UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Fakultät für Elektrotechnik, Informatik und Mathematik

Master's Thesis

Security Mechanism for Monitoring Peer-to-Peer Networks

Payman Alavi
Matriculation Number: 6662777
E-Mail: Payman@uni-paderborn.de

Paderborn, February 2015

Thesis Supervisors:
Jun.-Prof. Dr.-Ing. Kalman Graffi
Prof. Dr. Christian Scheideler

Declaration

(Translation from German)

I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Original Declaration Text in German:

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Paderborn, February 2015 Payman Alavi

Abstract

The peer-to-peer paradigm is a promising alternative to the classic centralized network model. In contrast to the centralized networks in which authorized nodes connect to and use resources on a single central computer, typically a server, each node connected to a peer-to-peer network often functions as both a client and a server simultaneously. Hence, peers may provide resources to other peers and also consume shared resources available from other nodes.

The lack of a centralized management system to capture the status and performance of peer-to-peer networks makes each node to be responsible for providing statistical information about itself to other nodes and also spread the received monitoring data from other nodes through the network. However, malicious nodes may disobey the monitoring mechanism by actions such as producing fake statistical data about themselves, manipulating the monitoring data received from other nodes or forwarding motoring data to peers other than the expected ones. Consequently, the malicious nodes can adversely affect the performance of the peer-to-peer networks.

In this thesis, we aim to analyze the impact of different types of malicious nodes on the monitoring mechanism provided by SkyEye.KOM, a decentralized and structured monitoring solution that functions by constructing a tree structure as a monitoring layer on top of a DHT-based peer-to-peer overlay. To illustrate, we define three types of attacks which cause different types of anomalies in that monitoring tree and propose counter-measures against each of our defined attacks. Afterward, we implement the three defined attacks and one of our counter-measure solutions by employing the PeerfactSim.KOM simulation framework. We then analyze and compare the effect of different types of malicious nodes while considering the impact of various parameter settings, such as the branching factor of the monitoring tree and the ratio of the present malicious nodes in the network. Lastly, we examine our implemented counter-measure and assess the extent to which our solution is able to neutralize the effect of malicious nodes.

Contents

1 Introduction

Contents

The Internet is one of the most popular and influencing technologies of the century and many technologies have been introduced based on the power provided by this platform. Since the advent of the Internet, sharing the resources has always been the aim of many Internet users and for that purpose different solutions and paradigms have been introduced and continuously improved until now. One of the earliest solutions for sharing data and computational power is the centralized network paradigm, also known as client/server model, in which powerful servers provide different resources or services and authorized clients have access to the shared resources on the servers. The client/server paradigm is still widely used because of the ease of implementation and availability of the effective network management mechanisms which has been improved over years. However, the high cost of servers' maintenance/management and the potential problem of servers as a bottleneck in the network made computer scientists think of new paradigms besides the centralized networks.

Based on Moore's Law [23], the number of transistors, which can be fitted on a chip, doubles about every two years and according to Glider's law [11], "bandwidth grows at least three times faster than computer power". Consequently, the computation power of the users' devices connected to the Internet and the amount of data, which can be transferred over the Internet in a certain amount of time, is increasing constantly. Currently, even some Personal Computers (PCs) are powerful enough to act as a service provider to some other devices in the network and the number of such powerful nodes connected to the Internet is increasing every day. Thus, why not use this enormous computational power

and the available enhanced Internet bandwidth for providing users the services they require with the help of the user's resources themselves?

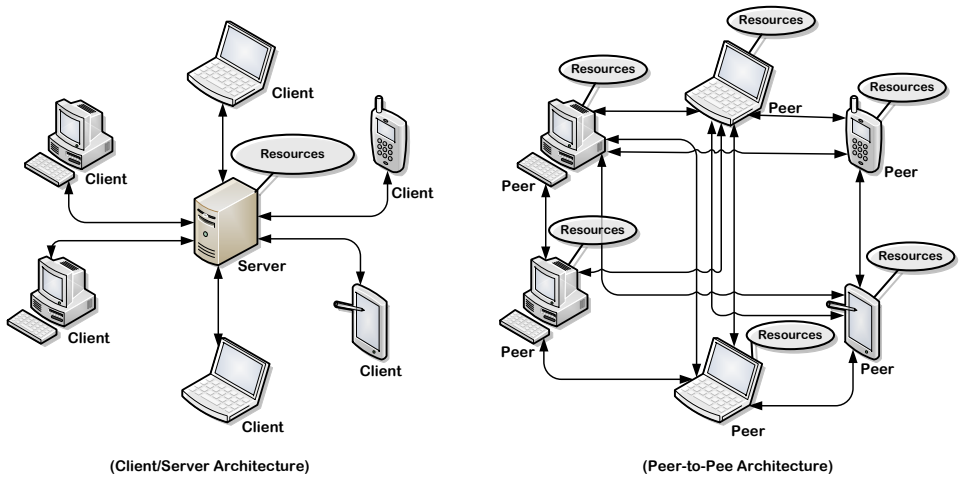


Figure 1.1: A Simplified Overview of Client/Server and Peer-to-Peer Architecture

Peer-to-peer (p2p) paradigm is an answer to the question posed above. In p2p networks, each device connected to the network (node) may provide services to other nodes or consume the resources provided by others. In other words, each node acts as a client and a server at the same time and instead of communication via servers, as in centralized networks, peers communicate with each other directly. Figure 1.1 depicts the p2p architecture in comparison to the client/server paradigm.

Since the advent of Napster [6] in 1999 as the first successful p2p application, millions of Internet users have been using p2p applications to share their files with each other directly without the need to upload them to servers in advance. The success of the p2p paradigm could be observed by having a look at the Internet traffic statistics of the popular p2p applications such as BitTorrent [1] and Skype [3] over the last decade. According to the study by Washington university published in 2003 [14], p2p applications account for 60 to 80 percent of the total Internet traffic.

However, although the p2p paradigm is quite promising, the recent researches reveal a decrease in the share of Internet traffic by p2p applications in the recent years [15]. We believe that one of the reasons for this decline is the challenges in the way service providers offer a controlled level of the Quality of Service (QoS) for users and provide a reliable platform for software developers over the p2p networks. One of the most important elements of providing a controlled level of QoS is an accurate and effective monitoring mechanism.

Besides challenges in monitoring p2p networks, the possibility of malicious behavior by peers make the situation even more complicated. The lack of a centralized management system for monitoring the status of the network makes each peer to be responsible for providing statistics about itself to other peers and also spread the received monitoring information from other nodes through the network. Providing wrong monitoring information, manipulating the monitoring data received from other nodes or forwarding the statistical information to peers other than the expected ones are examples of the possible malicious behavior which can damage the monitoring mechanism and worsen the performance of p2p networks.

In the remainder of this chapter, we discuss the motivation and problem statement for the thesis in Section 1.1 and our goals in Section 1.2. Finally, Section 1.3 briefly presents the organization and outline of this document.

1.1 Motivation

The quality of service (QoS) is defined by Schmitt [28] as "the well-defined and controllable behavior of a system with respect to quantitative parameters". Based on this definition, the better monitoring mechanism for p2p network, the better control over the network behavior and consequently moving towards the QoS. As an example, in case of detecting a long response time based on the monitoring information, nodes can increase their routing table size by keeping the information for more peers and, as a result, the hop count to reach a peer will be decreased and the response time will be improved. By solving current problems in the way of monitoring p2p networks and providing software developers an infrastructure with high and constant quality of service, we can expect more commercial applications based on p2p platform.

Attackers in p2p networks with no counter attack strategy are able to cause serious side-effects on monitoring results. Between the different p2p monitoring strategies, which are discussed in Chapter 3, our focus in this thesis is the possible attacks on structured monitoring mechanism. In structured monitoring mechanism, malicious nodes can disobey the monitoring system by actions such as producing false monitoring data, manipulating the received monitoring information from other nodes, preventing the propagation of monitoring data or forwarding the data to non-responsible peers. Attackers can become even more harmful by initiating a combination of different attacks at the same time, or by cooperating with other malicious nodes in the network.

Hence, this master thesis aims to simulate and measure the side-effects of different attacks on structured monitoring mechanism. In addition, we offer our

solutions to detect and neutralize malicious nodes in order to move the p2p paradigm one step ahead in the direction of achieving QoS.

1.2 Overview on Goals

In this section, we define and summarize our main goals in this thesis.

1. Defining Attacks: Our first goal is to define some possible malicious behaviors which can cause problems for structured monitoring mechanism. To design the attacks, our focus is the monitoring mechanism provided by SkyEye.Kom [13]. The SkyEye.Kom monitoring mechanism is explained in Section 3.1.1.

2. Proposing Security Solution for the Defined Attacks: After defining the attacks, we look for solutions for neutralizing the effect of our previously defined attacks. The solutions should help the structured monitoring mechanism, provided by SkyEye.Kom, to produce correct statistical information with an acceptable level of accuracy in the presence of the malicious nodes.

3. Implementing Our Attacks and Security Solution: Implementing the defined attacks is our next goal. In addition, considering the limited time, we also implement one of our proposed security solutions. Our work is based on the recent implementation of SkyEye.Kom developed by Giesen [10] on top of the PeerfactSim.KOM simulation framework [9]. We introduce and review the most important features of PeerfactSim.Kom simulator in section 2.2.

Our implementation priority is to implement the attacks, along with our security solution, with a systematic and clear structure which can also be employed for the implementation of more advanced attacks and counter-attacks in the future. We also provide an easy way for configuring the type of attacks, ratio of malicious nodes, ratio of secured nodes and other related parameters for simulating various security scenarios.

4. Evaluating the Attacks: The implementation of the attacks enables us to simulate networks with different configurations in the presence of various types and ratios of malicious nodes. The simulations are performed in the rich simulation environment provided by PeerfactSim.Kom simulator. We evaluate the effect of our attacks by analyzing the simulation results.

5. Evaluating the Effectiveness of Our Security Solution: Our last goal is to evaluate our implemented security solution by the help of the Peerfact-Sim.Kom simulator to observe the effectiveness of the solution. We measure the effectiveness of our solution by simulating networks with and without the security counter-measure and comparing the simulation results.

1.3 Outline

In this chapter we introduced the importance and challenges of monitoring mechanisms for providing a controlled level of the Quality of Service in p2p systems. We discussed that how malicious nodes can create problems for structured monitoring mechanisms in p2p networks and presented our goals and motivations for solving the security issues. The rest of this thesis is structured as follows:

Chapter 2 gives an introduction to structured and unstructured peer-to-peer networks. It also introduces the PeerfactSim.KOM simulation framework and shortly its functional layers. Chapter 3 introduces structured and unstructured monitoring approaches and provides more details about SkyEye.KOM as an example of structured monitoring mechanism. In Chapter 4 we define our monitoring attacks and propose our solutions against them. In Chapter 5 we provide details of our implementation for the attacks and the counter-measures as defined in the previous chapter. In Chapter 6, after an introduction to our simulations goals, method and setups we present our simulation results and evaluate the attacks and our security solution by analyzing the presented results. Finally, we conclude our thesis in Chapter 7 and give an outlook on future work.

2 Background

Contents

This chapter provides an introduction to the concept of p2p networks in Section 2.1. Section 2.1.1, introduces the general mechanism of structured p2p overlays. Furthermore, in this section, the Distributed Hash Table, which is an important concept in structured overlays, is explained and the procedures of join/leave of nodes, in addition to the reaction of structured DHT-based overlays to nodes failure, are reviewed. Later on, after a brief explanation of unstructured p2p overlays in Section 2.1.2, we introduce the PeerfactSim.KOM simulator, which has been used in this thesis for simulation and evaluation purposes, in Section 2.2. We close this chapter with a short summary in Section 2.3.

2.1 Peer-to-peer Networks

Similar to other distributed systems, p2p networks follow the common goals of sharing resources such as information, software and hardware in order to reduce costs, enhance the availability and reliability of resources, and finally load balancing.

However, unlike the traditional centralized network model in which servers are responsible for providing almost all resources, in p2p networks resources are provided voluntarily by the users of the network. Moreover, instead of

management of the whole network communications via servers, peers in p2p networks communicate and interact directly with each other without any server intervention.

Since in p2p systems resources are widely spread over the network, the mechanisms for finding required data in the network are different from the centralized networks. Additionally, unlike servers which are supposed to be online permanently, nodes in p2p networks are connected to the network for a limited amount of time and join/leave the network autonomously. This behavior, known as *churn*, is a norm, rather than an exception, and cause the p2p network behavior to become highly unpredictable.

In p2p networks, peers have mostly similar rights and roles in the network and roles are distributed between nodes based on their capabilities. Strictly speaking, in homogeneous overlays like Chord [29], Kademlia [21] and Gnutella 0.4 [12] peers have identical rights and roles. On the other hand, in overlays with heterogeneity support such as Gnutella 0.6 [18] and Omicron [8], nodes are assigned different roles based on their capabilities. These characteristics made the management of the p2p networks different from the centralized networks.

The term overlay is often used for p2p networks since it is always built on top of an existing network, the so-called *underlay*, and the overlay adds an extra layer of abstraction, new potential and functionality to the underlay. In this way, the overlay makes use of the existing underlay functionality and modifies the underlay functions for its own use. A common distinction is to divide p2p overlays into structured and unstructured categories. These overlay types are explained in the following sub-sections.

2.1.1 Structured P2P Overlay

In structured p2p overlays, each node is assigned a unique ID, and peers maintain a routing table for sending messages to other nodes in the network. Most structured p2p overlays, e.g., Pastry [27], Kademlia and Chord, use Distributed Hash Table (DHT) for routing. In DHT-based overlays, identifier space is independent of the physical location of the peers and routing of messages is based on the nodes' IDs. All objects in structured overlays, including resources and peers, are identified by IDs which are commonly calculated by hashing the object's content and lies in the identifier space. However, there are some overlays which use methods other than DHT to accomplish the routing task. For example, Globase.Kom [19] (Geographical LOcation-BAsed SEarch) uses the geographical location of nodes in order to find the closest peers to a specific

location or all the peers in a given area.

In structured p2p overlays, objects and peers share a same ID space and the ID of each resource is assigned to a single peer in the overlay. Therefore, since each object can be identified uniquely the main functionality is to look up the object which is identified by a given ID. To make it more clear, we will review the Distributed Hash Table in more detail in the remainder of this section.

Distributed Hash Table (DHT)

Hash Table is a well-known data structure in computer science. It was invented in 1953 [22], and it is available in most major programming languages. Hash Table maps values to keys and one can look up an indexed value, fast and efficiently, by knowing the key.

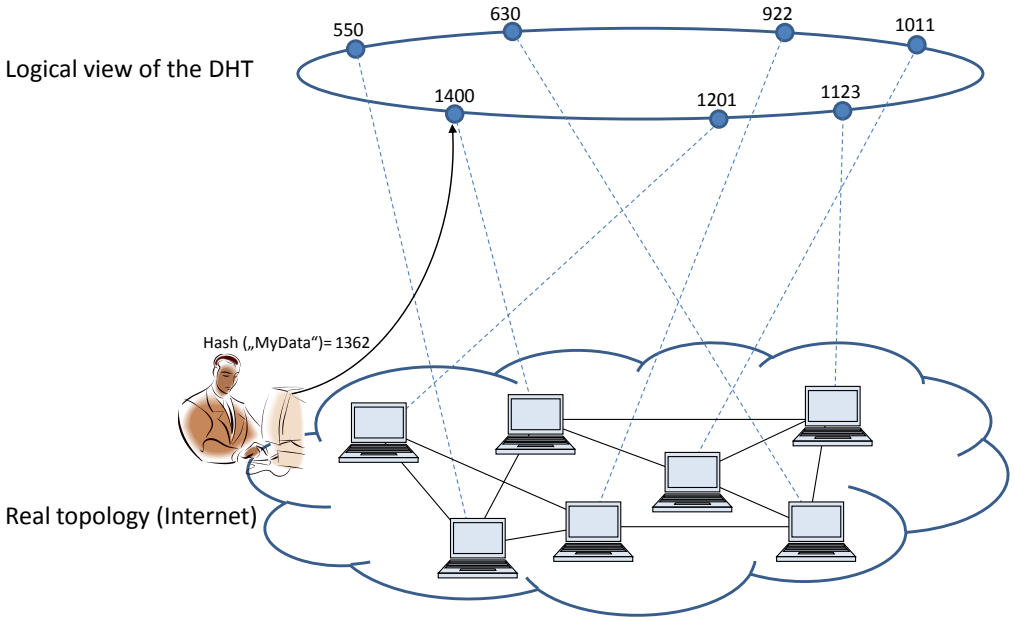


Figure 2.1: Overview on the Distributed Hash Table

Distributed Hash Table (DHT), which in comparison with Hash Table is relatively a new idea, is still based on the concept of key/value pairs in Hash Table with the difference that it is distributed between the nodes in a network. DHT is used in most well-known structured p2p overlays such as Chord, Kademlia and Pastry. In many p2p distributed systems, DHT is utilized for implementing a scalable and efficient key-based routing mechanism for looking up the responsible node for a given ID. DHT, as a routing table, is distributed between all the peers in the overlay, and each peer maintains a small part of it, including a certain number of references to some other nodes. As mentioned earlier, nodes

maintaining the DHT are identified with IDs from the same identifier space as data items.

There are various approaches for employing DHT as a routing mechanism. However, generally the idea is that each node provides a narrow view of overlay by maintaining a limited number of links to other nodes. In case a node receives a request for an object ID, which it is not responsible for itself, the node forwards the request to the closest ID it knows based on the stored links in its routing table. The forwarding process is repeated until the responsible ID is found. The result of the look up can be the address of the node that contains the looked up data item or the resource ID itself. Based on the strategy used in overlay, a node which shares a data item, can store the address of the resource or the resource itself on the responsible peer for the object's ID. Figure 2.1 shows an overview of the DHT and a peer which stores its hashed data item directly on the responsible peer with the ID of 1400.

If the resource is available in the overlay, the Distributed Hash Table guarantees that it is found. The DHT is scalable and produces a small amount of overhead even in large networks with millions of nodes because each node coordinates only with a few other nodes in the overlay and in case of challenges like a new node joining or leaving, and failure of peers, only a few nodes will be affected. To continue we explain the reaction of the DHT-based overlays to the challenges of nodes joining and leaving as well as failure of nodes.

