Detection of Malicious Behavior in Structured P2P Monitoring Systems

Master Thesis

by

Sebastian Brink

born in
Haan

submitted to

Technology of Social Networks Lab
Jun.-Prof. Dr.-Ing. Kalman Graffi
Heinrich-Heine-Universität Düsseldorf

December 2015

Supervisor:
Jun.-Prof. Dr.-Ing. Kalman Graffi
Prof. Dr. Martin Mauve

Abstract

Peer-to-peer (P2P) networks represent an decentralized and scaling alternative to client-server architectures. Due to their decentralization, monitoring systems are necessary to gain detailed information about the network's status and manage these networks. As every peer contributes information to the monitoring system, malicious peers are able to influence the results respectively procedure in various ways.

This thesis defines different possible attacks against the tree-based monitoring system SkyEye.KOM, a monitoring system for structured P2P networks. We introduce several evaluation criteria to determine whether we are communicating with a malicious peer or not. We bundle these criteria into a rating system called DOMiNo which takes each criteria into consideration to evaluate other peers. DOMiNo aims to detect malicious peers precisely without introducing new possible attack paths and without introducing new overhead to the network. Additionally its computational costs are kept as low as possible. Furthermore, we also introduce tweaks for SkyEye.KOM to be able detect different attacks.

DOMiNo and the tweaks, along with the defined attacks against SkyEye.KOM, were implemented in the P2P simulation framework PeerfactSim.KOM. Several attack scenarios were simulated and evaluated, showing that our approach provides very good detection results in cases where peers do not behave according to the message forwarding algorithm defined by SkyEye.KOM.
Scenarios with manipulated information still need further investigation, especially when using different data than the used sinus function. The evaluation shows several problems regarding the used data base which may not be present using real network status data. At last, considering our achieved results, we suggest further possible work to improve the rating system.

Contents

List of Figures

List of Tables

Chapter 1

Motivation

Peer-to-peer (P2P) networks offer many advantages compared to client-server network models. They are able to build networks using only the capacities of every single peer participating in the network. Due to the decentralization, applications, e.g. file sharing applications, can be realized without expensive server equipment and especially without a peer having superior rights. However, as no central instance is present it is hard to manage these networks to offer a certain Quality of Service. Monitoring systems present a solution to gain information about the status of the network so peers may adapt several parameters to improve the overall performance (e.g. reducing the output during high package loss or increasing their routing tables' size in unstable networks). In particular, structured monitoring systems provide highly precise monitoring results by building a topology to aggregate and distribute data effectively. Since every peer is responsible for the network and has to contribute its known information to the monitoring system to gain a profound view on the underlying P2P overlay, new attack possibilities arise. Even more as no trustworthy instance, i.e. a server, is present.

Attacks against monitoring systems can be motivated by different point of views. On the one hand attackers may want to benefit from the P2P overlay without contributing much themselves. On the other hand attackers may want to modify specific monitoring results or prevent specific nodes from participating in the monitoring system. Some attackers may even just want to *have some fun* by starting attacks without deeper meaning. It gets particularly critical if the monitoring mechanism is used in a usually trustworthy context, e.g. a company network, and parts of the network get compromised.

As an example the Department of Computer Science from the University of Oldenburg [Old] uses a constructed communication overlay to exchange data between several power plants which monitor each other. If one of those power plants gets compromised the results could be fatal. Power plants getting compromised is no fictional example. The computer worm Stuxnet [NF11] already demonstrated that even these highly restricted areas are not completely safe from attacks. While power plants are one instance where misbehaving participants are particularly crucial, malicious behavior is generally not desired and therefore detection techniques are necessary.

Another realization of such a monitoring approach is SkyEye.KOM [GKXS08] which builds a structured tree-based monitoring system independently of the underlying overlay. This thesis will define different possible attacks against it and present a possible solution to detect malicious behavior.

1.1 Objectives

From a previous thesis done by Payman Alavi [Ala15] it is known that attacks against monitoring systems are possible. We want to use and extend this knowledge and define the aims of this thesis as follows:

Define Attack Scenarios As we know that attacks can be launched against monitoring systems like SkyEye.KOM we want to define and to classify them according to their behavior and influence on the monitoring system.

Discuss and Develop Possible Detection Mechanisms As an attacker may launch his attack against different parts of the monitoring system, we want to discuss and develop different options to identify an occurring attack regardless of the used attack type.

Implement the Attacks and Detection Mechanisms Furthermore we do not only want to develop different options, but also implement them into an application, which we can work with later on. Our implementation should be able to detect attacks against the system with the highest precision possible, i.e. high positive and low false positive rates.

Evaluate the Pros and Cons We want to determine how good our solution works and where we need to add further effort to improve our solution. If this is not possible we want to determine the deciding factors which prevent us to improve it.

1.2 Thesis Structure

The remainder of this thesis is structured as follows:
Essential knowledge, definitions and assumptions regarding this thesis are given in Chapter 2. Additionally, Chapter 3 supplements Chapter 2 with an introduction to outlier detection.

With the information provided in Chapter 2 and Chapter 3 our solution called DOMiNo, a rating system, is presented in Chapter 4. It will explain possible solutions against different attack types as well as remaining problems in detail.

In Chapter 5 we describe how we realized our solution and the different attacks against the used monitoring system in the simulation framework PeerfactSim.KOM [SGR$^+$11].

Chapter 6 shows and evaluates our accomplished detection results followed by the conclusion of our thesis in Chapter 7 which also gives an overview on possible future work.

Chapter 2

Essentials

In this chapter we give a short summary of SkyEye.KOM [GKXS08], a structured monitoring system, which we will work with in this thesis. Understanding the structure of SkyEye.KOM is mandatory as we will then discuss general thoughts regarding nodes that misbehave in such a structured monitoring system and categorize them in different attack types. Especially since SkyEye.KOM uses a tree structure in contrast to other possible structures, e.g. a mesh topology, which would offer other attack types due to different node relations, we need to know its behavior.

After that we define requirements we want to fulfill when detecting malicious behavior and discuss general problems. An important note is, that, as we will operate in a P2P overlay we need to develop a mechanism to identify malicious nodes without having information about the complete network, i.e. we need to work with only little information. How we get information and what information we get will be explained in the next section.

2.1 SkyEye.KOM

SkyEye.KOM is a structured monitoring system which can operate on any P2P overlay which offers key-based routing like Pastry [RD01] or Chord [SMK+01]. To operate independent from specific overlays SkyEye.KOM introduces its own ID space S_{ID}. It maps a peer's ID p of the overlay's ID space O_{ID}, to its ID space $S_{ID} \in [0,1]$ by using a deterministic function m

$$m : O_{ID} \rightarrow S_{ID}, m(p) = \frac{O_{ID}^{p}}{max(O_{ID})} \tag{2.1}$$

As its topology, it uses a tree topology in which parent nodes only communicate with their child nodes respectively their own parent node, thus reducing the message overhead in the network and letting information pass from one tree level to the next one. For each level l in the tree, the ID space S_{ID} will be divided into bF equidistant, non-overlapping areas whereas bF is the branching factor of the tree with $bF \in 2^n, n \in \mathbb{N}$. Each of these parts is a so called *Domain* D_l^p which represents a specific interval in the tree topology on level l containing the peer ID p. A *Coordinator* C_l^p is the associated node responsible for such a Domain. C_l^p is, if it is responsible for the corresponding Overlay ID of C_l^p the middle point of the correlating Domain. Figure 2.1 shows an example tree with a branching factor $bf = 2$. As long as a peer has not yet found the Coordinator ID it is responsible for, the peer calculates the Domain D_l^p it belongs to for each level l of the tree. If a peer has found a position it is responsible for, it stops the calculation, since only the Coordinator ID closest to the root is used by a peer. A peer can determine if it is the Coordinator for a specific Domain, by checking if it would be responsible for the correlating overlay ID belonging to the Coordinator ID. This can be calculated by using Equation (2.1) in reverse

$$m^{-1} : S_{ID} \rightarrow O_{ID}, m^{-1}(p) = S_{ID}^p \times max(O_{ID}) \tag{2.2}$$

and starting a lookup in the P2P overlay.

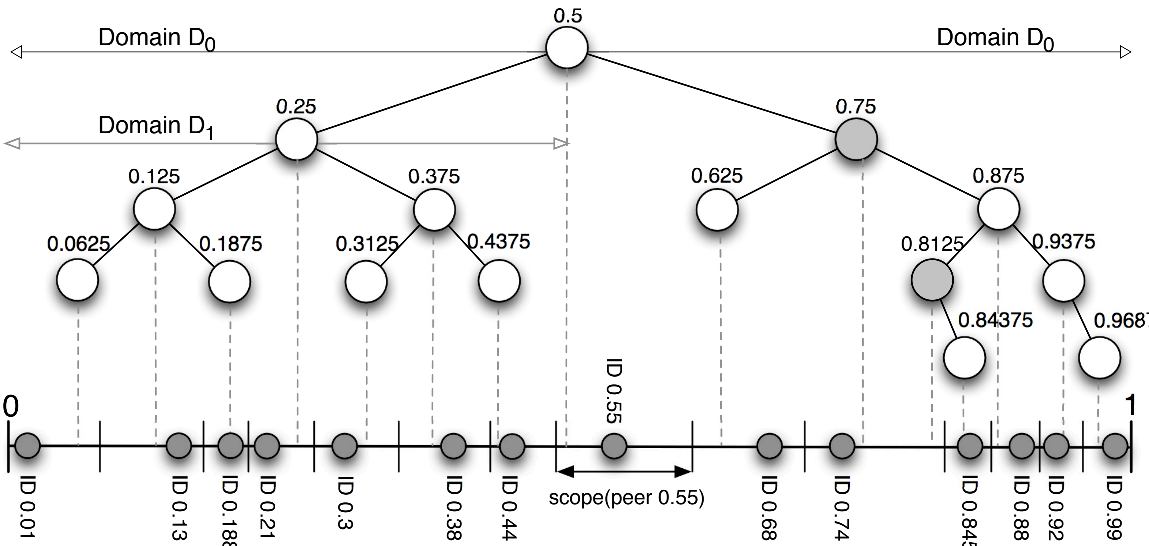


Figure 2.1: Example tree in SkyEye.KOM, bF = 2, provided by the authors of [GKXS08]

Note that, due to the fact that peer responsibilities in the overlay operate in intervals, a peer might be Coordinator on multiple levels of the tree, but the peer itself only recognizes its Domain closest to the root. Nodes provide information by sending periodically updates to the Coordinator one level above in the tree, i.e. C_{l-1}^p. The time between each update is called *update interval* and holds for each node in the tree. The information sent upwards contains not only the local data of the node itself but also the data from its child nodes aggregated to a single data set. If a Coordinator receives aggregated data[1] from a child node, the child node in return receives the current global view data[2] from the parent node via an acknowledgment, so that information not only goes up the tree, but also is spread from the root down the tree. The whole messaging process in SkyEye.KOM follows an asynchronous behavior as nodes start sending data upwards periodically once they join the monitoring overlay and not in a certain time slot. This behavior reduces spikes in the network traffic, but needs to be kept in mind while waiting for data. More details about SkyEye.KOM can be found in [GKXS08] and [Gra10].

For the remainder of the thesis we will use the words *high/er* and *low/er* respectively *up* and *down* in accordance with the depicted tree. I.e. a node lower in the tree is further away from the root node as a node higher in the tree. Additionally, we use the words *peer* and *node* synonymously.

In the next section we will define different attack types a malicious node can use to attack a structured tree-based monitoring system.

2.2 Attack Types

As mentioned in Chapter 1 malicious nodes may have different intentions to behave not in accordance with the monitoring algorithm. To be able to differentiate malicious behavior even further, we will define different attack types a malicious node is able to launch in the current implementation.

We distinguish the following attack types in two bigger groups which both have approaches to disturb the results of the structured monitoring system. The first group tries to influence the results in the system, but wants to keep the system running with manipulated data, e.g. by sending reduced data values to the child and parent nodes or by sending data to nodes not being a legitimate child or parent node. In contrast the second group tries to break links in the monitoring system and to exclude respectively eclipse specific nodes or whole sub trees from participation.

Figure 2.2 shows an example topology in SkyEye.KOM with branching factor = 2. Black lines denote relevant respectively *active* connections of an attack. I.e. data will be sent without any problems in the direction the arrow points. Dotted lines show connections which should be established following the monitoring algorithm, but will not be maintained as a result of an occurring attack. If marked with an arrow, a dotted line shows that one side tries to establish the connection unsuccessfully.
Gray lines show existing connections according to the SkyEye.KOM algorithm which are not affected by either attack depicted in the Figure.

[1]This is the data sent from the bottom of the tree to the top of the tree, i.e. the root.
[2]This is the data being sent from the top of the tree downwards towards the leaves.

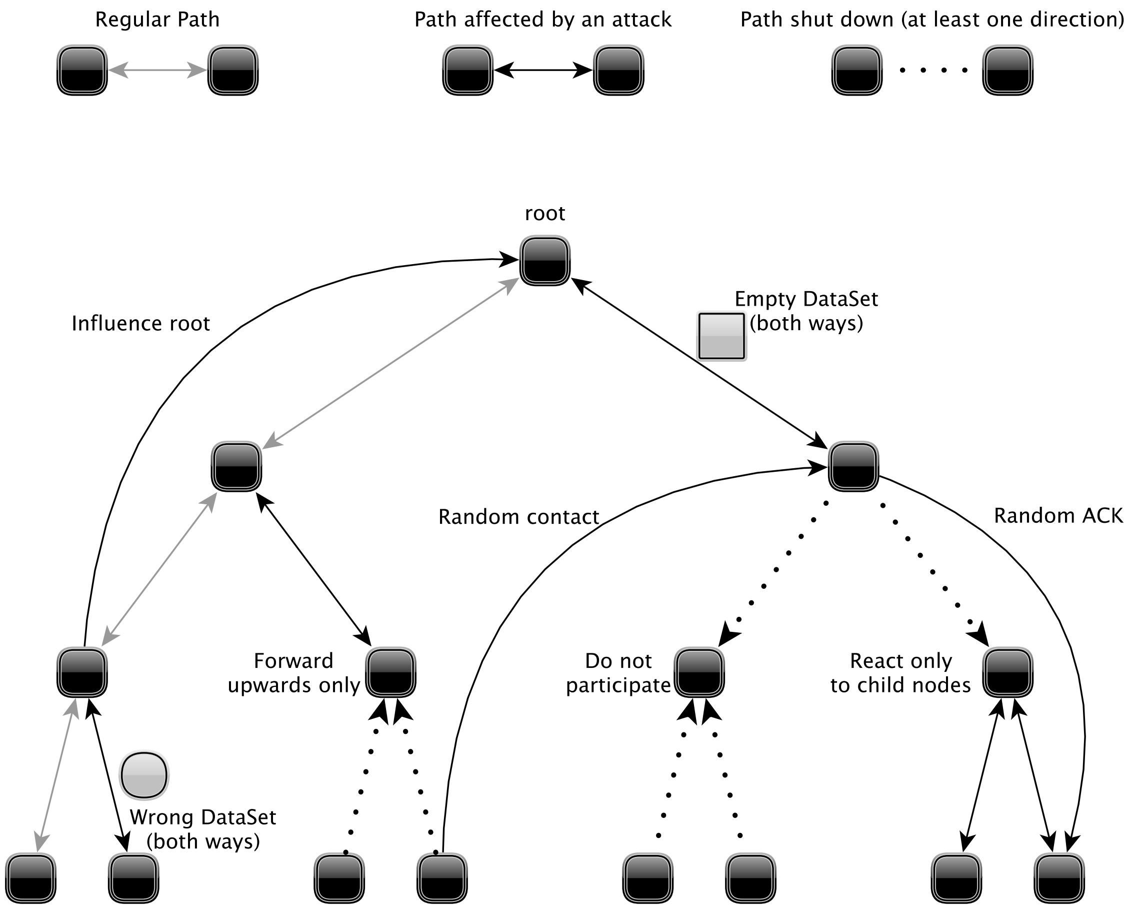

Regular Path
Path affected by an attack
Path shut down (at least one direction)
root
Influence root
Empty DataSet (both ways)
Random contact
Random ACK
Forward upwards only
Do not participate
React only to child nodes
Wrong DataSet (both ways)

Figure 2.2: Overview of attack types against the monitoring system

We will denote the different attack types via the schema $AT_{abbreviation-for-attack}$ for future references in this thesis.

2.2.1 Manipulation of Monitoring Results

This group contains malicious nodes which send data to other nodes to change the current view of them. With this they still keep the monitoring procedure running, but can influence the behavior of the whole network, e.g. by sending parent and child nodes the information that only little bandwidth is available in the corresponding sub tree. Another possibility would be to declare high available bandwidth which may lead to the failure of single nodes due to congestion or similar.

Next, we want to differentiate the following attacks in this group.

Send modified Data Sets to the Parent Node – AT_{Parent}

If a peer sends wrong data to its parent node it is able to modify the local view in the parent node and therefore such a peer has indirect influence on adjacent sub trees. This is due to the fact that the parent node will send the influenced local view up towards the root in the next iteration. At some point the root will get the modified data and uses it to build the global estimation which will be sent down the tree. Furthermore peers may present themselves as weak nodes to maximize their benefit from the running P2P overlay without providing too much resources themselves as many overlays have mechanisms to balance load under the peers according to their capabilities.

This attack type is best suited for having influence on the monitoring results while being as unobtrusively as possible since the node behaves *correct* regarding the message forwarding algorithm. I.e. the malicious node addresses the correct parent node.

Send modified Data Sets to the Child Node(s) – AT_{Child}

Peers can influence all their child nodes or only a specific child node by forwarding incorrect global data downwards the tree. With this they are able reduce the profit child nodes get from the P2P overlay. Child nodes may receive data that states that the network is in a weak condition so that these nodes reduce their demand regarding the P2P overlay (e.g. in a filesharing overlay). On the other hand if child nodes receive the information that the network capabilities are much higher than they really are, they may increase their demand and appear suspicious or malicious themselves to other nodes.

Malicious nodes using this attack type behave correctly, too, as they send information according to the monitoring procedure.

Send Data to a Random Contact – AT_{Random}

Sending data, regardless of if it is manipulated or not, to random nodes corresponds to sending messages upwards the tree and lets the sender appear as a child node of the receiver. As depicted in Figure 2.2 this attack can affect arbitrary nodes at any position. This is due to the fact that all data sets which are not marked with the `ACK` flag are treated as if the sender is further away from the root as the receiver and the sender is not verified by the receiver. With this all statistics that rely on aggregating values (e.g. the node count) will be forged as the receiving node has more information than it should have. Furthermore the number of child nodes may exceed a possible maximum, e.g. the branching factor in SkyEye.KOM if all possible child nodes are present, which may lead to further actions depending on the monitoring system. Such an action could be a forced reorganization of the monitoring system as the number of child nodes is obviously incorrect[3].

Send ACK to a Random Contact – $AT_{RandomACK}$

ACK messages are usually messages sent downwards the tree. Sending ACK messages will influence the global view in the receiving node since ACK messages are answers received from parent nodes presenting the current known status of the network. As these ACK messages are independent from those on the Transport Layer they do not belong to a specific message flow and will not be validated automatically, e.g. via sequence numbers. This attack type leads to the same results as sending wrong data to child nodes, but in different parts of the tree and even different tree levels.

Using this attack results in a more suspicious behavior as parent nodes are usually well known since child nodes initiate the communication between themselves and the parent.

Send Data to the Root directly – AT_{Root}

Data sent to the root node directly by a node that is not a legitimate child is a special case of the attack AT_{Random} where data is sent to a random contact. As previously mentioned, child nodes cannot be verified. The difference of this attack is the influence the malicious node gets (on the overlay), which is at a maximum with this attack due to the root using the information to spread the global view downwards the tree.

2.2.2 Shutting down of Monitoring Mechanisms

The following attack types aim to shut down specific mechanisms of the monitoring system or to seal off single nodes from it. Attackers may reach those aims by not participating regularly or sending packets without monitoring information. These attacks are more aggressive towards the concerned parts of the system as they do not provide any information at all.

[3]Note that this is just an example which is not present in SkyEye.KOM but may be in other monitoring systems.

Forward Empty Data Sets – AT_{Empty}

Forwarding empty data sets results in the erasure of parts of the gathered statistics if the forwarding is upwards to the root or even in the erasure of the whole statistics the child nodes gain if empty sets get forwarded downwards the tree. Attackers are able to shut down significant parts of the monitoring tree completely as the receivers have to work with sparse data or even no data if the attacker is the only connection to the other tree parts. In Figure 2.2 this would mean that if the root node uses this attack, the right sub tree would not be able to gain any global data according to the message forwarding algorithm.

Forward Data only upwards – $AT_{Upwards}$

If nodes forward data upwards without sending information back to their children they take advantage of the monitoring system without contributing much to it except the message they send towards the root and the contribution to the global view. Like the attack AT_{Empty} against child nodes, the sub tree of the malicious node will get no information at all and therefore be excluded from the monitoring system's results.

