paluno - The Ruhr Institute for Software Technology
Institut für Informatik und Wirtschaftsinformatik
Universität Duisburg-Essen

Software Systems Engineering
Prof. Dr. Klaus Pohl

Bachelor's Thesis

Consideration of Computation Offloading Strategies in a Simulation-Based Analysis of Threats to Location Privacy in Fog Computing

Vorgelegt der Fakultät Wirtschaftswissenschaften
der Universität Duisburg-Essen (Campus Essen),
von:

Markus Maximilian Schlotbohm
Schnorrenbergstraße 42
47877 Willich
markus.schlotbohm@stud.uni-due.de
Matr.Nr. 3030838

Willich, den 1. August 2021

Betreuer:	Dr. Zoltán Ádám Mann, Dr. Andreas Metzger
Erstgutachter:	Prof. Dr. Klaus Pohl
Zweitgutachter:	Prof. Dr.-Ing. Amr Rizk
Studiengang:	Wirtschaftsinformatik (B.Sc.)
Semester:	Sommersemester 2021

Abstract

Fog computing is the extension of cloud computing giving (mobile) end-devices access to low-latency computing and storage capacities provided by nearby fog nodes. To take advantage of these capacities, end-devices offload computation and storage tasks to so called fog nodes. This is known as computation offloading in which various offloading strategies attempt to make optimal decisions regarding whether to offload, where to offload to, when to offload and how much to offload. Over the years, a lot of research has been done in the area of computation offloading. Fog computing, however, comes with other challenges besides finding the optimal offloading strategy.

Another problem of fog computing is the protection of personal data and privacy. This has once again come to the forefront since the introduction of the EU's General Data Protection Regulation. In particular, protecting the location data of mobile end-devices poses a challenge in fog computing. An attacker who has control over a few fog nodes can infer the location of a mobile end-device as well as its trajectory, especially if the attacker knows the location of all fog nodes. These types of attacks are also known as location privacy attacks.

With the help of the *LocPrivFogSim* simulator, attacks on the location privacy of a mobile end-device in a fog computing environment can be simulated and evaluated. The *LocPrivFogSim* simulator was developed in an earlier bachelor thesis at the University of Duisburg-Essen. However, currently the simulator only chooses the nearest fog node for a given mobile device to connect to. In reality, this might not always be the case, as other fog nodes might be better suited for computation and/or storage as the nearest fog node. Changing the simulators behavior however might have implications on the results on location privacy.

The goal of this bachelor thesis is to investigate the impact of computation offloading strategies on the accuracy of location privacy attacks on mobile end-devices. For this purpose, the simulator is extended to allow the selection of a fog node from a set of fog nodes based on an offloading strategy instead of the proximity of the mobile end-device to the fog node. Next, two different offloading strategies which are based on the estimated response time (i.e., the total time of offloading a task from a local device to a remote device and waiting for the response) of an offloading task are implemented. Both implemented strategies reduce the set of available fog nodes to a set of fog nodes whose response time is below a certain threshold. Then, one offloading strategy chooses the target fog node randomly, while the other strategy chooses the fog node with the lowest response time. Next, the experiments from the paper by *T. Wettig* and *Z.Á. Mann* are extended and repeated to cover the newly implemented strategies. Finally, the results are compared with the findings of the paper to evaluate the impact.

Zusammenfassung

Fog Computing ist die Erweiterung des Cloud Computing, mit der (mobile) Endgeräte auf Rechen- und Speicherkapazitäten mit geringer Latenz zugreifen können, die von nahe gelegenen Fog-Knoten bereitgestellt werden. Um diese Kapazitäten nutzen zu können lagern Endgeräte Rechen- und Speicheraufgaben auf sogenannte Fog-Knoten aus. Dies ist auch als Computation Offloading bekannt, bei welchen verschiedene Offloading Strategien versuchen optimale Entscheidungen zu treffen in Bezug auf ob ausgelagert werden soll, wohin ausgelagert werden soll, wann ausgelagert werden soll, und wieviel ausgelagert werden soll. Über die Jahre wurde in dem Bereich des Computation Offloading viel geforscht. Fog Computing kommt aber mit noch weiteren Herausforderungen neben dem Finden einer optimalen Auslagerung.

Ein weiteres Problem des Fog Computing liegt in Schutz personenbezogener Daten und der Privatsphäre. Dies ist seit der Einführung der DSGV der EU nochmal besonders in den Vordergrund gerutscht. Insbesondere stellt im Fog Computing der Schutz der Standortdaten eines mobilen Endgerätes eine besondere Herausforderung dar. Ein Angreifer, der Kontrolle über wenige Fog-Knoten besitzt, kann leicht den Standort eines mobilen Endgerätes in Erfahrung bringen sowie dessen Trajektorie ableiten, insbesondere wenn der Angreifer den Standort aller Fog-Knoten kennt. Diese Art der Angriffe sind auch bekannt als Location-Privacy Angriffe.

Mit Hilfe des *LocPrivFogSim* Simulators können Angriffe auf Standortinformationen eines mobilen Endgerätes im Fog Computing simuliert und evaluiert werden. Der *LocPrivFogSim* Simulator ist in einer bereits früheren Bachelorarbeit an der Universität Duisburg-Essen erarbeitet wurden. Allerdings wählt der Simulator derzeitig nur den nächstgelegenen Fog-Knoten aus, mit dem sich ein mobiles Endgerät verbindet. In der Realität ist dies jedoch nicht immer der Fall, da andere Fog-Knoten für Berechnungen und/oder Speicherung besser geeignet sein könnten als der nächstliegende Fog-Knoten. Eine Änderung des Verhaltens des Simulators könnte jedoch Auswirkungen auf die Ergebnisse zur Location Privacy haben.

Das Ziel diese Bachelorarbeit ist es die Auswirkungen von Computation Offloading Strategien auf die Genauigkeit von Angriffen auf die Standortinformationen eines mobilen Endgerätes zu untersuchen. Dazu wird der Simulator erweitert, um das Auswählen eines Fog-Knoten aus einer Menge von Fog-Knoten auf Basis der Antwortzeit zu ermöglichen statt der Nähe des mobilen Endgerätes zum Fog-Knoten. Als nächstes werden zwei verschiedene Computation Offloading-Strategien implementiert, die auf der geschätzten Antwortzeit (d. h. der Gesamtzeit für das Auslagern einer Aufgabe von einem lokalen Gerät zu einem entfernten Gerät und das Warten auf die Antwort) einer Auslagerungsaufgabe basieren. Beide implementierten Strategien reduzieren die Menge der verfügbaren Fog-Knoten auf eine Menge von Fog-Knoten, deren Antwortzeit unter einem bestimmten Schwellenwert liegt. Dann wählt eine Strategie den Ziel-Knoten zufällig aus, während die andere Strategie den Fog-Knoten mit der niedrigsten Antwortzeit auswählt. Anschließend werden die Experimente aus der Arbeit von *T. Wettig* und *Z.Á.* Mann erweitert und wiederholt, um die neu implementierten Strategien abzudecken. Zum Ende werden die Ergebnisse mit den Erkenntnissen aus dem Papier verglichen, um die Auswirkungen zu bewerten.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AP	Access Point
AWS	Amazon Web Services
FC	Fog computing
GDPR	General Data Protection Regulation
IMEI	International Mobile Equipment Identity
IoT	Internet of Things
ISP	Internet service provider
MAC	Media access control
MB	Megabytes
MB/s	Megabytes transferred per second of time
MCC	Mobile cloud computing
MEC	Mobile edge computing
MI	Million instructions
MIPS	Million instructions per second of time
NIST	National Institute of Standards and Technology
OFC	OpenFog Consortium
QoS	Quality of Service
VM	Virtual machine

1 Introduction

1.1 Motivation

Cloud computing is the on-demand delivery of IT resources over the Internet which gained a lot of attention with the launch of Amazon Web Services (AWS) in 2006.[1] It delivers major benefits (e.g., agility, elasticity, cost savings, disaster recovery, reduction of electronic waste and energy consumption, as well as global deployment within minutes) compared to other server concepts [1, 2]. However, a rapid increase in Internet of Things (IoT) devices – Cisco expects nearly 30 billion devices and connections by 2023 with 45% of those being mobile[2] – results in a huge growth in the amount of data that needs to be transported from and to the cloud, as well as data that needs to be processed and/or stored [3]. Although the cloud is still unbeatable in offering on-demand scalable computation and storage capacities [4, 5] and data processing speeds have increased in the past years, the current cloud model is not designed for that volume, variety, and velocity of data [3, 6]. Furthermore, the geo distributed nature of IoT devices, and the limited bandwidth might result in network bottlenecks, high delays, and low quality-of-service (QoS) [3, 4, 7].

Fog computing aims at extending the cloud to the edge of the network [8] using devices such as base stations, routers, gateways with different computational, storage, network, and power capabilities. These devices – also known as fog nodes – are already present in the network owned and operated by various service providers (e.g., Internet service providers (ISP), cloud operators) or are purposely built and deployed [3, 7]. By dynamically pushing software code to those fog nodes, i.e., moving the processing and storing of the generated data closer to the end-devices, several advantages compared to the traditional cloud computing paradigm arise: For one, already present resources are utilized. Further, compute-intensive tasks can be executed on devices with more computational capabilities compared to end-devices. The closeness to the end-devices results in lower network latencies and reduced network congestions on the fibre backbones of the internet [7, 8]. This results in highly scalable networks with low latency access to computing capacity provided by nearby fog nodes [9]. However, the fog computing paradigm poses also new challenges [10].

One of these new challenges lies in offloading modules, tasks, and code on present devices in the network. “[…] fog computing should ensure that computation over the collected data happens wherever it is best placed, based on various application (eg, hardware, software, and quality of service (QoS)) or stakeholder (eg, cost and business-related) requirements.” [3] Computation offloading tries to schedule code on nearby, more powerful devices to overcome resource limitations of for example less powerful mobile end-devices. The code targeted for the computation

[1] Amazon Web Services, Inc., "What is cloud computing," Amazon Web Services, Inc., 2021. [Online]. Available: https://aws.amazon.com/what-is-cloud-computing.

[2] Cisco, "Cisco Annual Internet Report (2018-2023)," 03 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

offloading task can be either already present on the targeted device or must be transferred to it. Transferring the code however entails additional costs such as initially increased response times (response time is the total time spent for offloading a task from a local device to a remote device and waiting for the answer). Computation offloading can be used in mobile cloud computing (MCC), as well as in mobile edge computing (MEC) and fog computing (FC) architectures. In general, different offloading strategies try to make decisions on whether to offload, where to offload to, when to offload and how much to offload [11]. The objective(s) an offloading algorithm tries to optimize for can differ from offloading strategy to offloading strategy and can range from time optimizations (e.g., response time), to other objectives like cost reduction or QoS improvements [12].

Another challenge lies in security and privacy of the user data. With computing architectures such as MCC, MEC, or in the context of this work fog computing, the user data is not gathered, processed and/or stored at one trusted central location (i.e., the cloud data centre), but instead is transmitted to different mostly nearby devices in the network for processing and storage [13, 14, 15]. In addition, fog nodes are operated by various potentially unknown entities such as Internet service providers (ISP), cloud operators, or many others, potentially including malicious entities [7, 10]. At any point in time, the configuration or properties of the fog network can change. End-devices can dynamically connect to a nearby fog node. This results in a highly dynamic environment with end-devices connecting to potentially untrusted or malicious fog operators [15, 16]. Moreover, as fog nodes are location aware, a fog node operator or even an adversary controlling a fog node can under certain circumstances infer information about the user's location. This location awareness of fog nodes leads to location privacy issues, which is in fact one of the most critical challenges [10].

Every time a mobile end-device that is carried along by a user (e.g., smartphone, smartwatch, or fitness tracker) connects to a nearby fog node, the provider of the fog node or even an adversary can derive information about the user's location, as the device must be nearby. Hence, the provider gains information about the user's location [10, 16]. As the paper by *T. Wettig* and *Z.Á. Mann* [16] shows, an attacker controlling a modest ratio of fog nodes can infer precise information about the location and even the trajectory of an end-device, and therefore the user. In that paper, a simulator called *LocPrivFogSim*[3] was used to analyse threats to the user's location privacy. This simulator was developed to simulate location privacy attacks in a fog computing environment in a previous bachelor's thesis at the University of Duisburg-Essen [17]. In the conducted simulations it was assumed, that mobile end-devices always connected to the closest fog node. However, in a real-world scenario this is probably not always the case. Based on the used offloading strategy mobile end-devices might select a fog node that is farther away but provides for example higher computational resources or faster response times.

[3] The source code of the *LocPrivFogSim* simulator is publicly available at https://git.uni-due.de/snthwett/locprivfogsim.

1.2 Research Question and Research Objective

The main objective of this bachelor's thesis is to investigate the impact of computation offloading strategies on the accuracy of location privacy attacks using *LocPrivFogSim*. Therefore, a total of three sub-goals are pursued in this bachelor's thesis. The first sub-goal is the extension of *LocPrivFogSim* to enable selecting a fog node based on its response time rather than its closeness to the mobile end-device. The second sub-goal is that two offloading strategies should be implemented to select a fog node from a set of possible nodes i.e., nodes that have a response time below a certain threshold. For one, a random node should be selected from the set, secondly the fog node with the lowest response time should be selected. The third sub-goal is that the experiments conducted in the paper by *T. Wettig* and *Z.Á. Mann* [16] should be extended and repeated to cover the newly implemented offloading strategies. The results should be compared to the results of the paper, especially regarding the impact of computation offloading strategies on the accuracy of location privacy attacks.

1.3 Research Methodology

In the present bachelor's thesis, the *LocPrivFogSim* simulator [16] is modularly extended to include computation offloading strategies for choosing fog nodes beyond the nearest fog node. Therefore, it is considered that an end-device would like to offload a known computation task with a known input, output data size, and the number of instructions needed to complete the task. Also, it is assumed that the available computing capabilities of fog nodes and the network bandwidth between the fog nodes and the end-devices are known. The end-device shall select a fog node that can process the task with a response time below a certain threshold. The response time is calculated based on the computation time needed for the tasks and the network transmission time between the fog node and the end-device (the formula is based on [18]). Now, selecting the fog node used as target to offload to is dependent on the implemented and used offloading strategy.

To achieve the presented goals, the simulators data model and functionality is extended. For one, the unit of work a mobile device wants to offload i.e., an offloading task, is introduced. This offloading task data model contains information about the input as well as output data size (in megabyte (MB)) and the number of instructions (in million instructions (MI)) needed to complete the task. Further, the functionality of a mobile end-device is expanded to include the capability of scheduling offloading tasks and choosing a target fog node using an offloading strategy to offload the task to. Lastly, the observer pattern used in *LocPrivFogSim* is adjusted to notify the attacker not only when a mobile end-device connects or disconnects to or from a fog node or access point due to a migration or handoff event, but also when a fog node receives an offloading task to execute. This allows the attacker to filter for specific events and create different scenarios to simulated and evaluated attacks on location privacy of mobile end-device in a fog computing architecture. It should be noted that to reduce the complexity of the implementation in this bachelor's thesis an offloading task which started executing on a selected fog node will also complete its execution on that fog node, although the mobile end-device might have moved, and another fog node might be

better suited. In other words, in this bachelor's thesis there no offloading task migration will be performed. Migrating an offloading task, and thus the adversary acquiring additional location information might positively influence the knowledge of the adversary. This scenario could be investigated in a follow-up work.

In this bachelor's thesis, two offloading strategies are used. Both strategies reduce a set of available fog nodes down to a set of fog nodes whose response time is below a certain threshold. The first strategy then selects one random fog node from the result set as offloading target, while the second strategy selects the fog node with the lowest response time. The experiments executed in the paper by *T. Wettig* and *Z.Á. Mann* [16] are adjusted and repeated with these offloading strategies. Finally, the results are compared to the findings of the paper and the impact of the offloading strategies on the accuracy of location privacy attacks are examined. Therefore, the existing metrics of *LocPrivFogSim* i.e., trace comparison value and size of uncertainty region, are used and additionally extended by two new metrics called relative path hit value and area hit duration. This is due to the fact, that the two present metrics are based on the assumption of the adversary always selecting the closest fog node to the mobile end-device. However, in computation offloading the mobile end-device can select a fog node other than the closest fog node. Hence, the two new metrics evaluate to which degree the adversary is mistaking about the gained knowledge.

1.4 Structure of the Bachelor Thesis

The rest of this thesis is structured as follows: Chapter 2 introduces all necessary fundamentals, including the terms fog computing, computation offloading, privacy with the focus on location privacy. Lastly, the simulation environment including a short explanation of the history of the different evolutions of the simulator will be given. Finally, the *LocPrivFogSim* simulator will be explained in more detail.

In chapter 3, the simulator will be modified and extended to cover computation offloading strategies for choosing the next fog nodes for a mobile end-device. Different computation offloading strategies will then be implemented, including picking a random fog node as well as selecting a fog node with the lowest response time. In chapter 4, the experiment design of this bachelor's thesis will be described. In chapter 5, the experiments described in chapter 4 will be executed and the result will be presented. Then, the results will be compared to the previous findings of the paper by *T. Wettig* and *Z.Á. Mann* [16]. Lastly, the findings of this bachelor's thesis will be discussed.

Finally, in chapter 6, a short conclusion with findings within this thesis, extending the simulator, implementing the offloading strategies, and executing the simulations will be given. Possible future research challenges will be presented.

The word fog nodes and fog devices can be used interchangeably. This bachelor's thesis however uses the word fog node(s) while the class in the simulator is called `FogDevice`. Therefore, when fog device is used, it is always referred to the `FogDevice` class from the simulator.

2 Foundations

The following sub-chapters cover the needed fundamentals for this bachelor's thesis. Firstly, the fog computing paradigm with its individual layers will be presented. Then the term computation offloading will be explained. Next, privacy and its relevance in the fog computing paradigm will be explained. Lastly, the used simulation environment will be presented.

2.1 Fog-Computing

Fog computing is a new field, first introduced by Cisco in the year 2012, extending the cloud computing paradigm to the edge of the network. It enables a new set of applications and services [9]. The definition of fog computing was altered many times, still having no final consensus on it. The two most used definitions were stated by the OpenFog Consortium (OFC)[4] [19], and the National Institute of Standards and Technology (NIST) [20]. The NIST states the type of applications that benefits from the extension of the cloud, namely distributed and latency-aware applications and services. The NIST states further those devices found between smart end-devices and the centralized cloud, so-called fog nodes or fog devices[5], can be either of physical or virtual nature [20]. Both definitions present fog computing as a layered architecture with the goal of taking advantage of the distributed hardware and network capabilities of fog nodes [19, 20].

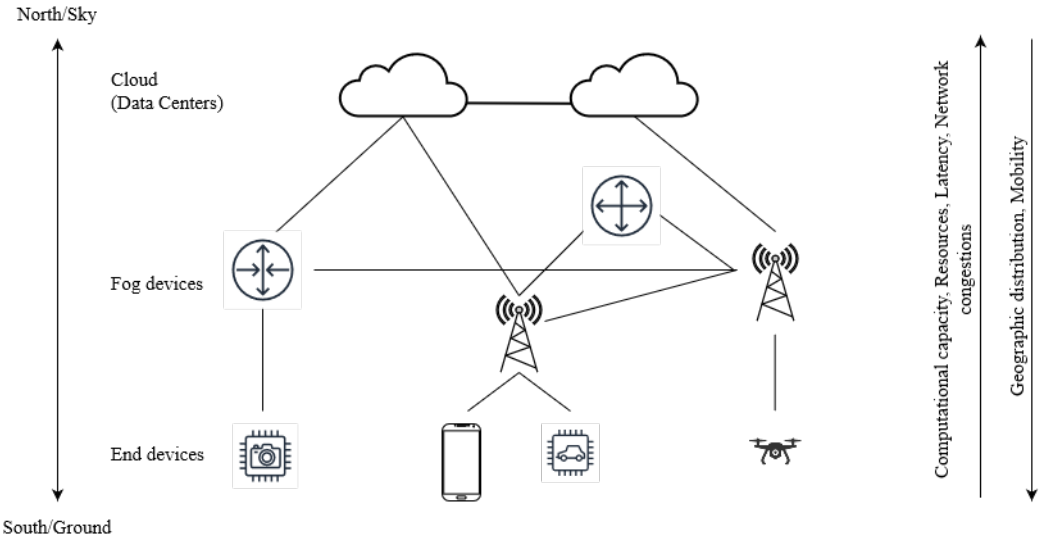


Figure 2-1: Fog computing architecture. Based on [3] and [7].

Figure 2-1 shows the general idea of the fog computing architecture and its layers. As it illustrates, the network is divided in three layers. The cloud can be found in the uppermost layer at

[4] The OFC merged with the Industrial Internet Consortium (IIC) in 2019: "The Industrial Internet Consortium and OpenFog Consortium join forces," Object Management Group, Inc., 2018. [Online]. Available: https://www.iiconsortium.org/press-room/12-18-18.htm.

[5] The terms fog devices and fog nodes are interchangeable. In this work devices in the fog are referred to as fog nodes.

the sky or north of the network. IoT devices or mobile end-devices can be found in the lowest layer. Devices in this layer are referred to as devices in the south or at the ground of the network. Lastly, the layer in the middle is the new layer introduced by the fog computing paradigm. This layer uses compute and storage resources of devices already present in the network or purposely build and deployed devices [3, 7]. Currently, there are ongoing debates on whether based on the layer, fog computing should be further (sub-)classified or not [21, 22]. The next paragraph gives a brief overview of the individual layers, its devices, and problems that arise.

The lowest layer in the network (i.e., at the ground or south of the network) contains the largest number of devices. Those devices are known as end-devices or IoT devices [3]. IoT devices are mostly connected to sensors acting as a data source. Further, those devices can also be connected to actuators which then act as a data sink. Data emitted by devices in this layer usually flows up to the north to be processed and/or stored. Similarly, data can flow down from the north to the south back to the end-devices to be consumed [7]. This layer is characterized by its massive diversity of devices [3]. Cisco further describes devices in this layer by some key characteristics, including highly geographical distribution of those devices, mobility, heterogeneity, and a predominant role of wireless access [9]. Consequently, devices in this layer have various hardware capabilities. Also, mobile devices (e.g., smartwatch or smartphone) are often constrained by their size and battery as the power source, thus limiting the computational capabilities even further. To summarize, due to the limited hardware resources, processing of resource intensive tasks or the processing of parallel tasks is restrained. Hence, more powerful devices are needed to offload high compute-intensive, and parallel tasks [23].

The layer in the north or the sky of the network is referred to as cloud layer. This layer is known for its data centers placed at a few strategic locations in the world. Data centers are directly connected via redundant links to the fiber backbone of the internet. The cloud is characterized by its massive amount of computational, data storage and power capabilities. Some literatures even state it as virtually unbounded [3]. Placing all applications in the cloud is known as the use-the-cloud approach. However, due to the long distance to the end-devices in the lowest layer of the network the latency can be a limiting factor for latency-aware applications or services. Further, transferring all data from the ground to the cloud for processing and/or storage, and from the cloud back to the ground, results in high bandwidth usage and network congestions on the fiber backbones of the internet [7]. Therefore, applications that rely on some key characteristics stated by Cisco are better suited for fog computing [9]. Further, while the number of connected devices is still rising, network congestions are a problem for the use-the-cloud approach. Use-the-cloud approach is therefore considered neither scalable nor suitable for the sheer amount of data [7].

Hence, the introduction of fog computing. This intermediate layer, also known as fog layer, consists of devices that are already present in the network. As already stated, those devices range from base stations, routers to mobile towers and more. By utilizing devices in this layer, code can be deployed closer to the ground of the network and therefore closer to end-devices reducing for example the latency [7]. The capabilities of devices in this layer range between those of devices in the cloud and end-devices at the ground of the network. Devices in this layer are highly

geographically distributed and sometimes even mobile. Further, fog nodes are location aware. The quantity of fog nodes in this layer is higher than in the cloud but lower than the sheer number of devices at the ground of the network. This results in nearly every end-device at the ground having access to a fog node in its proximity. As figure 2-1 shows, fog nodes in this layer form an intertwined network with more devices at the south and less devices at the north. The hardware of those devices is more powerful than the hardware of devices at the ground, but less powerful than the hardware in the cloud. By offloading tasks (mostly latency-sensitive and smaller task for latency-aware applications) into this layer, and only pushing larger, and computationally intensive task, as well as delay-tolerant tasks to the cloud, the network traffic on the backbone of the internet can be reduced and the load more evenly spread. Furthermore, the intermediate layer can be used as a preprocessor or data filter to reduce the load even further [3, 7, 22].

The introduction of fog computing and therefore the utilization of resources between the south of the network and the north results in a reduction of network congestions. However, it should be noted that the fog computing paradigm also brings its challenges. As already stated in chapter 1.1 those challenges range from optimizing computational offloading to security and privacy issues.

2.2 Computation Offloading

Applications executed on the user's end-devices require storage, computing capacity and energy [12]. However, as previously described, end-devices are typically limited in their hardware resources. Additionally, mobile end-devices are also constrained by their battery lifetime (e.g., smartwatches or smartphone). These mentioned characteristics and many others are the reason why computation offloading is considered a promising technique to cope with resource limitations. With computation offloading end-devices can push computation to more powerful devices [11, 12]. Computation offloading can be found in many different environments such as mobile cloud computing (MCC), mobile edge computing (MEC), fog computing [11]. Traditionally, applications forward their resource intensive computation tasks to the cloud for processing and/or storage. This is known as the use-the-cloud approach [7]. However, problems such as network latencies and network congestions arise due to this approach.

This is where fog computing and computation offloading to fog nodes comes into play [12]. In general, different offloading strategies try to make decisions in respect of whether to offload or not, when to offload, how much to offload, and decisions on improving the offloading process itself, as well as decisions regarding security, and privacy [11, 12]. Offloading therefore comes with its own challenges such as optimizing the computation offloading. Mathematical based offloading strategies (also known as model) can be divided into deterministic and non-deterministic models. Due to the dynamic behaviour of mobile end-devices most proposed offloading models are stochastic based [11].

In the survey by *A. Shakarami et al.* [11], offloading is characterised by two different properties. Firstly, there are internal properties, shown in figure 2-2. Internal properties are directly related to offloading. Secondly, external properties are attributes that influence the offloading or are

influenced by offloading, shown in figure 2-3. The following paragraphs describe those internal and external properties.

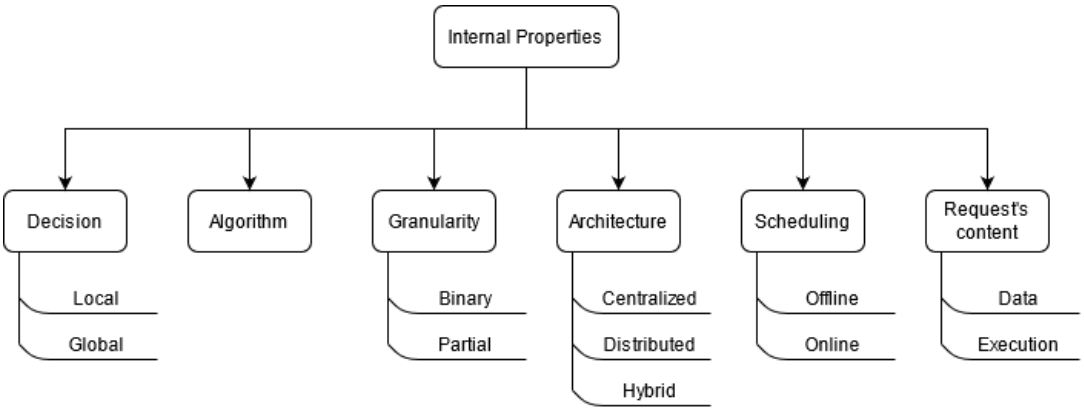


Figure 2-2: Internal properties of offloading based on [11].

