Karlsruhe Institute of Technology

Evaluating a network address shuffling approach as DDoS protection for Edge Clouds

Bachelor's Thesis of

Julian Todt

at the Department of Informatics
Institute for Telematics (TM)
Chair for IT Security Methods and Systems

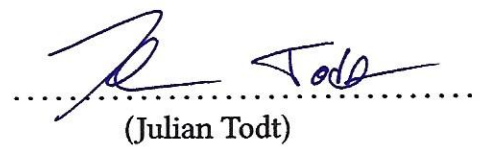
Reviewer:	Prof. Dr. Thorsten Strufe
Second reviewer:	Jun.-Prof. Dr. Christian Wressnegger
Advisor:	M. Sc. Simon Hanisch

24th February 2020 – 20th July 2020

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 20. Juli 2020



(Julian Todt)

Abstract

Edge clouds which are a new form of cloud on the edge of the internet, require new lightweight mechanisms to mitigate DDoS attacks. Network address shuffling approaches like Mole Hunt and MOTAG that mitigate attacks by isolating attackers are being proposed but their effectiveness is unclear. To assess this, an evaluation framework is designed, implemented and used to create realistic and in-depth analysis on these moving target defenses as well as compare different approaches.

I define requirements for the framework based on realistic assumptions about edge clouds and their attackers, especially their resources. The framework is designed to set up a model of an edge cloud infrastructure environment including attackers, clients, routers and the edge cloud itself. During an evaluation an exchangeable attack pattern directs the attackers to send realistic attack traffic to the edge cloud to which the clients are connected while the framework collects metrics on the mitigation's performance. Metrics include technical aspects like the number of required rounds and used addresses as well as quality of service statistics.

The results show that while the tested moving target defenses are generally able to defend edge clouds against DDoS attacks, they struggle against sophisticated attack patterns. In comparison, MOTAG's and Mole Hunt's algorithm perform similarly, but Mole Hunt's address-announcing technology performs better. Concluding, network address shuffling approaches as DDoS protection for edge clouds are promising but still have room for improvements.

Zusammenfassung

Edge Clouds sind eine neue Form der Cloud am Rand des Internets, welche neue ressourcenschonende DDoS Verteidigungsstrategien benötigen. Es werden „network address shuffling“-Ansätze wie Mole Hunt und MOTAG vorgeschlagen, die Angriffe abschwächen indem sie Angreifer identifizieren, aber die Effektivität dieser Ansätze ist unklar. Um dies zu beurteilen, wird ein Evaluationsframework entworfen, implementiert und benutzt, um realistische und tiefgründige Analysen dieser Verteidigungsstrategien zu erstellen und verschiedene Ansätze zu vergleichen.

Die Anforderungen an dieses Framework werden auf Basis realistischer Annahmen über Edge Clouds und ihrer Angreifer definiert, insbesondere deren Ressourcen. Das Framework ist dazu konzipiert, ein Modell einer Edge Cloud Infrastruktur aufzusetzen, die Angreifer, Klienten, Router und die Edge Cloud selbst beinhaltet. Während einer Evaluation steuert ein austauschbares Angriffsschema die Angreifer, die realistischen Angriffstraffic an die Edge Cloud schicken mit der die Klienten verbunden sind. Währenddessen misst das Framework Metriken über die Leistung der Verteidigungsstrategie. Diese Metriken beinhalten technische Aspekte wie die Anzahl an benötigten Runden und benutzten Adressen sowie Servicequalitätsaspekte.

Die Ergebnisse zeigen, dass während die getesteten Verteidigungsstrategien grundsätzlich Edge Clouds gegen DDoS Angriffe verteidigen können, diese Probleme mit ausgeklügelten Angriffsschemen haben. Im Vergleich zeigt sich, dass die Algorithmen von MOTAG und Mole Hunt ähnlich gute Ergebnisse zeigen, jedoch Mole Hunt’s Adressankündigungstechnologie zu besseren Ergebnissen führt. Zusammenfassend sind „network address shuffling“-Ansätze vielversprechende DDoS Verteidigungsstrategien für Edge Clouds, die jedoch noch Raum für Verbesserungen haben.

Contents

List of Figures

List of Tables

1. Introduction

Edge clouds are a new form of distributed cloud on the edge of the internet, designed to reduce latency to users and bandwidth to the core of the internet. Edge cloud nodes are small compute nodes that are physically distributed and are therefore closer to end users. This results in reduced latency which enables services requiring low latency such as cloud gaming [9]. Distributed denial-of-service (DDoS) attacks continue to pose severe threat to today's internet services as their intensity and sophistication increases [30]. Today's datacenter-based general purpose clouds defend against DDoS attacks by running sophisticated DDoS attack prevention, detection and mitigation services on their substantial computing power as well as simply outscale most attackers. Edge clouds, which are destined for rollout together with 5G in the coming years, will not have the required computing power to defend against DDoS attacks like datacenter clouds do. This means that edge clouds cannot run those DDoS defense mechanisms which would leave them undefended and prone to DDoS attacks. To protect edge clouds with their limited computing power from DDoS attacks, new lightweight mechanisms to mitigate DDoS attacks on edge clouds are being proposed.

One of these mechanisms is *Mole Hunt* [12], a moving target defense. Mole Hunt mitigates DDoS attacks by isolating attackers. Whenever the edge cloud is attacked, Mole Hunt shuffles the network addresses and assigns clients a new addresses to use. After a limited number of shuffles, the sequence of attacked addresses is unique and the attacker is identified. While Mole Hunt seems promising in first theoretical evaluations, in-depth and more practical evaluations are needed to determine how Mole Hunt compares against other moving target defenses and whether Mole Hunt is suitable to defend edge clouds against DDoS attacks.

In this Bachelor's thesis, I will create an evaluation framework for moving target defenses as DDoS attack mitigations in form of a virtual prototype that simulates attacks. The framework will be based on realistic expectations for edge clouds and DDoS attackers and will include multiple attack patterns. By implementing both Mole Hunt and MOTAG, another moving target defense, on this framework I will be able to compare the two mitigations in a realistic and practical environment. Additionally, because the framework is designed to model an edge cloud environment, I will be able to evaluate whether the mitigations are suitable as DDoS protection for edge clouds.

Related work is presented in chapter 2 and the necessary background information for this thesis is included in chapter 3. In chapter 4 I will use the acquired background information to write down requirements for the framework. The design of the frame-

work and how it fulfills all requirements is shown in chapter 5. The implementation of the framework including all implementation decisions made is shown in chapter 6. I use the framework to evaluate different mitigations and settings in chapter 7. I evaluate the framework in chapter 8 and show my conclusions in chapter 9.

2. Related Work

In this chapter I want to present related work in the field of evaluating DDoS mitigations. Research on general purpose DDoS mitigation evaluation is rather rare, instead most mitigations feature a specific evaluation of their strategies in the evaluation section of their respective papers. This makes comparisons between different strategies difficult because every mitigation is evaluated slightly differently. However, there are some publications that acknowledge the need for a common evaluation strategy. In the following I will first examine the specific evaluation strategies of some DDoS mitigations and will then examine some propositions for common evaluation strategies.

In the evaluation of the moving target defense MOTAG [19, p. 4], the main considerations are the number of required shuffles and the amount of overhead introduced by the mitigation. The number of required shuffles refers to the amount of times that the mitigation's algorithm has to change proxy assignments of clients until a specific percentage of clients are considered saved. This evaluation is done by simulating the shuffling algorithm of the mitigation using MATLAB. While for some variables there are different values being tested, e.g. number of insiders, clients and proxies, this is done without any reasoning why their values are realistic. Also, other assumptions are made and not varied, e.g. that insiders always attack which is the best possible scenario from the mitigation's perspective. The evaluation of introduced overhead examines the additional latency introduced by routing client's traffic via proxies and not direct and should be considered specific to the exact servers picked in the evaluation.

Similarly, in Mole Hunt's self-evaluation [12, p. 6] the key metrics being observed are the number of required rounds which is comparable to MOTAG's number of required shuffles and the maximum number of concurrent addresses. The experiments are once again simulations of the algorithms of the evaluated mitigations. Mole Hunt's evaluation focuses on comparing its approach with similar approaches, namely MOTAG and DoSE, showing it requires less rounds and fewer concurrent addresses. The evaluation includes comparisons for different numbers of clients, insiders and Mole Hunt's splitsize option but does not justify the chosen values. Like in MOTAG's evaluation, it is assumed that all insiders always attack, MOTAG's algorithm however is provided with the assumption that only one insider attacks, which leads to an unrealistic result for MOTAG.

To summarize, the mitigation's evaluations focus on simulating the mitigation's algorithms and extracting technical metrics. They do not evaluate the impact that a DDoS attack has on the connection with clients such as a degradation in quality of service and how this effect is mitigated by the defense strategies. Also, the evaluation parameters

are not necessarily realistic and especially do not consider edge cloud applications. Lastly, more sophisticated attackers that potentially have insider knowledge are not considered.

To solve the problem of not being able to compare the evaluations of different DDoS mitigations, common evaluation methodologies have been proposed. In [22], the authors create sophisticated and resource-intensive tools that automatically collect examples of legitimate traffic, attack traffic and network topologies. This allows them to create a very realistic evaluation environment. The proposed metric focuses on the effectiveness of the evaluated mitigation by observing the success of high-level transactions during attacks. The sophistication of this framework however results in it being hard to use and therefore mitigations are rarely evaluated using it. While the authors of [15] do not propose an evaluation methodology themselves, they identify five dimensions where evaluations have to be carefully designed and then give recommendations in all of these areas. The multitude of recommendations per category however means that multiple evaluations which are all based on this advice will not necessarily result in comparable results.

As a consequence, the framework designed and used in this thesis differs from other mitigation evaluations in the following points. Not only the algorithm of the mitigation is simulated but rather the entirety of the mitigation and its technology is implemented and evaluated. Since the goal is to realistically evaluate the mitigations for use on edge clouds, the experimentation parameters are based on realistic expectations about edge clouds and their attackers. Also, more sophisticated attackers that purposely try to disrupt the tested mitigations are implemented and used. Lastly, the framework is designed to be highly customizable and extendable while still being easy to use.

In other aspects, this thesis' framework is similar to other evaluations and expands on them. For metrics, on the one hand, there are technical metrics that allow the algorithms of the mitigations to be analyzed and compared. Using the number of required rounds and the number of used addresses as metrics allows comparisons with the mitigiation's self-evaluations. On the other hand, there are quality of service metrics that allow for an overall evaluation of the mitigation. They give an answer to the core question of whether the mitigations are able to successfully defend against DDoS attacks by evaluating whether clients experience a degradation of quality of service during an attack. Like the authors of [15] recommend, this metric is created by combining a multitude of base network statistics into a few meaningful metrics.

3. Background

In this chapter I want to provide all necessary background information that is required for the following chapters. Firstly, I explain the reasoning for edge clouds and their design as well as some of the consequences this design has. I also take a look at how edge clouds are being implemented. Secondly, I go into detail about DDoS and all of it's mitigations, especially the types and specific approaches that I will be evaluating in this thesis.

3.1. Edge computing

Current general purpose datacenter based clouds excel at providing computation and storage resources at scale. While this is sufficient for many internet services, trends like mobile computing and Internet of Things involve edge services that require lower latency and less bandwidth to the core of the internet infrastructure which includes internet exchange points (IXP) and main datacenter clouds [9], [29]. Latency to datacenters cannot be improved as it is limited by physical constraints and bandwidth demands increase as more and more data is created and consumed on the edge of the internet.

For example, cloud gaming is a new service where the player does not have the processing power for running the game at home but rather only has a small client that receives the input from the player, forwards it to a server where the game is actually run and then receives a video stream to display to the user. This service requires low enough latency between the client and the server in order for the user not to notice any lag and enough bandwidth to send the video stream of the game. At the same time, various smart devices and the Internet of Things (IoT) generate large amounts of data that are being sent to the cloud for processing and storage, increasing the bandwidth needs to the core of the internet [24].

Edge clouds are a new form of distributed cloud on the edge of the internet and can address these issues. By being physically closer to the user, the edge cloud can dramatically reduce the latency to the edge device, see Figure 3.1. By pre-processing data on the edge or caching data from the core of the internet, edge clouds can also reduce the required bandwidth to the core of the internet [9]. For example, data can be compressed or filtered on the edge cloud before being send to the datacenter cloud. For instance the edge cloud could remove all parts without motion from a security camera

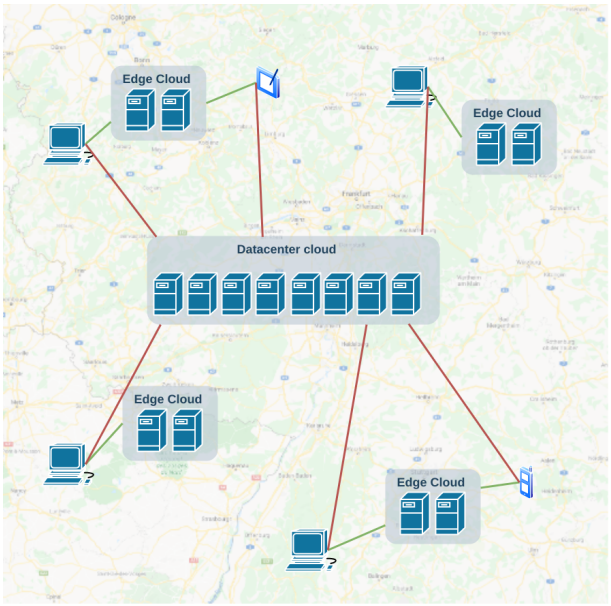


Figure 3.1.: Edge clouds have lower latency to users because they are physically closer

feed before it is archived in the datacenter, reducing bandwidth, see Figure 3.2. Caching data on the edge means that a copy of data on the datacenter cloud is saved on the edge cloud and when a user requests this data, it is served from the edge cloud rather than the datacenter cloud. This also reduces the bandwidth usage to the core of the internet, especially if the cached content is accessed frequently.

While low latency and bandwidth reduction are some of the advantages of edge clouds, the fact that an edge cloud node only has few computation resources in comparison to datacenter clouds in one of their disadvantages. This means that, while edge clouds generally increase the availability of internet services since they add additional distributed resources, the availability of a single edge cloud can easily be disturbed. During peak hours, even legitimate traffic only might supersede the computation resources of a single edge cloud [29]. These limited resources make edge clouds an especially attractive target for attackers [12], [29].

Some argue that edge clouds can be compared to peer-to-peer (p2p) networks, where every peer is both server and client [32]. Clients that want to access a service have multiple options for servers but every server is considered unreliable and might become unavailable at any time, requiring clients to change to a different server. Edge clouds are also server and client at the same time, providing services to end users while accessing the services of datacenter clouds as a client. And since edge clouds are considered easily overwhelmable because of their limited resources, they can be considered to be unreliable. But as opposed to peers in p2p networks, switching the service provider should an edge cloud become unavailable is not straightforward. Other edge clouds

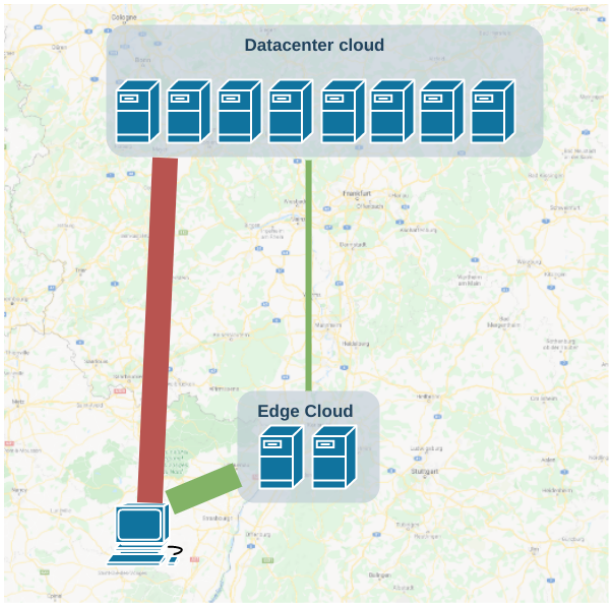


Figure 3.2.: Edge clouds can reduce traffic to datacenter clouds through pre-processing: Traditional traffic flow in red - Traffic flow using edge clouds in green

might not be able to handle the additional traffic of the now unavailable edge cloud since they also do not have a large number of resources, so the only choice for a fallback would be the datacenter cloud. But falling back to other edge clouds or the datacenter means that in most cases the requirements for edge services (especially low latency) can no longer be fulfilled, see Figure 3.3.

As a result of this edge cloud design, where a single edge cloud is vulnerable to an excessive amount of traffic, additional measures have to be put in place to make sure that edge clouds can provide the service they are designed to provide. The authors of [29] propose a hierarchical edge cloud infrastructure as opposed to the p2p approach where all edge clouds are equal to be able to handle all legitimate requests on edge clouds with best possible latency even during peak times. Still, low-latency DDoS mitigation for edge clouds is an open problem.

3.1.1. Implementation

In this section I want to give a quick overview on how the theoretical concept of edge clouds is implemented or planned to be implemented in the future.

Content Delivery Networks (CDN) are considered to be the predecessor of edge clouds. While they do not provide any computing resources for applications, they cache frequently accessed data from datacenter clouds in order to deliver this content to end users with lower latency than they would from the datacenter clouds. Examples of

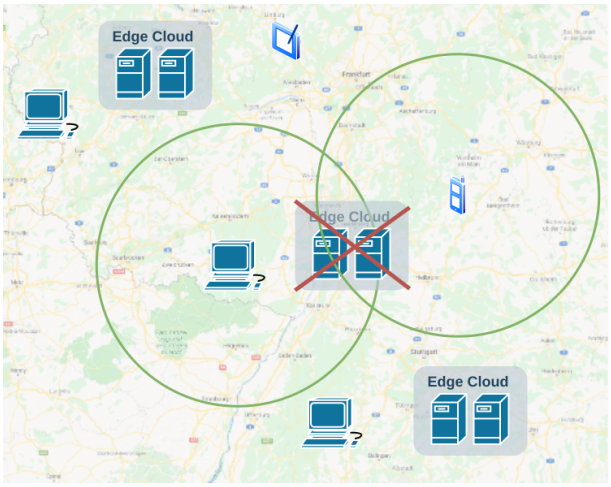


Figure 3.3.: Result of an unavailable edge cloud
When the edge cloud in the middle is unavailable, there are no datacenters within the latency perimeter of the computer and of the phone in the middle and edge services cannot be provided.

CDNs are Cloudflare with Points of Presence (PoP) in 200 cities [10] and Amazon Web Services CloudFront with 216 PoP in 84 cities, see Figure 3.4 [1].

There are also already cloud providers that run networks that they call "edge cloud platforms" like Akamai [4] or Fastly [11]. But in the case of Fastly, they run barely more PoP then some cloud platform providers (e.g. AWS) run datacenter clouds (and in similar locations) which means there is no latency reduction based on reduced distance to the end user and their PoP are actually designed to have a lot of computing power, see Figure 3.5 [11]. This means that their edge cloud implementation does not actually match my definition of edge clouds, but rather the one of datacenter clouds.

But this does not mean that there are no plans to implement edge clouds as I described them before. In addition to adding the ability to run computational tasks on Cloudfront PoP (Lambda@Edge), AWS is planning AWS Wavelength [2]. This service will deploy AWS infrastructure (compute and storage) at the edge of 5G networks in partnership with telecommunication providers like Verizon, Vodafone and SK Telecom. Wavelength will provide single-digit millisecond latency and is designed for edge services like game streaming [2].

3.2. Distributed Denial-of-service attacks

Denial-of-service (DoS) attacks try to prevent the legitimate use of internet services by exhausting the resources of the target. This can be done for example by exploiting bugs of the service or overwhelming the servers with a stream of packets. Distributed

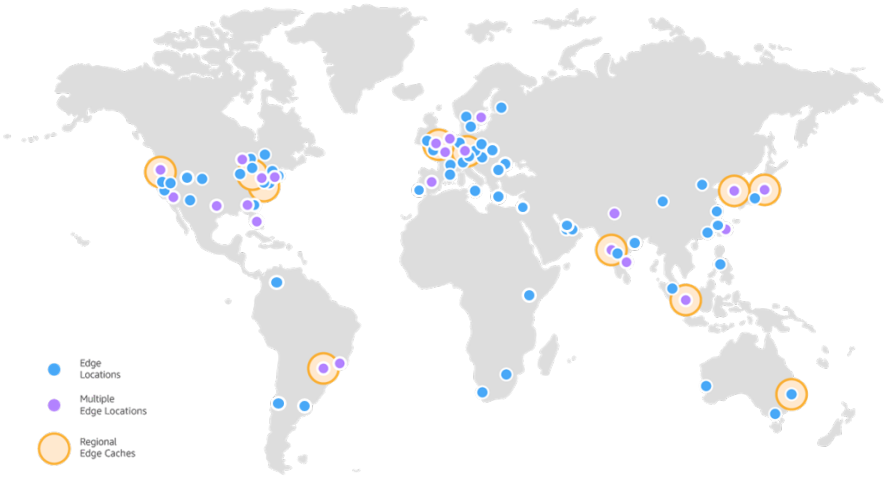


Figure 3.4.: Map of AWS CloudFront Points of Presence [1]

Denial-of-service (DDoS) means that the attack originates from multiple machines, often one handler plus multiple infected machines acting as agents (botnets) [23]. This is done to increase the number of malicious packets sent to the target.

There are a multitude of reasons why DDoS attacks work and there are many different types of DDoS attacks [23], which makes defending an infrastructure against DDoS attacks difficult.

According to [30], "Distributed-Denial-of-Service (DDoS) attacks represent the dominant threat observed by the vast majority of service providers — and they can represent up to 25 percent of a country's total Internet traffic while they are occurring. Globally the total number of DDoS attacks will double to 14.5 million by 2022 (from 2017)".

DDoS attacks on services hosted in the cloud often take shape as Economic Denial of Sustainability (EDoS) attacks. Since pay-as-you-go and autoscaling are core features of cloud computing, DDoS attacks often lead to an automated upscaling, causing economic damage to the service providers [28].

3.3. DDoS mitigation

Datacenter-based clouds as a whole are fairly DDoS attack resilient because of their substantial computing power, allowing them to run extensive attack prevention, detection and mitigation services as well as being able to outscale most attackers [28], [3]. There are various different DDoS defense mechanisms that evolve as quickly as attackers evolve their attacks. Many mechanisms try to filter attack traffic while protecting legitimate users' traffic by comparing it against known attack patterns, others try to locate attackers or modify communication protocols to stop attacks [15].

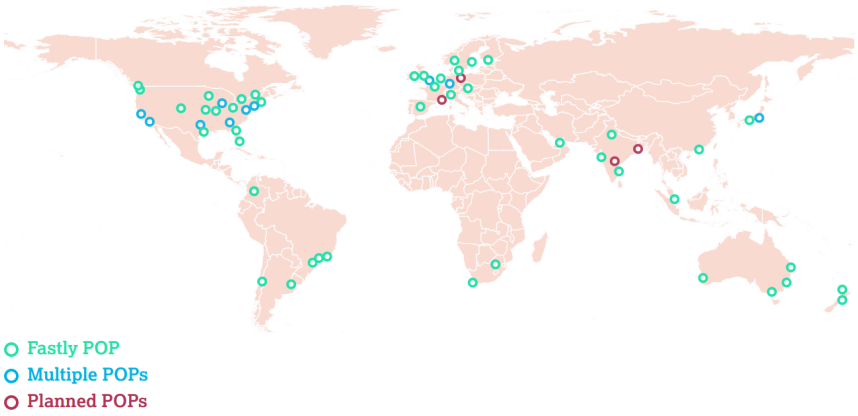


Figure 3.5.: Map of Fastly Points of Presence [11]

Edge clouds on the other hand, will not have the computing power to run any of these sophisticated mechanisms but instead will require a more lightweight approach to mitigate DDoS attacks [12]. For an overview of DDoS attack mitigations and their usability on edge clouds, see Table 3.1.

Current "edge cloud" implementations from Fastly and Akamai defend against DDoS attacks by rerouting traffic during attacks through scrubbing centers, which are high-bandwidth high-computing power datacenters designed to absorb and filter DDoS attacks [11] [4]. This obviously results in significantly increased latency for users and would defeat the purpose of an actual low-latency edge cloud.

3.4. Moving Target Defense

I will focus on mechanisms using Moving Target Defense (MTD), a technique where the attack surface is changed when attacks are detected, increasing uncertainty and randomization. Through continuous random reconfiguration of the target, the attacker is limited in gathering intelligence and the victim is able to evade the DDoS attack [21]. In the following I will take a closer look at some specific MTD approaches.

Mitigation	Short description	Problems	Edge cloud ready?
Filtering	Compare traffic against known attack traffic; separate legitimate traffic from attacks	• requires adequate resources • sophisticated attacks might not get filtered • unclear where 'known attack traffic' is from	No, not enough resources
Neural network filtering	Train a Neural Network with attack traffic then let it filter incoming traffic	• requires adequate resources • sophisticated attacks might not get filtered • legitimate traffic might get filtered	No, not enough resources
Rate limiting	Drop all traffic after a limit is hit	• also hits legitimate users • basically the same result as no mitigation	No, too restrictive
Locating attackers	Discard traffic from attacker after he is identified	• identification is potentially difficult • no mitigation while identifying • infrastructure has to be resilient enough to allow identifying	No, not enough resources
Outscaling	Have and use more resources than the attacker is able to overwhelm	• requires a lot of resources • requires resilient infrastructure that handles upscaling	No, not enough resources

Continues on next page.