Only React to Child Nodes – $AT_{Downwards}$

With this attack, an attacker can exclude the whole sub tree from the monitoring mechanism without being suspicious instantly. Nodes in the sub tree will not be able to contribute to the global view in the system and may only receive information about the sub tree itself if the attacker decides to send back any information back to the child nodes.

Do not participate – $AT_{DoNothing}$

Missing participation in the monitoring system will result in broken links between multiple tree parts since the denying nodes are the proper parent respectively child nodes and thus need to be contacted as part of the monitoring algorithm. This behavior will both influence the monitoring system's results and shut off specific nodes from the system. It combines both attacks of $AT_{Upwards}$ and $AT_{Downwards}$.

2.2.3 Summary

In this section we want to give an overview of the previous information regarding the different aims of the attack types. For this we will assign each possible aim of an attacker with a possible attack type to accomplish it.

Possible Aim	Attack Types
Try to appear weaker to benefit from the system	AT_{Parent}
Disturb the system by spreading false information	AT_{Parent} AT_{Child} $AT_{RandomContact}$ $AT_{RandomAck}$ AT_{Root} AT_{Empty}
Disturb the system by sending information to the wrong receiver	$AT_{RandomContact}$ $AT_{RandomAck}$ AT_{Root}
Exclude or eclipse specific nodes from the monitoring mechanism	AT_{Empty} $AT_{Upwards}$ $AT_{Downwards}$ $AT_{DoNothing}$

Table 2.1: Summary of the presented attack aims against SkyEye.KOM

2.3 Requirements

Before building a system to detect malicious behavior we will define some general aims which we want to fulfill with our solution.

Precision – $R_{Precision}$ We want to maximize our precision regarding the results. This means, we are willed to sacrifice some positive findings if we can minimize the false positives rate with a different or modified approach.

Low Overhead – $R_{Overhead}$ We want to add as few overhead as possible to the existing P2P network. Our detection should not be realized by increasing the message size significantly or introducing new messages which take up a noticeable amount of the distributed messages.

Introduce No New Attack Paths – $R_{NoNewPaths}$ We do not want to introduce new possibilities for attackers to disturb the monitoring system. We need to note, that we could use additional paths in the monitoring system to gain more data from different sources, e.g. a node could ask another node from a different sub tree for its data. As a results we would be able to work with a larger data base. But additionally, with this, we would open new attack paths for malicious nodes. If the node we ask is a malicious node it could influence our known data base so that we would rate nodes from our sub tree suspicious. The same could happen if we ask our grandparent nodes or similar. A malicious grandparent node could state that our parent node is malicious by sending data which makes the parent's data implausible.

Performance – $R_{Performance}$ Detection mechanism should need as little effort as possible. Nodes should not be force to spend a big part of their computation time for detection purposes. We define this requirement as the detection should not be the main task of the whole system. Instead the detection should happen besides the occurring other applications.

2.4 Discussion

The detection of malicious nodes can be seen as a classification problem since we want to separate decent from malicious nodes. The problem is that not all classification mechanisms are equally well suited. If we use partitioning clustering algorithms to determine the members of each node type, these algorithms do not necessary behave well when only one type of node is present. Some clustering algorithms always build the postulated number of clusters. E.g. if we claim that there are two types of nodes present, such a clustering algorithm will always distinguish all present nodes into two clusters, even if they all would belong into the same cluster. Other clustering algorithms may distinguish nodes of the same type into several clusters, although we do not want this behavior.
A better way to tackle the problem is to use mechanisms which assume that only one big cluster is present and try to determine which nodes fit best into this cluster. In particular, outlier detection matches our needs well as we want to find nodes that seem to be *different*. With the outlier detection approach, nodes may be classified as *different* but do not have to.

We need to keep in mind, that gathering data in a tree-based structure results in only few data points. If we use a branching factor of 8 we gain 8 data points at max in a single node and even less in most cases since there is no guarantee that our possible child tree is complete. Working with so few data points is difficult as the majority of the algorithms perform best when enough data to compare other data against is available. They often work on the complete data set which we can not provide since each node only has its local view. We need to be aware that there will be quality loss in these algorithms when working with this data base.

Alternatively, if we look at the data points we get over a certain time interval we could use algorithms that work on timelines and therefore would tackle the problem in a more suiting manner. The problem is algorithms try to find a change in the pattern which is difficult to determine since P2P networks are subject to churn and hence often show irregular patterns. Additionally, if attackers are sending wrong data regularly, there would be no pattern changes to determine.
Thus, we can try to use a mixture of both. We can collect the recent data we got, but do not treat them as a timeline. We will use this modification to have more information available to work on which should support the viability of detection algorithms. Note that we do not want with rather old data so that we need to work only with recent data as the old data could show information about the network which are not relevant anymore.

As a result of requirement $R_{NoNewPaths}$ we do not want to introduce new connections in the monitoring system and will work with the available data.
Furthermore we need to keep in mind that the different attack types call for different evaluation criteria since, e.g. false data may come from the correct node and therefore checking the origin would not lead to a suspicious behavior categorization. In contrast data that seems to be correct may be sent by a node which is not relevant for our position in the tree. For example, attacks aiming to shut down specific parts of the monitoring system can not be detected by comparing the received data with other data. Nevertheless, they are more obvious since the attacker's behavior is not conform to the monitoring system's rules and should be detected by checking a node's behavior.

Before we discuss further evaluation criteria, we will introduce the necessary basics of outlier detection as outlier detection represents a broad research field. We will present different techniques and discuss which one we will apply on each node's received data in the next chapter.

Chapter 3

Outlier Detection

In this chapter we will give an overview of existing approaches which may fit our needs to detect malicious behavior, e.g. data not provided correctly. At first we will define what an outlier is and the general categories in which outlier detection techniques can be divided. In detail we will only focus on techniques for numerical values although several additional ones exist since we are working with numerical data only in our thesis.

Hawkins [Haw80] defines an outlier as follows: "*An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.*"

For our cause, this means we assume that all non-malicious nodes represent one mechanism as they should behave according to the monitoring algorithm. In contrast a malicious node modifies the mechanism's behavior or the generated data and therefore forms its own mechanism.

Exemplary, Aggarwal [Agg13] states that outlier detection is used for several applications like Credit Card Fraud, Intrusion Detection, Sensor Events, and several more to detect anomalies in this data. These applications assume that a basic and known model behind the data exists and want to find significantly different data points which do not fit this model. For this outlier detection algorithms may either make a binary decision (yes/no) or provide a so called *Outlier Score* which can be compared to a predefined threshold. Such a threshold is also important to define *weak* outliers, so called *noise* and strong outliers, also called *anomalies*, where anomalies usually represent the interesting data points and noise data may stand for a random occurrence of data not fitting the normal model completely. For us interesting data points stand for possible attackers.

Outlier detection techniques can be divided in three categories: *supervised*, *semi-supervised*, and *unsupervised*.

Supervised Supervised means that there is both normal data and abnormal data available to train the detection algorithm, e.g. in a Credit Fraud application. Via human feedback this data will be used to improve the finding results.

Semi-supervised Algorithms used in a semi-supervised scenario can only be trained on one type of data. There is either normal or abnormal data available to feed the algorithm with basic information to use later on. Usually, there is a normal data set available which these techniques use to generate the normal model from to test additional data against.

Unsupervised Unsupervised algorithms need to determine outliers without any initial feedback or information. The general assumption is that the majority of the available data comes from the normal model and that there may be abnormal data which needs to be determined.

Aside from these categories, as mentioned in [Agg13] there exist several basic models to determine outliers like Probabilistic and Statistical Models, Linear Models, High-Dimensional Models, Time-Series Models, and even outlier detection models for categorical and text attributes or graphs. Based on the assumed data model - whose determination is crucial for the analysis results - each outlier model provides better or worse results. If one assumes a linear-regression as the underlying model, this data needs to be treated different in contrast to a Gaussian or Zipf distribution. One may even imagine a data distribution where two-dimensional data points when plotted are arranged in a circle where an outlier may be positioned exactly in the middle of this circle. One also needs to be aware that the method should not be overfit to a specific scenario. Each model may have outlier detection techniques fitting in one of the three categories mentioned above.

As we will work with numerical data, we will focus on models working with this kind of data. Next, we will introduce the ideas behind Proximity-based outlier detection and Probabilistic and Statistical Models for outlier detection more detailed.

3.1 Proximity-based Outlier Detection

Aggarwal [Agg13] defines Proximity-based methods as methods that check the proximity of a data point and define outliers based on the population in such a so called *neighborhood*. Proximity checks may either be cluster-based, distance-based, or density-based. While they all work with the definition of proximity for their calculations, they differ in how they examine the proximities.

Cluster-based approaches try to partition data points from a data set into different groups whereas Density-based approaches check the number of existing points in a defined location, e.g. in a radius around a certain point. That means that Density-based approaches try to segment the data space instead of the data points.

Distance-based approaches compare the distance between data points in the data set. Often the *nearest-neighbor distance* is used to evaluate a data point and build its outlier score. These approaches offer more granularity at the cost of more used computational time for all distance comparisons.

We will take a closer look at these three approaches in the next sections.

3.1.1 Cluster-based Classification

The aim of Cluster-based Classification is to divide a data set into subsets which should be as dense as possible. Each cluster consists of a so called cluster centroid which represents the middle of all data points in a cluster[1] and the data points belonging to a cluster. At first this technique does not fit the aims of outlier detection as outliers should not be part of a dense subset, i.e. a cluster.

Nevertheless, Cluster-based classification algorithms often consider this and determine *not fitting* points already while building clusters. They can make a binary decision whether a point belongs to a cluster or not. Furthermore a minimal threshold for cluster sizes may be determined so that not only single data points will be excluded but also small concentrations of data points will not be mistaken as a separate cluster.

To be able to not only make a binary decision the distance to the cluster's centroid could be considered, too. As clusters may have different shapes, a special distance measurement will be used in this case. Otherwise, e.g. in a two-dimensional data set, clusters would always be treated as if all points belonging to a cluster are distributed in a certain radius if one uses the Euclidean distance. The used distance measurement is called *Mahalanobis distance* and is able to consider a cluster's variance for different correlation directions. It enables us to find elliptical clusters and not only clusters arranged as a circle.

[1]Note that the cluster centroid must not be an existing data point itself.

3.1.2 Density-based Classification

Density-based Classification takes a different approach by counting the number of data points in a certain locality. The difference to building clusters is that cluster-based approaches take a point and try to build a good fitting space around it, while density-based approaches take a defined space around a point and examine the number of points in it. The general idea is that outliers should have a different density compared to their neighbor points. Density-based Classification may either consider a mean density across the whole data set or may examine existing densities only in a specific subset around the current examined point. The density-based approach makes use of the k-nearest neighbor distance, which is also often used for Distance-based Classification. We will explain it using the example of the Local Outlier Factor. Due to working on densities instead of distances only, they are able to identify outliers in heterogeneous data sets with different densities across the set. Distance-based approaches need to define a global threshold for the distance between two data points, while density-based approaches may work more locally and therefore may detect outliers with a small overall distance. Note that a distance may be small compared to all other data points in the data set, but when examined in the local context it may be significantly higher than the neighbor points' distances.

Example: Local Outlier Factor for Outlier Detection

The Local Outlier Factor (in short: LOF) was introduced by [BKNS00]. It defines a local neighborhood of a point via the k-nearest neighbor. It is then used to determine the local density of this point instead of comparing it against the global data distribution. To build a score LOF introduces the reachability distance of two points p_1 and p_2 as

$$rd_k(p_1, p_2) = \max\{k - nearestneighbordistance(p_2), distance(p_1, p_2)\} \tag{3.1}$$

to stabilize the results. To define the reachability distances an arbitrary distance measurement may be used, e.g. the Manhattan Distance or the Euclidean Distance.

One needs to note, that this distance measurement is not symmetrical, as p_2 may be the part of the k-nearest neighbor set of p_1, but p_1 does not need to be part of the k-nearest neighbor set of p_2, e.g. due to additional data points being close to p_2.

The idea is now to define a *local reachability density* (in short: lrd) which represents the inverse of the average reachability distance of p_1 and its neighbors. This is done for each point in the k-nearest neighbor set of the examined point and lrd_{p_2} will be compared to the other points to build the score.

A local outlier factor of 1 shows a homogeneous density comparing the examined data point's density to its neighbor's. A LOF significantly bigger than 1 shows that this point is more likely to be an outlier as its local density is more sparse compared to the neighbor points.

3.1.3 Distance-based Classification

Distance-based outlier detection is based on distance-based classification. Distance-based classification algorithms try to decide in which class an examined data point will fit in most likely. To make this decision a certain amount k of neighbor data points with the closest distance will be taken into consideration and the best-suited class will be chosen via a majority rule [CH67]. Which distance measurement will be used is not prescribed. One can use an arbitrary distance measurement, e.g. the Manhattan distance or the euclidean distance. There are several modifications of this approach available, e.g. giving each examined neighbor point a specific weight based on the distance.

Choosing the parameter k is a crucial part when using these algorithms as k influences the quality of the results due to being the determining factor while trying to fit a point to it's neighbors.

For outlier detection purposes, these approaches, which examine the distances a point has to its k nearest neighbors, can not only be used to classify a point, but are also used to determine a score to be able to give each point a certain rating. An outlier is then defined by having a bigger distance to its neighbors as other points.

In comparison to Cluster-based approaches, Distance-based approaches offer more granularity in their analysis as they do not only make a binary decision, but are connected to a numerical value which can then be interpreted to distinguish between weak and strong outliers.

Example: *k*-nearest neighbor Distance for Outlier Detection

Using the k-nearest neighbor distance for outlier detection is a famous approach due to the fact its idea is easy understandable and it often offers good results. As introduced in [RRS00] the concept used to make classification decisions can be adapted to build a score for individual points. The general idea behind using the k-nearest neighbor distance to determine whether a point is an outlier or not, is the assumption that outliers should have a significantly higher distance to the other data points than *normal* data points. The value k is usually chosen in relation to the size of the examined data set or as an absolute value.

To build a score a loop can be used to iterate over all known data points and then one can either use only the distance to the k-nearest neighbor or the cumulative or average distances from the i-nearest neighbor with $i = 1, 2, \ldots, k$.

3.2 Probabilistic and Statistical Models

While many Proximity-based techniques were developed to work on data examined by computer scientists, Probabilistic and Statistical models have their origin in the field of mathematics. They try to determine the likelihood that an examined data point fits to the underlying model. Therefore the outlier score is often connected to the probability value. A popular method is the extreme value analysis which takes values that are either too large or too small and declares them as outliers. To determine such regions where values are labeled *extreme* the so called *statistical tails* of the underlying distribution are determined and examined.

Extreme Value Analysis

As a measurement unit, Extreme Value Analysis produces a so called Z-Score. Calculating those Z-Scores is a common tool to determine the possibility of values be created by the same mechanism under the assumption, that values are normally distributed. The Z-Score of a value correlates with the corresponding standard deviation values. A Z-Score of $z_x = 1$ means that the distance between the arithmetic mean of a data set and the value x is exactly one standard deviation. We could also say that the Z-Score represents a sort of normalization of the data since it represents the examined data point in form of its deviation compared to the data sets mean.

If a data set has too few points one can use the t-value test which considers the missing knowledge of the underlying data due to too few data points by increasing the tail size according to the *degrees of freedom*, used for the estimation of the variances.

Using the arithmetic mean leads to a big influence of outliers regarding the results. In particular, it is possible that normal values will appear as strange values suddenly.
To counteract this the Modified Z-Score introduced by Iglewicz and Hoaglin [IH93] relies on the distance of a data point to the data set's median. With this, single extreme values will have less influence on the result and the analysis should be more robust. Additionally, they state their method should also be able to cope with small data sets.

For their calculation, they first calculate the median of a data set and then the deviation of each data point and the calculated median. They divide these deviations by the median value of all deviations to build the score. As their calculation method differs from the usually used Z-Score calculation, which calculates the standard deviation and puts a data point's distance to the mean in relation to it, they need to correct their calculation results. They use a predefined constant factor and multiply it to the results to approximate the standard deviation.

3.3 Techniques Suited For Our Data

As we are working with a P2P overlay there is no central instance which has a global view over the network. The root node, which would be most likely to have knowledge about the whole network, only has aggregated data itself and does not know individual details either. Thus, we need to be able to work with few data and only a limited view. Additionally we do not have any data which we could use as a *base case* to compare our data against. This means that we need to use a unsupervised technique. Furthermore, it is hard to determine how one should choose a good k-value for distance respectively density based approaches. Especially due to the limited view of each node and therefore the small data base where a value difference of 1 would already have a great impact. Such techniques would be great to use on the complete data for the whole network if we were able to collect it all. Because of these limitations, using extreme value analysis should be best suited for our approach.

Besides, we make an extra assumption: since key-based routing P2P overlays aim to distribute peer IDs homogeneous in the overlay, we assume that basic statistics e.g. about the network should not differ significantly for each peer. E.g. the average hop count should not be 4 for one node and 40 for another node. Analogous, round trip times etc. should be equally distributed.

Of course one needs to note that we cannot detect peer specific information with detection techniques as each peer may have significantly different possible resources, e.g. more bandwidth or more memory space.

Possible Problems with Cluster-based Outlier Detection Using a Cluster-based approach may result in new difficulties by definition, as outliers should be data points which explicitly do not belong to a cluster. Although there are Cluster-based classification methods, which already determine outliers as a *side product* while building clusters, these are only binary decisions. As a problem even noisy data which may not fit into a cluster, but still can be a *normal* data point, being produced by a regular mechanism, may be classified wrong. While these disadvantages can be counteracted by using the distance to a cluster's centroid instead of only a binary decision there are still additional problems present.

If we use clustering algorithms outliers may even have an influence on the clustering itself and e.g. may shift cluster centroids. Usually data sets are cleaned from possible outliers before using any cluster classification methods on it to prevent such behavior. Furthermore outliers may even appear as a cluster itself[2]. This can be counteracted by defining a minimum threshold of points which are necessary to build a cluster, but this may influence other *correct* clusters, too.

The biggest problem is the missing information detail we get from those methods if used in small data sets. As only few data points are present the granularity of Cluster-based approaches is low and if we want to work with a minimum threshold for cluster sizes, even small differences in the value we choose have a great impact on the results.

Possible Problems with Density-based Outlier Detection In comparison to distance-based approaches, density-based approaches can work with local densities. While this is a benefit of these techniques, there is the possibility of non-suspicious local densities due to multiple malicious values being close to each other. Furthermore, it is hard to determine a threshold for outlier declaration. In heterogeneous data sets a high LOF-value may still be from a normal data point since the densities differ much in the whole data set, whereas in homogeneous data sets values close to 1 may already indicate an outlier.

Another important point is the high computational complexity for each node as we need to calculate the density of the examined value and all values in its k-neighborhood. More important we would need to have detailed information about each point in the examined k-neighborhood, i.e. information about these point's neighborhood. With the present local view in each node, this is not a possible approach for us. Getting this data would put too much overhead on the P2P overlay and violate our requirement $R_{Overhead}$ beside the significant computational complexity which would be necessary for each data point's neighborhood contradicting requirement $R_{Performance}$.

Possible Problems with Distance-based Outlier Detection Due to the computation of all distance-pairs, these algorithms may need $O(n^2)$ computations for n data points in the worst case which is unwanted regarding our requirement $R_{Performance}$. If one wants only a binary decision the algorithm may end early once it has found enough other data points being close enough. Alternatively, this computation complexity can be further improved e.g. via a cell-based approach to exclude certain points

[2]Imagine that there are enough outlier data points which *reside* around the same locality.

from the computation at the beginning giving up some of its granularity. Nevertheless, the necessary effort is still high by comparison.

Additionally different cluster densities will not be considered which means that those techniques have problems in heterogeneous data point distributions. Imagine a wide spread cluster A with a sparse neighborhood and a cluster B with a dense neighborhood. To prevent the algorithm to detect the majority of the points in cluster A as outliers the threshold needs to be increased. With this an outlier near to cluster B may not be detected because it's distance may still be under this increased threshold although examined only locally it would clearly be an outlier. Besides we need expert knowledge to determine good values for the distance threshold. Additionally due to the nature of a P2P-network these values may change over time.

Possible Problems with Extreme Value Analysis If one uses Extreme Value Analysis there is always the risk that one may try to fit data (unknowingly) to a distribution which may not be appropriate. This is because statistical methods often make simplified assumption about the data base which may not hold in reality. This method is also more suited to check values regarding the network status and not the attributes of a single node since nodes may differ regarding their available bandwidth, free memory space or computing power. Further problems arise when one only has a small data base, which one needs to keep in mind when using Extreme Value Analysis.

A special possible problem arises when using the Modified Z-Score. If more than 50 percent of the examined values are the same the result of the median distance will be 0 and this technique will not work in this case. IBM [IBM] proposes a solution, being explained in detail in Section 4.2.