A. Decision

The decision-making process i.e., whether, when, and how much to offload, can be local or global. Local decisions are decisions made by the mobile end-device based on the information the mobile end-device possesses. Therefore, the conditions of the mobile end-device and the user preferences can be factored in the decision-making process. Global decisions are decisions made including the state of the total system. Local and global decisions can be either reactive or proactive, or a combination of both also known as hybrid. Reactive decisions are based on the existing situation of the mobile user, the application, and the network, while proactive decisions are according to both the existing situation and the predicted upcoming situation of the mobile user, application, and network.

B. Algorithm

Based on *A. Brogi et al.* [3] and *A. Shakarami et al.* [11], an algorithm set of instructions to find the optimal solution in the offloading process.

C. Granularity

The granularity property describes how much will be offloaded. One possibility is to use a binary approach, meaning that the app code is transferred either entirely or not at all. Alternatively, code can be transmitted partially. This means that only a part of the code is offloaded. This could be beneficial due to different factors e.g., due to costs, security, or privacy.

D. Architecture

Offloading can be based on a centralized, distributed, or hybrid architecture. In a centralized architecture, optimization algorithms, task allocations, resource allocations, and scheduling are performed by a central entity acting as an orchestrator. In a distributed architecture, the decision-making process and scheduling is performed by every individual entity on its own in a distributed way. This could be due to privacy or confidentiality policies of the user, as the user might not want to share their decision-making. Finally, the architecture could be a combination of both, known as hybrid architecture.

E. Scheduling

The scheduling property describes when the offloading decision is made. This could be either offline or online. Offline means that the application or a task to be executed remotely is offloaded before the application is started. On the other hand, online means that the offloading decision is taken while the application is running on the end-device. Offline scheduling is suitable when the request rate of the application is predictable.

F. Request's content

The offloading request's content can be of two types, namely data and execution. When the request's content is of type data, the application offloads data to be saved or processed remotely. When the request's content is of type execution, then an application requests to offload the execution of some code. This offloading request can be in different formats. The execution request can either consist of the code to be executed or it can be in a different format such as a container or virtual machine (VM).

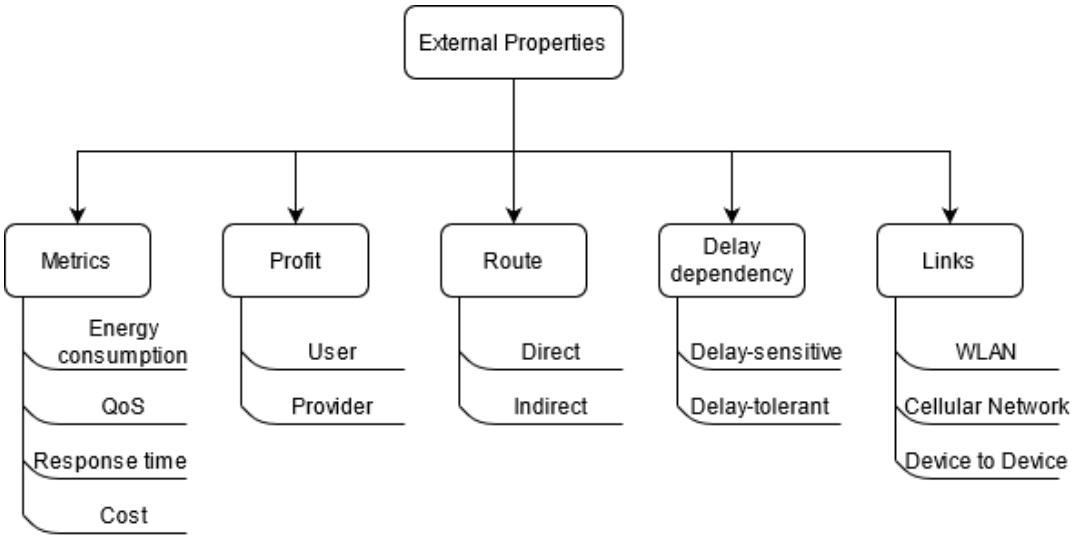


Figure 2-3: External properties of offloading based on [11].

Next, the external properties are described:

A. Metrics

Metrics are a very important property. Different computation offloading strategies aim at optimizing different metrics. Examples of metrics that can be considered include energy consumption, delay, QoS, response time, costs, and more. In the following, two of those mentioned metrics, namely response time and cost, will be shortly described:

- Response time is the total time of offloading a task from a local device to a remote device and waiting for the answer.
- Cost is defined as the cost of local and remote execution of each individual task as well as the communication costs.

B. Profit

Offloading can benefit different parties i.e., the end user or the provider. Profit can be enhanced by optimizing the costs or other mentioned metrics. As an example, the end user would gain a profit through computation offloading due to battery savings for his or her mobile device.

C. Route

The communication route of offloading can be either direct or indirect. With direct routing, requests are offloaded directly to the remote computation server for execution with no intermediary parties in between. However, direct routing has a higher overhead and therefore a lower performance compared to indirect routing. Indirect routing utilizes intermediary parties for offloading. Those maintain a repository of pre-stored code. Hence, indirect routing is faster, but the code that should be offloaded needs to be identified in advance.

D. Delay dependency

The delay dependency property describes how delay-sensitive or delay-tolerant an offloading request is. In a delay-sensitive request, the response time metric is considered as an essential metric. In a delay-tolerant request, other metrics such as energy consumption are more important than response time.

E. Links

The communication link used for offloading can be of numerous different types, including WLAN, cellular network, device-to-device communication and many more. Each type of communication link brings advantages and disadvantages.

2.3 Privacy

As already described, fog computing faces several new security and privacy challenges beside those inherited from cloud computing. Protecting the user's private information, such as data, location, or usage behaviour, is attracting a lot of research in the field of fog computing lately [10, 24]. This is especially true since the introduction of the regulation (EU) 2016/679 of the European parliament and of the council in April 2016. This regulation is also known as General Data Protection Regulation (GDPR) with the purpose of regulating the protection of natural persons with regards to the processing of personal data and the free movement of such data [25]. In article 4 paragraph 1 of the GDPR personal data is defined as following:

"'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;" [25]

In *S. Yi et al.* [10] privacy is divided into data privacy, usage privacy, and location privacy. The goal of data privacy is the protection of the end user's sensitive personal data. Usage privacy is about protecting the end user's usage pattern i.e., usage and communication behaviour. Lastly, location privacy is defined as the protection of the user's position information and the user's trajectory information [10]. Data security and privacy protection are important not only for fog computing but also for other architectures such as cloud or edge computing. It is required to protect the end user's privacy data such as personal data, personal identity, and location against unauthorized access and various attacks [24].

Since privacy is a huge subject this bachelor's thesis is only focused on one sub-area of privacy, namely location privacy. Hence, the following four sub-chapters refer to location privacy areas. Firstly, the term location privacy is defined. Next, the trustworthiness of individual devices in their layer of the network is described and attack vectors are defined. Lastly, attacks on the end user's location privacy are presented.

2.3.1 Location Privacy

As mentioned, fog computing is an extension of the cloud computing paradigm pushing computation and/or storage closer to the end-devices [7]. As mobile end-devices such as smartwatches or smartphones connect to the nearest fog node and fog nodes are location aware, the provider of the fog node or even an adversary can derive information about the user's location [10, 16]. Furthermore, when a provider or adversary owns or controls multiple fog nodes in a region, information's about the mobile end-devices trajectory can be derived. The trajectory consists of current location information and information about the past locations of the user's mobile end-device [26]. Gathering such data violates the user's location privacy [24]. This gathered data could be used to create a motion profile of a mobile end-device, thus uniquely identifying a user. If the location data is further combined with time data, the trace data of the end user can be analyzed and

even be used to precisely predict future movements. Therefore, it is required to protect the location data of a mobile end-device against unauthorized access [26].

2.3.2 Device Trustworthiness

Devices of different layers of the network are associated with different levels of trust. Starting with devices in the cloud layer i.e., cloud servers. Those devices are considered as semi-trusted [14]. They are protected against physical access, as cloud data centers are typically highly protected against unauthorized physical access [15]. Further, active attacks with the intention to get access to a cloud server is considered as very unlikely due to the strong IT safety measures. Although cloud servers will reliably execute their underling tasks, they are also considered as being "[…] curious about unauthorized secret information." [14].

On the other hand, fog nodes are already considered as malicious devices [14]. An attacker can gain physical access to a fog node, as they are mostly unprotected and accessible on public property and are not protected against physical access [15]. Further, an attacker can easily gain control over a fog node. Hence, fog nodes can be used to launch active attacks and/or be used to intercept traffic flowing from and to the fog node [14].

Lastly, devices in the lowest layer of the network i.e., end-devices, are considered as malicious devices as well. This is due to the end user having an interest in gaining access to data without having the proper consent or authorization [14]. Further, end user could try injecting false data into the system, to either disrupt the system or to seek benefits for himself/herself [14, 27].

2.3.3 Attackers and Attack Vectors

In the paper by *C. Huang et al.* [27], attackers are categorized into two kinds, namely external attackers, and insider attackers. While an external attacker has no access to the internal system, an insider attacker has either compromised the system device(s) or has access to the device(s) thus having access to internal parts of the system [27]. This could for example be an employee working in a cloud data center having access to cloud servers. Next, the intentions of an attacker can be classified into selfish attacks and evil attacks. An evil attacker tries to degrade the systems performance or availability, whereas a selfish attacker seeks benefits for himself/herself [27].

The authors of the paper by *C. Huang et al.* [27] further group attacks into active and passive attacks. Usually, a passive attack attempts to disclose the user's privacy information with for example eavesdropping attacks or unauthorized access to different parts of the system. For passive attacks insider attackers are more likely damaging for the system, as internal attackers can more likely bypass existing security control mechanisms. An active attack on the other hand tries to disrupt the system's functionality by modifying, deleting, or disrupting the network or the device itself. Typical active attacks are denial of service (DoS) or distributed denial of service (DDoS) attacks. Active attacks, due to their disruptive nature on the system, are more likely and easier to detect. However, short attacks i.e., attacks of a short duration, are no matter whether an insider or external attack is performing the attack, harder to identify [27].

2.3.4 Attacks on Location Privacy

Attacks on the location privacy of a user can be performed in all different kinds and ways. For one, if an end user uses a location-based service (e.g., google map, apple maps, or other location-based services), the mobile end-device will send geolocation information to this service. An attacker, whether an insider or external attacker, can eavesdrop this communication and therefore gain the location of the mobile end-device [26, 28]. This scenario not only applies to fog computing but rather to all kinds of different architectures [26]. Next, when an attacker controls a single fog node, the attacker can derive information about all the mobile end-devices in the vicinity of that fog node. The attacker must therefore know the location of the controlled fog node and eavesdrop the traffic [26]. Lastly, if an attacker controls multiple fog nodes of an area, the attacker can not only derive the current locations of a mobile end-device, but also can trace the mobile end-device trajectory, due to the connection transition from one fog node to the next fog node. This is known as trajectory attack. These trajectory traces can be used to construct movement profiles. If timestamps are included to the trajectory data, reliable predictions about future movements of a user can be made. This is known as trace analysis attack or inference attack [26, 28].

To uniquely identify a user, all the collected data of one mobile end-device must be assigned in some way to a real person's identity. Every mobile end-device, whether smartphone, smartwatch or other devices connecting to mobile networks, can be uniquely identified by their identification numbers. This number could be for example the phone number of a smartphone, its IMEI number, or its MAC address. The assignment of this unique identification number to a user is usually done by the provider or network operator. An attacker usually does not have this kind of assignment information. However, by eavesdropping the traffic and collecting information about the mobile end-devices, an adversary can easily perform this mapping on its own [16, 17]. To further clarify how an adversary would perform such assignment two examples presented by *T. Wettig* [17] will be given:

1. An adversary eavesdrops the traffic of multiple fog nodes in a city. While analyzing the traffic the adversary finds a device which connects every night at a specific fog node in one place till the next morning. This fog node is located in a resident area. The adversary finds the same device identification number connecting to a fog node located on a company premise during the day. By comparing the employee list of the company which is publicly available with the phone number in an online phone book the adversary can map the device identification number to a person.
2. An adversary wants to track and gather information about a person whose identity is already known to the adversary. The adversary therefore eavesdrops on the traffic at the persons home address and observes the home address at the same time. If the person of interest leaves his/her home area and his/her smartphone disconnects from the fog node the adversary controls, the adversary can gain the unique device identification number of the smartphone which disconnected.

2.4 Simulation Environment

The simulator used in this bachelor's thesis has gone through a lot of modifications and extensions in the past, to prepare it for the location privacy scenario covered in this thesis. Starting with *CloudSim*, an open-source framework for modelling and simulating cloud computing infrastructures and services, the simulator was first modified to cover fog infrastructures and applications and renamed *iFogSim*[6] [7]. From there on, *iFogSim* was extended to cover mobile end-devices with geographic positions, and more. This modification was renamed *MobFogSim*[7] [29]. Lastly, *MobFogSim* was extended by *T. Wettig* [17] at a previously conducted bachelor's thesis at the University Duisburg-Essen in the year 2020 to cover simulations of location privacy in fog computing. In the following sub-chapters, a brief history over the evolution of the simulators *iFogSim* and *MobFogSim* is given, followed by a more in-depth presentation of the *LocPrivFogSim* simulator.

2.4.1 Historical Evolution of iFogSim and MobFogSim

As already stated, *iFogSim* is an open-source simulator built on top of *CloudSim* which is an open-source framework for modelling and simulating cloud computing infrastructures and services. *iFogSim* can be used for modelling fog infrastructures and applications and provide an environment to test different application placement algorithms. In addition, *iFogSim* has different built-in metrics to evaluate the performance of these algorithms, including cost, latencies, timings, energy consumption [7]. However, *iFogSim* lacks some basic features in order to simulate mobile end-devices with individual geographic locations connecting to wireless mobile networks.

Hence, the extension *MobFogSim* was developed and presented by *C. Puliafito et al.* [29] in 2019. *MobFogSim* adds the capability of simulating mobile end-devices with an individual and dedicated geographical location. Further, *MobFogSim* adds the ability to simulate wireless networks through an access point (AP), as well as simulating and displaying mobile end-device connection transitions from one fog node to another fog node [29]. To control these transitions, *MobFogSim* introduced migration policies and strategies. Those policies and strategies make decisions based on the direction and speed of the mobile device. This way different mobile scenarios can be simulated [29].

However, *MobFogSim* lacks the capabilities to simulate threats to location privacy in fog computing environments. There is no concept of an attacker eavesdropping on the traffic of a fog node. Mobile end-devices are not mapped to a user or owner. Further, position data used by the mobile end-device are usually to precisely. This is due, an adversary normally only knowing the fog node a mobile end-device connected or disconnected from, resulting in a possible region rather

[6] The source code of *iFogSim* is publicly available from https://github.com/Cloudslab/iFogSim.

[7] The source code of *MobFogSim* is publicly available from https://github.com/diogomg/MobFogSim.

than the exact position of the mobile end-device. Therefore, the *MobFogSim* was extended to cover these scenarios.

2.4.2 LocPrivFogSim Simulator

The basic architecture of all simulators has not changed over the different stages in the evolution. A fog computing system is modelled and simulated using three layers, each responsible for different tasks [7]: physical components layer, logical components layer, management components layer (see figure 2-4).

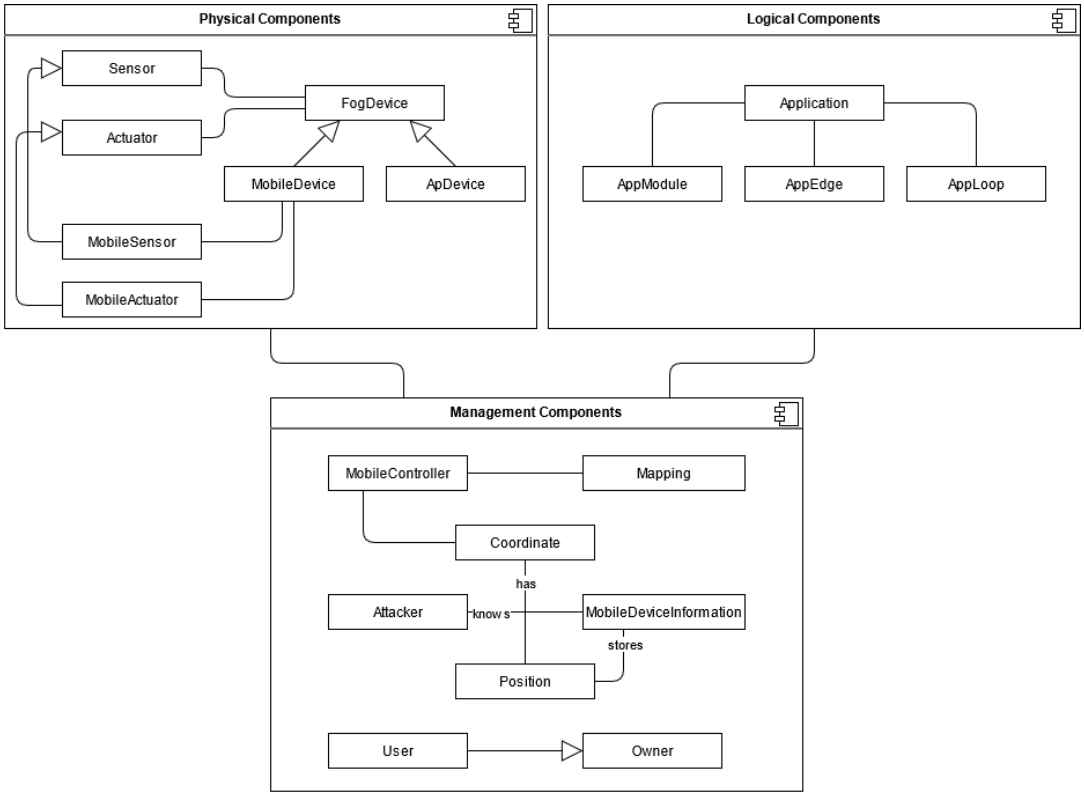


Figure 2-4: Components of *LocPrivFogSim* based on [7], [16], [17], and [29].

The physical component layer (top left of figure 2-4) represents all physical devices that are created in the fog computing infrastructure. *iFogSim* previously has not differentiated between different types of nodes (i.e., cloud data center nodes, fog nodes and end-devices). *MobFogSim* therefore introduced a *MobileDevice*, *ApDevice*, as well as a *MobileSensor* and *MobileActuator*, to model a mobile network fog computing infrastructure.

The logical component layer (top right of figure 2-4) contains the IoT Application. Applications in a fog computing environment often follow a distributed data flow (DDF) design and are modelled as a directed acyclic graph (DAG[8]). Figure 2-5 shows an example of an IoT application modelled as a DAG.

[8] A directed acyclic graph (DAG) is formally described as a pair $\langle V, \varepsilon \rangle$, where V (vertices) represent the application modules and ε (edges) represent the connections between the modules.

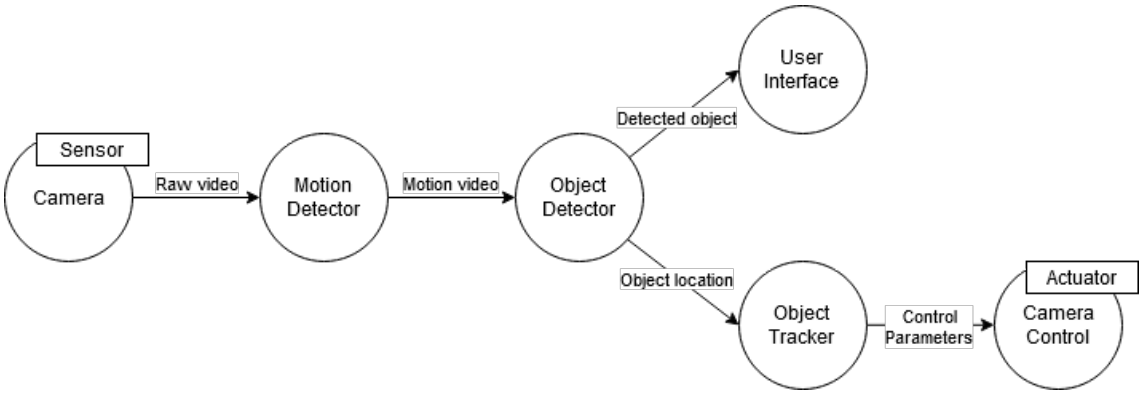


Figure 2-5: Example application based on [7] and DCNSFog implementation of *iFogSim*.

Lastly, the management component layer (bottom center of figure 2-4) represents all components responsible for managing the simulation, or classes that are neither part of the physical components as they are not a physical device, nor part of the logical components as they are not part of the application. Hence, the attacker, user, and owner are located in the management components.

The main classes of the simulator in the three component layers are described in table 2-1. The column superclass contains a reference to another main class. A class which superclass is shown as "-/-" does not mean there is no superclass for this class present, rather its superclass is not of interest for this thesis or does not provide significantly more information on the functionality of the simulator. Next the simulator column shows where the class was first introduced. The labels used in this column are shown in the legend below the table. The last column provides a description of the class.

Table 2-1: Main classes of the simulator based on [7], [16], [17], and [29].

Class	**Superclass**	**Simulator**	**Description**
FogDevice	-/-	iFS	Contains information about the physical device such as RAM, storage, bandwidth, latency, power, and more. Further, a fog device contains references to its associated sensors and actuators.
MobileDevice	*FogDevice*	MFS	As *iFogSim* does not differentiate between devices in the fog network, *MobFogSim* introduced this new device. A *MobileDevice* represents a mobile end-device with a specific position (see *Coordinate*).
ApDevice	*FogDevice*	MFS	An Access Point device used for mobile networks. This device is responsible for establishing a connection between a mobile end-device (see *MobileDevice*) and shutting a connection back down.

Class	Superclass	Simulator	Description
Sensor	-/-	iFS	Generates *Tuple* which represent the data flowing in the network. A sensor therefore forwards the generated *Tuple* to the first *AppModule* of the *Application*.
MobileSensor	*Sensor*	MFS	Gives the mobile end-device (see *MobileDevice*) the ability to have multiple sensors attached without allocating every individual sensor instance, for example a smartphone (represented by the *MobileDevice* class) in reality has a suite of multiple different sensors. Those are combined into this class.
Actuator	-/-	iFS	Receives a *Tuple* from the last *AppModule* in the *Application*.
MobileActuator	*Actuator*	MFS	Represents an *Actuator* of a *MobileDevice*.
Coordinate	-/-	MFS	Represents a position in a cartesian coordinate system.
Position	-/-	LPFS	*MobFogSim* represents the position of a device in a cartesian coordinate system. This is not what an adversary would gain, as an adversary would not receive an exact location of the device, but rather a region in which the device is located. Hence, a new class *Position* was introduced, containing a reference to the fog node, a flag indicating whether the device entered or left the range, and a timestamp.
Application	-/-	iFS	An *Application* in a fog computing environment often follows a DDF design, modelled as a DAG. An *Application* consists of multiple modules (see *AppModule*), which are distributed and executed on the available fog nodes. Connections between the different modules are modelled with an edge (see *AppEdge*).
AppModule	-/-	iFS	Represents a module of an *Application*. Receives *Tuple* to process and afterwards forwards the *Tuple* to the next module. Modules are distributed and executed on different fog nodes.
AppEdge	-/-	iFS	Represents the connection between two modules over which *Tuple* flow.

Class	Superclass	Simulator	Description
AppLoop	-/-	iFS	Required to measure metrics of a specific part of the *Application* during runtime.
Tuple	-/-	iFS	Represents the data generated by a *Sensor*, transmitted over an *AppEdge* from/to an *AppModule*, processed by *AppModule*, and finally consumed by an *Actuator*.
Controller	-/-	iFS	Controls the simulation.
MobileController	-/-	MFS	A new controller introduced by *MobFogSim* to control the simulator with the new mobile capabilities. The old *Controller* introduced by *iFogSim* becomes obsolete.
Attacker	-/-	LPFS	Represents an attacker in a fog computing environment. The *Attacker* registers itself by all compromised fog nodes (see *FogDevice*). This pattern is known as observer pattern. Whenever a mobile device connects or disconnects at a fog node the fog node automatically notifies the *Attacker* by calling its *update* method. The fog node passes the *MobileDevice* in question and the state whether the device connects or disconnects down to that method.
Owner	-/-	LPFS	Represents the *Owner* of a *MobileDevice* i.e., an *User* might carry the smartphone which belongs to a business. But the business is the *Owner* of that device.
User	*Owner*	LPFS	Represents the person carrying the device.
MobileDevice-Information	-/-	LPFS	Represents the information gathered of a unique *MobileDevice* by an *Attacker*. Each *MobileDevice* gets its own instance of this class. When a device connects or disconnects the adversary gets notified (due to the observer pattern). The adversary obtains the unique *MobieDevice* name, whether the *MobileDevice* entered or left the region, and the time of the event. This information is added to the instance of the *MobileDeviceInformation* class for the specific *MobileDevice*.
DeviceMap	-/-	LPFS	Represents the map including all fog devices.

Legend – iFS: *iFogSim*; MFS: *MobFogSim*; LPFS: *LocPrivFogSim*

All simulators use the basic event system introduced in *CloudSim*. Different simulation entities, such as *FogDevice*, *MobileDevice*, *ApDevice*, *MobileController*, *(Mobile-)Sensor, (Mobile-)Actuator*, communicate with each other by sending messages (also called events) between each other or its own instance. An event can additionally contain some data to be sent along. Events can be either sent by using one of the protected *SimEntity*'s *send* method overloads or can be scheduled using the public *SimEntity*'s *schedule* method overloads. Regardless of the used method, all events are registered in a so-called future queue with a time delay specifying the point in time the event should be dispatched. The simulation time of *CloudSim* is measured in ticks, where 1000 ticks represent one second of time. The maximum total simulation time is set to 30 minutes which is equal to 1.800.000 ticks. The difference between the *SimEntity*'s *send* and *schedule* method overloads is that the *send* method overloads will additionally add the network delay (latency) from the source entity to the destination entity if source and destination are not equal to each other.

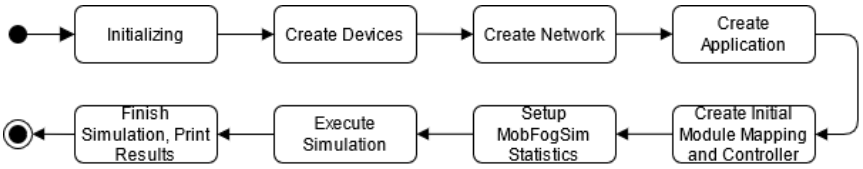


Figure 2-6: Sequence of the Initialization and Execution of the Simulator based on [17].

The simulation sequence of the simulator is represented in figure 2-6. First, all variables controlling scenarios and other factors of the simulation are passed from the string array into the main function. Further, other components such as the logging infrastructure and more are initialized. After initializing the basic and bare bone features the physical devices of the physical components layer are created in the create devices step. All cloud data centers, fog devices, access points, and mobile devices are instantiated. In this step the *LocPrivFogSim* simulator also creates the *Attacker* and passes down a random set of compromised fog nodes. The number of compromised devices is controlled by the *RATE_OF_COMPROMISED_DEVICES* variable. Next, in the create network step, all network connections are created. These connections are persistent and dynamic connections (e.g., an initial connection between an AP and a mobile end-device) between mobile end-devices and access points, mobile end-devices and fog devices, and lastly access points and fog devices. After initializing the network, the application is created. Now, before the simulation starts, an initial module mapping is created as well as the *MobileController* of *MobFogSim* instantiated and the statistics of *MobFogSim* are setup. The *MobileController* is responsible for controlling the simulation by distributing events to the different devices. Now the simulation is started. Now, the *MobileController* will start all *SimEntity*'s and schedules all static events for the duration of the simulation, as shown in figure 2-7. Events generated by sensors are distributed on various devices. These events can run in parallel as the devices are independent from one another. Different devices execute different types of events. The mobile end-devices (*MobileDevice*) for example execute the application for the user. Fog devices (*ServerCloudlets*) execute the *AppModules* and are also responsible for migrating a *MobileDevice* from one device to

the next device without losing data. Lastly, AP's cover different mobile end-devices of one region and are responsible for establishing and disconnecting to *MobileDevices* [7, 16, 17, 29].

