions of the game world.

Cell merging. Every CM (except the root) sends periodic load reports to its parent. The parent may deem the cumulative load sufficiently light to be able to manage it by itself. If so, it sends a Cell Dissolution Request (CDQ) to its children and receives Cell Dissolution Responses (CDP) with their respective cell state information.

Cell-to-cell avatar transfer. When an avatar enters a cell (or awakes from hibernation after the initial login) the cell’s CM assigns it a random in-cell avatar ID. This ID is included in an In-Cell Avatar ID Token (re)signed in every frame as long as the avatar stays in the cell.

When an avatar enters a new cell, the old CM constructs an Avatar Transfer Request (ATQ) to the new CM. The ATQ contains full information about the avatar. Note that some attributes (the avatar’s location in the game world, its permanent avatar ID, and, perhaps, others) are split into shares and must remain split in the new cell. These shares are transferred using one of the secret redistribution schemes discussed in Section 5.4.1. At the end of the transfer the avatar’s owner is notified through the old tunnel; the tunnel is torn down and a new tunnel is set up connecting the owner’s client with the new Cell Manager.

To locate the Cell Manager the avatar is about to cross into the old CM runs the following algorithm (see Figure 5.7):

* Assuming the total size of the game world is $L$, the maximum depth of the game world tree is $d$, the actual crossing point is $(x_0, y_0, z_0)$ and the avatar is crossing through a
Figure 5.7: Finding the new Cell Manager for an avatar leaving a cell

side parallel to the plane $yz$, find the imputed crossing point $(x_0 + L/2^{d+1}, y_0, z_0)$ (or $(x_0 - L/2^{d+1}, y_0, z_0)$ depending on the direction).

- Calculate the name of the smallest possible cell $z_{0\alpha_1\alpha_2...\alpha_d}$ containing the imputed crossing point.
- Propagate a Cell Manager Discovery Request (CMDQ) up the tree until the CM’s name prefix-matches the name $z_{0\alpha_1\alpha_2...\alpha_d}$.
- Propagate the request down the tree finding the longest prefix match with $z_{0\alpha_1\alpha_2...\alpha_d}$.
- The matching leaf CM contacts the originating CM.

**Queries.** As an avatar moves around the game world, new objects appear in its visibility range and old objects disappear. In each frame each CM computes this visibility information for all its avatars. The visibility range of an avatar located close to its cell’s boundary may protrude beyond the cell; thus, the CM must send a query to its neighbor requesting to provide data on all objects in its cell visible to the avatar. In a general case avatars may have multiple senses and the game world may provide obstacles to them. As we explained
in Section 5.2.1, FRAPPE only assumes the existence of a single sense - eyesight - and visibility is based only on the distance between the observing avatar and the object.

A query consists of the avatar’s in-cell ID, shares $L_i = (X_i, Y_i, Z_i)$ of its coordinates in the game world, and information about its owner’s tunnel with the avatar’s CM. The receiving CM runs the following algorithm:

- Split the square of the visibility radius $R^2$ into shares $R^2_i$.

- For each object likely to be in the avatar’s range split its coordinates into shares $l_i = (x_i, y_i, z_i)$. Using the Gennaro-Rabin-Rabin (GRR) Verifiable Secret Sharing scheme [155] calculate the shares of the function $f(L, l) = (X-x)^2 + (Y-y)^2 + (Z-z)^2 - R^2$. We chose the GRR because, unlike many other secret sharing schemes, it allows efficient privacy-preserving calculation of products (and, therefore, squares) of secrets. Note that splitting into shares must be done according to the GRR as well. If the object itself is an avatar its coordinates are already split albeit, possibly, using a different VSS scheme. Thus, the group must either re-compute the shares according to the GRR or request the avatar’s owner to provide the shares (the owner knows its avatar’s location whereas no CM member does).

- Reconstruct the value of $f(L, l)$ and compare it with zero. The avatar can see the object if and only if $f(L, l) < 0$.