A general problem will be the small data base each node has and the difficulty with the determination of good thresholds for each technique. Especially the proximity-based approaches usually need a sufficient data base to work with as they examined a data point in relation to its neighborhood. Nevertheless, statistical approach also benefit from a larger data base to work on. Given the fact that they need only little knowledge about the other data points, i.e. they only need to know the points themselves and not additional information about the neighbor points' densities or neighbor distances, Extreme Value Analysis should be most-suited for our problem. It has low computational complexity and the ability to work with sparse data. We will describe the algorithm used in Section 4.2.

With the description of different attack types malicious nodes could use against the monitoring structure explained in Chapter 2 and the basics of outlier detection we will introduce our rating mechanisms to detect malicious behavior in SkyEye.KOM in the following chapter.

Chapter 4

DOMiNo

A Rating System for SkyEye.KOM

As mentioned in Section 2.3 we want to build a detection system with as little influence on the nodes and the P2P overlay as possible, i.e. with few new overhead and little computation time (stated as requirements $R_{Overhead}$ and $R_{Performance}$). Additionally we want to have a good precision for our results and do not want to introduce new attack paths for malicious nodes (requirements $R_{Precision}$ and $R_{NoNewPaths}$). Moreover as pointed out in Section 2.4 there are different evaluation criteria for malicious or non-malicious nodes due to the structure of the used monitoring system and the different attack types a node may choose.

D*etection* ***Of*** ***Mal****icious* ***No****des* (DOMiNo) is our approach to identify inappropriate behavior in a tree-based structured monitoring system like SkyEye.KOM. It is based on a rating system containing multiple rating mechanisms, each returning a numerical rating for an examined data point or peer, respectively. Once a peer behaves malicious it should trigger one or more active rating systems and be detected.

In the following we will discuss possible criteria and which attacks these criteria are able to detect. To identify the different attack types we use the notation introduced in Chapter 2.2. We divide our rating mechanisms into *instant rating mechanisms* which are able to determine malicious behavior instantly and *collecting rating mechanisms* which need to work on a collected data base.

4.1 Instant Rating Mechanisms

Instant rating mechanisms provide a rating without working on a collected data base. Instead they check different properties of a received message like its origin or the sender's properties. Furthermore they may review the containing data sets regarding several points, e.g. the completeness or the plausibility of specific data points using simple, but effective, rules.

4.1.1 Checking Domain Ranges

Since Domains are fixed by the structured monitoring system, each node can calculate its parent's Domain. We can exploit this to detect nodes that send messages to random nodes[1] as sending nodes should have a Domain from a specific range depending on where the receiver is positioned in the tree. We are able to verify a sender's Domain easily by calculating our responsible Domain for non-ACK messages, i.e. child nodes must be in our responsibility range, and by calculating the Domain of our parent node for ACK-messages. As sender Domains are derived from the sender's peer ID they can be calculated from the sender's information which we receive within the message. While this approach can detect attackers using the attack AT_{Random} fairly easy, note that checking our Domain range for child nodes only works if a malicious sender is outside this range. If a node is inside our range, but actually not our direct child node and instead far deeper down the tree, we can not detect such a node with this criteria.

Whereas checking the position of a child node is fairly easy, due to the fact that we already know the information about our Domain range as we calculate it to find our position in the tree, calculating the parent's Domain is more difficult. Since sub trees must not be completely filled, a parent node may be Coordinator on multiple levels above us. Therefore its ID may be outside of the Domain range of our direct, i.e. our level minus one, parent. E.g. we operate on level 4 and our correct parent node is responsible on level 1, 2, and 3 since there are no other nodes between the child and the parent node[2]. This parent node is obviously responsible for a far bigger range than it would be if it only operates on level 3. As a result its Coordinator ID lies outside of the Domain range of level 3 which mean we cannot simply rely on the Coordinator ID contained in the message. Figure 4.1 depicts the described scenario with a branching factor of 2. The nodes drawn in a light gray color are the respective Coordinator IDs from parent node where it also operates, but which are not it Coordinator ID closest to the root. The parent node would state its Coordinator ID to be $C_1^{pID} = 0.75$. The child node may check if the parent's IP fits the expected range. It would expect the resulting Coordinator

[1]Or in general not the proper child respectively parent node.

[2]So there is no other node with an ID between the child's and the parent node's.

ID to for its parent to be in the interval $[0.875, 1]$ marked with the light gray area in Figure 4.1. More specifically it would expect its parent to send the corresponding Coordinator ID from level 3, i.e. $C_3^{pID} = 0.9375$ if it does not keep in mind that the parent may be active on several levels.

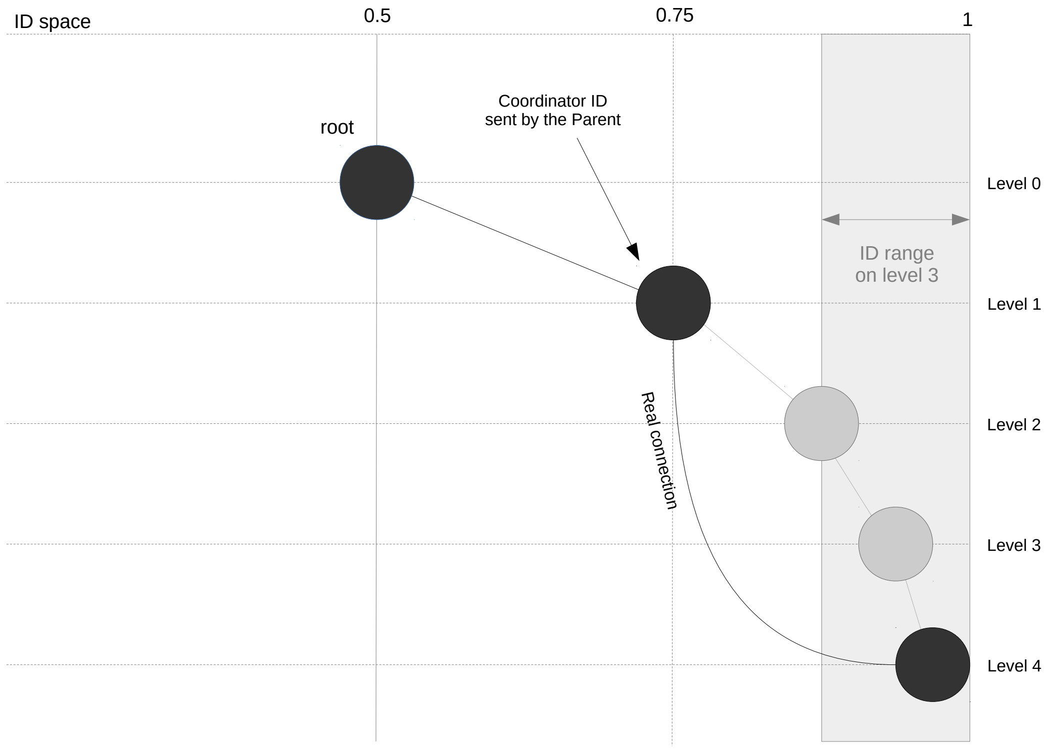


Figure 4.1: A sub tree in SkyEye.KOM with the parent node being Coordinator on level 1 and the child node being Coordinator on level 4.

To counteract this, we either need to allow bigger ranges which could end up to be equal to the root's Domain range since (in the worst case) there might be no node between us and the root. Every other approach would result in false positives if the tree is not well balanced. Nevertheless, the parent would still be suspicious as the child node can not be sure if there are not any other nodes between the alleged parent and itself. Additionally nodes may also spoof their ID which may let us assume that they are our *true* parent node. Getting confronted with a new ID would be nothing special since this node may have just joined the network or other topology changes may have lead to this new constellation. Due to the mentioned problems, we will tackle this problem differently and explain our approach in the next subsection.

Identifying correct Parent Nodes via Nonces

A possible approach would be to check messages from parent nodes by starting a lookup in the overlay if our message partner is our legitimate parent, i.e. by starting a lookup for our parent's ID one level above us. This is necessary since, as mentioned above, a parent node may have changed due to recent changes in the topology. Nevertheless, using additional lookups would be a fairly expensive solution as it puts additional overhead (see requirement $R_{Overhead}$) on the overlay. As a tweak the sender's IP could be used to calculate whether the sender could be a parent node at all before starting the lookup which may reduce the number of additional lookups in some cases.

A more important point is, parent nodes should, according to the monitoring algorithm, only react to messages from child nodes. Receiving messages from parents without the initiation (of the communication) by the child nodes are a strong hint for wrong behavior. To exploit this fact and to counteract this attack type, i.e. the behavior of the attack $AT_{RandomACK}$, we can use a different approach to verify parent nodes since they should only react if we have sent a message at first. With this in mind we can remember our request and check if an answer fits. Note that the node's ID we tried to contact and the node's ID we get a message from may not be the same by reason of the responsibility ranges where we contact the node responsible for a certain ID. For this reason we will not use the ID to verify an incoming message and instead add a random number called *nonce* to our request message. We demand this random number to be in the message to verify if a node was contacted by us before or not. The used number will be changed for each message so that attackers would need to guess our random number to verify that we contacted them before and not be detected. Note that this will not help us if a malicious parent spoofs his IP and therefore takes the place of our legitimate parent. It only prevents a node to send ACK messages arbitrarily without an initiated connection.

The algorithm for the checking of Domains (and nonces) is sketched in Algorithm 1.

Note that the introduction of this new criteria means that we need to extend the monitoring system SkyEye.KOM to use such nonces in each message. While this adds little new overhead to each message, e.g. using a 32-bit integer would add 4 additional bytes to each message, this approach adds less overhead than starting lookups to verify parent nodes and therefore supports our requirement $R_{Overhead}$.

Algorithm 1 Domain Check Algorithm

```
Input: A received message $msg$
procedure CheckDomain
    $result \leftarrow notsuspicious$
    if $msg_{ACK}$ is set then                                ▷ message from parent node
        if we are root then
            $result \leftarrow suspicious$
        else
            Check if the message's nonce $msg_{nonce}$ is correct/known
            if $msg_{nonce}$ is not correct then
                $result \leftarrow suspicious$
            end if
        end if
    else                                                      ▷ message from child node
        Calculate the parent's minimum Domain and maximum Domain
        if sender is out of calculated bounds then
            $result \leftarrow suspicious$
        end if
    end if
    return $result$
end procedure
```

4.1.2 Implausible Values

Based on the knowledge we have about the data we receive respectively expect we can perform several checks for implausible values. That means that if we expect information about maximum values and minimum values that $max >= min$ should hold. Furthermore a possible mean value should range between those two values, i.e. $max >= mean >= min$. We are also able to check the provided data for certain threshold if we know that those exist. As an example we could imagine data calculated via a formula where we know that the result must always be from the interval $[0, 1]$. We can then perform sanity checks on this data again.

These criteria support the detection of attacks which send insufficient data, i.e. AT_{Empty}, but also every attack which may manipulate data. Careless attackers or attackers not aware of specific restrictions regarding the data may violate several effective rules and thus getting detected by the rating mechanism.

Nevertheless, AT_{Empty} is most vulnerable to this detection since, as a proper participant of the monitoring mechanism, a node should be able to deliver some information for each data attribute, i.e. at least data about itself.

This criteria is especially important for AT_{Parent} and AT_{Child} as other nodes need to identify such attacks via the data sent and cannot use further clues, i.e. checking the sender's properties, e.g. the sender's ID, would not show suspicious behavior. While attacked parent nodes may have other data available provided by their other child nodes which can be used for comparisons, attacked child nodes only have the malicious parent node and their own data as a reference.

Furthermore note that the mentioned criteria will not be useful for data like available bandwidth or similar as this may change rapidly and is independent from every other network status. We can only use these where we are certain about specific ranges or where we know that we can demand specific information according to the used algorithm.

One could argue that there are several additional criteria possible depending on the exchanged data. E.g. if we exchange aggregated data values, then, except for the short time frame where we are new to the tree, the values from a parent node should usually be higher than values received from children nodes and vice versa as aggregated data will be supplemented upwards the tree. Especially, since parent nodes only send data as a reaction (ACK) message to child nodes, they were already able to use the data received from the child node. Therefore they should answer with values of at least the information of the child node plus their own. While this approach might be useful to detect nodes sending data from other positions in the tree as the corresponding sub tree may differ, relying only on this criteria would open new attack paths for attackers. We would need to keep in mind that the tree structure may change at any time. Consider the scenario depicted in Figure 4.2. The leaf node would send significantly higher values to the parent node than before, i.e. count = 1 vs. count = 3. If we assume that the parent's information needs to be at least as high as the information sent by the child node, the parent would rate the child node as malicious.

So we cannot make such general assumptions as a criteria as we only want to use cases where we can be completely sure at any time.

4.1.3 Awareness of Missing ACKs

To be recognized by the monitoring participants, nodes need to take part in the underlying P2P overlay; otherwise they would be ignored. This means that an attacker needs to be at least actively participating in the overlay and we can check his activity. As part of the monitoring mechanism we can expect an ACK message from our parent once we contacted him before and he is still active in the network.

As described in Section 2.2.2 attackers may try to break links between certain parts in the tree structure. To break down a link between two parts, the attacker needs to be at least the parent node of

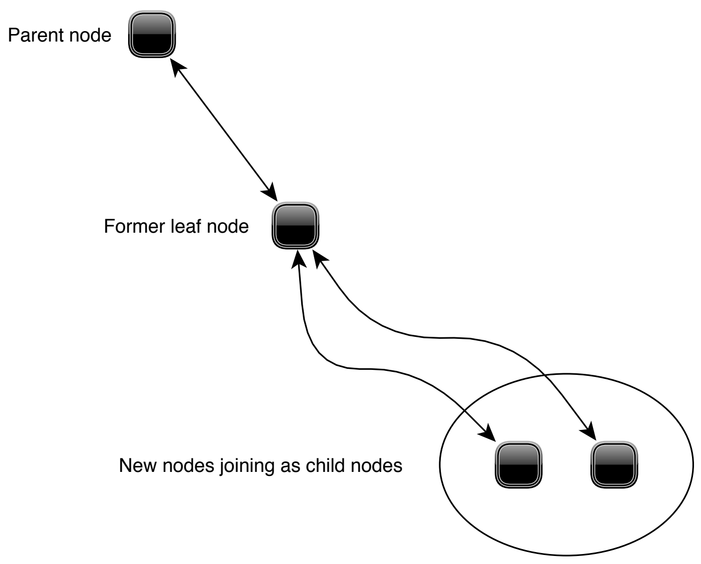


Figure 4.2: New nodes joining a sub tree in SkyEye.KOM

one other node. That means we can identify such attacks via the amount of ACKs we receive - or in this case do not receive. One thing we need to keep in mind is that there are two exceptions where a normal node may not be responding to our messages. The first case is that a receiver may have not receive our message or we did not receive the sender's ACK due to package loss. Another possibility would be that there is a restructuring process active in the tree structure as nodes join or leave the overlay. The message we sent may now have been delivered to a node which is not responsible for us anymore. In every other case we can assume that a not responding node behaves not in accordance with the monitoring algorithm.

To reduce false declarations of nodes being malicious, e.g. because of package loss or restructuring, we will give a parent node a certain time threshold in which it should have responded us. A possible value would be to wait, e.g. two or three update intervals. I.e. a parent node has two respectively three chances to send an ACK message.

We need to be aware that this criteria only works for parent nodes as each child knows its parent node, but a parent node does not know its child nodes since this would mean that the parent node needs to be aware of the topology, which would need a significant amount of new overhead to the overlay violating our requirement $R_{Overhead}$.

Due to the facts mentioned above[3], attackers using the attack type $AT_{DoNothing}$ or $AT_{Upwards}$ are highly suspicious. They needed to react in other situations, e.g. a lookup, so we know they must be present

[3]To be recognized, a node needs to join the overlay and react to overlay messages. Else it will not be considered for the monitoring mechanism.

and can expect them to send an ACK, but they do not show any presence during the monitoring message exchanges. The only nodes being excluded from the easy detection possibility are leaf nodes which are only present in their role as a child node and no lookup for a parent node will show them. As stated, they could only be detected with additional effort, i.e. lookups, but due to our requirement $R_{Overhead}$ this will not be done.
Moreover, the missing participation of leaf nodes has only minimal influence on the monitoring system and therefore stands for the weakest attack possible.

4.2 Collecting Rating Mechanisms

In contrast to instant rating mechanisms, collecting ones work with a gained data base. They try to determine the likeliness that all collected data was produced by the same mechanism and try to differentiate manipulated data from correct data. These mechanisms are highly dependent on the provided data base and cannot work with arbitrary data. To compare data against each other one needs to be sure what the underlying data model is so that he can choose a fitting method developed for this model.

Calculating Z-Scores

As our rating mechanism for recently collected data, we will use the Modified Z-Score introduced by Iglewicz and Hoaglin [IH93] which was developed as a technique which gets less influenced by extreme values due to using the median instead of the arithmetic mean.

As we discussed in Section 3.3, we expect this detection technique to be best-suited for our purpose and for our sparse data base.

While Z-Scores are usually not suited for small data sets, the authors [IH93] state their algorithm can cope with those sets. To provide a bigger data base for the algorithm to work on, we will put the data received during the last three update intervals. This also means that this algorithms performance is dependent on the branching factor as the possibility for a bigger data base increases as the branching factor increases.

The Modified Z-Score is calculated by using the median absolute deviation (denoted as MAD). The equation for calculating the MAD is depicted in Equation (4.1).

$$MAD = median(|x_i - median(x)|) \tag{4.1}$$

For an approximation of the standard deviation, which is the usual measurement for Z-Scores, this result needs to be multiplied by a constant as the median absolute deviation is used. The authors of the paper determined the constant to be 0.6745 for normally distributed data. The whole calculation for the Modified Z-Score M_i of a data point is denoted in Equation (4.2) with x_i being the examined data point and x being the whole data set.

$$M_i = \frac{0.6745(x_i - median(x))}{MAD} \tag{4.2}$$

The authors propose the threshold, for labeling a data point as an potential outlier, of M_i to be 3.5 which we will adapt for our analysis.

As noted in Section 3.3 we need to be aware if at least 50 percent of the provided data points have the same value, since the median deviation from the calculated median of the data set, i.e. the MAD value, will be 0 and we would try to divide by 0. Usually all points that do not equal the median will be marked as an outlier. As a countermeasure, we could denote that a result is not possible for the algorithm in this case. To be able to make a decision nevertheless, we use a solution used by IBM in their software SPSS Statistics. In their knowledge center [IBM] IBM states that SPSS uses the mean average deviation (MeanAD) in case the MAD equals 0. The calculation of the MeanAD is depicted in Equation (4.3) The corresponding calculation which uses the MeanAD is denoted in Equation (4.4). The MeanAD gets multiplied by a constant factor, too, to approximate the standard deviation.

$$MeanAD = average(|x_i - median(x)|) \tag{4.3}$$

$$M_i = \frac{0.7979(x_i - median(x))}{MeanAD} \tag{4.4}$$

If one uses the MeanAD, it can only be 0 if all data point values are the same and therefore no outlier is present.

The algorithm to calculate the Z-Score is denoted in Algorithm 2.

Algorithm 2 Modified Z-Median Score Algorithm

```
1: Input: A set of collected data $D$, the data point to examine $d_i$
2: procedure CALCULATEMODIFIEDZMEDIAN
3:     $dataSetMedian \leftarrow median(D)$
4:     for all $d_x$ in D do
5:         add $|dataSetMedian - d_x|$ to a median deviation list $MD$
6:     end for
7:     $MAD \leftarrow median(MD)$
8:     if MAD == 0 then                    ▷ If more than 50% of the value were equal
9:         $MeanAD \leftarrow average(MD)$
10:        if MeanAD == 0 then                          ▷ All values are equally big
11:            $M_i \leftarrow NOTSUSPICIOUS$
12:        else
13:            $M_i \leftarrow (0.7979 * (d_i - dataSetMedian))/MeanAD$
14:        end if
15:    else
16:        $M_i \leftarrow (0.6745 * (d_i - dataSetMedian))/MAD$
17:    end if
18:    return $M_i$
19: end procedure
```

As mentioned before statistical approaches are only suited for data which should be homogeneous and where we can expect a certain underlying data model. This means, we state that this method is more suited to check values regarding the network status like hop counts or TTL values where we can expect that the data is normally distributed. Although we were not able to check if those values are in fact normally distributed, there are strong hints as mentioned in [JWS03] which does not deal with P2P network traffic, but network traffic in general.

Furthermore, we assume that values like node counts should be close to each other for sub trees which originate close to the root as P2P overlays balance the ID distribution due to the cryptographic has functions used. That means, bigger differences should only occur close to the leaf nodes as sub trees far away from the root point are more likely to be unbalanced. Plus a data base needs to be present to compare data against, so that this criteria is most suited for attacks sending manipulated data upwards the tree, i.e. AT_{Parent}, AT_{Random}, and AT_{Root}.

Due to the possibility of false positives, especially given the reduced data base, one should keep in mind to control a node's behavior over a certain time interval and not prejudge a node because of a single statistical abnormality. There is no guarantee that a node may not look suspicious for a short amount of time due to sudden changes in the network or other reasons. We therefore propose to evaluate a node's behavior several times before making a final judgment when using statistical approaches.

In the next section we will summarize the presented criteria as a tabular to provide a better overview.

