University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Data Discovery in a DDoS Data Mesh Network

Tim Portmann
Zurich, Switzerland
Student ID: 19-915-883

Supervisor: Dr. Bruno Rodrigues, Katharina O. E. Müller and Prof. Dr. Burkhard Stiller
Date of Submission: 08.08.2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

Distributed-Denial-of-Service (DDoS) Angriffe stellen in der heutigen digitalen Landschaft weiterhin eine anhaltende Bedrohung dar. Kollaborative Verteidigungsansätze gewinnen immer mehr an Popularität, indem sie einen verteilten Verteidigungsansatz für einen verteilten Angriff vorschlagen. Ein zentraler Punkt solcher kollaborativer Abwehransätze ist der Austausch von DDoS Angriffsdaten unter den Parteien der Abwehrarchitektur.

In der Forschung werden zwar Konzepte vorgeschlagen, die den kollaborativen Austausch von DDoS Informationen ermöglichen, datenzentrierte Lösungen werden jedoch selten erforscht. Oftmals haben die vorgeschlagenen Konzepte einen gemeinsamen Nachteil: Sie sind von spezifischen Technologien oder Hardware abhängig, was ihre breite Anwendung einschränkt.

Ziel dieser Arbeit ist es, eine datenzentrische Lösung vorzuschlagen, die es dezentralen Parteien in einer kollaborativen DDoS-Abwehrarchitektur ermöglicht, DDoS Angriffsinformationen auszutauschen. Die vorgeschlagene Lösung nutzt die Idee hinter einem Data Mesh Network für den Informationsaustausch. Zusätzlich wird die vorgeschlagene Architektur durch einen Service ergänzt, welcher das Erforschen und Visualisieren von ausgetauschten Daten ermöglicht.

Zunächst wird eine umfassende Untersuchung des Themas und der verfügbaren Tools zum Aufbau einer DDoS-Data-Mesh-Architektur durchgeführt. Anschliessend wird ein Entwurfsvorschlag für die DDoS-Data-Mesh-Architektur, einschliesslich des Services für die Erforschung und Visualisierung der Daten, beschrieben. Auf der Grundlage dieses Entwurfs wird ein Prototyp des DDoS-Data-Mesh implementiert und eingesetzt, wobei die zuvor untersuchten Tools verwendet werden. Abschliessend wird der Prototyp im Hinblick auf seine Leistung und Datenererforschungsfunktionen bewertet.

Die vorgeschlagene Lösung nutzt einen Technologie-Stack, der aus MySQL-Instanzen als DDoS-Datenspeicher, Trino als verteilte Query-Engine und Apache Superset als Service für die Datenerforschung, besteht. Diese Kombination ermöglicht den effizienten Austausch und die Erforschung von DDoS-Daten, was sie zu einer effektiven und datenzentrischen Lösung für kollaborative DDoS-Abwehrszenarien macht.

Abstract

Distributed Denial-of-Service (DDoS) attacks continue to pose a persistent threat in today's digital landscape. Collaborative defense approaches continuously gain popularity by proposing a distributed defense approach for a distributed attack. Central to such collaborative defense approaches is the exchange of DDoS attack data amongst the parties of the defense architecture.

While existing research proposes concepts that enable the collaborative sharing of DDoS information, data-centric solutions remain scarce. Oftentimes, the proposed concepts share a common drawback: Their dependence on specific technologies or hardware that restricts their broad adoption.

This thesis aims to propose a data-centric solution that enables decentralized parties in a collaborative DDoS defense architecture to exchange DDoS attack information. The proposed solution utilizes a data mesh network to handle information exchange, complemented by a data discovery service to act upon the exchanged DDoS data.

First, extensive research into the subject and tools available to build a DDoS data mesh architecture is explored. Subsequently, a design proposal for the DDoS data mesh architecture, including data discovery capabilities, is described. Based on this design, a DDoS data mesh prototype is implemented and deployed, using the tools explored earlier. Finally, the data mesh is evaluated in regard to its performance and data discovery capabilities.

The solution proposed utilizes a technology stack consisting of MySQL instances as DDoS data repositories, Trino as a distributed query engine, and Apache Superset as the data discovery service. This combination enables the efficient exchange and exploration of DDoS data, making it effective for collaborative DDoS defense scenarios and a viable data-centric solution for the exchange of DDoS attack data.

Acknowledgments

I would like to express my deepest gratitude to the Communication Systems Group and specially my supervisor Dr. Bruno Rodrigues for the support during this thesis. The close communication with Dr. Rodrigues helped me focus on the relevant topics and ensured the timely submission of this thesis. Throughout the entire project, his insights, constructive feedback and creative approaches have been instrumental. Without his valuable contributions, this project would not have been possible.

Contents

Chapter 1

Introduction

1.1 Motivation

Distributed Denial-of-Service (DDoS) attacks have been a persistent and challenging issue on the Internet, leading to numerous proposals for countering these attacks from both centralized and distributed (cooperative) perspectives. One promising approach is the adoption of cooperative defense strategies. These can offer various benefits such as reducing the burden on individual domains, enhancing detection and mitigation capabilities and blocking malicious traffic closer to its source. However, implementing a cooperative defense in the highly diverse internet environment is a complex task. The environment is extremely heterogeneous and encompasses diverse technologies, organizational structures and legal frameworks that pose their respecting set of challenges [46].

Given the wide distribution of DDoS attacks, an effective defense strategy involves a distributed approach to block attacking traffic closer to its source [46]. With the growing emphasis on data-driven cybersecurity, a data mesh structure offers an intriguing solution to enhance defense systems that currently operate in isolation. Unlike a data lake, which centralizes data storage, a data mesh is a decentralized and distributed architecture enabling independent information exchange and collaboration between organizations [8]. For DDoS attacks, a data mesh can facilitate the sharing of crucial information, including attack origin, type, volume, detection methods and mitigation techniques. By leveraging a data mesh, organizations can collectively strengthen their defense against DDoS threats.

Several technologies and concepts have been employed to enable the collaborative sharing of DDoS information. Previous research [46] utilized Distributed Ledger Technology (DLT) to establish a signaling platform. Traditional approaches like D-WARD [38], MULTOPS [19] and Distributed Packet Filtering - DPF [42] proposed specialized hardware and protocols for information sharing. While these approaches have their respective advantages, they share a common drawback: Their dependence on specific technologies or hardware restricts their broad adoption. Given the diverse nature of systems forming the backbone of the Internet, an ideal solution should be data-centric.

1.2 Thesis Goals

This thesis focuses on exploring the possibilities of DDoS data mesh networks, directly addressing the motivation outlined above. The main objective is to investigate and implement data exchange and discovery aspects within the context of a collaborative DDoS defense. To achieve this goal, we assume a specific use case where certain components of a collaborative DDoS defense are already in place. Particularly, the capturing and analysis of DDoS attack data into DDoS fingerprints is assumed throughout the thesis. This allows us to narrow our focus to the design and implementation of data exchange and discovery using a data mesh network. More specifically, the goals of this thesis can be summed up into the following two main goals:

- Give an overview of DDoS attack basics and associated related work. Produce a basis for the comparison of a DDoS data mesh approach, listing its associated advantages and drawbacks.
- Design and implement a data mesh architecture, including a discovery service to find information about the nature and characteristics of DDoS attacks.

1.3 Methodology

To achieve the stated goals and create a baseline DDoS data mesh architecture, we followed this approach:

- Referenced Research: We start by surveying the existing research and available tools to gain an understanding of the current landscape. This allows us to design the architecture, taking into account the state-of-the-art research and tools available. Additionally, we formulate the necessary requirements based on the goals of the thesis and the nature of the architecture.
- Applied Research: With the architecture design outlined, we proceed to implement and configure the service as a prototype. During this phase, we also handle the deployment of the prototype. Additionally, we evaluate the performance and usability of the prototype to assess its effectiveness. Finally, we discuss performance, security and availability trade-offs that must be considered when deploying the architecture in a non-prototype setting.

1.4 Thesis Outline

This thesis is organized as follows: Chapter 2 provides the necessary background on various topics addressed throughout the thesis. Additionally, this chapter addresses the related work, listing and assessing existing tools for their potential use in implementing a

DDoS data mesh network. Chapter 3 presents the design of the DDoS data mesh network, outlining its architecture and components. Chapter 4 focuses on the implementation and configuration of the proof of concept prototype. In Chapter 5, we evaluate the architecture, examining its performance and usability. Additionally, we analyze architecture trade-offs and implementation considerations. Finally, Chapter 6 offers the concluding summary, considerations and insights into future work regarding the topic of this thesis.

Chapter 2

Fundamentals

2.1 Background

2.1.1 Distributed and Decentralized

It is crucial to distinguish between distributed and decentralized systems as this distinction has significant implications for designing and utilizing various technologies, systems and applications. In the realm of data architectures, this difference has a particularly noteworthy effect on their architecture, governance, security and potential applications. To visualize the contrast between these two types of systems, figure 1 depicts their respective network topologies.

While distributed and decentralized systems share traits like having multiple nodes, built-in redundancy and scalability, they differ in their organizational structure. This is outlined below (*cf.* figure 2.1):

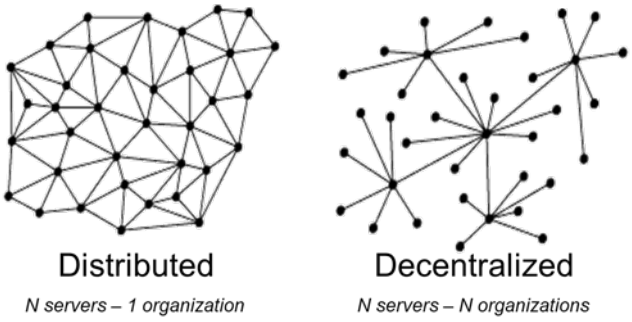

Figure 2.1: Distributed and Decentralized Systems

- Distributed systems: Managed by a central coordinator or a group of coordinators who handle node communication and synchronization.

- Decentralized systems: Lack centralized authority or coordinator. Each node has equal authority and can communicate directly with other nodes because there is no central point of control.

Distributed systems rely on a coordinator to ensure all nodes function correctly, while decentralized systems can self-organize and adjust to network changes without central coordination. The extent of centralization and control sets apart distributed and decentralized systems. The chapters 2.1.4 Data Lakes and 2.1.5 Data Mesh Networks touch on the benefits and drawbacks of centralized and decentralized data architectures.

2.1.2 DoS and DDoS

Denial-of-Service (DoS) attacks are typically characterized by preventing the legitimate use of a service. A Distributed-Denial-of-Service (DDoS) attack describes a DoS attack employing multiple attacking entities [37]. Over time, DoS and DDoS attacks have evolved to incorporate many different methods to render a victim machine unavailable. According to [31], the following are the most prominent DoS attack types:

- Packet flooding attacks
- Application layer attacks
- Protocol attacks

Packet flooding, or volume-based, attacks describe attacks where a large number of packets are sent to a victim machine. This results in excessive endpoint, transit and network bandwidth consumed at the destination [31]. The bandwidth consumption then renders the service unavailable for its legitimate clients. The packet types used in packet flooding attacks can vary. The most common packet types are TCP floods, ICMP echo request/reply, or UDP floods [31, 32].

Another approach to executing a DDoS attack is to exploit specific vulnerabilities in an application. A slow read attack, for example, starts with the attacker sending a GET request followed by a POST request advertising a window size of 0 bytes. The web server then stores the connection in its queue, waiting for the attacker to advertise a window of non-zero size. The attacker, however, does not advertise a new window size, which leads to the web server waiting an indefinite amount of time[60]. Application layer attacks like the slow read attack have been proven effective against many popular web servers [24].

Protocol attacks exploit the inner workings of certain protocols to attack a victim. For example, in a reflection amplification attack, the attacker spoofs the IP address of the victim. The attacker then sends a request for information, usually using UDP or TCP. The response is then sent to the IP address of the victim [44]. The reflected traffic is sent to a different target and amplified as small requests can trigger significantly larger responses [29]. The attack, therefore, exploits the inner workings of the UDP or TCP protocols to cause damage to the target system.

2.1.3 Characterizing DDoS Attacks

Even though the first publicly known DDoS attack occurred more than 25 years ago [9], the attack still threatens many online services. As mentioned above, many DDoS attack types exploit mechanisms deeply rooted in the internet. The widespread use of certain protocols makes it hard to prevent such attacks in the first place. The landscape of DDoS attack mechanisms continues to evolve. However, the circumstances enabling attacks have not significantly changed in recent years [29, 31]. This places research on attack mitigation and prevention in a particularly challenging environment.

Given the widespread landscape of DoS and DDoS attacks, most defense mechanisms require specific and up-to-date data to be effective [36]. This raises the question of how to represent DoS and DDoS attacks and what data to collect. One possible technique is to structure related data from attacks as fingerprints [2]. During an attack, DDoS attack traffic can be collected either from network flows or package captures (PCAPS). Existing tools can then be used to distill important characteristics of the attack traffic in the form of DDoS fingerprints [12]. These fingerprints therefore describe DDoS attack traffic data that can be used to uniquely characterize a DDoS attack. The data collected as fingerprints may include the type of attack, its duration, payload content, source IP addresses, traffic volume and frequency [32] [1]. This small and compact representation of an attack is easier to share and allows for a more efficient exchange of DDoS attack data. This, in return, can be used to update defense mechanisms to prevent attacks with similar patterns [3].