Join of a New Node

When a node attempts to join a structured DHT-based overlay, it firstly calculates its own ID. Afterward, the new node needs to contact an arbitrary peer in the overlay as an entry point to the DHT. Depending on the implementation of the DHT, the newly arrived node is positioned in the overlay and a number of nodes update their routing information in order to consider the new node in routing system.

At this point, the new node takes the responsibility for a part of the DHT by reducing the responsibility range of its neighbor in the overlay. The reduction of the responsibility range is performed when the neighboring node copies part of the key/value pairs from its own Hash Table to the Hash Table of the new node. The copying process is usually carried out with a configurable amount of redundancy that can help the system in case of peers' failure. After this step, the new node becomes responsible for the granted range of key/value pairs in the DHT.

Besides the nodes which join the network during the time, some other nodes leave the network and there are also nodes which lose their connection to the

network without any prior notification. The procedure for leaving, and the reaction of nodes in DHT-based overlays to failure of a node, are briefly explained in the following paragraph.

Leaving and/or Failure of a Node

When a node attempts to leave the overlay, it copies its range of key/value pairs to the routing table of the linked neighboring nodes and the linked neighboring nodes would then remove this node from their routing tables.

In case of failure, a node loses its connection to an overlay without any prior notification. Therefore, there is no time for delegating responsibilities to other nodes. Generally, in this situation, the redundant key/value pairs become helpful to cope with the loss of data, whereas, without an effective redundancy strategy, failure of nodes in DHT results in data loss. In addition, other nodes use alternative routing paths when they notice the absence of the failed node.

In this section, we introduced structured p2p overlays and Distributed Hash Table in more detail since it is an important part of many of the well-known structured overlays. To follow on, we introduce unstructured p2p overlays as the other major category of p2p networks.

2.1.2 Unstructured P2P Overlay

An unstructured overlay is "an overlay in which a node relies only on its adjacent nodes for delivery of messages to other nodes in the overlay." [5]. Unlike the structured overlays, unstructured P2P systems do not maintain any extra structure, and resources are not assigned to peers but are hosted directly by the owner of objects. Therefore, the main functionality of unstructured overlays is to search and find peers which provide the desired objects.

In unstructured overlays, the ID spaces for peers and objects are separated and objects can be retrieved by searching keywords that match the description of the object. Hence, in contrast to structured overlays, finding the desired objects in unstructured overlays does not require knowing the objects' ID. Although unstructured overlays provide more freedom in implementing complex queries but searches are not efficient.

Unstructured p2p overlays can be categorized mainly in three categories of centralized like Napster, Homogeneous like Gnutella 0.4 and finally Heterogeneous networks such as Gnutella 0.6 and FastTrack [2]. Considering that unstructured overlays are not the focus of this thesis, we will not be further explaining the specifications of these categories.

In addition to the introductory knowledge to p2p overlays provided in this chapter, by way of important background information to this thesis, we introduce PeerfactSim.KOM, the simulator that we will use for this thesis in the next section.

2.2 PeerfactSim.KOM Simulation Framework

PeerfactSim.KOM [9] is an event-based simulation framework, written in Java for simulating large scale distributed/p2p systems, with focus on the evaluation of inter-dependencies in multi-layered p2p systems.

PeerfactSim.KOM has been developed at the Technical University of Darmstadt in 2006 and since then has been improved and maintained at the University of Paderborn and the University of Düsseldorf.

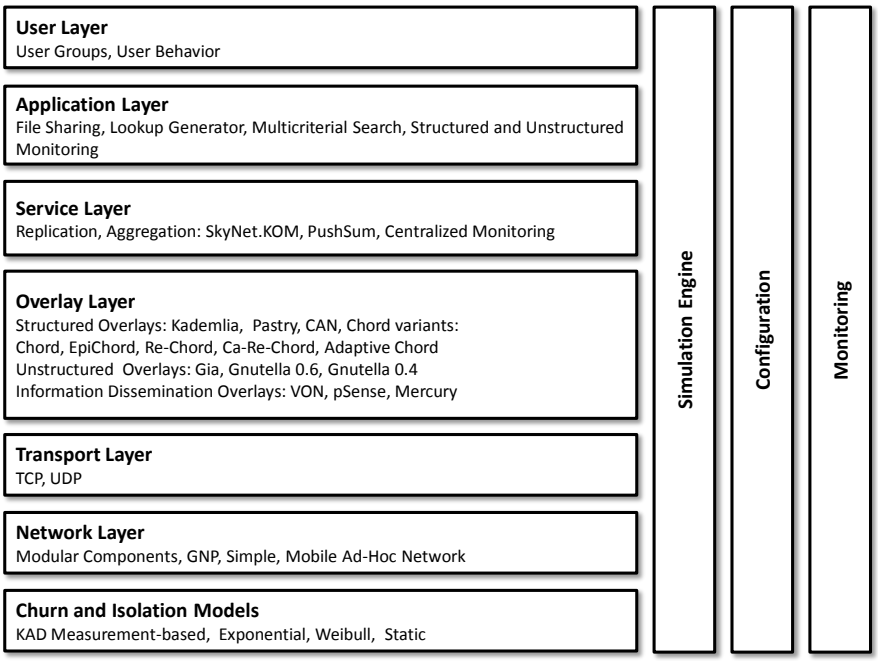


Figure 2.2: Overview on the Functional Layers of PeerfactSim.KOM

The simulator follows a layered architecture and each layer operates independently of the others. Therefore, each layer can be modified or exchanged easily

with slight modifications. The layered architecture of PeerfactSim.KOM is represented in Figure 2.2 and a brief introductory knowledge about each of the layers is provided in the following.

User Layer: The user layer provides facilities for capturing user's actions together with creating user groups with different roles.

Application Layer: Functions such as file sharing or search application, with custom searches based on keywords, are the examples of the application layer. The recent extensions of the structured and the unstructured monitoring are implemented in this layer as well.

Service Layer: The service layer consists of some enhancement protocols to improve the management and control mechanisms which are neither part of the application layer nor the p2p overlay. The SkyNet.KOM monitoring component is located in this layer.

Overlay Layer: In the overlay layer, different structured overlays such as Can, Chord, Kademlia and Pastry, unstructured overlays like Gnutella 0.4, Gnutella 0.6 and Gia and information dissemination overlays such as VON, pSense, Mercury are implemented. This layer operates on top of the transport and network layer.

Transport Layer: Besides providing methods for serialization, the transport layer covers the implementation of TCP [7] for large messages and UDP [26] for small messages which can be used in combination with the network layer to obtain more realistic values for throughput, delay, jitter and packet loss.

Network Layer: The network layer is responsible for modeling delay, bandwidth, jitter and packet loss. It covers different network strategies such as *simple network* with limited configurable parameters, *modular network* with design goal of extendibility, *mobile* and finally *Global Network Positioning* (GNP) [24] which provides a simulated network with many more configurable details than the simple network. Examples of these options are Round-Trip Time (RTT), jitter, geographical position and the real Internet packet loss.

Churn Models Layer: The Churn layer, models an unexpected online/offline behavior for nodes in the overlay. The currently available churn models are KAD, Static, Exponential and Weibull. Currently available churn models are implemented based on KAD measurements and weibull distribution. The *static churn* model and the popular *exponential churn* behavior are also among the available churn patterns.

Additionally, the simulator provides several interfaces like NetAnalyzer, TransAnalyzer, OperationAnalyzer, ConnectivityAnalyzer, ChurnAnalyzer and KBROverlayAnalyzer, for analyzing different aspects of simulations. The analyzers can be used for calculating network layer statistics such as the number of sent/received messages, bandwidth consumption, the number of hosts connected to the network and the number of hosts affected by churn.

In this section we briefly introduced PeerfactSim.KOM and each of its functional layers. A short summary to this chapter is provided in the next section.

2.3 Summary

In this chapter, we studied p2p networks and compared the characteristics of p2p and client/server networks. Following the common categorization of p2p overlays, we discussed the major features of structured and unstructured types. Furthermore, we reviewed the Distributed Hash Table and the characteristics of DHT-based p2p overlays since most of the structured overlays use the DHT for routing purposes. Finally, we introduced PeerfactSim.KOM, the simulator that we use in this thesis for simulating and evaluating our monitoring attacks and counter-attack.

3 Monitoring

Contents

The following chapter reviews the fundamental mechanisms of monitoring p2p networks. Following an introduction to the importance of monitoring systems in Section 3.1, we further divide p2p monitoring approaches into two separate categories, entitled structured and unstructured respectively. We introduce the common characteristics of structured monitoring systems in Section 3.1.1, and study the monitoring mechanism provided by SkyEye.KOM, as an example of structured monitoring solution in Section 3.1.2. Section 3.1.3, introduces the idea of unstructured monitoring systems and describes the gossiping approach for aggregating monitoring data in networks based on the Push-Sum algorithm. Finally, we summarize this chapter in Section 3.2.

3.1 Monitoring of Peer-to-peer Networks

Monitoring mechanisms aim to facilitate the control and management of networks by providing useful statistical information regarding network status at any time. Examples of the valuable statistical information in p2p networks include the number of peers in the network, peers average online time, routing delay, number of hop count to reach resources and network traffic.

An effective monitoring mechanism will enable p2p networks to achieve a controlled level of Quality of Service. However, there exist challenges in the design

process of a sound monitoring system for p2p overlays due to characteristics of p2p networks. These comprise churn, peers heterogeneity in capacity and connectivity, unpredictability of network size and peers' behavior.

By providing fresh statistical information, monitoring mechanisms yield a global view on networks which can be used for optimization decisions such as nodes stabilization frequency, or peers time-to-live factor. The global view additionally reveals the overhead and routing speed of p2p networks, which assist with regards to the on-the-fly automatic performance adaptation of overlays. Since redeployment of p2p networks is impossible, on-the-fly adaptation is vital for modifying the monitoring mechanisms of p2p networks. In addition, the provided statistical information contributes towards designing improved overlay mechanisms in future releases.

Akin to p2p overlays, monitoring mechanisms of p2p networks can likewise be grouped into structured and unstructured types. Whilst in unstructured monitoring mechanisms nodes distribute monitoring to other random nodes, structured monitoring systems introduce an extra overlay on top of p2p overlay which is fully specified for monitoring tasks. Additionally, nodes distribute monitoring information based on the strategy, defined by the monitoring overlay, only to specific nodes. The rest of this chapter provides more information about structured and unstructured monitoring mechanisms.

3.1.1 Structured Monitoring Systems

Structured monitoring mechanisms construct a so-called over-overlay on top of p2p overlays. Peers in p2p overlay participate in the construction of structured monitoring overlay, and each node determines its position in the monitoring structure itself. One of the structures, commonly used in structured monitoring systems, is the tree structure.

Usually, in addition to the ID that is used in p2p overlay, each peer also maintains a different ID for participating in the monitoring over-overlay. Typically, monitoring IDs are the result of some simple calculations in order to facilitate an easy translation of monitoring IDs to overlay IDs and vice versa. The monitoring ID space is used exclusively by nodes partaking in the monitoring overlay for identification of communication partners with respect to exchanging monitoring data. In order to look up the communication partners, nodes are dependent on the look up facilities delivered by p2p overlay.

Following the introduction above with regards to the general specifications of structured monitoring systems, in the next section, we study structured monitoring in more detail by concentrating on SkyEye.KOM, as a well-constructed

example of structured monitoring mechanism, introduced by Graffi et al.[13]. The SkyEye.KOM monitoring mechanism is also the base of our further implementations in this thesis.

3.1.2 SkyEye.KOM Monitoring Mechanism

SkyEye.KOM is a monitoring mechanism that operates by constructing a tree structure as an over-overlay on top of structured DHT-based p2p overlays. In SkyEye.KOM, each node determines its location within the monitoring tree structure independently and matches its overlay ID with an ID from the monitoring ID space as it joins the overlay. The proposed ID space in SkyEye.KOM is a unified space $S_{ID} \subseteq \mathbb{R}$ ranges from 0 to 1, and it is calculated by dividing the overlay ID to the size of the overlay ID space. Leave, join or failure of a node results in a partial or complete reconstruction of the monitoring tree. After joining the tree, each node becomes responsible for a part of the monitoring ID space known as domains. The responsible node for each domain is called domain coordinator.

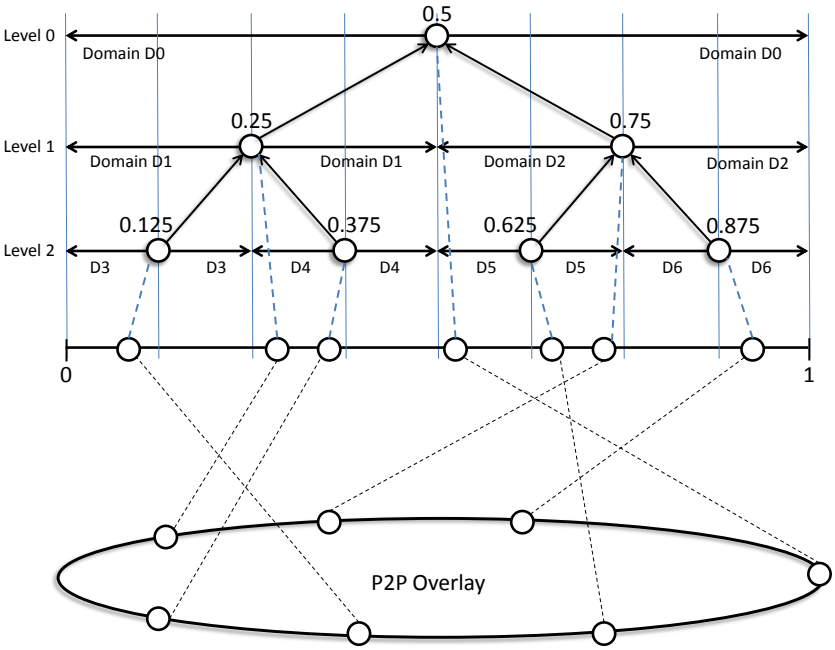


Figure 3.1: SkyEye.KOM as an over-Overlay on Top of the p2p Overlay

Figure 3.1 visualized both the structure of SkyEye.KOM tree and its relation to the p2p overlay. The first domain level, level 0, is the largest domain and

covers the whole ID space, $[0, 1]$, with the root of the tree as the Coordinator. Thus, the first node joining the overlay becomes the root of the tree, and the Coordinator of domain D_0 with tree ID of 0.5. According to the branching factor value, domains are recursively divided into sub-domains. In each domain, the node in the p2p overlay which is responsible for the tree ID in the middle of the domain becomes the Coordinator of the domain. A Coordinator maintains the information of all the peers, which hold their IDs within its Domain. In this way, when $bF = 2$ as an example, the next node which joins the tree after the root node becomes the Coordinator of either sub-domain $[0, 0.5]$ or $[0.5, 1]$.

Generally, data distribution between peers is achieved based on either pull or push mechanism [16]. In a pull-based method, each node sends data only when it receives a request from another node. In contrast, in push-based mechanisms, nodes propagate data periodically without being asked. The current implementation of SkyEye.KOM is based on push method which introduces less complexity and overhead with reference to the management of monitoring overlay.

In SkyEye.KOM, monitoring data is classified into three types, namely, *local metric*, *children aggregation* and *global estimation*. The local metric is the monitoring information each node generates in relation to its own status. The children aggregation is the aggregated monitoring data from all the children of a node. This is refreshed whenever a parent node receives new monitoring information from one of its children. Last but not least, the global estimation is the aggregation result of local metrics and children aggregated monitoring data. The global estimation, aggregated at the root node, represents the status of the whole monitoring tree.

Nodes in the tree periodically forward their own monitoring statistics, aggregated with received data from their children onto their parents in each update interval. The root node aggregates the received monitoring information from its children with its local data, and sets the result as the network global monitoring estimation. Additionally, it sends an acknowledgment message (ACK) containing the global monitoring estimation to its children. Figure 3.2 depicts a simplified overview of data flow in SkyEye.KOM. After receiving ACK from the parent, the receiver applies the received global estimation and further forwards it, included in an ACK, to their own children whenever it receives fresh monitoring data from one of its child nodes. Subsequently, global estimation is pushed through the tree, and all nodes become informed of the current state of the network.

Figure 3.3 shows an example of aggregating the maximum value at all nodes in the SkyEye.KOM monitoring tree. As it is depicted, each node measures its own local maximum value, and aggregates it with its children data before

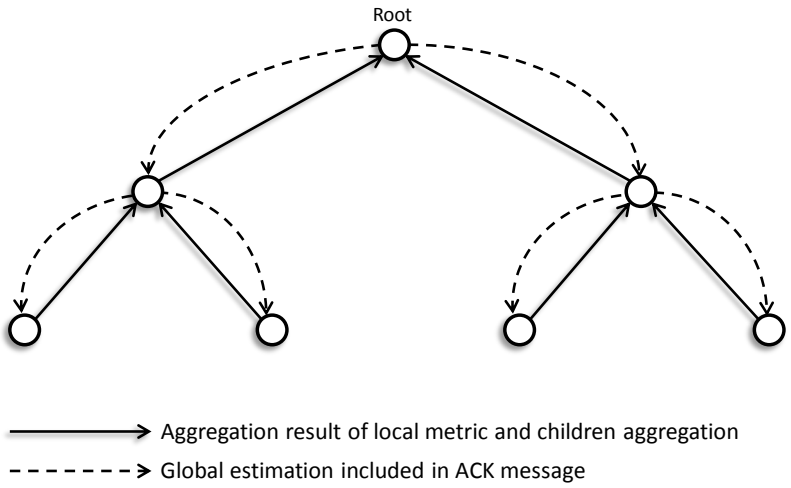


Figure 3.2: Simplified Overview of the SkyEye.KOM Data Flow

forwarding the result of the aggregation to its parent node.

Leaf nodes have no children and, as such, they only forward their own local metrics to their parents. Parents of leaf nodes compare received local data from their children against their own local data, and forward the maximum value as the result of maximum aggregation to their own parents periodically. The result of aggregation at the root node represents the maximum value of the tree which is 0.88 in our example.