4.3 Summary of Criteria

To present the criteria in a more compact form, we give an overview in the following tabular. We denote the mentioned criteria with both a short summary of the criteria on the left side and the attacks which may be detected via the used criteria on the right side.

Evaluation Criteria	Possibly detected Attack Types
Check the Domain a potential child node comes from	AT_{Random}
Check the message's nonce for a match	$AT_{RandomACK}$
Check if the message contains information for each demanded data point	AT_{Empty}
Check if the provided data stands against basic sanity checks	All AT_*
Check if our message upwards gets an answer	$AT_{DoNothing}$ $AT_{Upwards}$
Calculate the Modified Z-Score	AT_{Parent} AT_{Root} AT_{Random}

Table 4.1: Summary of the presented evaluation criteria for incoming messages in SkyEye.KOM

4.4 Blocking Data Sets

As we want to reduce the effect that abnormal behavior, i.e. nodes not acting according to the message forwarding algorithm of SkyEye.KOM, has on our evaluation criteria, we will block such data sets. That means if an instant rating mechanism rates a received message as malicious, we do not collect the contained data set for our data base to have no influence on the collecting systems. We can determine these data sets without the risk of false positives since our instant rating mechanism only checks if a participants behavior is correct or not. If a participant does not behave correctly, we do not want him to influence the monitoring mechanism, even if his provided data is not manipulated.

Note that we cannot block data sets which originate in our own Domain range as we have no mechanism available to check for proper child nodes. As a result, data provided by attackers using the attack AT_{Random} being positioned between a nodes minimum Domain and maximum Domain will still be considered for the statistical approaches.

4.5 Remaining Problems

We already mentioned in the section before, that we still cannot determine incorrect child messages originating in our own Domain range.

In particular, since the root manages the whole Domain range it is more difficult for it to decide whether a message is from a legitimate child or not. The root would need to start explicit lookups for each received message due to possible changes in the structure. Another possibility would be to check the values the message contains as a message from a higher level should contain lower values if the tree is well-balanced.

Furthermore, an attacker only reacting to its child nodes is hard to detect as the child nodes are the only nodes being aware that this malicious node is present[4]. Due to the fact that child nodes still receive global views from the malicious parent node and other parts of the tree are not aware of the missing participation without explicit lookups to find out more about the topology.

Another problem is the missing data base to compare against the data received by parents. It is not possible to detect malicious parents except for the case they send implausible values.

In the following chapter we will present the implementation done in the P2P simulation framework PeerfactSim.KOM [SGR+11]. We will present PeerfactSim.KOM and the previous work our thesis will work with. After that we explain and discuss the implementation details done by us, based on the subjects explained until this point.

[4]Since nodes in SkyEye.KOM only know about their parent node and not the whole topology.

Chapter 5

Implementation

In this chapter we will present the implementation done for this thesis. At first we introduce the event-driven P2P simulation framework PeerfactSim.KOM [SGR$^+$11] which we used to realize our rating mechanisms. We then explain how the monitoring system SkyEye.KOM was implemented and how our architecture is based on the implementation of SkyEye.KOM.
At last we explain more detailed how we implemented the attackers and DOMiNo in PeerfactSim.KOM.

5.1 PeerfactSim.KOM

PeerfactSim.KOM is a large-scale and event-driven P2P simulation framework written in Java. Being event-driven, specific events which are part of an operation, e.g. a lookup in the P2P overlay, occurring during a simulation are put into the simulator's time scheduler. These operations will then be executed sequentially according to the scheduler's timeline. It was launched at the Technical University Darmstadt and later extended at the University Paderborn. Currently the simulation framework is maintained at the Heinrich Heine University Düsseldorf.

A main advantage is its modular architecture. For this the simulation framework is divided into several layers, e.g. a Churn Model Layer where different churn models are available for use. In the Transport Layer the user can choose between TCP and UDP. Dependent on the layer, even multiple components may be used, e.g. in the Application Layer more than one application can be used simultaneously.

Which components and configurations should be loaded and used during a simulation can be determined via a XML configuration file. In a so called action file users are able to specify several operations for groups containing several nodes which should occur during a simulation at a specific time or equally distributed in a certain time interval. In our case, an example would be the start of the monitoring mechanism or the joining phase when peers join the P2P overlay.

In our case SkyEye.KOM is realized on the Application Layer and runs on top of an underlying P2P overlay which implements the KBR (key-based routing) interface [DZD$^+$03], i.e. Chord or Pastry. In the following sections we will describe the implementations we did in PeerfactSim.KOM.

5.2 Previous Work

As our basis we use the already implemented work from Philipp Giesen [Gie14]. In his Master's thesis he realized the monitoring system SkyEye.KOM as an application running on top of a running KBR overlay. It offers different classes handling specific tasks of the monitoring system. The `Data Manager` stores received data and prepares outgoing data, while the `Communication Manager` handles the communication by using the P2P overlay to do a lookup for the responsible node and the Transport Layer to send a message directly to this node. The `Monitoring Manager` is the class representing the peer itself since it initiates each further action. It is the heart of the application which handles the logic of the monitoring system. A defined node, e.g. the root node, also reports the system's information to the simulator's analyzer which is used to analyze the simulation results. If a `Monitoring Manager` wants to send a message, it can ask the `Data Manager` which aggregates all available data and returns a single data set. With this it is able to prepare a message which will then be delegated to the `Communication Manager` to send it to the receiver. As we kept his basic application structure the communication paths between the mentioned classes can be seen in Figure 5.1.

We extended the monitoring system already in a previous project where we introduced some aspects used in unstructured, so called *gossip based*, monitoring systems, to improve SkyEye.KOM's robustness against churn. The nodes can use gossip like connections on the same level to exchange data between each other. We furthermore introduced several tweaks to improve the monitoring system's behavior. In the original implementation nodes started to participate in the monitoring system immediately although they may not have received any child data at this point. As a result they send only parts of the actual data during a short period. We added a new minimal threshold value a node needs to wait after if joined the overlay before it starts participating. Another problem was the nodes' behavior when changing position where they removed all their previous known data. We added a new message type which a old node uses to provide the new node with previous known data if it takes over the old node's Coordinator ID. At last we changed the data saving mechanism which used a sender's peer ID before. In our extended approach the data saving mechanism now uses the sender's Coordinator

ID to be able to associate data sets with the according Coordinator. To identify the Coordinator ID without further effort, messages now contain the Coordinator ID as an additional information in our approach. We use the Coordinator ID as old data was not overridden correctly before, e.g. under churn scenarios, as data sets were associated with the according peer ID. When the tree structure changed and a peer's position changed, old data sets present in the parent node may not have been updated as they were connected to a different peer which did not provide information anymore. The problem is that this old data was still used when aggregating present data resulting in permanently increasing values for cumulative values and the consideration of strongly outdated data. With out solution this problem is not existent as the specific peer which sends the data is irrelevant and only the Coordinator ID matters.

Due to the improved behavior of this version, we will use and extend it further with this thesis. Note that we will not use respectively activate the gossip connections as they enable new attack paths.

5.3 Architecture

The general architecture of our implementation is depicted in Figure 5.1. Note that the `Analyzer` is used for the evaluation of the simulation and will be described in Section 6.2.

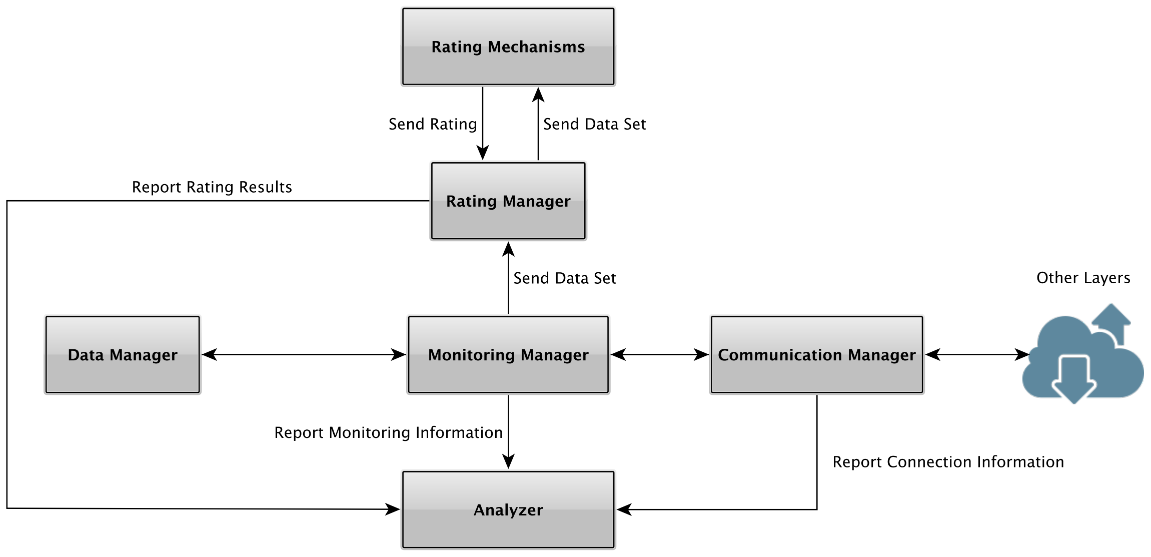


Figure 5.1: Application graph of our implementation done for this thesis

As we extend the already implemented structure, we keep the available communication paths and supplement them with new paths if necessary. We add our rating system DOMiNo represented by the

`Rating Manager` class and connect it with the existing application so that the `Rating Manager` can report its results to the `Analyzer` and the `Monitoring Manager` is able to communicate with it.

As mentioned before, we do not only add new classes but also extend the available classes with new functionality. Figure 5.2 shows the class hierarchy of this thesis. We will describe our additions to each class in detail in the following subsections. Each class containing the Infix *Gossip* was developed during our previous project. Every class with the Prefix *Mal* or *Rating* was developed for this thesis.

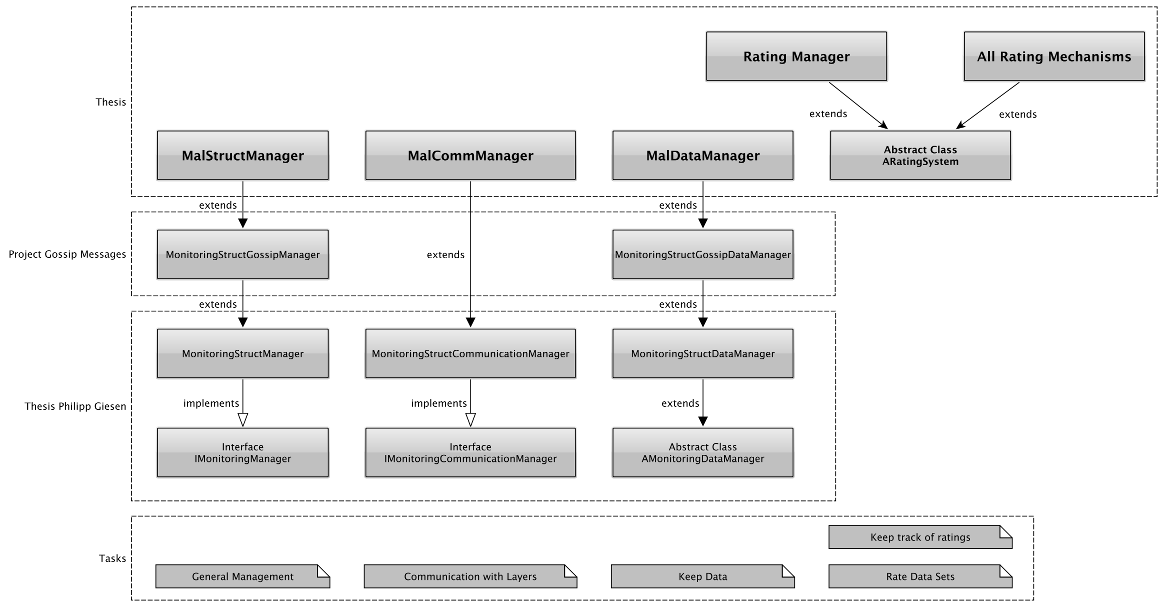


Figure 5.2: Class Hierarchy of the SkyEye.KOM implementation in PeerfactSim.KOM

Next, we will describe each component depicted in Figure 5.1 in detail.

5.3.1 Monitoring Manager

The `Monitoring Manager` class gets created by the application factory defined in the configuration file and represents the peer itself in the monitoring system. That means it keeps all the information a peer needs to participate in the monitoring system, e.g. basic information like the peer ID, the current status or where to find further information via its connections to the `Data Manager` and `Communication Manager`. It is also responsible for reporting the monitoring results to the analyzer[1].

[1]If it is the reporting node, i.e. the root.

On instantiation, the `Monitoring Manager` furthermore handles whether the peer is *malicious* and which attack scenario it uses if it is malicious. If the instance is not *malicious* it may either be *detecting*, i.e. using our rating system or *normal*, i.e. not detecting.

The monitoring algorithm is also implemented in the `Monitoring Manager`. It distributes data up- and downwards the tree and processes incoming messages based on its configured behavior. If our extended `Monitoring Manager` is malicious, it modifies the message forwarding and processing algorithm according to the attack type, e.g. by sending manipulated data or sending data to wrong receivers. If it is detecting it delegates the `Data Manager` to save the received messages for the collecting rating systems and also triggers the `Rating Manager` and its instant rating mechanisms each time a message will be received and its collecting rating mechanisms each time an update interval has passed.

As we did not want to use different *non-malicious* data sending functions for each implemented `Monitoring Manager`, we added our nonce tweak to the original `Monitoring Manager` from Philipp Giesen. Especially since we consider it as an add-on to the normal implementation independently from our rating system.

In our extended implementation of the `Monitoring Manager` we added the following information:

- A variable saving the last timestamp a child message was received.
  This variable is used to determine if a potential attacker (attacking child nodes) could be successful at all.[2]

- Information about how long the node is already tracking data sets.
  This is necessary as we wait a certain time to be able to work with a bigger data base.

- Information how many recent update intervals of data we want to keep.
  We only want to work on recent data since the network's status may change over time.

- A mechanism keeping track if any ACK messages are expected.
  We use this mechanism to determine if a parent node is communicating with us or not.

As we want to use as fresh data as possible, the `Monitoring Manager` also informs the `Data Manager` to clean the saved data sets regularly when an update interval has passed. The `Data Manager` then removes all old data sets being older than a certain threshold, which we configure via

[2]If he does not have any child nodes he cannot.

the configuration file.

Due to the `Monitoring Manager` being the unit holding the monitoring logic it is also the one being aware of timeouts and as an exception triggers the rating mechanism being in charge of timeouts. This behavior is different compared to the other rating mechanisms as it does not rely on the received data, but instead of the **not** received data which only the `Monitoring Manager` is aware of.

5.3.2 Communication Manager

The `Communication Manager` basically handles the connection to the underlying overlay. If a peer wants to join or leave the overlay, this procedure is handled via the `Communication Manager`. Furthermore, it covers the part where a prepared message should leave the application and be sent to another node. It can start the necessary lookups to determine the receiver of a message and passes it then, supplemented with the now received information, to the underlying overlay which handles the rest.

We extend the existing `Communication Manager` to handle messages based on the peers behavior as malicious nodes behave differently compared to a normal node. If it is connected to a malicious peer the received messages will be redirected to another message handling routine in the `Monitoring Manager`. If the peer is not malicious, the message will be passed to the usual routine.

The `Communication Manager` also informs the analyzer about all necessary information regarding the connection status of a peer, i.e. when a node joins or leaves the network, and about outgoing messages for the calculation of the amount of bytes sent through the network.

5.3.3 Data Manager

The `Data Manager` handles and processes all incoming and outgoing data. If a peer receives a message the `Monitoring Manager` will delegate it to the `Data Manager` and if the `Monitoring Manager` wants to distribute a data set to its child or parent node the `Data Manager` will prepare the necessary information which can be sent after. The information sent is explained in detail in the following subsection.

While the basic `Data Manager` only uses the incoming data for aggregation and holds aggregated values our version also keeps a list of received parent and child data used for DOMiNo and its collecting rating mechanisms. Note that we use two different lists as we treat parent data, i.e. global view data spreading downwards, and child data, i.e. aggregated local views spreading upwards, separately.

Furthermore we implemented the logic to manipulate the data, i.e. to provide malicious data sets, also in the `Data Manager`. The `Monitoring Manager` can inform the `Data Manager` that it wants to distribute a manipulated data set, so that it can prepare the modified information. This manipulation can be done either *intelligent* or not (*simple*). Both attack types follow basic rules like the observance of data value thresholds, i.e. if a value can never exceed a certain value due to constrains, an attacker will keep that in mind[3]. The difference is that the simple attacker will manipulate data by a constant factor, e.g. 0.5, while the intelligent attacker manipulates data in a certain interval. We use those two approaches as we state that the more random manipulation of data, i.e. the intelligent attacker, should be harder to detect.

The attackers algorithms are explained in detail later on.

Empty Data Sets We implemented a new type of data set which we called `Empty Data Set`. It is used for the attack AT_{Empty} and represents a set with not values respectively a set with missing values. The `Data Manager` will construct such a data set instead of a normal one if the corresponding attack type is active.

Database To Work On

For our implementation we use the data base provided by the implementation done by [Gie14]. The distributed data in SkyEye consists of four functions: the sinus, a dirac, a ziczac, and a rectangle function. Each peer calculates the current results of each function depending on the current simulator time. The functions are used to benchmark the freshness and the precision of the data present in the monitoring system as the root's information[4] at a certain timestamp can be compared against the expected value it should have.

Each data set does not only contain the current calculated function value but also additional information:

[3]E.g. information about latencies cannot be negative.

[4]Since the root is the reporting instance which also determines the global view.

- Count: Determines how many data sets were aggregated for this data set. While leaf nodes always send the value 1, a node with two leaf nodes would send the value 3 as it aggregated the two child sets plus his own data.
- Max: The maximal value found in all the data sets used to aggregate to create the inspected one.
- Min: Analogous the minimal value.
- Mean: The sum of all received values divided by the count value.
- Sum: All received function values summed up.
- Sum of squares: All received function values squared and summed up.
- Variance: The variance of the data calculated using the sum of squares, count, and sum information.
- Standard Deviation: The square root of the variance.

Note that the values will be newly calculated each update interval. A leaf node sending a new update will not use previous data if it was able to send it before. Further interesting data like the number of hops are available in the simulator's underlying layers, but not in the peer itself. Hence we need to work with the provided functions to test our implementation of DOMiNo. As stated in Section 2.4 and 4.2 we do want to work with a data base as big as possible while only working with fresh, i.e. recent, data. For this we implemented the `Data Manager` to only keep data sets received during the last three update intervals. When a new update interval starts, the `Data Manager` will be informed by the `Monitoring Manager` and removes the data from the oldest one. As a result we can only work with a small data base in exchange for more recent information.

Other Data Management Approaches As mentioned above we want to work with fresh data. If one does not care how recent the data to work with is, one could use a different approach to collect a minimal data base before starting to analyze it.

As an example, an option would be to keep a consequent minimal base for all peers like a data base db at least as big as e.g. $|db| = 3 \times branchingFactor$. With a branching factor of 8 this would mean that each peer collects at least 24 data sets before using any mechanisms operating on the data. In the best case scenario this equals to a waiting time of three update intervals as we do in our approach. But we also need to note that this waiting time can expand up to 24 update intervals, assuming the presence of at least one child node, in the worst case.

Note that this approach is not suited for data which may underlie permanent changes like data about the network status as if may operate on particular outdated data. It may be better suited for data which we do not expect to change much over a certain time interval, but nevertheless, while this approach would offer a bigger data base it is no good trade-off for us as we work with consequently changing data[5].

Simple Attackers

As stated in Section 5.3.3 our simple attackers are able to manipulate data while considering basic rules like staying in predefined intervals if a value respectively entry e cannot exceed a certain threshold, e.g. a function f producing results in the range of $f(x) \in [0,3]$. The used manipulation equation in shown in Equation (5.1). The manipulated value will be set to the respective threshold if it is bigger respectively lower than it. As a result the value may be the same as the original value if the original value was as big as the threshold already. If the original value was close to the threshold the difference between the manipulated data may be rather small.

$$e_{manipulated} = e \times sf \tag{5.1}$$

The attackers manipulate each value with a fixed sizing factor sf defined in the simulation's configuration file. The manipulation algorithm is depicted in Algorithm 3. The attacker, i.e. the attacker's `Data Manager`, prepares a normal aggregated data set and iterates over each function available in the data set. For each entry[6] the value will be multiplied by the predetermined sizing factor. After that the calculated value will reviewed to check if it violates the basic assumption, e.g. exceeding a threshold. If it does the maximal value will be chosen instead. The modified data set will then be passed to the `Monitoring Manager` which composes a message and sends it to the receiver via the `Communication Manager`.

[5]The described functions and later on with the statistics about the network once they are accessible.
[6]The ones described before.