According to [46], DDoS defense mechanisms see an increasing number of cooperative approaches. Collaborative defense has emerged as a promising countermeasure against distributed attacks [40]. Unlike centralized defense approaches, collaborative defenses leverage the collective intelligence and resources of a network of participants. This promises benefits in information quality and offers better scalability of the defense architecture [27] [41] [62]. Moreover, it allows to move from a reactive to a proactive network-based mitigation and response approach [51]. However, the implementation of collaborative defenses also creates the necessity of storing and exchanging data amongst the participants of the network. This consequently requires the establishment of resilient frameworks for data sharing and management within the collaborative defense architecture.

2.1.4 Data Lakes

Data lakes are centralized repositories where structured, unstructured and binary data can be stored. The data is usually stored in its raw format. After the data has been gathered in a data lake, it can be transformed and used for various tasks such as reporting, visualization, advanced analytics and machine learning.

Data lakes have gained popularity in recent years by providing a single point for collecting, organizing and sharing data [61]. In many organizations, data lakes are utilized to increase data accessibility and ease of integration. With the reduction of cost for storing data in recent years [13], companies manage increasingly larger data volumes for analytics or to

store for undetermined future use [50] [26]. Therefore, the centralized nature of data lakes offers a viable solution for that need.

Although data lakes offer many advantages in storing and managing large quantities of data, their use has some potential drawbacks. The lack of structure of the data stored in a lake might make it difficult to understand, access, or search for [47]. This, in return, may raise issues with data quality and reliability. Additionally, the centralized nature of data lakes leads to vast amounts of data being stored in a single location. Therefore, data lakes can become disorganized and challenging to handle without adequate governance [45]. Similarly, centralized data repositories may be exposed to cybersecurity threats such as data breaches or hacking. It is crucial to implement appropriate security measures to safeguard against such risks.

For one, large organizations often rely on centralized data storage in lakes or warehouses. A central data team is then assigned to manage access to that storage and handle requests from product owners and other parties of the organization [10]. For another, large organizations also invest in domain-driven design and autonomous domain teams. Even though these domain teams know what data they need, they still have to act through the centralized data team to get the insights they require [10]. Oftentimes the data team then becomes a bottleneck, hindering the domain teams to get the timely insights they need.

2.1.5 Data Mesh Networks

A mesh topology describes an interconnection of nodes that can communicate with each other [43]. This idea can be extended to a data mesh network, where the data is owned and managed by the nodes that are part of the mesh. The utilization of standardized interfaces and APIs then allows for data owned by a node to be shared to the other nodes in the network.

Data mesh networks, therefore, offer a decentralized approach to managing data inside an organization [15]. Further, data mesh networks promote distributed architectures and domain-driven ownership of the data [18]. This clear distribution of ownership may increase data quality, as every node is only responsible for its share of the data. Data organization and structuring can therefore be done at the scale of the data held by a node. As the size of datasets grows, the computational power required for filtering, organizing and searching in the data also increases [34]. Therefore, breaking down the data into smaller subsets, nodes in a data mesh network, may decrease the overall computational power needed to filter, organize and search in the data.

Data mesh networks are a novel approach to promise decentralized data governance. However, data mesh networks have their own set of disadvantages and challenges. For one, they require more time and coordination to set up than typical data architectures. Establishing decentralized ownership and ensuring standards for data quality is central for the mesh to be functional. Additionally, for nodes to efficiently exchange data within the network, it is essential to establish standardized interfaces and APIs. These APIs and Interfaces must be effective for all the nodes in the mesh while still adhering to the initially defined standards. This may prove challenging if there is a significant difference

in how the data is structured between all nodes in the mesh. Distributed data ownership may also come with potential security risks, as every node is individually responsible for securing the data it holds.

Data mesh networks are novel approaches to data architectures that aim to solve problems associated with more traditional, centralized data architectures. While centralized approaches provide the benefits of data accessibility and ease of integration, they tend to suffer from disorganization and a lack of governance [45] [15]. On the other hand, decentralized data architectures come with a higher cost for the establishment but promise better data quality and the utilization of distributed hardware. Ultimately, the effectiveness of both types of data architectures heavily depends on the application domain.

In a collaborative DDoS defense architecture, a data mesh can be used to implement some of the requirements that were described in section 2.1.3 Characterizing DDoS Attacks. The data mesh provides each domain team with their own storage of DDoS attack data. Additionally, it allows for the standardized exchange of DDoS attack data between the domain teams. The below figure 2.2 outlines the idea behind a collaborative DDoS defense utilizing a data mesh. The figure displays multiple independent domain teams as part of a data mesh network. The domain teams hold DDoS attack data that can be joined together to create an overview of the attack. Insights gained from this attack overview may then be used to more effectively update the defenses against DDoS attacks.

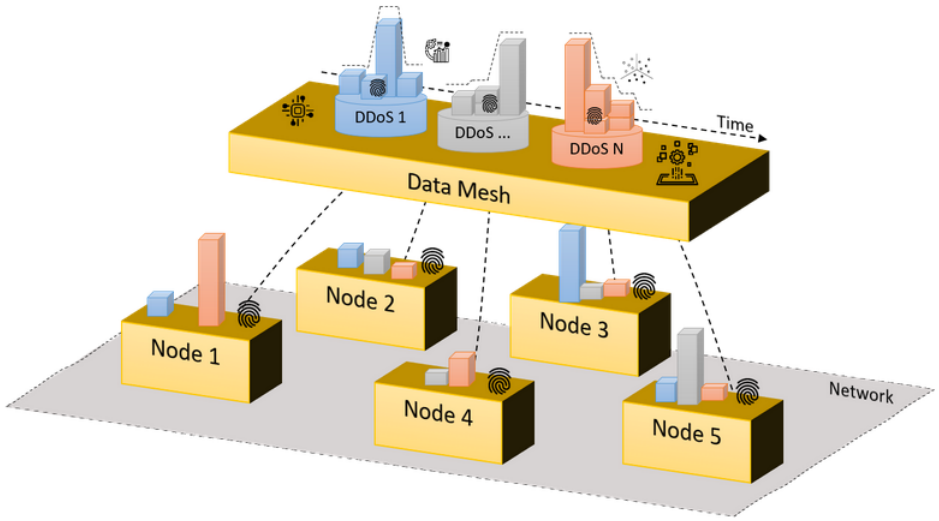

Figure 2.2: DDoS Data Mesh Network

2.2 Related Work

2.2.1 Existing Tools and Solutions

Data lakes have become popular for organizations looking to consolidate their data in a centralized repository in recent years. However, traditional data architectures like data lakes or warehouses face scalability, agility and ownership limitations as data sources become more complex and heterogeneous [**singhintercathedra**] [45]. In response to these limitations, the data mesh architecture has emerged as a new paradigm for building scalable and decentralized data platforms. Despite the growing interest in data mesh, there is a significant gap in research on this topic compared to traditional data architectures. Additionally, a lack of tools for implementing a data mesh network impairs its widespread adoption. The little research on existing data mesh networks suggests that the best approach to building your data mesh architecture is with a custom tech stack tailored to your specific needs [10] [14, 33]. In this related work section, we are focusing on existing tools that allow the implementation of a data mesh architecture. Next, we discuss whether these tools may be used to implement a data mesh network to share and organize DDoS-specific data.

In the below table 2.1, we have provided an overview of available tools that can be used to implement a data mesh architecture. The table is organized into three columns: Technology, Storage Principle and Source Availability. The Technology column lists the name of each tool under consideration. The Storage Principle column describes the type of data storage each tool uses. Some of the stacks listed below make use of distributed file systems. While the file systems themselves can not be considered decentralized, their combination with other tools and their position in the data mesh architecture might make the solution decentralized. The Source Availability column indicates whether the tool is open-source or proprietary.
The tools listed below can be grouped into three categories: Single tool solutions, pre-built stacks and tools for custom stacks. Single tool solutions are tools and products that allow the implementation of a data mesh architecture using a single tool. Pre-built stacks are combinations of tools recommended by existing research and may be used to implement a data mesh architecture [10]. Tools for custom stacks include tools that might be used to implement single parts of a data mesh architecture.
Some of the technologies listed in the table can not strictly be assigned to either decentralized or distributed and open or closed source. The Source Availability column entries with the value N/A denote that the technology is neither open nor closed source. BigQuery and Amazon Athena are managed services where the underlying architectures and codebases are not publicly accessible. However, both tools provide a public API that enables users to interact with their data using a variety of programming languages, tools and integrations. The API documentation is publicly available and in the case of BigQuery, a large community of users contributes to its development and improvement. The Confluent Data Mesh Prototype is not a fully-fledged software product but a prototype showcasing some of the key concepts and ideas behind data mesh architectures. As such, it can not be classified as open-source or closed-source software. The Storage Principle column entries with the value N/A denote that the technology does not provide

the ability to store data. Therefore, they can not be assigned to either be decentralized or distributed. BigQuery, Trino and Starburst are query engines and do not offer the ability to store data themselves. Similarly, Grafana and Apache Superset are data visualization and BI tools whose primary functionality is not data storage.

While this table displays an overview of helpful tools, it is not complete. As previously mentioned, existing research on data mesh architectures suggests building a custom solution tailored to the needs of a user. This can be done through a seemingly infinite amount of combinations of existing tools. Future research might expand on this thought and further evaluate the usability of existing and newly released tools in a data mesh architecture.

	Technology	**Storage Principle**	**Source Availability**
Single Tool Solutions	Dataplex [20]	Distributed	Closed Source
	Confluent Data Mesh Prototype [11]	Decentralized	N/A
Pre-Built Stacks	Google Cloud [21]	Distributed	Closed Source
	BigQuery [22]	N/A	N/A
	AWS S3 [5]	Distributed	Closed Source
	Amazon Athena [4]	Distributed	N/A
	MinIO [35]	Distributed	Open Source
	LakeFS [30]	Distributed	Open Source
	Trino [57]	N/A	Open Source
Tools for Custom Stacks	Starburst [49]	N/A	Closed Source
	InterPlanetary File System (IPFS) [25]	Decentralized	Open Source
	Trino [57]	N/A	Open Source
	Grafana [23]	N/A	Open Source
	Apache Hadoop [6]	Distributed	Open Source
	MySQL [39]	Centralized	Open Source
	Apache Superset [7]	N/A	Open Source

Table 2.1: Table of potential tools for the implementation of a DDoS data mesh architecture.

2.2.2 Requirements for the DoS Data Mesh Architecture

Our data mesh architecture has several requirements to be considered when evaluating potential tools and technologies. Firstly, in line with the core principles of a data mesh, it must be fully decentralized. This means that each domain team should own their data and be able to make autonomous decisions concerning how it is stored, managed and used. Secondly, we prefer on-premise data storage as it is more cost-effective and easier to test and implement. The data shared across the data mesh network will solely consist of DDoS fingerprints. As explained in chapter 2.1.3, these fingerprints only represent the necessary key performance indices needed to categorize the attack data. As a result, the DDoS fingerprints are not very large. Nonetheless, we value the performance of the data mesh as an important requirement, as it will heavily impact the data discovery experience down the line. Finally, the mesh should provide the option to implement monitoring capabilities later on, allowing us to monitor data quality and performance closely.

2.2.3 Single Tool Solutions

While searching for tools that might be used to implement a data mesh architecture, we noticed a distinct lack of single tool solutions. This could be due to the relatively novel nature of data mesh architectures or the difficulty in building a single solution that can cater to all the requirements of such a complex system. Although there are some single tool solutions available that can be used for building data mesh architectures, they tend to be rather restrictive. Dataplex and the Confluent data mesh prototype exclusively store data on the cloud. This lack of alternative storage options might not suit users looking for an on-premise data mesh solution. Further, these solutions introduce additional restrictions from cloud providers, such as rate limits or object size limits. As a result, we have determined that these limitations are unsuitable for implementing a data mesh architecture in our use case.

2.2.4 Pre-built Stacks

The pre-built stacks recommended by existing research on data mesh architectures [10] offer a more flexible alternative to the single tool solutions discussed above. On the one hand, the stacks of Google Cloud + BigQuery and AWS S3 + Amazon Athena take a similar approach to cloud-based storage as the previously discussed single tool solutions. We deem these stacks unsuitable for our use case for the same reasons of inflexibility and limitation.
On the other hand, the stack consisting of MinIO + LakeFS + Trino offers an alternative suitable for our use case. The stack utilizes MinIO as an object store with LakeFS as an independent hierarchical file system on top of the underlying object storage. Within a data mesh framework, each domain team offers access to their data through data products consisting of one or multiple SQL tables. To simplify storage and accessibility, a single LakeFS repository can store all data products for a domain team. These products can then be accessed via a shared SQL schema. Each data product is stored in its own directory tree within the LakeFS file system. Using standard SQL joins, Trino can then be used to merge data products from different domains [10]. This solution therefore offers the possibility of a decentralized data mesh architecture that is free from any limitations of cloud providers. Figure 2.3 shows a possible topology of a data mesh architecture using MinIO, Trino and LakeFS.