**Query responses.** Query responses, if any, are sent directly to the avatar’s owner via the tunnel it established with the avatar’s CM. Since CMs are trusted, the owner must only verify that the response comes from a legitimate CM.

**Notices and alerts.** In Section 5.3.4 we explained how FRAPPE guards against conflicts of interest: a peer is notified if its avatar gets too close to the cell’s border in case the peer is a member a CM for one of the surrounding cells. The peer is expected to withdraw from its CM if this is the case, and if it’s honest it will do so. A dishonest
peer may want to remain in the CM. If the avatar reaches the Alert Region of the cell, the CM alerts the affected neighboring CMs with the offending avatar’s in-cell ID it received with the query. When a CM receives an alert it requires all its members to prove that the offending avatar is not theirs. Each member opens an anonymizing tunnel to the avatar’s current CM and presents its own In-Cell Avatar ID Token. The CM compares the ID with that of the offending avatar and issues a Conflict of Interest Token containing the outcome of the comparison (true or false) and the encrypted IP address from the In-Cell Avatar ID Token that was presented to it. Then the requester presents the token and the symmetric key used to encrypt its IP address to the rest of the CM. The other members can verify the outcome and check if the token is bound to the presenting member’s IP address.

**Updates.** As we mentioned above, all updates originating at a playing peer are treated with suspicion and marked as tentative. The game client is free to reflect the changes immediately to provide the user with realistic virtual world experience but these changes only become real when the player’s avatar’s CM processes them and sends them back to all affected clients (including the one originating the change).

**Game state persistence.** The state of the game must be persistent so that users may log in and log out at will. One option could be to rely on the tree of Cell Managers to maintain the state of the game. This, however, will not work in the event of a catastrophic failure caused, for example, by partitioning of the game network or complete loss of a Cell Manager. In FRAPPE the Server is the locus of the game’s persistent state. All Cell Managers periodically dump signed transaction logs to the Server. The logs reflect the actual changes to each CM’s cell (rather than the tentative ones requested by the playing peers). Events in the logs are also properly ordered. The Server’s Game World Persistence Service (GWPS) is responsible for receiving the transaction logs, merging them and lazily applying them to its view of the game. These periodic checkpoints allow the GWPS to limit the size of the transaction logs it needs to keep and to quickly recover the game. Although
players may lose some of their most recent updates the GWPS will recover the game to a consistent state.

**Cheater detection and eviction.** In Section 2.7 we referred to several proposals for automatic cheating detection in P2P massively multiplayer online games. Most of these proposals require individual peers to provide information about other peers to the game’s infrastructure. This creates an opportunity for abuse. It can be countered by using a reputation system with *reputation discounting*, a feature that allows evidence from more reputable nodes to have more weight in calculating other nodes’ reputations. Using a reputation system, however, makes the system much more complex.

Our approach relies on a small set of trusted peers, either people or robots. The purpose of these peers is to monitor the game and to identify cheating clients. They collect evidence of bad behavior and make it available to the Server. The Server may run a rule-based system that identifies persistent cheaters based on collected evidence, and evict them from the game.

This "police force" must be able to detect attacks both against the game and against the game infrastructure. In addition to a variety of active and passive attacks a game relying on the peers for infrastructure services is vulnerable to "avoidance" attacks, i.e., refusal of some peers to share the burden of running the game. Since participation in a FRAPPE Service is frequently solicited in the dark, without specifically identifying the target peer, such avoidance may be difficult to detect. To address this issue the FRAPPE game infrastructure issues to peers *Service Participation Proof Tokens*. These tokens record the type of Service the peer participated in and the duration of such participation. The game-hosting company can allow redeeming of these tokens in exchange for price discounts or other perquisites for the players.
5.6 Evaluation