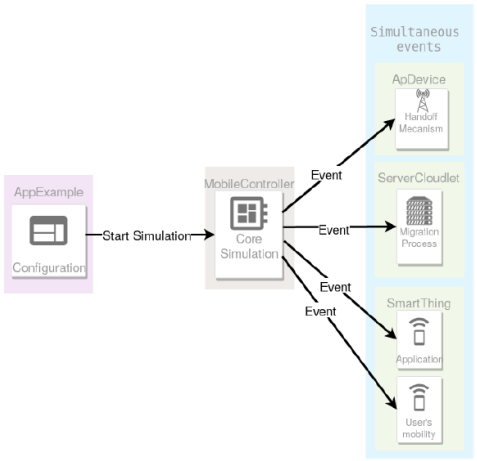


Figure 2-7: Main events generated by the *MobileController*, from [29].

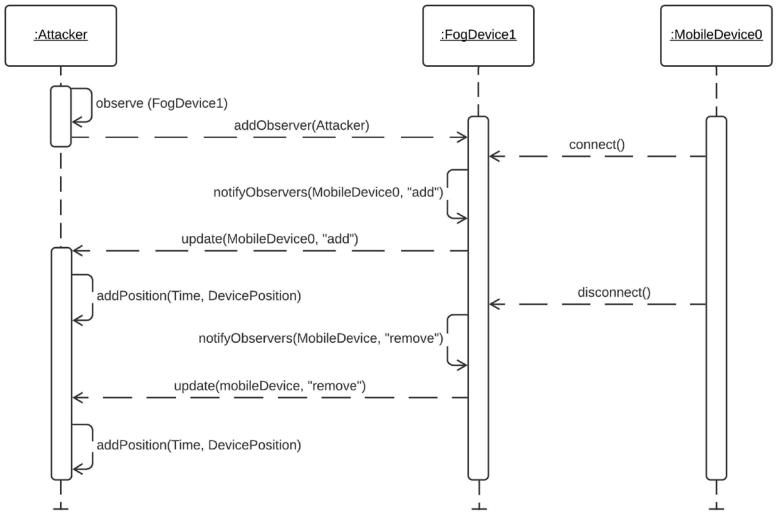


Figure 2-8: Sequence of instantiating and simulating the attacker with the observer pattern, from [17].

Lastly, figure 2-8 (from [17]) shows the process of instantiating the attacker's observer pattern and the flow during simulation. As stated, the attacker receives an instance of every compromised fog device. The attacker then calls the *addObserver* method on the compromised fog device registering itself. This should represent the attacker eavesdropping on the traffic on the fog device. Whenever a fog device connects to the compromised fog device, the fog device notifies the attacker (by invoking the attacker's *update* method) and passes down the mobile device which connected as well as a flag indicating whether the mobile device connected or disconnected. The attacker stores this information together with the position of the fog device and a timestamp. The same happens when a mobile device disconnects from the compromised fog device. This time the fog device notifies the attacker with a flag indicating that the mobile end-device disconnected. This information is at the end of the simulation used to reconstruct the path of the mobile device from all compromised fog devices [17].

3 Extending LocPrivFogSim to Cover Computation Offloading Strategies

This chapter describes the modifications made to extend the *LocPrivFogSim* simulator. First, sub-chapter 3.1 lists the assumptions and limitations made for this work. These assumptions and limitations arise for one from the previous bachelor's thesis by *T. Wettig* [17] and from the paper by *T. Wettig* and *Z.Á. Mann* [16], as well as new assumptions and limitations that are made for computation offloading strategies. Next, sub-chapter 3.2 describes the changes made and how the new computation offloading strategy abilities are implemented in the simulator. In the following sub-chapter (i.e., sub-chapter 3.3) the necessary changes for the simulation environment and the simulators parameter used for this bachelor's thesis are described. Lastly, sub-chapters 3.4 and 3.5 show the implementation of the different offloading strategies i.e., selecting a random fog node (sub-chapter 3.4) and selecting the fog node with the lowest response time (sub-chapter 3.5) are described. It should be mentioned that, both strategies select the node from a set of nodes having the response time below a certain threshold.

3.1 Assumptions and Limitations

In order to extend the *LocPrivFogSim* simulator to cover computation offloading strategies in simulating and evaluating location privacy attacks, assumptions and limitations had been made. Some of these assumptions and limitations are based on the previous bachelor's thesis by *T. Wettig* [17] and from the paper by *T. Wettig* and *Z.Á. Mann* [16]. Additionally, further assumptions and limitations arise through the extension of the *LocPrivFogSim* simulator [17]. Therefore, in this sub-chapter assumptions and limitations that were previously implied and are still valid, that were previously implied but are no longer valid, and that have been newly added are listed below:

- **Still valid** assumptions and limitations implied by [16] and [17]
  - The fog infrastructure is considered separately from the mobile network infrastructure. The fog infrastructure requires reliable network coverage. The locations of the fog nodes and the functionality of the fog system are independent of the mobile network infrastructure.
  - The fog infrastructure is located on a defined region (e.g., a city). This means the adversary can obtain further information through for example a city map, public transport network, traffic situation, and more, and can link this information to his observations.
  - The fog nodes of the fog infrastructure and the mobile network infrastructure cover the entire region.
  - The fog nodes are active and always connected to the fog infrastructure.
  - The adversary knows the locations of all controlled fog nodes.
  - Unless otherwise specified, the adversary does not know the locations of other devices (fog nodes, AP, mobile end-devices).

  - By intercepting traffic on fog nodes controlled by an adversary, the adversary can usually obtain a unique identification number of a connected device but not the name of the owner or user.
  - The adversary cannot normally infer to the identity of the user from the intercepted traffic. The mapping between the unique identification number and the user is done separately (see 2.3.4).
- **No longer valid** assumptions and limitations implied by [16] and [17]
  - Mobile devices are assumed to always connect to the closest fog node.
- **Newly implied** assumptions and limitations for this bachelor's thesis
  - It is assumed that a mobile end-device wants to offload a given computational task.
  - The number of instructions (in MI) to complete the task and the input as well as output data size of the task (in MB) are known.
  - The tasks computation time and the network latency between the fog nodes and the mobile end-device are considered to be known.
  - The calculated estimated response time is assumed to be the actual response time the task needs to complete.
  - A task offloaded to a selected fog node by a given offloading strategy completes its computation on the selected fog node, although the mobile end-device might have moved on and/or the selected fog node might no longer be suitable for the task. In other words, once the task is offloaded it completes on the selected fog node and no task migration will be done. This possibility could be evaluated in an extension of this work.

3.2 Implementing Task Offloading in LocPrivFogSim

This chapter describes the implementation of computation offloading tasks in *LocPrivFogSim*. First, minor changes and clean-ups that help the user getting the simulator up and running as well as help troubleshooting simulations are described. Also, bugs and improvements made while cleaning up the source code are also listed. Then, the process of calculating the response time for an offloading task is shown. This calculation is based on the formula presented in the paper by *J. Zhang et al.* [18] in 2020. Lastly, the design chosen for computation offloading tasks in *LocPrivFogSim* is presented and the implementation described.

3.2.1 Minor Simulator Changes and Clean-ups

The minor source code changes listed in table 3-1 below are divided into general code improvements (improvement) and bug fixes (bug). The file diffs containing the changes can be accessed by visiting https://git.uni-due.de/sjmsschl/locprivfogsim/-/commit/<COMMIT_SHA> and replacing <COMMIT_SHA> with the 8 characters long sha from table 3-1 below:

Table 3-1: Minor source code changes.

Description	Kind	Commit SHA
All hard-coded library path references are replaced with relative paths. The referenced jars are added to the repository.	Improvement	7109bc1c, cc08f22a
All *System.out.println* and *System.out.print* calls are refactored to use the simulators built-in logging system unifying all logs into one place.	Improvement	7749e10f, 106000b0, a1901758
Add the ability to log all events while the simulation is running.	Improvement	673a9b7c
All active observers (i.e., compromised *FogDevice*'s or *ApDevice*'s) only call the attackers *update* method when the simulation is started. This avoids polluting the attackers gained knowledge while the simulator is setup.	Improvement	07eaf562
Add a base id to *MobFogSim*'s event ids so it matches *iFogSim*'s implementation.	Improvement	4d4c139a
During debugging it was discovered that the *MobileDevice*'s path (i.e., the path the user walks during the simulation) set during the initialization of the simulator was never used during the simulation. This is due to an attribute (named travel time) being set to -1 which on the check for the users next step caused the *MobileDevice* being skipped. After this change it was verified that the device is actually moving along its set path by firstly setting a break point on the *getPath* method and secondly adding a log statement in the *nextStep* method responsible for moving the *MobileDevice*.	Bug	0d5dffc1

3.2.2 Calculating an Offloading Tasks Response Time

As described in the foundations (see chapter 2.2 – Computation Offloading), response time is the total time of offloading a task from a mobile end-device to a remote fog node and waiting for the response [11]. In the paper by *J. Zhang et al.* [18] response time is calculated by the following formular:

$$t_{i,j} = \frac{d_{i,j}}{r_{i,j}} + \frac{c_{i,j}}{f_j};$$

where i is a mobile end-device and j is a fog node; $t_{i,j}$ represents the response time; $d_{i,j}$ is the size of the data flowing from the mobile device i to the fog node j (i.e., the input data size); $r_{i,j}$ denotes the total uplink bandwidth from the mobile device i to the fog node j; $c_{i,j}$ is the total number of CPU cycles (in number of million instructions (MI)) required to accomplish the computation task, and f_j denotes the CPU-cycle frequency of the fog node (in million instructions per second (MIPS)), which is fixed for the duration of the computation task execution. This formula was extended to include the transmission time of the response i.e., the output data size divided by the downlink bandwidth from the fog node j to the mobile end-device i:

$$t_{i,j} = \frac{d_{i,j}}{r_{i,j}} + \frac{c_{i,j}}{f_j} + \frac{d_{j,i}}{r_{j,i}};$$

where $d_{j,i}$ is the size of the data flowing from the fog node j to the mobile device i (i.e., the output data size); $r_{j,i}$ denotes the total downlink bandwidth from the fog node j to the mobile device i.

To figure out what data is present in the simulator and what needs to be added, a data inventory was created (table 3-2). This data is correlated with the variable from the formula, the unit it represents, the class it is associated to i.e., where it can be retrieved from by calling a getter method.

In *LocPrivFogSim* the uplink bandwidth $r_{i,j}$ from a mobile end-device, access point, or fog node can be retrieved by calling *getUplinkBandwidth()*. Similarly the downlink bandwidth $r_{j,i}$ can be received by calling *getDownlinkBandwidth()*. As figure 3-1 illustrates, the fog network is a fully connected network, meaning all fog nodes are connected to each other. Every fog node in the network has an access point next to it. A mobile end-device connects to its nearest AP. So, to compute the total bandwidth along a given network path, it is assumed that the bandwidth is limited by the smallest value on the path. Therefore, while calculating the total uplink bandwidth all values on the network path are passed into *Math.min(…)* to calculate the minimum value. If mobile end-device 1 (MD1) is connected to access point 5 (AP5) and calculates the response time to fog node 5 (F5), its network path is limited by the minimum bandwidth between mobile end-device 1 (MD1) and access point 5 (AP5), as well as by access point 5 (AP5) and fog node 5 (F5). However, if mobile end-device 1 (MD1) is connected to access point 5 (AP5) and calculates the response time to a fog node other than fog node 5 (F5), an additional network hop is introduced, and the total bandwidth is further limited by this additional hop.

Table 3-2: Inventory of the existing data and newly added data.

Name	New	Variable	Unit	Class(-es)	Description
Upload Bandwidth	×	$r_{i,j}$	MB/s	*FogDevice, ApDevice, MobileDevice*	The maximum upload bandwidth a device is capable of in the network.[9] The bandwidth of two directly connected (wireless or wired) communicating devices is limited by the minimum value of either the uploading device (upload bandwidth) or the downloading device (download bandwidth).
Download Bandwidth	×	$r_{j,i}$	MB/s	*FogDevice, ApDevice, MobileDevice*	The maximum download bandwidth a device is capable of in the network.[9] The bandwidth of two directly connected (wireless or wired) communicating devices is limited by the minimum value of either the downloading device (download bandwidth) or the uploading device (upload bandwidth).
Available Compute Resources	×	f_j	MIPS	*FogDevice*	The current remaining available compute power of a device.
Input Data Size	√	$d_{i,j}$	MB	*OffloadingTask*	The input data size of the task to offload.
Output Data Size	√	$d_{j,i}$	MB	*OffloadingTask*	The output data size of the finished offloaded task sent back in the response message.
Required Compute Resources	√	$c_{i,j}$	MI	*OffloadingTask*	The required compute resources to execute the offloaded task (in million instructions (MI)).

[9] In the simulator's initialization phase, while creating all devices, every device gets a random uplink bandwidth between 300 MB/s and 1000 MB/s, as well as a random downlink bandwidth between 500 MB/s and 1000 MB/s assigned.

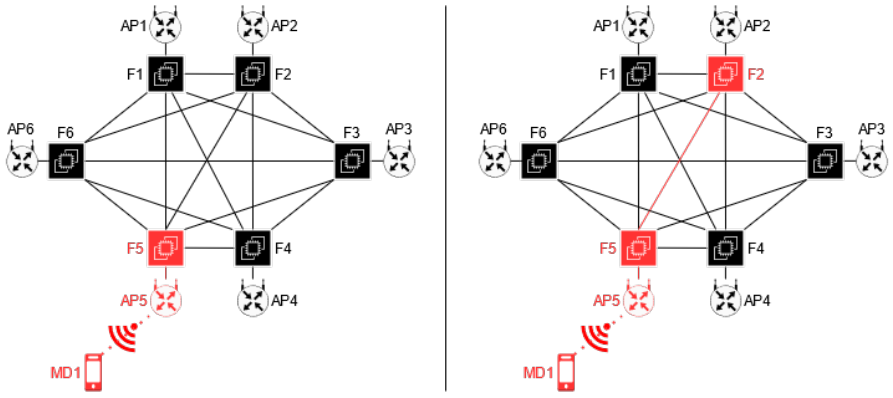


Figure 3-1: *LocPrivFogSim* Simulators Network Topology.

Legend: AP = *AccessPoint*; F = *FogDevice*; MD = *MobileDevice*; red path represents a possible network path

The CPU-cycle frequency of a fog node f_j (in million instructions per second (MIPS)) can be retrieved calling *getAvailableMips()* on the target fog nodes f_j processing element provisioner (called *PeProvisioner*) on the host. [10]

The missing data i.e., the input data size $d_{i,j}$ (MB), the output data size $d_{j,i}$ (MB), and the required compute resources $c_{i,j}$ (MI) are correlated to an offloading task. Therefore, it will be part of the new introduced class *OffloadingTask*. These values are set on a per simulation basis. The values used for this bachelor's thesis is described in chapter 3.3 – Simulator Environment, Setup, and User Code.

3.2.3 Implementing Task Offloading in LocPrivFogSim

The first defined sub-goal of this bachelor's thesis is the extension of *LocPrivFogSim* to enable selecting a fog node based on its response time rather than its closeness to the fog node. It is therefore assumed, that a given mobile end-device wants to offload a known computation task. Hence, the decision-making process on when to offload and what to offload (meaning the tasks values i.e., the input and output data size of the task and the number of instructions to complete the execution) is kept local on the mobile end-device. Therefore, the concept of a scheduler is introduced (see figure 3-2). The scheduler is responsible for scheduling an offloading task, effectively answering the question when to offload.

As figure 3-3[11] shows, when the simulation starts, all entities are started by the *MobileController's startEntity(): void* method as described in chapter 2.4.2 – *LocPrivFogSim*

[10] The implementation of the response time calculation can be found in the *BandwidthCpuResponseTimeCalculator* class in the offloading package.

[11] The *send* and *schedule* methods of the sequence diagrams in figure 3-3 to figure 3-8 are modelled as methods on the *CloudSim* object. However, in the code these methods are part of the *SimEntity* base class of which the class *FogDevice*, and therefore *ApDevice* and *MobileDevice* inherit. The *SimEntity* class however internally redirects all these method calls to the *CloudSim* static *send* method to register events in the simulators future queue for processing. To reduce the complexity of the sequence diagrams it was decided to simplify this call-chain to the *CloudSim* class.

Simulator. In this start method all mobile end-devices, created with an instance of an *IOffloadingScheduler*, are registered with a *MAKE_OFFLOADING_SCHEDULING_DECISION* event in the simulators future queue. This event is registered at time 0, meaning this event will be called first as soon as the simulation starts running. By checking whether the mobile end-device has an instance of the *IOffloadingScheduler* set or not, one can decide whether the mobile end-device is capable of offloading tasks to remote fog nodes or not. Hence, a mobile end-device having no scheduler set will be excluded from the simulators offloading event system, thus reducing the overall number of events flowing in the simulator.

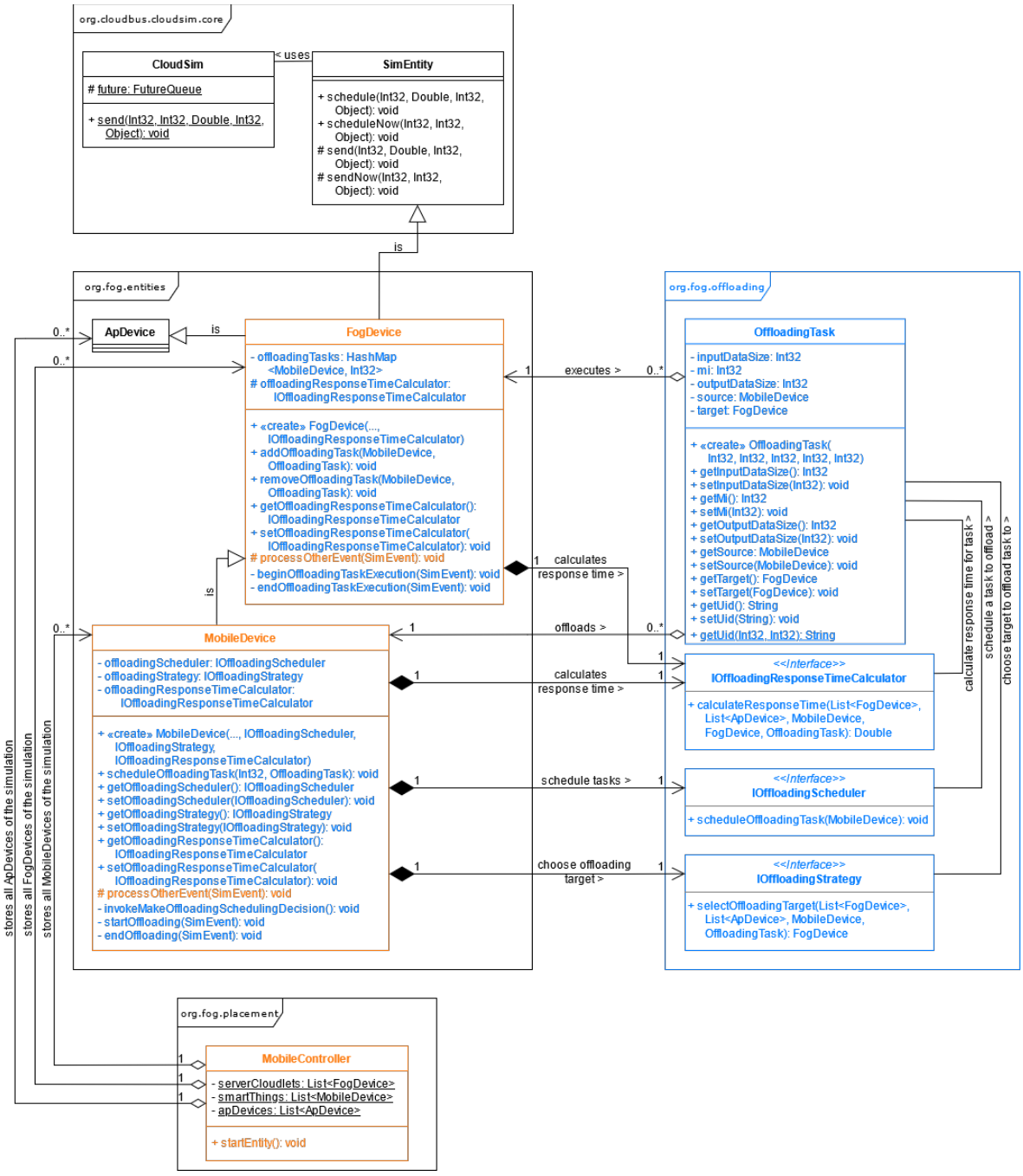


Figure 3-2: Extract of the UML class diagram.

Legend color codes: black = not modified; blue = newly introduced; orange = modified method

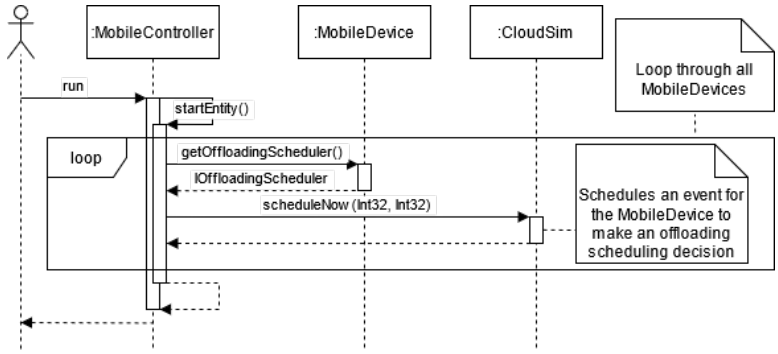


Figure 3-3: Sequence diagram of the initial scheduling of the *MAKE_OFFLOADING_SCHEDULING_DECISION* event for all *MobileDevice*'s that have an instance of *IOffloadingScheduler* set.

If the simulation starts running, the above-mentioned event is fired by the simulator. All *MobileDevice*'s that have this event registered receive the event for processing (see figure 3-4). The receiving *MobileDevice* calls its *invokeMakeOffloadingSchedulingDecision(): void* method. This method first fetches the offloading scheduler associated with the mobile end-device by calling *getOffloadingScheduler(): IOffloadingScheduler* on its own instance. Then it calls the *scheduleOffloadingTask(MobileDevice): void* on the scheduler passing itself down as source into the schedule method. The schedulers implementation now decides whether an offloading task should be scheduled or not, and when an offloading task should be started, meaning scheduling a task at a specific point in time to start running. To register an offloading task the implemented scheduler calls the *scheduleOffloadingTask(Int32, OffloadingTask): void* method on the *MobileDevice* to register an offloading task. The scheduler is allowed to call this method multiple times or even not at all, therefore, scheduling 0 to n number of task(s). The *scheduleOffloadingTask* method expects an instance of an *OffloadingTask*. Hence, the scheduler is responsible for creating the instance of the *OffloadingTask* class containing the tasks information such as the input data size, number of instructions, and the output data size. The offloading task passed down to the mobile end-device will now be registered in the simulators future queue to start the offloading process at the specified point in time. The *MobileDevice* archives this by scheduling a *START_OFFLOADING* event with the *OffloadingTasks* instance as event message data.

If the simulated time (in ticks) to offload a task is reached, the simulator dispatches a *START_OFFLOADING* event containing the *OffloadingTask* as event message data. The mobile end-device receiving this event calls its *startOffloading(SimEvent): void* method. As figure 3-5 shows, the *startOffloading* method then fetches the mobile end-device associated *IOffloadingStrategy* by calling the getter on the mobile end-device itself. On the retrieved offloading strategy the *selectOffloadingTarget(List<FogDevice>, List<ApDevice>, MobileDevice, OffloadingTask): FogDevice* method is invoked. The implementation of the offloading strategy is now responsible for choosing the offloading target i.e., the target fog node. As parameters the offloading strategy gets all available fog nodes and access points of the network, the mobile end-device as source for the offloading request, and the offloading task to select a target node for execution. The offloading

strategy implementations of this bachelor's thesis are described in chapter 3.4 – Offloading Strategy 1: Offloading to a Random Fog Node From a Pre-Filtered Set of Nodes and chapter 3.5 – Offloading Strategy 2: Offloading to a Fog Node With the Lowest Response Time From a Pre-Filtered Set of Nodes. When the offloading strategy returns null as fog node target, the offloading task request by the mobile end-device is cancelled and the offloading process is stopped until the next make offloading decision is invoked on the mobile end-device. Lastly, the mobile end-device schedules an *BEGIN_OFFLOAD_TASK_EXECUTION* event for the fog node returned by the offloading strategy with the offloading task as event message data. In other words, the fog node returned by the offloading strategy is be used as event target. This event will be immediately dispatched by the simulator.

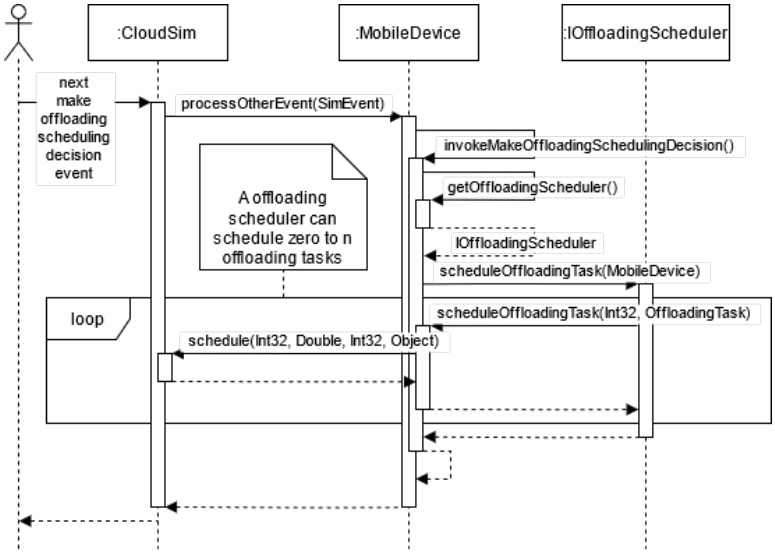


Figure 3-4: Sequence diagram of a *MobileDevice* making an offloading scheduling decision and scheduling one or more offloading task(s).

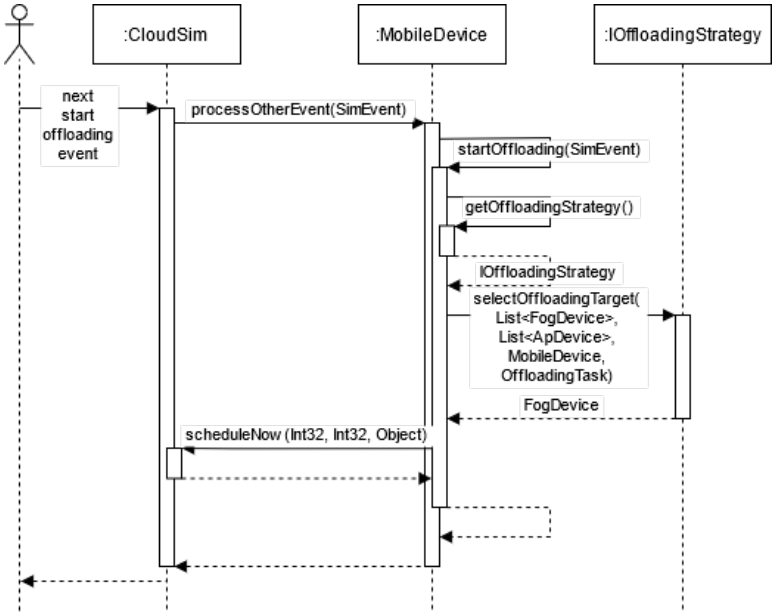


Figure 3-5: Sequence diagram of selecting an offloading target and scheduling task execution of the offloading task.