Table 3.1.: Comparison of DDoS attack mitigations [28], [3], [15] - Part 1

Mitigation	Short description	Problems	Edge cloud ready?
		Continuation from previous page.	
Turing Test	Verify legitimacy of users by requiring them to pass a Turing test (e.g. CAPTCHA)	• limits legitimate clients • increases access time • requires resilient infrastructure • bots might still pass	No, not restrictive enough (only a few presumed legitimate users have to be malicious)
Proof of Work	Clients have to solve time-consuming crypto-puzzle	• limits legitimate clients • increases latency • requires resilient infrastructure • attackers can solve as well	No, not restrictive enough. While the number of malicious users is restricted, it might still be high enough
Prioritize trusted users' traffic	Drop traffic of users that are not considered trustworthy	• requires calculation of trustworthiness • requires resilient infrastructure for calculation • only application layer	If the application supports it and trustworthiness calculation is lightweight enough in combination for mitigation against non application layer attacks
Latency perimeter	Limits maximum latency for users; drops all other traffic	• attackers can also satisfy latency constraint	Alone not restrictive enough, but can be useful part in Edge Cloud mitigation
Moving Target Defense	Randomly and continuously change target attack surface to make attacks harder	*Topic of this Bachelor's Thesis*	

Table 3.2.: Comparison of DDoS attack mitigations [28], [3], [15] - Part 2

3.4.1. MOTAG

MOTAG [19] offers DDoS protection by connecting clients to the application server via a large pool of proxies whose addresses are only known by authenticated legitimate clients. This means that the only way attackers can send DDoS traffic to the application is when an insider forwards the address of the proxy it is connected to to the attackers. In case a proxy gets attacked, that proxy is then replaced and the clients that were connected to it get distributed to other proxies. The insider can then forward the address of the new proxy it is connected to to the attackers and the attackers attack that proxy. This process is repeated until only a single client is connected to a proxy that is attacked which means that this client is the insider which results in the insider being blocked from accessing the application. During this shuffling process, legitimate users are connected to the application via other proxies which minimizes the impact the attack has on them.

MOTAG offers multiple algorithms for determining client to proxy assignments during shuffling, an optimal algorithm and a greedy algorithm. The greedy algorithm is preferred because it is much more performant while it's assignments are only slightly worse than optimal. MOTAG assumes a large pool of proxies that are geographically distributed but only a few proxies are actively used at the same time.

3.4.2. Catch Me If You Can

The authors of [20] present a similar MTD approach called Catch Me If You Can. Compared to MOTAG, it does not require authentication and instead of proxies, it replaces and shuffles the application servers themselves, leveraging the computing power in the cloud. During an attack, Catch Me If You Can uses the same algorithms as MOTAG to determine client to server assignments. Running as well as frequently starting and stopping multiple application server replicas results in Catch Me If You Can requiring significantly more computing power than MOTAG.

3.4.3. DoSE

Denial of Service Elusion (DoSE) [31] is another moving target defense that is similar to MOTAG. Instead of varying the technical implementation of MOTAG like Catch Me If You Can does, DoSE uses a different algorithm for proxy to client assignments. This algorithm tries to be more cost-efficient by using less proxies. Also, DoSE does not require authentication but instead uses a "smart management layer" that communicates via a CDN and requires Proof-of-Work on a client's first connection. The authors of DoSE [31] argue, that MOTAG's stateless algorithm can be exploited by intelligent attackers while DoSE's algorithm can reduce both the time until insiders are identified and the number of failed transactions of legitimate clients while using less proxies.

Approach	Tech?	Algorithm?
MOTAG	Lots of Proxies	Insider-Number-Assumption based Optimal and Greedy Algorithm
Catch Me If You Can	Replicate Application Servers	MOTAG Algorithms
DoSE	Less Proxies	Risk-Assumption based Algorithm
Mole Hunt	Directly add/remove network addresses	n-ary search

Table 3.3.: Comparison of MTDs [19], [20], [31], [12]

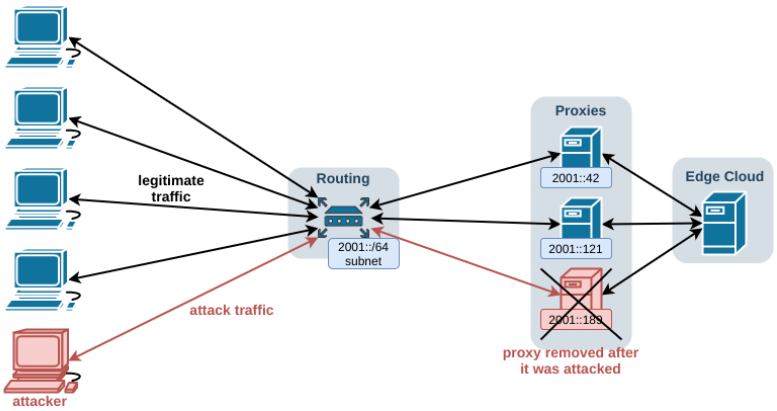
3.4.4. Mole Hunt

I will focus on a moving target defense mitigation strategy designed for use on edge clouds, Mole Hunt [12]. Mole Hunt works similarly as the other MTDs, but instead of using proxies with different network addresses like in MOTAG or DoSE, network addresses are directly assigned and removed from the application server. When there is no currently apparent attack, the application server has one network address that all clients use. Once the application server gets attacked, its network address is removed and new (random) network addresses are assigned to it. Clients that were connected using the just removed network address get distributed along the new network addresses and their connections are migrated. This process is repeated whenever a network address gets attacked until the insiders that forward the network address they are currently connected to, to their botnets to facilitate the attack, are isolated.

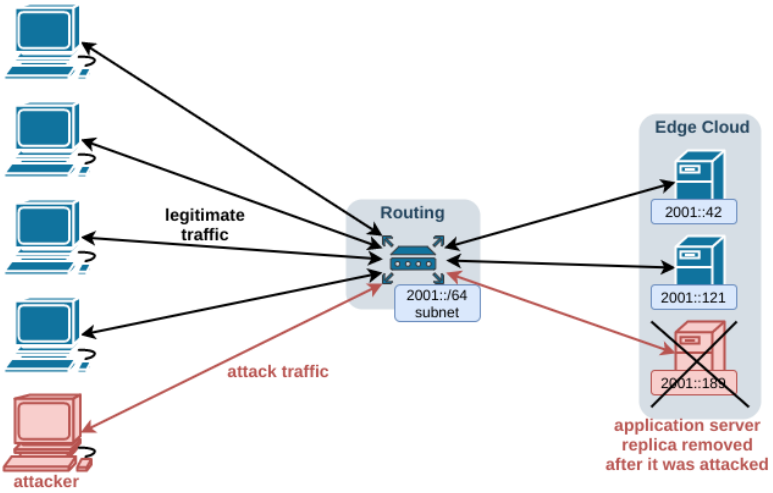
This new approach mitigates the DDoS attacks by repeatedly changing network addresses of the target leaving the attacks run into nothing shortly after they are detected. It can also detect and isolate the insider that is leaking the network addresses because the sequence of network addresses that a client is assigned to is unique after a sufficient amount of shuffles which means whichever sequence of network addresses a DDoS attack follows can be traced back to the insider. Because this approach does not use proxies (which require computing power) but only changes routing, this mechanism is very lightweight.

3.4.5. Comparison of MTD approaches

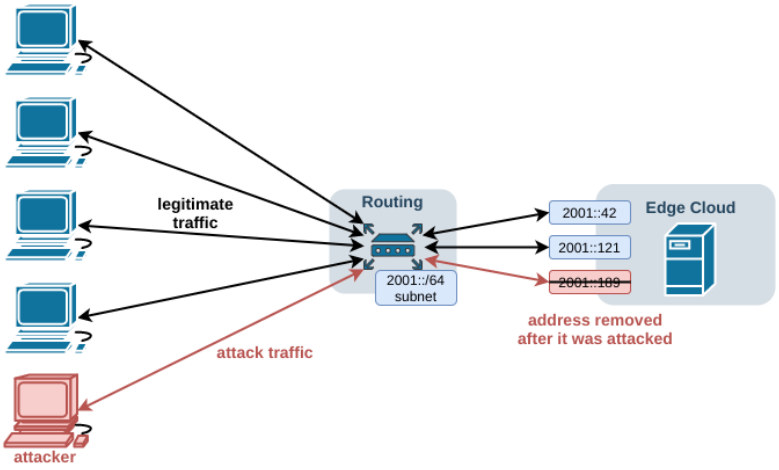
I compare the different moving target defense approaches in Table 3.3. I also show a graphic representation of how the different approaches work in Figure 3.6.



(a) MOTAG/DoSE



(b) Catch Me If You Can



(c) Mole Hunt

Figure 3.6.: Graphical representation of the different MTD approaches [19], [20], [31], [12]

4. Requirement Analysis

To evaluate network address shuffling approaches as DDoS protection for edge clouds, I plan on creating an evaluation and testing framework based on realistic assumptions about edge clouds as well as DDoS attackers. In this chapter, I will define requirements for the framework for it to be considered to be based on realistic assumptions as well as other requirements for the framework.

4.1. Edge Cloud model

The requirements on how to model edge clouds in my framework are based on the theoretical knowledge that I have about edge clouds, see chapter 3. In the following, I will list the characteristics of edge clouds that are vital to how edge clouds are required to be modeled in my framework, especially those that are distinct to edge clouds and differentiate them from datacenter clouds as well as those that are required by the DDoS mitigations which are planned to be implemented.

A010 Limited resources

Edge clouds have the resources to serve legitimate user's requests, but not a lot more. Even a higher number of legitimate clients than usual might overwhelm the resources of a single edge cloud. A DDoS attack on a single edge cloud without any mitigations can easily overwhelm the resources.

This is because edge clouds, as opposed to datacenter clouds are a distributed system with many points of presence, potentially in every 5G cell tower. Therefore a single edge cloud node does not require a lot of resources and therefore will not have a significant amount of resources more than required to server legitimate clients.

A020 Limited bandwidth

One of these limited resources is bandwidth. Edge clouds are connected with significantly limited bandwidth. Attackers might be able to overwhelm this bandwidth and cause legitimate traffic to be dropped.

Because edge clouds are only designed to serve a geographical limited number of clients, edge clouds will not have a significant amount of bandwidth more than they require. Also, since edge cloud nodes are connected at the edge of the internet and not the core, their bandwidth is limited.

A030 IPv6 ready

Edge will use IPv6 network addresses.

Network address shuffling DDoS mitigation approaches rely on a large pool of available network addresses. Legacy IPv4 addresses do not allow for this.

A040 Dynamic subnet-routing outside of edge cloud

The routing decision about traffic to addresses within the edge cloud subnet is done outside the edge cloud. The routing can also be quickly and dynamically changed by the mitigation running on the edge cloud.

This is required because network address shuffling approaches rely on hiding the actual network address of the edge cloud within a known subnet. If the entire subnet is routed to the edge cloud, all traffic addressed to the subnet will go to the edge cloud. Because the bandwidth of the edge cloud is limited (requirement A020), this means that traffic will be dropped before a routing decision is made, especially whether the recipient address is a valid edge cloud address. This means that even if the routing correctly drops all attack traffic and forwards all legitimate traffic, some legitimate traffic will still not reach the edge cloud. To solve this, the routing decision has to be done outside the system with significantly limited bandwidth. This means that in order for the mitigations to be able to block attackers from sending attack traffic to the edge cloud, they have to be able to modify the routing that is happening outside of the edge cloud.

A050 Connection migration

Continuous connections between the edge cloud and its clients can be migrated from one edge cloud network address to the next.

This is because some edge services will require a continuous connection between the client and the edge cloud, e.g. video game streaming and edge clouds are supposed to provide a service with high-availability. When mitigations shuffle network address assignments of clients, some of those continuous clients will be assigned new network addresses which requires them to migrate the connection to the new address.

4.2. Attacker model

Similarly, in this section I want to extract the key characteristics of DDoS attackers on edge clouds into requirements on how to model them in my framework.

B010 Sufficient, but limited resources

Attackers have sufficient resources to overwhelm a single edge cloud, but do not have enough resources to overwhelm the internet backplane or the routing infrastructure.

If the resources of attackers are too few to overwhelm an edge cloud node, no mitigiation would be necessary. At the same time, if the attackers have too many resources, and can overwhelm routing or internet backplane infrastructure, mitigations on the edge cloud are helpless, because legitimate traffic is dropped before the mitigation can save it. This scenario is also not realistic since the internet core has a substantial capacity and attackers are limited by their latency at accessing edge clouds. Therefore, it is realistic to assume that attackers have sufficient, but limited resources.

B020 Limited number of insiders

The number of clients of the edge cloud, that forward the network address they are currently connected to, to the attackers to be attacked (= insider) is limited.

Because of edge cloud's geographical/latency limitations, the number of overall clients of a single edge is limited. It can also be assumed that a large majority of the clients are legitimate based on how the edge environment will be designed and that an edge cloud will only be existent when edge services are in demand.

B030 Limited number of attackers

The number of clients that send malicious traffic is limited.

Since edge clouds are designed to only serve a specific geographical region, only clients within this region will be able to send malicious traffic. Edge clouds will do this by running a latency perimeter, dropping traffic that takes over a specific threshold to travel from the client to the edge cloud and back.

B040 DDoS flooding attacks

Attackers attack by using DDoS flooding attacks.

This is because other DDoS attacks have specific mitigations based on how exactly the denial-of-service is achieved that are much more effective for this specific type of DDoS attack. Flooding attacks are the most common type of DDoS attack and do not have a specific mitigation. Also, mitigations against flooding attacks will work against most other attacks.

B050 Different attack patterns

Attacks on edge cloud might use a variety of different attack patterns including straightforward approaches as well as sophisticated attacks which try to outmaneuver the mitigation. The framework should allow for different attack patterns to be tested against the mitigations. The patterns should be easily swappable in the framework and it should be possible to add new ones.

This is a requirement because while many DDoS attacks are straightforward and usually rely on their brute force, the sophistication of these attacks increases. Especially considering the limitations the edge environment brings with, sophisticated attacks should be expected.

B060 Insider information

Additionally to currently used network addresses by the insider clients, the attacker might have other insider information including but not limited to the mitigation being run and its configuration parameters.

Considering the current documentation on which DDoS mitigations are in use in datacenter clouds, similar information might be available about edge clouds when they are available.

4.3. Benchmarks

Aside from creating a realistic representation of an edge cloud under attack, the framework will need to capture data and create useful metrics so that mitigations can be properly evaluated. You find the requirements for these benchmarks in the following.

C010 Metric: Required Rounds

The number of rounds it takes a mitigation until the attackers are not able to attack anymore. A round is defined as a sequence of actions:

1. The attackers choose one or more of the network addresses of the edge cloud known to them through their insiders. This choice is based on the attack pattern.
2. The attackers start a flooding attack on the chosen address(es).
3. The mitigation mitigates the flooding attack which includes making the attacked address(es) unavailable.

Attackers are unable to attack anymore when all their insiders have been identified by the mitigation and have been blocked from accessing the edge cloud.

The number of required rounds is the key metric when determining the effectiveness of a shuffling algorithm as it quantifies the time until a series of attacks is mitigated without being implementation-specific.

C020 Metric: Used Addresses

The number of network addresses the mitigation requires until the attackers are not able to attack anymore.

While this number can be calculated when the number of required rounds as well as the implementation of the mitigation and its configuration parameters are known, it is an important metric when comparing mitigations or mitigation configuration parameters. Since the number of required rounds and the number of used addresses are usually a trade-off when designing a network address shuffling DDoS mitigation (a lower number of required rounds can be achieved by using more addresses), both metrics are required to properly compare mitigations.

C030 Quality of Service Metrics

The quality of service (QoS) legitimate clients connected to an edge cloud experience while this edge cloud is being attacked. Proper metrics to quantify QoS that distinguish different mitigations or their configuration settings have to be found.

This is a requirement because the main reason why DDoS mitigations are run on edge clouds will be to be able to continue providing service to legitimate clients while being attacked. This is why metrics about the provided service during attacks is an important benchmark on the effectiveness of the DDoS mitigation.

4.4. Technical Requirements

In the following I will list any other requirement for the evaluation framework.

D010 Reproducibility

The results of the framework have to be reproducible. Because various parts of the framework purposely include randomness (e.g. choosing the insiders from all clients at random), this means that the framework will need to run multiple iterations of a configuration with their average as the final result. This also means that the framework has to ensure a clean state between iterations, making sure one iteration do not influence another.

This is required because only reproducible results are useful in an evaluation of the mitigations.

D020 Configurability

All settings for the framework, the mitigation and the attacker have to be easily configurable in a single place. Additionally, runs with different configurations after another without interaction have to be possible.

Because iterations can take some time, and a run (multiple iterations with same configuration) even more, a common usage will run the framework on a remote server with interaction cut to a minimum.

D030 Hard-failing and extensive logging

An error during an iteration should always lead to the fail of the entire iteration, but should be documented.

This is because a result which is based on an iteration where an error occurred should always be ignored since the result cannot be considered correct anymore.

D040 Infrastructure tests

At the start of an iteration, the framework should run tests to make sure the infrastructure is properly setup and the configuration is valid.

This is a requirement because not testing the infrastructure could mean that the iteration is run on improperly configured infrastructure which would invalidate a result.

D050 Required resources

The resources the framework requires to run should be limited.

Although the framework aims to model an infrastructure with multiple distributed clients, the framework should not require the same amount of resources, but instead be able to run on one single (quite powerful) client.

D060 Pluggability and Extandability

The framework has to be able to support multiple mitigations and attack patterns and therefore should provide an API to be used. Mitigations and attacker patterns should be able to be modified and added without any changes to the core of the framework.

This allows future mitigations or attacker patterns to be added and compared against current mitigations and attacker patterns easily.

4.5. Limitations

There are some things that I do not expect my framework to do. They are listed below.

No attack detection Because detecting whether there is currently an attack is a whole other topic for itself and I want to evaluate mitigations not detections, attack detection is not part of the framework. Instead, I assume that an attack is reliably detected after a configurable timespan.

No new clients I assume a fixed number of clients that already know an initial network address to reach the edge cloud. Implementing a service discovery that gives out routing information to new clients while protecting against attackers and is attack-resilient is not straightforward and not the goal of this thesis.

5. Design

This chapter documents the design of the framework and how it fulfills all requirements. The evaluation framework developed here will be referred to as *dmef*, short for DDoS mitigation evaluation framework.

When dmef is run, it starts by creating a model of an edge cloud infrastructure environment. In this environment, all devices that would play a role in a real-world edge cloud environment are modeled individually. This includes the edge cloud which is modeled with its limited resources (**A010**) including limited bandwidth (**A020**) and is setup to use IPv6 (**A030**). It also includes a routing infrastructure that connects all modeled devices and can route known attack traffic into a blackhole (**A040**) as well as clients that are connected to the edge cloud using migrateable connections (**A050**). The model environment also includes a limited number of attackers (**B030**) with sufficient, but limited resources (**B010**) that can send flooding attack traffic to the edge cloud (**B040**).

After dmef has setup the model environment, it performs a series of tests on it (**D040**). Then, the mitigation to be tested is loaded dynamically (**D060**) and is setup on the infrastructure. When the setup is complete, a configured limited number of clients is elected to be insiders (**B020**) and the benchmarking is started. The configured attack pattern is then in charge of directing the attackers to send attack traffic to the edge cloud through the addresses known to it through the insider. Shortly after an attack is started, it is the mitigation's job to mitigate the attack and to make any changes

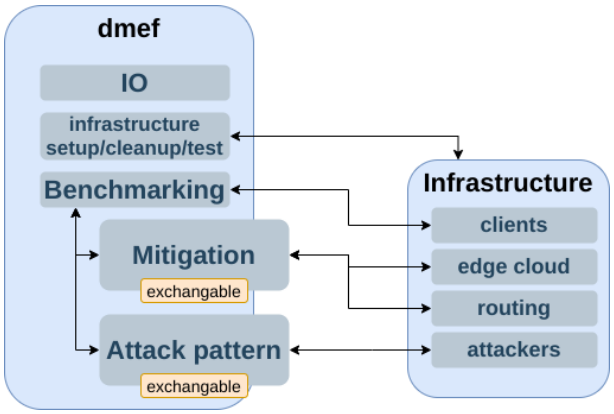


Figure 5.1.: dmef components and their interactions

it decides on to the routing before the attack pattern can once again decide which addresses to attack. This process repeats itself until the mitigation has identified all insiders and the attack pattern has run out of addresses to attack. This is when the benchmarking is stopped and the metrics are calculated. These metrics include the number of required rounds (**C010**), the number of used addresses (**C020**), the time with interrupted service and it's percentage of the entire runtime (**C030**). Afterwards, the infrastructure is cleaned up, to allow for reproducibility (**D010**) and the entire process is restarted when configured to do so. The different mentioned components of dmef and their interactions are displayed in Figure 5.1.

A number of different attack patterns (**B050**, **D060**) including ones that use insider information about the mitigation in order to try to outmaneuver it (**B060**) can be configured to be used as well as all other parameters of the evaluation (**D020**). The entire time, occurring errors lead to fail of the iteration, logging the error in the process (**D030**).

Figure 5.2 also shows the process of running a configuration on dmef.

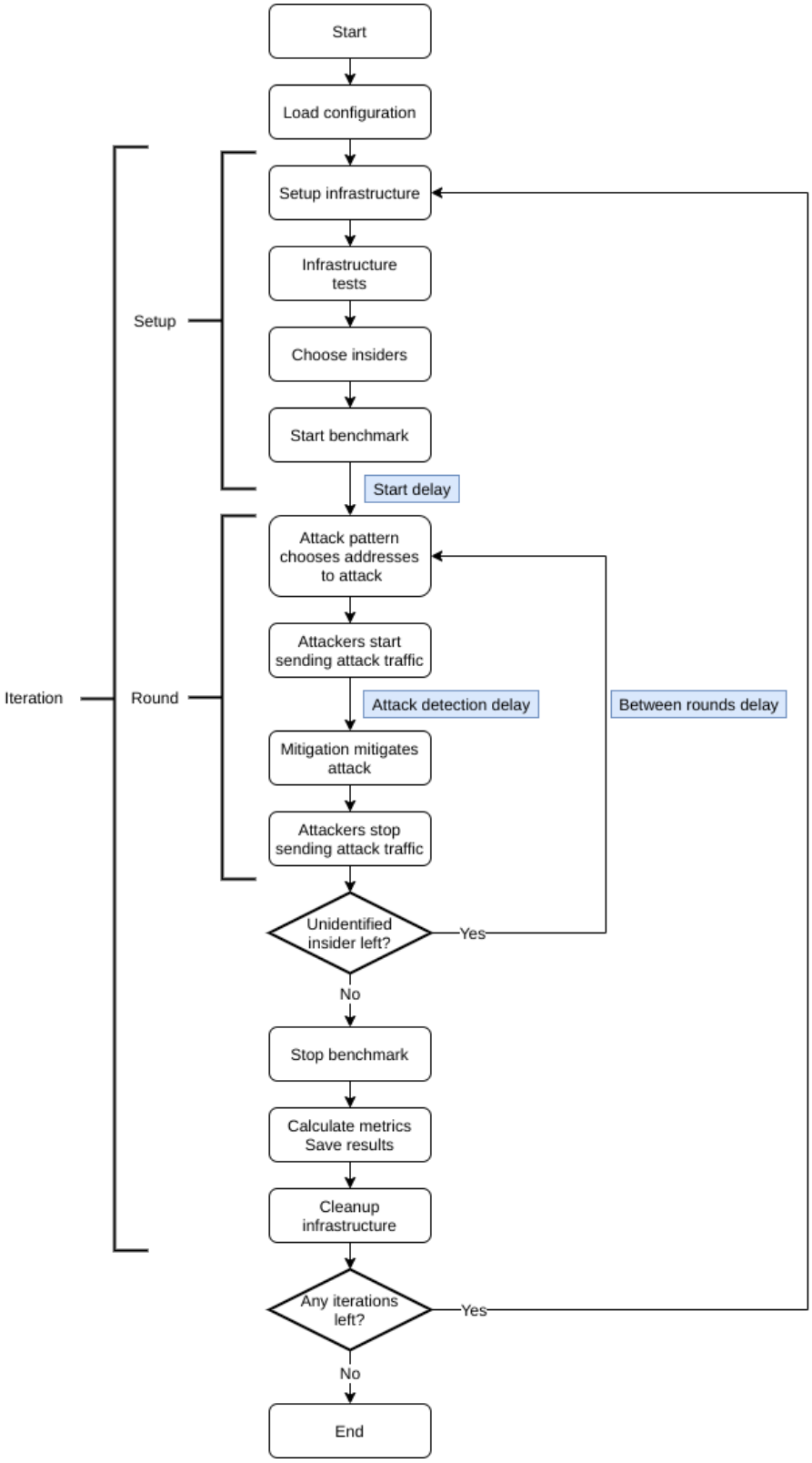


Figure 5.2.: Evaluation process

6. Implementation

In this chapter, I want to dive deeper into the different parts of dmef and show how they are implemented and explain why they were implemented this way.

6.1. The controller

The dmef controller is the central implementation of the framework logic. It handles input/output, sets up the infrastructure, loads and runs the mitigations and attack patterns, handles the benchmarking and logging and cleans up in the end. The controller is implemented as a number of object-oriented python classes running in a docker container [17] to reduce software requirements and increase portability. See Figure 6.1 for an overview of the controller classes.

Most importantly, the controller provides the API for mitigations and attack patterns and loads them dynamically. Mitigations and attack patterns are python classes inheriting a specific abstract class that implement specific methods. The methods to be implemented and the API that dmef provides these classes is documented in the dmef documentation. This allows more mitigations and attack patterns to be added seamlessly and existing ones can be modified on their own, not requiring changes to the core framework (**D060**).