In this section, we studied SkyEye.KOM as an example of a structured monitoring mechanism. We reviewed the mechanism of data flow and the different monitoring data types which are defined in SkyEye.KOM. In conclusion, to clarify the aggregation function, we provided an example in which the maximum values at all nodes in the tree were aggregated. The next section is dedicated to unstructured monitoring systems as the other main category of p2p monitoring systems besides structured mechanisms.

3.1.3 Unstructured Monitoring Systems

In unstructured monitoring mechanisms, there does not exist a separate overlay for monitoring. Nodes exchange monitoring data with their random neighbors, or in some approaches with arbitrary nodes in p2p overlay. When nodes receive monitoring information from other nodes, they aggregate the received data with their own local metric, and thereafter forward the result to some

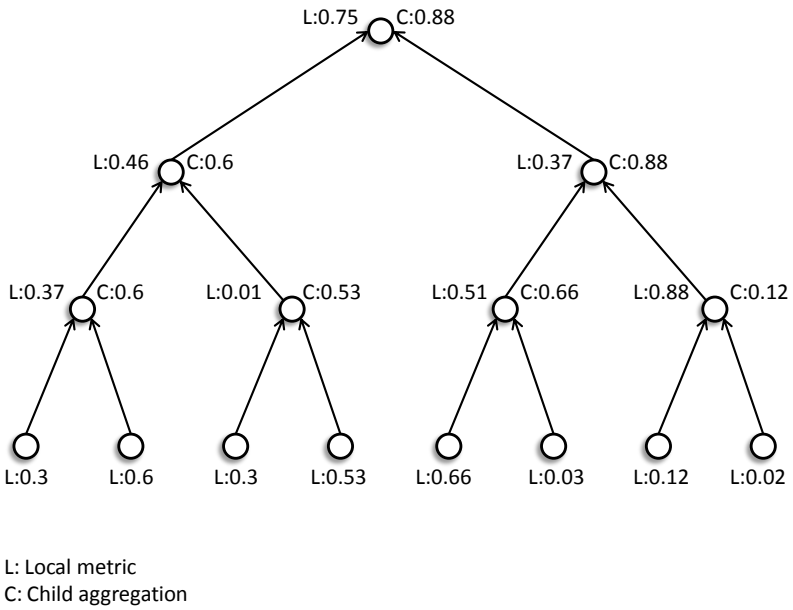


Figure 3.3: Aggregating the Maximum Value at All Nodes in SkyEye.KOM

other nodes. This method is also known as gossip-based monitoring strategy.

Gossip protocols are round-based, and a certain number of rounds is called an *epoch*. At the beginning of each epoch, nodes publish updated local metrics to guarantee the convergence of values at all nodes to an updated value representing the actual state of the network. One of the famous implementations of gossip-based monitoring is based on the gossip algorithm by Kempe et al. [17], named *Symmetric Push-Sum*, for computing global aggregation. The Symmetric Push-Sum protocol can be used as an example for calculating the sum or the average of values at all nodes in a network within a predictable number of rounds. The pseudo-code of the Symmetric Push-Sum Protocol is presented in Algorithm 1.

According to the Symmetric Push-Sum algorithm, in each round, node i selects a random node, node j, as its communication partner (line 2). Afterward, node i halved its local value, v_i, and its weight, w_i, and sends the resulted values as an *aggregation message* to node j. Node j, as the receiver of the aggregation message, also halved its own local value and weight and sends them to node i (line 6, 7). In addition, it adds the received local value and weight respectively to its own local value and weight. By initiating v_i and w_i according to the values in Table 3.1, different aggregation functions such as sum, count, average or weighted average of values can be achieved.

Algorithm 1 Symmetric Push-Sum Algorithm

At each node i
Require: v_0, w_0
The initial local value, v_0;
The initial local weight, w_0.
Initialization:
1: $(v, w) = (v_0, w_0)$
At each cycle:
2: $j \leftarrow$ *getNode*()
3: $v \leftarrow v/2, w \leftarrow w/2$
4: send an *aggregation message* to $j, \langle (v, w), \textbf{true} \rangle$
At event: received an *aggregation message* $\langle (v', w'), r \rangle$ from j
5: **if** r is **true then**
6: $v \leftarrow v/2, w \leftarrow w/2$
7: send an *aggregation message* to $j, \langle (v, w), \textbf{false} \rangle$
8: **end if**
9: $v \leftarrow v + v', w \leftarrow w + w'$

Function	Description
Sum	$v_i =$ local value $w_i = 1$ at a single node, 0 at all other nodes
Count	$v_i = 1$ $w_i = 1$ at a single node, 0 at all other nodes
Average	$v_i =$ local value $w_i = 1$
Weighted Average	$v_i =$ local value $\times$ local weight $w_i =$ local weight

Table 3.1: Symmetric Push-Sum Configurations for Different Aggregation Functions

Figure 3.4 depicted an overview of calculating the average value at all nodes in a network using Symmetric Push-Sum algorithm. As it is shown, the values at all nodes converge to the correct average value after each round until the values at all nodes become identical. Requiring no extra overlay in unstructured monitoring mechanisms, introduces less overhead than structured mechanisms and make them flexible and robust against churn in p2p networks.

On the other hand, structured monitoring mechanisms offer more accurate monitoring results at a higher overhead cost. Furthermore, they are not as flexible and churn resistant as unstructured monitoring mechanisms.

In this section, we introduced unstructured monitoring systems and specifically gossip-based monitoring mechanisms. We clarified that monitoring data

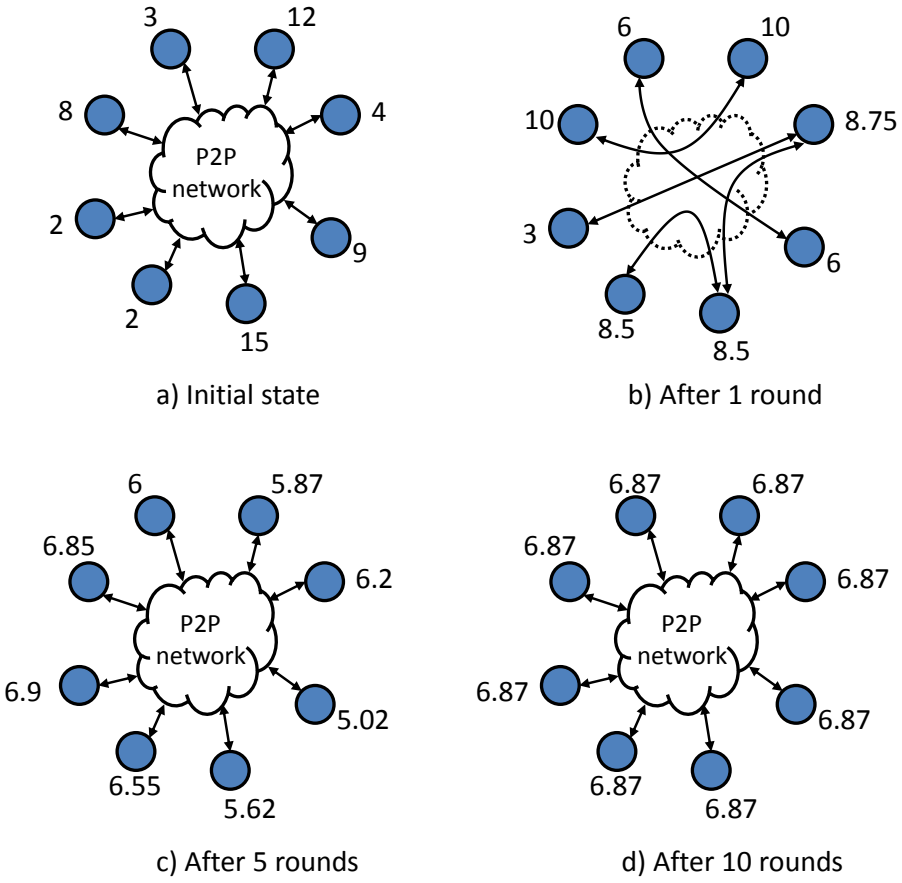


Figure 3.4: Overview on Aggregating the Average Value Using the Push-Sum Mechanism

is aggregated in gossiping approaches, although there is no specific node for aggregating global monitoring information comparable to the root node in the SkyEye.KOM monitoring overlay. Similarly, we reviewed the Symmetric Push-Sum algorithm commonly utilized in gossiping approaches for monitoring p2p networks. The next section provides a summary to this chapter.

3.2 Summary

In this Chapter we introduced the general idea of structured and unstructured monitoring mechanisms, and explored them in greater details by examining an approach of each type as an instance. Considering that our proposed monitoring security attacks and solutions in the subsequent chapter are based on SkyEye.KOM, we reviewed the SkyEye.KOM monitoring algorithm as an ex-

ample of structured monitoring mechanism in more detail. Furthermore, we explained the data flow in the tree structure of SkyEye.KOM monitoring overlay and considered the aggregation mechanism of monitoring information by illustrating an example in which the global maximum value in the tree were aggregating. Moreover, we highlighted the important role of the root node in SkyEye.KOM as the center that aggregates and propagates network global estimation representing the status of the whole network.

Subsequently, we introduced unstructured monitoring systems and reviewed gossip-based monitoring mechanisms as a significant unstructured method for aggregating data in p2p overlays. In addition, we studied Symmetric Push-Sum algorithm, as an effective algorithm used in many unstructured gossip-based monitoring systems. Finally, we compared structured and unstructured monitoring systems and evaluated the advantages and disadvantages of each mechanism in comparison to each other.

4 Monitoring Attacks and Solutions

Contents

In this Chapter, we define our monitoring attacks and counter-attacks. All of the attacks and security solutions in this chapter are designed based on the monitoring mechanism provided by SkyEye.KOM. In Section 4.1, we define FalseLocalData attack in which malicious nodes generate fake monitoring statistics about themselves. Afterward, in Section 4.1.1 we introduce our scoring system as a solution to detect FalseLocalData attackers. In addition, we suggest various enhancement strategies for the monitoring data flow in SkyEye.KOM to improve the accuracy of the scoring system. We also propose different strategies to react to malicious nodes after the detection phase. Similarly, Section 4.2 describes our definition of FalseChildData malicious behavior and Section 4.2.1 explains our counter-attack strategy against FalseChildData attack. The FalseChildData attackers manipulate monitoring information received from their children before aggregating and further forwarding them to their parents. In Section 4.3, we define our last attack, named FalseParentPeer, which makes the attackers forward monitoring data to a wrong destination, instead of the expected node, and in Section 4.3.1 we explain our idea to reduce the effect of this monitoring attack. Finally, the summary of this chapter is provided in Section 4.4.

Considering the limited time, in this thesis we concentrate on implementing and evaluating all our defined attacks and our security solution against FalseParentPeer attack. The implementation and evaluation of our proposed scoring system, as a solution against FalseLocalData and FalseChildData attacks, is considered as future work.

4.1 FalseLocalData Attack

In FalseLocalData attack, an attacker publishes fake monitoring information about itself. As a result, in overlays which support heterogeneity and different roles for peers with different capacities, the attacker can contribute as little as the weakest nodes in the overlay, or even act as a free rider. For example, by providing fake statistical information with low storage capacity or limited bandwidth, the malicious node pretends to be a weak node. Therefore, it benefits from the overlay without participating in the overlay routines, defined for non-weak nodes in the overlay.

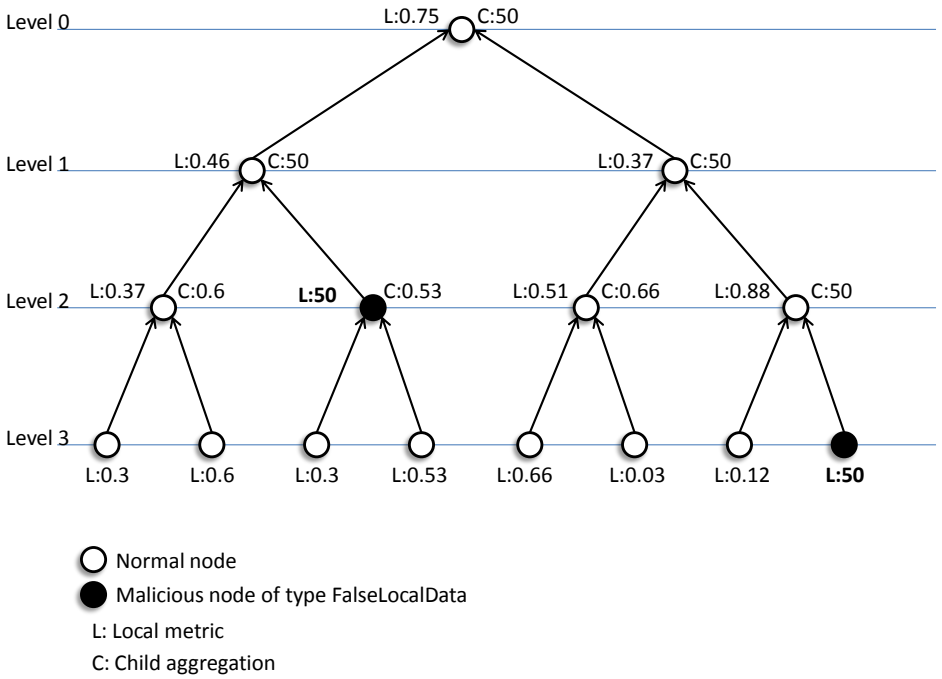


Figure 4.1: Aggregating the Maximum Value at All Nodes in the Presence of FalseLocalData Attackers

Figure 4.1 represents a monitoring tree with branching factor of 2 during the

aggregation of the maximum value at all nodes. There are 15 nodes in the tree and 2 of them are malicious nodes of type FalseLocalData as represented by the black nodes. The rest of the nodes are non-malicious and represented by the white nodes. In our example, the malicious nodes produce a fake value of 50 as their maximum value while the maximum local value at non-malicious nodes ranges between 0 and 1. The figure shows how the global estimation at the root node can be affected by the attackers. Attackers, which are closer to the root node, can manipulate the global estimation sooner than the nodes in the lower levels such as leaf nodes.

While we introduced the FalseLocalData attack in this section, in the following sub-section we propose our security solution against this attack.

4.1.1 Counter-Measure against FalseLocalData Attack

In order to neutralize the attackers who publish false monitoring information about themselves, each parent node, as a receiver of the monitoring data, needs to have the ability to detect improbable monitoring information coming from its children. However, making such a decision is very challenging and, in many cases, there is a possibility of considering genuine monitoring data as a fake piece of information. How does a parent node decide whether the received data is valid or not?

Our idea is that this decision can be made by utilizing an efficient scoring system, composed of series of checks and proper comparison strategies, for comparing received monitoring data from a child node with data from comparable nodes in the monitoring tree. The results of the scoring system can be normalized in a way that the parent nodes assign a scoring point between 0 and 1 to their potential malicious children after each test, and then use the average of the malicious scores as a measure for concluding its final decision.

By integrating the fuzzy logic [30] to our scoring system, the parent nodes are assisted in deciding to what extent a node has the potential to be malicious. Thus, received monitoring data, that is very different from other comparable monitoring values, is considered to be more potentially malicious than improbable values with lesser difference. Consequently, nodes with higher probability of being malicious are given an average malicious score of a value closer to 1 compared with nodes with lesser probability. Below, we explain the details of our scoring system and comparison strategies.

Comparison of Received Data with Global Estimation

When a node receives data from its child, the first series of checks can be the comparison of child aggregation with the global monitoring estimation to detect unrealistic values. For instance, where a child node claims that its aggregated count value is greater than the current global count estimation stored at its parent, it can be considered as an indicator of malicious behavior because in most cases the count value aggregated from the whole network is greater than the count value from part of the tree.

However this conclusion is not always true, and we cannot say that the node is definitely an attacker. Therefore, we consider it as a malicious indicator with the weight of $1 - \frac{|count|}{|globalCount - count|}$ which always results in a value between 0 and 1, and the greater difference between the count and global count results in a ratio closer to 1. By defining a threshold between 0 and 1, the secured nodes can consider senders with the mean malicious weight, greater than a defined threshold, as a malicious node.

Malicious Check	Malicious Score
If (received Count > globalCount)	$1 - \frac{\|Count\|}{\|Count - globalCount\|}$
If (received Min < globalMin)	$1 - \frac{\|Min\|}{\|globalMin - Min\|}$
If (received Max > globalMax)	$1 - \frac{\|\text{Max}\|}{\|Max - globalMax\|}$
If (received Sum > globalSum)	$1 - \frac{\|\text{Sum}\|}{\|Sum - globalSum\|}$
If (received SumSqr > globalSumSqr)	$1 - \frac{\|\text{SumSqr}\|}{\|\text{SumSqr} - global\text{SumSqr}\|}$
If (received Variance > globalVariance)	$1 - \frac{\|\text{Variance}\|}{\|\text{Variance} - \text{globalVariance}\|}$
If (received Mean > globalMean)	$1 - \frac{\|\text{Mean}\|}{\|\text{Mean} - \text{globalMean}\|}$

Table 4.1: Comparison of the Received Monitoring Data against the Global Monitoring Estimation and the Corresponding Malicious Scores

List of various comparisons of global monitoring estimation with received value from a child in addition to the calculation of the malicious score is provided in Table 4.1. In continuation, we introduce more parameters and methods to make

the process of detecting malicious nodes more precise.

Comparison of Received Monitoring Data with *Typical Values* from Sibling's Children

Besides comparing received monitoring data with global monitoring estimation, each parent node can also compare, received child estimation from its children, with child estimations from the children of other nodes on the same level as the expected *typical values*.