Many of the building blocks discussed in the previous sections were extensively studied and evaluated by other authors. Secure multiparty computation (SMC) is the subject of ongoing active research. Most pertinent results can be found in [154]. Lynch’s book on distributed algorithms [156] provides an excellent overview of efficiency results for authenticated Byzantine agreement. Jarecki evaluates two Proactive Secret Sharing schemes in his Master’s Thesis [142]. Boneh and Franklin analyze efficiency of multiparty RSA key generation in [144]. Wong et al. and Chen et al. evaluate their secret redistribution schemes (both of which can be used in FRAPPE) in [145] and [146], respectively. FRAPPE uses a simplified approach to formation and maintenance of anonymizing tunnels (Section 5.4.3). However, much of formal evaluation of generic anonymizing infrastructures, e.g., [147] or [148], is still applicable in our case. Network space-based routing is extensively analyzed in Ng and Zhang’s paper [157]. One important element of FRAPPE, namely, management of the game world tree still needs to be evaluated. Although Kabus et al. [109] proposed an approach similar to ours, they did not provide either formal and experimental evaluation for it. This will be one of the topics in our future research.

In the remainder of this section we discuss results of our evaluation of local broadcast with verification (LBV) used by FRAPPE to form trusted groups out of non-trusted peers. LBV is described in Section 5.4.5.

To evaluate LBV we created a simulator based on the PeerSim simulation package [158]. The simulator sets up a soliciting entity that invites a peer to participate in a peer group. The peer is expected to broadcast the invitation in its neighborhood. The soliciting entity collects replies, reconstructs the broadcast graph from the traces it receives, and compares it with the expected tree based on neighborhood information included in the traces (which are signed by the Server or its trusted delegate). A sample trace from one of our simulation
experiments is shown in Figure 5.8. The soliciting entity assigns ranks to the senders; ranks are real numbers between 0 and 1. If a peer’s rank is higher than a configured threshold it is accepted into the peer group. We recorded the number of accepted peers and compared it with the number of peers required to form the group.

In all experiments we simulated a fairly large peer-to-peer network (3000 nodes). We used two methods for network location assignment: (a) _uniform_, giving each node _x_- and _y_- coordinates drawn uniformly from the interval [0, 1] and (b) _geographic_, giving each node longitude and latitude. (In the first case Euclidean geometry was used, in the second, spherical geometry.) We based the geographic case on membership statistics for the online game _Hattrick_ [159]. Since the available data were categorized by country [160], we further processed it by attributing all members to a country’s capital, if the country was small, or evenly dividing the membership count between the largest metropolises in the country, if it was large. Figure 5.9 shows a fragment of the resultant configuration file.

The first experiment measured quality of our algorithm in a situation when a group of malicious peers attempts to supply its candidates to the soliciting entity: the "seed" peer, instead of forwarding invitations to its neighbors, sends them to the colluding peers. From the graph in Figure 5.10 one can easily see that in all cases the soliciting entity was able to recognize the attack and reject the selection. For both uniform and geographic coordinates only between 0% and 40% of all reporting peers were accepted; because the whole selection is rejected if the number of reporting peers is insufficient, the attacks failed.

Intuitively, LBV works because a malicious "seed" peer does not have control over selection of its neighbors (the Server does). The fewer of its potential neighbors are dishonest, the more likely it is that the broadcast tree it wants to construct differs from the "natural" broadcast tree induced by the neighborhood relationship. Indeed, Figure 5.11 shows that acceptance rate of rigged delivery trees precipitously falls as the fraction of dishonest potential neighbors decreases.
Figure 5.8: Sample trace received by the soliciting entity in a simulation
# Bolivia
city.16.name=La Paz
city.16.latitude=-16.5
city.16.longitude=-68.150001
city.16.count=2690

# Bosnia
city.17.name=Sarajevo
city.17.latitude=43.83
city.17.longitude=18.33
city.17.count=4313

# Brazil
city.18.name=Brasilia
city.18.latitude=-15.779999
city.18.longitude=-47.9
city.18.count=2349
city.19.name=Sao Paulo
city.19.latitude=-23.56650466037811
city.19.longitude=-46.6424560546875
city.19.count=2349

Figure 5.9: Fragment of the configuration file with membership information for each city

![Graph](image)