The controller also handles the input and output of the framework. If dmef is run without an argument, an interactive shell is provided where configuration settings can be set, mitigations initialized and attacks run. The framework makes sure to clean up any containers or networks set up when quitting dmef. Alternatively, dmef can be run with an argument which is the filename of a configuration file which specifies all configuration options including number of iterations, the mitigation and the attack pattern (**D020**). A configuration file can also include multiple sets of configuration options which will be run after another. Listing 6.1 shows an example for a configuration file.

The controller interacts with other docker containers by opening a shell in them using the docker exec-environment. While a HTTP-API would have been possible, this might have lead to problems when an interaction was required while a DDoS attack was in process where packets are potentially lost.

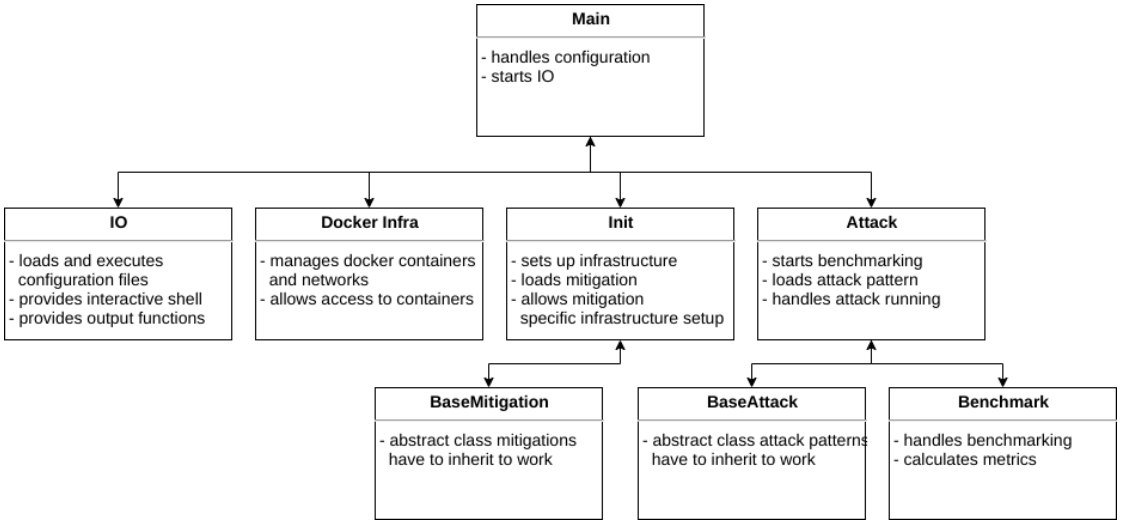


Figure 6.1.: Overview over dmef controller classes

Listing 6.1: config.yaml

```
---
- iterations: 20
  attackers: 3
  insiders: 3
  clients: 25
  mitigation: molehunt
  attack: oneatonce
  molehunt_splitsize: 2
- iterations: 20
  attackers: 3
  insiders: 3
  clients: 25
  mitigation: motag
  attack: oneatonce
  motag_proxycount: 1
  motag_preallocatedproxies: 0
```

6.2. Infrastructure

The design of the infrastructure that the framework sets up is based on the proof-of-concept of Mole Hunt [12]. The proof-of-concept setup infrastructure using docker containers for edge cloud, clients and routing and then made a single mitigation round where clients were migrated to a new address. There were no actual attacks, no benchmarking, no way to configure options or to change the mitigation and more things missing that my framework includes.

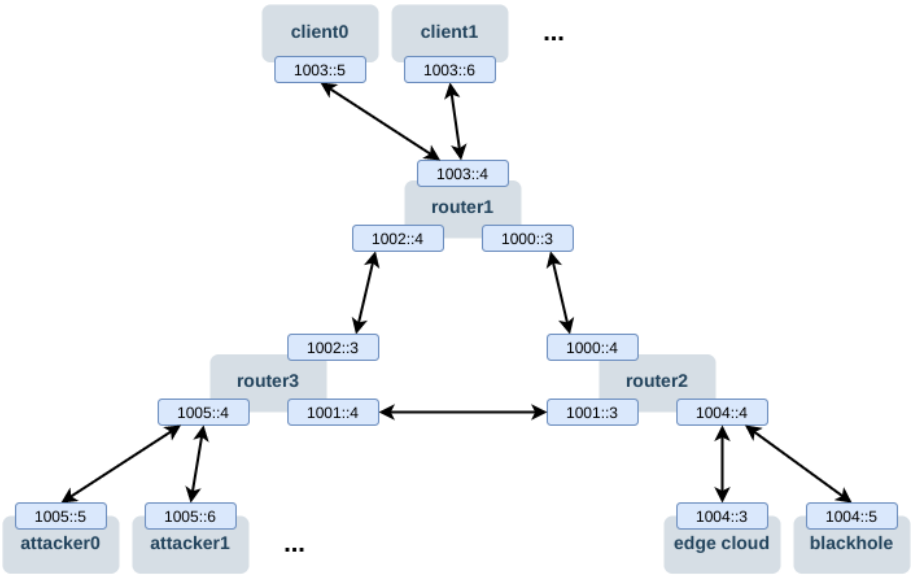


Figure 6.2.: Docker infrastructure setup by dmef [12]

Like the proof-of-concept, dmef uses docker containers to model any device in the edge cloud environment, namely the edge cloud itself, the blackhole, the clients, the attackers and the routers [17]. Together with the docker containers, docker networks are used to connect docker containers which are able to communicate with another and to assign them network addresses. All device containers are connected via routing containers which represent the internet routing infrastructure. They allow for dynamic routing which means that the mitigation can change the routing (**A040**). See Figure 6.2 [12] for a visual representation of the infrastructure dmef sets up. Connecting lines between containers represent that they are connected via a docker network, the network address of a container in this network is noted on the line.

Every type of device modeled also uses a dedicated docker image for its containers. A customization to this images can be done by basing a new image on the default ones and adding a new docker image layer. This allows for easy modification or extension of the infrastructure (**D060**).

6.3. Technologies

Similar to the basic infrastructure, most technologies used in dmef are taken over from the Mole Hunt proof-of-concept implementation. This had the advantage that I already knew those technologies worked in this context and that base configurations were already available to be used and adapted.

dmef uses docker to model devices and networks because its grade of virtualization allows us to almost always treat individual devices as dedicated bare-metal devices while dramatically reducing the number of required resources (**D050**) [17]. This means that while one can easily run up to 100 docker containers on a single machine, every e.g. client can be used and accessed as if it was an actual dedicated edge cloud client and the software run on these clients can be the same that is on run on actual clients.

The controller, as well as other software parts of dmef, are written in python because it has a docker module, I was comfortable at using it and the Mole Hunt proof-of-concept was written in it.

For the routing infrastructure that is set up by dmef, the routers run bird [5] and communicate via the OSPF (Open Shortest Path First) protocol [27]. Additionally, for the incoming edge cloud routing, the edge cloud announces and withdraws addresses using exaBGP [6] and the Border Gateway Protocol (BGP) (Mole Hunt only) (**A040**) [26]. For outgoing edge cloud traffic, dmef is using nftables [8] for the correct network address translation (NAT). All these are standard tools in networking which is why they were chosen.

For the application traffic, I am using QUIC [18], a new transport protocol based on UDP [25], and its python implementation aioquic [7] because it supports the migration of connections to new network addresses (**A050**). While this connection migration feature works seamlessly when the client of a connection changes network addresses, it does not when the server changes addresses. This is why in dmef, the edge cloud is always the client of a connection, after all edge clouds are expected to be both clients and servers depending on the service. Additionally, since the application is continuously sending a ping back and forth and logging its dates, being the client only means that the edge clouds initiates the connection and makes no difference while the connection is running.

6.4. Attack traffic

In scientific research, attack traffic is usually generated in one of two ways. One of which being that actual attack traffic is recorded at one point and then replayed at another, meaning the attack traffic is actual attack traffic, but this obviously brings up the question on where to record attack traffic [22, p. 3ff]. The other is synthetic attack traffic which is created based on sophisticated models taking account of the application and transport protocols of the application which is a resource-intensive procedure [15, p. 1f]. In the real world, the tools that are used to conduct actual DDoS attacks tend to be much more straightforward. For example, High Orbit Ion Cannon (HOIC) is a tool that is for example used by the group *Anonymous* [16]. It works by sending a flood of well-formed legitimate-looking HTTP-GET or POST requests to the target server.

This shows that real world attacks are more about brute-forcing with large number of packets rather than sophistication.

In my framework, sophisticated attack traffic is not needed since there are no filters or similar mitigations that could easily defend against trivial attack traffic. Also, since attack detection is not part of the framework and the detection is manually triggered anyway, the only objective of the attack traffic is to actually exhaust resources, namely the edge cloud bandwidth and edge cloud processing power, which can be achieved with any traffic as long as it is enough. This makes the only requirements for the attack traffic generation IPv6-support and performance, meaning the ability to create large number of packets. In dmef I am using a modified version of open-source THC IPv6 attack toolkit [13] written in C which fulfills these requirements and can produce about 4000 packets per second per attacking docker container. During my initial testing I find that this sufficient for overwhelming the edge cloud and does severely impact quality of service for legitimate users.

6.5. Limiting bandwidth

According to the requirements, edge clouds have significantly limited bandwidth (**A020**). When setting up the infrastructure using docker, all containers essentially have the same bandwidth which is limited by the host's resources. To model the edge cloud's limited bandwidth, I artificially limit its network link to the edge cloud router, see Figure 6.3 for the link in question. I do this by limiting a specific network interface on the docker host which one can find based on the docker network name and docker container name. To do this, the controller container has to be in the docker host network and requires the `NET_ADMIN` capability. On the specific network interface, dmef then imposes a queue limit using tc [14].

6.6. Bechmarking and Metrics

As required, dmef counts the number of required rounds as well as the number of used addresses (**C010** & **C020**).

For the QoS metric, I initially planned to use both latency as well as package loss percentage for packages sent between clients and the edge cloud (**C030**). This however turned out not to be ideal for two reasons. Firstly, the way the continuous connection between clients and edge cloud is implemented is that a ping message is sent back and forth with one being sent right after the preceding one was received. To properly measure the latency or package loss however, pings would have to be sent at a constant interval. This would require us to asynchronously send and receive pings, paying attention to sequence numbers, which is not supported out-of-the-box by the aioquic

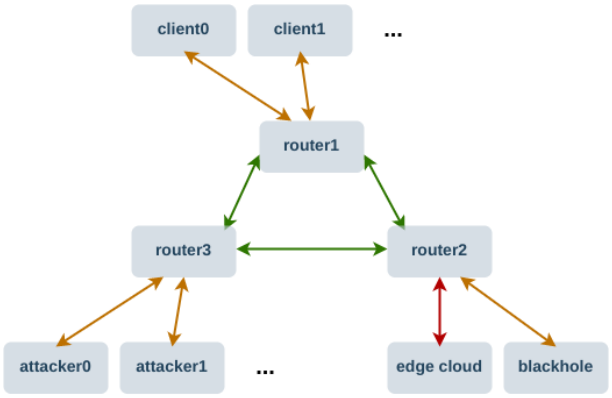


Figure 6.3.: Docker network links
Green links are assumed to have unlimited bandwidth and red links have limited bandwidth which results in them possibly being overwhelmed and are therefore manually limited by dmef. Orange links also have limited bandwidth, but are either not expected to be overwhelmed or overwhelming does not impact legitimate clients.

library. Additionally, QUIC automatically retries packages that were not received, which would result in the number of dropped QUIC pings being zero. Secondly, my initial tests showed that during the time that a DDoS flooding attack was active and not mitigated, basically no legitimate packets would make it to the edge cloud. This would mean that during this time, latency is infinite and package loss is 100%.

Instead, I introduce new metrics called Time with Interrupted Service (twis) and Time with Interrupted Service as Percentage of entire runtime (twis%). twis is the amount of time in seconds that pings get not returned by the edge cloud within the edge service limit. twis% is defined as twis divided by the entire runtime of the iteration in dmef from start of benchmarking to end of benchmarking as a percentage.

I calculate twis from the raw quic log for every client separately, see Listing 6.2 for an example. First I check if at any point the client and the edge cloud lost connection and the connection was restarted. If this happened, the time without a connection is considered time with interrupted service. Additionally, I consider the runtime of every ping with a runtime above a threshold as time with interrupted service. This threshold is defined as the highest runtime within the lowest 90% of runtimes plus its difference to the mean of the lowest 90% of runtimes. This calculation is visually shown in Figure 6.4. The sum of these times is the twis for this client. The twis% for this client is this twis divided by the runtime, defined as starttime plus runtime of last ping minus starttime of first ping. Lastly, I calculate the averages for twis and twis% over all legitimate clients for an iteration, as well as the sum of all legitimate clients twis. Finally, I have the results for an iteration, for an example see Listing 6.3.

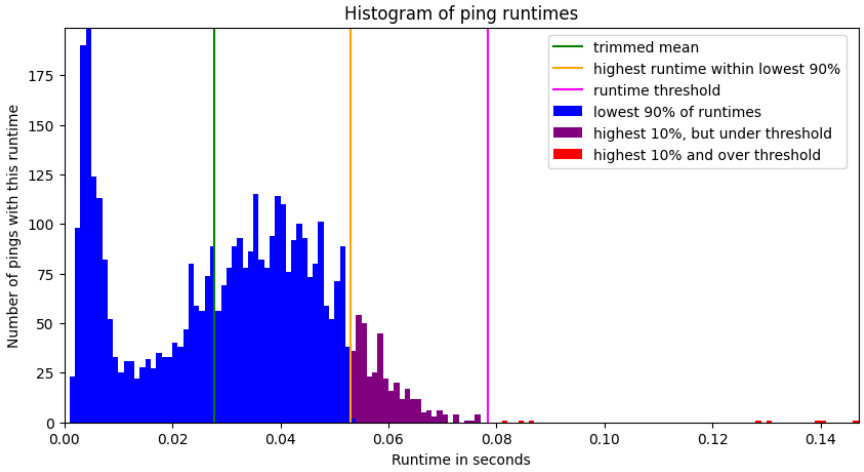


Figure 6.4.: Calculation of twis

Shown is a histogram of the runtimes of all measured QUIC pings of one client in one iteration. **Note, that right-most bar includes all runtimes over 0.145s, including multi-second runtimes during attacks.** The highest runtime of the lowest 90% of runtimes (blue) is marked in orange, their mean is marked in green. The threshold (magenta) is the highest runtime (orange) plus the difference between it (orange) and the trimmed mean (green). The sum of all runtimes over the threshold (red) is considered the time with interrupted service (twis).

Listing 6.2: quic-raw.log

```
[...]
2020-05-15 17:58:42,688 INFO client host;seq;starttime;runtime
[...]
2020-05-15 17:58:46,093 INFO client 1003::10;17;1589565526.0873663;0.005839347839355469
2020-05-15 17:58:46,199 INFO client 1003::10;18;1589565526.195922;0.0031845569610595703
2020-05-15 17:58:46,316 INFO client 1003::10;19;1589565526.3054483;0.010923385620117188
2020-05-15 17:58:46,435 INFO client 1003::10;20;1589565526.4299755;0.005669593811035156
2020-05-15 17:58:46,538 INFO client 1003::10;21;1589565526.5359917;0.0022122859954833984
[...]
```

Listing 6.3: result.log

```
Used Addresses: 21
Rounds required: 13
Sum time_to_attack_detection 130s
Sum time_between_attacks 520s
client          tmean    fast/slow       twis/twis%
client0         0.029s   4065/44         358.69s/39.75%
client1         0.029s   3899/32         379.43s/42.11%
client2         0.032s   3291/33         452.39s/50.15%
client3         0.028s   4186/39         345.33s/38.3%
client4         0.033s   3377/34         437.81s/48.54%
client5         0.028s   4141/39         351.84s/39.08%
client6         0.03s    3514/36         429.48s/47.61%
client7         0.031s   3797/37         389.02s/43.12%
client8         0.03s    3552/33         422.67s/46.93%
client9         0.031s   3524/36         423.11s/47.0%
```

client10	0.027s	4314/39	334.46s/37.09%
client11	0.028s	3654/40	416.46s/46.22%
client13	0.029s	3429/32	442.93s/49.18%
client14	0.031s	3718/36	397.37s/44.12%
client15	0.03s	3920/38	373.72s/41.45%
client16	0.03s	3612/34	412.15s/45.8%
client17	0.03s	3430/32	438.42s/48.64%
client18	0.027s	3496/33	438.59s/48.72%
client20	0.029s	4122/39	352.04s/39.05%
client21	0.031s	3584/29	414.62s/46.03%
client22	0.03s	3933/37	371.18s/41.19%
client24	0.028s	4200/41	343.2s/38.13%
AVG	0.03s	3761.73/36.05	396.59s/44.01%
SUM	–	82758/793	8724.9s/–

6.7. Infrastructure tests

At the beginning of each iteration, the infrastructure is tested to make sure the setup was successful. This is done by first making sure that all clients can connect to the edge cloud. Then an attack is started where DDoS attack traffic is send to the edge cloud without starting any mitigation. dmef makes sure that the service is then impacted by the attack by expecting at least one in five connection attempts to fail. dmef then stops the attack and after a short wait makes sure that all clients are able connect again.

6.8. Mitigations

For documentation on how and why these mitigations work, refer to chapter 3 or the original papers on these mitigations. This section is providing insight into how the mitigations are implemented on dmef.

I decided to only implement Mole Hunt and MOTAG because these two both differ in their algorithms as well as in the technologies used. DoSE uses the same proxy setup as MOTAG but with a different algorithm, comparisons using simulations are therefore sufficient and have already been done. Catch Me If You Can also uses the MOTAG greedy algorithm and its different technology makes it an unfit candidate for edge clouds.

6.8.1. Mole Hunt

I once again refer to the proof-of-concept implementation of Mole Hunt [12].

During the post-infrastructure setup phase, additional Mole Hunt specific services are started and configurations modified. For example, exaBGP to announce and withdraw network addresses is configured and setup, the default network address removed, a new default address (in the proper subnet) is announced, the edge cloud is set up to accept traffic addressed to any address in the subnet and NAT rules are set up.

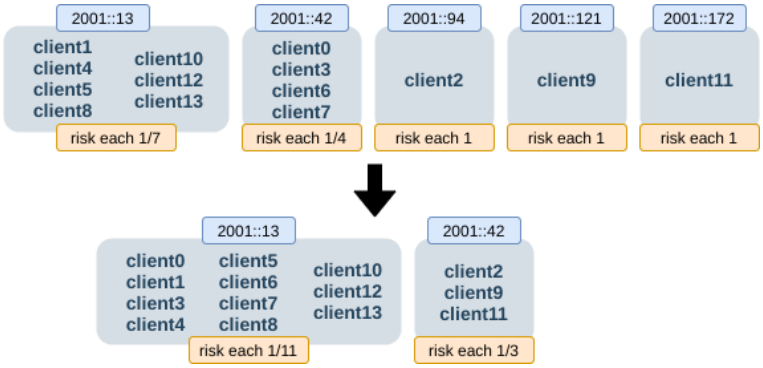


Figure 6.5.: Example of remerging in Mole Hunt

When the edge cloud is attacked, for every attacked network address, this address is first withdrawn via exaBGP and then the connected clients are handled. This means that if only one client was connected to an address, this client was an insider and is banned from accessing the edge cloud in the future. If more than one client is connected to an attacked address, splitsize-many new random addresses are generated and announced via exaBGP and the clients are then randomly divided into splitsize-many equal-size groups which are assigned to the new addresses. If less then splitsize-many clients were connected to an address, every client is assigned a dedicated new address but no additional addresses are announced. Assigning to a new address means removing the old NAT configuration for the client, updating its routing information and setting a new NAT configuration.

New in my implementation is the remerging strategy which is based on ideas by the Mole Hunt author. During each round of mitigations, new addresses are announced, the routing table gets larger and the number of connected clients per address decreases. Remerging means that clients are redistributed along a smaller number of addresses when no attacks are happening in order to lower the number of used addresses and shrink routing tables. I expect that remerging happens after no attacks have happened for a longer period of time and the mitigation expects to have caught all insiders. There are a variety of different strategies on how to redistribute the clients including just using a single address for all clients or using equal sized groups. I want to however anticipate that attackers might be aware of the strategy and purposely wait until the mitigation remerges in order to avoid the isolation of the insiders and only attacking when an address is used by a larger group of clients.

That is why I want to preserve the isolation progress while remerging as much as possible. This is done by assigning every client a risk factor which is defined as 1 divided by the number of clients connected to the same address. This means that a client that has an address on its own, has a large risk, namely 1 while 20 clients that share an address each have the lower risk of 0.05. Then, two groups with an address

each are created, a group of lower risk clients and a higher risk group, so that the cumulative risk for each group is about equal. This creates a group of many clients with a low risk, meaning that the probability of an insider being part of this group is low and group of clients the mitigation suspects might be insiders. When dividing groups, additional clients may be added to the higher risk group when there is already a client with the same risk value in the high risk group and the number of clients does not rise so high that based on the splitsize more rounds are required to identify an insider. See Figure 6.5 for an example on how remerging works.

6.8.2. MOTAG

In the post-infrastructure setup phase in MOTAG, instead of setting up exaBGP like in Mole Hunt, additional proxy containers with random network addresses are created that forward their incoming traffic to the edge cloud. The number of proxies that are setup in this phase is configurable as initial proxies. The initial routing assigns each proxy an equal number of clients to handle or in case of only one initial proxy assigns all clients to it.

When the edge cloud is attacked, MOTAG mitigates by first removing all proxies that handled attack traffic. If any proxy removed only handled one client during the attack, that client is an insider and is banned from accessing the edge cloud in the future. If not new proxies are created to be used as shuffling proxies, up to one for every proxy removed plus one additional. Using the MOTAG greedy assign algorithm [19, p. 5] any client that was connected via a proxy that is now removed is then assigned to one of the new shuffling proxies. The algorithm requires an assumption about the number of insiders for which the number of addresses attacked during this round is provided since this is a lower limit.

dmef allows the pre-allocation of proxies which means that the docker containers for proxies are created but not used at the beginning, and not only when they are needed. The number of pre-allocated proxies is configurable and defaults to zero. After all pre-allocated proxies are all used, MOTAG returns to creating the containers just when they are needed. Pre-allocation is useful because the creation and removing of containers is time-intensive which could mean that mitigations are more effective when more proxies are pre-allocated.

I planned on making the number of additional proxies per round configurable instead of always being one, but this lead to problems. Specifically, in the most common case, only one insider is assumed which means that the greedy algorithm will always distribute all clients along only two proxies with the exception of an uneven number of clients where a single client will be assigned to a third proxy. This of course is unexpected behavior and I therefore removed the option.

MOTAG does not include a remerging strategy.

6.9. Attack patterns

I implement both simple and sophisticated attacker patterns that potentially have insider information on the mitigation strategies and configuration options. In this section, I will give an overview over how these attack patterns work. Note that attack patterns that take advantage of mitigations remerging clients will only be tested with Mole Hunt, not with MOTAG.

6.9.1. AllAtOnce

This is the most straightforward attack pattern. It simply attacks all the network addresses that are known to it through the insiders.

Because of the abstraction and API that dmef provides, such a simple attack pattern can be implemented in dmef using only the four lines shown in Listing 6.4.

Listing 6.4: allatonce.py

```
from ..attack import BaseAttack

class allatonce(BaseAttack):
    def run_round(self, round_no):
        self.attack(list(map(lambda x: self.dmef.routing[x], self.insiders)))
```

6.9.2. OneAtOnce

This attack pattern works similarly, but instead of attacking all known addresses at once, it only attacks one of the known addresses per round. This increases the number of rounds required and therefore also the number of used addresses because the mitigation learns less about the insiders per round.

6.9.3. EvadeIsolation

This is a more sophisticated attack pattern. It requires knowledge about the mitigation, the configured splitsize and an approximation of the number of clients connected to the edge cloud. Based on this information, the pattern calculates the number of rounds it takes the mitigation to identify an insider and then attacks one round less, then waits for the mitigation to remerge to repeat the pattern. Since this is useless without remerging it is only tested on Mole Hunt. On Mole Hunt the number of rounds required to identify an insider is at least:

$$rounds_{min} = \lfloor \log_{Splitsize} NumberOfClients \rfloor + 1$$

because n-ary search takes $\log_n$ many steps to isolate and then one more round where only one client is connected to the address where it is attacked to identify.

After this calculation, the attack pattern then attacks one known address per round. When the address an insider is connected to is attacked, its counter is advanced by one even if this means multiple insider's counters are advanced. When the counter of an insider hits the pre-calculated maximum, its address is no longer attacked and when the attack pattern has no more addresses to attack, it waits for remerging. After remerging, it resets all insiders counters to one, not zero because there are already two groups not a single one as in the beginning which is equivalent to one binary search step already done.

This does not go on forever however because the remerging strategy does not remove all progress made in identifying the insiders and so they are identified eventually. The number of rounds this attack pattern can survive varies more than the simpler ones because the random choices in insiders and assignments have larger consequences. For example, after the first remerge, the decision whether an insider is assigned in the group of 6 or the group of 7 might result in the insider being identified or not, leading to an additional 4 or rounds or not.

6.9.4. SwitchingSets

This attack pattern expands on the idea of the EvadeIsolation pattern. Instead of only knowing how the mitigation and splitting process works, SwitchingSets also knows how the remerging process works and evades it. This is done by dividing the insiders into two sets and then only attacking with one set until Mole Hunt remerges, then switching the set of insiders until remerge and so on. Once again the knowledge of how often attackers can attack before insiders get identified and wait for remerging when it becomes to dangerous is applied.

Having two sets means that the attack pattern can attack less often between remerges since it has fewer insiders to work with, but because the other set of insiders did not attack at all before a remerge, these clients are considered innocent and will most-likely be placed in the large low risk group during a remerge. This means it can attack more often because the insiders are in a larger group of clients.