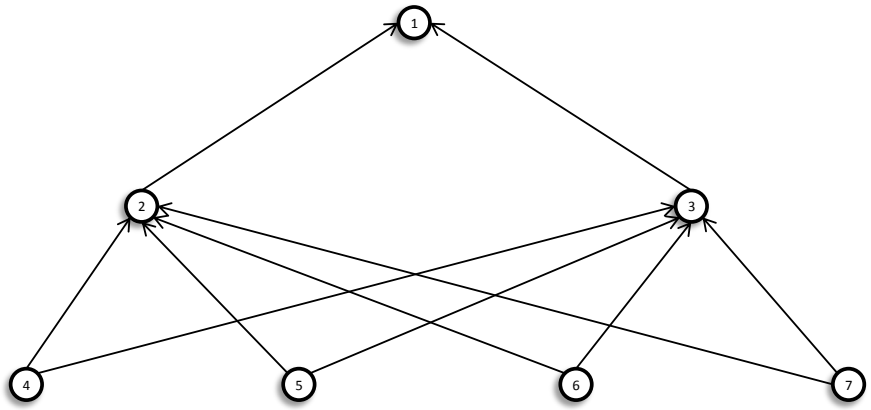


Figure 4.2: Receiving Additional *Typical Values* from the Sibling's Children Nodes

For this purpose, the current monitoring algorithm requires some modifications in order to make each node forward monitoring data not only to its parent but also to other nodes on the same level as its parent node. The suggested modified monitoring tree is depicted in Figure 4.2. The Figure illustrates that node 2 receives data not only from nodes 4 and 5, being its children, but also from node 6 and 7 which are on the same level as its children. Similarly, node 3 receives monitoring aggregation results from the children of node 2.

By receiving data from the children of the sibling nodes, each parent can compare received value from its child with values from nodes similar to its child and decide whether its child's reported value is probable or not. The child estimation is considered as malicious data where it is significantly different from other received values, from similar nodes.

To distinguish improbable values, which are very different from other received values, our proposal is to benefit from the *k-means clustering* algorithm [20]. The k-means clustering algorithm offers an effective method for grouping a given data set into k clusters in a way that values more similar to each other are

clustered in the same groups. For example, by applying the k-means clustering algorithm with $k = 2$ to the following data set $\{1, 2, 3, 4, 5, 20\}$ the resulting clusters will be $\{1, 2, 3, 4, 5\}$ and $\{20\}$.

Our idea of employing the k-means clustering algorithm for detecting improbable received monitoring data is that, after collecting data from children and the children of the sibling nodes, parent node groups the collected values once in 2 clusters and thereafter in 3 clusters (k-means clustering with $k = 2$ and $k = 3$). Subsequent to each clustering round, the parent node assigns a malicious score to the nodes which belong to the group with the minimum number of members. The amount of malicious score can be normalized through dividing 1 by the number of members in the smallest cluster. As a result, the malicious score assigned to a node will be closer to 1 where there are fewer number of nodes in the group, and it will be exactly 1 where the node is the only member of a group.

At the end of clustering phase, the final malicious score of each node can be normalized to a value between 0 and 1 through dividing the sum of the malicious scores by the number of clustering rounds (e.g. by applying $k = 2$ and $k = 3$ the number of clustering round is 2). So, for instance, by applying k=2 to the data set $\{1, 2, 3, 4, 5, 20\}$, we will have 2 clusters of $\{1, 2, 3, 4, 5\}$ and $\{20\}$. The parent will then assign 1, as the malicious score, to the sender of the value 20. In the next round, with k=3 we have 3 clusters of $\{1, 2, 3\}$, $\{4, 5\}$ and $\{20\}$, and again the parent node will assign another malicious score of 1 to the sender of value 20. The final malicious score is the average of various malicious scores which, in our example, is 1 for the sender of the value 20.

By introducing a threshold to the final malicious score, the parent can decide whether to accept data from a child or not. By increasing the threshold towards 1, the number of nodes, considered as malicious, will be decreased. For instance, by setting the threshold to 0.8, only the nodes with final malicious score greater than 0.8 will be considered as attackers.

However, by applying the k-means clustering method even on the data sets with no inconsistency, always some values are grouped in the smallest cluster and consequently some senders will always acquire malicious score from the scoring system. In the next paragraph, we introduce our solution to this issue.

Reducing the Probability of Assigning Malicious Score to Non-Malicious Nodes

As already stated, k-means clustering algorithm can help parent nodes to detect improbable values in data sets. However, by applying the k-means clustering

method even on a data set with no value significantly different from the other ones like $\{1, 2, 3, 4, 5\}$, always some senders will acquire malicious score. To illustrate the problem, if we apply the k-means clustering algorithm with $k = 2$ to the data set $\{1, 2, 3, 4, 5\}$, which has no value greatly different from the other ones, we will have two clusters of $\{1, 2, 3\}$ and $\{4, 5\}$, and as a result senders of values 4 and 5 will acquire malicious scores.

To avoid this problem, a pre-condition may be defined in order to measure the degree of variation in the data set. This can be achieved by utilizing the *Coefficient of Variation* measure [4], and perform the k-means clustering check only when the degree of variation in the data set is greater than a certain threshold. Coefficient of Variation is a statistical measure representing the ratio of variability in relation to the mean of a data set. It is used to measure and compare the degree of variation in data series. The Coefficient of Variation is calculated via dividing the standard deviation by the mean of a data set. Commonly, data series, with Coefficient of Variation greater than 1, are considered as high-variance data sets.

Hence, each parent, after receiving monitoring data from its children and the children of its sibling nodes, will firstly apply the Coefficient of Variation on the received values to determine the degree of variation. The data set will then be suspected of having malicious members only if the Coefficient of Variation is greater than 1. In this case, the parent node will proceed, by clustering the values based on k-means clustering algorithm, and will assign malicious scores to the nodes in the smallest clusters.

Detecting Misleading Typical Values

In case sibling's children send fake data, the received typical values will mislead the scoring system. If, however, in addition to the typical values from the children of sibling nodes, siblings themselves interchange their own child aggregation with each other, they can detect possible anomalies; between the directly received data from their sibling's children AND the reported child aggregation from the sibling nodes. In the case of detecting inconsistency, the sibling node can be reported as malicious to higher level nodes in the monitoring tree in order to prevent using them as typical values. Malicious nodes may, nevertheless, abuse this solution by reporting non-malicious nodes as malicious to nodes in higher levels. Therefore, to improve the solution to this issue, a more detailed study will be required in the future.

Figure 4.3 shows the additional direct interchange of child estimation between sibling nodes, node 2 and node 3, with the bold arrows. Using this method, even the root node benefits by receiving inconsistency reports from its non-malicious

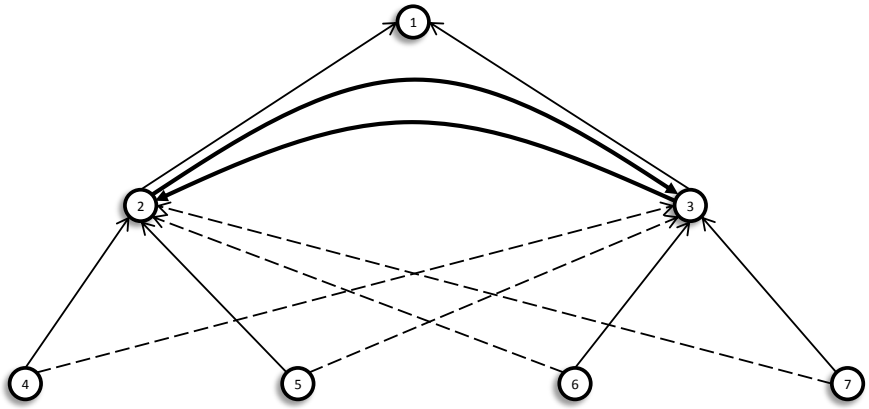


Figure 4.3: Additional Direct Interchange of Child Estimation between Sibling Nodes

children.

Detecting Inconsistencies by Analyzing the History of Received Data
Analyzing the history of received monitoring data for detecting any kind of inconsistencies can be considered as another enhancement to our scoring system. By comparing the monitoring data received from a child and data from the corresponding grand-children, some sort of anomalies can easily be detected.

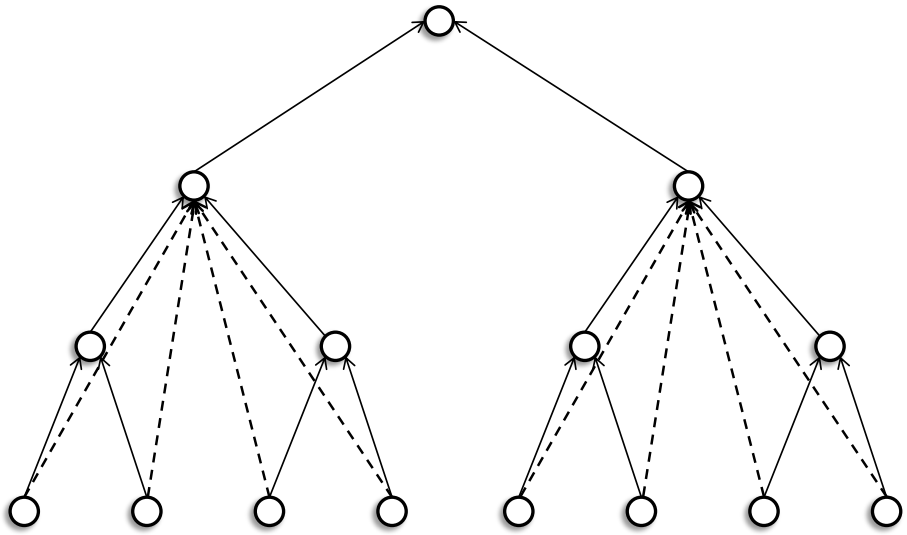


Figure 4.4: Additional Forwarding of the Monitoring Data to the Grand-Parent Nodes

In order to provide the possibility of analyzing the history of received monitoring data, nodes need to forward their monitoring aggregation results not only to their parent nodes but also to their grand-parents. Figure 4.4 shows the extra forwarding directions of monitoring information to grand-parent nodes by the dashed arrows.

Detecting Corrupted Aggregation Results by Analyzing the Received Monitoring Data Itself

Without any comparison of received monitoring information with typical values, parents can still detect some kind of corrupted aggregation results; generated by malicious nodes. This can be achieved by analyzing the received data from a child. For instance, based on the current available aggregation functions in SkyEye.KOM, if the received sum from a node is less than the received min or max, the inconsistency can be considered as an indication of malicious actions.

Malicious Check	Malicious Score
If (received Max > Mean + Standard Deviation)	$1 - \frac{1}{Max - (Min + Standard\ Deviation)}$
If (received Min < Mean - Standard Deviation)	$1 - \frac{1}{(Mean - Standard\ Deviation) - Min}$
If (received Sum < Max)	1
If (received Sum < Min)	1

Table 4.2: Detecting Corrupted Aggregation Results by Analyzing the Received Monitoring Data Itself

The same conclusion can be reached if the received min is less than the subtraction of the mean from the Standard Deviation, or $max > mean + StandardDeviation$. Similar checks, in addition to the ones mentioned above with the amount of malicious score for each, are listed in Table 4.2.

Strategies after Detecting Attackers

After detecting malicious nodes, different strategies can be defined as a reaction of the parent node that receives monitoring data from a FalseLocalData attacker. The receiver can ignore the potential malicious nodes by avoiding

the use of data from the attacker in the monitoring aggregation process. However, this strategy will also result in ignoring data from all descendants of the malicious node.

Therefore, a solution could involve the receiver contacting the children of the malicious node and aggregating their information instead of the data from the malicious node. Alternatively, the receiver could overwrite the monitoring data, from the malicious node, with the data received from one of its non-malicious children.

In this section, we introduced the FalseLocalData attack in which the attacker generates false monitoring information about itself. Afterward, we proposed the idea of utilizing a scoring system for detecting potential malicious child nodes. We proposed additional enhancements to improve the accuracy of our scoring system and, finally, we suggested some reaction strategies following the detection phase. In the next section, we introduce our next defined malicious behavior, named FalseChildData, and the possible counter-measures.

4.2 FalseChildData Attack

Our second designed malicious behavior is called FalseChildData attack. In this attack, the malicious node changes the received monitoring information from its children before aggregating it with its local metric. In fact, the FalseChildData attacker ruins the aggregation result of its children and all the nodes below them. The degree of harm by the FalseChildData attack is increased when the malicious nodes located on higher levels of the tree. This is because in such cases it can manipulate the monitoring data of more nodes.

In the monitoring tree, depicted in Figure 4.5, there are 2 malicious nodes of type FalseChildData, represented by the black nodes, that overwrite the maximum value reported by their children with the fake value of 50. The other 13 nodes, represented by the white nodes, are not malicious and aggregate the maximum value received from their children properly. The maximum value at non-malicious nodes is in the range of $0 - 1$. The effect of the malicious node at level 3, the leaf node, is limited to its own children while the attacker in level 2 damages the received aggregated data from its children as well as its grand-children.

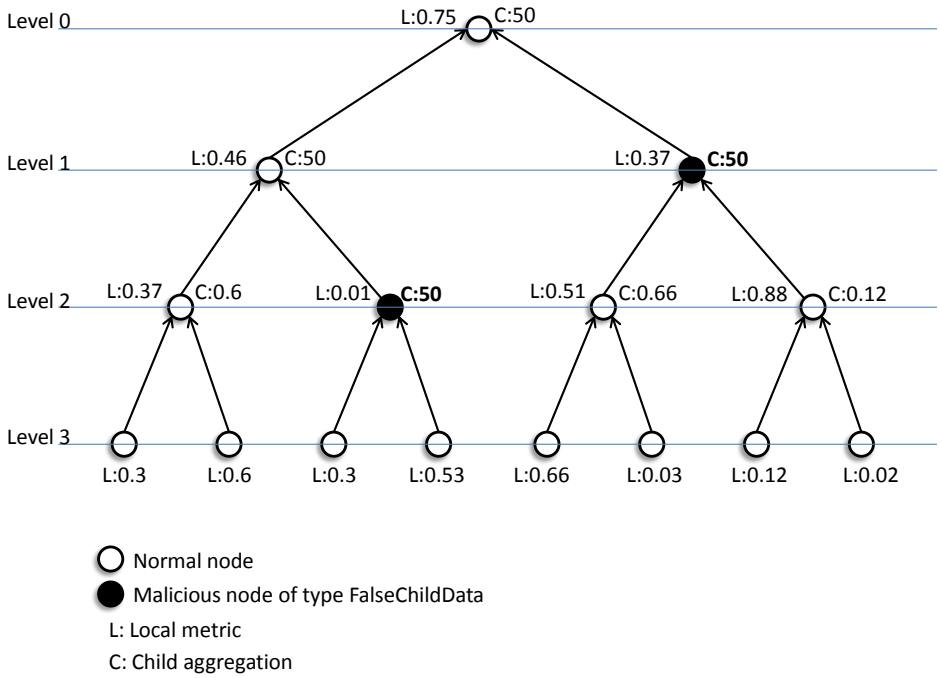


Figure 4.5: Aggregating the Maximum Value at All Nodes in the Presence of FalseChildData Attackers

4.2.1 Counter-Measure against FalseChildData Attack

Since the received monitoring information from a child is the aggregation result of its local data and its child estimation, the scoring system introduced in the previous section can also be beneficial for detecting the FalseChildData attackers that manipulate their children aggregation.

The basic concept behind our scoring system is the early detection of the improbable monitoring information by the parent nodes before aggregation. The improbable monitoring data can be the result of the FalseLocalData or FalseChildData attack. As the local data and child aggregation are not sent to parents as separate data sets, it is not easy for the parent nodes, being the receiver of the aggregation results, to decide whether the malicious data is the result of fake local statistics or manipulated child estimation.

Likewise, in line with the reaction strategies to the FalseLocalData attack, the monitoring data from the children of the malicious node can be used as the alternative information. As an another approach, the parent node can overwrite received malicious information with the data from one of its non-malicious child nodes.

In this section, we defined the FalseChildData attack in which the attacker manipulates the received data from its children before aggregation and further forwarding of the aggregated result. Moreover, we highlighted why our scoring system and the proposed reaction strategies, defined against FalseLocalData attack in Section 4.1.1, can also be utilized as a counter-measure strategy against FalseChildData attack. In the next section, we define our last attack named FalseParentPeer and our counter-measure solution against it.

4.3 FalseParentPeer Attack

Based on the SkyEye.KOM algorithm, each node in the monitoring tree is expected to aggregate and forward the monitoring information to its parent node. However, our last designed attacker aggregates the monitoring data correctly but, instead of forwarding the aggregated result to its parent, it sends the data to a random node in the tree other than its parent node. This attack has the ability to damage monitoring data dramatically, since it ruins the expected data flow in the monitoring tree, and it also has the potential to cause infinite loops in the monitoring structure.

Figure 4.6 illustrates an example of malicious behavior by FalseParentPeer attackers. According to the figure, there are 31 nodes in the construction of the tree and among them there are 5 FalseParentPeer attackers, specified by the black nodes. The forward direction of each attacker, to a random node other than its parent, is shown with the curved arrows while the expected direction, already violated, is depicted by the dashed arrows. The white nodes represent the non-malicious nodes and forward monitoring data to their parents as anticipated. In the illustrated example, the attackers can cause loops, as per area a, or even break apart the tree for a while, as per area b.

In this section, we defined the FalseParentPeer attackers which forward their aggregated monitoring data to a random node other than their parents. In the following sub-section, we explain our security solution to deal with this attack.