2.2.5 Tools for Custom Stacks

The list of tools available for constructing a custom stack for data mesh architectures is where the table is most incomplete. According to the requirements above, our data mesh mainly needs to ensure two core functionalities: The ability for the mesh nodes to store data and a way to query that data in a standardized manner.

Apache Hadoop offers one possible approach to provide the nodes of the mesh with data storage. Every domain team would manage one Hadoop repository in a data mesh architecture. Hadoop is a distributed, open-source platform that allows for storing and

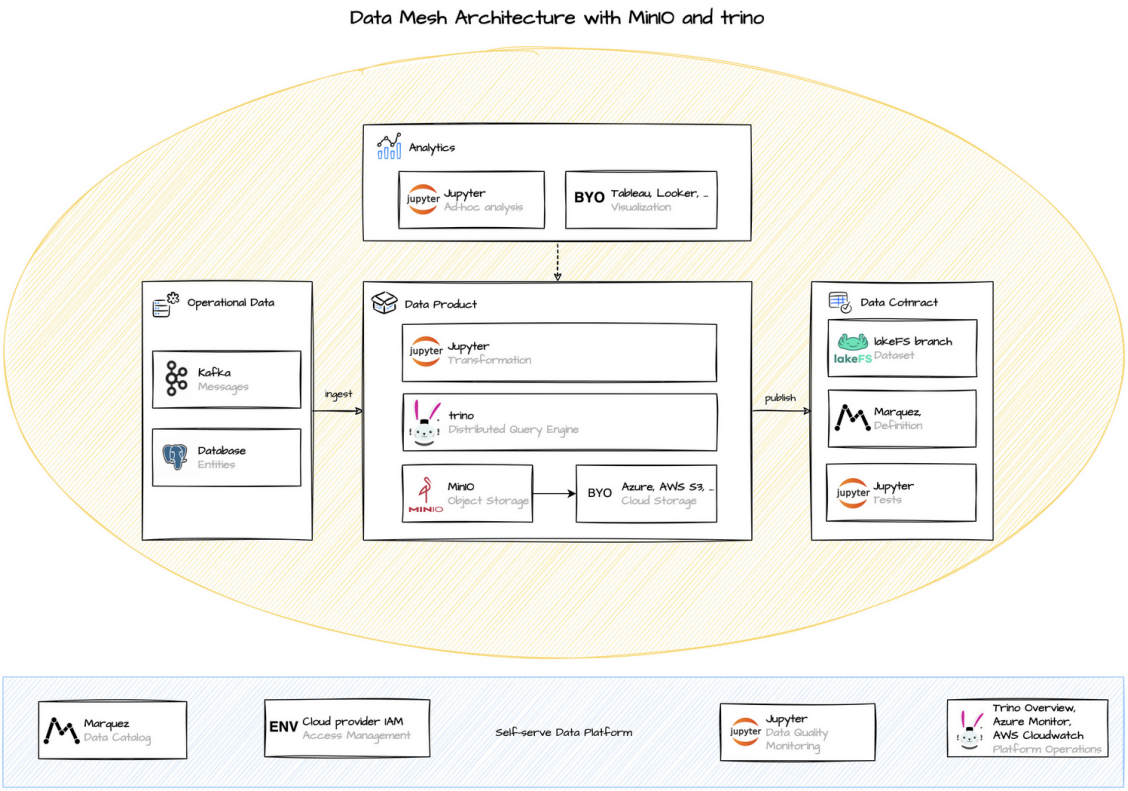

Figure 2.3: Data Mesh Architecture Topology with MinIO and Trino [28]

processing large datasets across clusters of computers using simple programming models [6]. The platform benefits from its long-standing position as a popular tool for implementing data lakes, data warehouses and other data architectures. Therefore, Hadoop offers a multitude of related projects, such as Apache Ambari. These related projects may be helpful in the implementation of a data mesh architecture, depending on the use case. MySQL offers another possible approach to handle the storage of DDoS attack data at the domain teams. MySQL is a relational database management system (RDBMS) that is commonly used for managing structured data in a centralized manner [39]. The RDBMS itself is not decentralized. However, if each domain team manages their own MySQL instance to store DDoS data, the overall storage of the data mesh network is decentralized.

Trino, Starburst and the InterPlanetary File System may be utilized to allow communication between the different nodes in a mesh architecture. Trino, formerly known as PrestoSQL, is a distributed query engine. It uses SQL-like syntax and high concurrency to quickly query large datasets across multiple data sources such as Hadoop, Cassandra, MySQL and others. It allows users to access and analyze data in real-time by leveraging distributed processing across multiple nodes in a cluster [53]. In a data mesh architecture, Trino can query the decentralized repositories of each domain team in a standardized way, similar to the pre-built stack of MinIO + LakeFS + Trino described above. Starburst is an enterprise-grade, fully supported version of Trino [49]. Although Starburst Enterprise and Trino share the same underlying technology, Starburst Enterprise offers additional features and services that make it better suited for enterprise-level applications.

The InterPlanetary File System (IPFS) is a network protocol that enables decentralized and distributed peer-to-peer sharing and storage of hypermedia through a distributed file system [25]. It allows users to access and share files without relying on centralized servers or hosting services. It utilizes a content-addressed system that allows data to be identified and accessed based on its unique hash [25]. In a data mesh architecture, IPFS can access decentralized data repositories by allowing data to be stored and accessed via a distributed file system. Each domain team would establish its own IPFS node for sharing and storing data. Other teams can then access this data by searching the IPFS network for the relevant node and obtaining the data. This approach allows data to be stored and accessed without needing a central server or data warehouse, resulting in a more robust, expandable and decentralized system. Additionally, IPFS delivers features like versioning, immutability and content-addressed storage, which can help ensure data integrity and traceability across the mesh.

Grafana and Apache Superset are open-source platforms designed for data exploration, visualization and analytics. Grafana specializes in real-time monitoring and observability, making it ideal for visualizing metrics, logs and time-series data [23]. Apache Superset offers support for data exploration, analysis and business intelligence [7]. Grafana supports many data sources like databases, cloud services and monitoring systems, providing a wide array of visualization options and interactive dashboards. It is advantageous in use cases that involve infrastructure monitoring, application performance monitoring and IoT data visualization [23]. Meanwhile, Apache Superset offers a user-friendly interface and versatile visualization choices to gain insights from complex datasets. It caters to various data sources, including databases, data lakes and popular data platforms. This facilitates the creation of interactive dashboards, reports and visualizations. It is commonly employed for ad-hoc analysis, data exploration and generating data-driven reports and visualizations [7].

2.2.6 Discussion

The existing research on data mesh architectures has provided us with a foundation of tools that we can utilize to design and implement our DDoS data mesh network. It is worth noting that the current body of research indicates a scarcity of readily available solutions for constructing a data mesh network, particularly ones that can be customized to suit our specific DDoS defense architecture. The single tool solutions found lack flexibility and customizability, which renders them unsuitable for our specific requirements in building a DDoS data mesh network. Similarly, tools from pre-built stacks tend to exhibit the same limitations in terms of adaptability. Many tools found in the existing body of research heavily depend on cloud storage. In our DDoS data mesh network, however, we want to allow the domain teams to make autonomous decisions with regard to how data storage is handled. While cloud-based storage can therefore not be ruled out entirely, we do not want to enforce it either. Therefore, for the design and implementation of our data mesh, we will primarily focus on tools that specialize in individual functionalities of the mesh. The above-mentioned domain-team-autonomy inside the data mesh also creates a challenge for the efficient exchange of the data across the mesh. To conquer this challenge, we will incorporate Trino in the design and implementation of the data mesh.

Trino specifically suits our needs by allowing for standardized queries across multiple, heterogeneous, storage instances. This provides the key functionality of data exchange across the data mesh network while offering the greatest amount of flexibility in data storage.

Both Grafana and Apache Superset can be used to implement the data discovery requirements needed for the second goal of this thesis. However, Apache Superset seems to offer better integration into some of the tools considered above. This is mainly due to better community support for libraries required to connect to distributed query engines such as Trino. Additionally, Superset offers more features for data discovery and visualization, which makes it better a better fit for our use case.

In conclusion, there is a distinct lack of readily available tools to fully implement a data mesh architecture for our use case. However, single existing tools can be utilized to implement certain aspects of the data mesh. For our use case, we will use Trino to provide the data exchange functionality of the data mesh network. Since Trino supports a host of data sources, this also grants a high degree of autonomy in the choice of the data storage for the domain teams. This allows us to design a data mesh capable of fulfilling all the requirements without compromising on autonomy.

Chapter 3

Design

3.1 Requirements

In this section, this thesis lists the requirements that must be fulfilled: The implementation of a data mesh network to share DDoS attack data and the implementation of a data discovery architecture that acts upon the data exchanged in the data mesh. While we have already touched on the requirements of the data mesh architecture in section 2.2.2 of the thesis, the following requirements also include the data discovery aspect of the thesis. Generally, the requirements for the complete architecture can be divided into the following three groups: Requirements for the data storage, exchange and discovery architecture.

Requirements for the data storage architecture:

- Decentralization: In line with the core principles of a data mesh, it must be fully decentralized. The domain teams of the data mesh are treated as autonomous instances, which can make autonomous decisions concerning how their data is stored, managed and used.
- Allows storing DDoS-related attack data at each domain team in the data mesh.

Requirements for the data exchange architecture:

- Allows exchanging DDoS-related attack data to provide an overview of the distributed attack.
- The performance of the data exchange does not negatively affect the data discovery process.
- Allows monitoring of the data exchanges, reducing the obscurity of the architecture.
- Allows integrating data visualization and BI (Business Intelligence) tools.

Requirements for the data discovery architecture:

- Act upon the data exchanged in the data mesh.
- Explore key fields of DDoS Fingerprints shared across the data mesh network.
- Grouping and aggregation of key fields from the fingerprints.
- Creation of metrics and visualizations to communicate an overview of the attack data.
- Allow for the intermediate storage of the data exchanged in the mesh. This reduces the traffic the data mesh experiences in the case of re-queries.

3.2 Assumptions

This thesis aims to expand a decentralized, collaborative DDoS defense architecture with the ability to share DDoS fingerprints and gain insights on attack overviews. Consequently, the design outlined below relies on other aspects within the collaborative defense setup.

First, we assume that DDoS data has already been captured and processed to generate fingerprints. These fingerprints serve as crucial indicators for identifying and classifying DDoS attacks. Therefore, these fingerprints are what is exchanged across the data mesh. Secondly, we assume these DDoS fingerprints have been stored according to a predefined storage schema at each participating party within the collaborative defense framework. The storage schema is known across the data mesh. This ensures a homogeneous representation of the data shared across the data mesh, simplifying the exchange of fingerprints across the architecture. Therefore, we can assume the presence of DDoS fingerprints stored within relational databases at the respective domain teams of our data mesh.

3.3 Architecture

The design presented in this section encompasses the integration of a data mesh architecture and data discovery capabilities. With this design, we aim to enable efficient querying of DDoS fingerprints stored across decentralized domain teams. Additionally, it facilitates the utilization of data discovery tools to analyze the collected fingerprints. This enables the user to get an overview of a DDoS attack. In line with the requirements for the architecture, the design can be divided into three parts: The data storage, the data exchange and the data discovery architecture.

3.3.1 Data Storage Architecture

In our collaborative DDoS defense architecture, storage of DDoS data is crucial for the overall functionality of the system. As the architecture of a data mesh network follows a

decentralized approach, the storage of this data is similarly decentralized. It is important to note that this decentralization refers to the position of the domain teams inside a data mesh. The storage implementation may take many topologies inside the domain teams themselves. Consequently, each domain team assumes responsibility for managing their storage, granting them autonomy in making storage utilization and practice decisions. The primary data stored within these decentralized repositories are DDoS fingerprints. Therefore, each domain team stores its perspective on DDoS attacks. This reflects their individual experiences from being the subjects of these attacks. These different perspectives can then be combined to create a more complete dataset.

3.3.2 Data Exchange Architecture

At the core of the collaborative DDoS defense architecture lies the functionality of exchanging attack-related data among the domain teams. To achieve this, a robust and efficient data exchange architecture is required. This architecture should be stateless, ensuring performance and scalability. Its primary objective is to enable each domain team to gather and aggregate data from other teams, allowing them to gain a comprehensive local overview of the combined dataset. However, the decentralized architecture and the autonomy granted to the domain teams present significant challenges to the data exchange process. It necessitates the establishment of standardized interfaces for querying and aggregating data stored in the repositories of the domain teams. These essentially serve as the interfaces allowing access to the data storage repositories maintained by each domain team. Standardizing how data is retrieved from the domain teams facilitates data aggregation from multiple domains. This standardized approach serves as the foundation for the data exchange architecture.