Figure 5.10: Relationship between the required number of peers and the fraction of malicious peers accepted by the soliciting entity
Figure 5.11: Relationship between the fraction of dishonest potential neighbors for a seed and the fraction of malicious peers accepted by the soliciting entity (geographic coordinates, each peer has 7 neighbors, TTL set to 5). Numbers in the legend represent the fraction of peers in the broadcast tree required by the soliciting entity. For example, in a system with 5 neighbors per peer and TTL=3 a broadcast tree will have $1 + 5 + 5^2 = 31$ peers; if the required fraction is 10%, at least 4 peers must be accepted by the soliciting entity for LBV to succeed.
The other experiments presented here dealt with "false negatives", i.e., cases where a legitimate set of peers is rejected because the reported traces deviate from precise neighborhood information. This may happen if some of the peers crash or log out while the local broadcast is in progress.

Figure 5.12 shows the number of rejected peers as a function of the neighbor list refresh frequency. We expected that the more frequently a peer updates its neighbor list (by contacting the Server or an NLUS instance), the better results our algorithm will produce. This is because fewer changes should happen between the time of the last update and the response to an invitation sent by the peer. The graph in Figure 5.12 confirms this. When the inter-update period was 12 minutes or less, the algorithm did not make any mistakes. For larger values its quality steadily declined reaching its nadir at about 1.5 hours (the largest value used).
Frequently contacting the Server or instances of the Neighbor List Update Service (Section 5.3.2) to get a neighbor list update will generate significant traffic in a successful MMOG with large membership. We propose that between these updates peers use a gossip-based mechanism to exchange information about their neighbors. If a peer’s neighbor crashes or logs out, the peer can update its neighbor list from the list of its neighbors’ neighbors. Although not precise, this approach allows FRAPPE peers to update their neighbor lists less frequently. Figure 5.13 gives results of the experiments where neighboring peers exchanged heartbeats containing data about their neighbors (signed by the Server or the NLUS). The figure shows an improvement over the previous graph. The fraction of accepted nodes increased and reached values between 75% and 95%. This, coupled with the fact that erroneous acceptance in case of an attack doesn’t rise higher than 40%, suggests that asking for a larger than required selection and combining the neighbor list update mechanism with the heartbeats will produce an optimal solution.

5.7 Conclusions

In this chapter we introduced FRAPPE, a fraud-resistant architecture for peer-to-peer environments, that addresses the problem of cheating in massively multiplayer online games (MMOG). The most important contribution of this chapter is the demonstration that it is practical to construct trusted "supernodes" out of non-trusted peers and have them provide infrastructure services traditionally performed by server components run by the game-hosting companies. Combining 5 to 11 peers into a supernode balances well security and efficiency requirements posed by MMOGs.

Second, we identified the full set of infrastructure services required to run a game. Among them are:

- Directory Service.
Figure 5.13: Relationship between the neighbor list update frequency and the fraction of honest peers rejected by the soliciting entity with heartbeats exchanged every 3 minutes (geographic coordinates, 11-peer selection)

- Time Service.
- Load Tuning and Availability Service.
- Network Beacon Service.
- Network Location Service.
- Neighbor List Update Service.
- Token Renewal Service.
- Service Participation Authorization Service.
- Game World Persistence Service.

Third, in FRAPPE we defined a P2P-based approach to management of the game world. It involves dividing the game world into cells and assigning peer group-based Cell
Managers to control all aspects of the cells’ lifecycle. Cell Managers serve as the sole source of authority for the state of their respective cells: updates submitted by game clients remain tentative until reconciled and approved by the Cell Manager. A dynamic and flexible game world management architecture supports expanding and shrinking of the game world tree as required.

Fourth, we showed that Proactive Secret Sharing schemes and threshold cryptography can be used to share sensitive attributes of avatars and other objects among non-trusted peers in a "supernode", generate private keys, sign messages, etc. FRAPPE pays special attention to avoidance of conflicts of interest. This is achieved through the use of the Service Participation Authorization Service that permits or denies a peer’s participation in a Service based on its other engagements and recent history as well as through the conflict-of-interest avoidance features supported by Cell Managers. To minimize the ability to link game clients to avatars, FRAPPE uses anonymizing tunnels for all communications between peers and respective Cell Managers. Anonymizing tunnels are used in other situations as well.