The *BEGIN_OFFLOAD_TASK_EXECUTION* event dispatched by the simulator will now start the task offloading process on the target fog node (see figure 3-6). The node begins executing the offloading task in the *beginOffloadingTaskExecution(SimEvent): void* method. The compute resources specified by the offloading task needed for executing the task will be allocated by invoking the *addOffloadingTask(MobileDevice, OffloadingTask): void* method on the fog node. Also, if the fog node is a compromised fog node, the *addOfllloadingTask* method will notify the attacker that the mobile end-device started a task execution on that fog node. After allocating the resources and potentially notifying the attacker, the fog node will calculate the time needed for executing the offloading task including the transmission time from and to the fog node by calling the *calculateResponseTime(List<FogDevice>, List<ApDevice>, MobileDevice, FogDevice, OffloadingTask): double* method on the *IOffloadingResponseTimeCalculator*. As described in the assumptions this calculated estimated response time will be used as actual response time for the offloading task, meaning potential delays while executing the task or fluctuations in the node's available resources or in the network are not influencing the tasks execution or transmission. Further, the transmission of data from and to the node is not simulated. Instead, the entire response time is awaited on the fog node itself before the mobile end-device is immediately notified about the task completion (i.e., no network transmission times included). Lastly, the calculated response time is used to schedule the *END_OFFLOAD_TASK_EXECUTION* event with the offloading task as event message data. This event is responsible for notifying the fog node that the task is finished executing i.e., notifying the mobile end-device of the task completion.

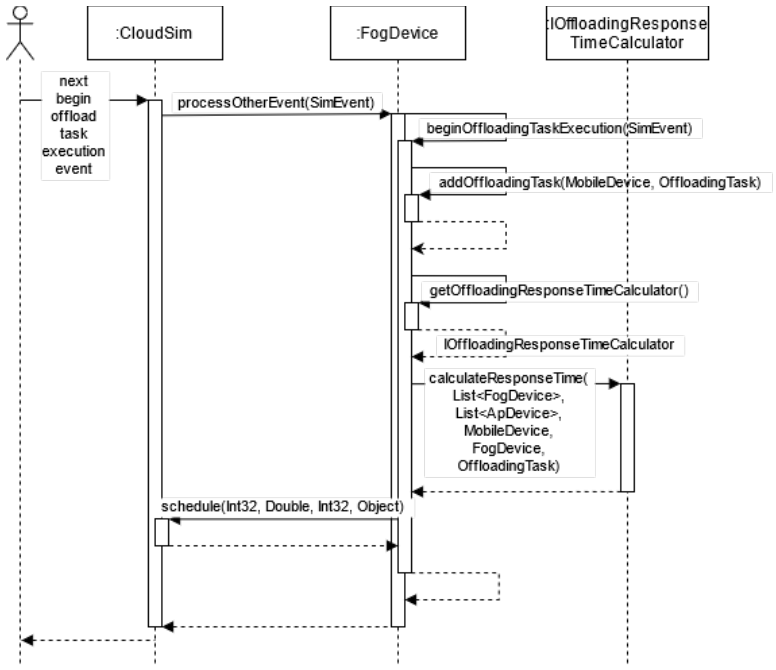


Figure 3-6: Sequence diagram of beginning the offloading task execution on the selected *FogDevice*.

When the offloading task is finished, that is when the *END_OFFLOAD_TASK_EXECUTION* event is dispatched by the simulator, the fog node calls its *endOffloadingTaskExecution(SimEvent event): void* method (see figure 3-7). The fog node deallocates the used resources of the task by calling

removeOffloadingTask(MobileDevice, OffloadingTask): void method on the fog node. Also, if the fog node is a compromised fog node and the offloading task is the last task of the source mobile end-device being executed on that fog node, the *removeOffloadingTask* method will notify the attacker that the mobile end-device finished executing all its tasks on that node. Lastly, the *FINISHED_OFFLOADING* event will be scheduled for immediate execution on the source mobile end-device. This event is intended to notify the mobile end-device of the tasks completion and therefore contains the offloading task as event message data.

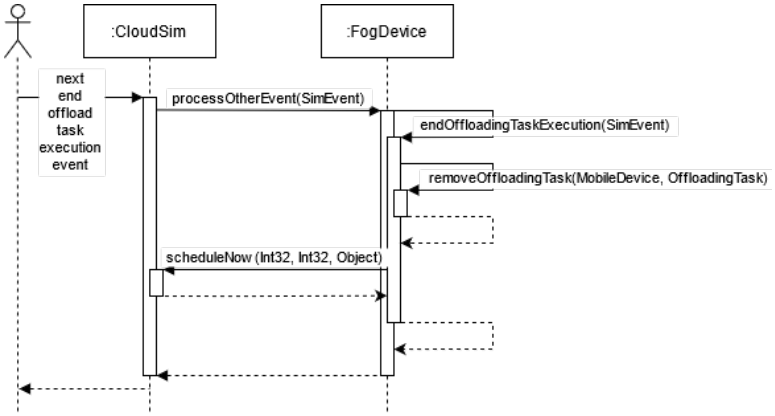


Figure 3-7: Sequence diagram of ending the offloading task execution and notifying the *MobileDevice* about task completion.

Lastly, after the offloading task is completed on the target fog node, the *MobileDevice* is notified by the *FINISHED_OFFLOADING* event dispatched by the simulator. The mobile end-device now calls its *endOffloading(SimEvent): void* method (see figure 3-8). The mobile end-device now registers a new *MAKE_OFFLOADING_SCHEDULING_DECISION* event in the simulators future queue for immediate dispatch. This is done so that the mobile end-device has the opportunity to immediately schedule a new offloading task, thus, completing the cycle (the cycle restarts at figure 3-4 with the mobile end-device making an offloading scheduling decision).

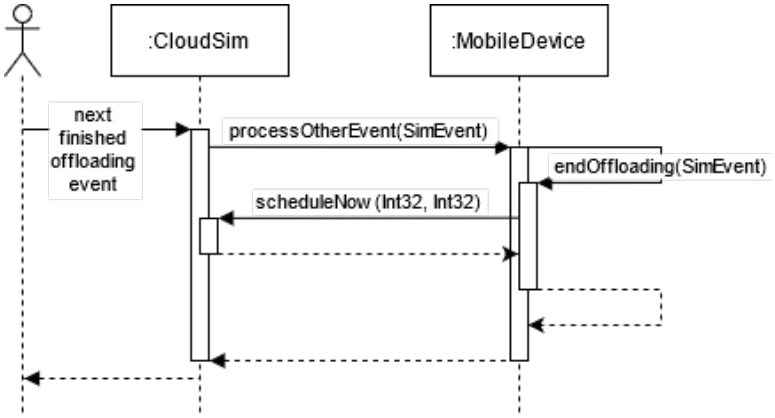


Figure 3-8: Sequence diagram of finishing an offloading task and rescheduling a *MAKE_OFFLOADING_SCHEDULING_DECISION* event for the *MobileDevice*.

In order to abstract the process of calculating the response time (as described in chapter 3.2.2 – Calculating an Offloading Tasks Response Time), an interface called *IOffloading-ResponseTimeCalculator* is introduced. The response time calculator implementation is associated in a per mobile end-device or fog nodes basis, giving those devices the ability to use different ways of calculating the response time. Also, one can implement and test different response time calculation algorithms. Whenever the response time must be calculated, the response time calculator instance can be retrieved by calling the getter (*getOffloadingResponse-TimeCalculator()*) on the mobile end-device or fog node.

3.3 Simulator Environment, Setup, and User Code

This sub-chapter describes the necessary changes in the simulation environment, setup, and the new user code i.e., the schedulers implementation and the common base class for the implemented offloading strategies. First, the common abstract base class and its algorithm used to pre-filter the set of fog nodes down to a set of fog nodes with a response time below a certain threshold is described. Then, the scheduler implementation used in the conducted simulations will be described.

As both offloading strategies need to reduce the available set of fog nodes down to a set of fog nodes below a certain threshold, an abstract base class was introduced. This enables a more isolated way of unit testing the algorithm and reducing code duplication. Figure 3-9 shows the pseudo code of filtering the devices.

Algorithm 1: Get devices below a certain threshold

Input: Set of fog nodes **F**, Set of access points **A**, Source mobile end-device **m**, Offloading task **t**, Threshold **c**
Result: Set of fog nodes below the threshold **R**
for *f in F* **do**
    rt = calculateResponseTime(F, A, m, f, t)
    if *rt is less or equals to c* **then**
        Add f to R
    end
end
Return R

Figure 3-9: Pseudo code of filtering devices below a certain threshold.

To filter the set of available fog nodes the algorithm loops through all fog nodes. For every fog node it calculates the response time by using the source mobile end-devices *IOffloading-ResponseTimeCalculator*. Then, it compares the calculated response time with the provided threshold and adds the fog node to the result list when its response time is less or equals to the threshold. Lastly, the method returns the result set of fog nodes.

Next, figure 3-10 shows the class diagram of the implemented offloading strategies. These strategies are considered as user code, as they are usually dependent on what the user tries to accomplish with the simulation.

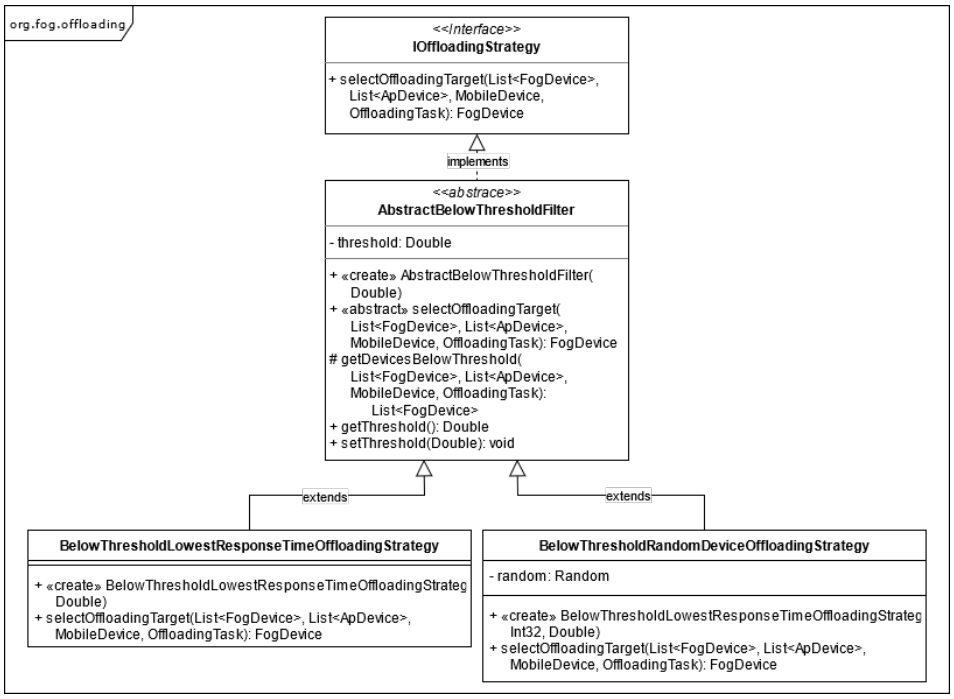


Figure 3-10: Class diagram of the implemented offloading strategies.

As described in chapter 3.2.3 – Implementing Task Offloading in *LocPrivFogSim* a mobile end-device needs a scheduler to schedule offloading tasks. This scheduler is also considered as user code. In the context of this bachelor's thesis the scheduler is kept simple. The class implementing the interface is called *FixedOffloadingScheduler* as it initially schedules offloading tasks at the start of the simulation at a specified interval (see figure 3-11). To create a fixed offloading scheduler an interval must be provided as well as the input/output data size and the number of instructions to complete the task execution. The parameters chosen for this bachelor's thesis are described in chapter 4.4 – Test Application.

```
public void scheduleOffloadingTask(MobileDevice mobileDevice) {
    if (hasScheduled())
        return;

    for (int i = 0; i < MaxAndMin.MAX_SIMULATION_TIME; i += getIntervall())
        mobileDevice.scheduleOffloadingTask(i,
                new OffloadingTask(id++, mobileDevice.getId(), inputDataSize, mi, outputDataSize));

    this.hasScheduled = true;
}
```

Figure 3-11: The schedule method of the *FixedOffloadingScheduler* implementation.

3.4 Offloading Strategy 1: Offloading to a Random Fog Node From a Pre-Filtered Set of Nodes

In the first offloading strategy, a random fog node will be selected from the filtered set of fog nodes. To pre-filter all fog nodes down to a set of fog nodes below a certain threshold, the strategy extends from the abstract base class *AbstractBelowThresholdFilter*. This offloading strategy's implementation can be found in the *BelowThresholdRandomDeviceOffloadingStrategy* class. Figure 3-12 shows the code of this strategy.

```
@Override
public FogDevice selectOffloadingTarget(List<FogDevice> serverCloudlets, List<ApDevice> apDevices,
        MobileDevice source, OffloadingTask task) {

    List<FogDevice> suitableDevices = getDevicesBelowThreshold(serverCloudlets, apDevices, source, task);

    Log.formatLine("BelowThresholdRandomDeviceOffloadingStrategy found %d devices below the threshold of %,.4fs",
            suitableDevices.size(), getThreshold());

    int i = this.random.nextInt(suitableDevices.size());
    FogDevice device = suitableDevices.get(i);

    Log.formatLine("Selected a random fog device: %s", device.getName());

    return device;
}
```

Figure 3-12: Implementation of offloading strategy 1.

As figure 3-12 shows, from the set of suitable fog nodes one random fog node will be selected and returned. To select a random node javas *Random* class is used. While instantiating the offloading strategy a seed is provided for the random number generator.

Finally, if the list of suitable fog nodes is empty, meaning there is no fog nodes below the defined threshold, the random number generator throws an exception. If an exception is thrown, the simulation is stopped, and the source mobile end-device is considered not being able to offload a task to any fog node. As this bachelor's thesis is about finding the impact of task offloading on location privacy attacks, offloading no task at all is considered as not useful.

3.5 Offloading Strategy 2: Offloading to a Fog Node With the Lowest Response Time From a Pre-Filtered Set of Nodes

In the second offloading strategy, the fog node with the lowest response time from the pre-filtered set of fog nodes will be returned. Again, to pre-filter all fog nodes down to a set of fog nodes below the provided threshold, this offloading strategy also extends from the abstract base class *AbstractBelowThresholdFilter*. This offloading strategy's implementation can be found in the *BelowThresholdLowestResponseTimeOffloadingStrategy* class. Figure 3-13 shows the code of this strategy.

As figure 3-13 shows, the set of suitable fog nodes will be searched for the fog node with the lowest response time. As starting point the maximum double value will be used. For every fog node in the suitable fog node list the response time will be calculated and compared with the current minimum response time. When the calculated response time is less than the current minimum response time, the fog node is stored in a local variable and the minimum response time is set to the newly calculated response time. At the end, the fog node will be returned.

```
@Override
public FogDevice selectOffloadingTarget(List<FogDevice> serverCloudlets, List<ApDevice> apDevices,
        MobileDevice source, OffloadingTask task) {

    List<FogDevice> suitableDevices = getDevicesBelowThreshold(serverCloudlets, apDevices, source, task);

    double minResponseTime = Double.MAX_VALUE;
    FogDevice device = null;

    // Search the device with the lowest response time
    for (FogDevice target : suitableDevices) {
        double responseTime = source.getOffloadingResponseTimeCalculator()
                .calculateResponseTime(serverCloudlets, apDevices, source, target, task);

        if (responseTime < minResponseTime) {
            minResponseTime = responseTime;
            device = target;
        }
    }

    Log.formatLine("BelowThresholdLowestResponseTimeeOffloadingStrategy: Selected %s device below the threshold of"
            + " %,.4fs with response time %,.4fs", device.getName(), getThreshold(), minResponseTime);

    return device;
}
```

Figure 3-13: Implementation of offloading strategy 2.

In this offloading strategy, if the list of suitable fog nodes is empty, meaning there is no fog nodes below the defined threshold, *null* will be returned as target fog node. This will lead to a *NullPointerException* in the simulator. If an exception is thrown, the simulation is stopped, and the source mobile end-device is considered not being able to offload a task to any fog node. As this bachelor's thesis is about finding the impact of task offloading on location privacy attacks, offloading no task at all is considered as not useful.

3.6 Modifying the Attackers Observer Pattern

In order to notify the adversary about events in the fog system i.e., a mobile end-device handoff between two access points, migration of an VM introduced in *MobFogSim*, or the newly introduced computation offloading events of starting and ending an offloading task on a fog node, *LocPrivFogSim* utilizes a programming pattern known as observer pattern. The *Attacker* registers itself as observer in all compromised fog nodes. If an event is triggered on a fog node the *Attacker* is notified by calling the its *update(MobileDevice mobileDevice, String event): void* method. The method signature of this *update* method is modified to cover some additional scenarios. Firstly, the *FogDevice* responsible for triggering the event is passed into the *update* method as *source* parameter. Previously, the *Attacker* was responsible for choosing the fog node which caused the firing of the event by taking the *MobileDevice* location and finding the closest fog node. However, in a more real-world scenario, the adversary knows which of its compromised fog nodes has triggered the event. Hence, passing the *source* fog node of the event as parameter. Secondly, as mentioned above different events can be the cause of notifying the adversary. Currently, those could be either a handoff event, a migration event, or an offloading event. To be able to test and evaluate different scenarios, the type of the event must be passed down in the *update* method so that the *Attacker* can filter for specific events. Hence, an *Integer* value representing the event type is passed into the *update* method. The *Attacker*'s modified *update* method signature now looks as following: *public void update(FogDevice source, MobileDevice mobileDevice, int eventType, String event)*.

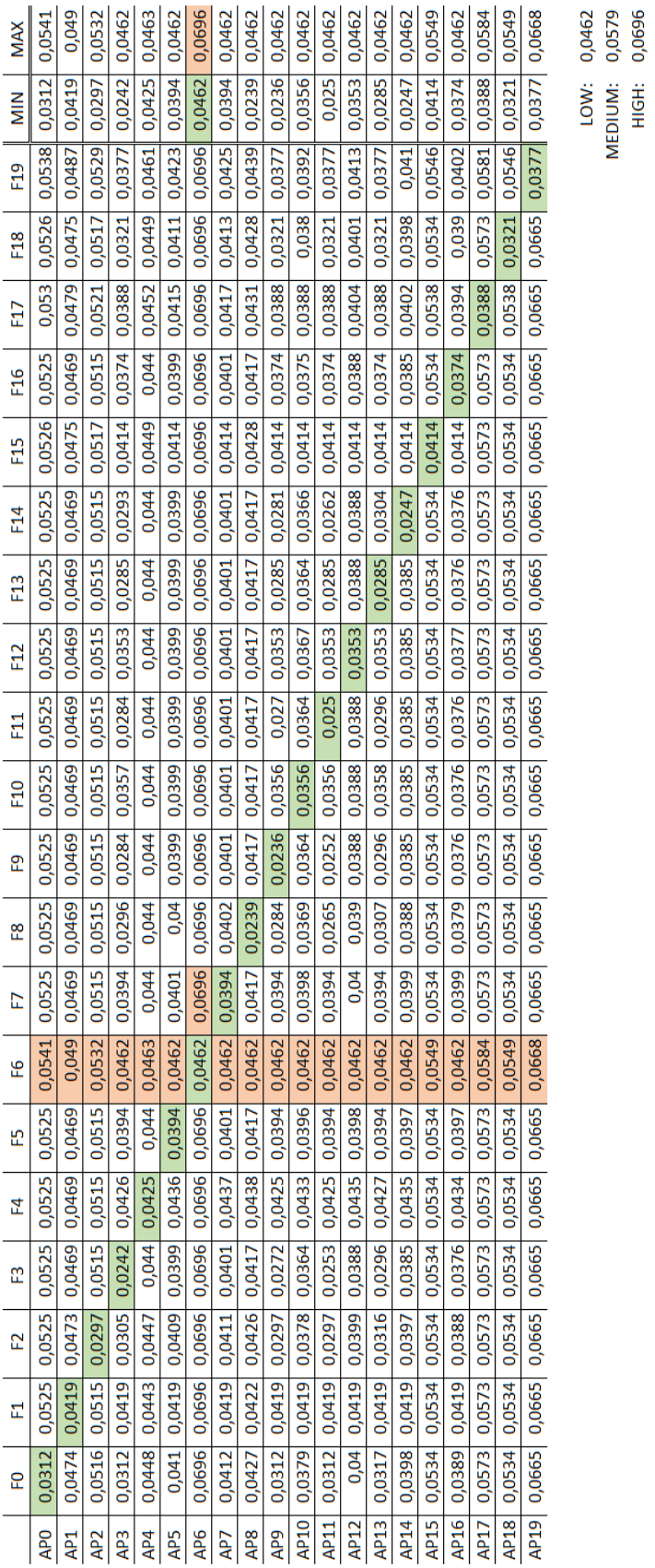
4 Experiment Design

At the beginning of this chapter, the test application used for this bachelor's thesis is described. Then, the experiments of the conducted simulations using the test system are presented. Next, an overview of how *LocPrivFogSim* measures the knowledge obtained by the adversary is given i.e., the two built-in metrics are presented. However, these two metrics are not sufficient for the evaluation of computation offloading, as in computation offloading a mobile end-device can choose a fog node other than the closest fog node as offloading target. Therefore, the two built-in metrics will be extended by two newly introduced metrics. In total, with the help of these four metrics, the experiments of computation offloading on attacks on location privacy is made comparable.

4.1 Test Application

To conduct the simulations and experiments of this bachelor's thesis the following test system, based on the test system used in the paper by *T. Wettig* and *Z.Á. Mann* [16] and the bachelor's thesis by *T. Wettig* [17], is used:

The test application defines a fixed region with a coordinate size of 50x50 representing for example a city. The region is divided into 0.1 intervals. In this region a network consisting of 20 fog nodes, 20 access points, and 1 mobile end-device is set up. The mobile end-device moves through the region on one random selected path of 50 predefined paths. The coverage of the fog nodes is assumed to be optimally circular. In reality, many other external factors influence the coverage of fog nodes so that the coverage is usually not optimally circular. The positions of the fog nodes and access points are randomly chosen but are kept consistent between all conducted simulations i.e., the positions are fixed in all simulation runs. The resources a device is created with, in this case the up or download bandwidth of a fog node or access point, are also chosen randomly but are kept consistent across all simulations runs as well. The CPU resources of a fog node is set to 28.000.000 MIPS. Based on the CPU and bandwidth resources the minimum possible response times can be calculated. Figure 4-1 shows a matrix with the minimum possible calculated response times from an access point on which the mobile end-device would connect to, to all fog nodes in the network. While calculating the minimum possible response time, the CPU resources are assumed to be not utilized. This matrix is later used to select the thresholds for the simulations (see chapter 4.2). Further, in the test application, one attacker is created that controls one or more random selected fog nodes. The ratio of compromised fog nodes is passed into the test application by a command line argument [16, 17]. To adapt the test system to the new offloading requirements i.e., choosing an offloading strategy and setting an offloading threshold, two new additional command line arguments are introduced. Table 4-1 lists the command line arguments that need to be passed in, as well as their data type, and a brief description.



	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	MIN	MAX
AP0	0,0312	0,0525	0,0525	0,0525	0,0525	0,0525	0,0541	0,0525	0,0525	0,0525	0,0525	0,0525	0,0525	0,0525	0,0525	0,0526	0,0525	0,053	0,0526	0,0538	0,0312	0,0541
AP1	0,0474	0,0419	0,0473	0,0469	0,0469	0,0469	0,049	0,0469	0,0469	0,0469	0,0469	0,0469	0,0469	0,0469	0,0469	0,0475	0,0469	0,0479	0,0475	0,0487	0,0419	0,049
AP2	0,0516	0,0515	0,0297	0,0515	0,0515	0,0515	0,0532	0,0515	0,0515	0,0515	0,0515	0,0515	0,0515	0,0515	0,0515	0,0517	0,0515	0,0521	0,0517	0,0529	0,0297	0,0532
AP3	0,0312	0,0419	0,0305	0,0242	0,0426	0,0394	0,0462	0,0394	0,0296	0,0284	0,0357	0,0284	0,0353	0,0285	0,0293	0,0414	0,0374	0,0388	0,0321	0,0377	0,0242	0,0462
AP4	0,0448	0,0443	0,0447	0,044	0,0425	0,044	0,0463	0,044	0,044	0,044	0,044	0,044	0,044	0,044	0,044	0,0449	0,044	0,0452	0,0449	0,0461	0,0425	0,0463
AP5	0,041	0,0419	0,0409	0,0399	0,0436	0,0394	0,0462	0,0401	0,04	0,0399	0,0399	0,0399	0,0399	0,0399	0,0399	0,0414	0,0399	0,0415	0,0411	0,0423	0,0394	0,0462
AP6	0,0696	0,0696	0,0696	0,0696	0,0696	0,0696	0,0462	0,0696	0,0696	0,0696	0,0696	0,0696	0,0696	0,0696	0,0696	0,0696	0,0696	0,0696	0,0696	0,0696	0,0462	0,0696
AP7	0,0412	0,0419	0,0411	0,0401	0,0437	0,0401	0,0462	0,0394	0,0402	0,0401	0,0401	0,0401	0,0401	0,0401	0,0401	0,0414	0,0401	0,0417	0,0413	0,0425	0,0394	0,0462
AP8	0,0427	0,0422	0,0426	0,0417	0,0438	0,0417	0,0462	0,0417	0,0239	0,0417	0,0417	0,0417	0,0417	0,0417	0,0417	0,0428	0,0417	0,0431	0,0428	0,0439	0,0239	0,0462
AP9	0,0312	0,0419	0,0297	0,0272	0,0425	0,0394	0,0462	0,0394	0,0284	0,0236	0,0356	0,027	0,0353	0,0285	0,0281	0,0414	0,0374	0,0388	0,0321	0,0377	0,0236	0,0462
AP10	0,0379	0,0419	0,0378	0,0364	0,0433	0,0396	0,0462	0,0398	0,0369	0,0364	0,0356	0,0364	0,0367	0,0364	0,0366	0,0414	0,0375	0,0388	0,038	0,0392	0,0356	0,0462
AP11	0,0312	0,0419	0,0297	0,0253	0,0425	0,0394	0,0462	0,0394	0,0265	0,0252	0,0356	0,025	0,0353	0,0285	0,0262	0,0414	0,0374	0,0388	0,0321	0,0377	0,025	0,0462
AP12	0,04	0,0419	0,0399	0,0388	0,0435	0,0398	0,0462	0,04	0,039	0,0388	0,0388	0,0388	0,0353	0,0388	0,0388	0,0414	0,0388	0,0404	0,0401	0,0413	0,0353	0,0462
AP13	0,0317	0,0419	0,0316	0,0296	0,0427	0,0394	0,0462	0,0394	0,0307	0,0296	0,0358	0,0296	0,0353	0,0285	0,0304	0,0414	0,0374	0,0388	0,0321	0,0377	0,0285	0,0462
AP14	0,0398	0,0419	0,0397	0,0385	0,0435	0,0397	0,0462	0,0399	0,0388	0,0385	0,0385	0,0385	0,0385	0,0385	0,0247	0,0414	0,0385	0,0402	0,0398	0,041	0,0247	0,0462
AP15	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0549	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0414	0,0534	0,0538	0,0534	0,0546	0,0414	0,0549
AP16	0,0389	0,0419	0,0388	0,0376	0,0434	0,0397	0,0462	0,0399	0,0379	0,0376	0,0376	0,0376	0,0377	0,0376	0,0376	0,0414	0,0374	0,0394	0,039	0,0402	0,0374	0,0462
AP17	0,0573	0,0573	0,0573	0,0573	0,0573	0,0573	0,0584	0,0573	0,0573	0,0573	0,0573	0,0573	0,0573	0,0573	0,0573	0,0573	0,0573	0,0388	0,0573	0,0581	0,0388	0,0584
AP18	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0549	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0534	0,0538	0,0321	0,0546	0,0321	0,0549
AP19	0,0665	0,0665	0,0665	0,0665	0,0665	0,0665	0,0668	0,0665	0,0665	0,0665	0,0665	0,0665	0,0665	0,0665	0,0665	0,0665	0,0665	0,0665	0,0665	0,0377	0,0377	0,0668

LOW: 0,0462
MEDIUM: 0,0579
HIGH: 0,0696

Figure 4-1: Matrix of the minimum possible response times (values in seconds).