In the event of an attack, only the underlying data stored in the data storage architecture is updated, while the interface provided by the data exchange architecture remains consistent. This design allows for seamless data integration and analysis without disruptions, ensuring a continuous and reliable flow of information throughout the collaborative DDoS defense architecture. Once the data is queried and obtained through the data exchange architecture, it can be further processed and analyzed in the data discovery architecture.

3.3.3 Data Discovery Architecture

The data discovery architecture enables the aggregation and analysis of key performance indicators (KPIs) derived from the data obtained through the data exchange architecture. This component is fundamental to every domain team within the collaborative defense architecture. It facilitates access to the exchanged data by directly retrieving the stored data resulting from queries made through the data exchange architecture or by integrating the querying capability of the data exchange architecture within its functionality.

Essentially, this component exists at each domain team, providing the means to access the exchanged data through the data exchange architecture or directly interact with the data exchange architecture to retrieve the required data. Once the data is obtained,

this component also enables the analysis of key data fields. This, in return, allows for extracting valuable insights from the combined dataset. In a collaborative DDoS defense, these insights can then be used to update the defense at each party of the collaborative defense.

3.3.4 Design Overview

Figure 3.1 depicted below provides an overview of the topology of the design. It depicts a collaborative DDoS defense architecture that consists of three parties or domain teams. Each domain team implements the three core services from the design: Data storage, data exchange, and data discovery. The data exchange interfaces act as intermediaries between visualization and storage services. They serve as the gateway through which domain teams can access the distributed data storages the respective parties maintain. Data exchange interfaces are accessed inside the data discovery services. In the topology below, every domain team has access to the data exchange interfaces of all other domain teams. Therefore, every domain team has access to the data repositories of all other domain teams. This allows the local aggregation of decentralized data. Since the data exchange interfaces are accessed from within the data discovery services, the aggregated data can be used for analysis and the creation of visualizations.

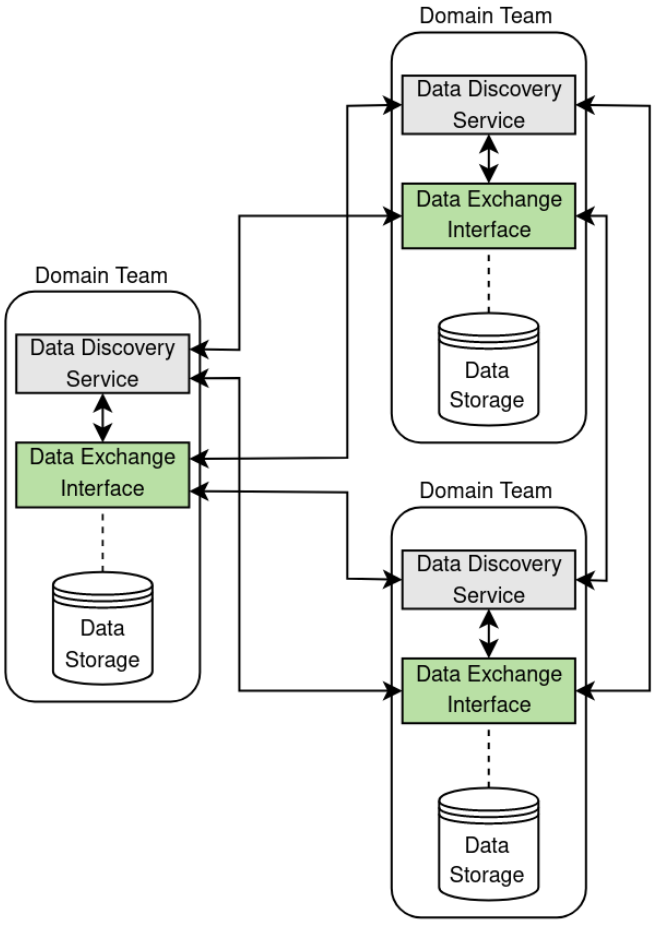

Figure 3.1: Mesh Design within Cooperative Domains

3.4 DDoS Analytics

The design enables each domain team to access and analyze an extended dataset by ensuring the implementation of three core services within the data mesh network. To illustrate, consider the following example within the collaborative DDoS defense architecture. Each party involved represents a domain team within the data mesh network. When a DDoS attack occurs, all domain teams locally store network metrics as DDoS fingerprints. One of the domain teams aims to analyze which ports were most affected to update their DDoS defenses accordingly. To reach the goal, our design outlines a five-step process for the domain team to follow:

1. Formulate a query that retrieves DDoS fingerprints from all domain teams.
2. Submit the query to the data exchange interfaces. These return the DDoS fingerprints stored for the respecting domain team.
3. Calculate the result data that contains which ports were hit and how many times. This is now based on the extended DDoS fingerprint dataset that includes fingerprints from all domain teams.
4. Create visualizations based on the result data calculated.
5. Update the DDoS defenses based on the insights gained from the analysis. Implement appropriate measures to mitigate future attacks targeting the identified ports.

The below sequence diagram 3.2 depicts the five steps explained above. It shows the interactions between the core services running on each domain team. Note that the visualization only depicts the services needed for the above example use case. In the proposed design, every domain team runs all three core services, as shown in figure 3.1. However, to reduce clutter in the visualization, the data discovery services on domain teams two and three and the data storage and exchange interface on domain team one have been omitted. Also note that in this use case, the data discovery service allows to write and submit the queries needed to retrieve the DDoS Data. Whether the data discovery service directly supports the writing and submitting of queries will depend on the actual implementation of the service.

Following this process, domain teams can utilize the combined data stored within each domain team. This allows them to gain valuable insights into the characteristics of DDoS attacks. The proposed design enables domain teams in the collaborative DDoS defense architecture to achieve the main goal of the thesis. It offers a comprehensive data mesh architecture that facilitates discovering and analyzing attack-related information.

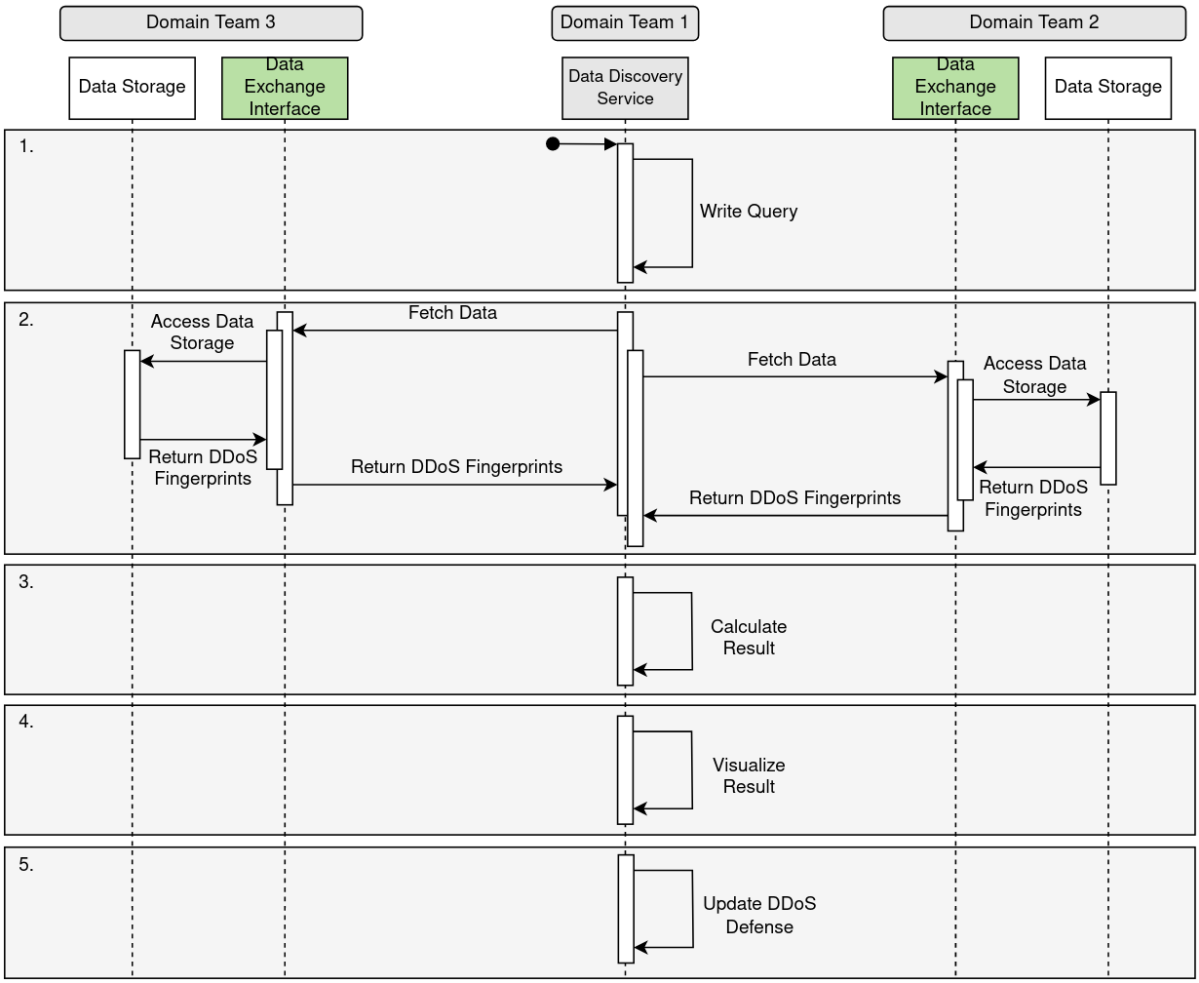

Figure 3.2: Mesh Design Interactions within Cooperative Domains

Chapter 4

Implementation

4.1 Tools

This Chapter focuses on implementing the design proposed in the previous chapter. With the tools described in this section, we aim to fulfill the requirements outlined in section 3.1. To store the DDoS Fingerprints, we have opted for MySQL databases. This choice was driven by the simplicity of converting the fingerprints from their JSON format into the tables supported by MySQL. Additionally, we have integrated Trino as a distributed query engine. Trino facilitates the exchange of fingerprints across the data mesh network in an ad-hoc manner. Furthermore, Apache Superset has been chosen as our data visualization and business intelligence tool. Superset offers seamless integration with Trino within the data mesh framework. In the upcoming sections, we will delve into the core functionalities of Trino and Superset, providing the necessary knowledge to comprehend the implementation fully.

4.1.1 Trino

Trino is a distributed query engine that can query large datasets distributed across one or more heterogeneous data sources. Trino is deployed as a cluster, allowing it to execute data processing operations in parallel across numerous servers. The tool consists of two types of instances: coordinators and workers [53]. The following sections provide a detailed exploration of these instances and other essential components that constitute the architecture of Trino.

4.1.1.1 Trino Cluster

A Trino cluster consists a coordinator and multiple worker instances. Users establish connections to the coordinator using SQL query tools. The coordinator interacts with the workers and together they access the connected data sources configured within catalogs.

The processing of each query is performed in a stateful manner. The coordinator orchestrates the workload and distributes it in parallel across all the workers in the cluster. Each node in the cluster runs Trino within a single JVM instance and the processing is further parallelized using threads [53].

4.1.1.2 Coordinators

The Trino coordinator instance parses statements, plans queries and manages Trino worker nodes. It acts as the central processing unit of the cluster and serves as the primary node to which clients connect to submit statements for execution. In any Trino setup, at least one coordinator and one or more worker instances must be available. The coordinator instance oversees the activities of each worker and coordinates the execution of queries. It constructs a logical model of a query consisting of multiple stages, which are then transformed into a series of interconnected tasks executed by the Trino worker cluster. Coordinators, workers and clients communicate via a REST API [53].

4.1.1.3 Workers

Trino worker instances are tasked with the execution of queries and the processing of data. The worker nodes retrieve data from connectors and exchange intermediate data among themselves. Once data has been retrieved from the connectors, the coordinator is responsible for retrieving the results from the workers and delivering the final results to the client. Upon startup, a Trino worker process advertises itself to the discovery server within the coordinator instance, making it available for task execution. Worker instances communicate with other worker and coordinator instances through a REST API [53].

4.1.1.4 Connectors

A connector in Trino serves as an interface between Trino and a specific data source, such as Hive or a relational database. Similar to how a driver facilitates the interaction with a database, a connector implements the SPI of Trino, enabling Trino to communicate with the underlying resource using a standardized API. Trino comes with several built-in connectors, including connectors for JMX, system tables, Hive and a TPC-H connector designed for TPC-H benchmark data. Numerous third-party developers have also contributed connectors, expanding the capability of Trino to access data from diverse data sources [53].

4.1.1.5 Trino Dashboard

Trino offers a web interface (UI) that allows users to monitor the Trino cluster and manage queries. The UI is accessible over HTTP or HTTPS and uses the port number specified in the *config.properties* of the coordinator node. On the dashboard, you can view the execution times, status, resources and query plans of queries ran on the cluster. Screenshots of the dashboard can be found in the appendix of the thesis (A.1, A.2, A.3).