In both EvadeIsolation and SwitchingSets the attack maximum is $rounds_{min} - 1$ before the first remerge, but while the maximum drops to $rounds_{min} - 2$ for EvadeIsolation after the first remerge, it increases to $rounds_{min}$ for SwitchingSets. If I were to keep the maximum at $rounds_{min} - 1$ for SwitchingSets, my tests show that the attack pattern would be able to attack forever and never get identified.

7. Experiments

In this chapter I am going to use my new framework to evaluate the implemented DDoS mitigations for edge clouds as well as the different configuration options.

7.1. Expectations

After I have conducted the experiments I want to check whether the results match theoretical expectations. This allows an evaluation whether the results are logically consistent and the framework works correctly. To do this, I present and explain ten hypotheses in this section based on theoretical knowledge to be confirmed by the results later.

H1 The number of iterations made for each set of configuration options (30) is sufficient.

Since randomness is involved in multiple situations during an iteration of the framework such as choosing the insiders, the metrics of iterations are expected to vary. The more iterations are done, the more stable the results which are the mean of all iterations will be. The number of iterations is considered sufficient when the result's variation when a new iteration is added is negligible which I expect to be after a maximum of 30 iterations. This is also a result of the reproducibility requirement (**D010**).

H2 The distribution of required rounds matches theoretical calculations.

If the algorithms of the mitigations are implemented correctly, the number of rounds it takes the algorithms to identify insiders in the framework should match theoretical calculations.

H3 More clients increase the number of required rounds and addresses.

Since more clients means more possible insiders, the algorithms require more rounds to identify them. More rounds also always means more addresses.

H4 In Mole Hunt, a larger splitsize reduces required rounds but increases used addresses.

A larger splitsize means less clients per address which means that when an address is attacked there are less suspects which means that less rounds are required to identify an insider. A larger splitsize also means that more new addresses are announced each round which means more used addresses.

H5 In MOTAG, more initial proxies reduce the number of required rounds.

More initial proxies means that there are less suspected clients the first time a proxy is attacked which means less rounds are required.

H6 In MOTAG, more pre-allocated proxies increase performance.

Since creating and removing proxies takes time, moving the proxy starting process to the beginning increases performance during a round where usually a new proxy would have to be created.

H7 More sophisticated attackers perform better.

Since the sophisticated attackers take account of how the mitigation works and try to outmaneuver with their knowledge, they are expected to perform better than the trivial attack patterns.

H8 For both MOTAG and Mole Hunt, twis% is consistent across configuration options.

Different configuration options mostly effect the number rounds required to identify the insiders. During each round, the edge cloud is attacked and time with interrupted service (twis) is detected which means that an increasing number of rounds also means an increase in twis. An increasing number of rounds however also means that the entire runtime increases. I expect that twis per round is consistent for a specific mitigation as it is based on the technology used and does not change with any of the configuration options. Therefore I expect twis% to be consistent across configuration options.

H9 Mole Hunt and MOTAG's algorithm perform comparably, but Mole Hunt's technology is superior.

Both algorithms essentially perform a binary search for the insiders which means that perform similarly. Only MOTAG's algorithm performs differently when multiple proxies are attacked at the same time which only happens with one of the attack patterns. On the technology side I expect that routing traffic through proxies, creating and removing proxies creates additional overhead that creates additional time with interrupted service.

Option	Default	Used Values
General options		
Number of Clients	25	• 15 • 20 • 25 • 30 • 35 • 40
Number of Insiders	3	• 3
Number of Attackers	3	• 3
Attack Pattern	OneAtOnce	• OneAtOnce • AllAtOnce • EvadeIsolation (Mole Hunt only) • SwitchingSets (Mole Hunt only)
Mole Hunt options		
Splitsize	2	• 2 • 3 • 4 • 5
MOTAG options		
Initial Proxies	1	• 1 • 2 • 3 • 4
Pre-allocated proxies	0	• 0 • 5 • 10 • 15

Table 7.1.: Configuration options

7.2. Methodology

Because a single iteration usually takes about 20 minutes to run and there are lots of configuration options, I am not able to run every combination of options. Instead, I run both Mole Hunt and MOTAG with default options and then vary all the options, but only one option at a time. This means in every configuration run all options are set to their default except for one. This allows us to observe the results of every configuration option with a greatly reduced number of iterations required.

Table 7.1 shows all the configuration options including the different values tested and the default value. I run every configuration for at least 30 iterations so that individual differences in runs only have a small impact.

7.3. Results

The full results of all runs and iterations can be found in the Appendix. In this section, I want to show the default benchmark and compare it against other settings. Note that whenever an errror bar is shown, it represents the standard deviation.

7.3.1. Mole Hunt

The full default benchmark for Mole Hunt (25 Clients and 3 Insiders running OneAtOnce-Attack Pattern and Splitsize 2) can be found in Figure 7.1. You can see that rounds, addresses and twis vary significantly between iterations but match within one iteration e.g. for one iteration they are all on the high end or all on the low end. You can also see that all variations of these metrics are within a limited range and their standard deviation is also limited. The twis% value is much more consistent across iterations. These variations of the metrics are also shown in Figure 7.2. I compare different splitsizes in Figure 7.3. You can see that rounds and twis decreases with increasing splitsize while twis% remains consistent and the addresses metric does not show a clear trend. Different number of clients are shown in Figure 7.4 where you can see that increased number of clients result in increased number of rounds, addresses and twis while twis% remains consistent. The different attack patterns that were implemented are compared in Figure 7.5. You can see that rounds, addresses and twis increases with attack pattern sophistication while twis%' trend is inconsistent.

7.3.2. MOTAG

In all MOTAG results, the information in brackets refers to the number of inital proxies and the number of pre-allocated proxies, in this order. The default benchmark, which is used as a basis for all comparisons, uses one initial proxy and has no pre-allocated proxies (1-0). Different numbers of inital proxies are compared in Figure 7.6. You can see that an increased number of initial proxies results in a reduced number of rounds, addresses and twis while twis% is consistent. Different numbers of pre-allocated proxies are compared in Figure 7.7. You can see that all metrics do not show a clear trend and remain rather consistent across different values for pre-allocated proxies. Different number of clients are compared in Figure 7.8 where you can see an increased number of rounds and addresses for an increased number of clients while twis and twis% do not show a trend. The different attack patterns on MOTAG are compared in Figure 7.9. You can see that MOTAG uses more addresses in less rounds and with a higher twis when attacked using the AllAtOnce attack pattern than with the OneAtOnce attack pattern.

7.3.3. Comparison

A comparison of Mole Hunt and MOTAG is done in Figure 7.10. There, I also added a version of MOTAG that uses address announcement and withdrawal using exaBGP like Mole Hunt but still uses the MOTAG greedy shuffling algorithm. You can see that the number of required rounds and the number of used addresses is very similar across mitigations while for twis and twis% only Mole Hunt and MOTAG-noproxy show similar values and MOTAG shows much higher values.

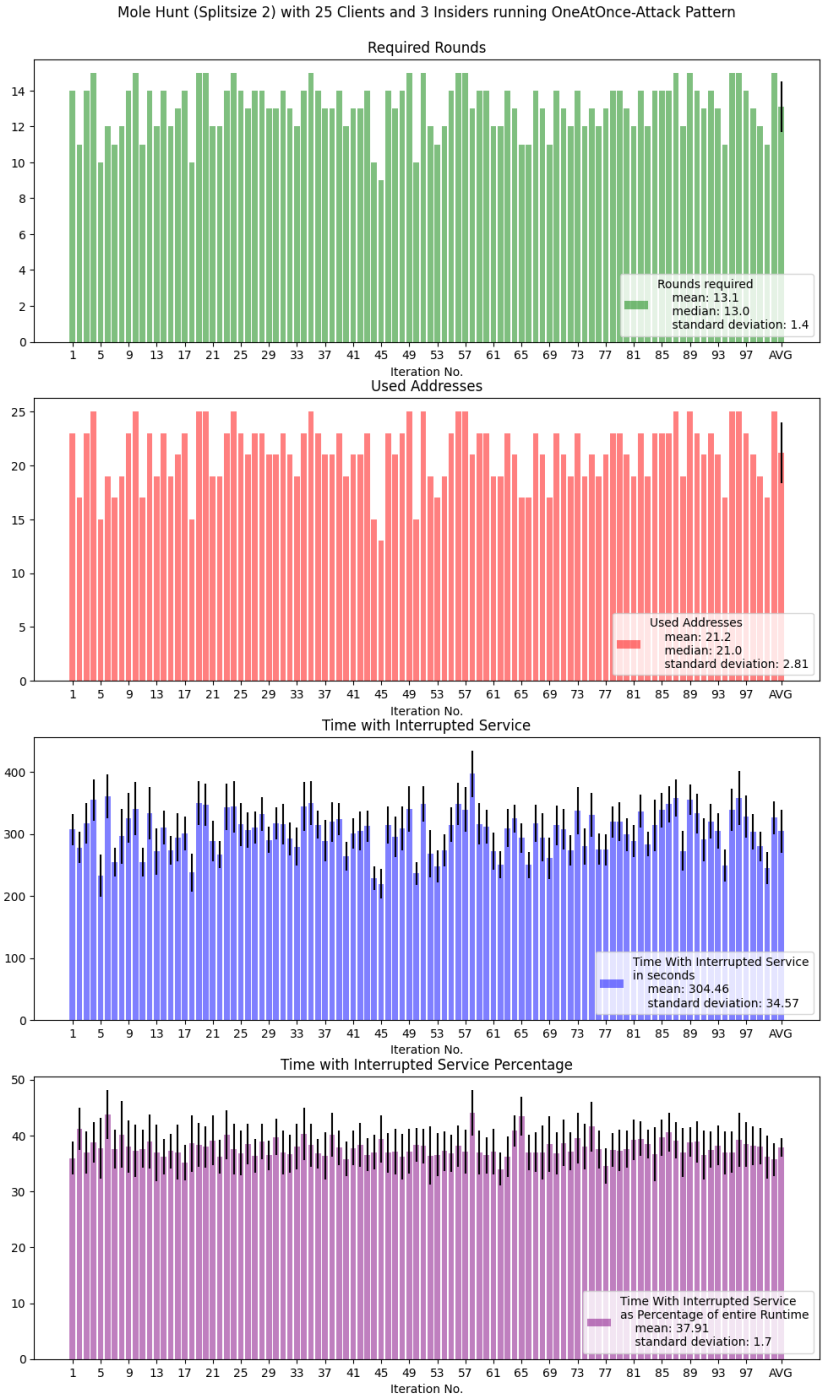


Figure 7.1.: Mole Hunt (Split Size 2) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

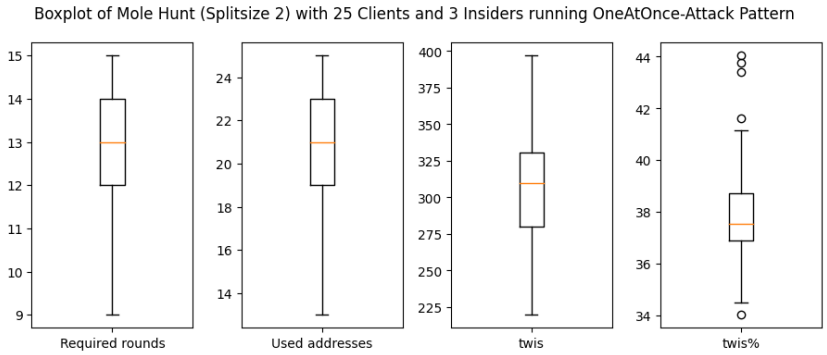


Figure 7.2.: Boxplot of the metrics of the standard benchmark

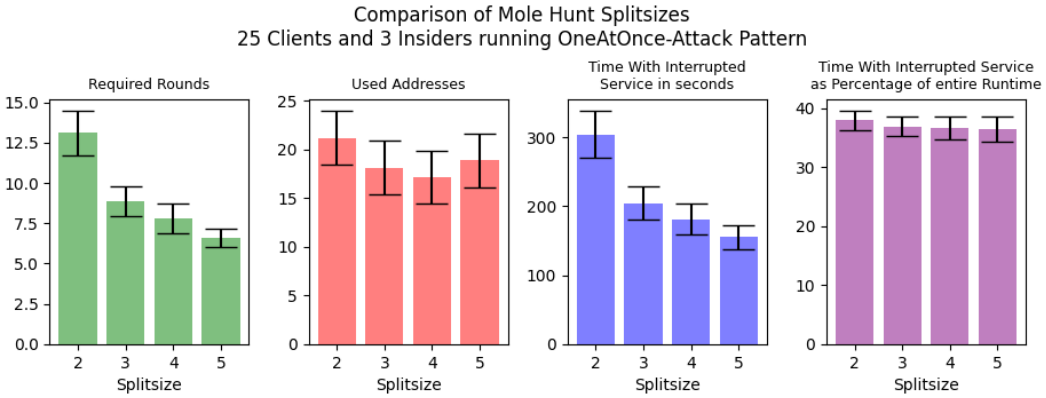


Figure 7.3.: Comparing Mole Hunt Splitsizes

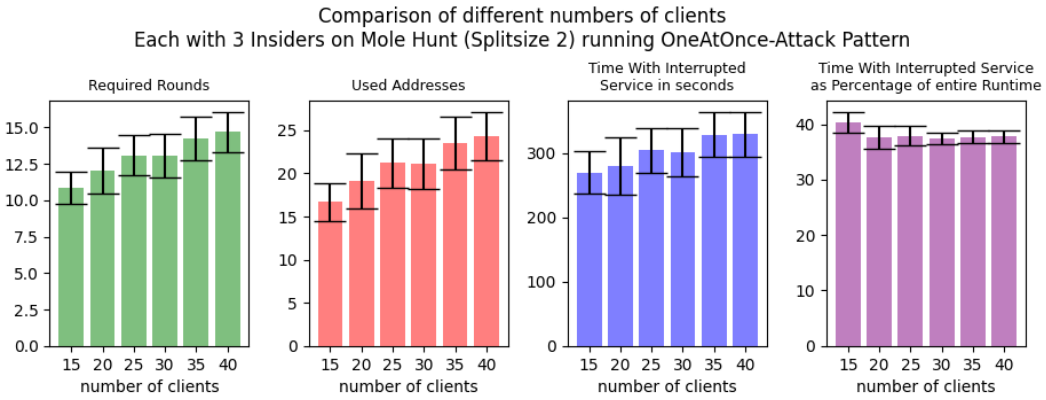


Figure 7.4.: Comparing number of clients in Mole Hunt

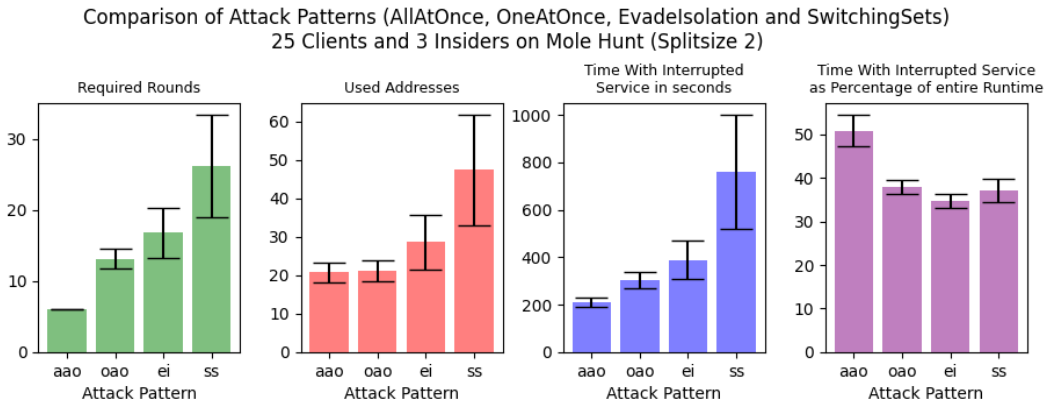


Figure 7.5.: Comparing Attack Patterns on Mole Hunt

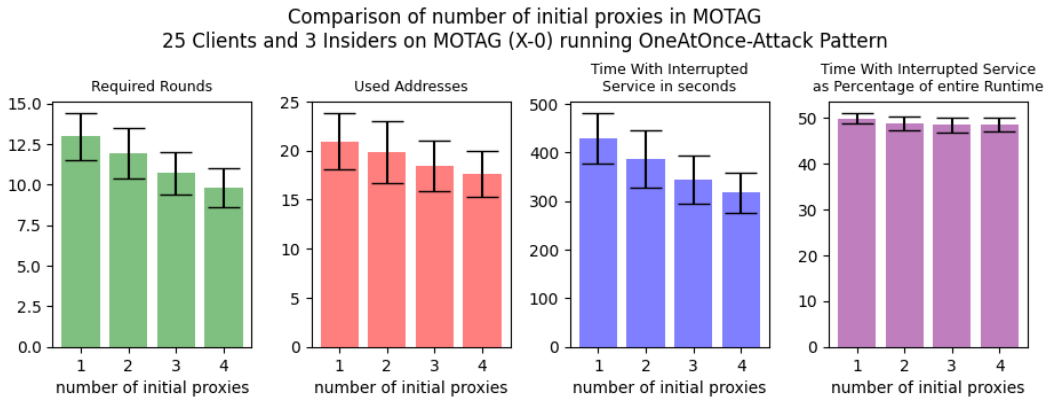


Figure 7.6.: Comparing different numbers of inital proxies on MOTAG

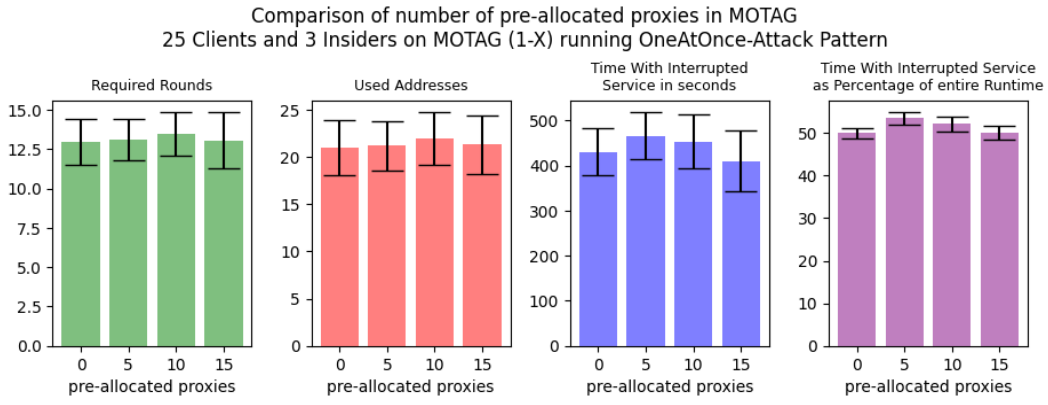


Figure 7.7.: Comparing different numbers of pre-allocated proxies on MOTAG

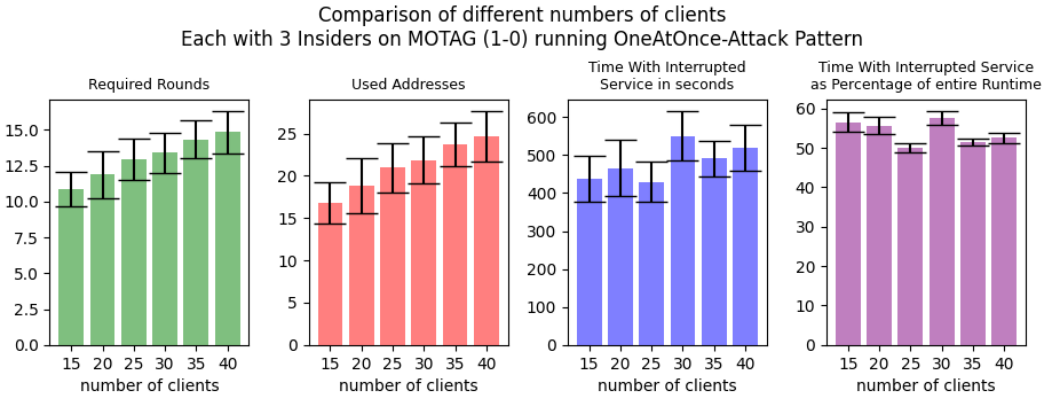


Figure 7.8.: Comparing different numbers of clients on MOTAG

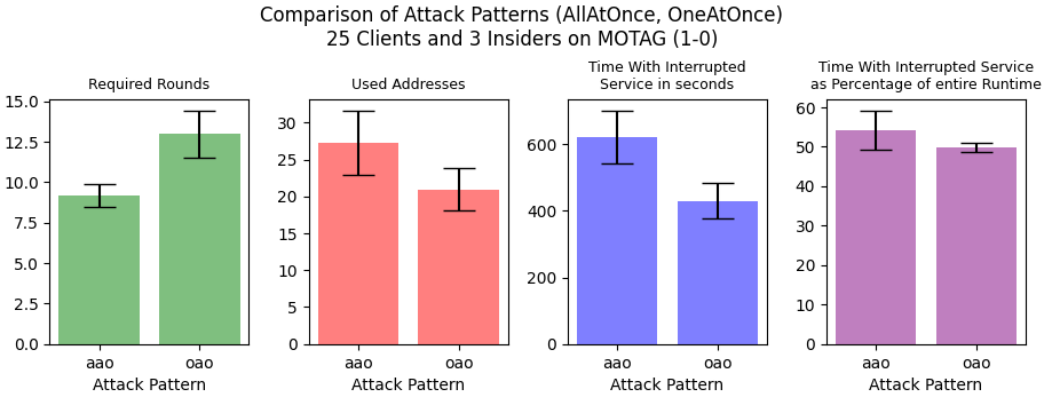


Figure 7.9.: Comparing Attack Patterns on MOTAG

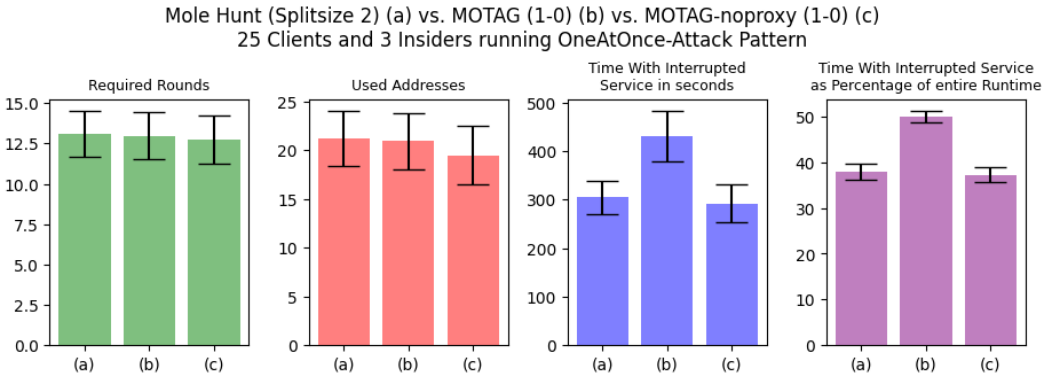


Figure 7.10.: Comparing Mole Hunt, MOTAG and a no-proxy MOTAG variant

8. Evaluation

After having designed, implemented and used my new framework for DDoS mitigation evaluation in the last chapters, I want to evaluate my framework in this chapter. To do this, I will examine whether my results are logically consistent and match my theoretical expectations and will compare dmef to related work, namely other methodology for DDoS mitigation evaluation and other results.

8.1. Expectation evaluation

In this section I will examine the previously recorded hypotheses about the results expected which were based only on the theoretical knowledge. When a hypothesis matches the actually measured results this means that my framework and methodology was logically consistent and matches theoretical expectations. When a hypothesis does not match my results, I evaluate whether my theoretical thinking was incorrect and why or whether my framework does not perform as expected.

H1 The number of iterations made for each set of configuration options (30) was sufficient.

To asses this, I ran the default benchmark for both Mole Hunt and MOTAG for 100 iterations rather than the standard 30. Figure 8.1 shows the deviation of my measured metrics from the final metric. This is done by calculating the average of the metric after each iteration only including the previous iterations. This means that for every iteration, I now know how much the mean would have been off if I would have had stopped after this iteration. I can see that the values are very stable after 30 iterations with only minor variations.

H2 The distribution of required rounds matches theoretical calculations.

Without loss of generality, I examine the default benchmark for Mole Hunt and I will examine the theoretical distribution of the number of required rounds to identify all insiders and compare it to the measured results. The first insider is isolated after four or five rounds ($log_2 25 \approx 4.64$) and is identified and blocked after an additional round of being the single connected client on an attacked address. To look at the probabilities of four or five rounds as well as the remaining group sizes in which the other two insiders are, see Figure 8.2.