4.3.1 Counter-Measure against FalseParentPeer Attack

Our plan for reducing the effect of FalseParentPeer attack is that each node must know the range of its acceptable tree IDs- based on its domain length- and ignore monitoring data from senders outside its responsibility area. In addition,

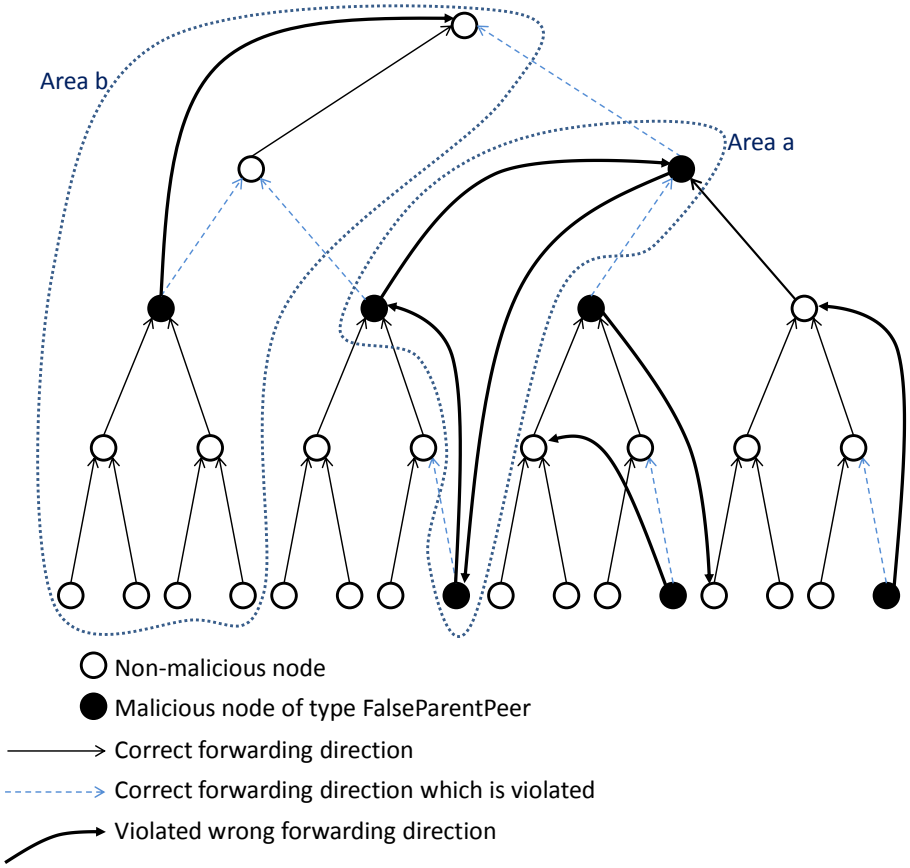


Figure 4.6: Overview on the SkyEye.KOM Monitoring Tree in the Presence of the FalseParentPeer Attackers

each node needs to predict the possible tree IDs of its potential children and define limited acceptable ranges in its domain based on the branching factor (bF). As a result, receivers of the monitoring messages reduce the possibility of accepting data from nodes other than their child nodes by ignoring senders outside their defined acceptable ranges.

For this purpose when the secured node receives a message, it retrieves the sender's peer ID and compares it with its maximum and minimum domain IDs. If the tree ID is in its domain range, the receiver applies the next validation strategy to the received data. The secured node identifies that, where the sender's tree ID is greater than its own tree ID, the sender must be located on its right-hand side, called the *right domain*. Analogously, senders with tree IDs less than the receiver's tree ID are expected to be on the *left domain* of the receiver.

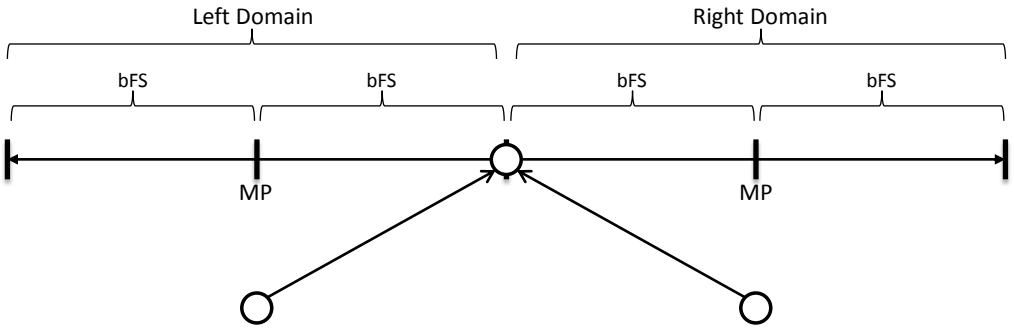


Figure 4.7: Definitions Used in FalseParentPeer Security Solution

At this point, the secured node divides its whole domain into $bF + 1$ parts with the same length, as we call this length *bFS*. Afterward, the secured node constructs ranges around the meeting points (MPs) of the parts on its left and right domain. An overview on the used terminology is given in Figure 4.7. In the following, we propose three examples of acceptable ranges.

Acceptable Ranges around MPs to the Size of bFS/2 on Each Side

Our first proposed ranges are located around MPs with half of the length of bfS on each side. Figure 4.8 depicted these ranges for a secured node in a monitoring tree with the branching factor of 2.

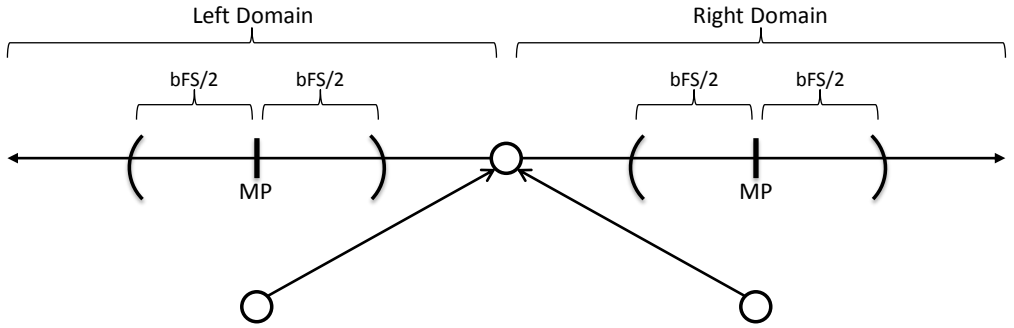


Figure 4.8: Acceptable Ranges around MPs to the Size of $bFS/2$ on Each Side

Acceptable Ranges on the Right-Hand Side of MPs and to the Size of bFS/2

The acceptable ranges can be varied depending on the p2p overlay. For instance, in Chord the responsible nodes are the successor node, therefore, they have a greater ID than a given node ID whereas in Pastry the closest node to

an ID is the responsible one. Therefore, based on the Chord overlay, a secured node should only accept monitoring data from nodes that are greater than its own tree ID.

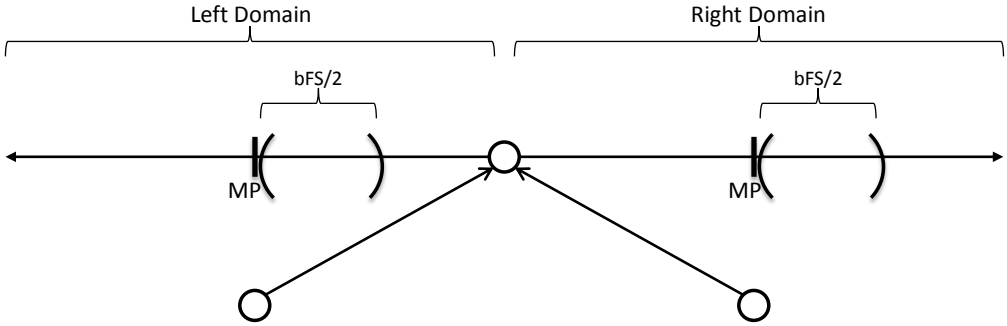


Figure 4.9: Acceptable Ranges on the Right-Hand Side of MPs and to the Size of $bFS/2$

Figure 4.9 depicted our second proposed acceptable ranges which consider Chord as the p2p overlay. Since we know that the responsible peer for a node in the Chord overlay is one of its successor nodes, we expect that most of the children are located in the right regions of MPs. Therefore, we halve the acceptable ranges by removing half of each range on the left side of MPs.

Acceptable Ranges Right-Hand Side of MPs to Size of bFS

In our final approach, we increase the previous ranges to the size of bFS on the right side of MPs in order to decrease the possibility of ignoring legitimated child nodes; which might be located out of the previous ranges. Increasing the size of the acceptable ranges may also result in the increasing possibility of malicious nodes being accepted by the parent nodes. Figure 4.10 shows the proposed extended ranges.

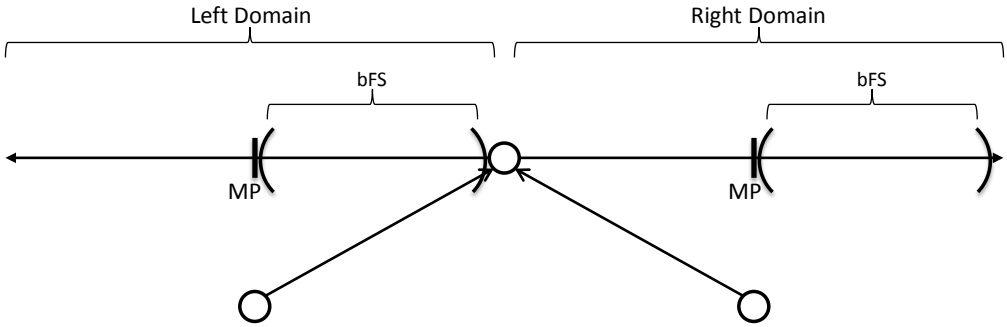


Figure 4.10: Acceptable Ranges on the Right-Hand Side of MPs and to the Size of bFS

We simulate and evaluate the secured nodes with all three acceptable ranges, which were introduced in this section, and the results of the evaluation can be found in the evaluation chapter, Section 6.3.

In this section, we introduced our counter-measure idea to reduce the effect of the FalseParentPeer attack. The goal of our solution was that parent nodes predict the ID ranges of their children and accept monitoring data only from nodes with IDs in those ranges. We explained the dependency of acceptable ranges on p2p overlay and introduced two acceptable ranges, both based on the Chord overlay. The following section summarizes this chapter.

4.4 Summary

In this chapter, we defined three types of malicious behavior for nodes taking part in the SkyEye.KOM monitoring overlay and, for each attack, we introduced our counter-measure solutions. In the first defined attack, FalseLocalData, attackers publish false monitoring data about themselves. Our second type of attackers, FalseChildData attackers, change monitoring data received from their children and in FalseParentPeer, our last defined attack, attackers send the monitoring data to a random node in the monitoring tree other than their parent nodes.

We proposed our scoring system as a solution to detect malicious nodes of types FalseLocalData and FalseChildData. Additionally, to improve the accuracy of our scoring system, we introduced further modifications to the data flow in the monitoring tree. We also suggested some reaction strategies after the detection of the malicious nodes.

As a counter-measure solution against FalseParentPeer we introduced acceptable ranges and suggested some example ranges based on the Chord overlay. In the following chapters, Chapter 5 and Chapter 6, we explain the implementation and evaluation result of our counter-measure solution against FalseParentPeer together with the implementation and evaluation of our three types of attacks.

5 Implementation

Contents

This chapter provides an overview of how our security attacks and counter-measure solution against FalseParentPeer attack, introduced in chapter 4, are implemented. This implementation is carried out in the application layer of the PeerfactSim.KOM simulation framework, in Java language.

After presenting a general overview of the structure of this implementation in Section 5.1, the details specific to the implementation of the FalseParentPeer, FalseLocalData and FalseChildData attacks are explained in Sections 5.1.1, 5.1.2 and 5.1.3 respectively. Subsequently, Section 5.2 details the implementation of our security solution against FalseParentPeer attack and Section 5.3 summarizes this chapter.

5.1 Implementation of the Attacks

The attacks and counter-measure solution against FalseParentPeer attack are implemented in an individual package called *structuredmalicious*. Most of the classes in this package are an extension of those in the *structured* package. The structured package contains the primary classes for the implementation of the

structured monitoring. The malicious behaviors are designed to be easily configured in the XML configuration files. If the newly-implemented *structured-malicious* monitoring type is selected, the type of attack must be also declared from the available options of *FalseParentPeer*, *FalseLocalData*, *FalseChildData* and *Non*, prior to run a simulation. The ratio of available malicious nodes in each simulation is configured by the *ratioOfMaliciousNodes* variable in the simulation configuration file. For example, setting the *ratioOfMaliciousNodes* variable to 0.05 will result in 5% of the nodes in the overlay behaving maliciously according to the behavior defined by the *attackType* variable.

The *MonitoringApplicationFactory* class checks the variables related to the selected type of monitoring, in accordance with the simulation configuration file, and initiates the monitoring application. Following to the selection of the overlay from the supported hosts, *AdaptiveChord*, *Chord*, *EpiChord*, *MaliciousChord*, *ReChord* and *Pastry*, monitoring type is checked. The current available monitoring types are *Structured*, *Unstructured* and *StructuredMalicious*.

Where *StructuredMalicious* is the monitoring type, a random number of type double is generated and compared with the value of the *ratioOfMaliciousNodes* variable before each node is generated in the overlay. If the generated random value is greater than the ratio of malicious nodes, a non-malicious node is created. On the other hand, if the generated value is less than the ratio of malicious nodes, the *attackType* variable is checked and a malicious node of the defined type is generated.

The false values for the FalseLocalData and FalseChildData attacks are also configured in the configuration file. The *localMaliciousVal* and *childMaliciousVal* variables are designed for this purpose. The following subsections provide details about the implementation of each attack type.

5.1.1 FalseParentPeer Attack

The class *FalseParentPeer* is responsible for simulating the type of malicious behavior in which the malicious node forwards the monitoring data to a peer other than its parent node. To achieve this, the malicious node firstly generates a random tree ID and calculates the corresponding peer ID to the generated tree ID. Next, instead of forwarding its monitoring aggregation result to its parent, which is responsible for this peer ID in the p2p overlay, the malicious node sends its monitoring data set to a node with the peer ID corresponding to the generated random tree ID.

The FalseParentPeer class is an extension of the *MonitoringStructManager*

class. The MonitoringStructManager class is responsible for calculating the position of nodes in the monitoring tree, in addition to sending monitoring data and processing any monitoring messages received. The *distributeData* method, which is responsible for distributing monitoring data, is overridden in the FalseParentPeer class.

While the original distributeData method simply sends monitoring data to the current node's parent, when the current node is not the root of the tree, the overridden method, used in the FalseParentPeer class, sends monitoring data to a random node in the monitoring tree. The distributeData method is periodically called whenever a local node needs to send its aggregated data to its parent.

5.1.2 FalseLocalData Attack

The *FalseLocalData* class in the monitoring layer simulates the FalseLocalData attack in which the malicious node provides false monitoring data about itself. The MonitoringApplicationFactory calls the FalseLocalData class by providing the *falseVal* as the argument. The falseVal is set by the *localMaliciousVal* variable in the configuration file. FalseLocalData class creates a new data manager called *MonitoringStructMaliciousLocalDataManager* as an extension of the *MonitoringStructDataManager* class, overrides the *aggregatePresentData* method and is responsible for aggregating the local data set and child estimation of the current handler.

The overridden aggregatePresentData method overwrites all data entries except *count* including *max, mean, min, std_deviation, sum, sum_squares* and *variance* for each of the *sinus, ziczac, dirac* and *rectangle* functions with the value of falseVal variable. The manipulated results are stored in a monitoring data set named *maliciousLocalDataSet*, which is used as the local monitoring data set for the final aggregation with child estimations in malicious nodes of type FalseLocalData.

5.1.3 FalseChildData Attack

In order to generate malicious nodes that manipulate the monitoring data received from child nodes, the *FalseChildData* class in the monitoring layer is implemented. Similar to the *FalseLocalData* class, *FalseChildData* class is also an extension of *MonitoringStructManager* class.

The *FalseChildData* class uses the *MonitoringStructMaliciousChildDataManager* in its constructor, which is called by the MonitoringApplicationFactory

class, when providing false values as the argument. FalseChildData class creates a new data manager, named *MonitoringStructMaliciousChildDataManager*, which is an extension of the MonitoringStructDataManager class.

The aggregatePresentData method is overridden in the MonitoringStructMaliciousChildDataManager class, since this method is responsible for aggregating local data set and child estimation in the current handler. FalseChildData attacker overwrites all data entries received from its children with the pre-configured value of the *childMaliciousVal* variable and uses the resulting manipulated data set as child estimation.

This section explained the implementation of the attacks defined in this thesis. The next section outlines how our security solution against FalseParentPeer attack is implemented.

5.2 Implementation of the Counter-Measure Solution

Our countermeasure against FalseParentPeer attack is implemented in the *Secured* class, located in the *monitoringlayer* package. The *Secured* class extends the MonitoringStructManager class. Since our solution, as explained in section 4.3.1, is based on processing the received messages, the core of our security solution is implemented by overriding the *processMsg* method.

The processMsg method is responsible for processing the incoming monitoring messages. In this method, if the received message is an acknowledgment (ACK), the receiver overwrites its global estimation with the contained global estimation in the ACK message. Otherwise, if the current node is the root of the tree, the global estimation is calculated by aggregating the local data and the received child estimations.

Nodes other than the root node hand over the received global estimation data set from their parents to their children as part of the ACK messages. Messages other than ACK type are considered as child messages and aggregated as child estimation if the comparison of the *message timestamp* and the *data timeout* shows the received data has not yet expired.

In the overridden version of the processMsg method, in the Secured class, receiver of monitoring messages retrieves sender's peer ID from the message and calculates the sender's tree ID by calling the *calcTreeIDFromPeerID* method.

In the first step, after receiving monitoring data, a secured node compares the calculated tree ID with its maximum and minimum domain values, and ignores messages with tree ID outside its boundary of responsibility. The calculation of the minimum and maximum boundary of domains is implemented in the *calculatePosition* method which is part of the MonitoringStructManager class, inside the structured monitoring package. The calculatePosition method is responsible for building and maintaining the monitoring tree structure. The minimum and maximum values are set every time the responsibility of a node in the monitoring tree changes due to restructuring of the tree and re-positioning of the nodes.

In the second step, if the message's tree ID is within the receiver's domain, the coordinator of the domain divides its domain length into $bF + 1$ parts and constructs its acceptable ranges in line with the chosen strategy for acceptable ranges introduced earlier in Section 4.3.1. By comparing the sender's tree ID with its own tree ID, the receiver determines that the tree ID must be located on its right-hand side or the left hand-side domain. The receiver only stores received child estimation only if the sender's tree ID is within its defined acceptable ranges and drops messages from other senders.

This section covered the implementation of our counter-measure against FalseParentPeer attack and the dependencies of the relevant classes for achieving this goal. In the next section, we summarize this chapter.

5.3 Summary

In this chapter, we provided the details of the implementation of our attacks and counter-measure against the FalseParentPeer attack respectively. We introduced the configuration options required for simulating overlays with a certain ratio of malicious, non-malicious and secured nodes. Furthermore, we explained the dependencies of the classes in relation to each of the defined monitoring attacks in the *structuredmalicious* package. Additionally, we reviewed the other classes relevant to our work and provided a description of the newly-created classes and methods involved in generating malicious and secured nodes.