Algorithm 3 Basic creation of a malicious data set

```
1: Input: A data set $d$ to manipulate, a sizing factor $sf$
2: procedure FUDGEDATASET
3:     for all functions $f$ in $d$ do
4:         for all entries $e$ in $f$ do
5:             $e_{new} \leftarrow e \times sf$
6:             if $e_{new}$ is out of bounds then
7:                 $e_{new} \leftarrow boundValue$
8:             end if
9:             put $e_{new}$ in $f$
10:        end for
11:        put $f$ in $d$
12:    end for
13:    return $d$
14: end procedure
```

More Intelligent Attackers

Our more intelligent attackers manipulate the same data described in the previous section. While they also iterate over each entry of the normal data set, they change the contained values by choosing the result from a certain interval. The corresponding equation is denoted in Equation (5.2). The variable r denotes a random number $r \in (-0.2, 0.2)$ which is used to decrease the random added value and therefore the interval from which an attacker may choose the manipulated value. Furthermore it is used to not only add random values but also subtract random values in some of the cases.

$$e_{manipulated} = e \times sf + (|e - (e \times sf)| \times r) \tag{5.2}$$

For this they first calculate what the simple attacker would use as a value, i.e. multiplying the data value by the sizing factor. They use this information the calculate the absolute difference $entryDiff$ between the original and the manipulated value. This difference is used to add or subtract it to the value a simple attacker would use. The whole manipulation algorithm is denoted in Algorithm 4. Note that, as we chose $randomAddition$ to be a fraction of $entryDiff$, we will not be able to produce the original value, again. This may only occur if the original value was already the same as one of the threshold values.

Algorithm 4 More intelligent creation of a malicious data set

```
1: Input: A data set $d$ to manipulate, a sizing factor $sf$
2: procedure FudgeDataSetIntelligent
3:     for all functions $f$ in $d$ do
4:         for all entries $e$ in $f$ do
5:             $e_{tmp} \leftarrow e \times sf$
6:             if $e_{tmp}$ is out of bounds then
7:                 $e_{tmp} \leftarrow boundValue$
8:             end if
9:             $entryDiff \leftarrow |e_{tmp} - e|$
10:            $randomAddition \leftarrow entryDiff \times randomNumber$    ▷ randomNumber $\in (-0.2, 0.2)$
11:            $e_{new} \leftarrow e \times sf + randomAddition$
12:            put $e_{new}$ in $f$
13:            if $e_{new}$ is out of bounds then
14:                $e_{new} \leftarrow boundValue$
15:            end if
16:        end for
17:        put $f$ in $d$
18:    end for
19:    return $d$
20: end procedure
```

5.3.4 Rating Manager

The `Rating Manager` represents our implementation of DOMiNo as it handles all of our implemented rating mechanisms. It receives the incoming messages from the `Monitoring Manager` and distributes them further to each rating mechanism. In return the rating mechanisms send their results to the `Monitoring Manager` which is also connected to the simulator's analyzer where the `Rating Manager` sends the collected results to.

A more detailed view of the realized implementation will be given in Section 5.6.

5.4 Nonces

As mentioned in Chapter 4.1.1 we introduce nonces as an extension to the SkyEye.KOM algorithm. In Section 5.3.1 we already stated that we added nonces to the core of the SkyEye.KOM implementation. For each message, child nodes use a random 32-bit integer which will be sent together with the child's data to the parent node. With this nonce the child is able to check whether it is waiting for an ACK or not. Furthermore it can confirm an incoming message and determine if the message is a correct

ACK or part of an occurring attack, since attackers need to guess the correct nonce to not appear suspicious.

Nonces are implemented part of the `AMonitoringMessage` class which is the base for each implemented message type and can be set via the `setNonce()`-method before sending the message. The `Monitoring Manager` remembers each awaited nonce in a HashSet.

5.5 ACK Awareness

We want nodes not only be able to confirm ACK messages, but also be aware of the situation when an ACK is absent. Finding the parent node is part of the standard message distribution routine. A lookup gets initiated in the `Communication Manager` and the determined parent node is added as the receiver to the message's information.

We extend this behavior to now also remember contact information the message was sent to. We propose that if a receiver did not answer three times in a row, i.e. over the time span of three update intervals, should be considered malicious. Giving a node three update intervals to answer should prevent false positives due to package loss or restructuring of the monitoring system.

We furthermore take the local view as global view due to missing global data to be able to send any data if we noticed that our parent does not answer us. Note that we do not try to get information from another source as our requirement $R_{NoNewPaths}$ aims to keep the original communication structure.

5.6 DOMiNo

In this section we will describe how we implemented DOMiNo in the existing monitoring system's structure. Figure 5.3 shows how the `Rating Manager`, i.e. DOMiNo, is connected to the `Monitoring Manager` and rating mechanisms in detail.

As mentioned in Chapter 4 and this chapter each rating mechanism connect to DOMiNo should provide a rating. We implemented a new abstract class `ARatingSystem` which contains the basic function a rating mechanism must provide, i.e. it should be able to rate data received by parent and child nodes. Such a rating method should return an object called `Rating` which contains information about the rated peer, the rating mechanism, and the determined rating score from the rating

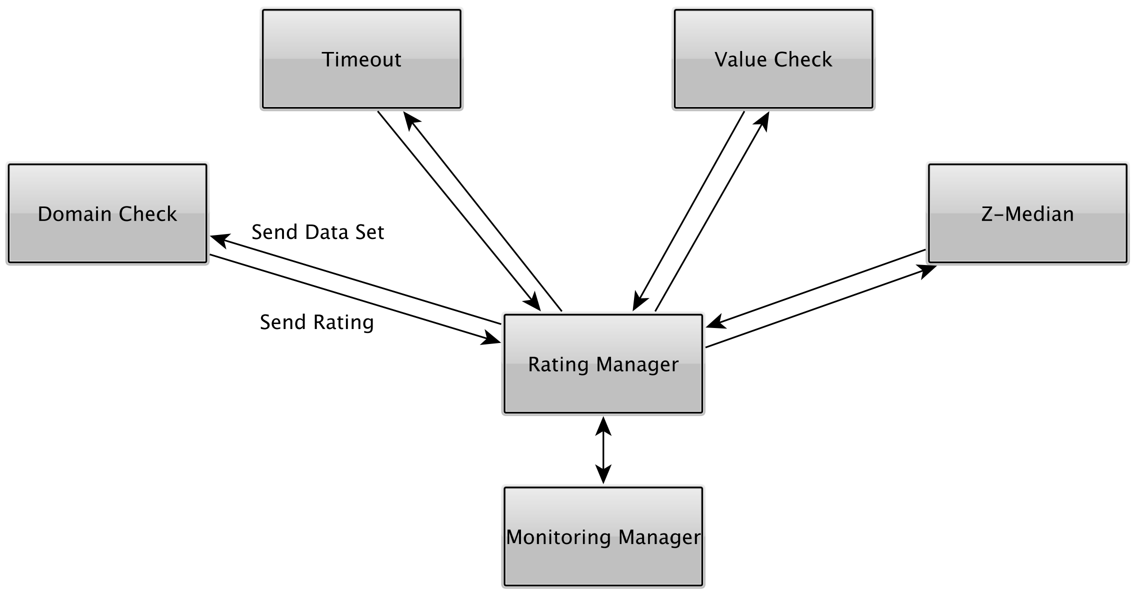


Figure 5.3: Rating Mechanisms in detail

mechanism.

As we want DOMiNo to make an aggregated decision, too, it also extends the `ARatingSystem` class. The difference is that is does not rate the data itself, but the scores it collected from the several rating mechanisms. To be able to compare return ratings better, we normalize all returned rating scores so that a score bigger than 1 denotes a classification as malicious[7]. Otherwise we would need to keep in mind each individual threshold of a rating mechanism to evaluate the received ratings.

The `Rating Manager` itself will be created by the `Monitoring Manager` if it is configured to be a detecting node. The `Rating Manager` then invokes all rating mechanisms which are defined in the simulation's configuration file.

When the `Monitoring Manager` receives a message, it passes the copy to DOMiNo to let the instant rating mechanisms start working. Furthermore every update interval the collecting rating mechanisms will be triggered. These mechanisms will work on the data collected by the `Data Manager`. After they are done DOMiNO uses the calculated ratings per each examined peer and makes a decision itself. For this it uses a threshold itself which determines how high the cumulative rating scores must be, before DOMiNo classifies it as malicious, too.

[7]E.g. as mentioned earlier we take a Z-Score of 3.5 as the threshold for the Modified Z-Median. This means the Rating Manager would take each provided score and divide it by 3.5 to normalize the results

For the collecting rating mechanisms DOMiNo will rate a node only if the corresponding rating mechanisms classified it as malicious multiple times - here we use at least three times denoted via a node-specific counter. If a node gets rated after that, but this time as non-malicious the counter will be decreased. As explained in Chapter 4.2 we use this approach to reduce the false positives which may happen due to the small data base the collecting rating mechanisms need to work with. Furthermore we do not want to punish (statistically) suspicious values instantly. As an example, consider the following two scenarios depicted in Figure 5.1 where + denotes a malicious rating and − a non-malicious rating.

Time point	1	2	3	4	5	6	7
Node 1	+	+	**+**	**+**	−	−	**+**
Node 2	+	+	**+**	−	−	−	+

Table 5.1: Possible successive Z-Median ratings for two nodes

Node 1 will get reported at time point 3, 4, and 7 as it has at least three malicious ratings at these points. Node 2 will get reported only at time point 3. At time point 7 it will have a total cumulative count of 0 and therefore the malicious rating will not exceed the threshold of at least 3 ratings so that no report happens.

In the following sections we will provide additional information regarding the individual rating mechanisms.

5.6.1 Instant Rating Mechanisms

The instant rating mechanisms will be triggered once an incoming message gets proceeded. Each rating mechanism will examine the message and return an individual `Rating`. These ratings will be collected by the `Rating Manager` and reported to the `Analyzer`. The `Rating Manager` itself uses the ratings to make a decision once the collecting rating mechanisms provided a rating, too.

As mentioned in Chapter 4.4 these mechanisms are able to trigger the `Data Manager` to not use obviously malicious data for the collecting rating mechanisms. This can be done via the `Rating Manager` which will inform the `Monitoring Manager` to prevent the `Data Manager` from doing so.

Domain Check Rating The algorithm used is already depicted in the Algorithm 1 in Section 4.1.1. While the rating of child nodes makes use of the given Domain ranges, the rating of parent nodes will rely on the nonces to determine if a sender is the correct parent node.

Value Check Rating In the current implementation we need to be aware we cannot check mean since it is calculated by $mean = \frac{sum}{count}$. Since the nodes do not react on false values yet, forged data by child nodes may lead to mean being less respectively more than the min respectively max value. Imagine an attacker doubling the count value but not the sum value. A node using these values to aggregate its data and distribute it to other nodes will send values which do not fit the minimum bound anymore. Because of this we decided to exclude all checks on values which are based on a further calculation in our implementation. Our implementation contains checks for the minimum respectively maximum value provided and the knowledge about the used function, e.g. the sinus function should produce results in the interval $[-1,1]$. Additionally, we check if the minimal value is smaller than the maximum value.

Further behavior needs to be added when the nodes themselves are able to react to false values.

Timeout Rating The `Timeout Rating` is an exception as it does not operate on data but gets triggered via the information provided by the `Monitoring Manager`. The `Monitoring Manager` will wait three update intervals before triggering the `Timeout Rating` since there may also be problems present in the overlay as nodes switch position or package loss occur, which we need to be aware of. If a timeout will be reported, the `Timeout Rating` will return a corresponding `Rating` to the `Rating Manager`.

5.6.2 Collecting Rating Mechanisms

As mentioned in Section 5.3.3 we cap the number of recent data to contain only data from the last three update intervals to ensure a certain freshness of the data we use. The collecting rating mechanisms get activated by the `Rating Manager` in each update interval. They work with the data collected by the `Data Manager`.

Z-Median Rating The implemented algorithm is already depicted in Algorithm 2 in Section 4.2. We realized the algorithm as proposed by Iglewicz and Hoaglin [IH93] with a threshold of 3.5. We extended the original approach by using the tweak by IBM [IBM] in case more than half of all data points hold the same value. We let the Z-Median rating work with the mean values provided by the sinus function. The mean value represents the calculated value of the function a node holds at a certain time. It is the only value of our available data base which we can expect to be almost the same for each peer and which changes only slowly. We expect this behavior to fit real network status data best, as values like round trip times should be similar for each peer and usually should not change suddenly by an significant amount.

In the next section we want to present our simulation setup and discuss our evaluation results. We defined different scenarios using one or several attacks and different node ratios to examine how good our approach works.

Chapter 6

Evaluation

To evaluate our developed rating system we want to simulate several scenarios using our implementation. We want to find out how well DOMiNo and the specific rating mechanisms perform depending on the used attack scenario. As an evaluation metric we want to distinguish between positive detection results, i.e. malicious nodes getting reported, and false positive detection results, i.e. non-malicious nodes getting reported.

In this chapter we present our simulation setup and the results. The used parameters for the simulations are described in Section 6.1. We give an overview about the created attack scenarios in Section 6.1 and explain how the analyzer will evaluate our results in Section 6.2. We configured different scenarios which we describe and evaluate in Section 6.3.

Next we want to start with the detailed description of the different used simulation parameters.

6.1 Parameters

All parameters can be configured via the settings in the XML-based config file. We added new parameters to the config file if it was necessary, e.g. the setting for the sizing factor. The different actions which should be executed at a specific time are denoted in the corresponding action file which is depicted in Figure 6.1.

As the configuration file is very extensive we only describe the relevant parameters for our simulation results. For our simulations we used a network consisting of 5000 different nodes participating in a Pastry [RD01] overlay. We ran a total of ten simulations for each scenario, using a different seed for the random generator in each run. At the end we aggregated the results for each scenario.

```
reporter 1s MonitoringApplication:join
all 3s-60m MonitoringApplication:join
reporter 90m MonitoringApplication:distributeMonData
all 90m-91m MonitoringApplication:distributeMonData

all 60m-80m FileSharingApplication:publishResourcesFromSet files 10
all 81m FileSharingApplication:lookupResourceFromSetPeriodically files 10m

all 95m MonitoringApplication:activateDataSetTracking
all 100m MonitoringApplication:activateDetection
```

Figure 6.1: Used action file for our simulations

We simulated a network without churn active over a total of 200 minutes with the first 60 minutes being the join phase where all nodes join the overlay and start distributing data in the monitoring system in the interval from minute 90 until minute 91. For SkyEye.KOM we used a branching factor of 8 to increase our data base, i.e. the potentially available child data, in each node. Nodes distribute monitoring data every 30 seconds. In parallel we had a small filesharing application running, starting lookups in the overlay every 10 minutes to keep the routing tables in Pastry up-to-date. Each detecting node started tracking incoming data sets for the collecting rating mechanisms of our rating system DOMiNo at minute 95. From minute 100 ongoing, each detecting node starts with the rating phase. Note that, as stated before, malicious nodes do not use DOMiNo.

For the sinus curve we decided one measure interval, i.e. a sinus curve's length, that means the time until a value repetition occurs, to be 50 minutes, so that we fit exactly two curves in our detection time. We chose the time to be rather long to reduce the relative data value changes each update interval. Small values, e.g. using 5 minutes, would result in significant changes every single update interval whereas an measure interval of 50 minutes enables us to see the changes in the curve more precisely and to understand specific effect better.

Tabular 6.1 summarizes the chosen parameters we just described for the general configuration. We describe the parameters we used for the attacking nodes in Section 6.1.

Simulation Scenarios

In our simulations we created different scenarios differentiating in the configuration of which and how many attackers are present in the monitoring system. We ran one scenario without any attacker present to see if our rating system produces false positives without any attack happening which we present in Section 6.3.1 and furthermore scenarios for each attack type being the only attack type present which are shown in Section 6.3.2. Additionally, we also ran simulations where each attacker

Parameter	Value(s)
Number of nodes for the simulation	5000
Number of runs per scenario	10
Churn	Inactive
Simulation time	200 minutes
Used P2P overlay	Pastry
Time for one whole sinus curve	50 minutes
Update interval	30 seconds
Branching factor	8
Data manipulation behavior	Simple, Intelligent
Data set sizing factor	0.25 (only simple attackers), 2
Attack types	None, Each attack type separately, All together chosen randomly
Parameters when using one attack type separately	
Ratio of malicious nodes	1%, 5%, 10%
Parameters when using all attacks	
Ratio of malicious nodes	8%, 20%
Specific actions during the simulation	
Starting point for SkyEye.KOM	Minute 90 until 91
Starting point for DOMiNo	Minute 100

Table 6.1: Parameters chosen for our simulations

will choose a random attack type at the start. These scenarios are evaluated in Section 6.3.3. Note that an attacker does not switch his attack type during a simulation run. As noted in Tabular 6.1 we used different ratios for the present nodes in each scenario. A scenario with only one attack type present was configured with a total of 1 percent, 5 percent, and 10 percent of malicious nodes. If we let the malicious nodes choose their attack type randomly (once at the beginning), we used a total of 8 respectively 20 percent of malicious nodes. That means we aimed to get about 1 percent respectively 2.5 percent for each of the 8 attack types.

If attack types were used which manipulate data we also made adjustments regarding the used sizing factor and chose either intelligent or simple attackers as described in Section 5.3.3 and Section 5.3.3. Simple attackers manipulated the data to be a fourth of the original size in one scenario and doubled the values in the other. For the intelligent attackers we only used the scenario where the values get doubled initially and then a value out of a specific interval is chosen.

6.2 Preliminary Notes

As hop counts or other network statistics are not available at the moment in SkyEye.KOM we had to use the implemented functions mentioned in Chapter 5.3.3. We chose to use the implemented sinus function to be able to work on data without significantly value changes during each update interval. In contrast, as an example, the rectangle function would consist of flat lines with sudden changes jumping from 0 to 1 or vice versa.

We also need to note that some attackers can not be successful against the monitoring mechanism. This would be the case if an node using attacks against child nodes would participate in the monitoring system as a leaf node. We will not consider those attackers for our results. Furthermore note that malicious nodes are **not** using our rating mechanism. That means if a malicious node attacks another malicious node it cannot be reported.

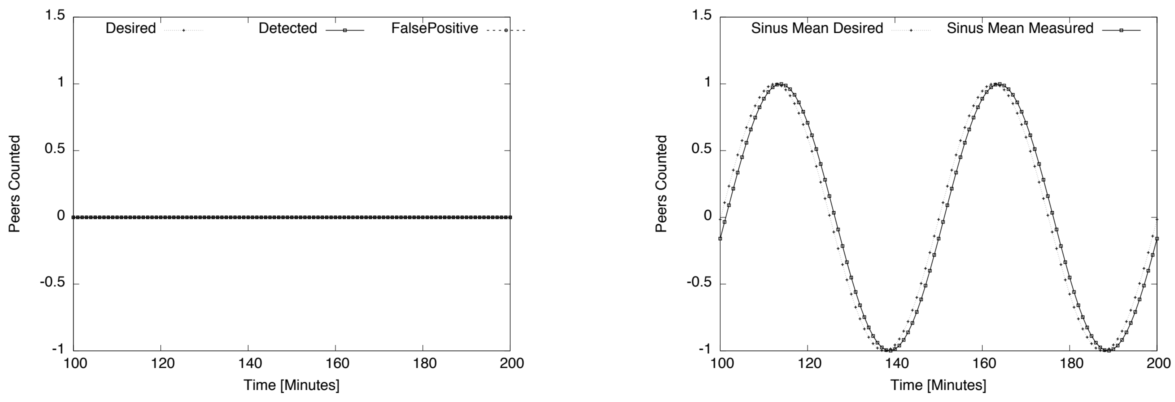
We modified the existing metric to work with our reported ratings. At the start of the simulation each node reports its own type respectively behavior to the analyzer, so that we can determine our desired values for the detection. As mentioned in the introducing sentences, we do not count any attackers which cannot be successful, e.g. leaf nodes attacking only child nodes. To be more specific this affects the attack types AT_{Child} and $AT_{Upwards}$ which only launch attacks against child nodes. For this each attacking node checks if it received any child messages during the last three update intervals. If it did not, it will report to the analyzer that it is an inactive attacking node marking its attack type as *None*. Such a node may reactivate itself once it receives any child data.

Based on the information the analyzer has about each node it will classify received ratings as false positive or positive. Nodes reporting that another node is malicious may revoke their report by sending a report claiming that the node is not malicious (anymore). A node reported by several other nodes will be only counted once. But, it will be counted as long as at least one other node reports it as malicious.

In the next section we will present our results and evaluate each scenario.

6.3 Results

In the following sections we present the results of the different simulated scenarios. As we had 55 different combinations overall we mainly present the most meaningful, i.e. the worst case, results and discuss possible reasons for the behavior. We mostly will focus on the plots for DOMiNo to provide



(a) Results of DOMiNo (b) Sinus values reported by the root node

Figure 6.2: Simulation without attackers

a better overview. Nevertheless, we will describe which rating mechanisms produced the results in each scenario.

Additionally we also show the resulting sinus curves for a scenario to show how the reporter, i.e. the root node, was affected by an attack, if something happened in contrast to the scenario without attackers.