Fifth, we proposed a simple approach to detection and eviction of misbehaving peers. It requires a small "police force" of people (paid by the game-hosting company) and/or trusted robots to record all violations of the game and report them to the Server. To discourage "free loading" by those members that do not want to share the burden of running the game’s infrastructure, we proposed issuance of Service Participation Proof Tokens that can be exchanged for credits at the game-hosting company’s store.

Finally, we introduced a useful primitive, called local broadcast with verification (LBV). It is used by the Load Tuning and Availability Service and by Cell Managers to invite peers in a particular neighborhood to participate in a peer group-based FRAPPE Service. In LBV a soliciting entity locates a "seed" peer and sends it an invitation to join; the seed is responsible for broadcasting the invitation to all its neighbors, and they, to theirs, etc. All peers reached by the broadcast are expected to contact the designated recipient
allowing the latter to select the most suitable peer subset. The seed or any other peer may cheat and forward invitations only to peers of their choice (for example, hoping to form a completely malicious instance of a Service). LBV attempts to prevent this. We evaluated this approach and showed that it works quite well: a dishonest seed trying to build a malicious group achieved at best a 40% acceptance rate (i.e., only at most 40% of responding peers were accepted) which resulted in the group’s rejection in all cases. On the other hand, honest peers were accepted at the rate of 100% when the neighbor list update frequency was 12 minutes or less. When heartbeats (discussed in Section 5.5) were turned off, it steadily declined after that reaching about 60% when the inter-update period increased to an hour and a half. However, with heartbeats exchanged between neighbors every 3 minutes, the acceptance rate significantly improved to 75%-95%. These experiments indicate that requesting a larger than needed number of peers and combining neighbor list updates and neighbor list exchanges (heartbeats) will result in an optimal approach: instead of rejecting a selection if it’s incomplete, the receiving party can safely accept 75%-95% selections because an attacker can only achieve 40% acceptance and, thus, will not be able to succeed in forming a malicious group.
Chapter 6: Conclusions

In this dissertation we set out to study methods of service assurance in insecure networks with Byzantine adversaries. We considered three types of systems: (a) publish/subscribe networks, (b) aggregating sensor networks, and (c) peer-to-peer massively multiplayer online games containing non-trusted elements as part of their infrastructure. Conceding that these networks cannot be completely protected against attacks, we developed several mechanisms to make them more resilient and withstand partial compromises by Byzantine attackers while continuing to provide service required of them.

First, we considered Internet-scale content-based publish/subscribe networks. We have set the following goals for our work:

- Development of mechanisms ensuring service integrity in Internet-scale content-based publish/subscribe networks with non-trusted intermediaries using overlay multicast.
- Design of efficient algorithms for countering "omission" attacks by malicious intermediaries in Internet-scale content-based publish/subscribe networks.
- Development of an analytical model for estimating cost of delivery trees in Internet-scale content-based publish/subscribe networks based on subscription predicate relationships and network topology information.
- Development of simulation models to study effectiveness and efficiency of the proposed mechanisms improving service assurance in Internet-scale content-based publish/subscribe networks.
To achieve these goals we used the dynamic service composition approach to construct overlay multicast trees in publish/subscribe networks. The predicates in the subscriptions form an implicative relationship: if an object is rejected by the parent it will surely be rejected by the child. Taking advantage of this property of subscriptions in pub-sub systems our method constructs a multicast tree simultaneously laid over the network topology graph and the predicate graph. It eliminates the need for complex matching algorithms and data structures since only a single predicate is assigned to each filtering/routing service. To prevent attacks by intermediate nodes many of which may belong to non-trusted and possibly malicious administrative domains across the globe we combine hosts from non-trusted domains into trusted clusters and use these clusters as "supernodes" on the delivery tree. The proposed mechanism prevents insertion, deletion, modification, reordering, misdirection, and delaying of messages by malicious parties.