6 Evaluation

Contents

In the following chapter, we evaluate our defined malicious behaviors, as well as our security solution against FalseParentPeer attack. After providing an overview of our evaluation goal, method and the simulation configurations, used in the evaluation scenarios, in Section 6.1, we further split our evaluation into two individual parts, including the evaluation of both the attacks and the security solution. In Section 6.2, we evaluate the effect of our defined malicious behaviors through four scenarios with different setups. In Section 6.3, we evaluate the effectiveness of our security solution against FalseParentPeer attack, involving three scenarios, by simulating secured nodes with the three different acceptable ranges, as defined earlier in Chapter 4. Finally, we summarize the conclusions based on our observations in Section 6.4.

6.1 Evaluation Overview

In this part, we firstly make our evaluation goals clear in Section 6.1.1. In Section 6.1.2, we introduce our evaluation method and the measurement functions which are employed during the whole evaluation phase. In Section 6.1.3, we explain the action file used in our simulations alongside the simulation setup.

6.1.1 Evaluation Goal

The aim of this evaluation is to analyze and assess the impact of FalseLocalData, FalseChildData and FalseParentPeer attacks on the structured monitoring mechanism, offered by SkyEye.KOM. In addition, we evaluate our security solution against FalseParentPeer attack to assess the extent to which our solution can neutralize the effect of malicious nodes.

For each attack, we measure the effect of increasing the ratio of malicious nodes on the monitoring results. Furthermore, we study the impact of different branching factors on the monitoring results, in the presence of different types of malicious nodes, to discover whether changing the branching factor can alter the effect of malicious nodes. We also evaluate the effect of the attacks, in a more realistic situation, by enabling churn in the simulated network.

We examine our security solution with the three acceptable ranges, defined in Section 4.3.1, in order to determine which ranges help the secured nodes produce more accurate monitoring results, in the presence of malicious nodes.

6.1.2 Evaluation Method

Our evaluation is based on analyzing the results of a series of simulation scenarios, which are simulated with the help of the PeerfactSim.Kom simulation framework. In each scenario, we take advantage of the four measurement functions of sinus, dirac, rectangle and ziczac; provided by the simulator as the exemplary measurements. The predictability of these functions is ideal for our evaluation purposes since we can always compare the simulation results against the expected values in order to observe the proximity of the measured values, reported by the root node, in relation to the desired results.

The measurement functions are initiated based on the simulation time, as shown in Table 6.1. The length of measure intervals, which specifies the re-

Measurement Function	Value
sinus	$\frac{(time\ mod\ SI)2\pi}{SI}$
dirac	$\begin{cases} 1, & \frac{(time\ mod\ SI)}{SI} \geq 0.9 \\ 0, & else \end{cases}$
rectangle	$\begin{cases} 1, & \frac{(time\ mod\ SI)}{SI} \geq 0.5 \\ 0, & else \end{cases}$
zigzag	$\frac{(time\ mod\ 2SI) - SI}{2SI}$

Table 6.1: Measurement Functions

newal periods of values, is represented by the *SI* variable in the table. In this chapter, we only focus on the sinus function results and include the results of the other functions in the appendices. This is due to the fact that the monitoring mechanism behaves in a similar way toward any of these four different measurement functions.

For analyzing the results of each simulation, we use the *.dat files, generated by the PeerfactSim.Kom simulator as output, together with their graphical representations, produced by utilizing GnuPlot Script.

6.1.3 Simulation Setup

In PeerfactSim.KOM, the network model actions are described in the so-called monitoring *action files*. The action files are *.dat files, with their content written in a self-descriptive pseudo-code-like language, and are used to define which action, or series of actions, should be performed by a node, or a group of nodes, in the network during the simulation. Furthermore, action files enable us to define the timing of actions and each action can then be scheduled to be executed at specific intervals throughout a simulation.

Based on the content of our action file, presented in Figure 6.1, nodes join the overlay from the beginning of the simulation, represented by minute 0, for the duration of 60 minutes. The joining phase is equally distributed between nodes, in a way that the first node joins the network in the first few seconds of the joining phase and the last node in the last few seconds. After the joining phase, a time period of 30 minutes is allocated for the stabilization of nodes.

```
#Monitoring Application Action File
reporter 1s MonitoringApplication:join
all 3s-60m MonitoringApplication:join
reporter 90m MonitoringApplication:distributeMonData
all 90m-91m MonitoringApplication:distributeMonData
```

Figure 6.1: Content of the Action File

The stabilization period is spent on distributing the configuration among all the peers which joined the network in the previous phase. The amount of time, specified for joining and the stabilization of nodes, should be determined carefully with reference to the number of nodes participating in each simulation. Allocating insufficient time for these two periods will produce faulty and confusing simulation results because, due to lack of time, there might exist nodes with no responsibility assigned and consequently the positions in the monitoring tree may change.

Finally during the last 90 minutes of the simulation, representing the monitoring phase, nodes periodically report their monitoring data to their parents and the simulator analyzer interface simultaneously. To ensure comparability of our simulation results, we use the same action file in all of our simulations.

To evaluate our defined attacks, we simulate all the scenarios with 1000 and 5000 nodes in order to examine the scalability of our implemented attacks. In this chapter, however, we study the details of the each simulation results by focusing on the outputs with 5000 nodes. The results of all simulation runs, with 1000 and 5000 nodes, are shown in the appendices.

simulation details	• Simulator: PeerfactSim.KOM • Number of simulated nodes: 1000 and 5000 nodes • Overlay: Chord • Update Interval: 30 sec • Delay based on global network coordinates • No Packet Loss
Network model Actions	• 0-60 min: join the overlay • From 90 min: start monitoring • At 180 min: simulation ends

Table 6.2: General Settings

Table 6.2, presents the general settings of our simulations. According to the evaluation result in [10], using the SkyEye.KOM monitoring approach, the optimal monitoring results can be achieved by setting the update interval to 30 seconds while considering the loss of precision and the adaption rate. Thus, we

also benefit from the fixed value of 30 seconds as the update interval through our simulations. The update interval defines the interval between two distribution runs of a single node.

Our evaluation is based on the SkyEye.KOM monitoring overlay on top of Chord, with no packet loss and a network delay model based on the global network positioning [25].

In scenario A, Section 6.2.1, we analyze and compare the effect of our defined malicious behaviors, FalseParentPeer, FalseLocalData and FalseChildData, in a network with 5% of malicious nodes and the branching factor of 4. The overview of the simulations for evaluating the attacks is presented in Table 6.3. In scenario B, Section 6.2.2, we study the behavior of the monitoring mechanism with different branching factors of 2 and 4 in the networks polluted with 5% of malicious nodes. An increase in the branching factor results in a decrease of the monitoring tree's height. Consequently, in a low height monitoring tree, data can reach the root node in a shorter amount of time. In scenario C, Section 6.2.3, we increase the ratio of the malicious nodes by up to 20 percent. Lastly, in scenario D, Section 6.2.4, we include the exponential churn to our simulated network polluted with 5% of malicious nodes.

In the second phase of the evaluation, we study the effectiveness of our security solution against FalseParentPeer attack by simulating 300 nodes, the branching factor of 2 and the same action file that we utilized in the evaluation of attacks. We simulate three scenarios with different acceptable ranges which were introduced earlier in Section 4.3.1. In each scenario, we simulate networks once with 95% of secured nodes, and then without any secured nodes, in order to measure the degree of improvements in the accuracy of the monitoring results that are produced by the secured nodes.

The simulation setups for evaluating the security solution are summarized in Table 6.4. In scenario E, Section 6.3.1, the monitoring results from the secured nodes with limited acceptable ranges, around MPs and to the size of $bFS/2$ on each side, are examined. Scenario F, Section 6.3.2, provides the result of secured nodes with more limited acceptable ranges, reduced to the right-hand side of MPs and to the size of $bFS/2$. Finally, in scenario G, Section 6.3.3, we present the result of secured nodes, with acceptable ranges still on the right-hand side of MPs, but this time to the size of bFS.

A) Comparing different malicious behaviors	
Variation	• Malicious node type: FalseParentPeer , FalseLocalData, FalseChildData
Fixed	• Ratio of malicious nodes: 5% • Branching Factore: 4 • Local False Value for FalseLocalData attack: 50 • Child False Value for FalseChildData attack: 50 • Churn: Disabled

B) Impact of various branching factors	
Variation	• Malicious node type: FalseParentPeer , FalseLocalData, FalseChildData • Branching Factor: 2, 4
Fixed	• Ratio of malicious nodes: 5% • Local False Value for FalseLocalData attack: 50 • Child False Value for FalseChildData attack: 50 • Churn: Disabled

C) Impact of different ratio of malicious nodes	
Variation	• Malicious node type: FalseParentPeer , FalseLocalData, FalseChildData • Ratio of malicious nodes: 5%, 10%, 20%
Fixed	• Malicious node type: FalseParentPeer , FalseLocalData, FalseChildData • Ratio of malicious nodes: 5%, 10%, 20%

D) Churn Impact	
Variation	• Malicious node type: FalseParentPeer , FalseLocalData, FalseChildData • Churn: Disabled, Exponential
Fixed	• Ratio of malicious nodes: 5% • Branching Factor: 4 • Local False Value for FalseLocalData attack: 50 • Child False Value for FalseChildData attack: 50

Table 6.3: Simulation Setups for Evaluating the Attacks

E) Acceptable ranges around MPs	
Variation	• Ratio of secured nodes: 0%, 95%
Fixed	• Acceptable ranges: Around MPs to the size of bFS/2 on each side • Ratio of malicious nodes: 5% • Branching Factore: 2 • Churn: Disabled

F) Acceptable ranges based on Chord overlay	
Variation	• Ratio of secured nodes: 0%, 95%
Fixed	• Acceptable ranges: Right-hand side of MPs and to the size of bFS/2 • Ratio of malicious nodes: 5% • Branching Factore: 2 • Churn: Disabled

G) Extended acceptable ranges based on Chord overlay	
Variation	• Ratio of secured nodes: 0%, 95%
Fixed	• Acceptable ranges: Right-hand side of MPs and to the size of bFS • Ratio of malicious nodes: 5% • Branching Factore: 2 • Churn: Disabled

Table 6.4: Simulation Setups for Evaluating the Counter-Measure Solution

6.2 Evaluation of the Attacks

This part is dedicated to the evaluation of our defined attacks. In Section 6.2.1, we analyze and compare the effect of our three types of attacks, which are FalseLocalData, FalseChildData and FalseParentPeer. In Section 6.2.2, we examine the impact of changing the branching factor on the monitoring results in the presence of the attackers. In Section 6.2.3, we simulate networks with various proportion of malicious to non-malicious nodes and Section 6.2.4 shows the effect of the defined attacks on the churn enabled simulated networks.

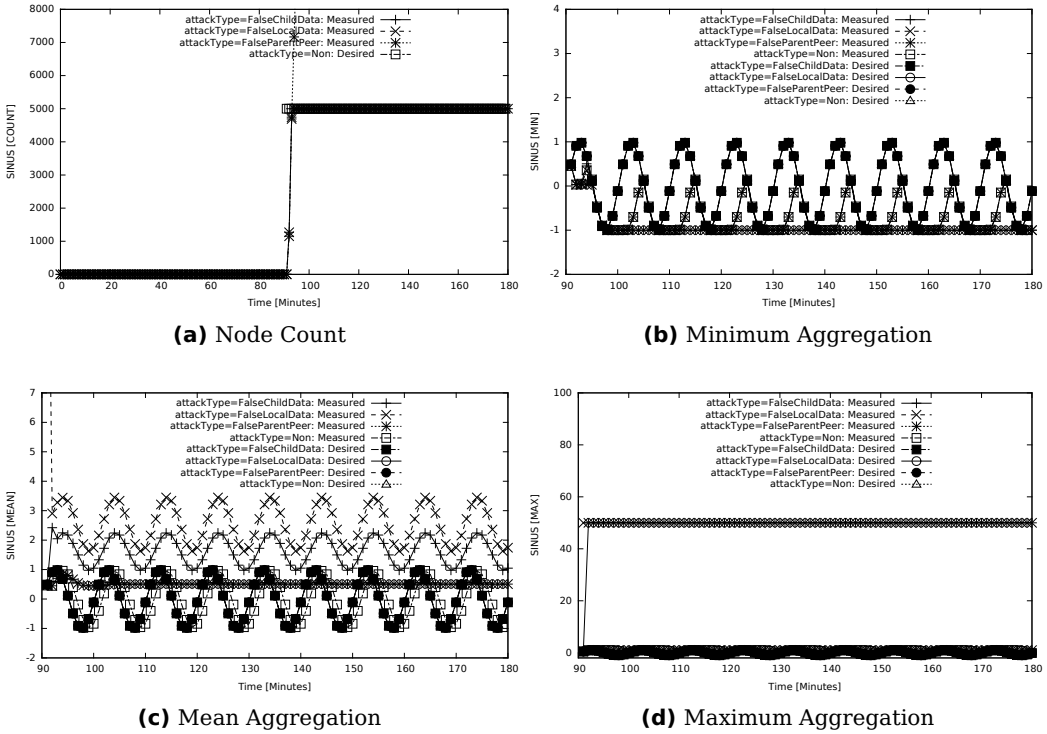
6.2.1 Scenario A: Comparing Different Malicious Behaviors

Figure 6.2 shows the comparison of our three types of attacks, with 5% of malicious nodes and the $bF = 4$. Figure 6.2(a), reveals the node count measured in the networks polluted with the three types of malicious nodes, and additionally the node count measured in a network with no attackers. Since the total number of the nodes in the tree does not change during each simulation, the measured node count remains at the correct value of 5000 nodes once the monitoring data from all the nodes in the tree reaches the root node.

As the attackers in FalseChildData and FalseLocalData attacks manipulate the values of the aggregation functions, but not the number of aggregated values, the node count in the presence of these two attackers is not affected and the measured value in the networks polluted by them is reported correctly. However, we observe a continuous steep increase in the count value in the presence of the FalseParentPeer attackers, which begins from the first minutes of the monitoring phase. The reason is that the FalseParentPeer attackers have a high potential of producing loops in the monitoring tree. In addition, each time a node in the overlay receives data from a FalseParentPeer attacker; it reports a new increase in its count aggregation value to its parent node in the next update interval.

Figure 6.2(b) shows the result of the minimum aggregation function, aggregated from all the nodes by the root node. FalseLocalData and FalseChildData attacks show a similar effect on the minimum value while the adaptation delay by these two attacks is rooted in the value of the update interval which is every 30 seconds.

In the network polluted with FalseParentPeer attackers, we observe that the minimum aggregation result converges toward the constant value of -1 in a short amount of time; typically in less than 10 minutes from the start of the monitoring phase. This is because the messages are not sent along the ex-



(a) Node Count **(b)** Minimum Aggregation

(c) Mean Aggregation **(d)** Maximum Aggregation

Figure 6.2: Comparison of the Three Types of Attacks

pected structure but rather to the random nodes in the monitoring tree; which can result in loops in the message. Message loops are very likely when the FalseParentPeer malicious node forwards its monitoring data to one of its descendants.

We observe a similar behavior for the mean aggregation in Figure 6.2(c) by FalseParentPeer attack. The mean value is reported constantly at around 0.5 after 10 minutes from the beginning of the monitoring phase. The reason is that the mean aggregation is the result of the sum aggregation divided by the node count and both of these values increase dramatically, under the impact of the loops in the tree, from the first minutes of the aggregation.

Figure 6.2(c) also reveals that the FalseChildData attack produce lesser effect in comparison to the FalseLocalData attack, although the false value in both cases is set to the similar value of 50. The reason for this observation is that in the structure of the monitoring tree around half of the nodes are the leaf nodes, which have no children. Consequently, malicious behavior by FalseChildData attackers which are located in the lowest level of the monitoring tree cause no harm simply because they have no children to manipulate their monitoring

data. Therefore, in this plot we observe that FalseChildData attack shifts the mean aggregation result by 2 on the y-axis while the amount of shift on the y-axis by the FalseLocalData attack is around 2.5.

Furthermore, the adaptation delay is more obvious in this plot. Due to the update interval of 30seconds in the networks polluted with the FalseLocalData and FalseChildData attackers, the adaptation delay is similar. This also applies to the network with no malicious nodes.

The false value, configured for simulating the FalseLocalData and FalseChildData attack, can be observed in Figure 6.2(d) which constantly shows the maximum aggregation of 50 from the very early minutes of the monitoring phase.

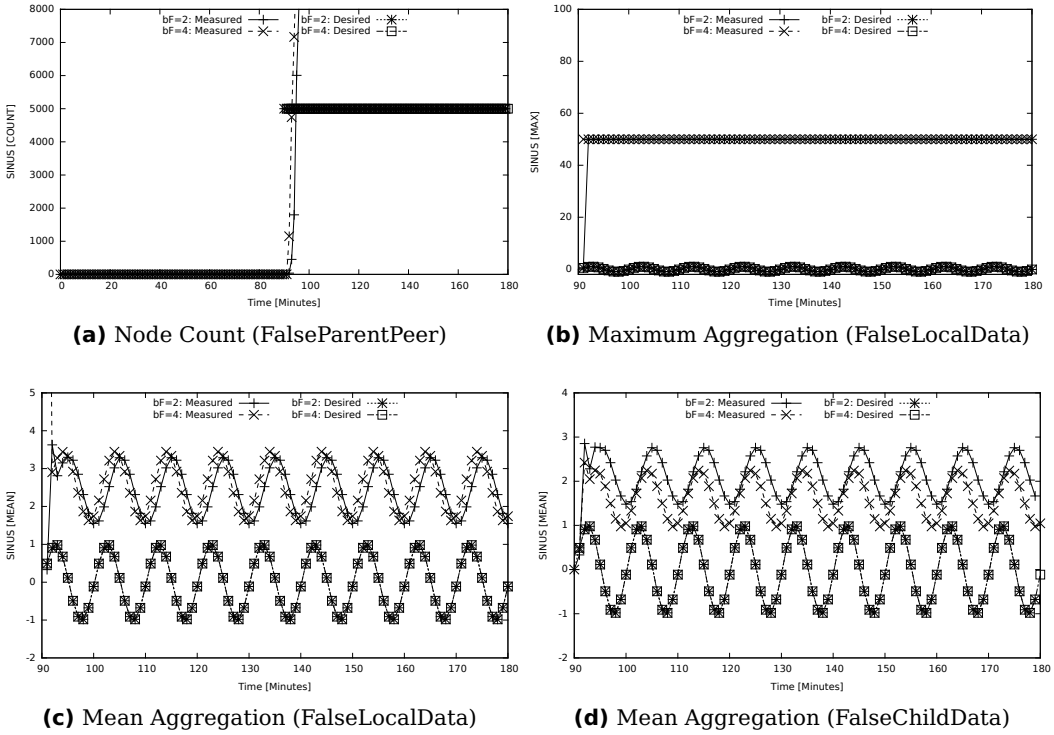
6.2.2 Scenario B: Impact of Various Branching Factors

In this section, we study the effect of different branching factors on the monitoring results in the presence of different types of malicious nodes. Figure 6.3(a) illustrates the node count, aggregated in a network including 5% of FalseParentPeer malicious nodes. In general, since the increase of the branching factor, bF, results in a tree with fewer levels, monitoring data from all the nodes in the tree can reach the root node in a short amount of time. Therefore, we observe that the malicious nodes show their effect on the monitoring tree with $bF = 4$ quicker than with $bF = 2$.