Concerning the sinus curves we will use the term *turning point* describing a point in the plot where the direction of the sinus curve changes, i.e. reaches its local extrema 1 and -1, respectively.

We want to mention that we expect that the attackers sending a fourth of the original size should be harder to detect as the relative difference between the values is smaller.

We determine the false positive ratios by calculating $rate_{falsePositive} = \frac{\#falsepositives}{\#nodes - \#attackers}$ and the false negative ratios by calculating $rate_{falseNegative} = 1 - \frac{\#detected}{\#desired}$.

6.3.1 Scenario without attackers

This scenario is mainly used to test the whole implemented system. As we can see in Figure 6.2a our rating system does report neither positives nor false positives as intended. Figure 6.2b shows only the freshness and precision of the data in the root node in this case. Among other things, the freshness is determined by the update interval. We can see the small offset on the x-axis since the monitoring system needs some time to distribute the data to the root node.

6.3.2 Scenarios with one attack type

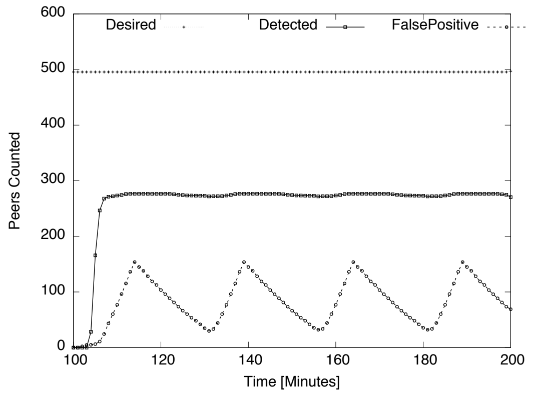
In this section we evaluate the scenarios with only one attack type present. As mentioned before we used a malicious node ratio of 1, 5, and 10 percent, respectively. If data got manipulated attackers either used a sizing factor of 0.25 (simple) or 2 (simple and intelligent).

Attackers using AT_{Parent}

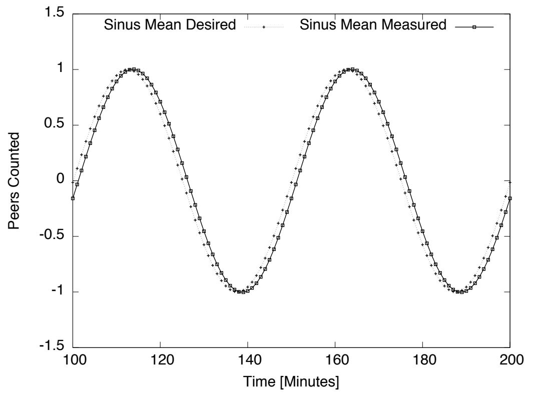
Figure 6.3 shows the results for DOMiNo respectively the sinus values present in the root node for different parameters when attackers send modified data upwards the tree. The results of DOMiNo are based on the ratings reported by the Z-Median mechanism. The other mechanisms did not make any reports. As expected Figure 6.3a and 6.3b with attackers using a sizing factor of 0.25 show the worst results. We expected this because the relative value changes are smaller than using a sizing factor of 2. More important we also need to note that these value changes happen consequently. In contrast to this the increasing of data point sizes may be capped by the interval bounds. E.g. if a malicious node doubles a sinus value of 0.7 it will not send 1.4 but 1.0 as a value instead. Our detection rate is about 55 percent in this scenario respectively our false negative rate is 45 percent. We note that the false positives show a specific pattern with spikes around each turning point. The turning points are specifically critical as the distance between two points from two different time points is rather low at these points. While a sinus curve falls and rises consequently, the differences will almost stagnate there. Therefore even small data point difference may look suspicious. Unfortunately, this is one of the problems when using the sinus curve for our evaluation. The clinched function outputs around the turning points will result in suspicious looking values even when only small differences occur. This is a general problem regarding functions which operate in specific intervals and reach the thresholds regularly.

Regarding the sinus curve, it may be counterintuitive at first that the root node sends correct average sinus values although attackers send modified values upwards the tree. The reason lies in the calculation of the mean value in each parent node. Although parent nodes receive the malicious average values and the rating mechanisms compare these received values, a receiving node will calculate the value again while aggregating the received data in the implementation of SkyEye.KOM. It will use the sent count and sum values as follows: $mean = \frac{sum}{count}$. As the values get manipulated by the same constant factor, the newly calculated mean will still be correct. This also means that in case of simple attacks non-malicious nodes will not spread wrong values further through the tree.

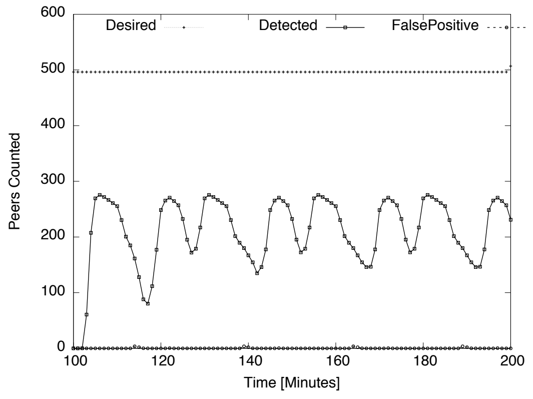
We can see in Figure 6.3c and 6.3d that this attack results in almost no false positives. We only get



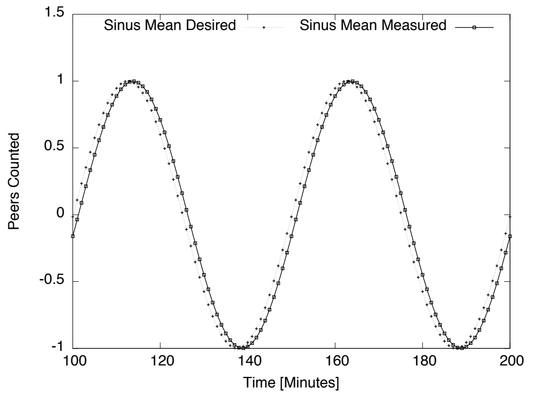
(a) Results of DOMiNo, sizing factor = 0.25, simple attackers, ratio of 10%



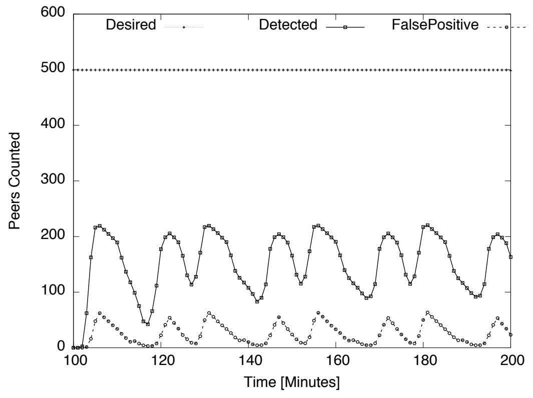
(b) Sinus values reported by the root node, sizing factor = 0.25, simple attackers, ratio of 10%



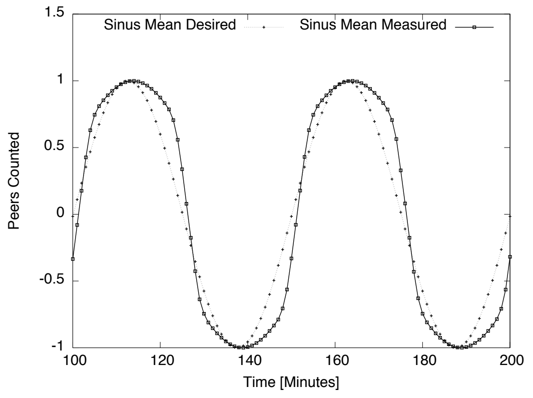
(c) Results of DOMiNo, sizing factor = 2, simple attackers, ratio of 10%



(d) Sinus values reported by the root node, sizing factor = 2, simple attackers, ratio of 10%



(e) Results of DOMiNo, sizing factor = 2, intelligent attackers, ratio of 10%



(f) Sinus values reported by the root node, sizing factor = 2, intelligent attackers, ratio of 10%

Figure 6.3: Attackers sending malicious data sets to their parent node

few false positives around the turning points of the curve. More interestingly the positive detection now shows a specific pattern, too, with detection rates between approximately 20 and 55 percent respectively false negative rates between 45 and 80 percent. Among others the detection drops shortly after the turning points. This is reasonable as the malicious values will be capped at those points and therefore are close to the valid data points. E.g. if the sinus curve is above 0.5 the malicious nodes will send 1.0. The closer we get to the turning point at 1.0 the more alike the valid data points will get. The malicious data points will not look suspicious in this case. As the collecting rating mechanisms rate with a short delay and the previously malicious detections need to be reduced, the drop happens with a delay.

If we take a look at the intelligent attackers in Figure 6.3e and 6.3f we can see the same pattern in the detection as before. Additionally the false positives show the same pattern just weaker. The detection rates are smaller in this scenario, ranging between approximately 10 and 44 percent. This means we get a false negative rate of 90 percent at the negative spike and about 56 percent around the spikes. Regarding the sinus curve we can finally see some influence as the sum and count values will not be scaled equally by the attackers.

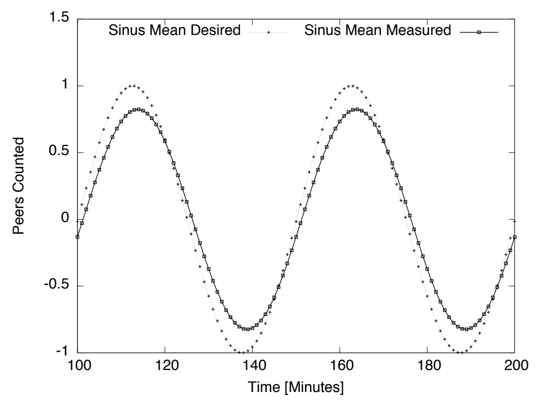
In the next section we will discuss the results of the attack scenarios sending wrong data to child nodes.

Attackers using AT_{Child}

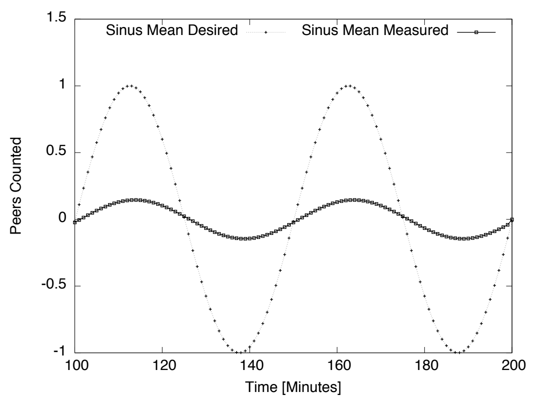
As we did not develop a mechanism to detect wrong values send by parent nodes DOMiNo showed neither positive nor false positive reports. Only the value check rating mechanism could have detected malicious behavior if we would use attackers which do not follow our previously mentioned basic rules.

Regarding the sinus curves depicted in Figure 6.4 we unexpectedly see influence of malicious parents on the reporting root node. Due to that, we strongly assume that the reporting root node was also Coordinator on further levels than level 0. As the number of nodes in the monitoring system grows, it is obvious that smaller Coordinator ranges will occur. Given the smaller Coordinator ranges, the root node is more likely to be a Coordinator more than once.

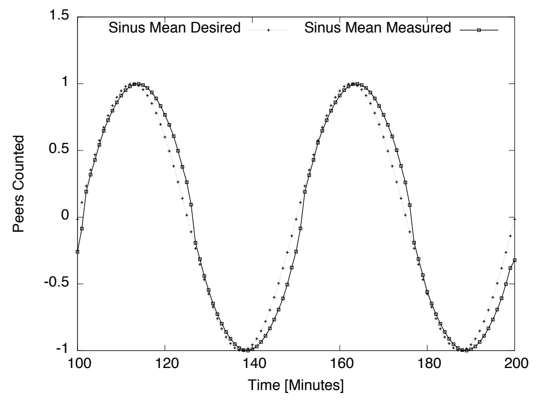
While this would explain how the reporting node was able to receive manipulated parent data, the reporting node should be configured to report **only aggregated child data** and not consider available parent data.



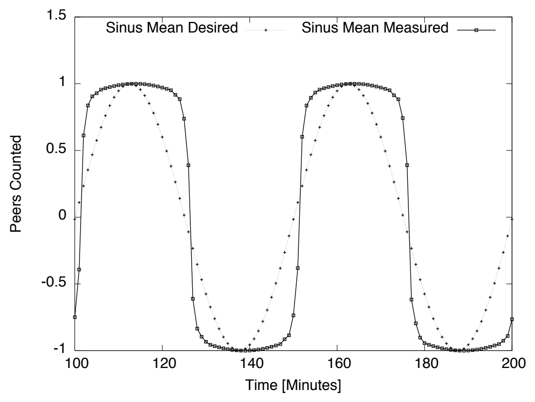
(a) Sinus values reported by the root node, sizing factor = 0.25, simple attackers, ratio of 1%



(b) Sinus values reported by the root node, sizing factor = 0.25, simple attackers, ratio of 10%



(c) Sinus values reported by the root node, sizing factor = 2, simple attackers, ratio of 1%



(d) Sinus values reported by the root node, sizing factor = 2, simple attackers, ratio of 5%

Figure 6.4: Attackers sending malicious data sets to their child nodes

In the context of this thesis we were not able to determine the reasons for this behavior.

Attackers using AT_{Random}

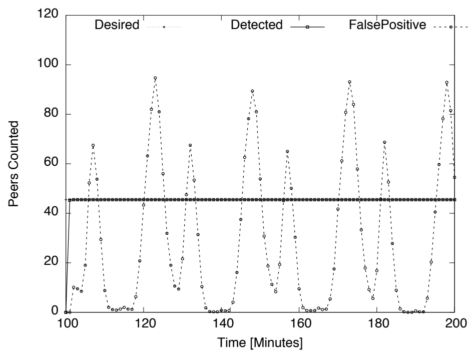
Figure 6.5 shows different malicious node ratios for intelligent attackers. We included all three scenarios as the results differ noticeably.

While our detection rate is at 100 percent, respectively our false negative rate at 0 percent, with the domain check mechanism the Z-Median rating produces false positives as shown in Figure 6.5c and Figure 6.5d exemplary. The previously patterns are present again which we associate with the behavior of the sinus curve. We do not have any positive detection results by the Z-Median as we determined that nodes need to be rated as malicious for at least three times at a node. As the malicious nodes change the receiver every round it is unlikely that they address the same node twice during a simulation. Nevertheless those random messages still have the influence that the regular child nodes look suspicious as the values do not fit to each other.

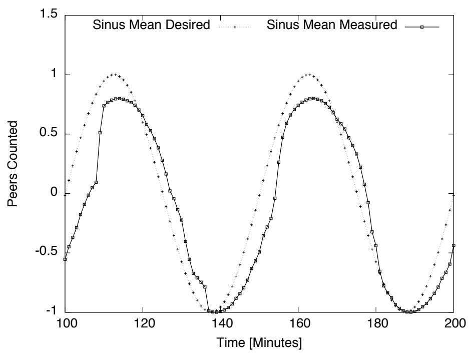
Note that many data sets are already excluded by the domain check rating mechanism as the instant rating mechanisms will force the node to block the data set so that the collecting rating mechanisms do not rate them. The messages going through and being evaluated by the Z-Median rating mechanism are messages coming from random child nodes from inside the own domain range, as the domain check mechanism only blocks messages from outside the Domain range.

As the number of malicious nodes grows it is more likely that message circles may occur in the monitoring system. We can see strange behavior in Figure 6.5h where 10 percent of the nodes are malicious. We assume that one or multiple message circles lead to a chain reaction so that malicious values accumulated rather fast.

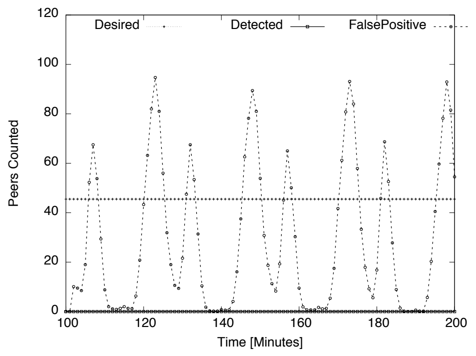
Regarding the simple attackers in Figure 6.6 we can denote that the sizing factor of 0.25 produces worse results, again. While the detection and false negative rate stays at a perfect result, the false positives thrown by the Z-Median rating mechanism even exceed the positive detections by the domain check rating mechanism. In particular, for our worst case, we got a false positive rate of about $rate_{falsePositive} = \frac{350}{5000-250} \approx 7.4\%$ at the peaks of Figure 6.6c.



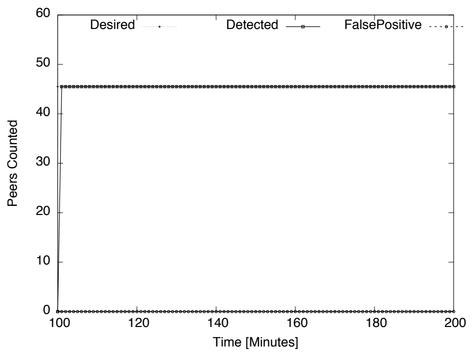
(a) Results of DOMiNo, sizing factor = 2, intelligent attackers, ratio of 1%



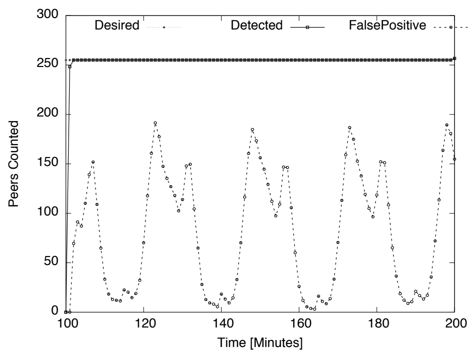
(b) Corresponding sinus values reported by the root node



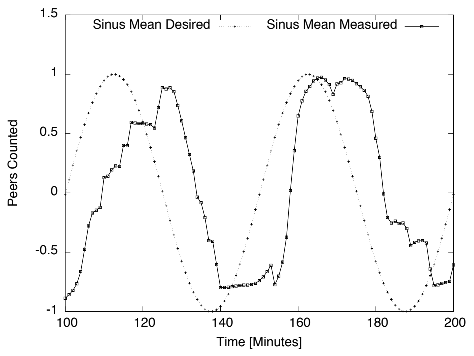
(c) Corresponding results of the Z-Median rating mechanism



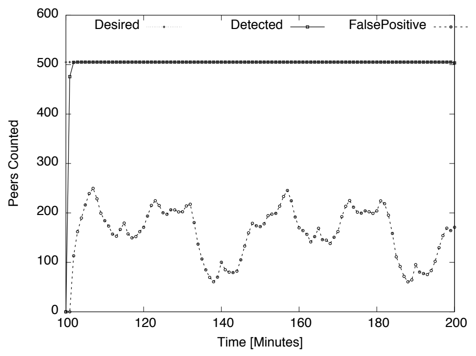
(d) Corresponding results of the domain check rating mechanism



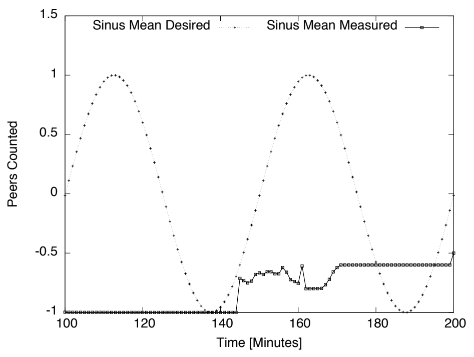
(e) Results of DOMiNo, sizing factor = 2, intelligent attackers, ratio of 5%



(f) Sinus values reported by the root node, sizing factor = 2, intelligent attackers, ratio of 5%

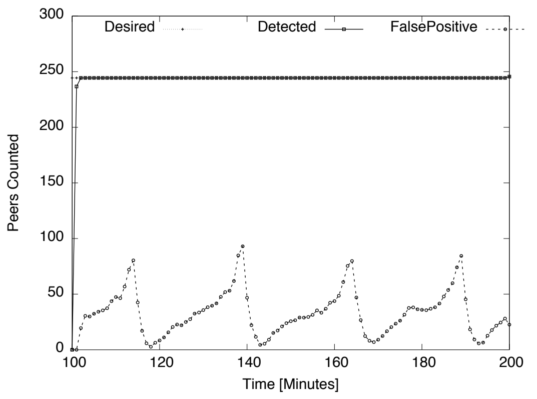


(g) Results of DOMiNo, sizing factor = 2, intelligent attackers, ratio of 10%

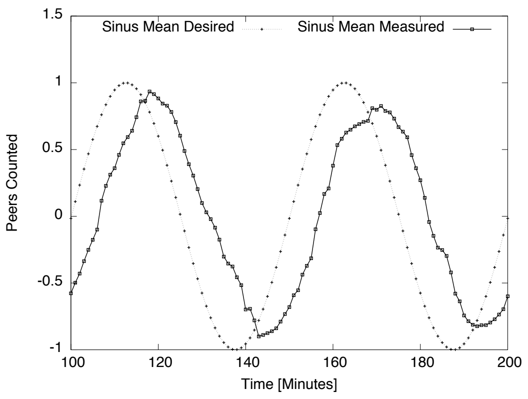


(h) Sinus values reported by the root node, sizing factor = 2, intelligent attackers, ratio of 10%