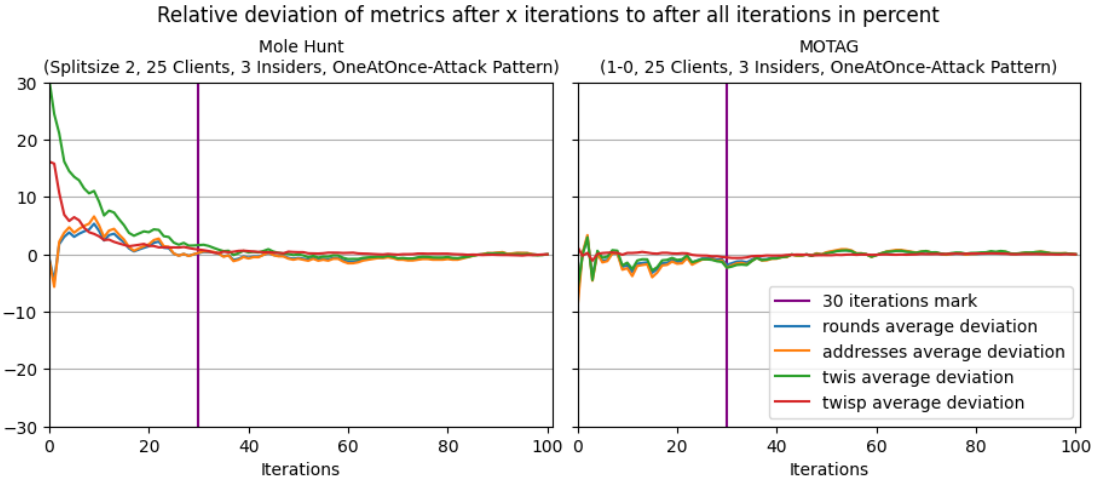


Figure 8.1.: Deviation of metrics after x iterations to after all iterations

I find that the first insider is isolated after round 4 in 43.75% of cases and after round 5 in 56.25% of cases. I also find that based on the group sizes of the remaining clients after the first insider is isolated and identified, the second and third insider have a high probability of still being in a large group of clients. This means that higher numbers of required rounds to identify all insiders have a higher probability. At the same time, it is likely that after the first two insiders have been identified, the third insider was grouped together with another one at least once where it did not necessarily have to which results in the most-likely number of rounds required to identify all three insiders being 14 while the maximum is 15. The minimum number of required rounds is 8 although this is highly unlikely because it means that all three insiders were grouped together as long as possible. In conclusion, my theoretical analysis shows that I expect the number of required rounds to be in the interval $[8, 15]$ with 14 being the most-likely number of required rounds.

The actual measured distribution is shown in Figure 8.3. I find that it matches my expected results with the only difference being that the minimum of 8 rounds was never hit which is not surprising since this case is highly unlikely.

H3 More clients increase the number of required rounds and addresses.

I find in my results, that this is true for both Mole Hunt (Figure 7.4) as well as MOTAG (Figure 7.8).

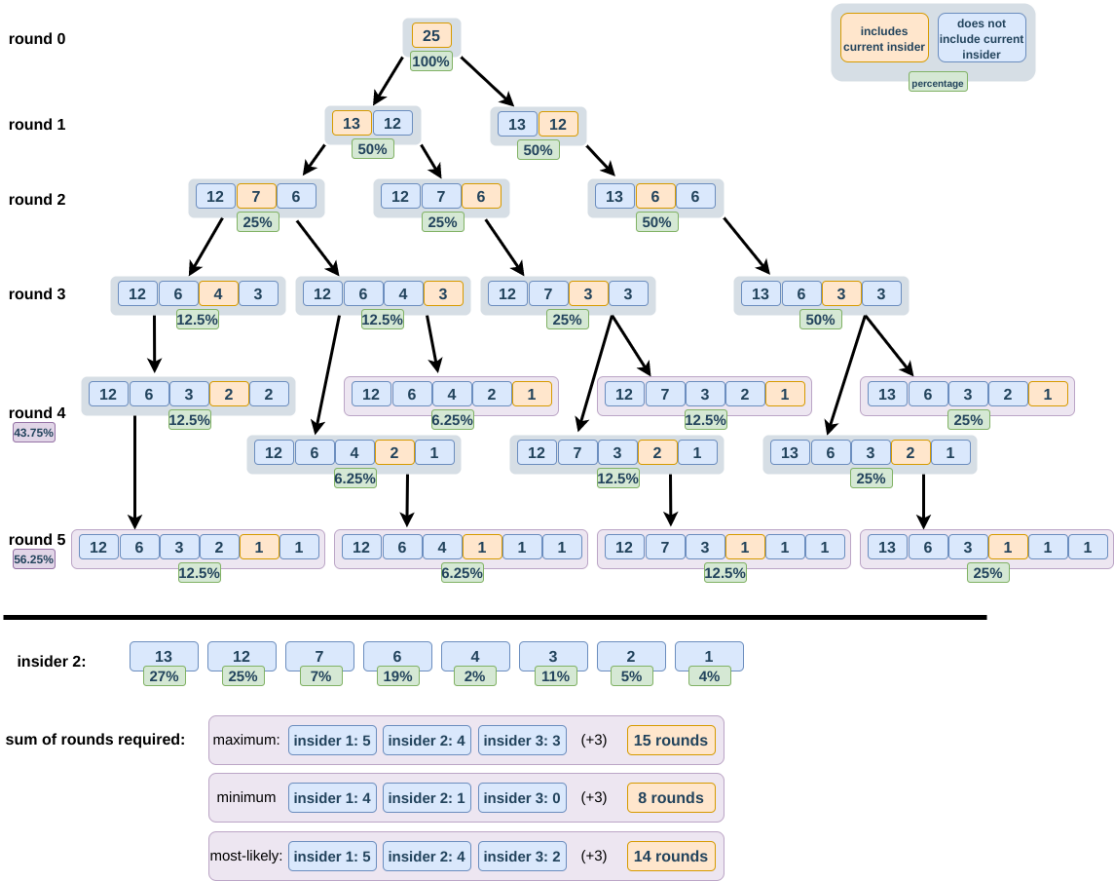


Figure 8.2.: Analysis of the number of required rounds

The first part of the figure shows the process of the first insider being identified. Each orange or blue bubble represents a group of clients that uses the same network address for the edge cloud, the number in the bubble refers to the number of clients in this group. An orange bubble represents the group of clients that the first insider is part of. The surrounding gray or purple bubble represents a possible state in the insider identification process. A purple bubble means that the insider has been isolated in this state. The second part of the figure shows the probabilities for different group sizes that the second insider is part of after the first insider is identified. The number within the blue bubble refers to the number of clients in the group and the green number is the probability of the second insider being in a group of this size. The last part of the figure concludes the overall number of required rounds to identify all three insiders.

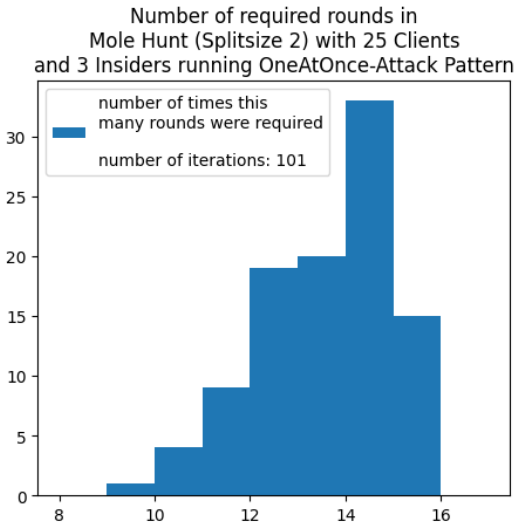


Figure 8.3.: Histogram of required rounds in Mole Hunt (Split Size 2) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

H4 In Mole Hunt, a larger splitsize reduces required rounds but increases used addresses.

I find that based on my results in Figure 7.3 that while larger splitsizes reduces the number of required rounds there is no obvious trend for the number of used addresses. Contrary to my expectation, I find that up to a splitsize of 4, the number of used addresses actually also decreases because a reduced number of rounds means there are fewer occurrences of addresses being announced. Only for a splitsize of 5 I find the expected trend. I have to note though, that this is heavily influenced by the relatively small number of clients in my test cases and I expect that the splitsize with the lowest number of used addresses varies by the number of clients and is not always 4.

H5 In MOTAG, more initial proxies reduce the number of required rounds.

I find this is true based on Figure 7.6.

H6 In MOTAG, more pre-allocated proxies increase performance.

This is incorrect based on Figure 7.7. There I cannot find any trend but rather all results are within margin of error. This means that the delay it takes to setup another proxy using docker is too small to be noticeable. Since actual implementations of MOTAG would not use docker to facilitate proxies, I recommend testing pre-allocation of proxies on the target environment to see if the delay is noticeable there.

H7 More sophisticated attackers perform better.

I find this is true for Mole Hunt based on Figure 7.5. For MOTAG however, OneAtOnce performs worse in used addresses and twis, while better in required rounds than AllAtOnce from the viewpoint of the attacker, see Figure 7.9. Although I would not necessarily call any of those two patterns sophisticated, this shows that MOTAG's algorithm handles attack scenarios with multiple insiders at the same time very differently, using a lot more addresses but requiring fewer rounds.

H8 For both MOTAG and Mole Hunt, twis% is consistent across configuration options.

I find that this is true with exceptions. Mole Hunt almost always achieves twis% values of 38-40%, see Figure 7.1, Figure 7.3, Figure 7.4 and Figure 7.5. MOTAG's twis% is continuously between 48% and 55%, see Figure 7.6, Figure 7.7, Figure 7.8 and Figure 7.9. Only in the case of the AllAtOnce attacker, I find that twis% is higher because the overall runtime is very low. This applies to both mitigations.

H9 Mole Hunt and MOTAG's algorithm perform comparably, but Mole Hunt's technology is superior.

I find this is is true based on Figure 7.10. I examine the algorithms performance based on the number of required rounds and the number of used addresses which I find to be very similar across Mole Hunt and MOTAG. At the same time, I find that twis and twis% are much higher for MOTAG than for Mole Hunt, showing that Mole Hunt's technology with announcing and withdrawing addresses via exaBGP is superior to MOTAG's proxies. I can also see this based on the MOTAG-noproxy variant that uses MOTAG's algorithm but Mole Hunt's technology. MOTAG-noproxy performs very similarly to Mole Hunt, showing that the algorithms have similar results, but Mole Hunt's technology reduces Time with Interrupted Service.

8.2. Framework methodology

In this section I want to compare how dmef works to other DDoS mitigation evaluations, namely [22] and [15].

In [22], the authors include a number of requirements that DDoS mitigation evaluations have to fulfill before presenting their system that matches these requirements. I want to first check whether dmef also fulfills the named requirements and then examine how some requirements were solved differently.

A first named requirement is that test scenarios are realistic and comprehensive [22, p. 1f]. In dmef this was one of the main focus points which I solved by basing its

implementation on the key characteristics of the edge cloud environment based on theoretical knowledge and planned implementations to make dmef realistic. I create my comprehensive tests by adding a variety of attack patterns from trivial to sophisticated with insider knowledge. Another requirement are accurate and expressive performance metrics [22, p. 2]. I solved this by measuring the number of required rounds which allows an assessment on how quickly the mitigations defend against attacks and the number of used addresses as the operational cost the mitigations require. Additionally, the QoS metrics twis and twis% give accurate information on how damaging an attack is and how well an attack is mitigated. The last named requirement is that the evaluation methodology includes guidelines to describe a system's security [22, p. 2]. While this requirement is important for collaborative defenses, it is not an issue for the mitigations evaluated in this thesis. Still, the attack patterns that take advantage of insider information allow us to also evaluate the mitigations when their security might be compromised.

The authors of [22] solve the requirement for realistic test scenarios by creating automatic tools that extract the necessary information from real world infrastructures and attacks. This includes attack traffic and legitimate traffic and network topologies. While this, if it works, definitely creates a very realistic and comprehensive test scenario, this was not necessarily the goal of dmef. For dmef the focus was on edge clouds which is an infrastructure that does not yet exist in the real world. I also explained why dmef does not require very realistic attack traffic in chapter 6. And while dmef's legitimate traffic is not very realistic, this would have been impossible because of the variety of edge services that will be running on edge clouds.

For their performance metric, the authors of [22] examine high-level tasks that legitimate users want to perform and whether their QoS requirements can be met by the service under attack. The metric is the percentage of those transactions that do not meet QoS requirements and that are therefore considered failed. Because this is an application level metric, it would not have been possible to be implemented on dmef. Additionally, it assumes that during attacks at least some transactions succeed which my tests showed was not the case on dmef with the modeled edge cloud environment where during attacks, no legitimate traffic would be received.

In [15], the authors name five areas to carefully design for an effective evaluation of DDoS defenses and show their findings in these areas. Firstly, in the area of attack mechanisms, the authors recommend to use one of some named scientific attack generation tools. While dmef does not use one of the named tools, it uses a more recent IPv6-ready tool that offers comparable features. Secondly, a number of tools for the generation of legitimate traffic are recommended, but those sophisticated tools would not have been possible on dmef due to resource constraints as well as not bringing any advantage over dmef's simple legitimate traffic. Thirdly, network topology is mentioned as an important area to design which in the case of dmef is given by the edge cloud environ-

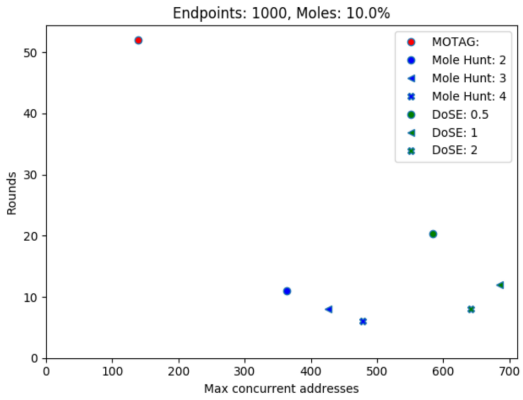


Figure 8.4.: Evaluation of Mole Hunt [12, p. 5]

ment which is modeled. Fourthly, the authors recommend that the defenses should be challenged with specifically crafted attacks that attack their weak points which is done in dmef using the sophisticated attack patterns with insider knowledge. Last but not least, [15] analyzes used metrics in DDoS defense evaluation and recommends combining multiple traffic base metrics into a meaningful DDoS-centric metric that is observed to degrade during attacks which is exactly what I do in dmef with twis and twis%.

8.3. Results comparison

In this section I want to examine how my results stack up against the results from the mitigations' self-evaluations. All of these self-evaluations only consider simulations of the algorithms and do not consider the implementation. Therefore I only have results for required rounds and used addresses and not for QoS metrics. Additionally, in those simulations, a much larger number of clients is assumed, making comparisons hard.

MOTAG's self-evaluation assumes that insiders always attack, as in dmef's AllAtOnce attack pattern [19, p. 6f]. Contrary to my findings, the evaluation assumes that an attacked proxy means that only clients connected to this proxy experience a DoS and not every client connected to the application. This also means that clients can be saved from the attack before an insider is finally identified. That is why MOTAG's self-evaluation shows the number of required shuffles to save 80% or 95% of clients not 100% like in dmef. Nevertheless, I find that the trends shown in the self-evaluation of MOTAG align with the trends observed in my results. Additionally, MOTAG evaluates the overhead introduced by proxies, but since this assumes proxies are geographically distributed from each other as well as the application, this is not comparable to my setup and would defeat the purpose of edge clouds.

Mole Hunt's self-evaluation also assumes that attackers attack all addresses known to insiders every round, as in dmef's AllAtOnce attack pattern [12]. Their evaluation also measures maximal concurrently used addresses rather than overall used addresses like in dmef. When I compare Figure 7.3 and their results for Mole Hunt, see Figure 8.4 [12, p. 5], I find that both absolute values as well the trends match. However, I do not see similar results for MOTAG. While I find similar results in terms of algorithm performance for Mole Hunt and MOTAG in dmef, see Figure 8.4, Mole Hunt's self-evaluation [12, p. 5] shows very different results. This is because MOTAG's greedy algorithm gets provided with an assumption about the number of insiders. In Mole Hunt's evaluation this assumption is always set to 1, assuming the mitigation's worst situation, even when multiple addresses get attacked at the same time which is common in the AllAtOnce attack pattern. In dmef this assumption is instead set to the number of attacked addresses since this is lower bound for the number of insiders. This results in a much more plausible and realistic performance of MOTAG in dmef's evaluation.

9. Conclusion

In this chapter, I want to present the conclusions that can be drawn from my newly created DDoS mitigation evaluation framework.

When comparing the two implemented DDoS mitigations, Mole Hunt and MOTAG, I find that Mole Hunt is decisively superior. In fact, Mole Hunt shows a variety of advantages and no obvious disadvantages over MOTAG. I find that on the technology side, announcing and withdrawing addresses using exaBGP instead of using proxies that forward traffic not only saves computing power and makes it more edge cloud friendly, it also reduces the effects of DDoS attacks, namely degraded quality of service. On the algorithm side, I find that in my test cases, the two shuffling algorithms behaved very similarly not giving an advantage to any of the two defense strategies. Additionally, Mole Hunt's splitsize configuration option allows the mitigation to be more tuned towards a specific use case, trading required rounds versus used addresses. I however propose MOTAG to be adapted to also support a configurable number of additional proxies per round instead of being fixed to one per round.

Mole Hunt also performs fairly well against sophisticated attackers, but it also shows that its remerging strategy still needs to be improved. It can very well be argued that when a sufficiently long timeout before remerging is in place, any attack pattern trying to evade isolation would be very limited. Also, the predictions this kind of attack pattern has to make are very difficult in dynamic real world scenarios, requiring the pattern to be very conservative in its estimation, limiting attacks. Still, the fact that a slightly more conservative SwitchingSets attacker would be able to continue attacking infinitely is a sign for concern. Possible improvements could include randomly varying the splitsize to make predictions about the number of rounds until isolation harder or adapting the remerging strategy to pass along more information and potentially at some point ban clients that are not 100% guaranteed to be insiders but only with a high probability.

In general, I find that both Mole Hunt and MOTAG work well to eventually stop DDoS attacks against edge clouds. During the mitigation process however, clients still experience significant quality of service degradation. This is mostly caused by the delay in attack detection which means that a very fast and precise attack detection mechanism is favorable for use in edge clouds. I find that the migration process from one network address to another is fast and seamless, especially using Mole Hunt.

In a nutshell, moving target defenses like Mole Hunt are a promising DDoS defense strategy on edge clouds, but there are still some improvements to be made.

Bibliography

[1] Amazon Web Services (AWS). *Amazon CloudFront Key Features*.
URL: https://aws.amazon.com/cloudfront/features/.
(accessed: 26th May 2020).

[2] Amazon Web Services (AWS). *AWS Wavelength*.
URL: https://aws.amazon.com/wavelength/. (accessed: 26th May 2020).

[3] N. Agrawal and S. Tapaswi. "Defense Mechanisms Against DDoS Attacks in a Cloud Computing Environment: State-of-the-Art and Research Challenges". In: *IEEE Communications Surveys Tutorials* 21.4 (Fourthquarter 2019), pp. 3769–3795. ISSN: 2373-745X. DOI: 10.1109/COMST.2019.2934468.

[4] Akamai. *The Akamai Intelligent Edge Platform*.
URL: https://www.akamai.com/us/en/what-we-do/intelligent-platform/.
(accessed: 26th May 2020).

[5] CZ.NIC Labs et al. *BIRD internet routing daemon*.
URL: https://bird.network.cz/. (accessed: 10th July 2020).

[6] Exa Networks Limited et al. *ExaBGP*.
URL: https://github.com/Exa-Networks/exabgp. (accessed: 10th July 2020).

[7] Jeremy Lainé et al. *aioquic*. URL: https://github.com/aiortc/aioquic.
(accessed: 10th July 2020).

[8] Pablo Neira Ayuso et al. *nftables*.
URL: http://netfilter.org/projects/nftables/index.html.
(accessed: 10th July 2020).

[9] Hyunseok Chang et al. "Bringing the cloud to the edge". In: *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2014, pp. 346–351.

[10] Cloudflare. *Cloudflare CDN*. URL: https://www.cloudflare.com/cdn/.
(accessed: 26th May 2020).

[11] Fastly. *More than a CDN*. URL: https://www.fastly.com/.
(accessed: 26th May 2020).

[12] Simon Hanisch and Thorsten Strufe.
Mole Hunt: A DDoS mitigation for Edge Clouds. Tech. rep.
Center for Tactile Internet, TU Dresden, Feb. 2020.

[13] van Hauser. *THC IPv6 attack toolkit*.
URL: https://github.com/vanhauser-thc/thc-ipv6. (accessed: 6th June 2020).

[14] Stephen Hemminger and Alexey Kuznetsov. *iproute2*.
URL: https://wiki.linuxfoundation.org/networking/iproute2.
(accessed: 19th July 2020).

[15] Alefiya Hussain et al. "DDoS experiment methodology". In: *Proceedings of the DETER community workshop on cyber security experimentation*. Vol. 8. 2006.

[16] Cloudflare Inc. *High Orbit Ion Cannon*.
URL: https://www.cloudflare.com/learning/ddos/ddos-attack-tools/high-orbit-ion-cannon-hoic/. (accessed: 10th July 2020).

[17] Docker Inc. *Docker*. URL: https://docker.com. (accessed: 10th July 2020).

[18] Ed. J. Iyengar and Ed. M. Thomson.
QUIC: A UDP-Based Multiplexed and Secure Transport.
URL: https://tools.ietf.org/html/draft-ietf-quic-transport-27.
(accessed: 10th July 2020).

[19] Q. Jia, K. Sun, and A. Stavrou.
"MOTAG: Moving Target Defense against Internet Denial of Service Attacks".
In: *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. July 2013, pp. 1–9. DOI: 10.1109/ICCCN.2013.6614155.

[20] Q. Jia et al. "Catch Me If You Can: A Cloud-Enabled DDoS Defense".
In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. June 2014, pp. 264–275.

[21] V. Kansal and M. Dave. "DDoS attack isolation using moving target defense".
In: *2017 International Conference on Computing, Communication and Automation (ICCCA)*. May 2017, pp. 511–514. DOI: 10.1109/CCAA.2017.8229853.

[22] J. Mirkovic et al. "Benchmarks for DDOS Defense Evaluation".
In: *MILCOM 2006 - 2006 IEEE Military Communications conference*. Oct. 2006, pp. 1–10. DOI: 10.1109/MILCOM.2006.302006.

[23] Jelena Mirkovic and Peter Reiher.
"A taxonomy of DDoS attack and DDoS defense mechanisms".
In: *ACM SIGCOMM Computer Communication Review* 34.2 (2004), pp. 39–53.

[24] Jianli Pan and James McElhannon.
"Future edge cloud and edge computing for internet of things applications".
In: *IEEE Internet of Things Journal* 5.1 (2017), pp. 439–449.

[25] David P. Reed and J. Postel. *User Datagram Protocol*.
URL: https://tools.ietf.org/html/rfc768. (accessed: 10th July 2020).

[26] The Internet Society. *A Border Gateway Protocol 4 (BGP-4)*.
URL: https://tools.ietf.org/html/rfc4271. (accessed: 10th July 2020).

[27] The Internet Society. *OSPF Version 2*.
URL: https://tools.ietf.org/html/rfc2328. (accessed: 10th July 2020).

[28] Gaurav Somani et al.
"DDoS attacks in cloud computing: Issues, taxonomy, and future directions".
In: *Computer Communications* 107 (2017), pp. 30–48.

[29] Liang Tong, Yong Li, and Wei Gao.
"A hierarchical edge cloud architecture for mobile computing".
In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, pp. 1–9.

[30] Cisco/Cybersecurity Ventures. *2019 Cybersecurity Almanac*.
URL: https://cybersecurityventures.com/cybersecurity-almanac-2019/.
(accessed: 26th May 2020).

[31] P. Wood, C. Gutierrez, and S. Bagchi.
"Denial of Service Elusion (DoSE): Keeping Clients Connected for Less".
In: *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*. Sept. 2015, pp. 94–103. DOI: 10.1109/SRDS.2015.31.

[32] Gala Yadgar et al. "Modeling The Edge: Peer-to-Peer Reincarnated".
In: *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*. 2019.

A. Appendix

A.1. Full Results

This section includes the results of every iteration of every run made with dmef that was used in chapter 7, chapter 8 or chapter 9. The results are ordered by mitigation name (Mole Hunt, MOTAG or MOTAG-noproxy), their configuration options (splitsize or configuration tuple), the attack pattern (AllAtOnce, EvadeIsolation, OneAtOnce and SwitchingSets) and the number of clients, each in ascending order. See the List of Figures at the beginning of this thesis for a table of contents.