Legend: F = *FogDevice*; AP = *ApDevice*

Table 4-1: Description of the simulation parameters set by the shell script.

Name	Unit	Description
Scenario	*Int32*	Controls the scenario executed by the simulator. Possible numerical values are: 1. Adversary knows location of compromised fog nodes only 2. Adversary knows location of all fog nodes
Rate of compromised devices	*Double*	Controls the percentage of compromised devices.
Seed 2	*Int32*	Radom number generator seed for creating the *MobileDevice*, connections, and more.
Seed 3	*Int32*	Random number generator seed for the *Attacker*. This seed is also used by the random offloading strategy to select a random fog node. It should be noted that a new random number generator instance is created with the seed.
Grid interval	*Double*	Grid resolution.
Offloading threshold	*Double*	Response time threshold below which the devices are either selected randomly (i.e., offloading strategy 1) or selected by the lowest response time (i.e., offloading strategy 2).
Offloading strategy	*String*	Name of the offloading strategy to use in the simulation. Possible values are: - *BelowThresholdRandomDevice* (Offloading Strategy 1) - *BelowThresholdLowestResponseTime* (Offloading Strategy 2)

Next to the two new command line arguments some other input parameters must be set such as the schedulers interval and the offloading task data. Therefore, as scheduler interval 1000 ticks were chosen which is equal to offloading a task every second. For the offloading tasks, the input/output data size and number of instructions were chosen based on some of the numbers in the paper by *H. Gupta et al.* [7] case study 2 (see table 4-2). These offloading parameters are fixed throughout the simulations and cannot be changed or controlled externally.

Table 4-2: Description of the chosen fixed offloading task parameters.

Parameter	Value
Input data size	20 MB
Output data size	2 MB
Number of instructions	2000 MI

The simulator and the implemented experiment[12] are built and packed into an executable jar. This jar file will be invoked by a shell script running the simulation. Each scenario is run 20 times incrementing the rate of compromised fog nodes by each run about 5 percent. The sample size within each run is set to 100. Each run is executed with a different random seed. To control the scenarios of the simulation and set the simulation parameters, javas command line arguments are utilized. The metrics produced are gathered and appended into a csv file by the simulator. The individual scenarios i.e., the experiments conducted in this bachelor's thesis are described in the next sub-chapter.

4.2 Scenarios of the Conducted Experiment

The simulations conducted in this bachelor's thesis are based on the simulations presented in the paper by *T. Wettig* and *Z.Á. Mann* [16]. In this paper, 4 scenarios were evaluated (see table 4-3).

Table 4-3: Overview of the considered scenarios in the paper by *T. Wettig* and *Z.Á. Mann* [16].

	Adversary knows location of compromised fog nodes only	**Adversary knows location of all fog nodes**
Mobile device is always connected to the fog system	Scenario 1.0	Scenario 2.0
Mobile device is allowed to disconnect	Scenario 1.1	Scenario 2.1

[12] The class *TestExample2* contains the experiments implementation and simulator setup.

In this bachelor's thesis the impact of computation offloading on the accuracy of location privacy attacks is evaluated. In computation offloading data formed in a request is offloaded to a remote node. This data, more precisely the requests content can be of different formats e.g., VM, container, raw input data. An offloading task executed on a remote node is a single unit of work, on which after completion a response is transmitted back to the source mobile end-device. The mobile end-device now might schedule a new offloading task and keep the connection alive or even end the connection as the offloading task is completed. Hence, in computation offloading, there is no guarantee that the source mobile end-device making the offloading request is connected at all times to the fog system. Furthermore, the mobile end-device is mobile and might transition from one access point to the next losing the connection to the remote fog node [11]. Therefore, it was decided to omit the scenarios related to: "Mobile device is always connected to the fog system". In the previous conducted researches i.e., the paper by *T. Wettig* and *Z.Á. Mann* [16] or the bachelor's thesis by *T. Wettig* [17] this was not an issues, as those relied on the migration and handoff implementations of *MobFogSim*, keeping the mobile end-device connected at all times by preforming live migrations [29].

Hence for this bachelor's thesis, the two scenarios where the mobile end-device is allowed to disconnect from the fog network remain and will be examined. In scenario 1.1 from the paper [16], the adversary only knows the location of the compromised fog nodes only, while in scenario 2.1 of the paper [16], the adversary knows the location of all fog nodes. The adversary can gain the information about all locations of all fog nodes in a multiple of different ways. For one, the adversary can triangulate and/or make observations with own known mobile end-devices. Furthermore, the adversary can use public available information [17]. Gaining this information is not part of this bachelor's thesis, as there are many known methods.

The offloading strategies implemented in this bachelor's thesis are both dependent on a threshold value for the response time, below which, depending on the strategy, a fog node is selected. It is assumed that the higher the threshold, the more fog nodes are available in the suitable nodes set for the strategy to choose from. As a result, it is presumed that, as more fog nodes are available to offload to, based on the strategy it is more likely that the adversary will not gain accurate information on the location of the mobile end-device. Contrary, the lower the threshold, the fewer fog nodes are available in the suitable nodes set for the strategy to choose from. Hence it is presumed, the accuracy should be higher. Therefore, to evaluate the impact of the threshold on the adversary's accuracy and either prove or disprove the hypothesis three thresholds are picked. For each threshold, the simulations are run with the adversary knowing only the location of compromised fog nodes (scenario 1.1), and the adversary knowing the locations of all fog nodes (scenario 2.1). In each simulation step, the rate of compromised fog nodes is incremented by 5%. Table 4-4 gives the overview of the scenarios conducted in this bachelor's thesis.

To select the thresholds for the offloading strategies a matrix with the minimum possible response times was calculated with the device and fog nodes available resources (figure 4-1). Based on this matrix, the lowest and highest thresholds are chosen. The medium threshold is selected as median threshold between the lowest and highest threshold.

Table 4-4: Overview of the conducted scenarios in this bachelor's thesis.

	Adversary knows location of compromised fog nodes only	**Adversary knows location of all fog nodes**
Low Threshold (46,2ms)	Scenario 1.1	Scenario 2.1
Medium Threshold (57,9ms)	Scenario 1.1	Scenario 2.1
High Threshold (69,6ms)	Scenario 1.1	Scenario 2.1

4.3 Measurability and Comparability of the Knowledge an Adversary Can Obtain in LocPrivFogSim

To evaluate the accuracy of the adversary's gained information on the locations of a mobile end-device and to make the above-described scenarios measurable and comparable, *LocPrivFogSim* provides two built-in metrics: trace comparison value, accuracy value.[13] These metrics are applied to the scenarios by running each scenario multiple times and presenting the adversary's gained knowledge.

Trace comparison

The trace comparison metric is a useful metric for determining the level of ambiguity of the adversary about the path of the mobile end-device in trajectory attacks. To measure the trace comparison value an "anonymity set" must be defined. The "anonymity set" is defined as the set of paths a mobile end-device can take in a defined region (e.g., a city) and is consistent with the observations the adversary made [16]. For example, to find the possible set of paths the adversary can identify paths on a city plan [17]. The real path of a mobile end-device is among these paths however the adversary does not know which path is the real path [16]. The set of paths is defined as $\mathrm{P} := \{p_1, \dots, p_n\}$.

The trace of a path p (where $p \in \mathrm{P}$ and the trace is denoted as $tr(p)$) is the sequence of fog nodes to which a mobile end-device connects while traversing p [16].

The observed trace of a path p (where $p \in \mathrm{P}$ and the observed trace is denoted as $otr(p)$) is what the adversary can observe from the trace of a given mobile end-device [16].

Periods in which the mobile end-device is connected to a non-compromised fog node or in which the mobile end-device is not connected to the fog system at all are marked with the symbol "*" in the observed trace $otr(p)$, as the adversary cannot distinguish between these states [16]. Therefore, $otr(p)$ is the trace $tr(p)$ of a path p, with sections of non-compromised fog nodes

[13] The complete description of the two built-in metrics "trace comparison" and "size of uncertainty region" are based on references [16] and [17].

replaced with "*". It is assumed that the adversary knows the observed trace $otr(p)$ of each possible path in P, for example by walking with its own mobile end-device along these paths capturing traces (sequence of connections the mobile end-device establishes to compromised fog nodes) [16, 17].

An observed trace for a mobile end-device following a path p recorded by an adversary is denoted as $otr'(p)$. The adversary checks which of the known observed traces $otr(p_i);\ p_i \in$ P is consistent with the recorded observed trace $otr'(p)$ i.e., $otr(p_i) \sim otr'(p)$. To check whether two traces are consistent with each other, the observed trace and the recorded observed trace need to be processed. For one, all occurrences of "*" need to be removed. Further, if this leads to two consecutive occurrences of the same fog node in the observed trace, those instances are replaced by a single occurrence of the fog node. Now, two observed traces are considered consistent if they are identical [16].

By comparing the recorded observed trace to all observed traces in P finding only the traces that are consistent, the adversary gets a set of possible paths consistent with the observations. The adversary also knows the mobile end-device followed one of the paths in the set. This is denoted as: $\mathrm{K} = \{p_i \mid 1 \leq \mathrm{i} \leq \mathrm{n}, otr(p_i) \sim otr'(p)\}$ [16].

The trace comparison value, showing the adversary's success, is defined as the number of paths in K i.e., $TrC = |\mathrm{K}|$ [16]. The probability of a mobile end-device paths being one of the observed traces in K is defined for every trace as $Pr(p_i) = \frac{1}{|K|}; p_i \in$ K [17]. The lower the number of paths in K the more successful the adversary is. The ideal case would be a trace comparison value of exactly 1, meaning the attacker determined the path of the mobile end-device. The worst case would be n, being the number of available paths in P [16].

Size of uncertainty region

The size of uncertainty region is the area to which extent the adversary can narrow down the mobile end-device location.[14] While the mobile end-device moves, this region changes in size. So, to measure the adversary's knowledge, the size is averaged over time. Hence, the time is sliced into intervals of size Δt with $t_1, \ldots, t_r$ point in times and $t_j = t_{j-1} + \Delta t$. The total time is $T = r * \Delta t$, where r is the number of points in time i.e., the number of slices [16].

At a specific point in time t_j, the area in which the adversary can narrow down the location of the mobile end-device is denoted as Q_{t_j}. The total considered area is represented by V [16]. At points in time where the mobile end-device is connected to a non-compromised fog node, the adversary can only narrow down the area Q_{t_j} to the area of the non-compromised fog nodes. If the mobile end-device is not connected at all, the area Q_{t_j} cannot be narrowed down at all.

[14] This is also referred to as accuracy value in the paper by *T. Wettig* and *Z.Á. Mann* [16].

The size of uncertainty region over time is computed as $SoUR = \left(\sum_{j=1}^{r} Q_{t_j} * \Delta t\right)/(T * V)$ and ranges between 0 and 1. The lower the size of uncertainty region $SoUR$ over the time the more precise is the knowledge of the adversary about the mobile end-device location [16].

4.4 Extending the Metrics in LocPrivFogSim

As already mentioned, the two built-in metrics are not sufficient for the evaluation of computation offloading, as the two metrics are built with the assumption of the compromised and notifying fog node always being the closes fog node to the mobile end-device. However, in computation offloading a mobile end-device can choose a fog node other than the closest fog node as offloading target. To still be able to compare the results of this bachelor's thesis to the results of the paper by *T. Wettig* and *Z.Á. Mann* [16], the two built-in metrics will not be modified. This means the adversary still assumes the mobile end-device connects to the physical closest fog node. Therefore, for each of the two metrics a new additional metric is introduced, and it is assumed that the adversary can also be wrong.

Relative path hit

In the calculation of the trace comparison value, the adversary calculates a set of possible paths the mobile end-device could have taken. If the actual path the mobile end-device took through the region is not in the set of paths K the adversary has observed, then the relative path hit is zero, meaning the adversary was not able to identify the correct path. Otherwise, the relative path hit is calculated as $rph = \frac{1}{TrC}$, which is the same as the probability of a mobile end-device path being one of the observed traces in K; $Pr(truePath) = \frac{1}{TrC}$; $truePath \in$ K. The values of rph range between 0 and 1. The higher the value is the less false paths are in the calculated possible paths set K, meaning the more precise is the adversary prediction about the mobile end-devices path, with a value of 1 being the exact path of the mobile end-device being found be the adversary.

Area hit duration

The size of uncertainty region is the area to which extent the adversary can narrow down the mobile end-device location over time. However, under the assumption that the adversary does make mistakes, the uncertainty region is not necessary equal to the region where the mobile end-device was in reality. Therefore, the area hit duration gives insights about the portion of time the mobile end-device was actually in the region where the adversary thought it was. The values of the area hit duration range between 0 and 1, where the higher the value the more precise the adversary was about the position in the uncertainty region.

5 Results, Comparison and Discussion

This chapter presents the results of the conducted simulations in the test system by discussing the results of the individual offloading strategies in chapter 5.1 and chapter 5.2 respectively. In each chapter the comparison is performed for each individual metric. For each metric, scenario 1.1 (the adversary knows the location of the compromised fog nodes only) and scenario 2.1 (the adversary knows the location of all fog nodes) of the paper [16] within each threshold are compared. For every gathered metric, the 3 thresholds are presented as 3 individual figures. In the last sub-chapter, the found results are discussed.

5.1 Offloading Strategy 1: Offloading to a Random Fog Node From a Pre-Filtered Set of Nodes

In this chapter the results of offloading strategy 1 i.e., offloading to a random fog node from a pre-filtered set of nodes, are compared with the findings of the paper by *T. Wettig* and *Z.Á. Mann* [16]. First, the trace comparison values are compared with the findings in [16]. Then, the relative path hit values are correlated with the findings of the trace comparison values. These values however are not compared with [16] as the paper does not include this new metric. Next, the size of uncertainty region is compared with the findings in [16]. Lastly, the area hit duration is correlated with the finding of the size of uncertainty region. These values are also not compared with [16] as the paper also does not include this new metric.

5.1.1 Trace Comparison Value

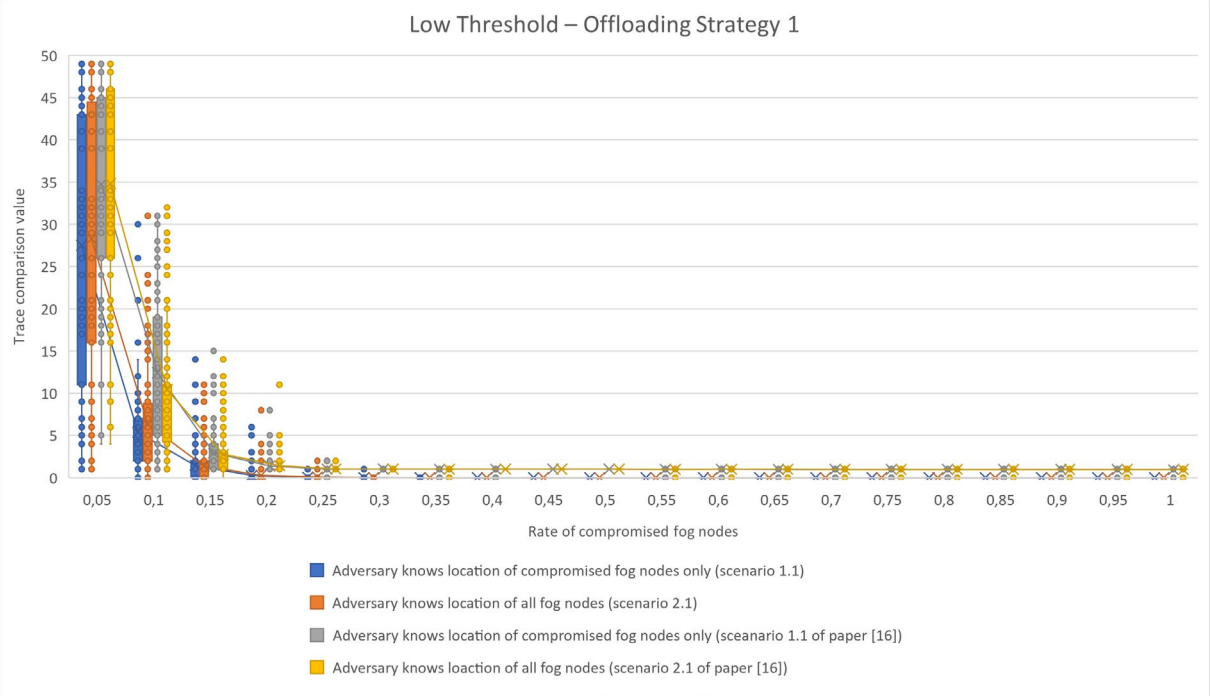


Figure 5-1: Results in terms of the trace comparison value for offloading strategy 1 with a low threshold.

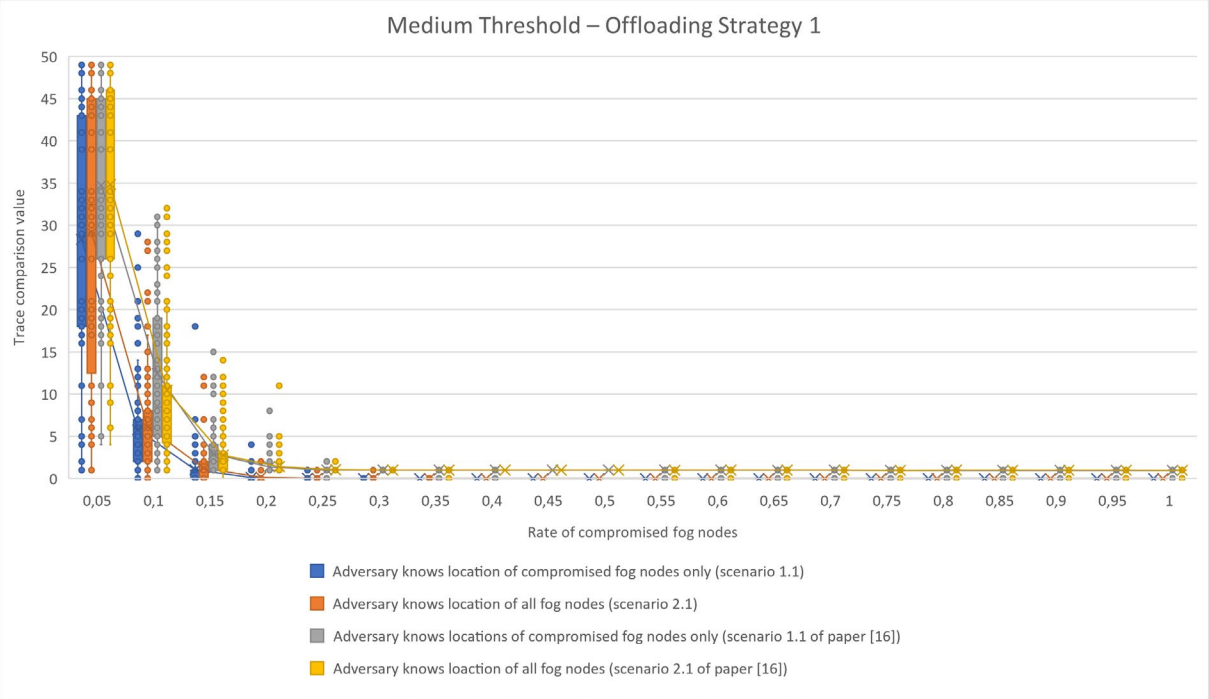


Figure 5-2: Results in terms of the trace comparison value for offloading strategy 1 with a medium threshold.

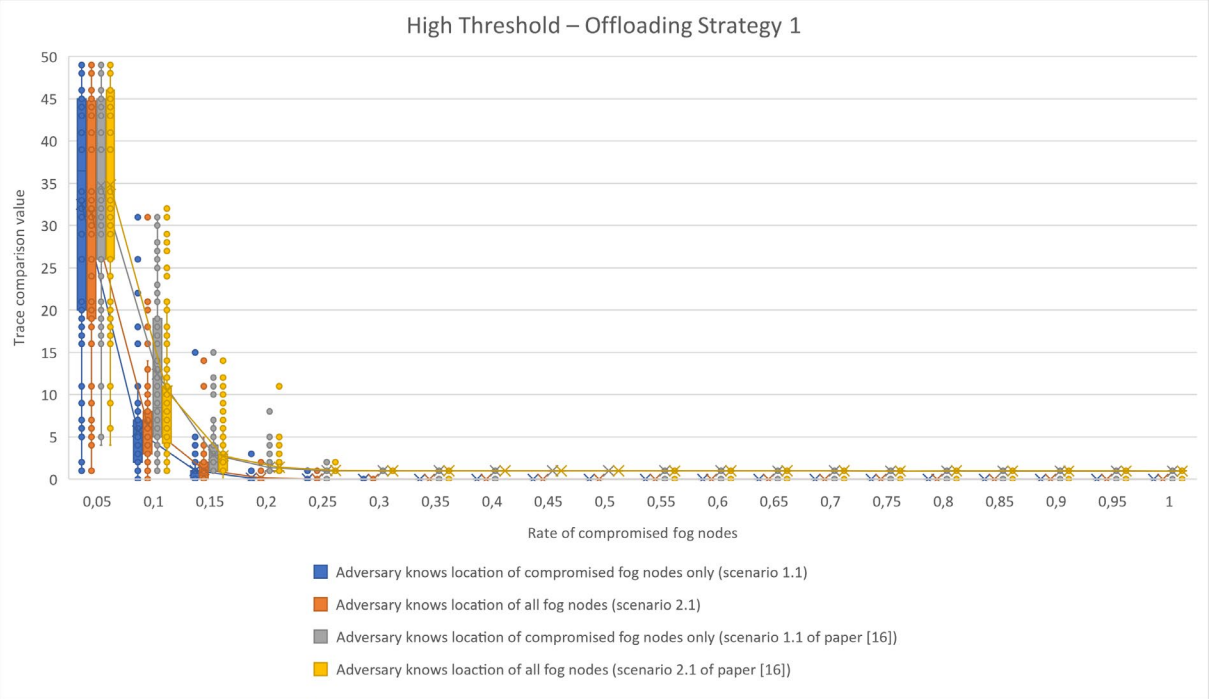


Figure 5-3: Results in terms of the trace comparison value for offloading strategy 1 with a high threshold.

Comparing the trace comparison values of offloading strategy 1 with the results of the paper [16] one can see that with a ratio of 5% of compromised fog nodes the number of possible paths is lower than the findings in the paper. The distribution of the found possible paths is higher than in the paper but decreases at 10% of compromised fog nodes way faster than in the paper. The overall shape of the curves of the paper and this bachelor's thesis are the same, meaning the number of

found possible paths decreases the higher the percentage of compromised fog nodes. Lastly, at 35% of compromised fog nodes the adversary does not find any possible path i.e., the trace comparison value is equal to 0, while in the paper the adversary is able to narrow down the set of path to exactly the right path i.e., the trace comparison value is exactly one.

In terms of the offloading threshold one can see the higher the offloading threshold is i.e., the more fog nodes are in the set from which the strategy can choose from, the lower is the initial scattering distribution of possible path at the ration of 5% of compromised fog nodes. In the lowest threshold the trace comparison value ranging from 12 up to 43 found possible paths, in the medium threshold the value ranges from 18 to 45 found possible paths, and in the highest threshold having the smallest scattering the value ranges from 20 to 45. However, the higher the threshold the more is the initial scattering distribution moved up i.e., resulting in more possible paths. After the initial level 5% of compromised fog nodes there is no difference in the distribution between the different thresholds.

In terms of attacks on location privacy for mobile end-device one can see, that considering computation offloading within offloading strategy 1 regardless of the offloading threshold at 35% of compromised fog nodes the adversary is not able to find the path of the mobile end-device. However, for the values below 35% of compromised fog nodes the adversary finds less possible paths for a mobile end-device giving the adversary a slight advantage compared to the findings in the paper.

5.1.2 Relative Path Hit Value

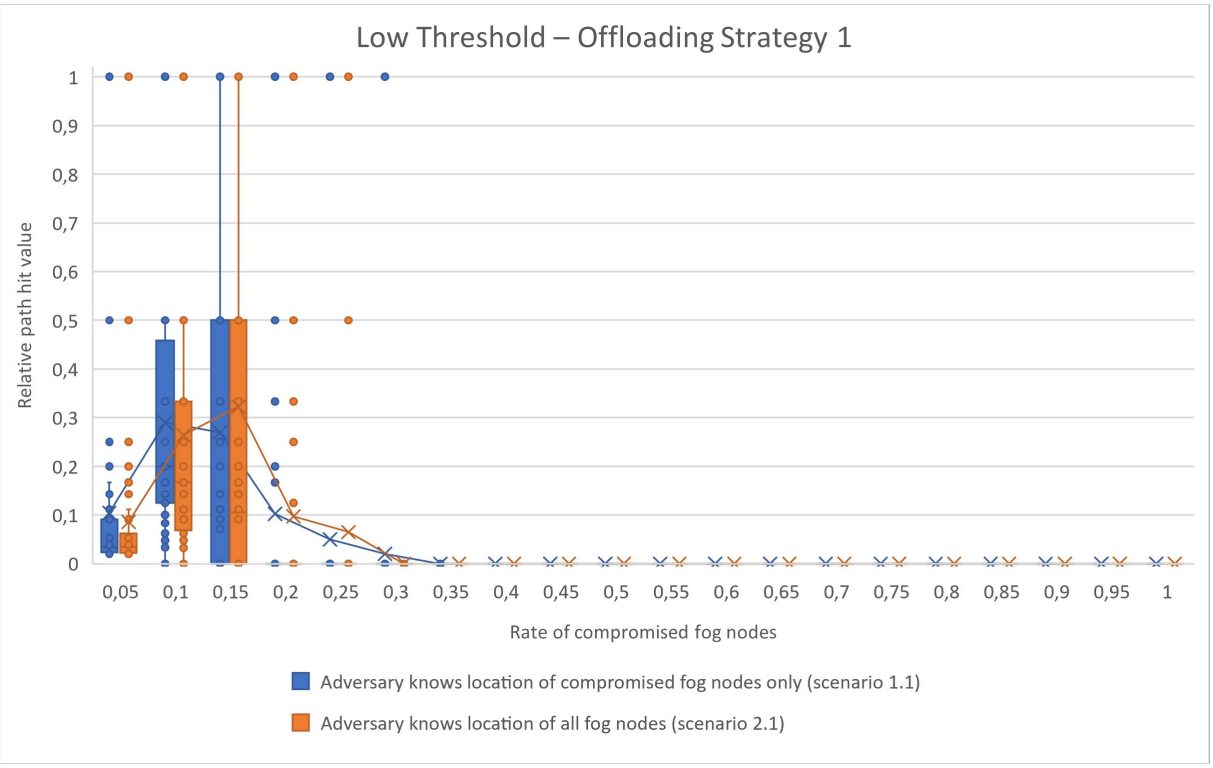


Figure 5-4: Results in terms of the relative path hit value for offloading strategy 1 with a low threshold.