Figure 6.3(b) shows the effect of the different branching factors on a network polluted with FalseLocalData attackers. As we observe, the fake maximum value of 50 is reported by the root node with a minor delay in both cases. However, in our test with $bF = 4$ the fake maximum value is reported from the very beginning of the monitoring phase. This type of situation arises when the root node or a node close to the root of the tree behaves maliciously.

The result of mean aggregation with branching factor of 2 and 4, in Figure 6.3(c), shows the stronger impact of the FalseLocalData attack by the branching factor of 4. However we observe, per Figure 6.3(d), an increase in the impact of FalseChildData attack by decreasing the branching factor due to the increase in the number of leaf nodes. Therefore the higher the number of leaves in the tree, the greater is the probability of FalseChildData attackers located in a tree as the leaf nodes. As the leaf nodes have no children, FalseChildData malicious nodes behave in the same manner in the network as the non-malicious ones.

According to our observations, a lower branching factor results in an increase



(a) Node Count (FalseParentPeer)

(b) Maximum Aggregation (FalseLocalData)

(c) Mean Aggregation (FalseLocalData)

(d) Mean Aggregation (FalseChildData)

Figure 6.3: Impact of the Branching Factor Variations

in the adaptation delay. In addition, by increasing the branching factor, the probability of having malicious nodes close to the root of the tree is increased. The malicious nodes closer to the root node are able to falsify the global monitoring data, aggregated by the root node, in a shorter time.

6.2.3 Scenario C: Various Ratios of Malicious Nodes

For the purpose of evaluating the effect of various ratios of malicious nodes, we simulate and compare three ratios of malicious nodes in the network; each polluted with one type of our defined attackers. For each attack, we test the ratios of 5%, 10% and 20% of malicious nodes with a fixed branching factor of 4.

Figure 6.4(a) clearly demonstrates that, by increasing the ratio of the attackers, the mean aggregation is affected by the malicious nodes to a higher extent. The presence of the 20% of FalseLocalData attackers means that every fifth node propagates the false value of 50 as its local monitoring data. Therefore, since the mean aggregation by non-malicious nodes oscillates between 0 and

1, the result of the global monitoring aggregation for the mean aggregation is around $50/5$. Likewise, by having 10% of attackers, every 10th node is malicious and we further observe that the mean aggregation oscillates around the $50/10$. In the network with 5% of FalseLocalData attackers, the mean aggregation oscillates around the value of 2.5. As anticipated, the effect of FalseLocalData and FalseChildData attacks on the sum aggregation, presented in Figure 6.4(b) and 6.4(d), is similar to their effect on the mean aggregation, presented in Figure 6.4(a) and 6.4(c).

According to the most obvious observation, Figure 6.4(c) shows a smaller peak value of the mean aggregation for the FalseChildData attack in comparison to the FalseLocalData attack, presented in Figure 6.4(a). As mentioned earlier, the reason for this is that more than 50% of the nodes in the monitoring tree are the leaves and have no children. The FalseChildData attack shows an increase in the peak value of the mean aggregation from 1, in the non-malicious network, to 3 in the network with 5% of the FalseChildData attackers. By increasing the ratio of the present attackers to 10%, the mean aggregation increases four times more than the desired value, a value around 4. Where the ratio is increased by 20%, the measured value reaches a value around 6.5.

In contrast to FalseLocalData and FalseChildData attacks, Figure 6.4(e) shows that by having 5% of FalseParentPeer attackers, the mean aggregation turns to the constant value of 0.5 after 15 minutes. The increase in the ratio of the malicious nodes decreases this period to 8 minutes. However, we observe no further decrease for this period where the ratio of attackers is increased from 10% to 20%. Also, increasing the ratio of the attackers, from 5% to 10%, leads to an increase in the mean aggregation value from 0.5 to 0.7 in their constant intervals. However, an increase in the malicious ratio to 20% does not show any further obvious changes.

Figure 6.4(f) depicts that the sum aggregation starts to increase continuously in the first minutes of the monitoring phase. Additionally, based on Figure 6.3(d), in the previous section, we also observed that FalseParentPeer attack has the same effect on the node count aggregation. Therefore, by taking both observations into account, we can explain the convergence of the mean aggregation result, calculated through dividing the sum by the count aggregation, to a constant value where FalseParentPeer attackers are present.

6.2.4 Scenario D: Churn Impact

In this section, we enable churn in order to compare the monitoring state of a robust network with a network in which peers leave and join the overlay

continuously in a random manner. For simulating the churn behavior, we utilize the exponential churn algorithm with the mean session length of 5 minutes. For creating a better overview of the churn effect, we delay the starting time of churn until minute 120. Therefore, the monitoring mechanism begins from minute 90, thus allowing for 60 minutes after joining and a further 30 minutes for the stabilization phase, and continues until the minute 120 without churn. Afterward, the churn is enabled until the end of the simulation at minute 180. So the monitoring works for 30 minutes without churn and then for 60 minutes with it.

Firstly, we study the effect of churn on a network without any malicious nodes. Figure 6.5(a) shows the node count aggregation stays mostly on zero where churn is enabled while the desired value is around 4000. The reason is that whenever a node leaves the tree, all of its descendants need to re-position themselves in the tree and also reset their monitoring data. Leaving the nodes in the higher levels of the monitoring tree, closer to the root node, results in the re-positioning of more nodes in comparison with leaving the nodes in the lower levels.

In the event of the failure of a child, its parent continues to retain the monitoring data from the absent child until the next update interval and keeps receiving data from its other children as well. In case the re-positioning of the sub-tree, affected by the absent node, happens in a shorter amount of time than the update interval, the parent will again receive the monitoring data from the newly repositioned nodes. As a result of this aggregation, we observe intervals like the last 4 minutes of the simulation in which the reported node count aggregation is greater than the desired value.

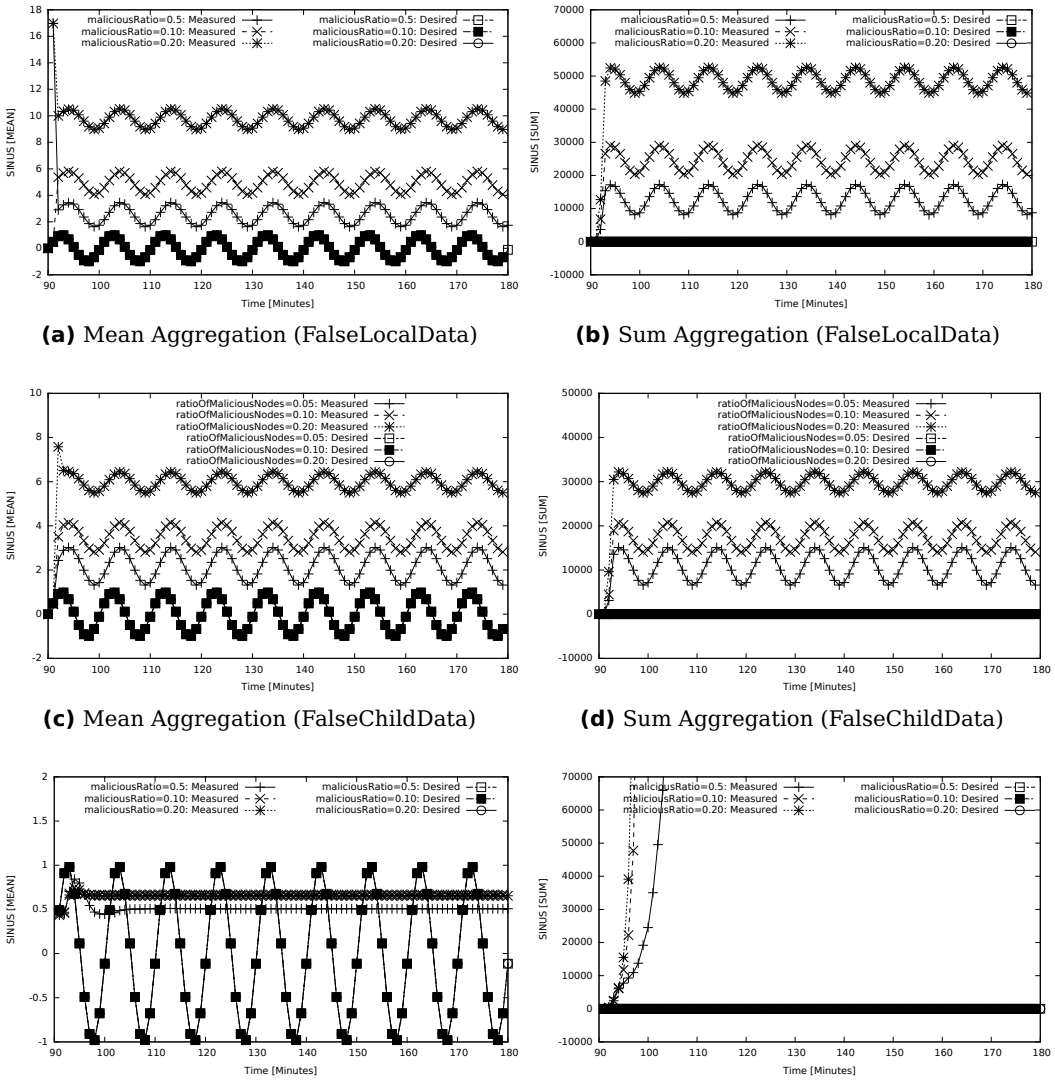
Since FalseLocalData and FalseChildData attacks cause no effect on the count value, we do not study them any further. Nevertheless, we observe the lack of precision in the mean aggregation in those two attacks which are depicted in Figure 6.5(c) and 6.5(e).

Figure 6.5(b) shows an imprecise maximum aggregation result under the impact of churn. We observe a continuous wrong value, reported by the root node, for several minutes in some intervals like during the minutes 100 to 110. This is due to the lack of updated data reaching the root node.

The absence of fresh data, in addition to the false value reported by the malicious nodes of type FalseLocalData and FalseChildData, can be observed in the maximum aggregation depicted in Figure 6.5(d) and 6.5(f). The same lack of precision, as well as, the fake maximum value of 50 is observed in the both cases.

The evaluation of churn effect on the networks polluted with FalseParentPeer

attackers in Figure 6.5(g) shows a more severe effect on the node count. As it is depicted, the reported node count aggregation by the root node, is either 0 or a much greater value than the desired number. In Figure 6.5(h) the convergence of the mean aggregation to a constant value can be observed. Furthermore, in the intervals where the count aggregation falls below the desired value, as a result of churn, we observe the fall of the mean aggregation too. Examples of this effect can be seen around the minute 140 or during the last minutes of the simulation.



(a) Mean Aggregation (FalseLocalData)

(b) Sum Aggregation (FalseLocalData)

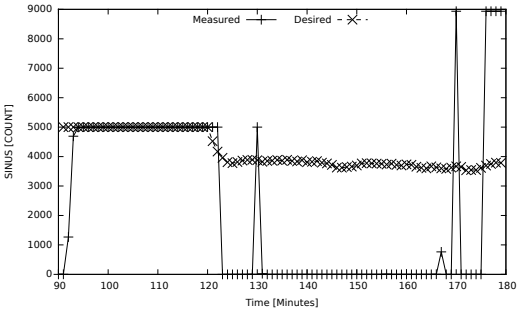
(c) Mean Aggregation (FalseChildData)

(d) Sum Aggregation (FalseChildData)

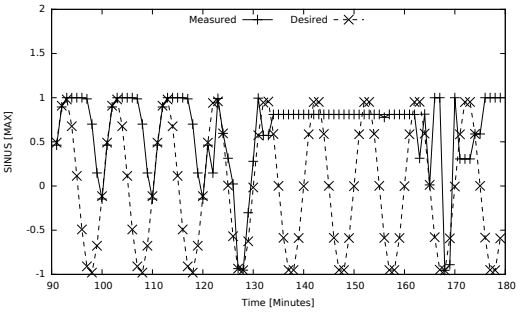
(e) Mean Aggregation (FalseParentPeer)

(f) Sum Aggregation (FalseParentPeer)

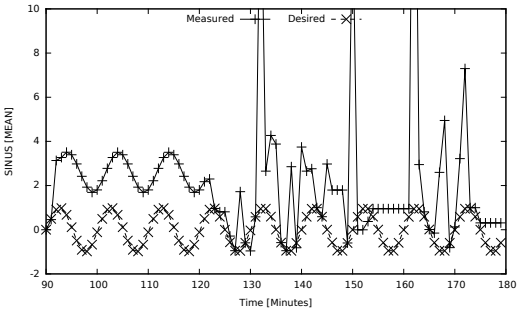
Figure 6.4: Comparing the Different Ratios of the Malicious Nodes



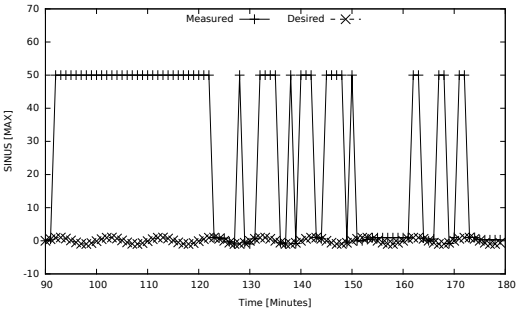
(a) Node Count (No Malicious)



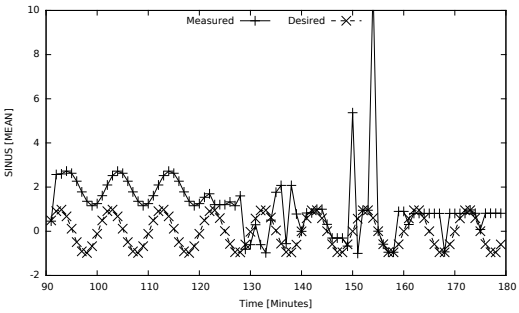
(b) Maximum Aggregation (No Malicious)



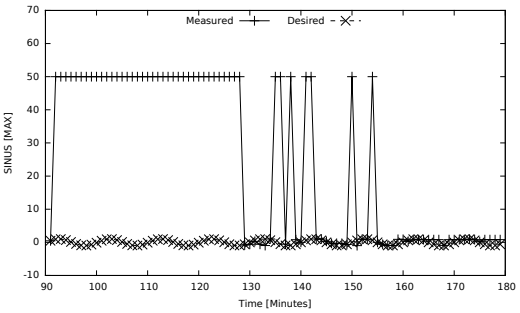
(c) Mean Aggregation (FalseLocalData)



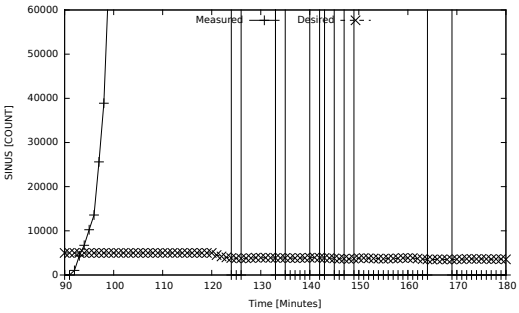
(d) Maximum Aggregation (FalseLocalData)



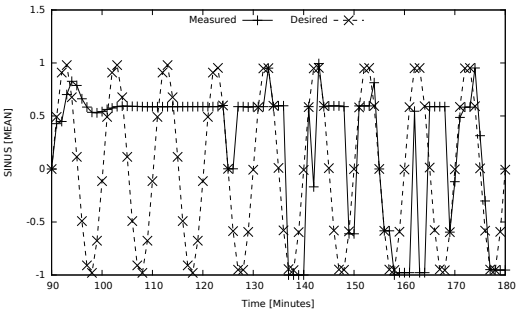
(e) Mean Aggregation (FalseChildData)



(f) Maximum Aggregation (FalseChildData)



(g) Node Count (FalseParentPeer)



(h) Mean Aggregation (FalseParentPeer)

Figure 6.5: Attacks with Churn

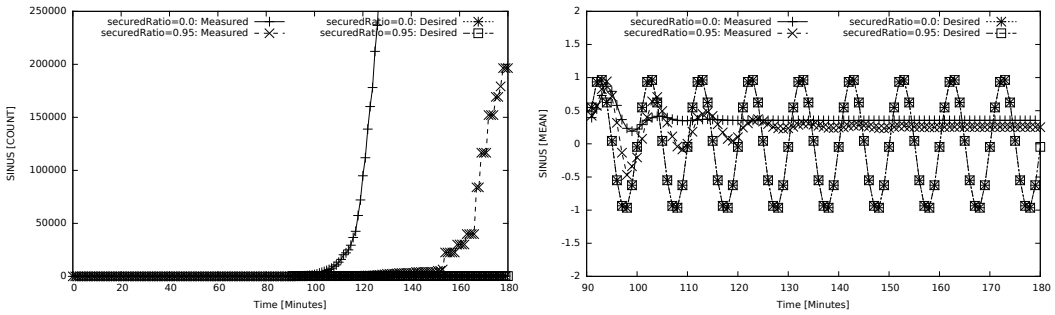
6.3 Evaluation of the Security Solution

In this part, we evaluate the effectiveness of our proposed solution against FalseParentPeer attack, introduced in Section 4, through three scenarios. In each scenario, we examine the secured nodes with one of the proposed acceptable ranges, on top of the Chord overlay. In Section 6.3.1, the examined acceptable ranges are around MPs and to the size of $bFS/2$ on each side, while the acceptable ranges in Section 6.3.2 are only on the right-hand side of MPs and to the size of bFS. The most limited acceptable ranges, which are still defined on the right side of MPs but to the size of $bFS/2$, are evaluated in Section 6.3.3.