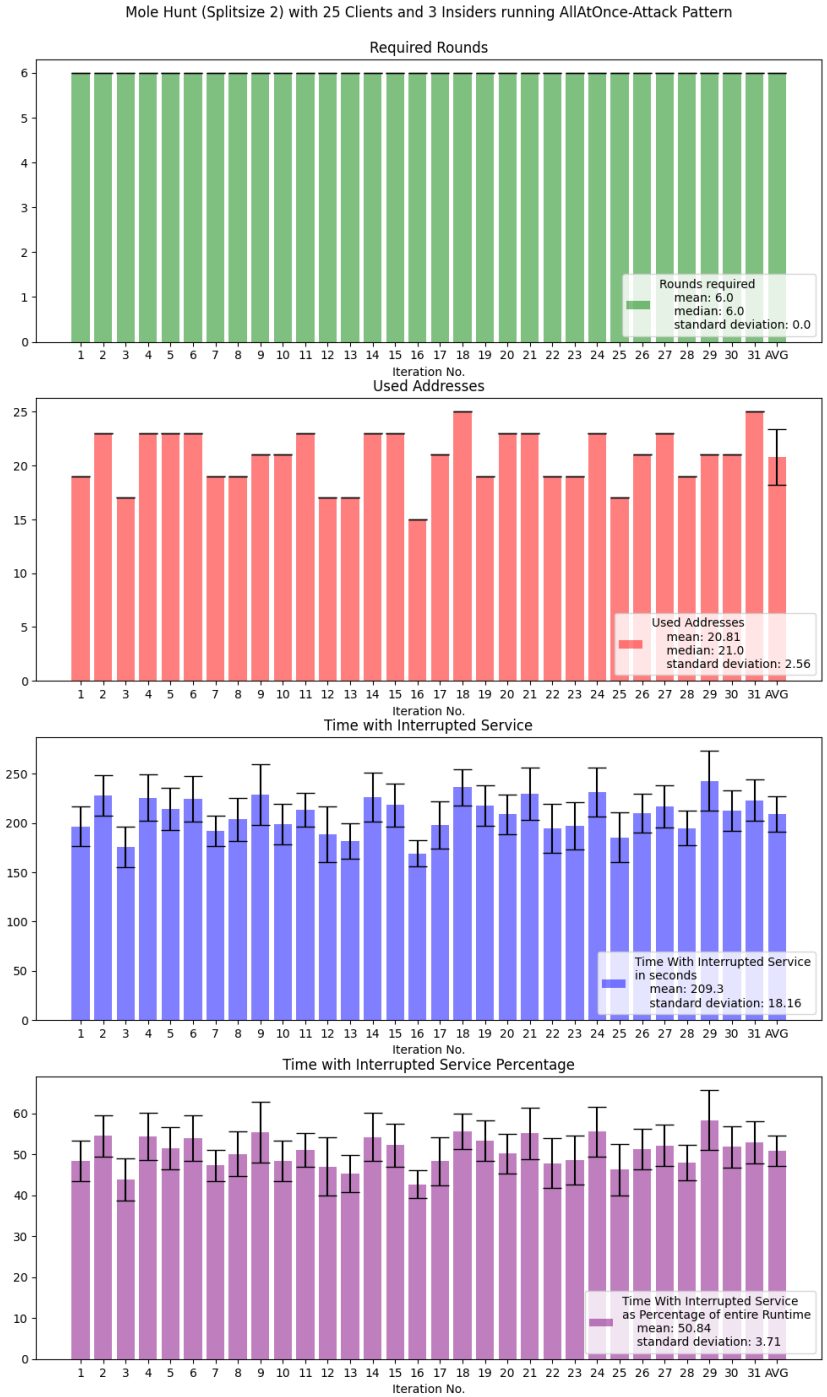


Figure A.1.: Mole Hunt (Split Size 2) with 25 Clients and 3 Insiders running AllAtOnce-Attack Pattern

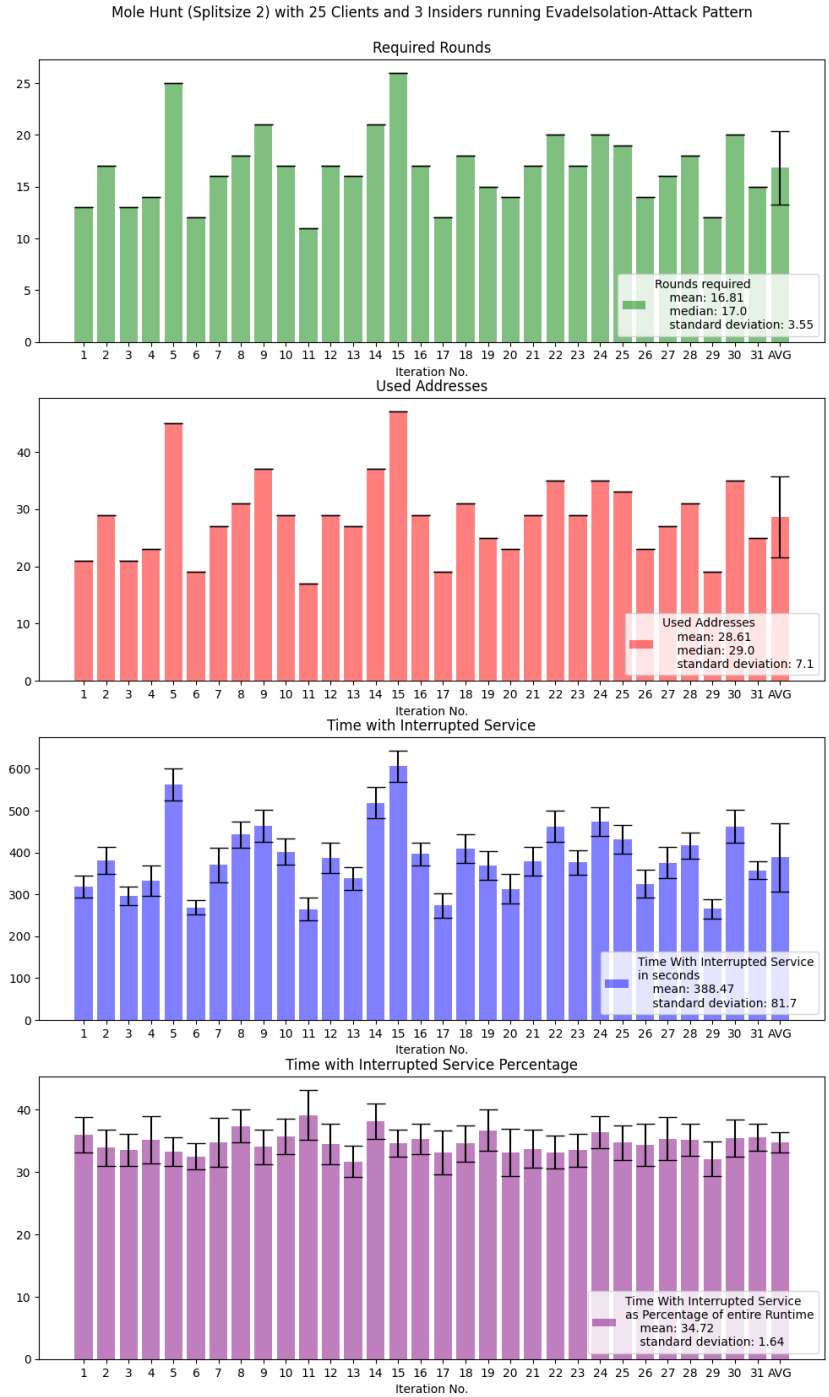


Figure A.2.: Mole Hunt (Split Size 2) with 25 Clients and 3 Insiders running EvadeIsolation-Attack Pattern

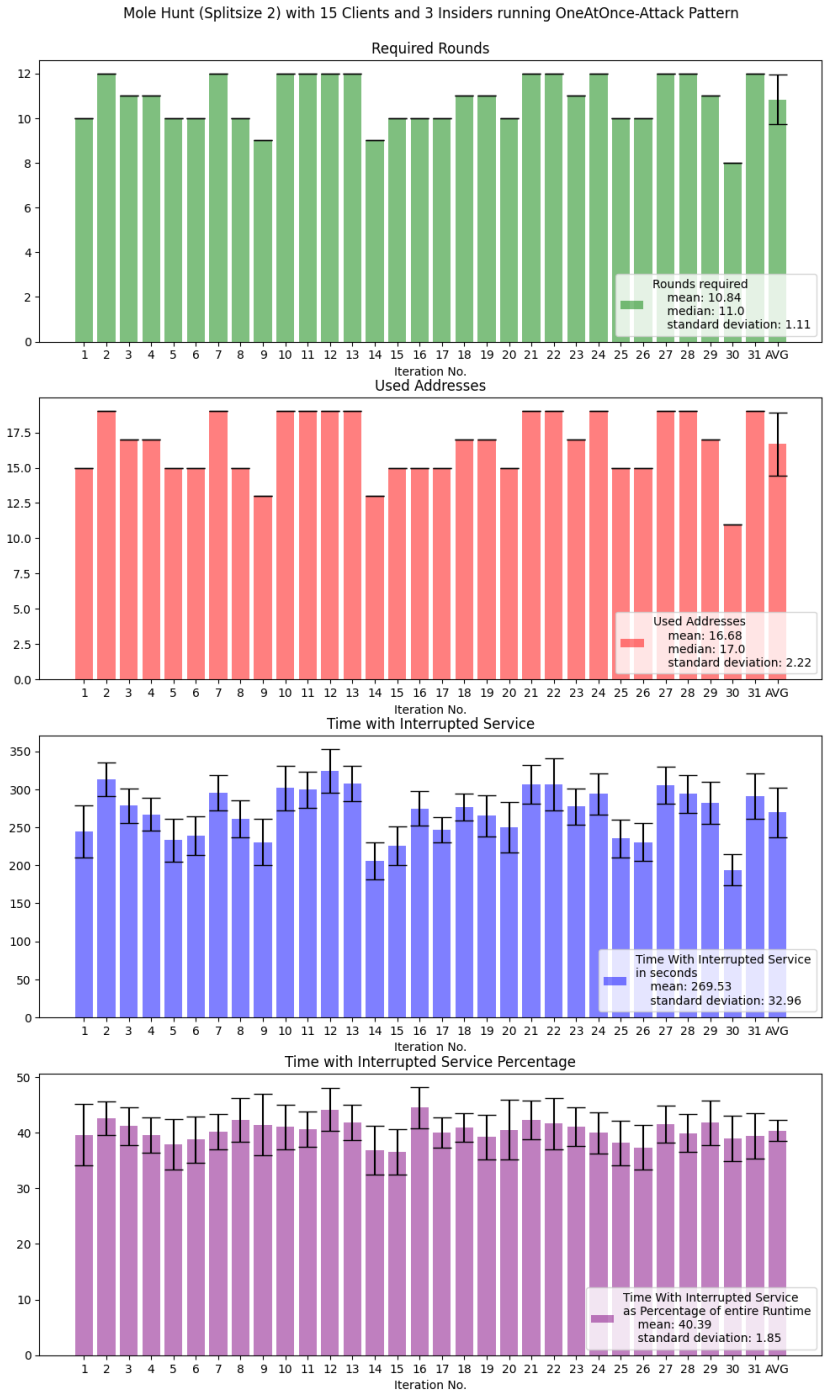


Figure A.3.: Mole Hunt (Split Size 2) with 15 Clients and 3 Insiders running OneAtOnce-Attack Pattern

Figure A.4.: Mole Hunt (Split Size 2) with 20 Clients and 3 Insiders running OneAtOnce-Attack Pattern

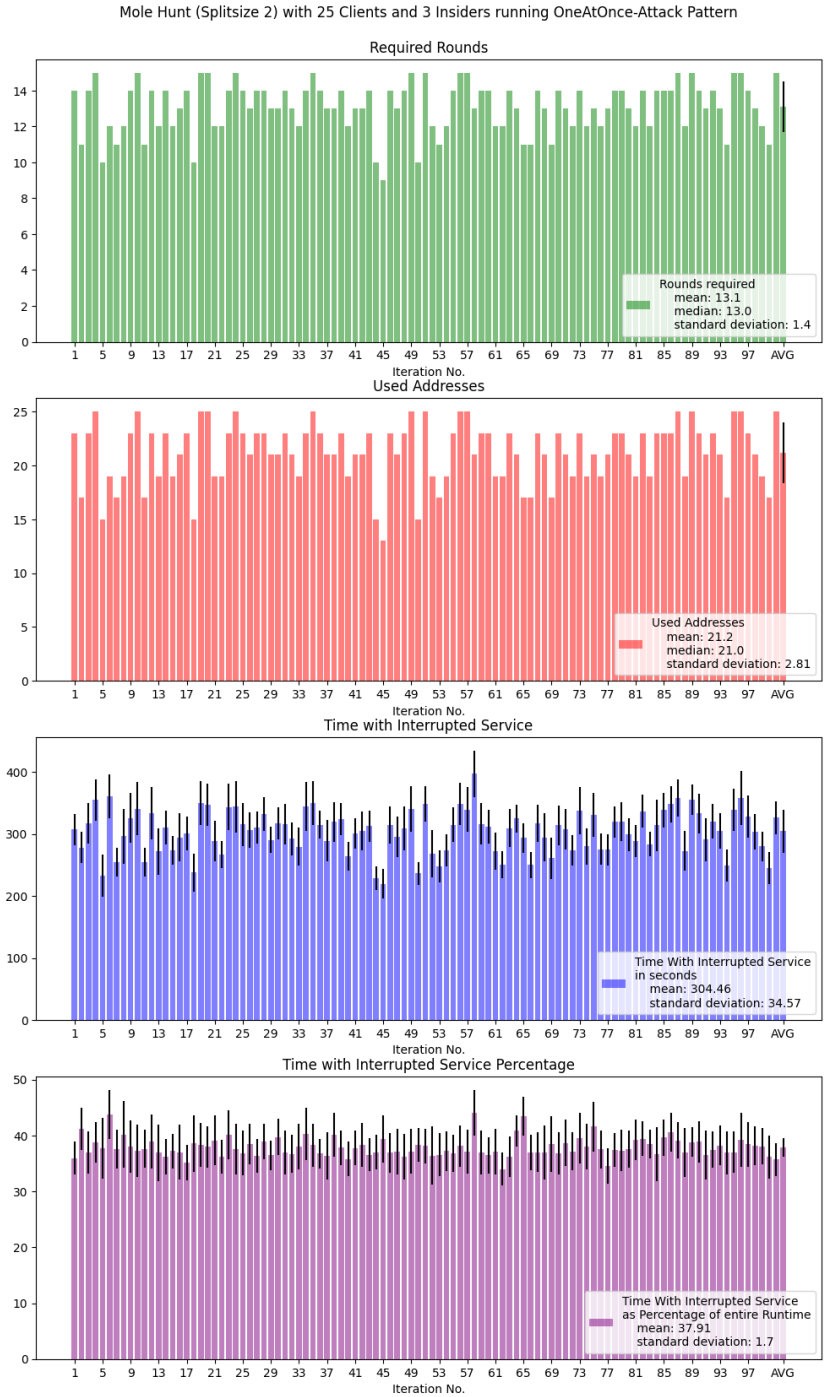


Figure A.5.: Mole Hunt (Split Size 2) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

Figure A.6.: Mole Hunt (Split Size 2) with 30 Clients and 3 Insiders running OneAtOnce-Attack Pattern

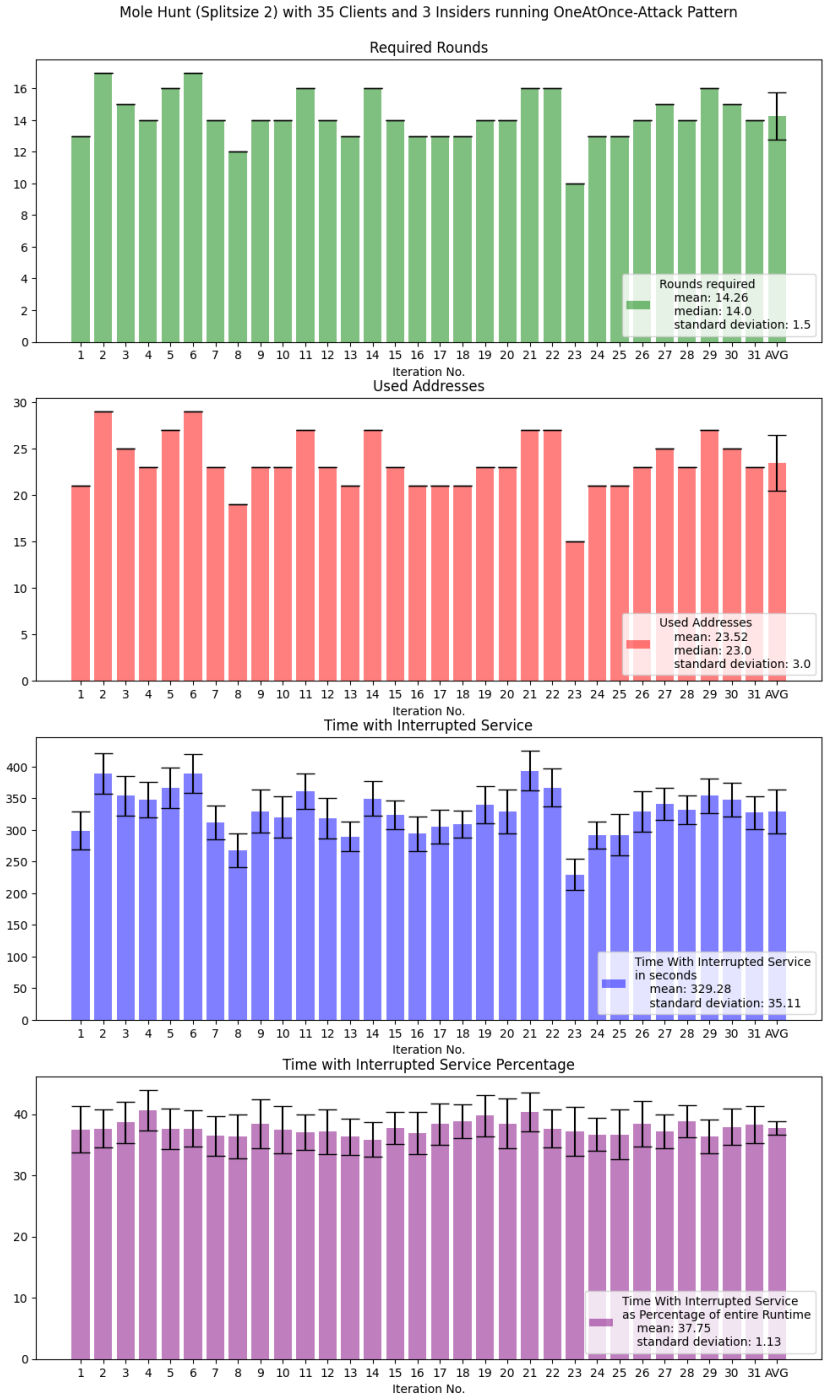


Figure A.7.: Mole Hunt (Split Size 2) with 35 Clients and 3 Insiders running OneAtOnce-Attack Pattern

Figure A.8.: Mole Hunt (Split Size 2) with 40 Clients and 3 Insiders running OneAtOnce-Attack Pattern

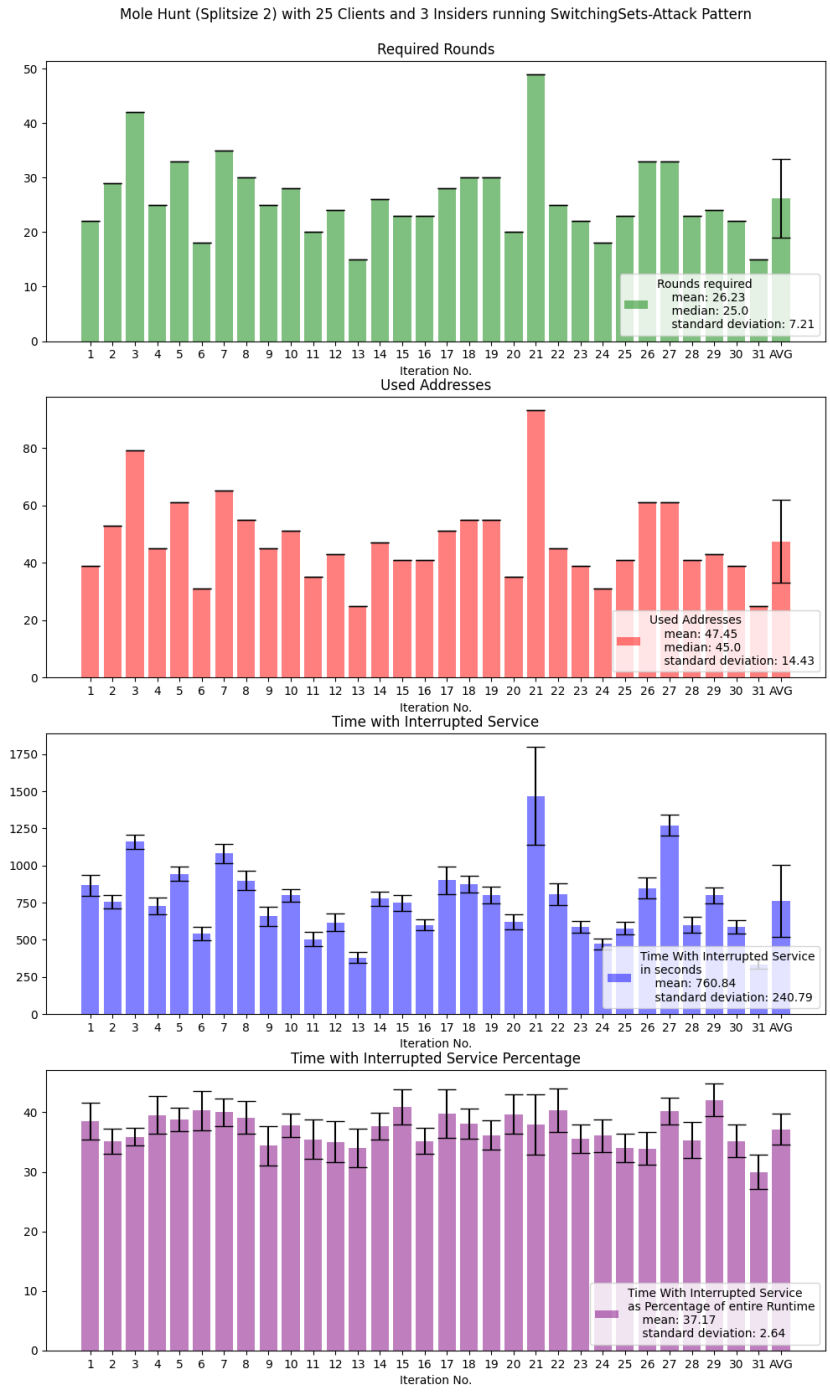


Figure A.9.: Mole Hunt (Split Size 2) with 25 Clients and 3 Insiders running SwitchingSets-Attack Pattern

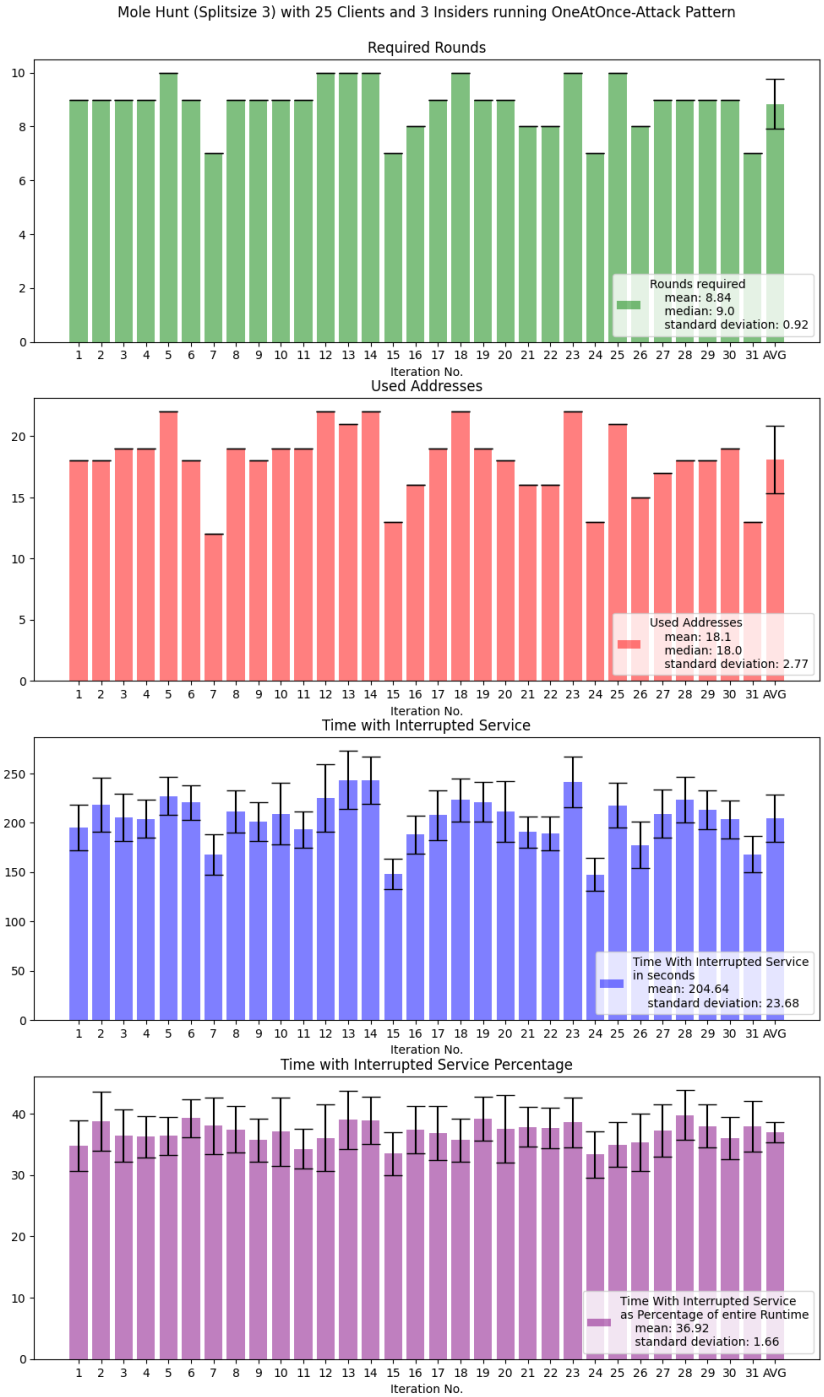


Figure A.10.: Mole Hunt (Split Size 3) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