Having reduced the original problem to an instance of the Minimum Steiner Tree Problem (MSTP) we devised an efficient algorithm for constructing a nearly optimal delivery tree. Although MSTP is NP-hard, we showed that an efficient construction is, nevertheless, possible for the types of graphs generated by our reduction procedure. To improve performance of our basic algorithm, we proposed two heuristics: the Random Selection heuristic selecting a random logarithmic-size cluster subset from the set of all possible clusters and the Divide-and-Conquer heuristic solving the problem for subsets of domains and using as a candidate cluster set the Cartesian product of only those clusters that delivered optimal solutions for each of the subproblems.

We also developed an analytical technique to quickly estimate the cost of an optimal delivery tree; it can be used in resource-constrained networks to find the maximal subset of subscribers that can be supported without violating the network’s limitations.

Our simulation experiments focused on studying the efficiency of our enhancement heuristics described above. We compared our heuristics with each other and with the
Benchmark Algorithm. The experiments showed that the Divide-and-Conquer heuristic outperforms the other two algorithms when the number of non-trusted domains is less than 16.

Second, we studied aggregating sensor networks and developed mechanisms for protecting them from stealthy attacks where an attacker attempts to (a) mislead the system into believing in aggregation results it finds advantageous and (b) remain undetected. Our contributions in this area include the following:

- Development of CoDeX, a collect-detect-exclude framework for secure aggregation in sensor networks.
- Development of the "neighborhood watch" (distributed constraint validation) mechanism for secure aggregation in sensor networks.
- Development of the accompanying protocols for secure aggregation and network maintenance in sensors networks.
- Development of a server-based reputation subsystem and a probabilistically approximately correct (PAC) learning model-based subsystem for detection and removal of compromised sensors.
- Development of simulation models to study effectiveness and efficiency of the proposed mechanisms improving service assurance in aggregating sensor networks.

Since many physical phenomena exhibit strong spatial correlation between point measurements, our model exploits them by introducing measurement constraints. Sensors observe their neighbors during transmission and file "complaints" if their neighbors’ data violate the constraint when compared with their own.

Our collect-detect-exclude framework allows the server to collect information from the network, detect compromised sensors, and exclude them from the network by notifying other
sensors about the compromise. We introduced three protocols: Data Collection Protocol to supply actual data and evidential information to the server, Sensor Exclusion Protocol to enable the server to securely exclude compromised and isolated sensors from the network, and Network Abandonment Protocol to abandon the network in case of complete compromise.

To counter attacks on aggregators our solution forms a randomized delivery (and aggregation) tree while requiring that in each epoch sensors transmit information about $s > 1$ recent epochs. This allows the server to reconcile contradictory results (by voting) and to detect an attack when a fairly small fraction of sensors have been compromised.

To identify compromised sensors our server uses a beta reputation system [93] based on complaints from the network. The system also uses the probabilistically approximately correct (PAC) learning model [132] to help the server discern the adversary’s strategy and identify coordinated attacks involving multiple compromised sensors.

In addition we simulated sensor networks under several attack models. Our results show that increasing the frequency of sensors’ intercepting one another’s messages decreases the average time to detect a compromise. It also increases the chances of catching all compromised sensors. Our experiments also demonstrated that when the number of a sensor’s neighbors is 16 or higher, the time to detect a compromised sensor decreases with the increase in density. Preliminary results indicate this may be true for sparser networks as well.

Lastly, we studied protection methods for peer-to-peer massively multiplayer online games (MMOG). Cheating in online games is widespread, and in P2P games its impact will only increase since players’ computers themselves will become part of the game’s infrastructure. The scope of our effort in this project included:

- Development of a fraud-resistant architecture for peer-to-peer environments (FRAPPE)
based on the use of clustering, secure multiparty computations, threshold cryptography, and anonymity services.