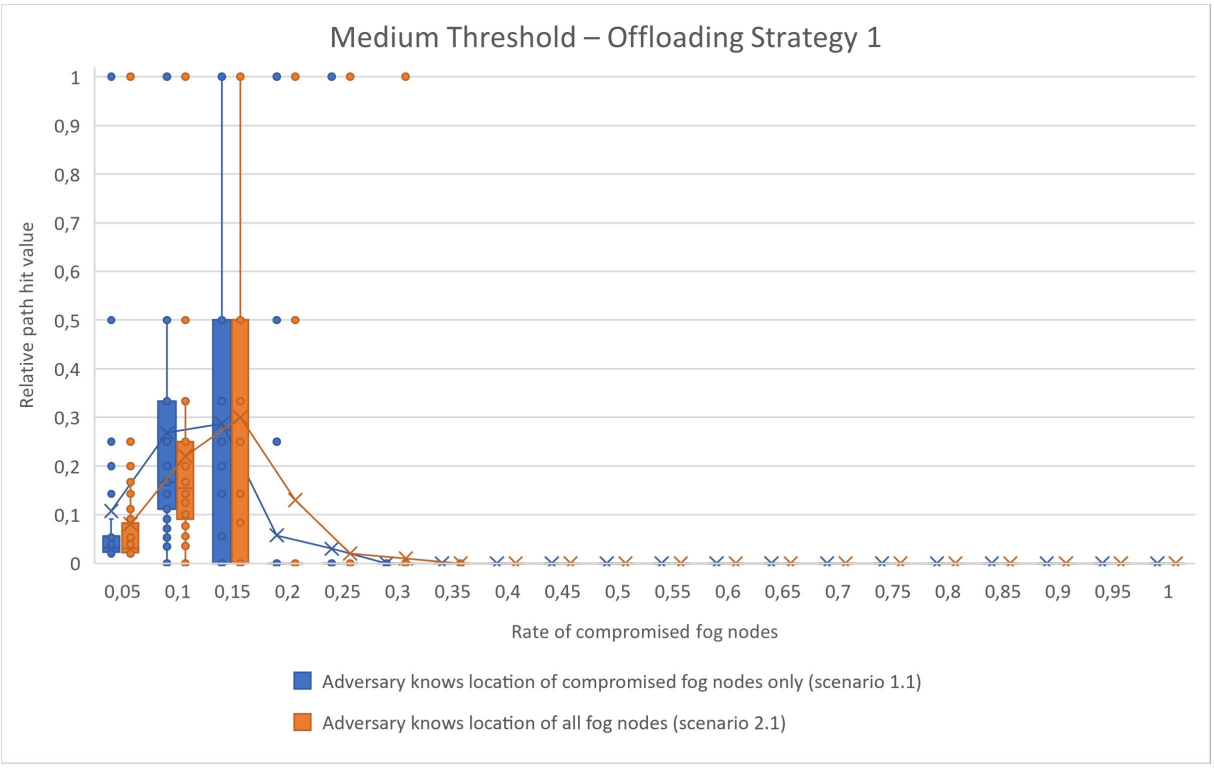


Figure 5-5: Results in terms of the relative path hit value for offloading strategy 1 with a medium threshold.

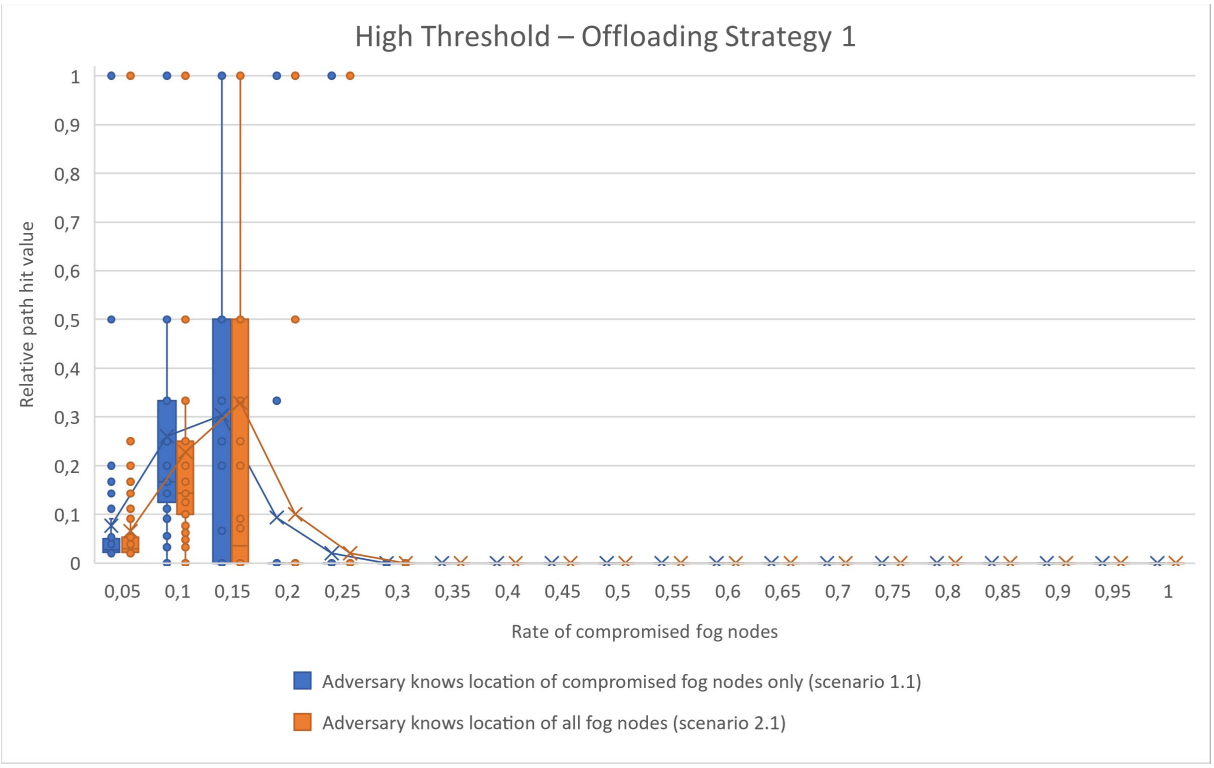


Figure 5-6: Results in terms of the relative path hit value for offloading strategy 1 with a high threshold.

Looking at the relative path hit values one can see that they correlate to the findings in the trace comparison value. At 35% of compromised fog nodes regardless of the offloading threshold the adversary is not able to find any possible path, therefore the relative path hit value is equal to 0.

One can also see that at 15% of compromised fog nodes the adversary has the highest probability in finding the correct path with a distribution from 0 to 0,5 and a mean value of around 0,3. After 15% of compromised fog nodes the relative path hit values also decline analog to the trace comparison values. Prior to a level of 15% of compromised fog nodes the relative path hit value increases with the increasing level of compromised fog nodes. This is due to the fact that with the decreasing number of found possible paths in the trace comparison values the denominator decreases in the calculation of the relative path hit values therefore increasing the adversaries confidence.

Comparing the relative path hit values within the different offloading thresholds one can see that the overall shape of the curve is similar between all thresholds. For 15% of compromised fog nodes the mean values and the distribution are the same between all thresholds. Further at 10% of compromised fog nodes one can see that the distribution and the mean value in scenario 1.1 is higher than in scenario 2.1. Lastly, after 15% of compromised fog nodes the higher the threshold value is, the faster the relative path hit values declines in the direction of 0.

Combining the findings of the relative path hit values and the trace comparison value one can see that an adversary's best possible chance in finding the mobile end-device path is at 15% of compromised fog nodes regardless of the offloading threshold and regardless of the adversary's knowledge i.e., scenario 1.1 or scenario 2.1, due to the trace comparison value being low around 2 found paths and the relative path hit value being around 0,3 mean and 0,5 at the top of the distribution.

5.1.3 Size of Uncertainty Region

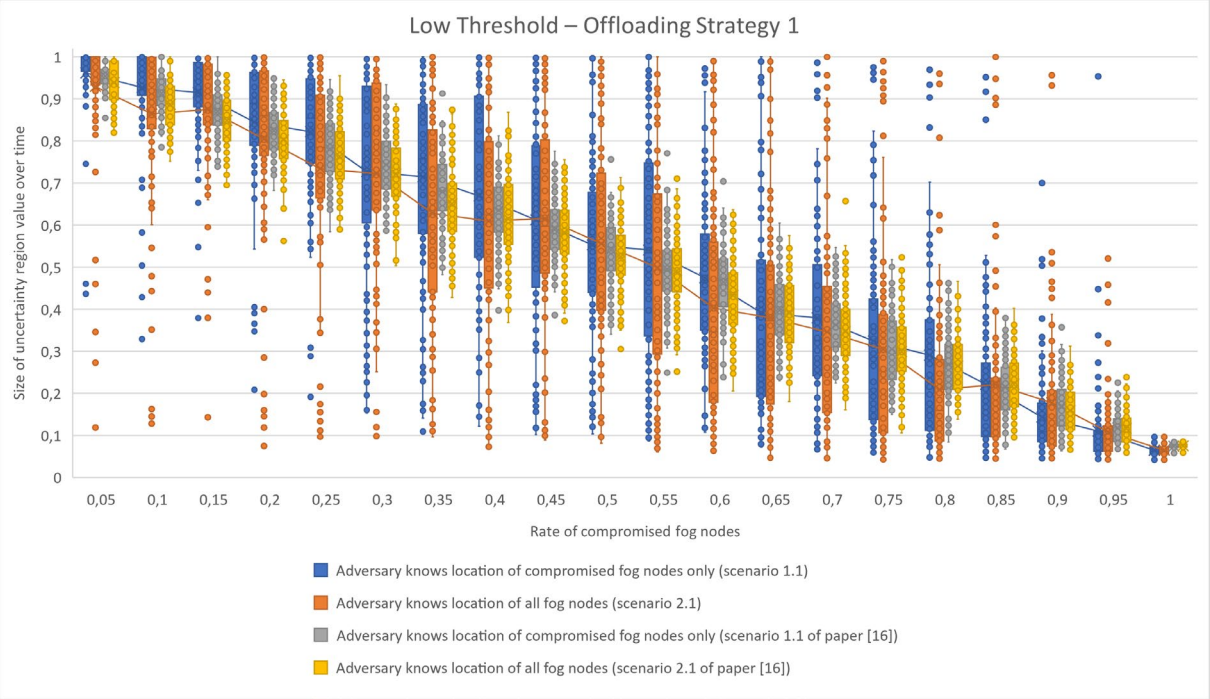


Figure 5-7: Results in terms of the size of the uncertainty region over time for offloading strategy 1 with a low threshold.

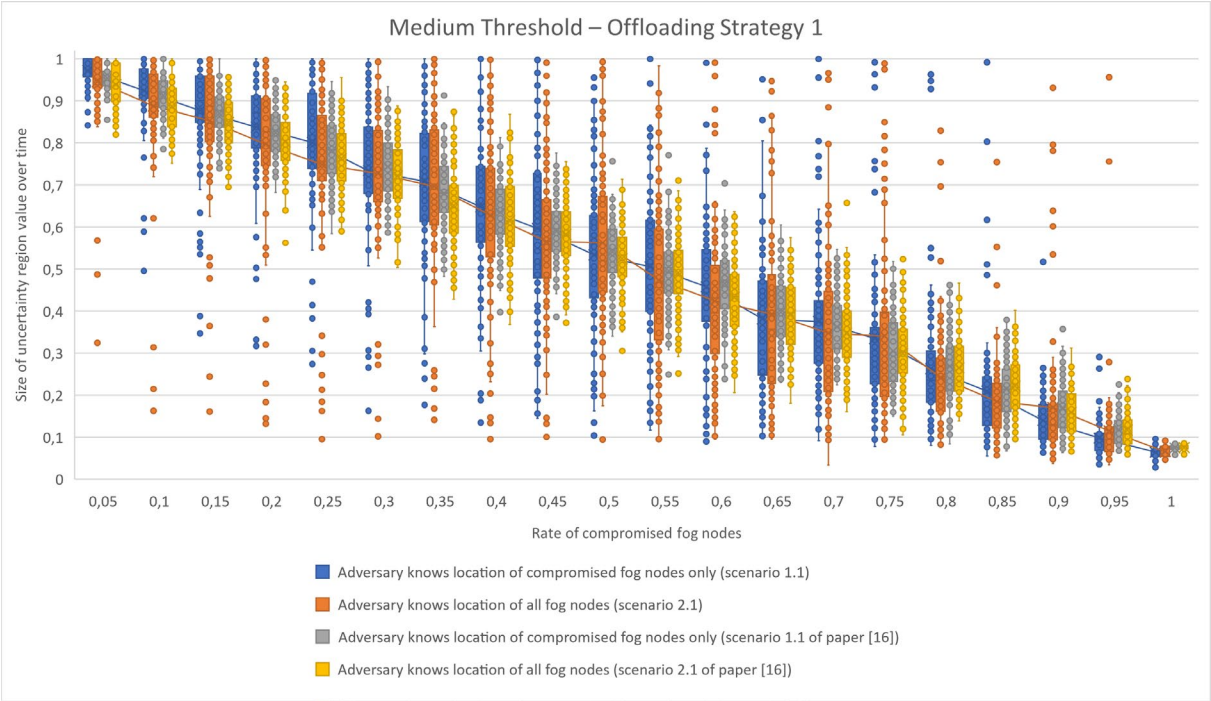


Figure 5-8: Results in terms of the size of the uncertainty region over time for offloading strategy 1 with a medium threshold.

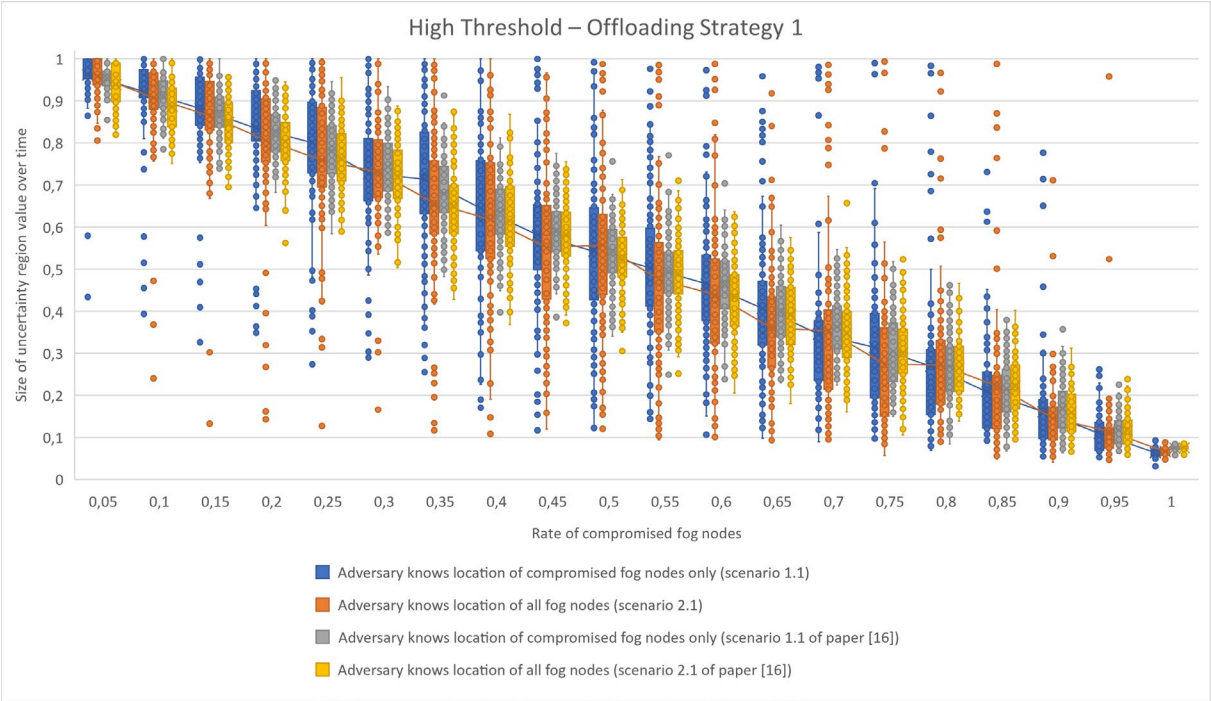


Figure 5-9: Results in terms of the size of the uncertainty region over time for offloading strategy 1 with a high threshold.

While the trace comparison values result in the adversary not being able to find any possible trace for a mobile end-device (i.e., the trace comparison value is equal to 0) with a rate of 35% of compromised fog nodes or more – regardless of the offloading threshold and of the used scenario –, the size of the uncertainty region, however, shrinks with the increasing number of compromised

fog nodes until it reaches a minimum of $\frac{1}{20} = 0{,}05$ at 100% of compromised fog nodes.[15] At 5% of compromised fog nodes the adversary hardly obtains any information about the location of the mobile end-device. This pattern is analog to the findings in the paper.

Comparing the three offloading thresholds with each other one can see that the overall shape of the curve is the same for each threshold. Only a small difference in the scattering of the distribution can be seen in the lowest threshold, where the distribution is broader, as compared to the other thresholds. This is probably due to the fact, that in the lowest offloading threshold the same fog nodes are selected repeatedly by the mobile end-device as offloading target, while in the other offloading thresholds the number of fog nodes that can be selected as offloading target is higher.

Looking at the results of the size of uncertainty region, one would think that as the rate of compromised fog nodes increases the adversary can infer more precise information about the position of the mobile end-device. However, in the scenario of computation offloading the mobile end-device can choose a fog node other than the closest fog node. Therefore, the adversary may be mistaken over the region of the mobile end-device. Hence, this metric must be combined with the area hit duration, to find out to what extent the adversary is wrong.

5.1.4 Area Hit Duration

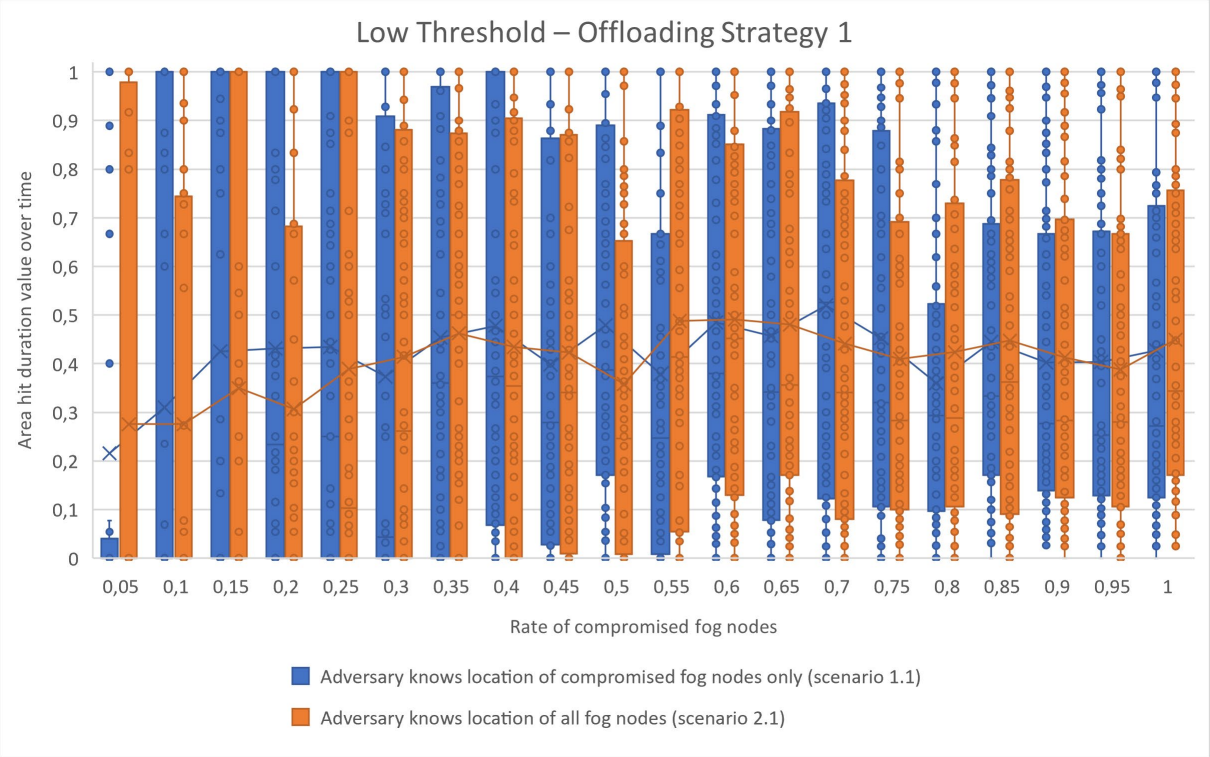


Figure 5-10: Results in terms of the area hit duration value for offloading strategy 1 with a low threshold.

[15] The minimum size of uncertainty region is equals to the region of one fog node, hence it can be calculated by dividing 1 with the number of fog nodes of the region which is in this bachelor's thesis set to 20 fog nodes.

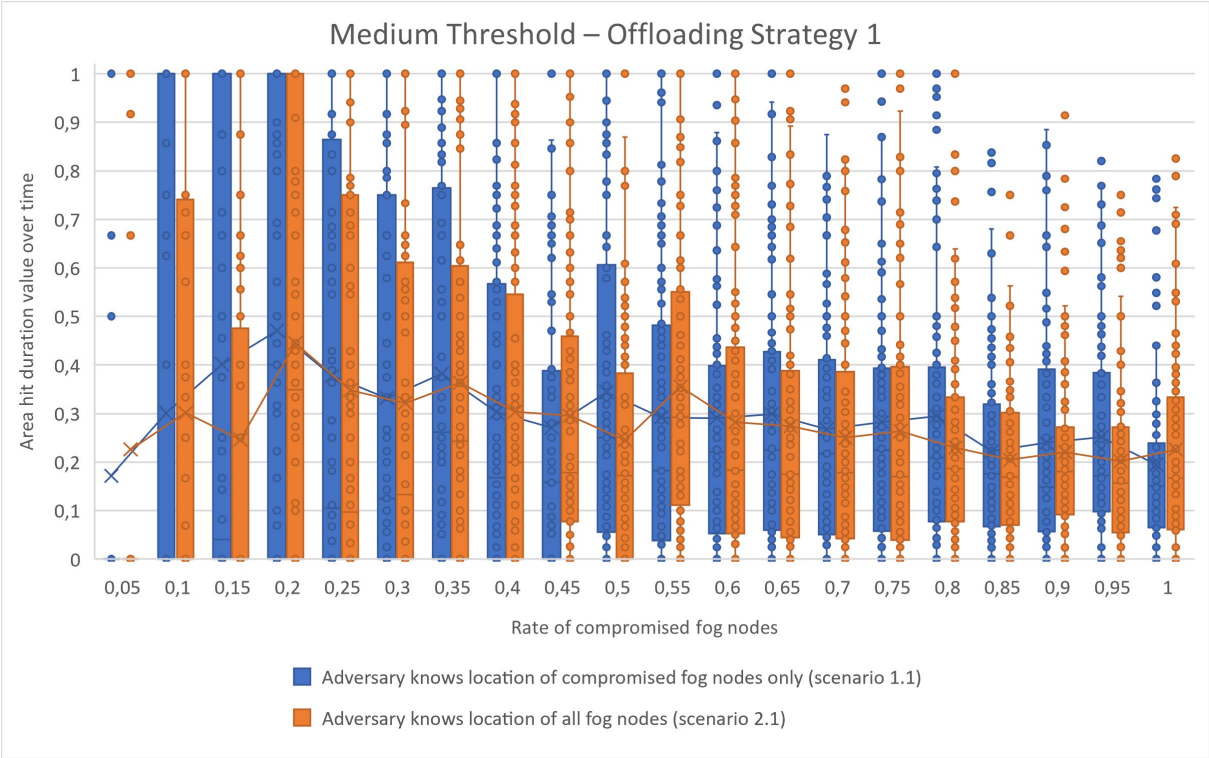


Figure 5-11 : Results in terms of the area hit duration value for offloading strategy 1 with a medium threshold.

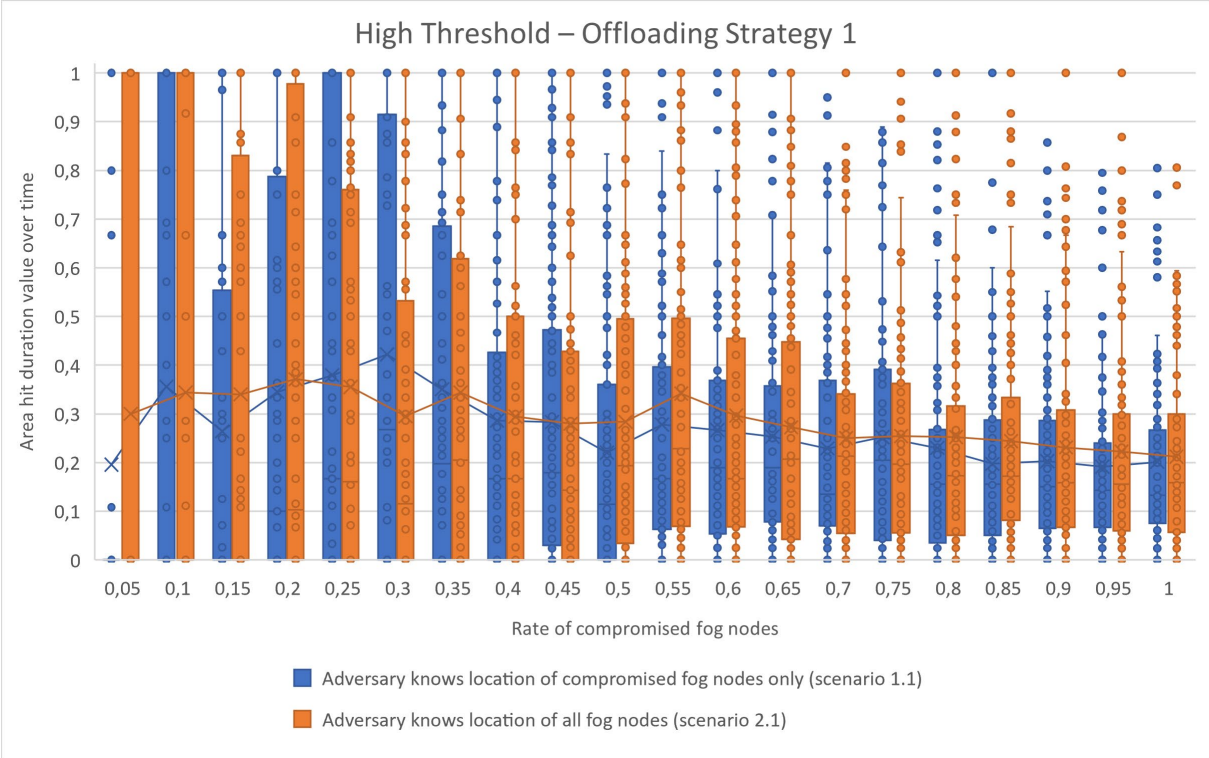


Figure 5-12: Results in terms of the area hit duration value for offloading strategy 1 with a high threshold.