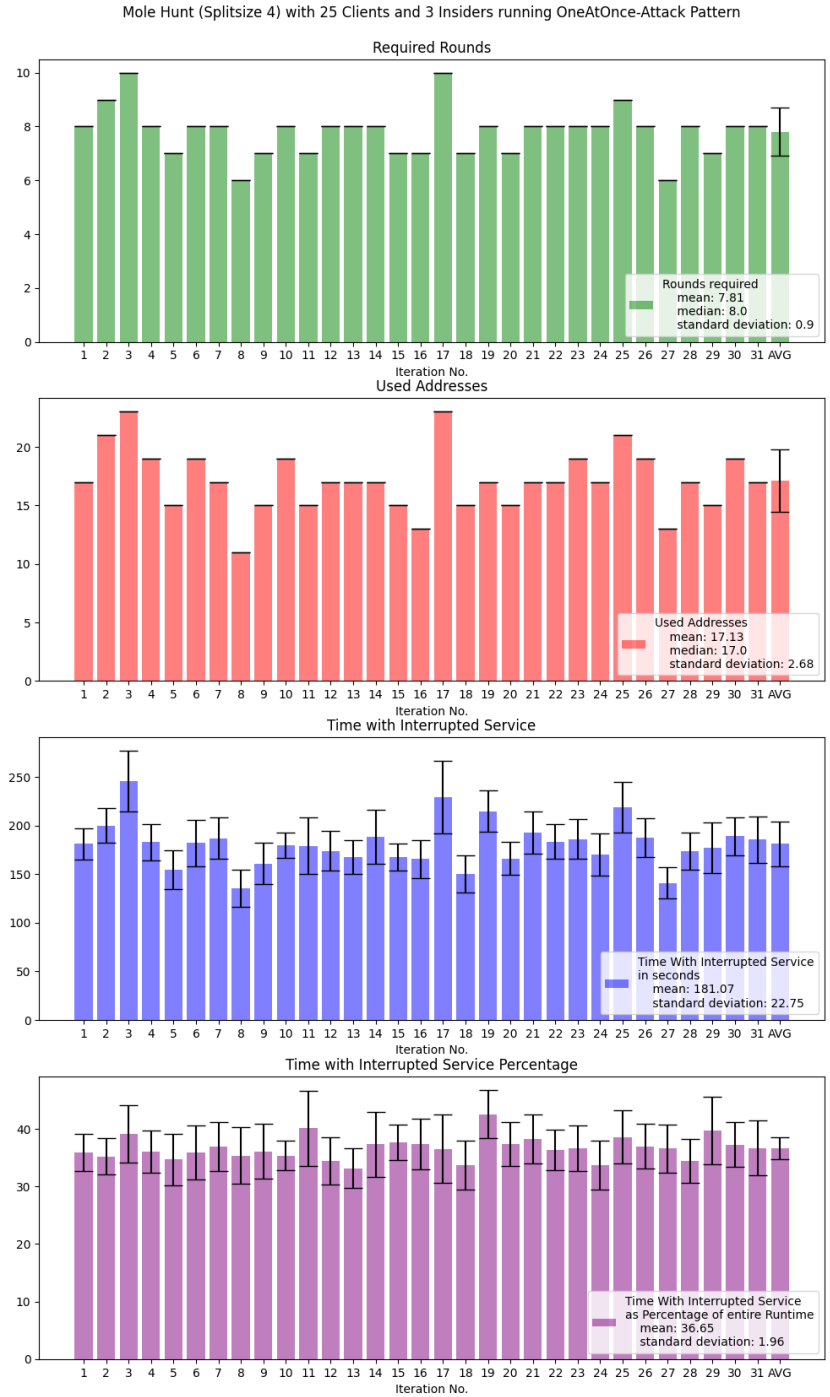


Figure A.11.: Mole Hunt (Split Size 4) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

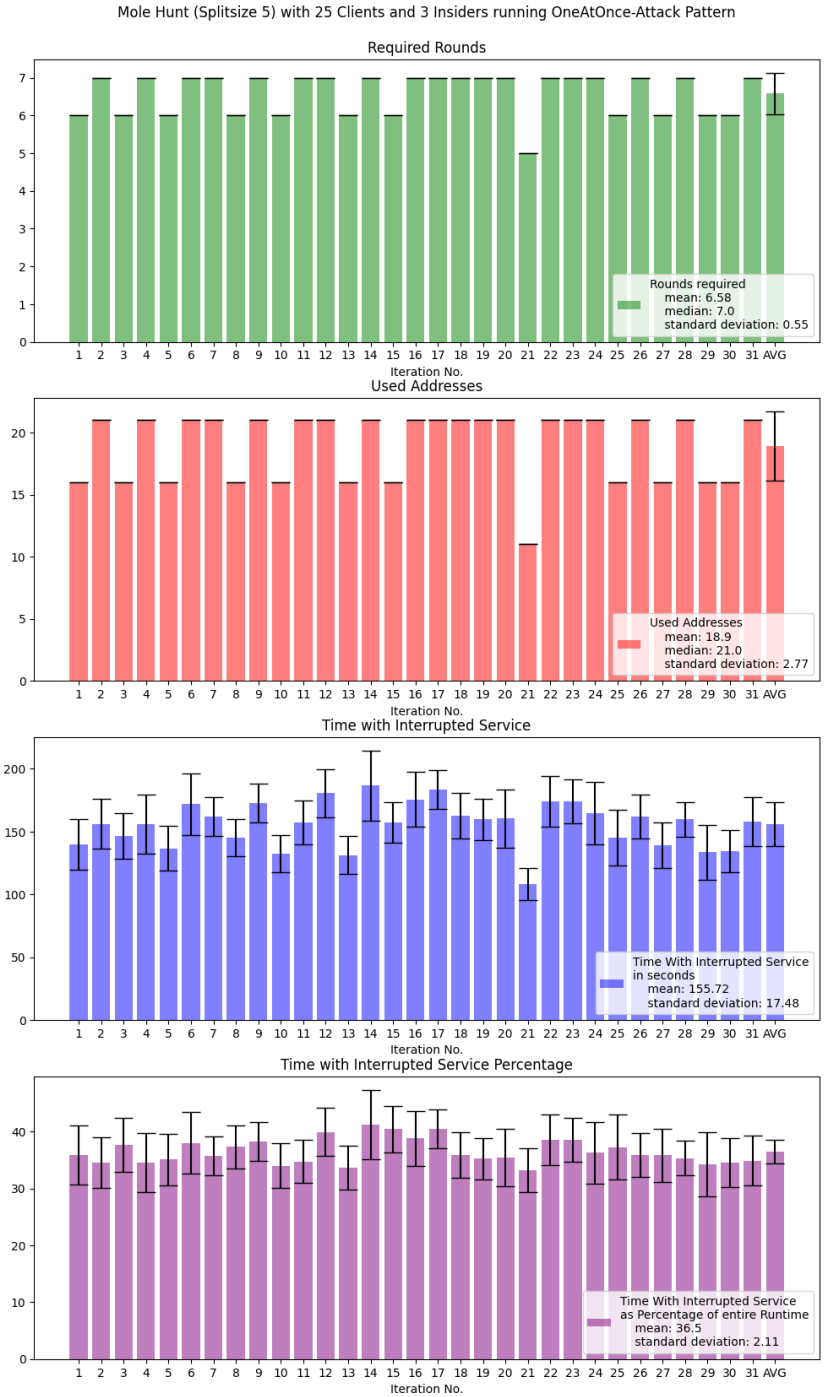


Figure A.12.: Mole Hunt (Split Size 5) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

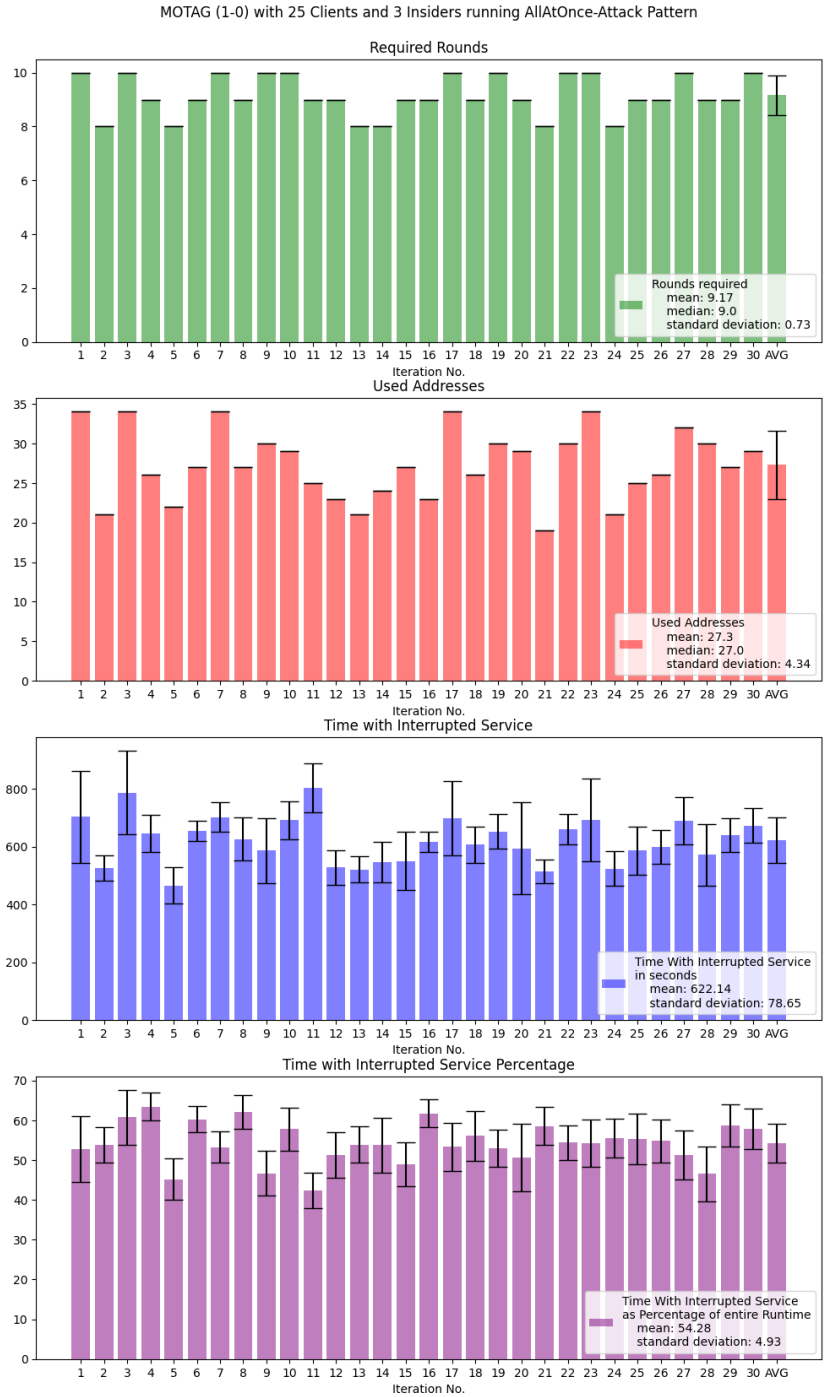


Figure A.13.: MOTAG (1-0) with 25 Clients and 3 Insiders running AllAtOnce-Attack Pattern

Figure A.14.: MOTAG (1-0) with 15 Clients and 3 Insiders running OneAtOnce-Attack Pattern

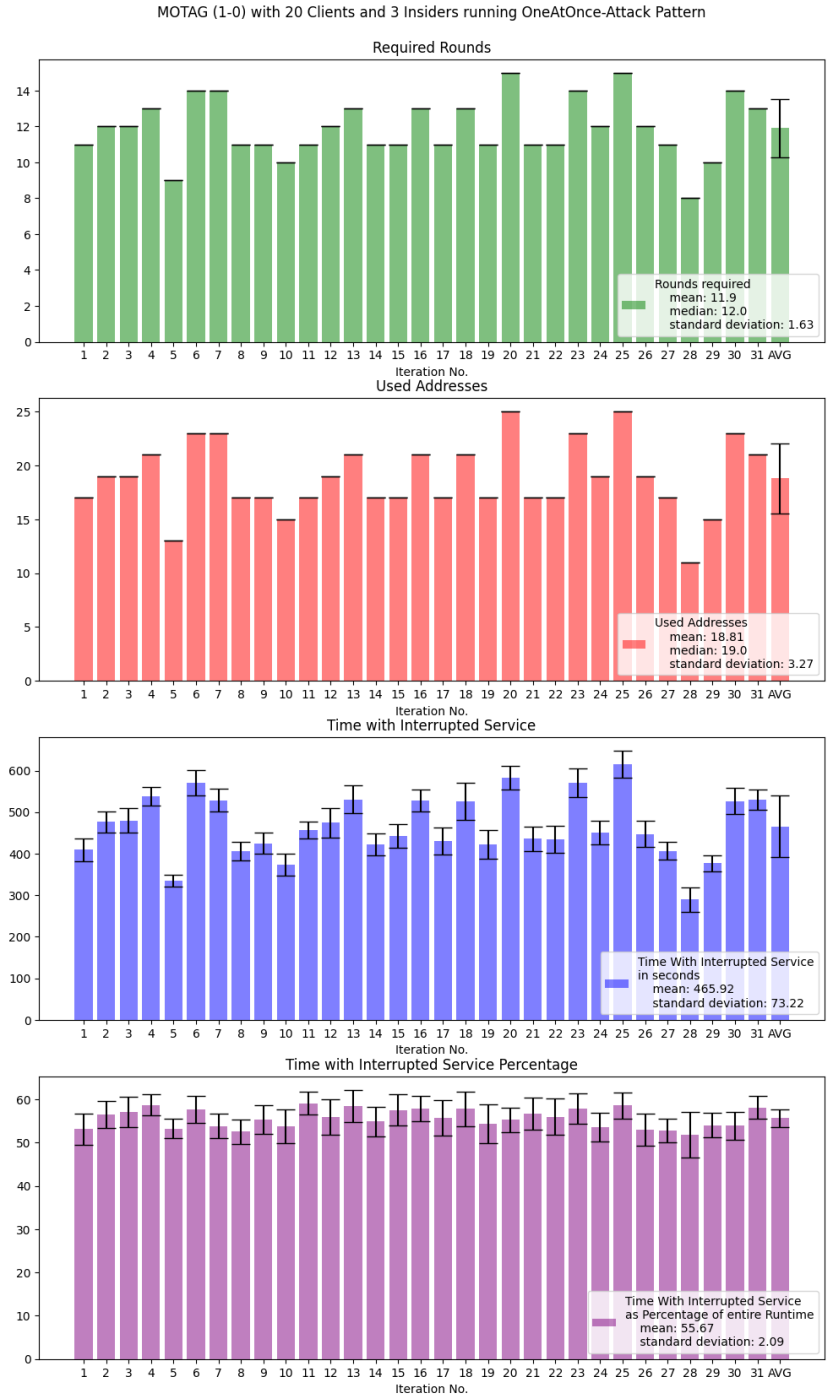


Figure A.15.: MOTAG (1-0) with 20 Clients and 3 Insiders running OneAtOnce-Attack Pattern

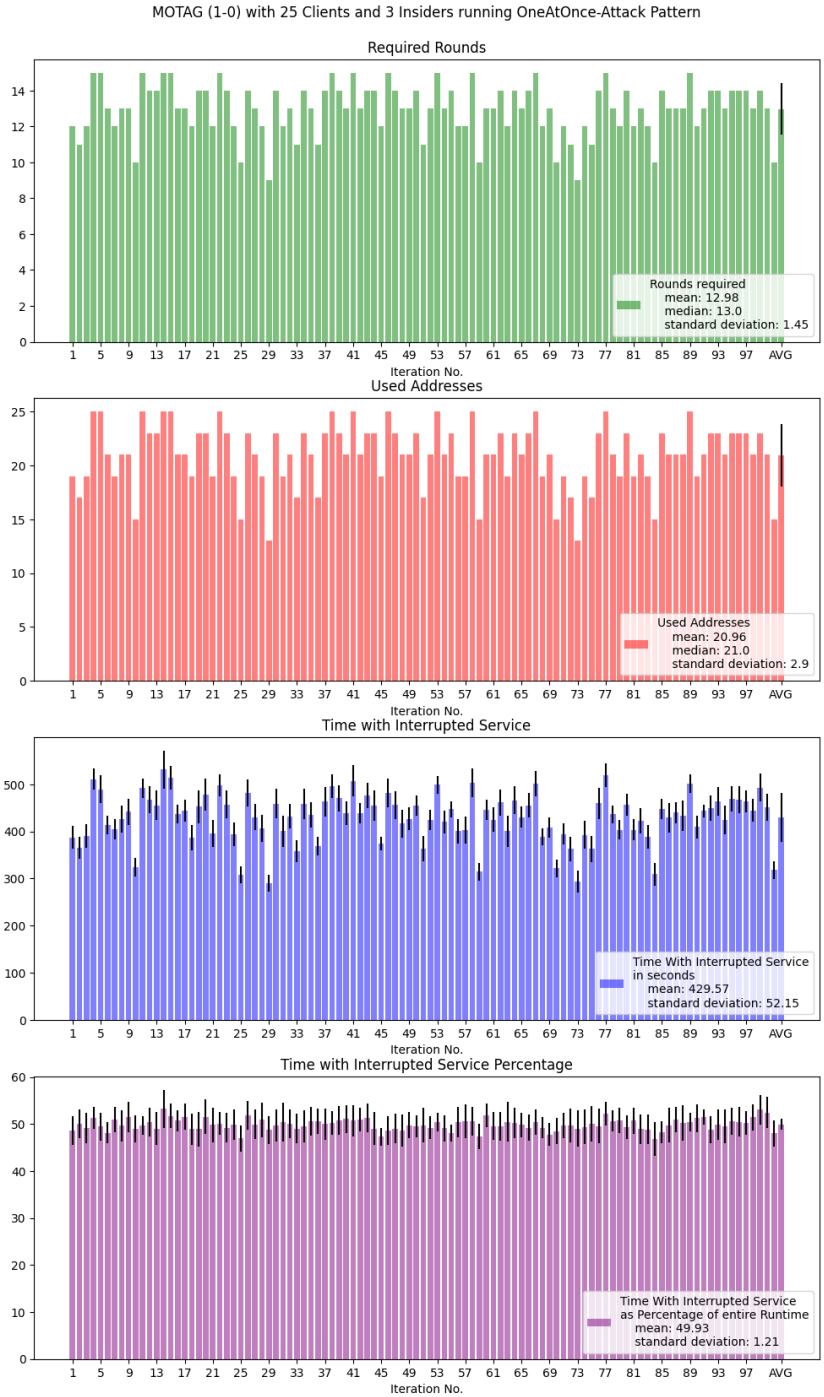


Figure A.16.: MOTAG (1-0) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

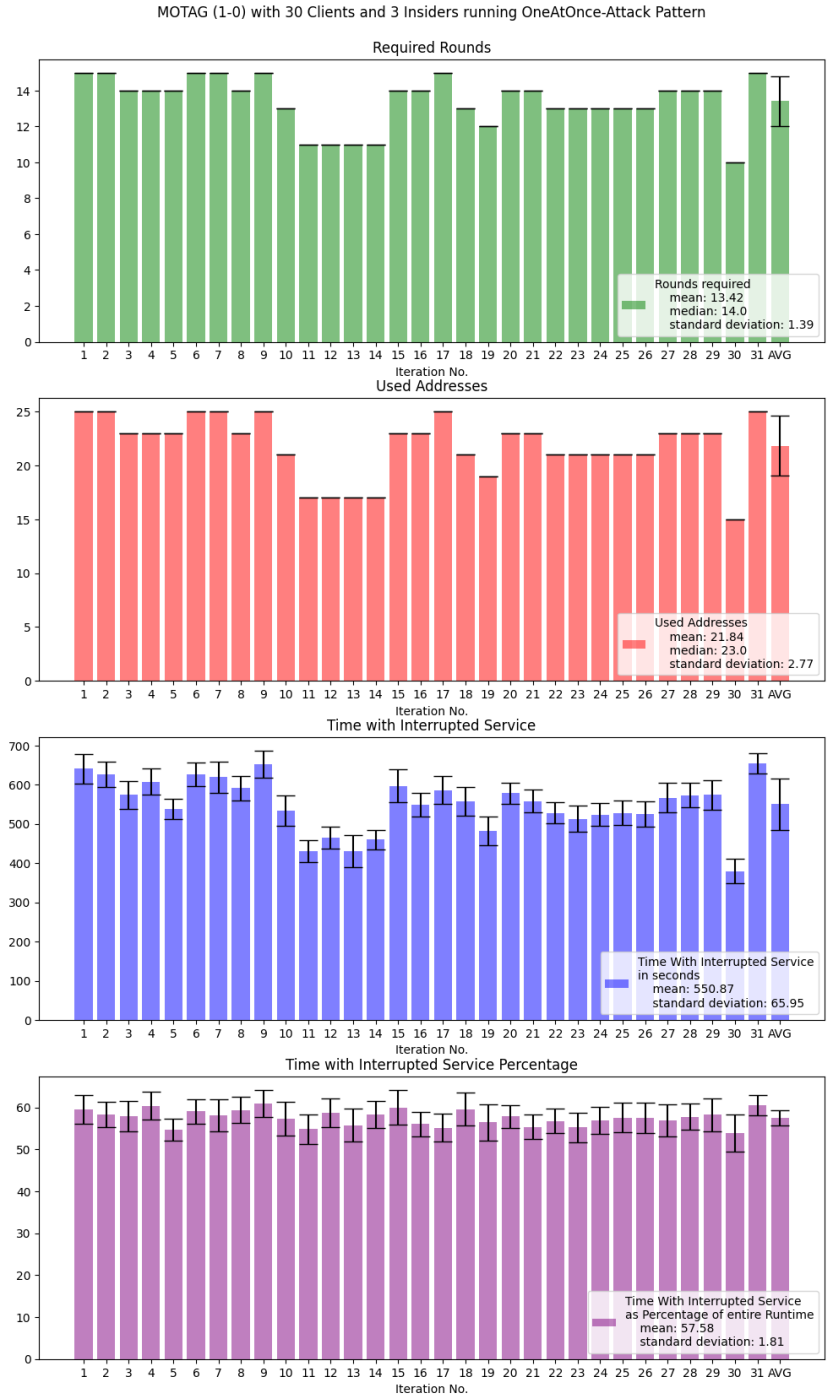


Figure A.17.: MOTAG (1-0) with 30 Clients and 3 Insiders running OneAtOnce-Attack Pattern

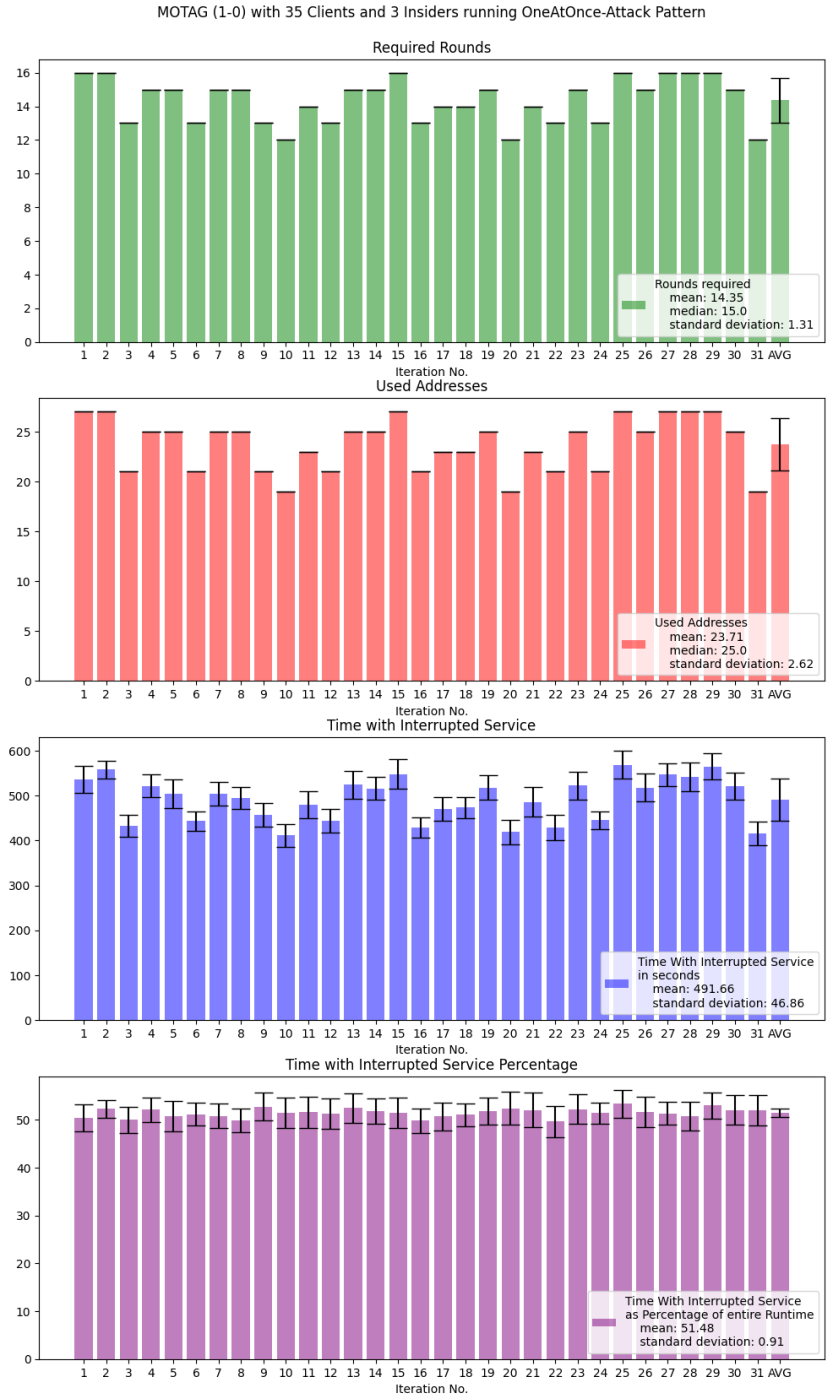


Figure A.18.: MOTAG (1-0) with 35 Clients and 3 Insiders running OneAtOnce-Attack Pattern