4.1.2 Apache Superset

Apache Superset is an open-source data exploration and visualization platform. It provides an interface and a wide range of interactive visualizations that allow users to explore and analyze data from different sources. Superset can be deployed in various ways. For example, locally, using the available docker-compose, or on Kubernetes, using the available helm chart [7]. Superset runs as a web application that users can interact with through a configured URL.

4.1.2.1 Database Connections

Superset can be configured to retrieve data from various data sources. The app does not come pre-installed with built-in connectivity to databases, except for SQLite, which is included in the Python standard library. To connect to other databases, you are required to install the needed packages for that database. A list of compatible and preferred databases and the installation instructions for the required packages can be found in the documentation of Superset [7].

4.1.2.2 SQL Lab

What sets Superset apart from many other data visualization and BI tools is the ability to use SQL statements to retrieve the data necessary for your data discovery needs. The SQL Lab is a SQL IDE integrated into the web application of Superset. This allows you to query the data sources previously configured as database connections within the SQL Lab. Once a query is complete, the resulting data is available inside Superset. Returned data can also be stored in the form of datasets. This allows the reuse of data without having to fetch it again from the database. Multiple visualizations can be grouped into dashboards.

4.2 Storage Implementation

Data storage inside the domain teams should be flexible. Since this implementation utilizes Trino to query the data, the available tools to store the fingerprints are limited to those supported by Trino. Trino supports a list of both relational, as well as NoSQL connectors. A complete list can be found in the documentation of Trino [54]. For our implementation, we use MySQL instances running on each domain team. This decision was mainly driven by the fact that MySQL is a relational database. This allows the storage of DDoS fingerprints according to a uniform schema. As a result, querying and aggregating the data becomes easier down the line. Even though Trino can query heterogeneous data sources, we do not utilize that feature in this implementation. Since the data is homogeneous in our use case, we can use homogeneous data sources. Introducing heterogeneous data sources would add unnecessary complexity to this proof of concept architecture.

4.3 Data Exchange Implementation

The core functionality of the data mesh is the exchange of DDoS Fingerprints across domain teams. As mentioned before, our design utilizes Trino to guarantee this functionality. In our DDoS data mesh architecture, every domain team runs its own Trino instance. One domain team is selected as the Trino coordinator instance among the data mesh nodes. While this instance could simultaneously also be a worker node, it is recommended to have a dedicated coordinator node [55]. However, this does not exclude the domain team running the coordinator instance from storing DDoS data. All other domain teams inside the data mesh are configured as Trino worker instances. All Trino instances are configured to access all the fingerprint repositories of all domain teams. Queries regarding the exchange of DDoS fingerprints are sent to the coordinator node. The coordinator node then optimizes that query and creates a query plan for the worker nodes to execute. Once the query is completed, the coordinator node returns the result to the client that submitted the query.

In our implementation of the DDoS data mesh network, every domain team is therefore running the following services:

- A MySQL instance storing DDoS fingerprints
- Either an instance of a Trino worker node or an instance of a Trino coordinator node

The below figure 4.1 outlines the topology of our DDoS data mesh design. The figure depicts a data mesh with three domain teams. Each domain team is running a local, decentralized data storage. In that storage, the DDoS fingerprints are stored. Each domain team is also running a Trino instance. In the topology depicted, there are one coordinator node and two worker nodes. The coordinator and worker instances can access all the fingerprints repositories of all domain teams. This is depicted with the dashed line. A client queries the coordinator node to retrieve joint data from all domain teams. The coordinator node then optimizes the query and creates a query plan. The query plan is then distributed amongst the worker nodes which fetch the data. The result is returned to the coordinator node, which returns the final response to the client.

4.4 Data Discovery Implementation

Additionally to data exchange functionalities, the DDoS data mesh architecture should also include data discovery capabilities. The idea is to directly feed the queried data from the data mesh into one tool capable of handling the requirements specified above. In our implementation, we utilize Apache Superset to guarantee that functionality. Each domain team inside the data mesh network runs its instance of Superset. It would be possible to only run one instance of Superset on one of the domain teams. That instance could then be made public across the data mesh network. The other domain teams could then use that Superset instance to satisfy their data discovery needs. However, we have

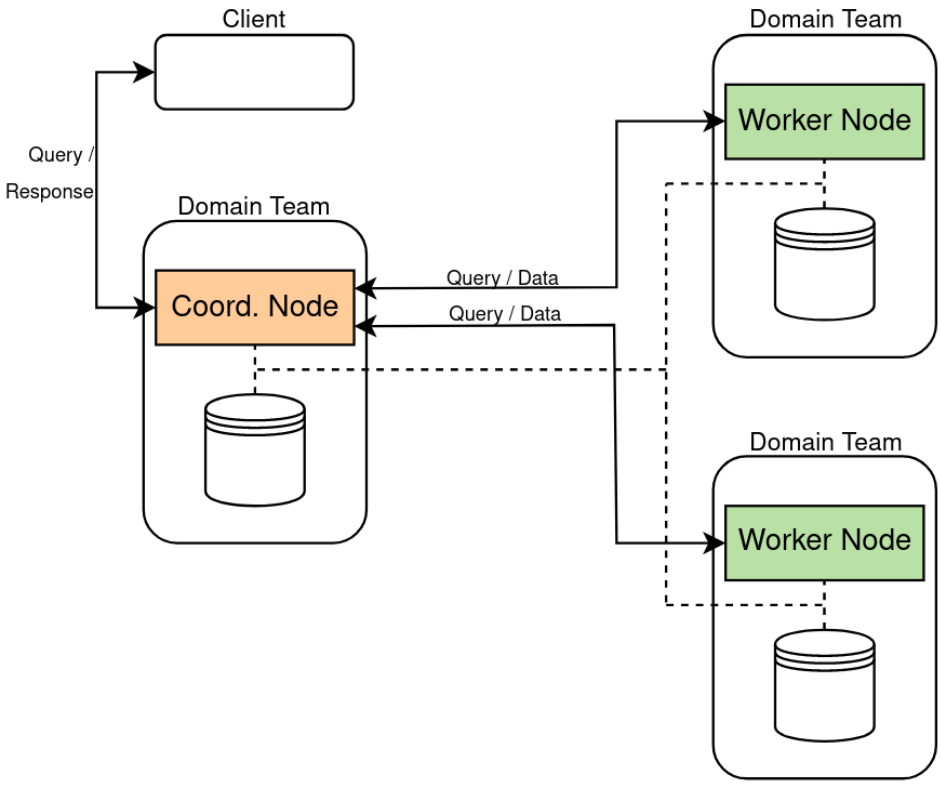

Figure 4.1: Data Mesh Architecture Topology with Trino

decided against that implementation to make the architecture as distributed as possible. Running only one instance of Superset creates a centralized dependency for all domain teams. Similarly to how dependencies on centralized data teams lead to bottlenecks in centralized data architectures, a central Superset instance may create a bottleneck for the data discovery needs of the domain teams. As a result, each domain team runs its own instance of Superset.

In the configuration of the Superset instances, the coordinator node is specified as a database connection. While the coordinator node of a Trino cluster is not a database instance, it is treated as one by Superset. This allows configuring a Trino coordinator instance as the recipient of the queries ran from within Superset. Since Superset also allows the formulation of SQL queries inside the SQL Lab, we can seamlessly integrate this data discovery platform on top of our previously implemented DDoS data mesh architecture. Figure 4.2 depicts the topology of our data mesh architecture with the inclusion of data discovery capabilities. Every Domain team now runs a Superset instance that can send queries to the coordinator node of the Trino cluster.

4.5 Implementation Overview

To provide an overview of the whole architecture, the below figures 4.2 and 4.3 describe the topology and interactions of the DDoS data mesh implementation. In line with the

assumptions in chapter 3.2, we assume that the domain teams hold relevant DDoS data collected after a DDoS attack. We also assume the same topology described in figure 4.1 consisting of three domain teams. All domain teams hold their local view of the DDoS attack in the form of DDoS fingerprints inside a relational database that is part of their domain team. Each domain team runs an instance of Trino and an instance of Superset. There are two Trino worker instances and a dedicated coordinator instance. In every Superset instance, the coordinator node has been specified to receive queries executed from within Superset. The purple arrows depicted in figure 4.2 indicate the ability of each domain team to query the coordinator node and retrieve data from the data mesh.

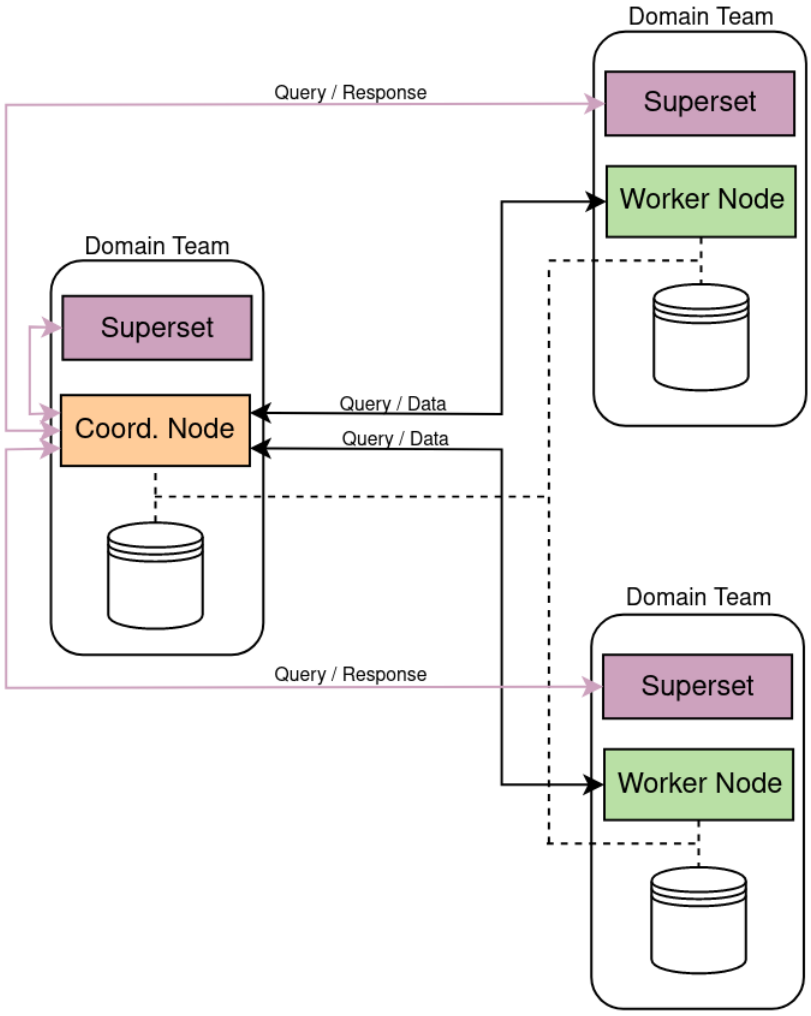

Figure 4.2: Topology of the Data Mesh including Data Discovery Capabilities

The sequence diagram 4.3 shows the flow of interactions across the DDoS data mesh architecture. The diagram depicts the following example use case: Domain team two submits a query to join the DDoS fingerprint tables from all three domain teams. The query is sent to the Trino coordinator instance on domain team one. The coordinator instance optimizes the query and creates a query plan. The query plan is then distributed to the worker instances running on domain teams two and three. The worker instances fetch the data from the respective data sources. Once the data has been retrieved, it is aggregated and joined on the worker nodes as far as possible. The results are then returned

to the coordinator node, which returns the final result to the Superset instance of domain team two. From there on, domain team two can use the data to create a visualization depicting an overview of the attack that includes the information of all domain teams combined. In a further step, the insights gained from the visualization can be used to update DDoS defense mechanisms across the data mesh.

Note that for this example, the distribution of the query across the worker nodes has been chosen arbitrarily. In reality, assigning query partitions to worker nodes depends on many factors, including network latency, parallelizability and the minimization of data movement. Section 4.11 explains this in more detail.