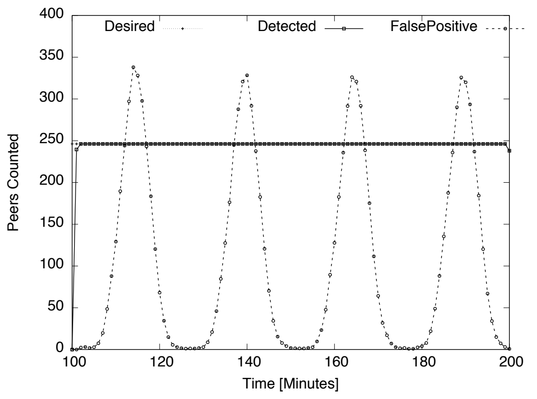
Figure 6.5: Intelligent attackers sending malicious data sets to random contacts



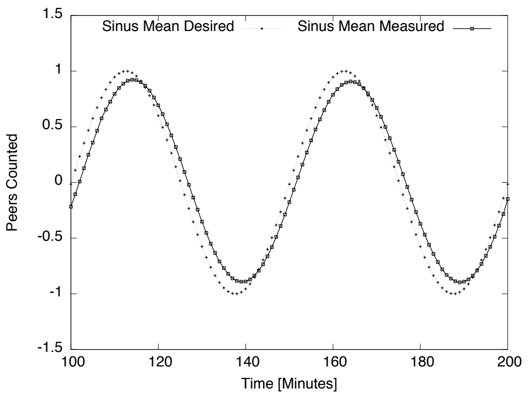
(a) Results of DOMiNo, sizing factor = 2, simple attackers, ratio of 5%



(b) Sinus values reported by the root node, sizing factor = 2, simple attackers, ratio of 5%

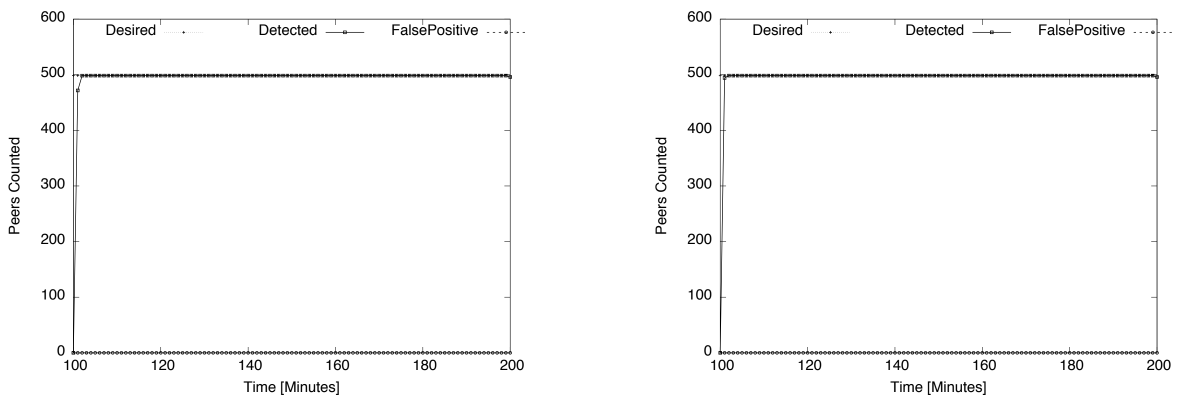


(c) Results of DOMiNo, sizing factor = 0.25, simple attackers, ratio of 5%



(d) Sinus values reported by the root node, sizing factor = 0.25, simple attackers, ratio of 5%

Figure 6.6: Simple attackers sending malicious data sets to random contacts



(a) Results of DOMiNo (b) Results of the domain checking mechanism

Figure 6.7: Attackers sending random ACK messages

Attackers using $AT_{RandomACK}$

As we can see in Figure 6.7 we get a detection rate of 100 percent with our newly introduced nonces. That means that we are able to detect attackers sending global data randomly through the network without further effort. Hence, our false negative rate is 0.

These results are independently of the ratio of malicious nodes available in the network. As we block messages from wrong parents and do not consider them for the further data distribution the monitored values will not get influenced by this attack. The small delay in the detection of DOMiNo results from the time it waits until the collecting rating mechanisms were able to rate.

Attackers using AT_{Root}

As we expected the regular child nodes got rated malicious by the Z-Median rating mechanism. If we take a look at Figure 6.8b as all malicious attackers send their data to the root node directly the root node will have a data base which contains a large amount of malicious data and the data from its child nodes. In our case the number of malicious nodes exceeds the number of child nodes by a large amount. While the root node may be Coordinator on several levels and therefore have more than $\#branchingfactor$ nodes, even with only 1 percent of malicious nodes we have around 50 malicious sender in our network. The large amount of malicious data will look less suspicious than the regular data as it builds the majority of the data points. As a result we got a detection rate of 0 percent, respectively 100 percent false negatives. The false negative rate is about 0.2 percent, i.e. all the honest child nodes of the root.

As the root's Domain range spans the whole ID space we cannot determine malicious nodes by their ID at the moment. To detect them a mechanism which is aware of the topology would be necessary. We can imagine an approach using additional lookups to gain more information about the topology which we did not want to use due to our requirement $R_{Overhead}$

A possible approach would be to extend the domain check rating mechanism to check for each node ID which is the closest one to the expected Coordinator IDs from the child nodes. We need to be aware that the root may be Coordinator on several levels which makes the determination of the proper child Coordinator IDs harder.

Figures 6.8c until 6.8f show unexpected behavior as DOMiNo's results should be conform with the Z-Median mechanism's results. We still need to investigate this behavior. Nevertheless, we can see the sinus curve's characteristics again, as the false positive results disappear around the turning points. We expected this as the attacking nodes will consequently send a value of 1 and -1, respectively, in this time frame and the normal nodes will send the correct data points around this range, too.

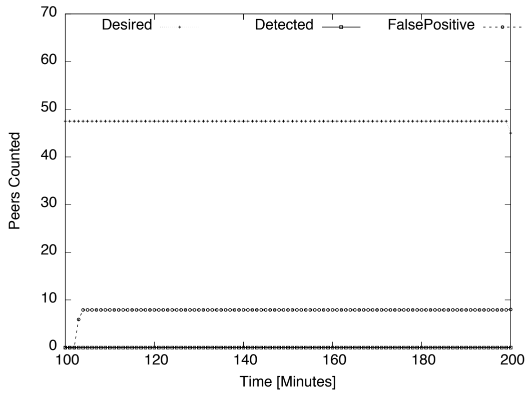
Attackers using AT_{Empty}

Figure 6.9a shows a short delay in contrast to Figure 6.9b as DOMiNo will make its decision once the collecting rating systems were able to deliver a rating, too.

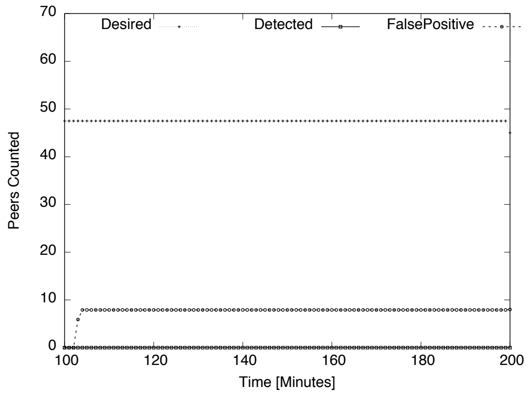
Figure 6.9b depicts the behavior of our timeout rating very good. As we see the malicious nodes get rated the first time at minute 102; 2 minutes after the detection started. This is due to the fact that each node waits for three update intervals to receive an ACK message before reporting a timeout. Regarding the update interval of 30 seconds it shows exactly the behavior we expect. More precisely the timeouts should occur around minute 101 and 30 seconds[1] but we need to keep in mind that the plots are done in minute intervals.

The resulting values may seem low at first sight, but due to the characteristics of the monitoring tree about half of the attackers should be leaf nodes. As we can only detect parent nodes that do not participate with the timeout rating mechanism, we can expect the results to be about the half of the desired value. If we also keep in mind that, as stated in Section 6.2, the other malicious nodes do not report other malicious nodes, the values seem to be optimal regarding this mechanism. We can see this fact regarding the leaf nodes again in the evaluation of attack type $AT_{Upwards}$ where significantly fewer malicious nodes are considered for the desired value than the number of malicious nodes being present.

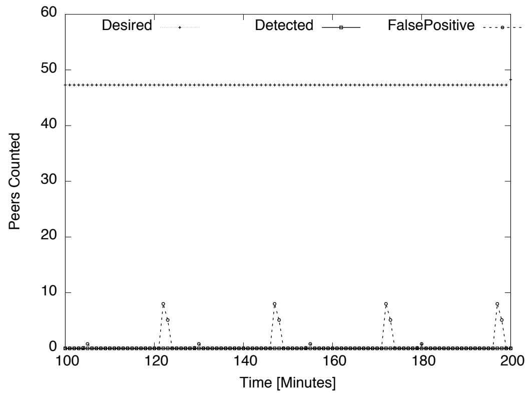
[1]Detection start time is at minute 100 plus 3 update intervals.



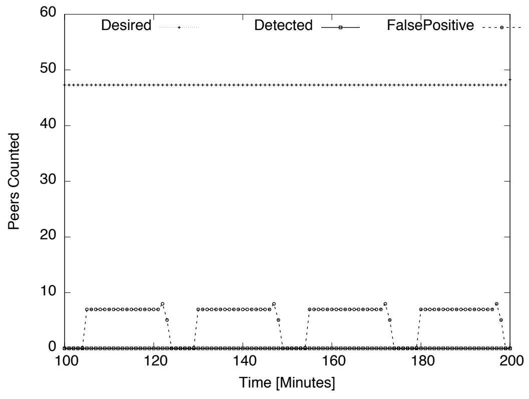
(a) Results of DOMiNo, sizing factor = 0.25, simple attackers, ratio of 1%



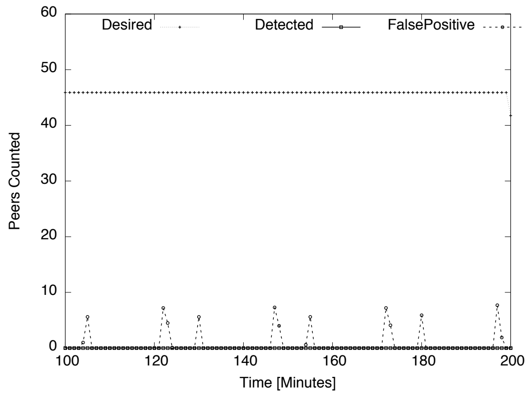
(b) Results of the Z-Median rating mechanism, sizing factor = 0.25, simple attackers, ratio of 1%



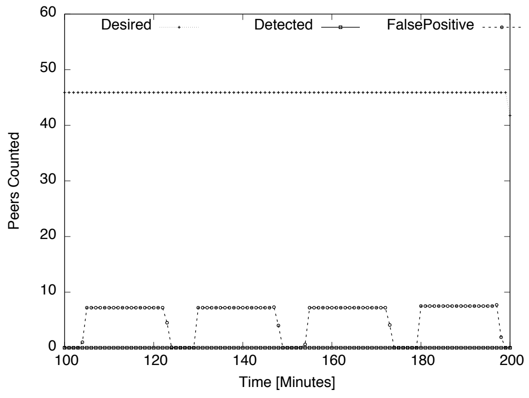
(c) Results of DOMiNo, sizing factor = 2, simple attackers, ratio of 1%



(d) Results of the Z-Median rating mechanism, sizing factor = 2, simple attackers, ratio of 1%

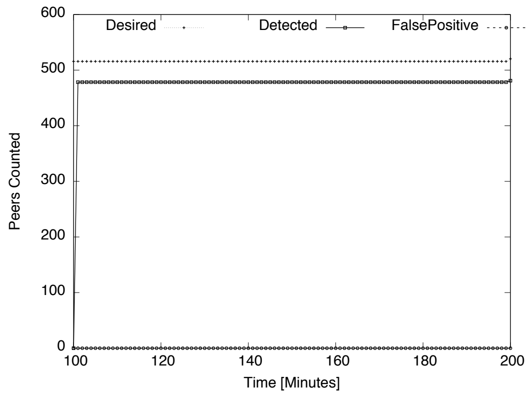


(e) Results of DOMiNo, sizing factor = 2, intelligent attackers, ratio of 1%

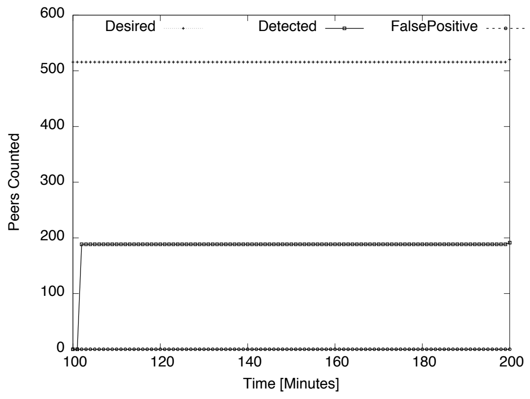


(f) Results of the Z-Median rating mechanism, sizing factor = 2, intelligent attackers, ratio of 1%

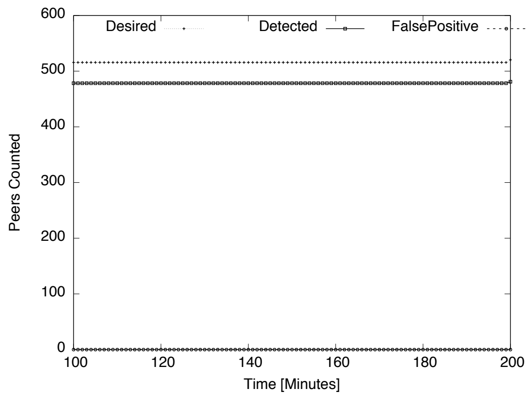
Figure 6.8: Attackers sending malicious data sets to the root directly



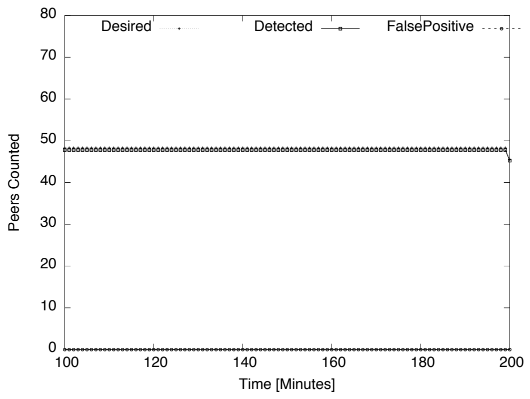
(a) Results of DOMiNo, ratio of 10%



(b) Results of the timeout rating mechanism, ratio of 10%



(c) Results of the value check rating mechanism, ratio of 10%



(d) Results of the value check rating mechanism, ratio of 1%

Figure 6.9: Attackers sending empty data sets

Figure 6.9c shows that the value check mechanism is close to perfect with a detection rate of about 90 percent which results in a false negative rate of 10 percent. The assume the missing detection is due to the other malicious nodes again which do not use DOMiNo. Our other simulations show for the other malicious node ratios that the lower the ratio the higher our detection is. Using only 1 percent malicious nodes the detection is almost at 100 percent (0 percent false negatives) as seen in Figure 6.9d.

Empty data sets have no influence on the sinus curve since the average value is calculated and information is still spread by the remaining non-malicious nodes.

Attackers using $AT_{Downwards}$

We did not simulate this attack as we cannot detect this attack with our implemented solutions. Furthermore this attack does not manipulate any data so that the simulations would not have differed from the simulation without any attack.

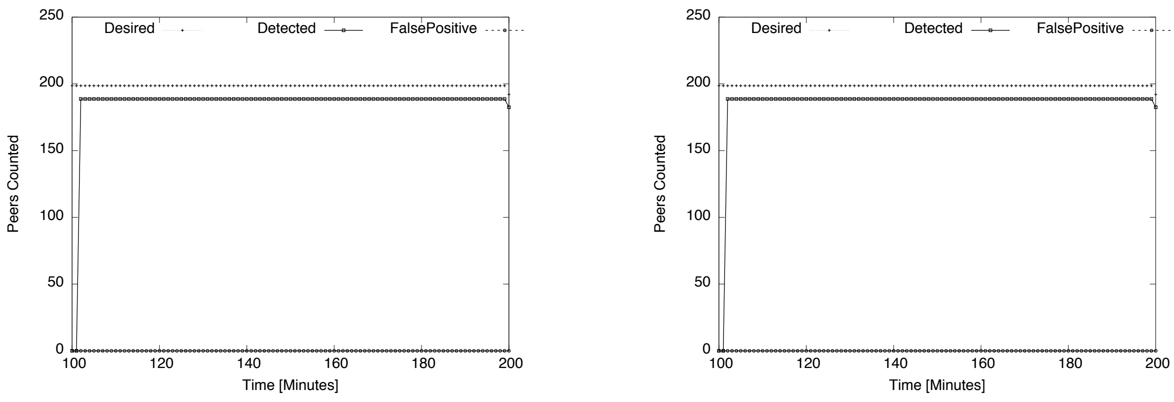
Due to the push-based approach of SkyEye.KOM one could only detect such an attack if he is aware of the nodes present in the network and which node should be his child node. As the aim of the monitoring system is that a node does not need to be aware of the majority of the network we think that this would not be reasonable. Additionally it violates our requirement $R_{Overhead}$.

Attackers using $AT_{Upwards}$

As seen in the evaluation of AT_{Empty} and Figure 6.9 some attackers did not get detected as other malicious nodes do not rate. The less malicious nodes we had the closer we were to 100 percent, i.e. a false negative rate of 0 percent. With a ratio of 10 percent as depicted in Figure 6.10a we had a detection rate of about 95 percent respectively a false negative rate of about 5 percent.

$AT_{Upwards}$ has no effect on the sinus curve, too, as root gets provided with information and only child nodes do not get information back.

As we can see there are only about 200 nodes as the desired value although we defined a ratio of 10 percent, i.e. about 500 nodes. This is due to the fact that nodes not having a child node will report themselves as inactive as they cannot use their attack successfully. Note that the chance to be a leaf node correlates with the value of the branching factor. For tree of height n and branching factor bf on level n there may be bf^n nodes and on the other levels above there may be $\sum_{i=0}^{n-1} bf^i$ nodes. For a branching factor of 2 that means about 50 percent of the nodes are leaf nodes in a complete tree. Whereas using a branching factor of 8 about 86 percent of all nodes are leaf nodes **if** the tree is filled completely. In the simulations it is about 60 percent since we do not have tree structures that are completely filled. Regarding our branching factor of 8, the number of 5000 nodes is too small. In the best case scenario, i.e. the tree is balanced perfectly, we would need $\#nodes = \sum_{i=0}^{4} 8^i = 4681$ or $\#nodes = \sum_{i=0}^{5} 8^i = 37449$ nodes to have a completely filled tree. We observed this ratio when evaluating the attack type AT_{Empty}, too, where the timeout rating mechanisms only found about 40 percent of the malicious nodes, i.e. had a false negative rate of around 60 percent.



(a) Results of DOMiNo (b) Results of the timeout rating mechanism

Figure 6.10: Attackers sending data sets only upwards, ratio of 10%

Attackers using $AT_{DoNothing}$

As described in the evaluation of AT_{Empty} some attackers did not get detected as other malicious nodes do not use the rating system. The less malicious nodes we had the closer we were to a detection rate of 100 percent respectively a false negative rate of 0 percent.

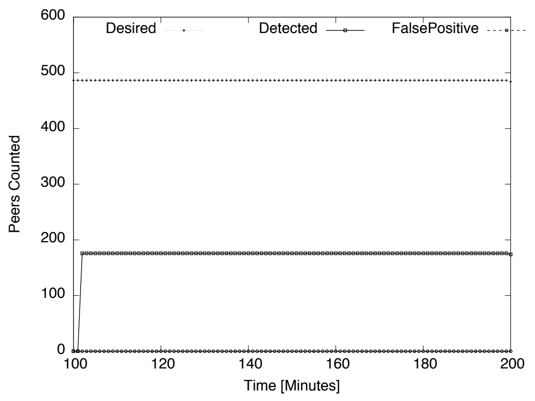
This attack has also no effect on the sinus curve as root gets provided with correct information and only child nodes do not get any information back.

As we can see in Figure 6.11 there are only about 200 nodes as the desired value although we defined 10 percent, i.e. about 500 nodes. This is due to the fact that nodes not having a child node will report themselves as inactive as they cannot use their attack successfully. As before we can assume that about 60 percent of the malicious nodes must have been leaf nodes. This ratio obviously holds regardless of the number of malicious nodes in the network so that we only present the case with a ratio of 10 percent.