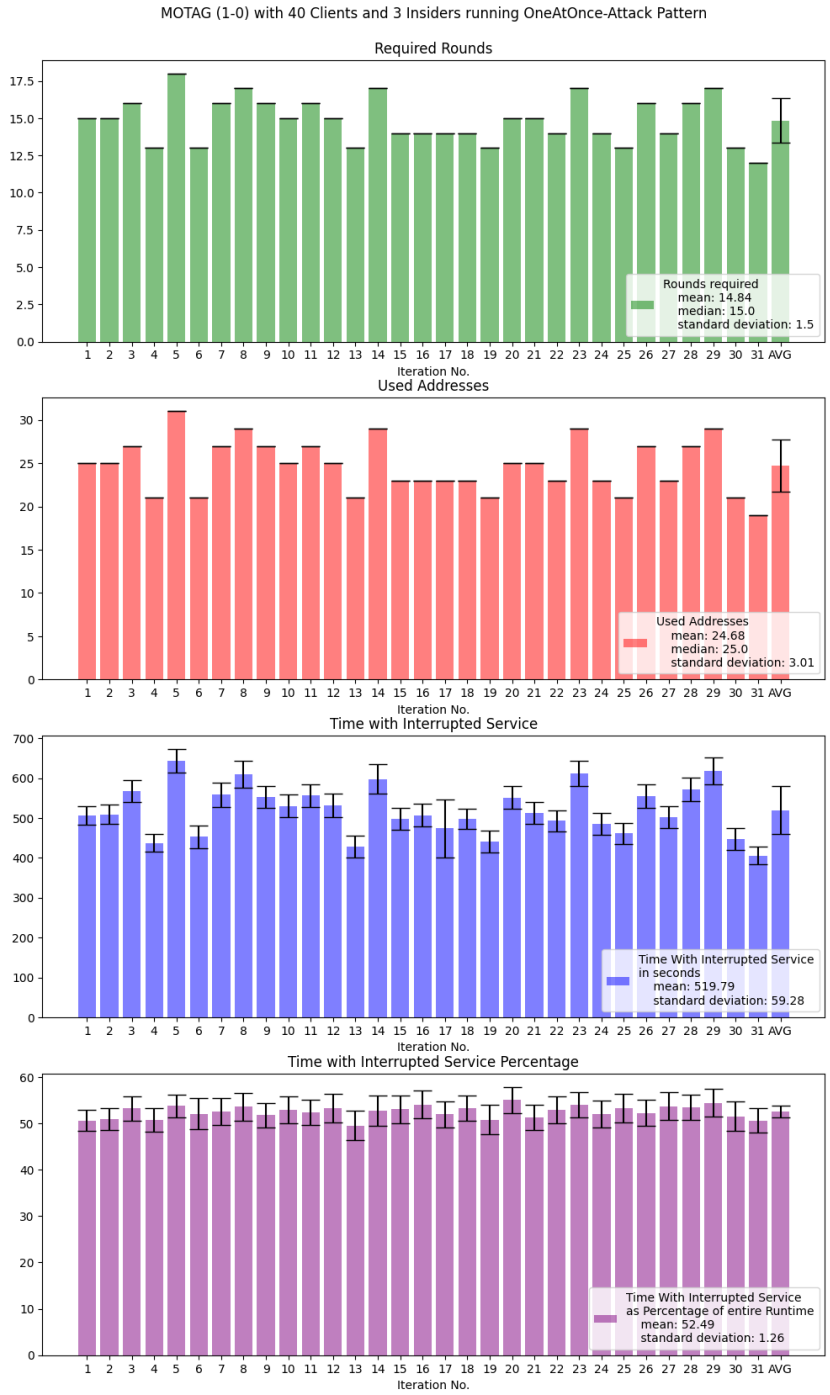


Figure A.19.: MOTAG (1-0) with 40 Clients and 3 Insiders running OneAtOnce-Attack Pattern

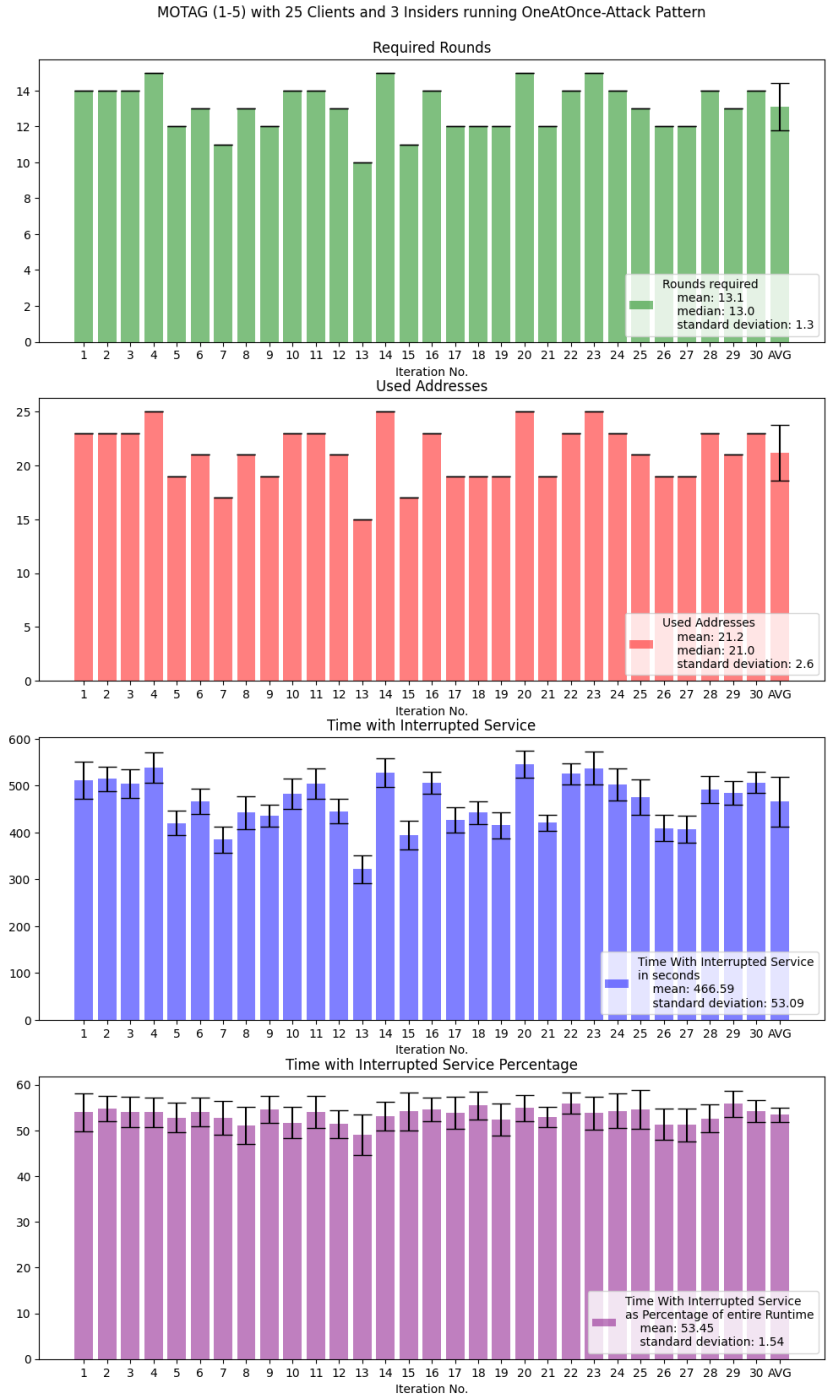


Figure A.20.: MOTAG (1-5) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

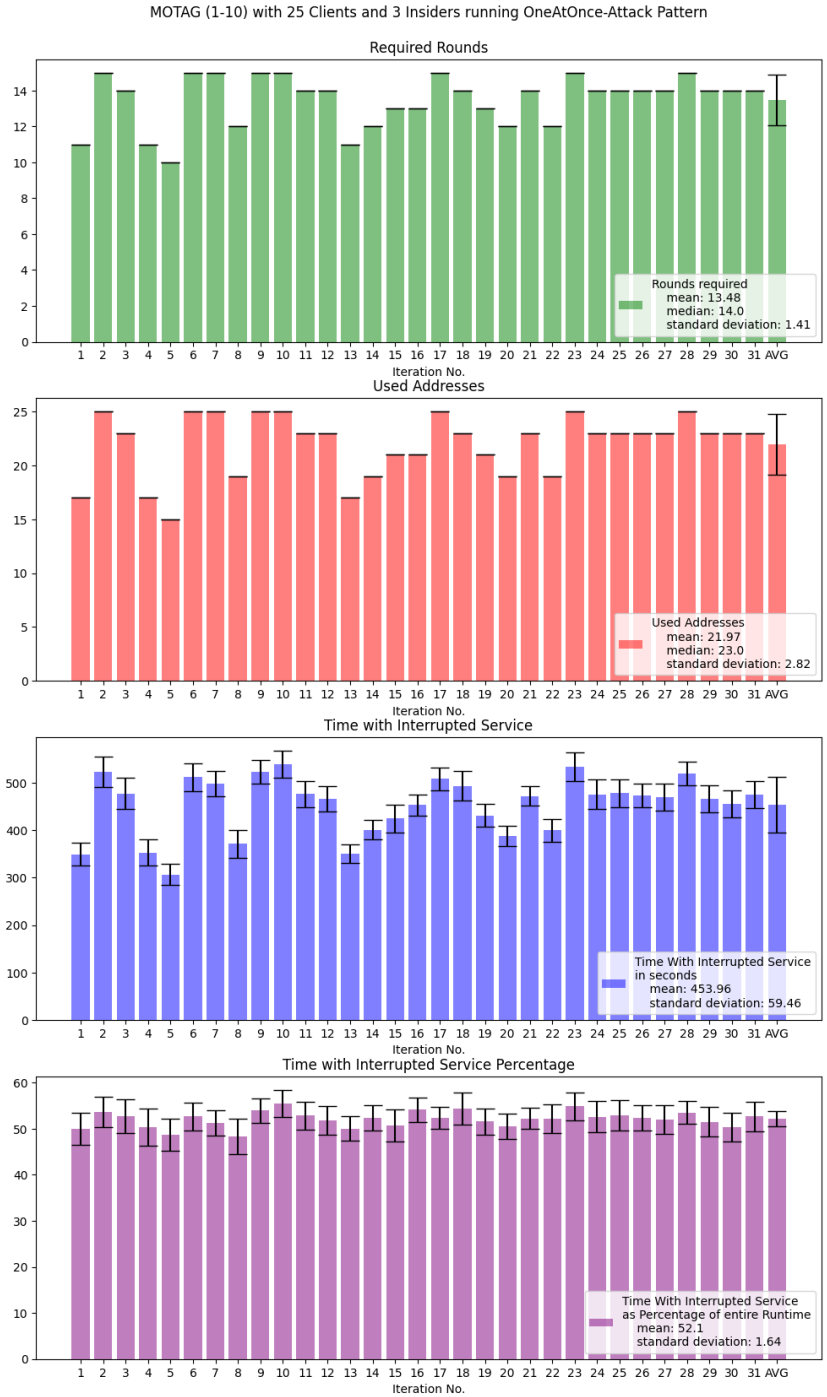


Figure A.21.: MOTAG (1-10) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

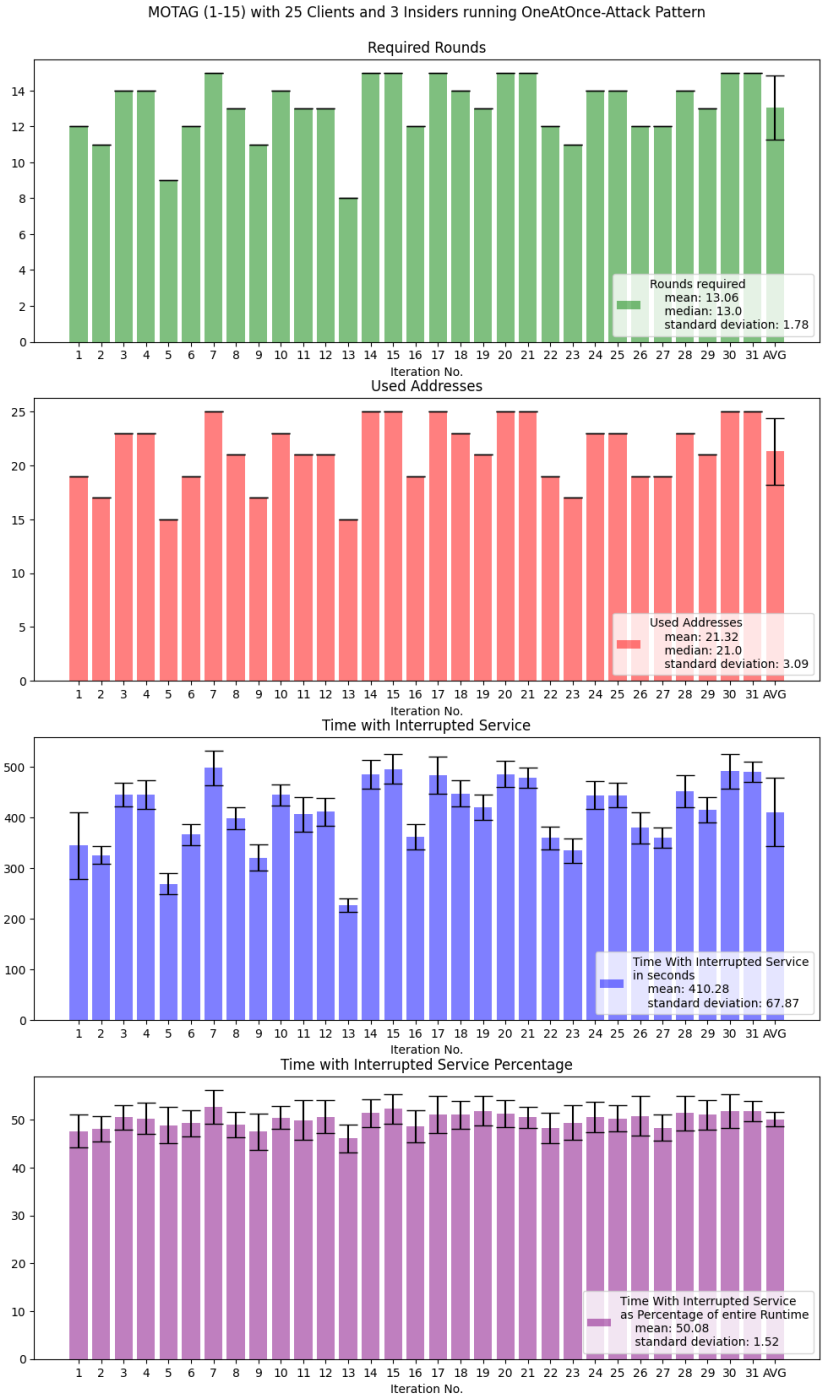


Figure A.22.: MOTAG (1-15) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

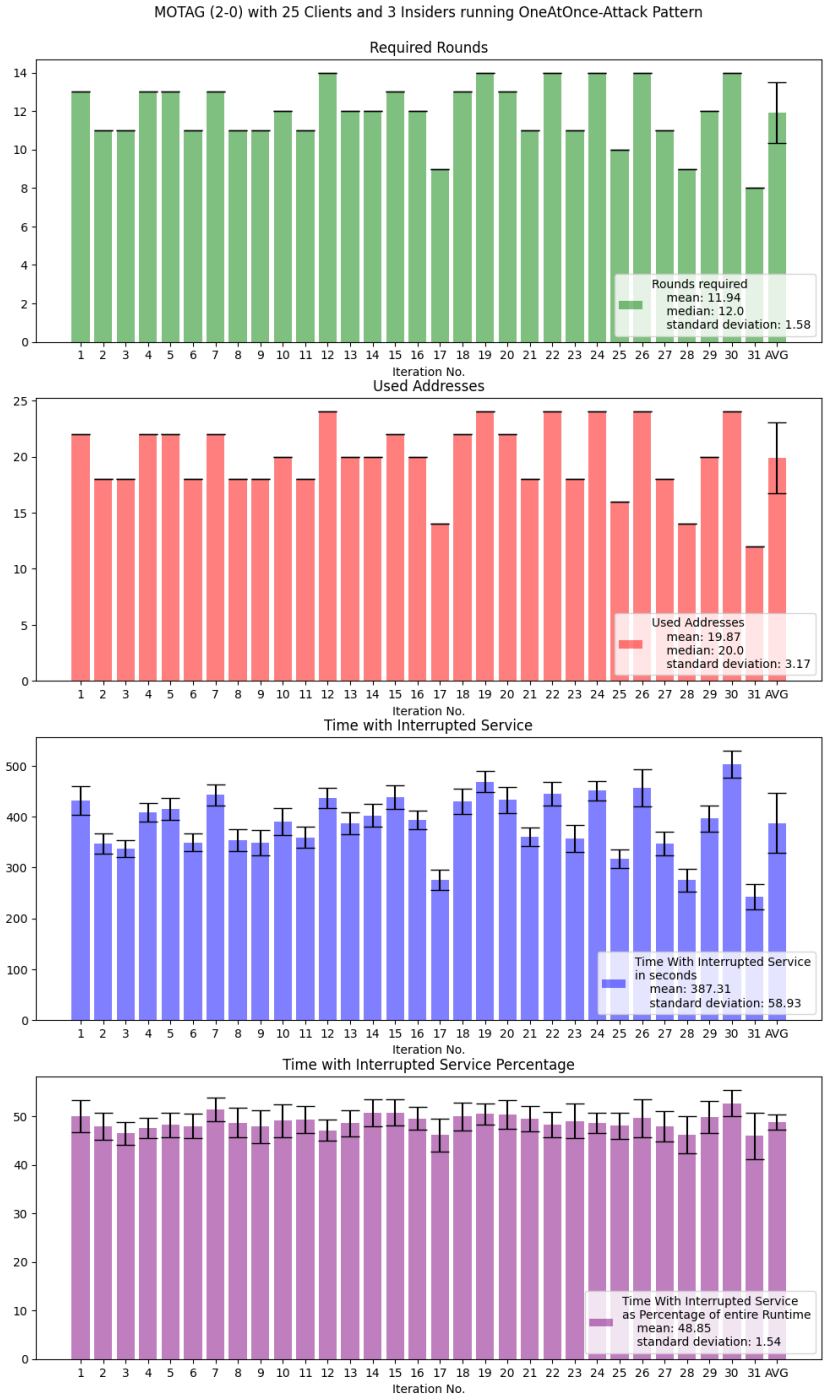


Figure A.23.: MOTAG (2-0) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

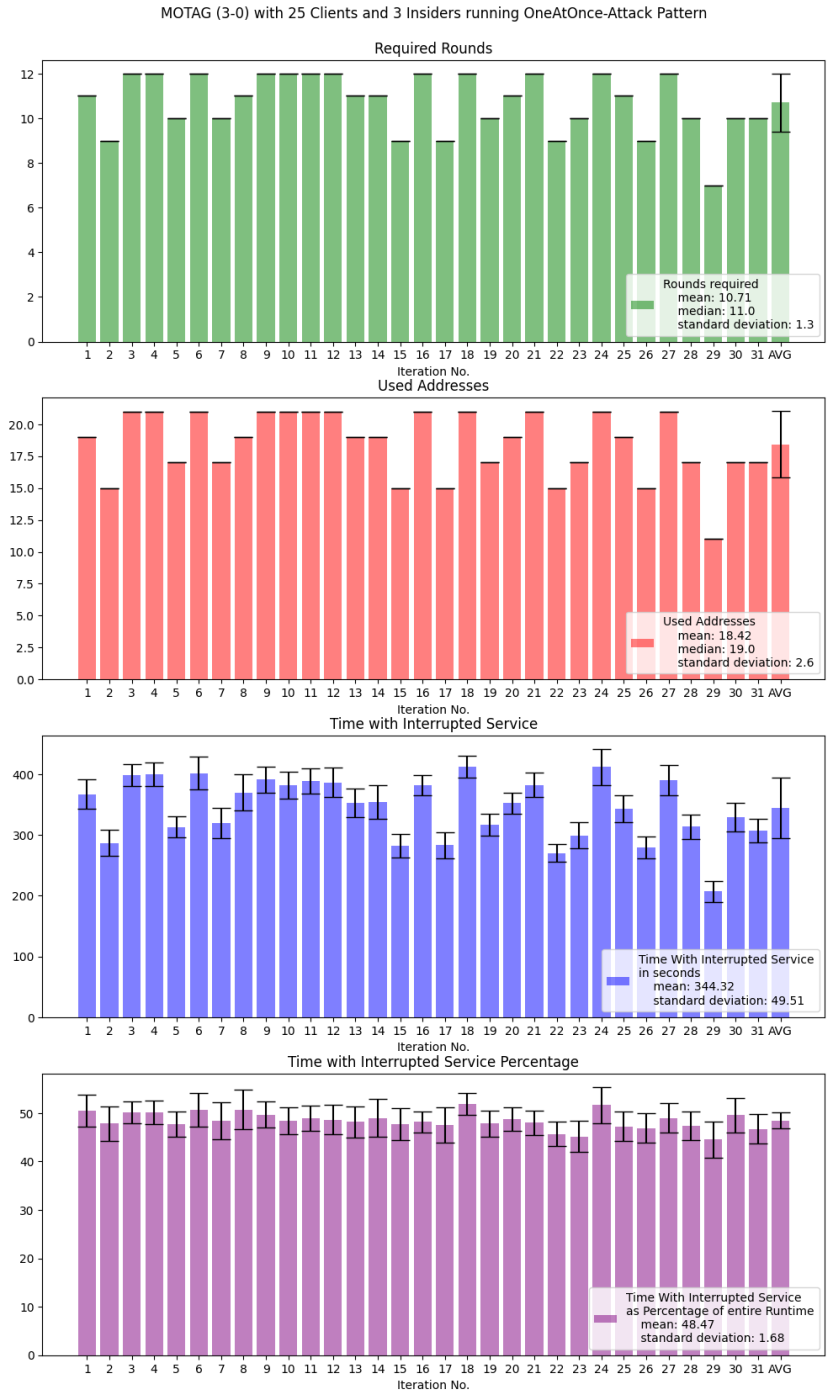


Figure A.24.: MOTAG (3-0) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

Figure A.25.: MOTAG (4-0) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern

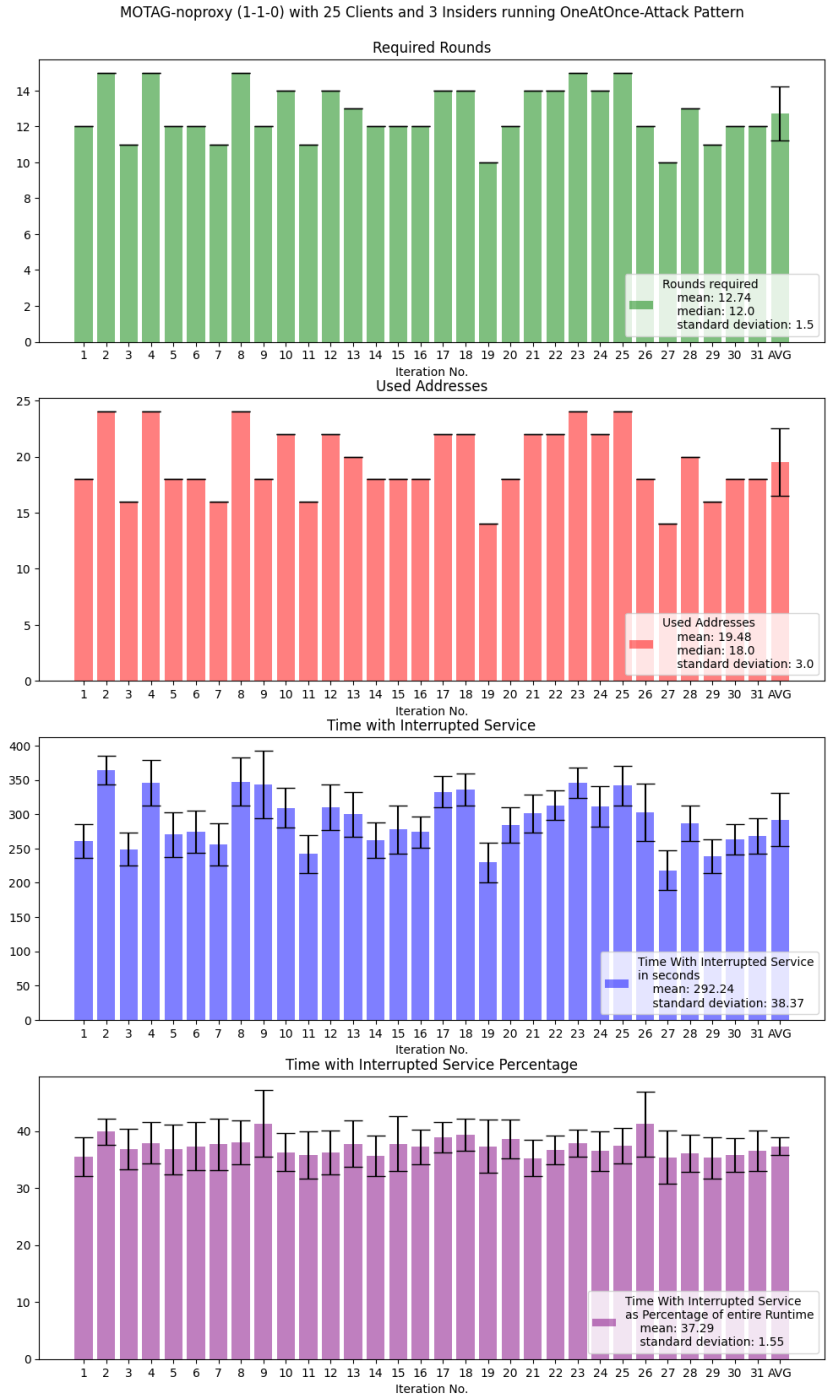


Figure A.26.: MOTAG-noproxy (1-0) with 25 Clients and 3 Insiders running OneAtOnce-Attack Pattern