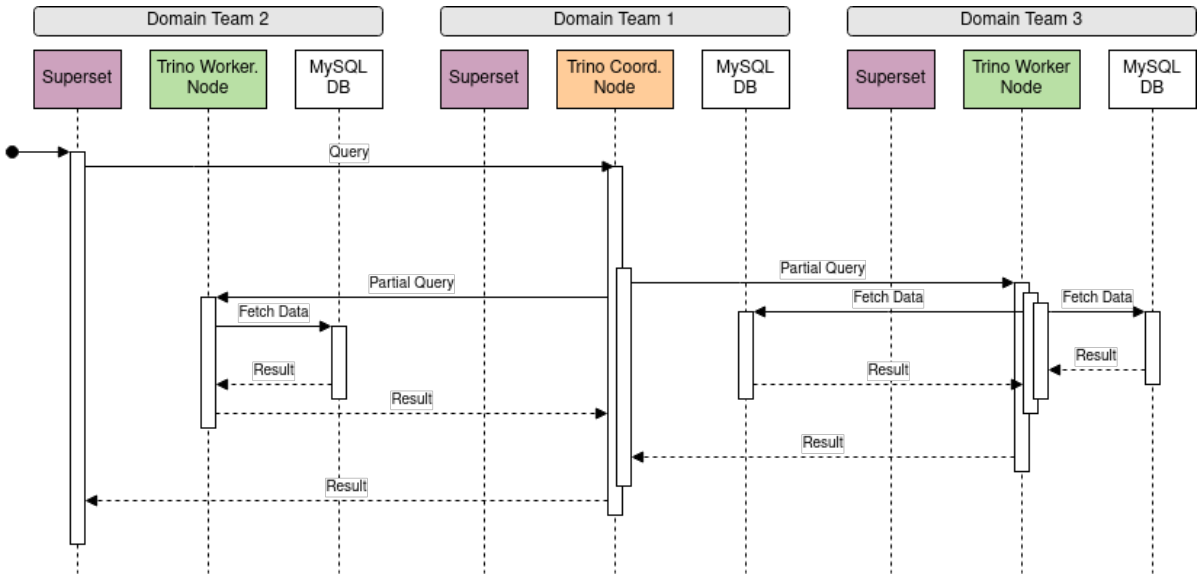

Figure 4.3: Data Mesh Architecture Sequence of Interactions

4.6 Deployment on Virtual Machines

For the deployment of the design proposed, we utilize three virtual machines. The specifications of the VMs are listed in table 5.1 of the Evaulation chapter. Each virtual machine simulates an independent domain team as part of a collaborative DDoS defense architecture. With three domain teams, we can simulate a collaborative defense architecture that includes multiple Trino worker nodes and a dedicated Trino coordinator node. Further, it allows us to have three separate, decentralized, storage instances where DDoS fingerprints are stored. This allows us to simulate a high degree of decentralization across the data mesh network. All virtual machines are part of a Tailscale zero-trust network. Inside the zero-trust network, the VMs strictly communicate through the IPs and domains provided by the network. Further, all services have been configured to use the IPs and domains provided by the network strictly. Therefore, the communication between Superset, Trino, and MySQL instances is not handled over public IP addresses. The Tailscale IP addresses for the VMs are provided in table 5.1. The inclusion of a zero-trust network allows us to almost completely shut down any access from the outside the network to the inside

of the network. This provides an additional layer of security for this proof of concept implementation.

4.7 Storage Deployment

To store the DDoS fingerprints, we deploy MySQL instances on each virtual machine within our architecture. This choice was driven by the simplicity of converting the fingerprints from their JSON format into the tables supported by MySQL. Each VM in the setup hosts a dedicated MySQL instance, configured with a user with local and remote access privileges to the schema and tables described below. Further, the bind address in the configuration file of the instances was modified to align with the Tailscale IP of the corresponding VM. Consequently, each MySQL instance can only be accessed via its network-specific IP address.

DDoS fingerprints are stored in the MySQL instances along the below schema (4.4). Every domain team (VM) holds a collection of DDoS fingerprints inside a MySQL instance. That instance can be accessed through a user with remote access privileges over the Tailscale IP of the respecting VM. The schemas have been named after the respecting VM hosted. For example, *vm2_ddos_data* is the schema name for VM2. This distinction is important because it makes it easier to understand where your data comes from inside the Trino query statement.

4.8 Trino Deployment

We have established a consistent setup across all virtual machines to deploy the Trino instances for our data mesh network. Each VM hosts a dedicated Trino instance, with specific roles assigned to different VMs. VM1 accommodates the Trino coordinator instance, while VM2 and VM3 are configured as Trino worker nodes. To ensure efficient deployment and management, we leverage Docker containers for running Trino on each VM. Trino provides a container image that facilitates the instantiation of Trino inside a container. To configure the Trino instances, we mount the configuration directory from the host machine into the Docker container. This approach allows us to conveniently modify instance configurations without having to access the container itself. Docker further allows us to force communication with the container through the IP provided by Tailscale. As a result, we avoid modifying the internal configuration of Trino to specify the desired IP address.

4.8.1 Trino Configuration

A Trino instance can be configured by providing the below files and directories inside an */etc* folder in the installation directory of Trino. In our case, the */etc* folder on the

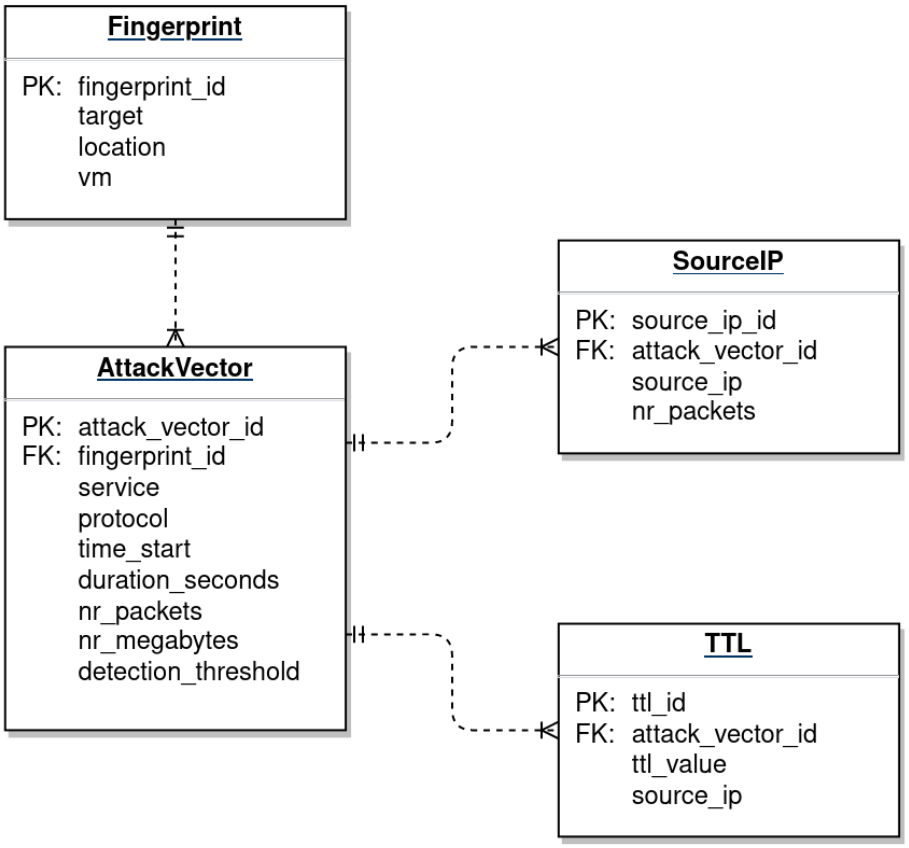

Figure 4.4: Database Schema for DDos Fingerprints

host machine is then mounted to */etc/trino* inside the Docker container. Trino automatically detects the mounted configurations folder and uses that instead of the default configurations.

- Config Properties: Configuration for the Trino server
- Catalog Properties: Configuration for connectors (data sources)
- JVM Config: Command line options for the JVM
- Node Properties: Node-specific environmental configuration

4.8.1.1 Config Properties

The configuration file, *etc/config.properties*, holds the settings for the specific Trino instance. Therefore, this is where it is specified whether that instance is a coordinator or a worker. Below are the configurations that we use for the coordinator and worker instances:

Coordinator Node: *etc/config.properties*

```
coordinator=true
node-scheduler.include-coordinator=false
```

```
http-server.http.port=8080
discovery.uri=http://100.76.172.123:8080
```

Listing 4.1: Trino Coordinator Configuration Properties

Worker Nodes: *etc/config.properties*

```
coordinator=false
http-server.http.port=8080
discovery.uri=http://100.76.172.123:8080
```

Listing 4.2: Trino Worker Configuration Properties

- `coordinator`: Allows a Trino instance to function as a coordinator. It can therefore accept queries from clients and manage query execution.
- `node-scheduler.include-coordinator`: Allows the instance to be treated as both a coordinator and a worker. However, in the case of larger clusters, executing tasks on the coordinator can reduce query performance since resources are not fully dedicated to the crucial responsibilities of scheduling, managing and monitoring query execution [55].
- `http-server.http.port`: Specifies the port for the HTTP server.
- `discovery.uri`: Is responsible for facilitating communication between Trino nodes. Each Trino instance registers itself with the discovery service upon startup and regularly sends heartbeats to maintain its registration. The discovery service shares the same HTTP server as Trino, using the same port.

4.8.1.2 Catalog Properties

Trino accesses data via connectors. Connectors are mounted in catalogs. The connector provides all of the schemas and tables inside of the catalog [53]. Catalogs are registered by creating a catalog properties file in the *etc/catalog* directory. To implement our DDoS data mesh, we configure one catalog property file for each MySQL instance. In line with the access patterns described in the design chapter, all Trino instances have access to all MySQL instances. The following three files are therefore configured on every instance:

/etc/catalog/vm1_data.properties

```
connector.name=mysql
connection-url=jdbc:mysql://100.76.172.123:3306
connection-user=
connection-password=
case-insensitive-name-matching=true
```

Listing 4.3: Trino Catalog Properties for Data Source: VM1

/etc/catalog/vm2_data.properties

```
connector.name=mysql
connection-url=jdbc:mysql://100.78.69.35:3306
connection-user=
connection-password=
case-insensitive-name-matching=true
```

Listing 4.4: Trino Catalog Properties for Data Source: VM2

/etc/catalog/vm3_data.properties

```
connector.name=mysql
connection-url=jdbc:mysql://100.93.249.10:3306
connection-user=
connection-password=
case-insensitive-name-matching=true
```

Listing 4.5: Trino Catalog Properties for Data Source: VM3

- `connector.name`: Has to match the name of the underlying storage tool used. A list of all connector names can be found in the documentation of Trino [52].
- `connection-url`: MySQL connection URL for remote access to the database.
- `connection-user`: The MySQL user with remote access privileges to the schema and tables.
- `connection-password`: The password of the database user.
- `case-insensitive-name-matching`: Allows using case-insensitive schema and table names.

The above configuration allows every instance of Trino to access all data sources inside the data mesh network. Further, all database connections are established over the IPs provided by the zero-trust network. Trino can access the configured catalogs through the dot notation. For example, you can access the *Fingerprints* table stored in the *vm1_ddos_data* schema of VM1 like this: `vm1_data.vm1_ddos_data.fingerprints`. Generally, database access follows the pattern `<catalog_name>.<schema_name>.<table_name>`.

4.8.1.3 JVM Config

The *etc/jvm.config* file is where you can specify the command line options for launching the Java Virtual Machine. This file follows a specific format, with each option listed on a separate line. It is important to note that the shell does not interpret the options in this file, so it is unnecessary to include quotes even if an option contains spaces or special characters. Except for the `-Xmx` entry the below fields represent the default settings recommended in the documentation of Trino [55] and have been specified for all VMs. `-Xmx` specifies how much memory you allow Trino to use on that node. We have adapted this value to fit the specifications of the hardware we use for the implementation.

/etc/jvm.config

```
-server
-Xmx2G
-XX:InitialRAMPercentage=80
-XX:MaxRAMPercentage=80
-XX:G1HeapRegionSize=32M
-XX:+ExplicitGCInvokesConcurrent
-XX:+ExitOnOutOfMemoryError
-XX:+HeapDumpOnOutOfMemoryError
-XX:-OmitStackTraceInFastThrow
-XX:ReservedCodeCacheSize=512M
-XX:PerMethodRecompilationCutoff=10000
-XX:PerBytecodeRecompilationCutoff=10000
-Djdk.attach.allowAttachSelf=true
-Djdk.nio.maxCachedBufferSize=2000000
-XX:+UnlockDiagnosticVMOptions
-XX:+UseAESCTRIntrinsics
# Disable Preventive GC for performance reasons (JDK-8293861)
-XX:-G1UsePreventiveGC
```

Listing 4.6: Trino JVM Configuration

4.8.1.4 Node Properties

The *etc/node.properties* file consists of configuration settings specific to each Trino node. A node represents a single installed instance of Trino on a machine. The below configurations have been specified for all Trino instances of the data mesh network:

/etc/node.properties

```
node.environment=bthtest
node.id=ffffffff-ffff-ffff-ffff-ffffffffffff
```

Listing 4.7: Trino Node Property Configuration

- `node.environment`: Specifies the name of the environment. All Trino nodes in a cluster must have the same environment name.
- `node.id`: This is the unique identifier for this installation of Trino. This must be unique for every node. We generated these IDs for the nodes in our data mesh with the `uuidgen` command on the VMs.