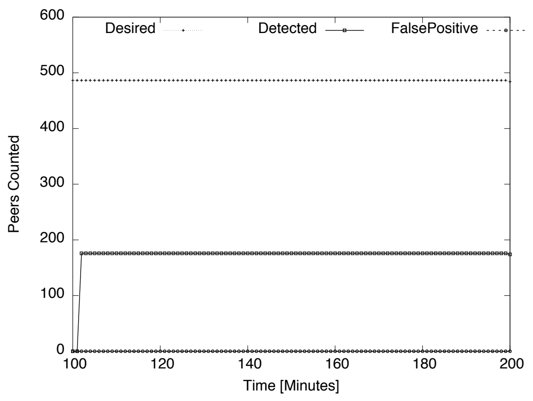
6.3.3 Scenarios with random attack types

In this section we want to refer to the scenarios with all attack types active.

In the first scenario we will also show the attack type distribution to confirm that the different attack types were equally distributed. We will present the results of the intelligent attackers at first and end our result presentation with the simple attackers.



(a) Results of DOMiNo



(b) Results of the timeout rating mechanism

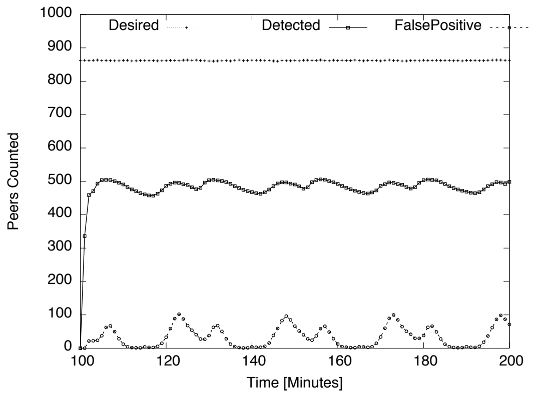
Figure 6.11: Attackers not participating, ratio of 10%

Intelligent Attackers

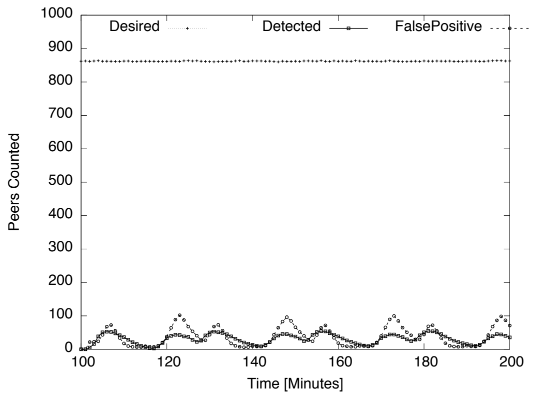
If we take a look at the attack type distribution in Figure 6.12d we can see about 150 of the present nodes were marked as *None* meaning they were not able to launch their attack. If we look more precisely we see that the number of malicious nodes using the attack types *upwards only*, i.e. attackers sending data only to parent nodes and not child nodes, and *wrong data to child*, i.e. attackers sending wrong data values to the child nodes, is only around 50 nodes. Those two attack types are the only ones where this can happen if they are placed as a leaf node in the tree. Correctly, no node is marked as random anymore as they got each a attack type assigned.

As we can see in Figure 6.12b the false negatives were only produced by the Z-Median rating mechanism. Again, we can see a regular pattern like in the sinus curve. Regarding the detection rate around 60 percent, we need to keep in mind that malicious nodes do not report other malicious nodes and that leaf nodes not sending data upwards are not detectable. As we saw before we can expect that about 60 percent of the malicious nodes are leaf nodes.

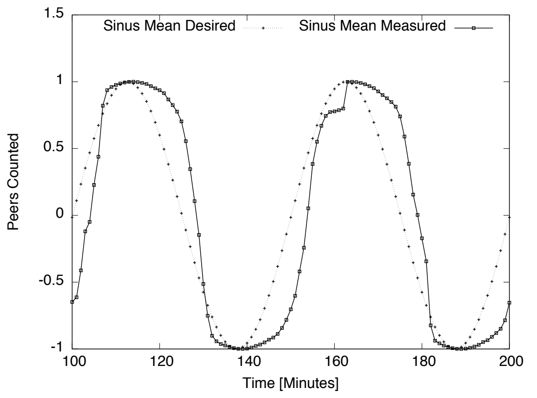
To approximate the real detection rate better we want to continue this thought:
If we look at Figure 6.12d the number of malicious nodes not able to attack is about 145 which are already excluded in Figure 6.12a. But, we can furthermore subtract about 130 nodes which send wrong data to child nodes where we have no mechanism to detect them. Thus, we have about $860 - 130 = 730$ nodes left. If we take a value of about 490 detected nodes we get a detection rate of about 67 percent respectively a false negative rate of about 33 percent. We still need to be aware that the malicious nodes do not report each other and that they represent 20 percent of the whole network. Regarding the false positive rate we will take the maximal number of reported false positives and divide it by the number of non-malicious nodes. I.e. with a value of false positives of 100 at the peak we get about



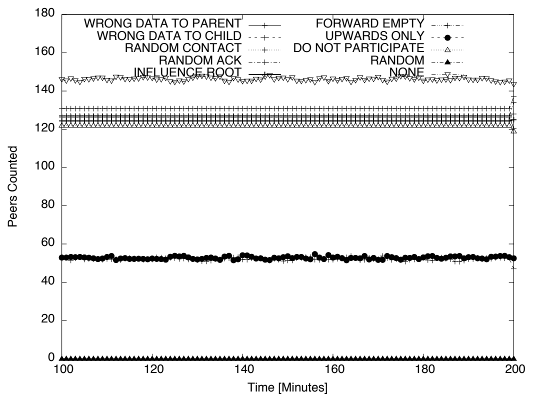
(a) Results of DOMiNo, sizing factor = 2, intelligent attackers, ratio of 20%



(b) Results of the Z-Median rating mechanism, sizing factor = 2, intelligent attackers, ratio of 20%



(c) Sinus values reported by the root node, sizing factor = 2, intelligent attackers, ratio of 20%



(d) Attack type distribution, ratio of 20%

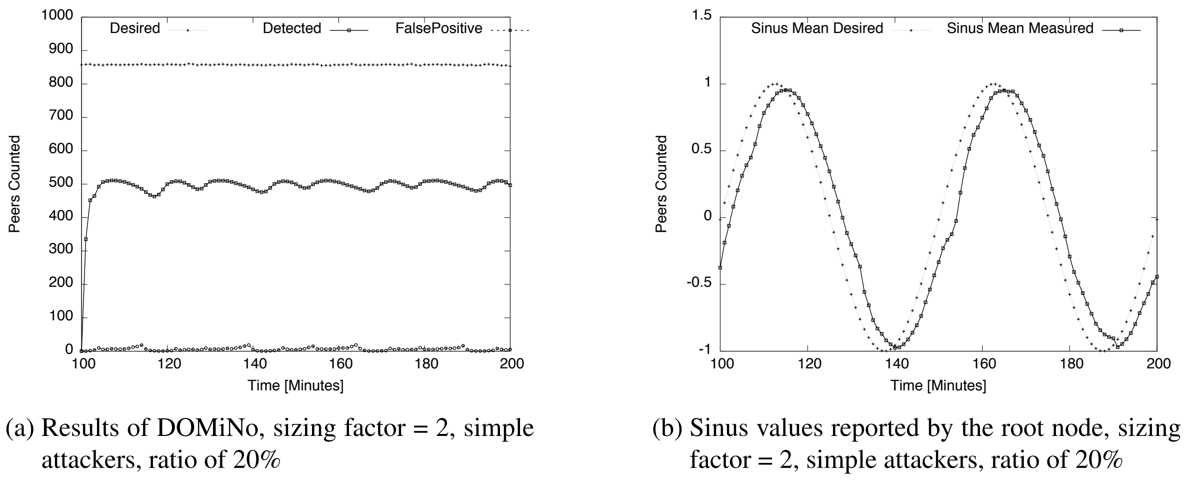
Figure 6.12: Intelligent attackers using different attack types

$\frac{100}{5000-860} \approx 2.42$ percent of reported false positives.

Simple Attackers

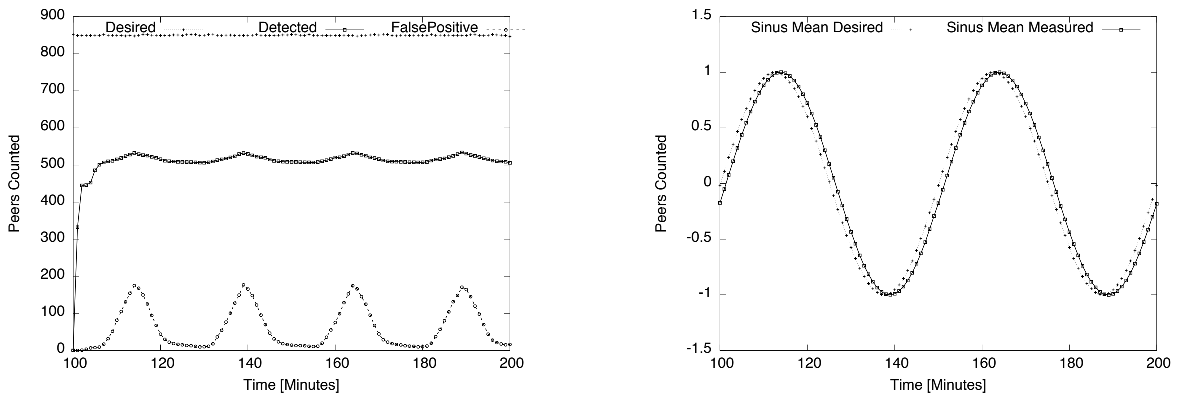
Taking a look at Figure 6.13 we can see similar detection rates as when being confronted with the intelligent attackers. Considering Figure 6.13a we have almost no false positives when attackers use a sizing factor of 2. As stated before we expected this case to be the easier for DOMiNo as the manipulation of data behaves differently when multiplying the values instead of building fractions of it due to different relative values and maximal and minimum values of the used functions which limit the attackers possibilities.

If attackers use a sizing factor of 0.25 depicted in Figure 6.13c the false positive rates are significantly



(a) Results of DOMiNo, sizing factor = 2, simple attackers, ratio of 20%

(b) Sinus values reported by the root node, sizing factor = 2, simple attackers, ratio of 20%



(c) Results of DOMiNo, sizing factor = 0.25, simple attackers, ratio of 20%

(d) Sinus values reported by the root node, sizing factor = 0.25, simple attackers, ratio of 20%

Figure 6.13: Simple attackers using different attack types

higher showing peaks of about approximately 4.8 percent shortly after the turning points of the sinus curve as observed before. Again, we strongly assume the problem to be the sinus curve as a data basis instead of the used mechanism.

For both sizing factor scenarios we get a consequent detection rate of about 60 percent respectively a false negative rate of 40 percent.

6.3.4 General Observations

In this section we want to summarize some general observations made in the different scenarios in short.

The most obvious observation is the behavior of the false positives using the sinus function as the data input. This can be explained via the following example. Around a turning point a data base *db* containing the values sent by child nodes during the last three update intervals may look as follows: $db = \{0.996, 0.9961, 0.998, 0.999, 0.998, 0.997\}$. The values will be relatively close to each other and the median deviation is rather small. After that the relative differences will get bigger as the curve gets steeper again. With that a value possible of 0.98 being sent in the next update interval will look suspicious in contrast to the available data base.

The same may happen in the middle of two turning points around the sinus value of 0 as the ascent respectively drop of the sinus curve is maximal at this point. This explains the pattern behavior with spikes around the mentioned points.

The sinus is, as expected, not really suited for our approach. We expect better results once we are able to use the network's data regarding hop counts and more. In the current implementation we are not able to use this data which is already available at the underlying layers of the simulator framework.

Due to missing topology awareness we cannot detect specific attack scenarios at the moment. I.e. we are not able to detect attacks which do not send data to their parent node as the parent node is not aware of its child nodes. The missing awareness is also the reason that malicious nodes sending from the according Domain range seem less suspicious than nodes sending outside this range as it may be possible that they are honest child nodes. Their behavior needs to be checked via the statistical approaches.

Additionally, as we are missing reference values regarding global data, child nodes cannot detect malicious behavior of parent nodes. Their are able to perform several sanity checks to verify that the values may be correct, but they cannot determine whether the values were manipulated or not if the data set passes the sanity checks.

Overall we sometimes achieved detection rates with almost 100 percent. If we did not we need to consider that the malicious nodes do not rate each other which is the reason we missed the 100 percent.

In the next section we will conclude our work and describe possible future work we see for DOMiNo.

Chapter 7

Conclusion

In this thesis we dealt with malicious behavior in structured P2P monitoring systems and its detection. In particular we considered SkyEye.KOM [GKXS08] and defined different attack scenarios against it. We then introduced different evaluation criteria to identify malicious behavior which we used to build a rating system called DOMiNo using the different criteria to evaluate other nodes.

To test our approach we implemented both the different attack scenarios and DOMiNo in the event-driven P2P simulation framework PeerfactSim.KOM [SGR$^+$11]. We simulated and evaluated different scenarios in a network consisting of 5000 different nodes with different ratios of malicious ones. Our approaches showed good results regarding attacks that do not behave according to the message forward algorithm of SkyEye.KOM. For this we extended messages in SkyEye with nonces to verify answering parent nodes without using any additional lookups and a timeout mechanism so that nodes are aware that they are waiting for incoming messages. As nodes in SkyEye can calculate Domain ranges of arbitrary peer IDs, given a tree level, easily, we take advantage of this to determine if a potential child node can possibly be a legitimate child.

Regarding the manipulation of values we used the implemented sinus function as the simulator's network status data was not accessible during our work. We used the Z-Median score to evaluate and categorize incoming data in nodes using DOMiNo. The results showed the potential of the Z-Median score in some scenarios, e.g. when nodes send manipulated data to their parent, but also showed that the Z-Median score is not really suited to evaluate when using the sinus function. We were confronted with false positive rates up to 7.5 percent in the worst case. The plots showed that the false positive reports are noticeable connected to the sinus function values as spikes often occurred at the sinus curve's turning points. Nevertheless that does not exclude the Z-Median as a mechanism for a different data base. We expect the mechanism to work better on network status data like hops where we expect a normal distribution. At the moment DOMiNo is not able to detect attacking child nodes from inside the own Domain range, as nodes are not aware of the topology in SkyEye.KOM.

Additionally, malicious parent nodes are not detectable due to the missing data base to compare them against.

7.1 Future Work

The introduced statistical rating mechanism shows poor performance on the available sinus function. We strongly suggest to test its behavior on real network data like hop counts or TTL values where we expect a normal distribution. We are sure that the results will improve significantly. When tested on real data, bundling each data point to a multidimensional data point should be tested, too. Treating individual data points as multidimensional data may reveal new relations, which can be taken into consideration, too. E.g. a peer stating that is has high computation power may also have high available bandwidth. If we identify present relations, malicious peers may be identified by the likeliness their provided data matches those.

Techniques to rate a parent node's data need to be improved. At the moment we are only able to check if a parent node's data passes basic sanity checks, e.g. the maximum value needs to be at least as high as the minimal value. As we have only the parent node as a data source for the global data we can only rely on such checks. Therefore we need to develop more criteria to test the provided data values against when used on real data. The possible introduction of new communication paths in SkyEye.KOM should be examined more in detail. The benefits from having additional information resources are obvious. If one would be able to gain additional information without being vulnerable to new attacks the present data base would be improved noticeably.

As we only covered the detection of malicious nodes, different possible reactions to the detection results need to be examined. Upon detecting missing ACK messages a node may talk to another parent node or its grand parent via special messages. Nevertheless, different reactions may provoke new attack types. Based on the reactions, DOMiNo may be improved to act more dynamically, adjusting to different scenarios with more or less restrict rating behavior.

For our simulations we used a branching factor of 8. One needs to determine DOMiNo's performance when using lower and higher branching factors. While the statistical approaches may not be feasible with lower branching factors, they may provide better results when using higher ones.

Some of our provided ideas may be reused for monitoring systems which use different topologies. E.g. if one has a monitoring system based on a mesh topology, statistical approaches should be suited for it as well.

Bibliography

[Agg13] AGGARWAL, Charu C.: *Outlier Analysis*. Springer, 2013. http://dx.doi.org/10.1007/978-1-4614-6396-2. http://dx.doi.org/10.1007/978-1-4614-6396-2. ISBN 978–1–4614–6395–5

[Ala15] ALAVI, Payman: *Security Mechanism for Monitoring Peer-to-Peer Networks*. Germany, University Paderborn, Master Thesis, 2015

[BKNS00] BREUNIG, Markus M.; KRIEGEL, Hans-Peter; NG, Raymond T.; SANDER, Jörg: LOF: Identifying Density-based Local Outliers. In: *SIGMOD Rec.* 29 (2000), Mai, Nr. 2, 93–104. http://dx.doi.org/10.1145/335191.335388. DOI 10.1145/335191.335388. ISSN 0163–5808.

[CH67] COVER, Thomas M.; HART, Peter E.: Nearest neighbor pattern classification. In: *Information Theory, IEEE Transactions on* 13 (1967), Nr. 1, S. 21–27.

[DZD+03] DABEK, Frank; ZHAO, Ben Y.; DRUSCHEL, Peter; KUBIATOWICZ, John; STOICA, Ion: Towards a Common API for Structured Peer-to-Peer Overlays, Springer, 2003. ISBN 3–540–40724–3, 33-44.

[Gie14] GIESEN, Philipp: *Systematic Benchmarking of Monitoring Protocols in Distributed Systems*. Germany, Heinrich Heine University Düsseldorf, Master Thesis, 2014

[GKXS08] GRAFFI, Kalman; KOVACEVIC, Aleksandra; XIAO, Song; STEINMETZ, Ralf: SkyEye.KOM: An Information Management Over-Overlay for Getting the Oracle View on Structured P2P Systems. In: *IEEE ICPADS '08: Proc. of the Int. Conf. on Parallel and Distributed Systems*, IEEE, 2008.

[Gra10] GRAFFI, Kalman: *Monitoring and Management of Peer-to-Peer Systems*, Technische Universität Darmstadt, Diss., August 2010. http://tuprints.ulb.tu-darmstadt.de/2248/

[Haw80] HAWKINS, D.M.: *Identification of Outliers*. Chapman and Hall, 1980 (Monographs on applied probability and statistics). https://books.google.de/books?id=fb0OAAAAQAAJ. ISBN 9780412219009

[IBM] IBM: *IBM Knowledge Center*. http://www-01.ibm.com/support/knowledgecenter/?lang=de#!/SSWLVY_1.0.0/com.ibm.spss.analyticcatalyst.help/analytic_catalyst/modified_z.html, . Accessed: 2015-12-20

[IH93] IGLEWICZ, B.; HOAGLIN, D.C.: *How to Detect and Handle Outliers*. ASQC Quality Press, 1993 (ASQC basic references in quality control). ISBN 9780873892476

[JWS03] JIN, Cheng; WANG, Haining; SHIN, Kang G.: Hop-count Filtering: An Effective Defense Against Spoofed DDoS Traffic. In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. New York, NY, USA : ACM, 2003 (CCS '03). ISBN 1–58113–738–9, 30–41.

[NF11] NICOLAS FALLIERE, Liam O.; SYMANTEC (Hrsg.): *W32.Stuxnet Dossier*. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, 2011.

[Old] OLDENBURG, University of: *COHDA*. https://www.uni-oldenburg.de/en/computingscience/ui/research/topics/cohda, . Accessed: 2015-12-20

[RD01] ROWSTRON, Antony I. T.; DRUSCHEL, Peter: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: *IFIP/ACM Middleware '01: Proc. of the Int. Conf. on Distributed Systems Platforms* Bd. 2218, Springer, 2001 (LNCS).

[RRS00] RAMASWAMY, Sridhar; RASTOGI, Rajeev; SHIM, Kyuseok: Efficient Algorithms for Mining Outliers from Large Data Sets. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, 2000 (SIGMOD '00). ISBN 1–58113–217–4, 427–438.

[SGR$^+$11] STINGL, Dominik; GROSS, Christian; RÜCKERT, Julius; NOBACH, Leonhard; KOVACE-

VIC, Aleksandra; STEINMETZ, Ralf: PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems. In: *Proc of HPCS '11*, 2011.

[SMK+01] STOICA, Ion; MORRIS, Robert; KARGER, David; KAASHOEK, M. F.; BALAKRISHNAN, Hari: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: *SIGCOMM '01: Proc. of the Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ACM, 2001. ISBN 1–58113–411–8.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 21.December 2015 Sebastian Brink

Please add here

the CD holding sheet

This CD contains:

- A *pdf* Version of this master thesis
- All LaTeXand graphic files that have been used, as well as the corresponding scripts
- The source code of the software that was used during the master thesis
- The data that was measured during the evaluation
- The referenced sources