Looking at the area hit duration values one can see that the adversary is only up to a maximum of 50% of the time right about the position of the mobile end-device. Also, one can see that the

difference between the two conducted scenarios is relatively small, and the curves are very similar to each other.

However, regarding the offloading thresholds one can see that the higher the threshold is the lower is the percentage of time where the adversary is right about the position of the mobile end-device in the region. This confirms the hypothesis (for this offloading strategy), that as more fog nodes are available to offload to i.e., the higher the threshold, the adversary will not gain accurate information on the location of the mobile end-device. Contrarily, the lower the threshold, the fewer fog nodes are available in the suitable nodes set for the strategy to choose from. Hence the time the adversary is right is higher.

5.2 Offloading Strategy 2: Offloading to a Fog Node With the Lowest Response Time From a Pre-Filtered Set of Nodes

In this chapter the findings of offloading strategy 2 i.e., offloading to a fog node with the lowest response time from a pre-filtered set of nodes, are compared with the results of the paper [16]. First, the trace comparison values are compared with the findings in [16]. Then, the relative path hit values are correlated with the findings of the trace comparison values. These values however are not compared with [16] as the paper does not include this new metric. Next, the size of uncertainty region is compared with the findings in [16]. Lastly, the area hit duration is correlated with the finding of the size of uncertainty region. These values are also not compared with [16] as the paper also does not include this new metric.

5.2.1 Trace Comparison Value

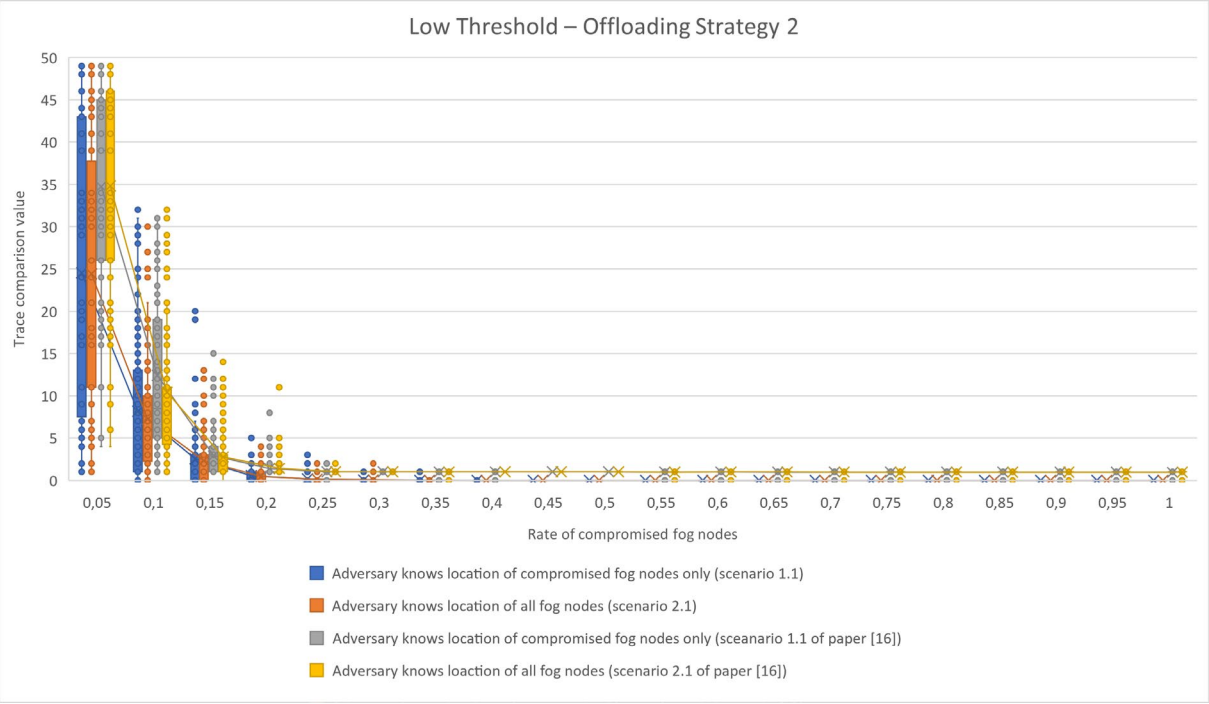


Figure 5-13: Results in terms of the trace comparison value for offloading strategy 2 with a low threshold.

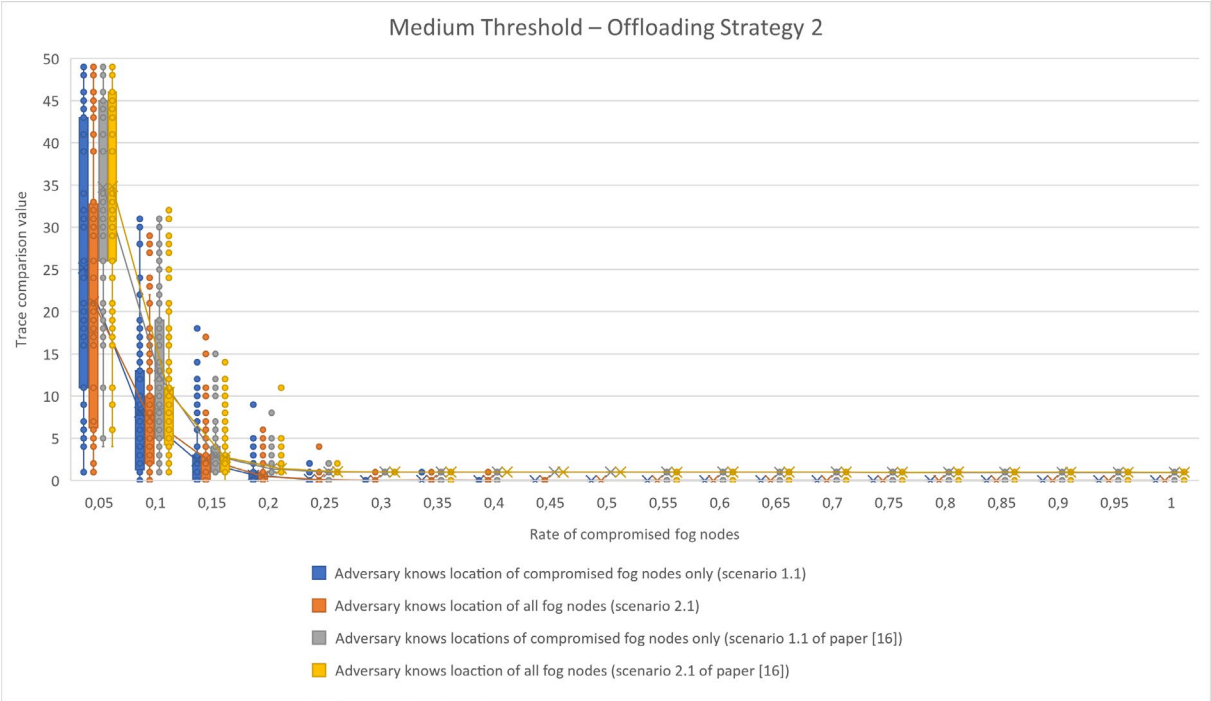


Figure 5-14: Results in terms of the trace comparison value for offloading strategy 2 with a medium threshold.

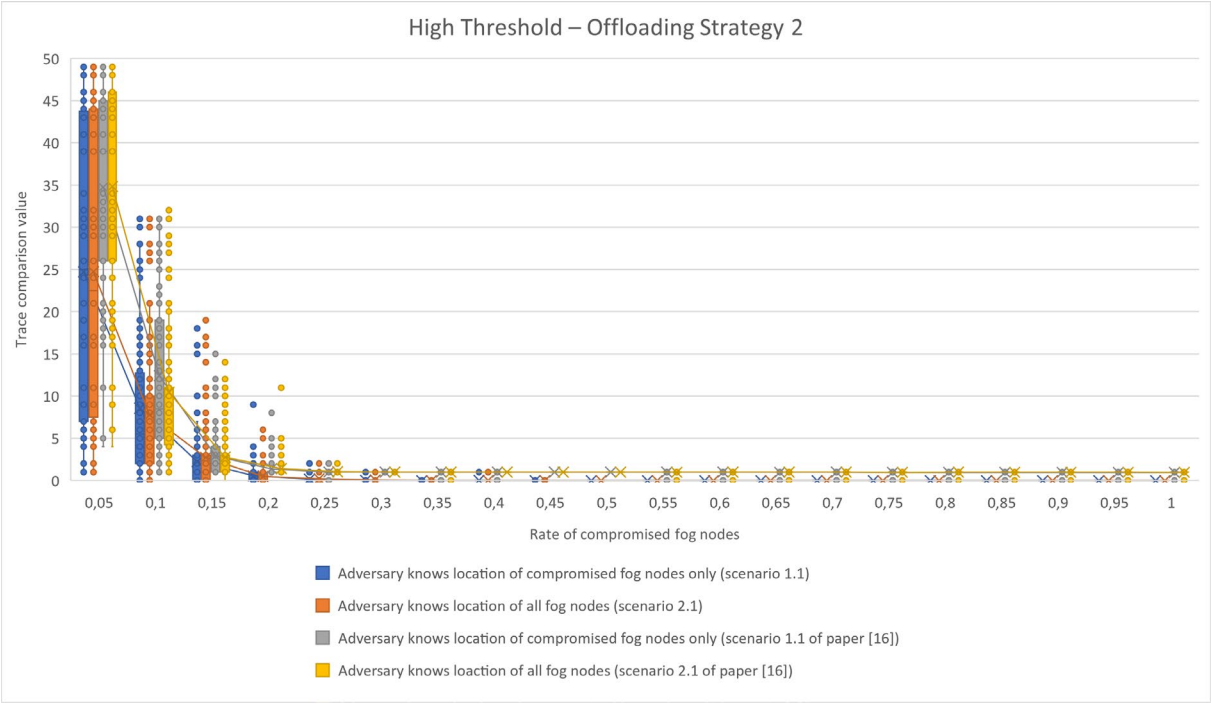


Figure 5-15: Results in terms of the trace comparison value for offloading strategy 2 with a high threshold.

Looking at the trace comparison values of offloading strategy 2 one can see that with this strategy at a ratio of 5% of compromised fog nodes to a ratio of around 35% to 40% of compromised fog nodes the number of possible paths found by the adversary is lower than in the findings of the paper [16]. Having a lower trace comparison number i.e., a lower number of possible paths is beneficial for the adversary. However, as observed in offloading strategy 1, in this

case at a ratio of around 40% of compromised fog nodes the adversary is unable to find any possible traces for a mobile end-device i.e., the trace comparison value is equal to 0. Otherwise, the overall shape of the curves of the paper and this bachelor's thesis are alike.

Comparing the two scenarios, between 5% and 20% of compromised fog nodes, one can see that for this offloading strategy the scenario 2.1 (adversary knows location of all fog nodes) results in a lower overall scattering of the distribution and a lower numbers of possible paths, while in the other scenario the scattering of the distribution is higher, and more paths are found. This is only visible for the first two offloading thresholds, while in the highest offloading threshold the scattering of the distribution of the two scenarios are more alike.

In terms of the offloading thresholds one can see that apart from the different scattering of the distribution (especially at around 5% of compromised fog nodes), the overall curve is the same. Also, as the offloading threshold rises i.e., the more fog nodes are available in the set for the offloading strategy to choose from, the more the upper limit of the scattering of the distribution rise to the same levels as in the paper [16].

In terms of attacks on location privacy for mobile end-device, one can see that considering computation offloading within offloading strategy 2 regardless of the offloading threshold at 40% of compromised fog nodes, the adversary is unable to find the path of the mobile end-device. However, for the values below 40% of compromised fog nodes the adversary finds less possible paths for a mobile end-device compared to offloading strategy 1 and the paper, giving the adversary a slight advantage.

5.2.2 Relative Path Hit Value

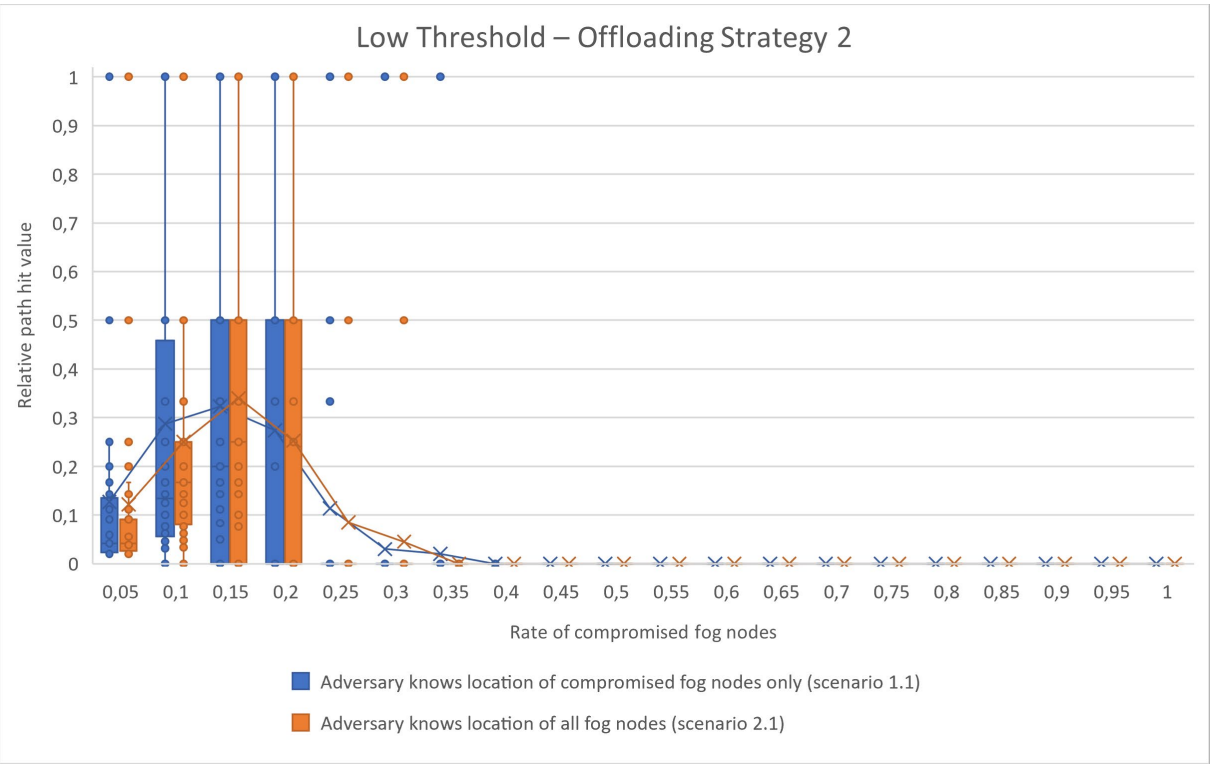


Figure 5-16: Results in terms of the relative path hit value for offloading strategy 2 with a low threshold.

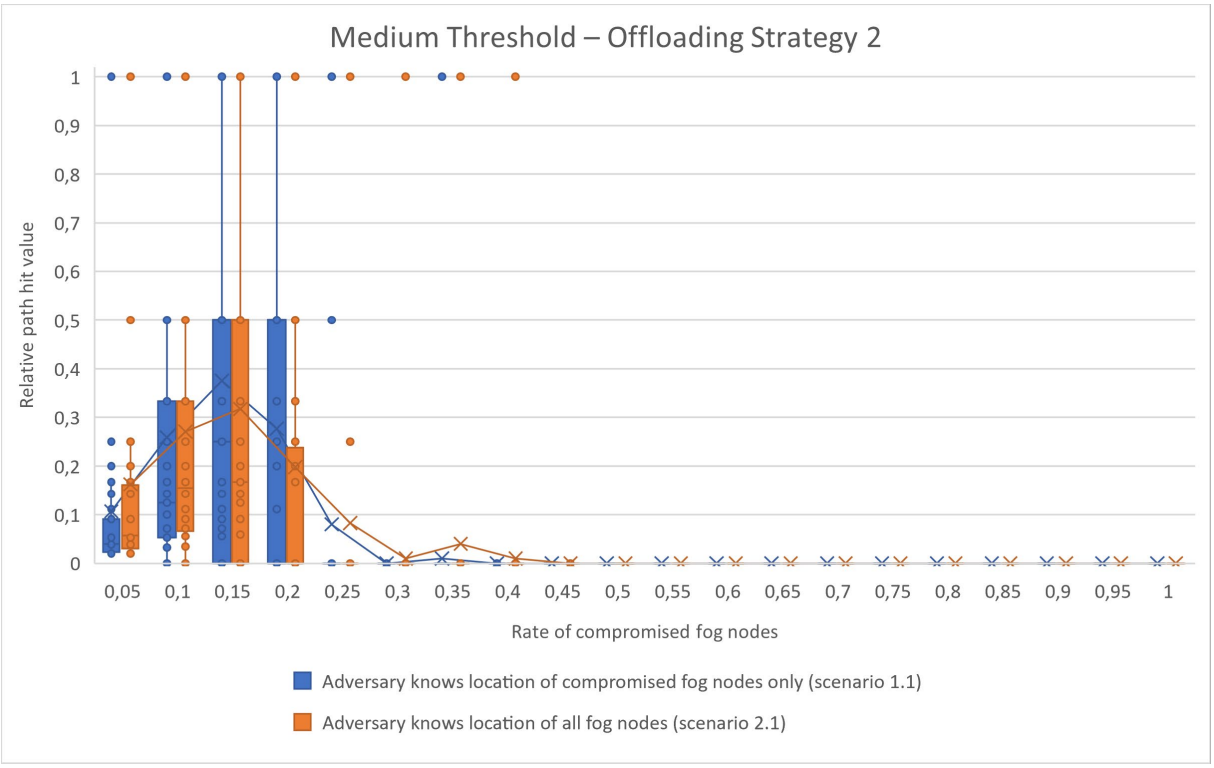


Figure 5-17: Results in terms of the relative path hit value for offloading strategy 2 with a medium threshold.

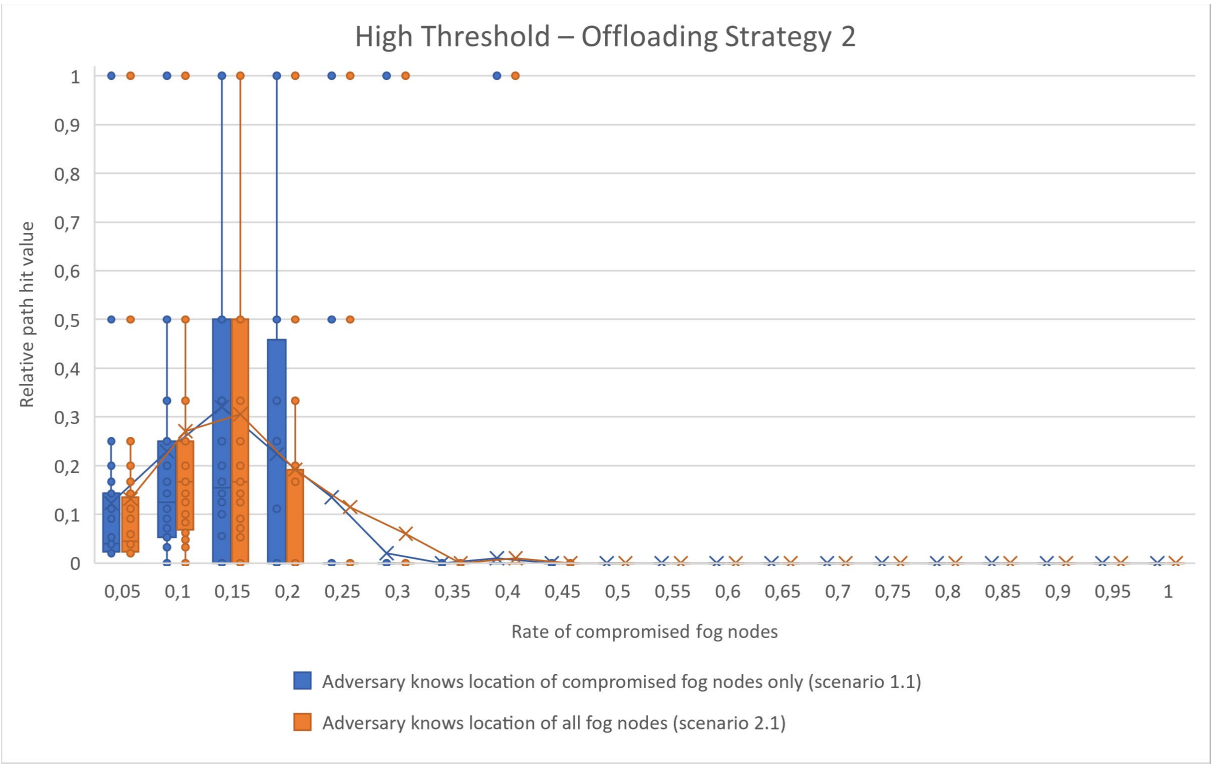


Figure 5-18: Results in terms of the relative path hit value for offloading strategy 2 with a high threshold.

As in offloading strategy 1, with the increasing number of compromised fog nodes, until 15% of compromised fog nodes is reached, the accuracy of the adversary finding the correct path of a mobile end-device increases. This increasing accuracy is due to the decreasing number of possible

paths found by the adversary. At 15% of compromised fog nodes, the value for this accuracy is around the same value (0,3 mean and 0,5 in the highest distribution) as in offloading strategy 1. After 15% of compromised fog nodes the relative path hit values decline until reaching 0 at 40% of compromised fog nodes. This confirms the results found in the trace comparison values of this offloading strategy.

Comparing the relative path hit values within the different offloading thresholds one can see that the overall shape of the curve is similar between all thresholds. For 15% of compromised fog nodes the mean values (equal to 0,3) and the distribution (upper limit is equal to 0,5) are around the same between all thresholds.

Looking at the two scenarios conducted, one can see that regardless of the offloading thresholds used, the overall trend of the graph is the same.

Combining the findings of the relative path hit values and the trace comparison value one can see that an adversary's best possible chance in finding the mobile end-devices path is around 15% to 20% of compromised fog nodes regardless of the offloading threshold and regardless of the adversary's knowledge i.e., scenario 1.1 or scenario 2.1, due to the trace comparison value being low with around 2 found paths and the relative path hit value being around 0,3 mean and 0,5 at the top of the distribution.

5.2.3 Size of Uncertainty Region

Figure 5-19: Results in terms of the size of the uncertainty region over time for offloading strategy 2 with a low threshold.

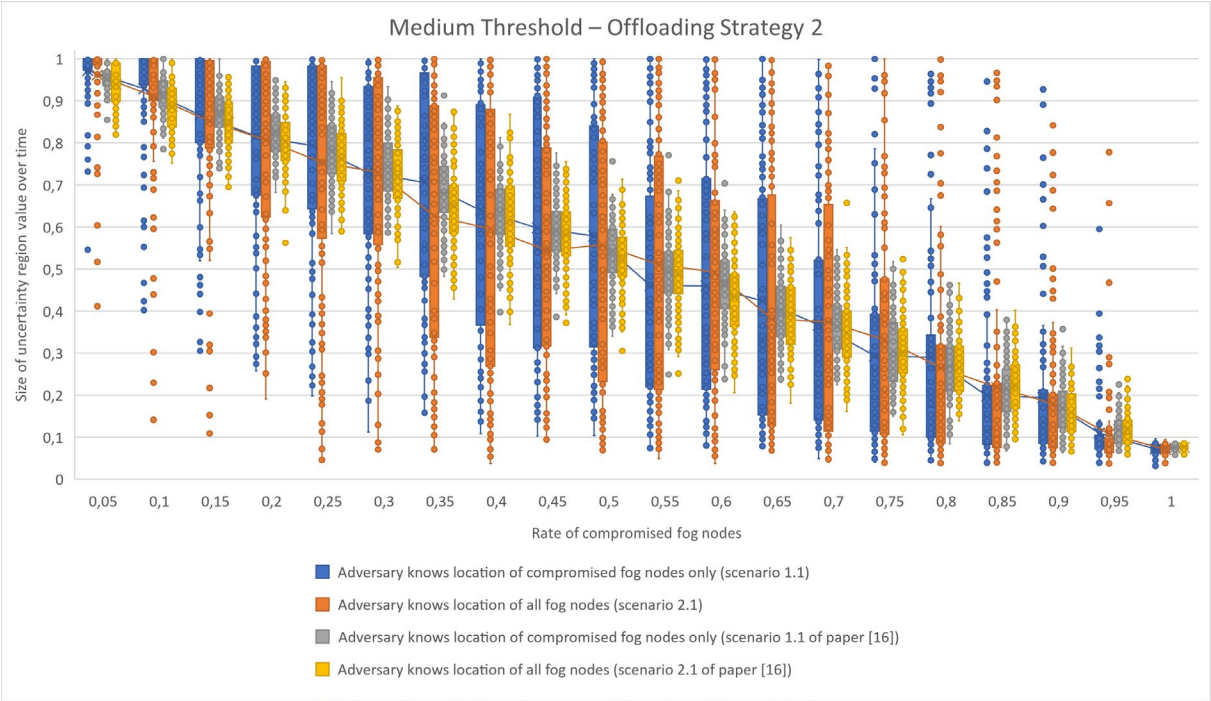


Figure 5-20: Results in terms of the size of the uncertainty region over time for offloading strategy 2 with a medium threshold.

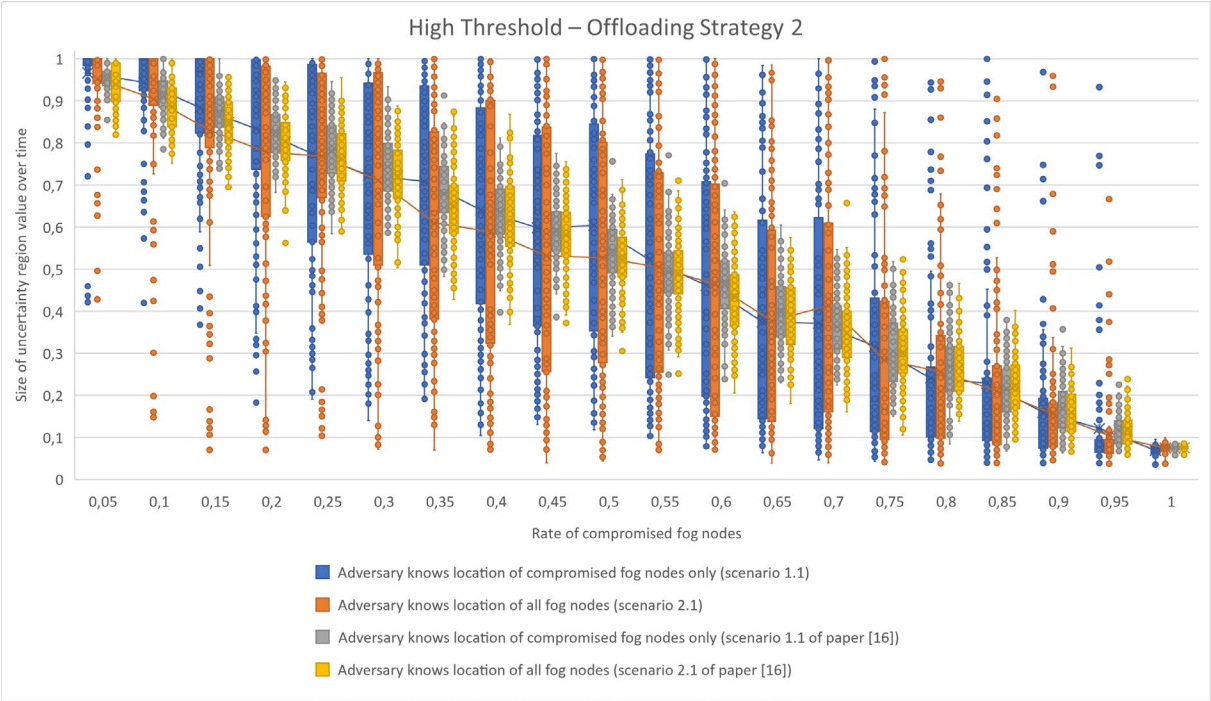


Figure 5-21: Results in terms of the size of the uncertainty region over time for offloading strategy 2 with a high threshold.