4.8.2 Running Trino

Every VM stores the above-specified configuration files inside the */etc/trino-server-416/etc* directory. We run the following command to start a Docker container running the Trino image:

```
sudo docker run --name trino -d -p 100.76.172.123:8080:8080 --volume /etc/trino-server-416/
etc:/etc/trino trinodb/trino
```

- `--name`: Name of the Docker process.
- `-d`: Run the container in detached mode.
- `-p`: Maps the Tailscale IP of the respecting VM and port 8080 to port 8080 inside the docker container. This forces all communication with the Trino instance through the IP of the zero-trust network. Note that this IP has to be changed for every VM when running the command above. The example provided lists the IP of VM1.
- `--volume`: Mounts the folder at */etc/trino-server-416/etc* on the host machine (VM) to the folder */etc/trino* inside the Docker container. This allows us to access the configurations for the Trino instance by accessing */etc/trino-server-416/etc* on the host machine.
- `trinodb/trino`: Specifies the image used to run the container. This is the official Trino image provided by the documentation [56].

4.9 Superset Deployment

Similarly to how we run a Trino instance on every VM, we also run a Superset instance on every VM. This allows our domain teams to act on the combined DDoS data without going through some centralized party. We also run Superset inside Docker containers for the same reasons of efficient deployment and management. The Apache foundation provides a git repository that contains all the necessary files to run Superset containerized [7]. For the deployment in our data mesh, we cloned the git repository on all VMs.

4.9.1 Superset Configuration

Once the git repository has been cloned, there are two changes that we have to configure. For one, Superset does not come pre-installed with built-in connectivity to databases. To use Trino with Superset, we must first install the necessary drivers. Since we are running Superset inside a Docker container, a simple `pip install trino` will not suffice. To install the Trino drivers inside the container, we add the `trino` library to the *docker/requirements-local.txt* file of the cloned repository. This ensures that the Trino library, including the necessary drivers to use Trino with Superset, is installed within the Docker container running Superset. The second change we have to configure is using the Tailscale IP. In the *docker-compose-non-dev.yml* file of the cloned repository, we change the service configuration for Superset to include the Tailscale IP of the respecting VM. We change the `ports` field to include the IP. Note that the IP has to be changed for each VM to match the assigned Tailscale IP for that VM. An excerpt of the service configuration for Superset running on VM1 is depicted below:

docker-compose-non-dev.yml

```
...
superset:
    env_file: docker/.env-non-dev
    image: *superset-image
    container_name: superset_app
    command: ["/app/docker/docker-bootstrap.sh", "app-gunicorn"]
    user: "root"
    restart: unless-stopped
    ports:
      - 100.78.69.35:8088:8088
    depends_on: *superset-depends-on
    volumes: *superset-volumes
...
```

Listing 4.8: Excerpt from Docker-Compose Configuration File

Once the above changes have been configured and Superset is running, we can add the Trino database connection. In the settings of the Superset web app, we add the following connection URL: `trino://timportmann@100.76.172.123:8080`

The connection URL points to the Trino coordinator node on VM1 on port 8080. We also pass a user to establish the connection with. For our configuration of Trino, this user can be any placeholder and is not configured in the Trino configurations. We will touch more on Trino users in the Performance Analysis section of the evaluation chapter (see 5.3.1)

4.9.2 Running Superset

The previously cloned repository contains a *docker-compose-non-dev.yml* that runs all necessary containers for Superset. We run the following command in that directory to start the containers: `sudo docker compose -f docker-compose-non-dev.yml up -d`

- `-f`: Specifies that we want to execute the docker compose file with the given path.
- `-d`: Runs docker compose in detached mode.

The docker compose file does not only run a single container with Superset. A total of six containers are started. These containers include the following images:

- `apache/superset`: superset_app
- `apache/superset`: superset_worker_beat
- `apache/superset`: superset_worker
- `apache/superset`: superset_init
- `postgres`: suberset_db
- `redis`: superset_cache

Once all containers are running, the Superset web application can be accessed at `http://<vm_tailscale_ip>:8088/login/`. Note that `<vm_tailscale_ip>` is a placeholder for the Tailscale IP address of the respecting VM where you run Superset.

4.10 Deployment Overview

The below deployment diagram depicts the services and their accessibility within our implemented architecture. Each service has been configured as explained above. VMs and Docker containers are represented by a rectangular box and are accessed through the IPs and ports depicted above them. In section 4.9.2, we have described how Superset starts multiple services inside separate Docker containers. In the deployment diagram, however, Superset is depicted as a single container to reduce unnecessary cluttering of the visualization.

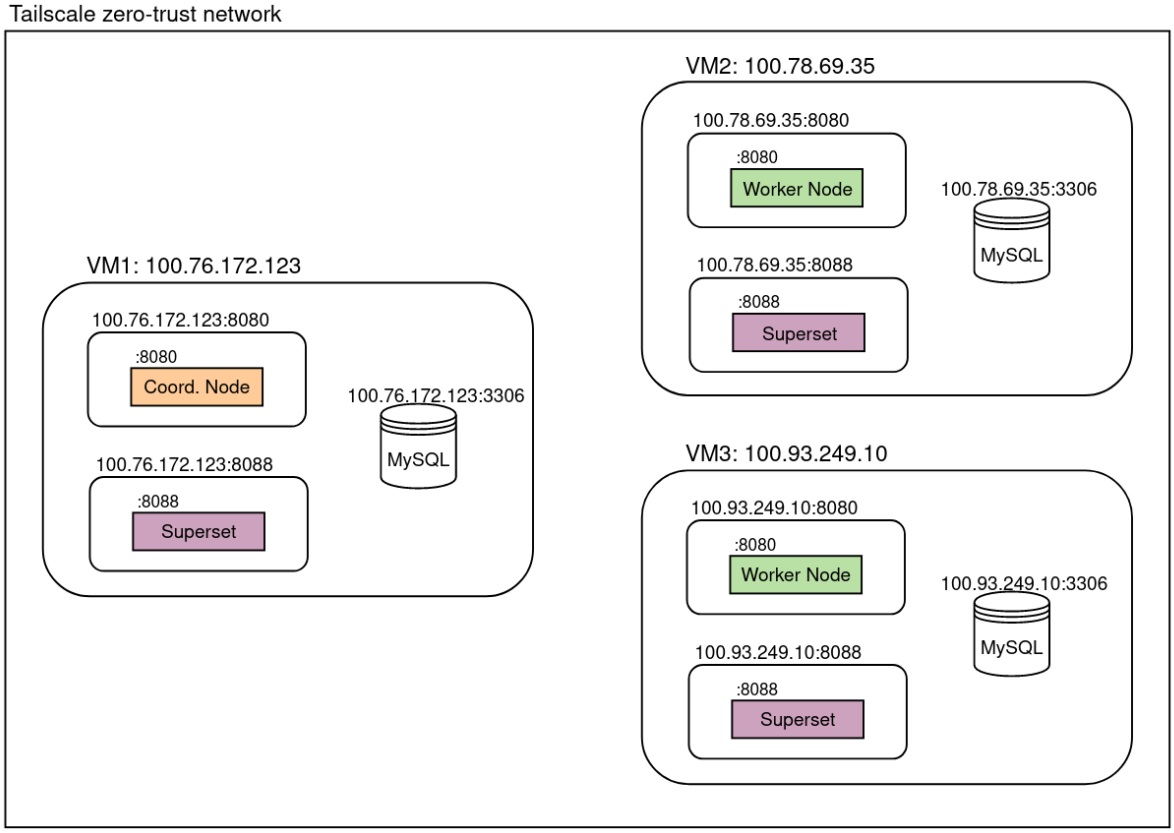

Figure 4.5: Data Mesh Architecture Deployment

4.11 Discussion on Performance and Availability

The performance of this architecture can primarily be defined as the time it takes to submit a query and receive the result. Trino plays a crucial role in handling this performance

aspect, as Superset adds no significant overhead except for creating visualizations. Once the data has been stored or cached in Superset, the retrieval process becomes instant, as there is no need to re-run the query.

Performance is a crucial metric for the overall usability of this architecture. To optimize performance, deploying the system as a cluster offers numerous benefits. By distributing the workload across multiple worker nodes, the work can be divided into partitions and parallelized, improving efficiency. Trino leverages this distributed nature to increase query performance. When creating the query plan, Trino distributes the query based on several criteria [59]:

- Minimizing data movement: Queries are distributed across worker nodes to minimize data movement. Nodes located close to the data source needed for the query are preferred [59].
- Maximizing parallelism: If parts of the query can be executed in parallel, these parts are distributed across multiple worker nodes [59].
- Considering network latency: Trino considers network latency, which refers to the time required for data to travel between worker nodes. The query optimizer aims to minimize the impact of network latency by assigning tasks to worker nodes that are geographically close or show low network latency between them [59].
- Node capabilities: When assigning query partitions, the resources of the worker nodes are considered. The query optimizer considers CPU cores, memory capacity and disk I/O capabilities to make informed decisions about worker node assignment [59].

Furthermore, Trino employs multiple threads on each worker node (based on the specific configuration) to fully utilize the available resources [59].

Part of the performance of a system is also its availability. In the implementation described in this chapter, we only deploy one Trino coordinator node. This creates a single point of failure for all queries across the data mesh network. If the coordinator node is unavailable, no data can be retrieved. One coordinator node is sufficient for this proof of concept implementation, as high availability is not a primary concern.

In section 5.3 we will propose concrete measures that can be taken to increase the performance, availability and security of your implementation.

Chapter 5

Evaluation

5.1 Deployment Specifications

Table 5.1 below provides an overview of the virtual machine specifications used to implement our data mesh architecture. These VMs were rented and operated with the Ubuntu 22.04 operating system, selected based on personal preference and familiarity with the Ubuntu environment. Each VM has 4GB of RAM and runs on an Intel Haswell processor with two physical cores operating at 2.394GHz. These specifications were the minimum requirements to support this proof of concept implementation. Further, the VMs are equipped with 85GB of storage, of which 15GB are used after all services for the DDoS data mesh have been installed. It is important to note that this also includes the DDoS fingerprints stored on each of the VMs. For our use case, the set of Fingerprints is rather small. However, larger fingerprint datasets can quickly require more storage capacity. To simulate decentralization and introduce additional latency challenges, we tried to distribute the VMs as much as possible globally. Given the decentralized nature of data mesh networks, this approach aimed to create a more realistic environment.

	Location	**OS**	**CPU**	**RAM**	**Storage**	**Tailscale IP**
VM1	London (UK)	Ubuntu 22.04.2 LTS	Intel (Haswell, no TSX) (2) @ 2.394GHz	4GB	85GB	100.76.172.123
VM2	Singapore (SGP)	Ubuntu 22.04.2 LTS	Intel (Haswell, no TSX) (2) @ 2.394GHz	4GB	85GB	100.78.69.35
VM3	Beauharnois (CA)	Ubuntu 22.04.2 LTS	Intel (Haswell, no TSX) (2) @ 2.394GHz	4GB	85GB	100.93.249.10

Table 5.1: Virtual Machine Specifications

5.2 Data Discovery

As outlined in the requirements for the overall architecture designed and implemented in this thesis, the architecture should also allow for data discovery capabilities. In this chapter, we assess the effectiveness of our architecture in facilitating data discovery with DDoS data. With the term data discovery, we specifically understand the ability to join, analyze and visualize the decentralized DDoS data stored at the domain teams.

To accomplish this, we conduct a series of concrete queries on the cluster, allowing us to examine the ability to retrieve relevant information. Subsequently, we showcase the process of visualizing the retrieved data.

5.2.1 Dataset

The DDoS fingerprints utilized for the queries have been generated with EDDD, a tool created as part of a master thesis at UZH [17]. DDoS fingerprints are represented in a JSON format. An overview of the fields contained in a fingerprint can be found in the form of an example fingerprint in the appendix (A.1) of this thesis. To store the fingerprints, we utilize the schema outlined in section 4.7. Therefore, the fingerprints are stored in the MySQL instances on every VM and can be retrieved via the configured MySQL user with remote access privileges. The base set of generated fingerprints consisted of a total of 596 DDoS fingerprints. To augment the base set, fingerprints have been duplicated at random to increase the total number of fingerprints to 2051. For the duplicated fingerprints, the `key` field has been newly generated. This means that in the augmented set of 2051 fingerprints, some fingerprints contain the exact same information, but have a different unique identifier key. In a real-world environment, repeated entries in the fingerprint dataset could be attributed to malicious actors or any of the participants trying to manipulate the data. This can provide an adversarial scenario for the evaluation of the data discovery capabilities of the architecture. The base set of generated fingerprints only contains attack vectors, including the TCP protocol. In order to make the data discovery in the following sections more interesting, we have changed the protocol on random fingerprints to ICMP. The remaining fields were left untouched. The augmented set of fingerprints has been distributed across the VMs at random. This resulted in the following distribution: 609 fingerprints on VM1, 693 fingerprints on VM2 and 749 fingerprints on VM3.

5.2.2 Data Retrieval

The first step of the data discovery process is to define what information we would like to retrieve from the DDoS attack data. Once we have a concrete idea of the information we want to analyze, we can think about what data has to be extracted that yields the information wanted. Based on that data, we can formulate queries that retrieve the data from the data mesh. For this showcase, we are considering the following information we would like to retrieve from the joint DDoS attack data:

- We want to find out which protocols are present in the attack data and how many packets were received via each protocol on each VM.
- We want to get an impression of the magnitude of the attack and the total duration that attack traffic data was recorded.
- We want to extract the maximum, minimum and average detection threshold values for all attack vectors for each VM.