6.3.1 Scenario E: Acceptable Ranges Around MPs

In this scenario, we evaluate the secured nodes with acceptable ranges located on both sides of MPs and to the size of $bFS/2$ on each side, as described earlier in Section 4.3.1. We simulate 300 nodes, including 5% of FalseParentPeer attackers. Figure 6.6(a), shows the result of node count with the introduced acceptable ranges for secured nodes. The result indicates a significant reduction in the number of the accepted nodes in the network with 95% of secured nodes (securedRatio = 0.95) in comparison to the network with unsecured nodes (securedRatio=0.0).

In the network with no security measurement, we observe a sharp increase in the number of the node count aggregation from the early minutes of the monitoring phase. In contrast, we observe a considerable decrease in the node count aggregation, where the nodes only accept received monitoring data within their designated acceptable ranges.



(a) Node Count **(b)** Mean Aggregation

Figure 6.6: Results of the Acceptable Ranges around MPs

While the node count aggregation in the unsecured network increases rapidly from the beginning of the monitoring phase, the secured network shows an increase at a much lower rate, and in a step-wise manner. It should be noted that, although the secured nodes are able to filter many of the malicious nodes, the final reported number of node count, 200000 nodes in this test, is still much greater than the desired value of 300 nodes.

Figure 6.6(b) shows the result of the mean aggregation function. It reveals the direct effect of the enormous increase in the node count on the mean aggregation result. However, the secured network provides a relatively better result than the unsecured network for the first 40 minutes of the monitoring phase. Yet, after 40 minutes, the mean value in both secured and unsecured network converges toward a constant value.

This is evidenced by the potential loops in the monitoring tree, created by the FalseParentPeer attackers, which result in a dramatic increase in the aggregated value at malicious nodes. Eventually, when a malicious node falls within the acceptable range of a non-malicious node, its huge aggregated result is appended to the network global monitoring aggregation.

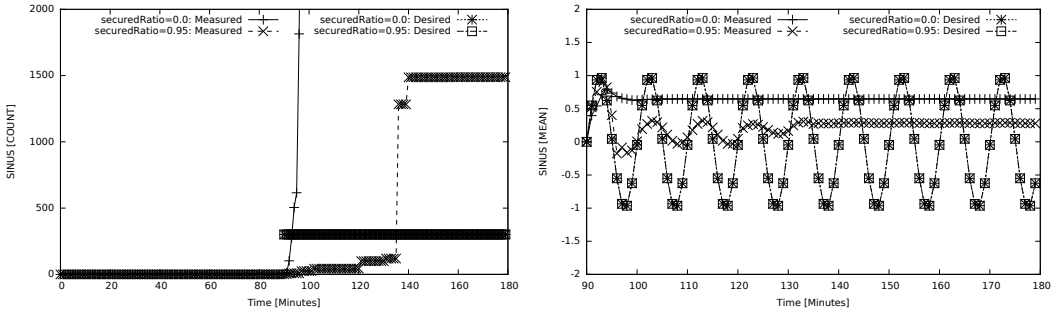
6.3.2 Scenario F: Acceptable Ranges Based on Chord

In this scenario, we evaluate the impact of halving the acceptable ranges tested in the previous scenario. Considering Chord as the overlay, the secured nodes in this scenario accept messages from the senders with the tree IDs within their acceptable ranges located on the right-hand side of MPs and to the size of $bFS/2$.

Figure 6.7(a) shows a significant increase in the node count aggregation when it reaches the constant value of 1500 nodes in its peak. Hence, the reduction of the reported node count value can be the result of detecting more malicious nodes. However, reducing the size of the acceptable ranges can also lead to a situation where parents omit their legitimate child nodes. The mean value in Figure 6.7(b) shows a slight improvement comparing to the previous case, but again it converges toward a constant value from half way through the monitoring phase.

6.3.3 Scenario G: Extended Acceptable Ranges Based on Chord

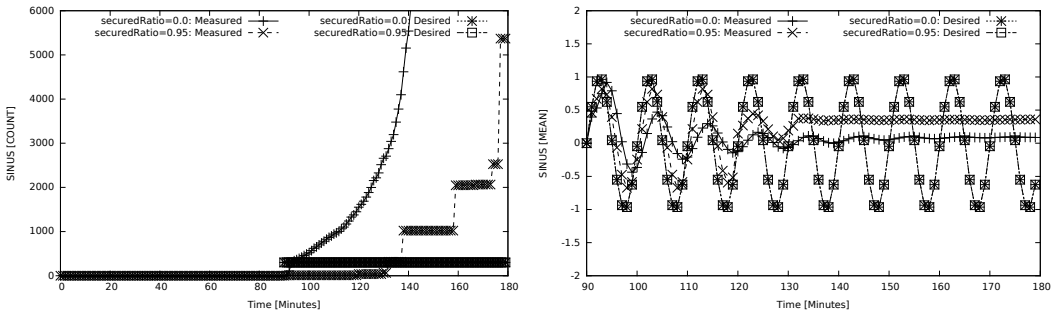
Finally, we evaluate our last proposed acceptable ranges, which are on the right side of MPs and to the size of bFS. In Figure 6.8(a), we observe that the node



(a) Node Count **(b)** Mean Aggregation

Figure 6.7: Results of the Acceptable Ranges on the Right-Hand Side of MPs Based on Chord

count reaches the number of 6000 with fewer steps while increasing, compared with the first acceptable ranges tested in Section 6.3.2. Figure 6.8(b) shows a better precision for the mean aggregation when compared with the two other acceptable ranges tested in the previous scenarios. However, as a result of the enormous increase in the node count aggregation, similar to the previous cases, the mean aggregation converges toward a constant value from half through the monitoring phase.



(a) Node Count **(b)** Mean Aggregation

Figure 6.8: Results of the Extended Acceptable Ranges on the Right-Hand Side of MPs Based on Chord

Since, in our simulation, the monitoring tree is implemented on top of the Chord overlay, the expected acceptable ranges on the right-hand side of MPs produce slightly better results although they are not satisfactory enough.

All in all, finding the proper acceptable ranges are not easily achievable. In addition the relatively larger acceptable ranges for the nodes, located on the higher levels of the monitoring tree, increase the probability of accepting

malicious nodes by the parents in higher levels in comparison to the ones in the lower levels. The other challenge in this approach is the increase in the probability of ignoring legitimate children by decreasing the size of acceptable ranges.

In this section, we evaluated our proposed solution for acceptable ranges. The results showed that employing this approach, as a single counter-measure solution, is not sufficiently effective for a long period and further study and additional solutions are needed. The next section provides a summary to this chapter.

6.4 Summary

In this chapter, we evaluated the defined attacks and our proposed counter-measure in seven different simulation scenarios. The evaluation of the defined attacks revealed that the FalseParentPeer is the most destructive type of attack among our three types of monitoring attacks. Furthermore, Between the FalseChildData and FalseLocalData attacks, FalseChildData attackers showed a lesser impact on the monitoring result due to the fact that more than half of the nodes in the monitoring tree are the leaf nodes with no children.

Our examination of the different branching factors showed that by increasing the branching factor the adaptation delay is reduced. We observed the impact of the malicious nodes, with lesser delay, in the monitoring trees with $bF = 4$, in comparison to $bF = 2$. On the other hand, by increasing the branching factor, there will be more nodes on each level of the tree and consequently it is more likely that there are malicious nodes close to the root of the monitoring tree. Bearing in mind that the root node is responsible for aggregating and publishing the global monitoring data, malicious nodes, located closer to the root node, affect the global monitoring data more quickly.

The evaluation results of the various ratios of malicious nodes showed an increase in the impact of the malicious nodes when the number of malicious nodes in the network is increased. Furthermore, By evaluating the attacks in the churn enabled networks, we noticed the high vulnerability of the monitoring mechanism even in the networks with no malicious nodes. The reason behind this vulnerability is that once a node fails other nodes change their position in the tree and the monitoring tree is restructured.

The evaluation of the counter-measure against FalseParentPeer attack revealed that our proposed solution could not make the network resistant against the

attackers for a long duration since, when a malicious node locates in the acceptable range of a node in the monitoring tree, it dramatically affects the network. Additionally, predicting the child node's tree IDs is not easily achievable. However, the evaluation of the secured nodes with acceptable ranges based on Chord showed, using this approach, by taking the overlay into account the accuracy of monitoring result improves.

7 Conclusions and Outlook

Contents

We conclude our work in this final chapter and suggest some further research areas with regard to this thesis. In Section 7.1, we provide a summary of this document and draw our conclusions. Finally, Section 7.2 gives an outlook beyond the scope of this thesis and outlines possible enhancements to our work.

7.1 Conclusions

Monitoring of p2p systems is a challenging task due to the absence of a central point for collecting the monitoring data and the unquantifiable number of nodes that autonomously join and leave the network. In addition, the possibility of the presence of malicious nodes adds an extra level of complexity to the challenges in the way of capturing the status and performance of p2p networks. In this thesis, we aimed to analyze the impact of different malicious nodes on the monitoring mechanism offered by SkyEye.KOM and find solutions against different types of attacks.

In Chapter 1, we introduced the peer-to-peer paradigm as an alternative solution for sharing resources in distributed systems. Furthermore, we explained the important role of the monitoring systems for providing a controlled level of the Quality of Service in networks. In the first chapter, we stated the motivation for and the goals of this thesis and then in Chapter 2 we studied the p2p networks in more detail by reviewing the characteristics of structured and unstructured p2p overlays. PeerfactSim.KOM, the simulation framework that we utilized for our evaluation tasks in this work, was also introduced in the same

chapter. In Chapter 3 we discussed structured and unstructured monitoring types. In addition, we focused on the structured monitoring mechanism provided by SkyEye.KOM. We explained how the SkyEye.KOM monitoring tree is constructed as an over-overlay on top of the DHT-based p2p overlays and how the monitoring information is propagated and aggregated in the tree.

In Chapter 4, we introduced three possible security attacks against SkyEye.KOM monitoring system, named FalseLocalData, FalseChildData and FalseParentPeer, and for each attack we proposed our solutions. The aim of the FalseLocalData and FalseChildData attacks both was to provide false monitoring data for the parent nodes in the monitoring tree. While the strategy of FalseLocalData attackers to achieve this goal was producing fake local monitoring information, the plan of FalseChildData attackers was to manipulate the received monitoring data from their child nodes before aggregating it with their local monitoring information.

As a solution, we proposed our scoring system based on the statistical concepts such as the k-means clustering and the coefficient of variation. We employed those concepts to assist the parent nodes to judge the validity of the received monitoring data from their children. We also suggested different reactions to malicious child nodes after the detection phase besides the ignoring strategy.

The Last defined attack, FalseParentPeer, was designed to violate the expected flow of the monitoring data in the tree. In this attack, each malicious node forwards the monitoring data to a random node other than its parent node which is the intended destination. Our idea to mitigate the effect of the FalseParentPeer attack was to assist the parent nodes to predict the ranges of their children IDs and only accept the monitoring data from the nodes with IDs within those predicted ranges. Considering the overlay, we suggested the further limiting of those acceptable ranges in order to filter a larger number of malicious nodes.

We implemented the three defined types of attacks alongside our countermeasure against FalseParentPeer attack and, in Chapter 5, we explained the details of the implementation phase of the thesis. The evaluation of the attacks, per Chapter 6, shows that FalseParentPeer attack has more destructive potential than the other two types of attacks. Additionally, FalseChildData attack showed a lesser impact than FalseLocalData attack on the monitoring result due to the fact that more than half of the nodes in the monitoring tree are the leaf nodes with no children. Besides, we analyzed the impact of changing the branching factor of the monitoring tree as well as the effect of the different ratios of the malicious nodes.

We observed the vulnerability of the monitoring system in the churn enabled networks and, by including the malicious nodes in those networks, a dramatic decrease in the accuracy of the monitoring results was recorded. The evaluation of the counter strategy against FalseParentPeer attack revealed that predicting the ID range of the child nodes is hardly achievable and, even by defining proper limited acceptable ranges, once a malicious node positions itself in the acceptable range of a non-malicious node it will be able to harm the system.

7.2 Outlook

Evaluation of the proposed scoring system can be considered as the next step. Also, the scoring system can be further extended by appending more checks and comparisons strategies. In addition, the possibility of receiving misleading typical values from the sibling's children needs to be considered. The introduced security solution against FalseParentPeer attack requires a greater amount of research and improvement. For example, our solution could be improved by allowing the parent nodes to dynamically define their acceptable ranges by reference to the count value reported by the child nodes. Using this approach a greater count value, reported by a child node, results in smaller acceptable ranges by its parent node.

A future study may also examine networks in situations where multiple types of attack occur at the same time. Furthermore, the definition of the currently defined malicious behaviors could be extended and new types of attack could be introduced. Last but not least, due to the frequency of the nodes leaving and rejoining in real p2p networks, particular attention will need to be paid to the impact of churn on the monitoring system.

Bibliography

[1] *BitTorrent*. – http://www.bittorrent.com

[2] *FastTrack*. – http://en.wikipedia.org/wiki/FastTrack

[3] *Skype - Peer-to-Peer Internet Telephony*. – http://www.skype.com

[4] Brown, CharlesE.: Coefficient of Variation. In: *Applied Multivariate Statistics in Geohydrology and Related Sciences*. Springer Berlin Heidelberg, 1998, S. 155–157. – URL http://dx.doi.org/10.1007/978-3-642-80328-4_13. – ISBN 978-3-642-80330-7

[5] Buford, John ; Yu, Heather ; Lua, Eng K.: *P2P networking and applications*. Morgan Kaufmann, 2009

[6] Carlsson, Bengt ; Gustavsson, Rune: The rise and fall of napster-an evolutionary approach. In: *Active Media Technology*. Springer, 2001, S. 347–354

[7] Cerf, Vinton G. ; Icahn, Robert E.: Transmission control protocol. (1981). – URL http://tools.ietf.org/html/rfc793

[8] Darlagiannis, Vasilios ; Mauthe, Andreas ; Steinmetz, Ralf: Overlay Design Mechanisms for Heterogeneous Large-Scale Dynamic P2P Systems. In: *Journal of Network and Systems Management* 12 (2004), S. 371–395

[9] Feldotto, M. ; Graffi, K.: Comparative evaluation of peer-to-peer systems using PeerfactSim.KOM. In: *High Performance Computing and Simulation (HPCS), 2013 International Conference on*, July 2013, S. 99–106

[10] Giesen, Philipp: *Systematic Benchmarking of Monitoring Protocols in Distributed Systems*. Master thesis, Heinrich-Heine-Universität Düsseldorf. 2014

[11] Gilder, George: *Telecosm: How infinite bandwidth will revolutionize our world*. Simon and Schuster, 2000

[12] Gnutella: *The Annotated Gnutella Protocol Specification v0.4*. 2002. – URL http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[13] Graffi, Kalman: *Monitoring and Management of Peer-To-Peer Systems*. PhD thesis, Technische Universität Darmstadt. 2010

[14] Gummadi, Krishna P. ; Dunn, Richard J. ; Saroiu, Stefan ; Gribble, Steven D. ; Levy, Henry M. ; Zahorjan, John: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: *ACM SIGOPS Operating Systems Review* Bd. 37 ACM (Veranst.), 2003, S. 314–329

[15] Holpuch, Amanda: *Netflix and YouTube make up majority of US internet traffic, new report shows*. – URL http://www.theguardian.com/technology/2013/nov/11/netflix-youtube-dominate-us-internet-traffic

[16] Jelasity, MÃirk ; Montresor, Alberto ; Babaoglu, Ozalp: Gossip-based aggregation in large dynamic networks. In: *ACM Transactions on Computer Systems* 23 (2005), S. 219–252

[17] Kempe, David ; Dobra, Alin ; Gehrke, Johannes: Gossip-Based Computation of Aggregate Information. In: *IEEE Symposium on Foundations of Computer Science*, 2003, S. 482–491

[18] Klingberg, T. ; Manfredi, R.: *Gnutella 0.6*. 2002. – URL http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[19] Kovacevic, Aleksandra ; Liebau, Nicolas ; Steinmetz, Ralf: Globase.KOM - A P2P Overlay for Fully Retrievable Location-based Search. In: *Peer-to-Peer Computing*, 2007, S. 87–96

[20] Li, Xiaokun ; Chen, Genshe ; Blasch, Erik ; Pham, Khanh: Detecting missile-like flying target from a distance in sequence images. In: *SPIE Defense and Security Symposium* International Society for Optics and Photonics (Veranst.), 2008, S. 69680G–69680G

[21] Maymounkov, Petar ; Mazieres, David: *Kademlia: A peer-to-peer information system based on the XOR metric*. 2002. – 53–65 S

[22] Mehta, Dinesh P.: *Handbook of data structures and applications*. CRC Press, 2004

[23] Moore, Gordon E. u. a.: *Cramming more components onto integrated circuits*. 1965

[24] Ng, T. S. E. ; Zhang, Hui: Global network positioning: a new approach to network distance prediction. In: *Computer Communication Review* 32 (2002), S. 61–61

[25] Ng, T. S. E. ; Zhang, Hui: Predicting Internet Network Distance with Coordinates-Based Approaches. In: *IEEE INFOCOM* Bd. 1, 2002

[26] Postel, Jon: User datagram protocol. In: *Isi* (1980). – URL http://tools.ietf.org/html/rfc768

[27] Rowstron, Antony I. T. ; Druschel, Peter: *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*. 2001. – 329–350 S

[28] Schmitt, Jens ; Wolf, Lars: Quality of Service - An Overview / Darmstadt University of Technology. April 1997 (TR-KOM-1997-01). – Forschungsbericht

[29] Stoica, Ion ; Morris, Robert ; Karger, David ; Kaashoek, M. F. ; Balakrishnan, Hari: Chord: A scalable peer-to-peer lookup service for internet applications. In: *Computer Communication Review* 31 (2001), S. 149–160

[30] Zadeh, Lotfi A.: Fuzzy sets. In: *Information and control* 8 (1965), Nr. 3, S. 338–353

List of Figures

List of Tables