Looking at the size of uncertainty region, one can see that regardless of the offloading threshold and of the used scenario, the size of uncertainty region decreases with the increasing number of compromised fog nodes. As in offloading strategy 1, the findings for offloading strategy 2 are very similar. The overall curve of the graph is analog to the curve of the paper or the findings of

offloading strategy 1. Hence, at 100% of compromised fog nodes, the size of uncertainty region also reaches the minimum of $\frac{1}{20} = 0{,}05$. Similarly, at 5% of compromised fog nodes the adversary hardly gains any information about the location of the mobile end-device. The only difference that can be seen between the results of the paper and also the findings in offloading strategy 1 is that the scattering of the distribution is higher.

Comparing the offloading thresholds with one another one can see that the overall patterns of the graphs are the same. As in offloading strategy 1 a difference in the scattering of the distribution between the low offloading threshold and the medium/high offloading threshold could be observed, for offloading strategy 2 these differences in the scattering cannot be observed.

Looking at the two scenarios conducted, again no real difference can be observed.

Lastly, as well as for offloading strategy 1, the results of the size of uncertainty region must be correlated with the results of the area hit duration to gain meaningful information. This is due to the assumption of the adversary that a mobile end-device always connects to the closest fog node. In computation offloading however, a mobile end-device can choose another fog node than the closest. Therefore, the adversary may be mistaken over the region that the mobile end-device is located in. Hence, this metric is combined with the area hit duration, to find out to what extent the adversary is wrong.

5.2.4 Area Hit Duration

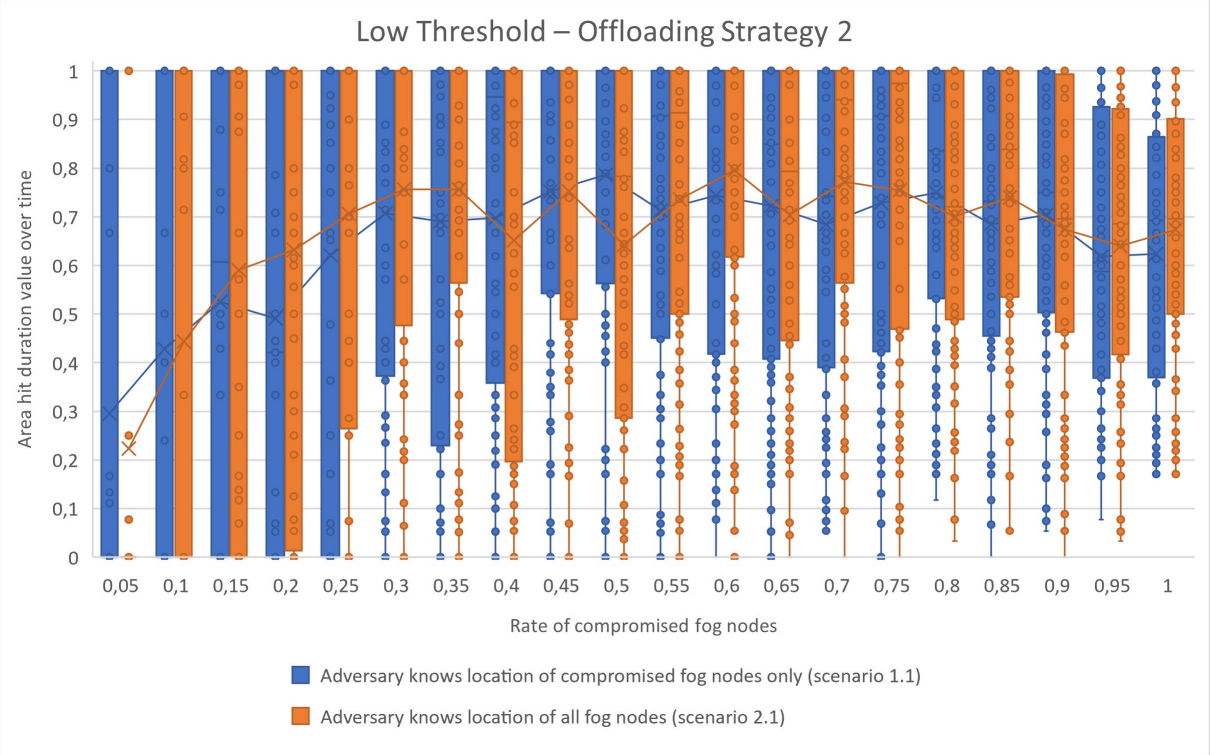


Figure 5-22: Results in terms of the area hit duration value for offloading strategy 2 with a low threshold.

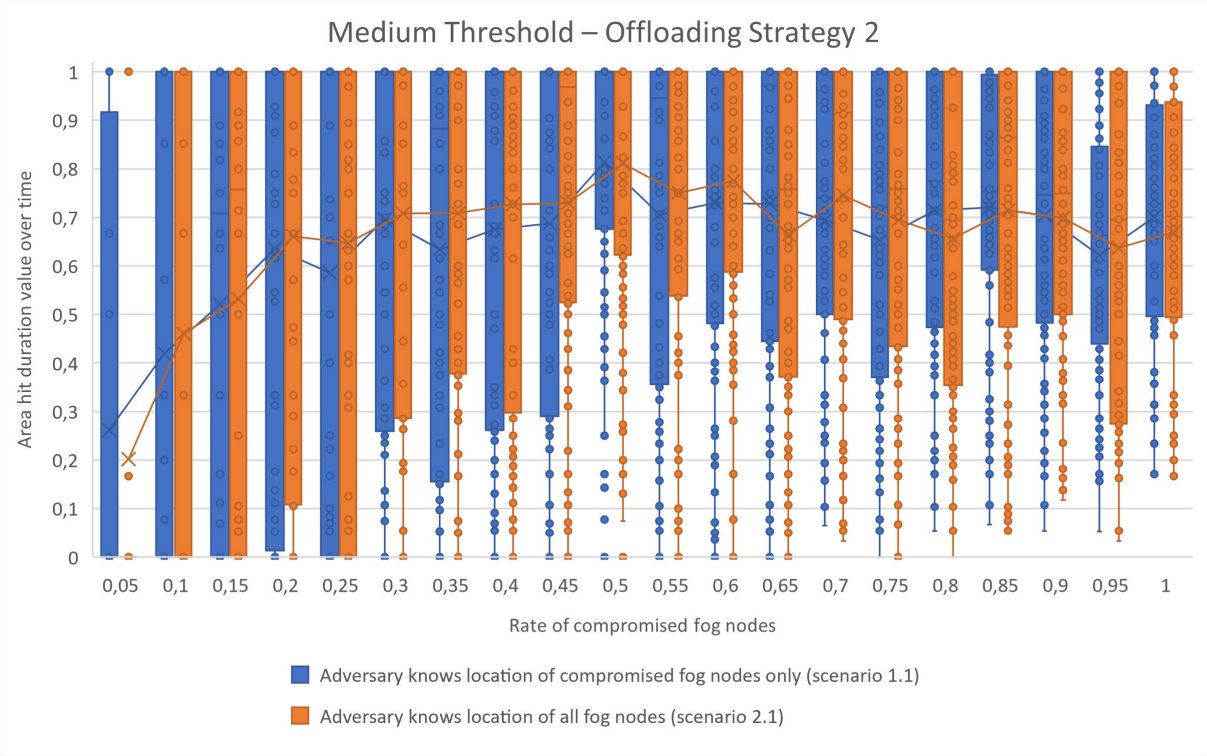


Figure 5-23: Results in terms of the area hit duration value for offloading strategy 2 with a medium threshold.

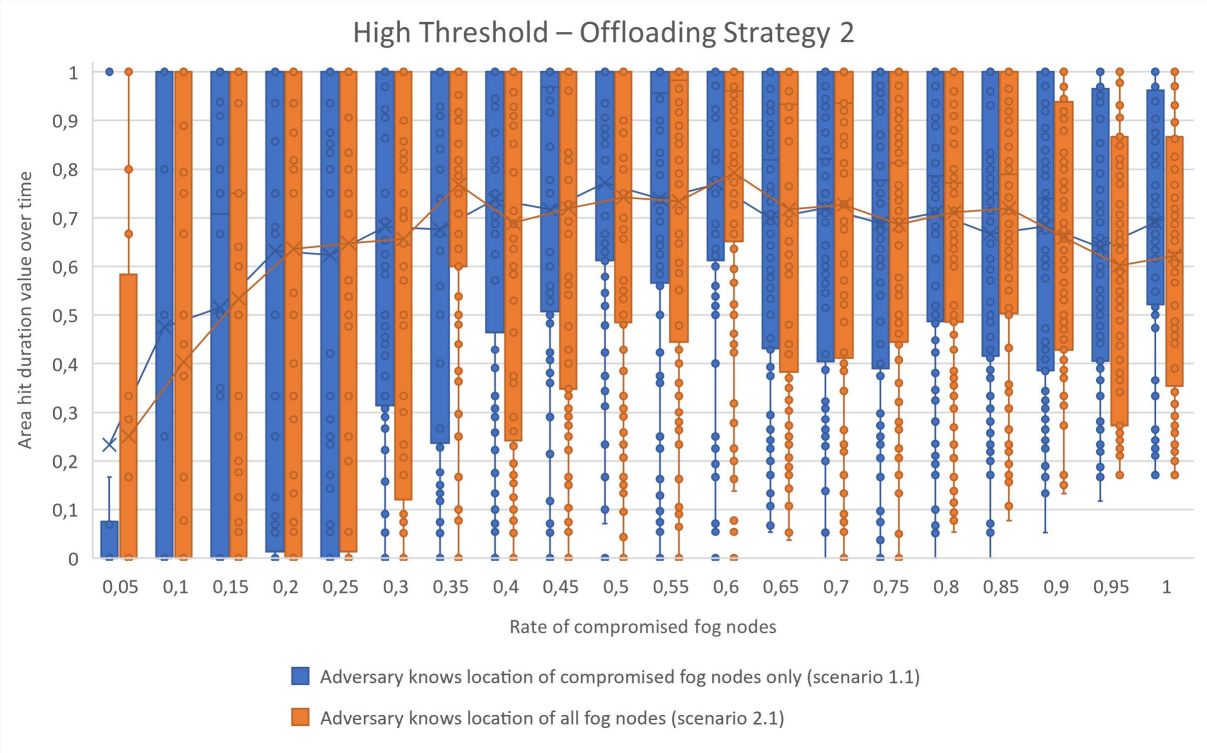


Figure 5-24: Results in terms of the area hit duration value for offloading strategy 2 with a high threshold.

Combining the area hit duration values with the size of uncertainty region one can see that the adversary is up to 70% of the time right about the position of the mobile end-device. One can also see that the area hit duration increases steadily until around 35% of compromised fog nodes, where

it then fluctuates around the 70% with a deviation of about 10%. Also, one can see that the difference between the two conducted scenarios is relatively small, and the curves are very similar to each other.

Regarding the offloading thresholds one can see that the higher the threshold is, the lower are the fluctuations in time where the adversary is right about the position of the mobile end-device in the region. This invalidates the hypothesis (for this offloading strategy), that as more fog nodes are available to offload to i.e., the higher the threshold, the adversary will not gain accurate information on the location of the mobile end-device. For this offloading strategy, the adversary gains at a ration over 35% of compromised fog nodes 70% of the time accurate information of the location of the mobile end-device regardless how high the offloading threshold is. This is due to the offloading strategy not selecting a random fog node, rather selecting the fog node with the lowest response time, which most of the time is the fog node located at the access point to which the mobile end-device is connected to (see figure 4-1). Only if the fog nodes CPU resources at the access point to which the mobile end-device is connected are occupied and thus the response time is higher, other fog nodes with a lower response time are chosen. Choosing another fog node results in the adversary being mistaken, hence the area hit duration is not 100% but only around 70%.

5.3 Discussion

Looking at the results of the simulation conducted in this bachelor thesis one can see that the difference in the two evaluated scenarios for both offloading strategies in all four metrics is small and thus negligible. Even with an increasing offloading threshold the difference between the two scenarios is small, suggesting that the adversary might not need to get all positions of all fog nodes (scenario 2.1) but that having positions of compromised fog nodes only is sufficient (scenario 1.1). This can be explained through the way the two metrics (trace comparison value and size of uncertainty region) are computed: for the computation of the trace comparison value only compromised fog nodes are used in the process of trace comparison, while all other either unknown fog nodes i.e., a non-compromised fog node, or periods in which the mobile end-device is not connected to the fog network, are replaced with a placeholder (‘*’) and are therefore ignored; For the computation of the size of uncertainty region, at points in time where the mobile end-device is connected to a non-compromised fog node, the adversary can only narrow down the size of uncertainty region to the area of the non-compromised fog nodes, which is the same area regardless whether the adversary knows the position of all fog nodes or not. In the paper by *T. Wettig* and *Z.Á. Mann* [16] the same observations are found and are further explained. As conducted in the paper [16] and in the bachelor thesis by *T. Wettig* [17], the adversary’s knowledge about the positions of the fog nodes is divided into two categories: either the adversary knows the positions of all fog nodes (scenario 2.1), or the positions of only compromised fog nodes only (scenario 1.1). In reality, the knowledge of the adversary about the positions of the fog nodes may be somewhere between the two extremes expressed by the scenarios examined here.

Evaluating the accuracy on the trace comparison values one can see that with computation offloading the initial number of found traces, up to 30% of compromised fog nodes for offloading strategy 1 and 35% of compromised fog nodes for offloading strategy 2, is lower than the number of found traces in the paper by *T. Wettig* and *Z.Á. Mann* [16]. Within the first 15% to 20% percent of compromised fog nodes the adversary gains the most accurate information on the traces. This is probably due to the fact that in computation offloading the mobile end-device is not always choosing the closest fog node, hence several traces are omitted, which in return results in a lower number of found traces compared to the findings in the paper [16]. However, it should be noted that the number of found traces is still quite high i.e., around 30 traces for 5% of compromised fog nodes and over three or more found traces up to 15% of compromised fog nodes, and thus one could argue that the data gained by the adversary is not precise. It should also be noted that at a ratio of 30% of compromised fog nodes in the offloading strategy 1 and 35% of compromised fog nodes in offloading strategy 2, the adversary is unable to find any traces for a mobile end-device i.e., the trace comparison value is equal to 0. Contrary, in the results of the paper the adversary is able to narrow down the number of paths to exactly 1 path, which is the path the mobile end-device took. Hence, in the scenario of computation offloading regardless of the two evaluated offloading strategies one can see that for over 30% of compromised fog nodes the adversary cannot obtain any information about the path of the mobile end-device. This is due to the fact that with the increasing number of compromised fog nodes the adversary, who assumes that mobile end-device always connects to the closest fog node, is more and more irritated about the position of the mobile end-device, as the collected trace events do no longer match the observed traces.

As the trace comparison value is based on the assumption of the adversary that mobile end-device always connects to the closest fog node, the trace comparison value is combined with a metric describing the level of misconception of the adversary. Therefore, the relative path hit values are included into the observations for the trace comparison. Looking at the relative path hit values, one can see that within the first few percentages (up to around 30% to 35% of compromised fog nodes) although the adversary is able to find less paths than in the paper, only at 15% of compromised fog nodes 50% of the time the found paths included the actual path of the mobile end-device. This means the optimal number of compromised fog nodes for the adversary trying to narrow down the traces of a mobile end-device is 15% for both offloading strategies regardless of the offloading threshold.

Looking at the size of uncertainty region the behaviour regarding the ratio of compromised fog nodes is different to the behaviour for the trace comparison values. With the increasing number of compromised fog nodes the size of uncertainty region decreases. This is analogue to the findings of the paper. Furthermore, in the scenario of computation offloading the two evaluated offloading strategies and their offloading threshold hardly have any influence on the size of the uncertainty region. However, as the metric is based on the adversary's assumption of mobile end-devices always connecting to the closest fog node, this metric is expanded by the area hit duration capturing to what degree the adversary is mistaken about the real location of the mobile end-device.

The higher the area hit duration is the more accurate is the adversary about the size of uncertainty region.

Combining the area hit duration with the size of uncertainty region one can see that for the offloading strategy 1 the size of uncertainty region is below 40% of the time right and slightly decreasing with an increasing number of compromised fog nodes. Furthermore, with an increasing offloading threshold the area hit duration declines with an increasing number of compromised fog nodes. This is reasonable as due to the higher threshold more fog nodes are available in the suitable fog nodes set for the offloading strategy to choose from. As offloading strategy 1 in that case chooses a random fog node, the adversary is more often mistaken about the position of a mobile end-device. For offloading strategy 2 the accuracy gained by the adversary is higher at around 70% to 80%. Also, as the offloading threshold increases the deviations in the distribution decline. This is a result of this offloading strategy selecting the fog node with the lowest threshold from the available set of fog nodes. As one can see in figure 4-1, the fog node closest to the access point has the lowest response time if the CPU resources are not occupied. A mobile end-device connecting to an access point and offloading a task to a remote fog node therefore will choose most of the time the fog node directly located at the access point i.e., closest to the mobile end-device. Only if the CPU resources of that fog node are occupied and thus the response time is higher other fog nodes, located farther away, are chosen as offloading target. For offloading strategy 2 the adversary is therefore more able to narrow down the region of the mobile end-device as compared to offloading strategy 1.

6 Conclusion

In this bachelor's thesis the *LocPrivFogSim* simulator was extended to support offloading task for mobile end-devices in the evaluation of attacks on location privacy. Therefore, three sub-goals were pursued. The first sub-goal was the extension of *LocPrivFogSim* to enable selecting a fog node based on its response time rather than its closeness to the mobile end-device, section 3.2. The second sub-goal was the implementation of two offloading strategies to select a fog node from a set of possible nodes i.e., nodes that have a response time below a certain threshold. For one, a random node should be selected from the set (section 3.4), secondly the fog node with the lowest response time should be selected (section 3.5). The third sub-goal was the extension and repetition of the conducted experiments in the paper by *T. Wettig* and *Z.Á. Mann* [16] to cover the newly implemented offloading strategies (chapter 4). The results were compared to the results of the paper, especially regarding the impact of computation offloading strategies on the accuracy of location privacy attacks (chapter 5).

Looking at the results, it can be said that in the case of computation offloading on location privacy attacks as long as the ratio of compromised fog nodes is above 35% the adversary is unable to get any information about the trace of a mobile end-device. This is due to the way the trace comparison value is computed and the fact that the adversary assumes a mobile end-device always connects to the closes fog node, which is not the case in computation offloading. Below the 35% of compromised fog nodes the adversary gains information about traces but cannot narrow the number of traces down to 1 i.e., the exact trace of the mobile end-device. For the trace comparison values these results are independent of the offloading strategy or the offloading threshold. In the case of the size of uncertainty region it can be said that with offloading strategy 2 the adversary is able to narrow down the region of the mobile end-device with about 80% accuracy, while for offloading strategy 1 the adversary is only able to narrow down the region to a degree below 50% accuracy. This is due to offloading strategy 1 selecting a random fog node of the set of possible fog nodes, while offloading strategy selects the fog node with the lowest response time. For offloading strategy 2 looking at figure 4-1 one can see that this mostly results in selecting the closest fog node to the access point to which the mobile end-device is connected to.

The results of this bachelor's thesis are based on the assumptions made and described in chapter 3.1 – Assumptions and Limitations. In reality, many other factors might influence the accuracy of the adversary. In the following paragraphs, four ideas that emerged during the implementation and elaboration of this bachelor's thesis are presented. Those ideas could be used as potential next steps in further evaluation computation offloading in fog computing on location privacy attacks:

1. In this bachelor's thesis it was assumed that an offloading task started on a selected fog node is also completed on that fog node, although the mobile end-device might have moved into another region where another fog node might be better suited. As implemented in *MobFogSim* for VM migrations, one could imagine a migration and handoff system for offloading tasks, whereas the mobile end-device is moving through the region, the target fog node will be checked if it is still suitable for the offloading

task, or whether another fog node is more suitable, thus migrating the task to the next fog node. The impact of this task migration then had to be measured whether it positively affects the knowledge the adversary gains. One could assume that the knowledge the adversary gains is more accurate if the offloading tasks moves along with the mobile end-device through the region.

2. In this bachelor's thesis the offloading task scheduler was implemented as a fixed offloading task scheduler scheduling every 1000 ticks an offloading task with the same resource requirements. In reality, the data of an offloading tasks would probably change from offloading request to offloading request. It therefore could be investigated what impact the frequency of scheduling offloading tasks has on the accuracy an adversary gains. Also, it could be investigated whether different offloading tasks resource requirements have any effects on the adversary's accuracy. In a nutshell, one could investigate whether many short offloading tasks or fewer but longer offloading tasks are more favourable for an adversary.
3. The calculated response time in this bachelor's thesis is assumed to be the actual response time of the offloaded task. In a real-world network however, many different aspects influence the transmission and execution of an offloading task so that the response time calculated by a mobile end-device might not be true. It therefore could be investigated what influence a delayed or even a not completed task has.
4. The information gathered by an adversary could be enriched by metadata of the fog network. For example, if the adversary is able to create a matrix with the average response times of fog nodes by using one of its own mobile end-devices for testing and under the assumption that the adversary knows the used offloading strategy, the adversary could use this information to backtrace which access points the mobile end-device was connected to.

References

[1] V. V. Arutyunov, "Cloud computing: Its history of development, modern state, and future considerations," *Scientific and Technical Information Processing,* vol. 39, pp. 173-178, 2012.

[2] Y. Jadeja, K. Modi, "Cloud Computing - Concepts, Architecture and Challenges," in *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, Nagercoil, India, 2012.

[3] A. Brogi, S. Forti, C. Guerrero, I. Lera, "How to place your apps in the fog: State of the art and open challenges," *Software: Practice and Experience,* vol. 50, no. 5, pp. 719-740, 2019.

[4] M. Taneja, A. Davy, "Resource Aware Placement of IoT Application Modules in Fog-Cloud Computing Paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, 2017.

[5] M. Sudip, S. Subhadeep, "Theoretical modelling of fog computing: A green computing paradigm to support IoT applications," *IET Networks,* vol. 5, pp. 23-29, 2016.

[6] Cisco, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are," 2015. [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf. [Accessed May 2021].

[7] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments," *Software: Practice and Experience,* vol. 47, no. 9, pp. 1275-1296, 2017.

[8] L. M. Vaquero, L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing.," *ACM SIGCOMM Computer Communication Review,* vol. 44, no. 5, pp. 27-32, 2014.

[9] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing (MCC '12)*, Helsinki, Finland, 2012.

[10] S. Yi, Z. Qin, Q. Li, "Security and Privacy Issues of Fog Computing: A Survey," in *International Conference on Wireless Algorithms, Systems and Applications (WASA}*, Springer International Publishing, 2015, pp. 685-695.

[11] A. Shakarami, M. Ghobaei-Arani, M. Masdari, M. Hosseinzadeh, „A survey on the computation offloading approaches in mobile edge/cloud computing environment: a stochastic-based perspective,“ *Journal of Grid Computing,* Bd. 18, Nr. 4, pp. 639-671, 2020.

[12] R. Dadmehr, N. Mohsen, „Computation Offloading and Scheduling in Edge-Fog Cloud Computing,“ *Journal of Electronic & Information Systems,* Bd. 1, Nr. 1, 2019.

[13] A. Alrawais, A. Alhothaily, C. Hu, X. Cheng, "Fog computing for the Internet of Things: Security and privacy issues," *IEEE Internet Computing,* vol. 21, no. 2, pp. 34-42, 2017.

[14] Y. Guan, J. Shao, G. Wei, M. Xie, "Data security and privacy in fog computing," *IEEE Network,* vol. 32, no. 5, pp. 106-111, 2018.

[15] Z. Á. Mann, A. Metzger, K. Pohl, „Situativer Datenschutz im Fog-Computing,“ *Informatik Spektrum,* Bd. 42, Nr. 4, pp. 236-243, 2019.

[16] T. Wettig, Z. Á. Mann, "Simulation-based analysis of threats to location privacy in fog computing," in *Fifth International Workshop on Security, Privacy and Trust in the Internet of Things (SPT-IoT)*, Kassel, 2021.

[17] T. Wettig, „Erweiterung des Simulators MobFogSim zur Simulation von Location-Privacy-Angriffen auf Fog-Computing-Systeme,“ *Universität Duisburg-Essen,* 2020.

[18] J. Zhang, X. Hu, Z. Ning, E. C. Ngai, L. Zhou, J. Wei, J. Cheng, B. Hu, „Energy-Latency Tradeoff for Energy-Aware Offloading in Mobile Edge Computing Networks,“ *IEEE Internet of Things Journal,* Bd. 5, Nr. 4, pp. 2633,2645, 2018.

[19] OpenFog Consortium Architecture Working Group, "OpenFog Reference Architecture for Fog Computing," February 2017. [Online]. Available: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf. [Accessed May 2021].

[20] M. Iorga, L. B. Feldman, R. Barton, M. J. Martin, N. S. Goren, C. Mahmoudi, "Fog Computing Conceptual Model," Special Publication (NIST SP) 500-325, 2018.

[21] R. Mahmud, R. Kotagiri, R. Buyya, "Fog Computing: A Taxonomy, Survey and Future Directions," in *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, Singapore, Springer, 2018, pp. 103-130.

[22] R. Buyya, S. N. Srirama, Fog and Edge Computing: Principles and Paradigms, John Wiley & Sons, Inc., 2019.

[23] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, D. S. Nikolopoulos, "Challenges and Opportunities in Edge Computing," in *IEEE International Conference on Smart Cloud (SmartCloud)*, New York, NY, USA, 2016.

[24] J. Zhang, B. Chen, Y. Zhao, X. Cheng, F. Hu, "Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues," *IEEE Access,* vol. 6, pp. 18209-18237, 2018.

[25] REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL, *on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation),* 27 April 2016.

[26] P. Asuquo, H. Cruickshank, J. Morley, C. P. A. Ogah, A. Lei, W. Hathal, S. Bao, Z. Sun, "Security and Privacy in Location-Based Services for Vehicular and Mobile Communications: An Overview, Challenges, and Countermeasures," *IEEE Internet of Things Journal,* vol. 5, no. 6, pp. 4778-4802, 2018.

[27] C. Huang, R. Lu, K.-K.R. Choo, „Vehicular Fog Computing: Architecture, Use Case, and Security and Forensic Challenges," *IEEE Communications Magazine,* Bd. 55, Nr. 11, pp. 105-111, 2017.

[28] Q. Wang, T. Ji, Y. Guo, L. Yu, X. Chen, P. Li, „TrafficChain: A Blockchain-Based Secure and Privacy-Preserving Traffic Map," *IEEE Access,* Bd. 8, pp. 60598-60612, 2020.

[29] C. Puliafito, D. M. Gonçalves, M. M. Lopes, L. L. Martins, E. Madeira, E. Mingozzi, O. Rana, L. F. Bittencourt, „MobFogSim: Simulation of mobility and migration for fog computing," *Simulation Modelling Practice and Theory,* Bd. 101, Nr. 1569-190X, p. 102062, 2020.

Eidesstattliche Erklärung

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Essen, am 1. August 2021