- Development of a verification mechanism for local broadcast in P2P environments allowing network entities to solicit service participation via local broadcast at a remote location and verify that the remote peers honestly executed the protocol.

- Development of simulation models to study effectiveness and efficiency of the proposed mechanisms improving service assurance in massively multiplayer online games.

In this dissertation we introduced FRAPPE, a fraud-resistant architecture for peer-to-peer environments, that addresses the problem of cheating in MMOGs. Assuming that no more than half of all peers are cheaters, we demonstrated that trusted "supernodes" of reasonable size (5 to 11) can be constructed out of non-trusted peers; these supernodes can reliably and securely provide infrastructure services traditionally performed by server components.

We also identified the full set of general-purpose and security-related infrastructure services required to run a game. FRAPPE divides the game world into cells and assigns peer group-based *Cell Managers* to control all aspects of the cells' lifecycle. Cell Managers serve as the sole source of authority for the state of their respective cells: updates submitted by game clients remain tentative until reconciled and approved by the Cell Manager. A dynamic and flexible game world management architecture supports expanding and shrinking of the game world tree as required.

We showed that Proactive Secret Sharing schemes, threshold cryptography and secret redistribution schemes can be used to share sensitive attributes of avatars between non-trusted peers in a "supernode" and between supernodes, generate private keys, sign messages, etc. FRAPPE also pays special attention to avoidance of conflicts of interest: for example, a client whose avatar is located in some cell cannot be a member of the Cell Manager handling
Finally, we introduced a new primitive, called *local broadcast with verification* (LBV). It is used by the Load Tuning and Availability Service and by Cell Managers to invite peers in a particular neighborhood to participate in a peer group-based FRAPPE Service. In LBV a soliciting entity locates a ”seed” peer and sends it an invitation to join; the seed is responsible for broadcasting the invitation to all its neighbors, and they, to theirs, etc. All peers reached by the broadcast are expected to contact the designated recipient allowing the latter to select the most suitable peer subset. The seed or any other peer may cheat and forward invitations only to peers of their choice (for example, hoping to form a completely malicious instance of a Service). LBV attempts to prevent this. We evaluated this approach using a network simulator based on the *PeerSim* software package [158]. Our results showed that LBV works quite well, properly rejecting subverted candidate peer sets and accepting correctly constructed candidate peer sets even in the face of node and network link failures.

Our future work will focus on the following areas:

- Development of algorithms for secure delivery in Internet-scale content-based publish/subscribe networks with *dynamic* membership and *multiple* publishers.

- Identification of classes of applications, aggregation functions and generalized attack models in aggregating sensor networks to which CoDeX can be applied.

- Evaluation of, and, if required, development of enhancements to, the game world management infrastructure in FRAPPE.

- Implementation of the FRAPPE architecture in C# and development of a prototype massively multiplayer online game that takes advantage of this implementation.
Bibliography
Bibliography


Curriculum Vitae

I was born and raised in Odessa, Ukraine, part of the former Soviet Union. After graduating magna cum laude from Moscow Transportation Academy (Moscow, Russia) I started to work as a software engineer developing systems for computer-aided design. In 1992 my family and I settled in the United States. Here I continued in software design and development in the financial and software industries with emphasis on distributed systems, databases and security. In 2000 I received a Master of Science degree from George Mason University.

In 2002-2003 I authored TrapBlaster, a secure intelligent SNMP trap (event) router used by many companies and government organizations (Sprint, XO Communications, U.S. Federal Reserve Board, XM Satellite Radio, Sirius and others). Since 2004 I have been with Exostar, LLC, working as a software architect specializing in Public Key Infrastructure, Identity and Access Management, and Identity Federation. Most recently, I was a member of the Design Authority at Transglobal Secure Collaboration Program (TSCP), a consortium of leading aerospace and defense companies and defense agencies in North America and Europe, focusing on adoption of common policies, methodologies and technologies enabling its members to collaborate securely across multiple jurisdictions and policy domains.

I like to spend my spare time outdoors, hiking, biking and swimming.