The next step is to formulate the queries that allow for the retrieval of the necessary data. For this, we utilize the Superset SQL Lab. We can directly formulate and run the queries inside the SQL IDE. Below are the three queries that retrieve the data necessary for the analysis outlined above.

Query 1 aggregates data from three different tables (`vm1_ddos_data.AttackVector`, `vm2_ddos_data.AttackVector`, and `vm3_ddos_data.AttackVector`) that are located in three different databases (`vm1_data`, `vm2_data`, and `vm3_data`). The query sums up the `nr_packets` field for each `protocol` from each table. Additionally, the query also keeps track of the total number of packets per protocol (`packets_total`). The result is a list of protocols with the total number of packets for each VM and the total number of packets across all VMs.

```
SELECT
    protocol,
    SUM(packets_vm_1) AS packets_vm_1,
    SUM(packets_vm_2) AS packets_vm_2,
    SUM(packets_vm_3) AS packets_vm_3,
    SUM(packets_total) AS packets_total
FROM
  (
    SELECT
        protocol,
        nr_packets AS packets_vm_1,
        0 AS packets_vm_2,
        0 AS packets_vm_3,
        nr_packets AS packets_total
    FROM
        vm1_data.vm1_ddos_data.AttackVector
    UNION ALL
    SELECT
        protocol,
        0 AS packets_vm_1,
        nr_packets AS packets_vm_2,
        0 AS packets_vm_3,
        nr_packets AS packets_total
    FROM
        vm2_data.vm2_ddos_data.AttackVector
    UNION ALL
    SELECT
        protocol,
        0 AS packets_vm_1,
        0 AS packets_vm_2,
        nr_packets AS packets_vm_3,
        nr_packets AS packets_total
    FROM
        vm3_data.vm3_ddos_data.AttackVector
  )
GROUP BY
    protocol
```

Listing 5.1: Data Discovery Query 1: Packets per Protocol

Query 2 aims to retrieve data that allows us to get an overview of the duration and gravity of the attack. The query consists of multiple subqueries that join attack data from all

three data sources:

- The innermost subquery (`CombinedAttackVectors`) combines data from the `AttackVector` tables of all three VMs. The subquery selects the VM ID (`vm_id`), the number of packets (`nr_packets`) and the start time of the attack (`time_start`).
- The next subquery (`BaseQuery`) is aggregating this combined data. The subquery counts the number of vectors from each VM and the total number of packets, grouped by the start time.
- The outermost query is then joining this base query with three other subqueries (`vm1_duplicates`, `vm2_duplicates`, `vm3_duplicates`), each of which calculates the duplicate rate for each VM. The duplicate rate is calculated as the total count of packets divided by the distinct count of packets, multiplied by 100. This results in the percentage of total attack vectors that are duplicates per VM per second. If there are no duplicates for a VM, the rate is set to 0 using the `COALESCE` function. In order to gain insight into how unique the recorded attack vectors are, you need to specify what makes them unique. In our case, we chose that if two attack vectors have the exact same amounts of packets, they are treated as duplicates. Since we are working with a generated dataset, we allow for this abstraction. When dealing with a complex dataset, it could be advantageous to more carefully define when two attack vectors are considered to be duplicates.
- The final result of the query is a table with the following columns: `time_start`, `vm1_vector_count`, `vm2_vector_count`, `vm3_vector_count`, `total_packets`, `vm1_duplicate_rate`, `vm2_duplicate_rate`, `vm3_duplicate_rate`. Each row in the table represents a unique start time and for each start time, it shows the count of vectors from each VM, the total number of packets and the duplicate rate for each VM.

```
SELECT
    BaseQuery.time_start,
    BaseQuery.vm1_vector_count,
    BaseQuery.vm2_vector_count,
    BaseQuery.vm3_vector_count,
    BaseQuery.total_packets,
    COALESCE(vm1_duplicates.duplicate_rate, 0) AS vm1_duplicate_rate,
    COALESCE(vm2_duplicates.duplicate_rate, 0) AS vm2_duplicate_rate,
    COALESCE(vm3_duplicates.duplicate_rate, 0) AS vm3_duplicate_rate
FROM (
    SELECT
        time_start,
        SUM(CASE WHEN vm_id = 'vm1' THEN 1 ELSE 0 END) AS
            vm1_vector_count,
        SUM(CASE WHEN vm_id = 'vm2' THEN 1 ELSE 0 END) AS
            vm2_vector_count,
        SUM(CASE WHEN vm_id = 'vm3' THEN 1 ELSE 0 END) AS
            vm3_vector_count,
        SUM(nr_packets) as total_packets
    FROM (
        SELECT
            'vm1' as vm_id,
```

```
            nr_packets,
            time_start
        FROM
            vm1_data.vm1_ddos_data.attackvector
        UNION ALL
        SELECT
            'vm2' as vm_id,
            nr_packets,
            time_start
        FROM
            vm2_data.vm2_ddos_data.attackvector
        UNION ALL
        SELECT
            'vm3' as vm_id,
            nr_packets,
            time_start
        FROM
            vm3_data.vm3_ddos_data.attackvector
    ) AS CombinedAttackVectors
    GROUP BY
        time_start
) AS BaseQuery
LEFT JOIN (
    SELECT
        time_start,
        COUNT(*) * 100 / COUNT(DISTINCT nr_packets) as duplicate_rate
    FROM
        vm1_data.vm1_ddos_data.attackvector
    GROUP BY
        time_start
) AS vm1_duplicates ON BaseQuery.time_start = vm1_duplicates.time_start
LEFT JOIN (
    SELECT
        time_start,
        COUNT(*) * 100 / COUNT(DISTINCT nr_packets) as duplicate_rate
    FROM
        vm2_data.vm2_ddos_data.attackvector
    GROUP BY
        time_start
) AS vm2_duplicates ON BaseQuery.time_start = vm2_duplicates.time_start
LEFT JOIN (
    SELECT
        time_start,
        COUNT(*) * 100 / COUNT(DISTINCT nr_packets) as duplicate_rate
    FROM
        vm3_data.vm3_ddos_data.attackvector
    GROUP BY
        time_start
) AS vm3_duplicates ON BaseQuery.time_start = vm3_duplicates.time_start
ORDER BY
    BaseQuery.time_start
```

Listing 5.2: Data Discovery Query 2: Attack Overview

Query 3 retrieves the average, maximum and minimum detection thresholds for the set of fingerprints on each of the three virtual machines. It does this by querying three differ-

ent tables (`vm1_data.vm1_ddos_data.attackvector`, `vm2_data.vm2_ddos_data.attackvector` and `vm3_data.vm3_ddos_data.attackvector`), each representing a different VM. The result of the query is a list of VMs with the maximum, minimum and average detection threshold values.

```
SELECT
    'VM1' as VM,
    CAST(AVG(detection_threshold) as DECIMAL(10,2)) as
       Average_Detection_Threshold,
    CAST(MAX(detection_threshold) as DECIMAL(10,2)) as
       Max_Detection_Threshold,
    CAST(MIN(detection_threshold) as DECIMAL(10,2)) as
       Min_Detection_Threshold
FROM
    vm1_data.vm1_ddos_data.attackvector
UNION ALL
SELECT
    'VM2',
    CAST(AVG(detection_threshold) as DECIMAL(10,2)),
    CAST(MAX(detection_threshold) as DECIMAL(10,2)),
    CAST(MIN(detection_threshold) as DECIMAL(10,2))
FROM
    vm2_data.vm2_ddos_data.attackvector
UNION ALL
SELECT
    'VM3',
    CAST(AVG(detection_threshold) as DECIMAL(10,2)),
    CAST(MAX(detection_threshold) as DECIMAL(10,2)),
    CAST(MIN(detection_threshold) as DECIMAL(10,2))
FROM
    vm3_data.vm3_ddos_data.attackvector;
```

Listing 5.3: Data Discovery Query 3: Detection Thresholds

When we enter and run the queries in the Superset SQL Lab, the result is displayed in a table below the SQL IDE. From there, the result data, as well as the query, can be saved for future use. A Screenshot of the interface of Superset displaying a Query and the structure of the resulting data can be found in the appendix of this thesis (A.4).

5.2.3 Data Visualization

Once we have the data from the queries in Superset, we can create visualizations based on that data directly inside Superset. Superset offers the ability to create interactive visualizations based on the data queried. For each of the queries above, we have created a separate visualization.

For *Query 1*, we have created a grouped bar chart 5.1. Each Protocol retrieved from the combined set of fingerprints is one group. Inside a group, we display the number of packets monitored per VM as individual bars. With the augmented set of fingerprints, two protocols are present: TCP and ICMP. The total number of packets received via TCP

amounts to 2.18 billion, while the total number of packets received via ICMP amounts to 275 million. The visualization allows you to see that the total amounts of packets received per protocol is not dominated by a single VM. Additionally, the majority of the packets were received via the TCP protocol.

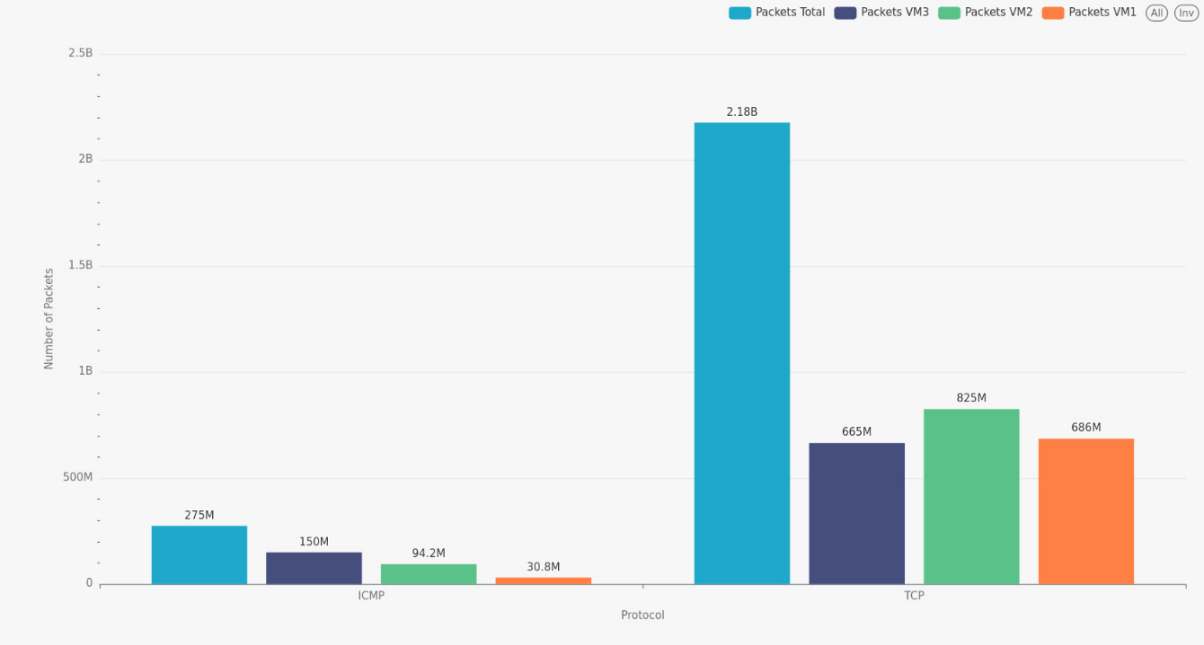

Figure 5.1: Number of Packets per Protocol per Source

For *Query 2*, we chose to visualize the data retrieved as a mixed chart. Figure 5.2 depicts a close-up of the created visualization. The complete visualization can be found in the appendix of the thesis A.5. On the x-axis of the chart is the duration of the attack in seconds. Each attack vector present in the dataset has a timestamp. An attack vector refers to a specific type of DDoS attack, including details like the service protocol used, source IPs involved and their corresponding real IPs if any. In the dataset we use, the attack vectors are either ICMP- or TCP-based, involving multiple source IPs. For each second of the attack, we present specific data. The total count of attack vectors for each virtual machine is visualized as a stacked bar. The percentage of duplicate attack vectors for each virtual machine is depicted as a line. Hovering over one of the visual features allows you to see the specific values as a tooltip menu. From the visualization, we can see that each of the VMs recorded a similar amount of attack vectors throughout the attack. We can also see that each VM has quite a high attack vector duplication rate. For example, a duplication rate of 315% for VM1 in second two of the attack means that each distinct attack vector appears 3.15 times in the attack vector set of that second. In other words, for every unique attack vector, there are approximately two additional duplicates in the data of that time frame. This makes sense, given that we have artificially augmented the dataset from 596 to 2051 fingerprints.

For *Query 3*, we chose to visualize the results as a grouped bar chart 5.3. The average, maximal and minimal detection thresholds retrieved from each VM are represented as individual bars. The three resulting bars are then grouped per VM from where they have been retrieved. This visualization lets the user get a quick overview of how the

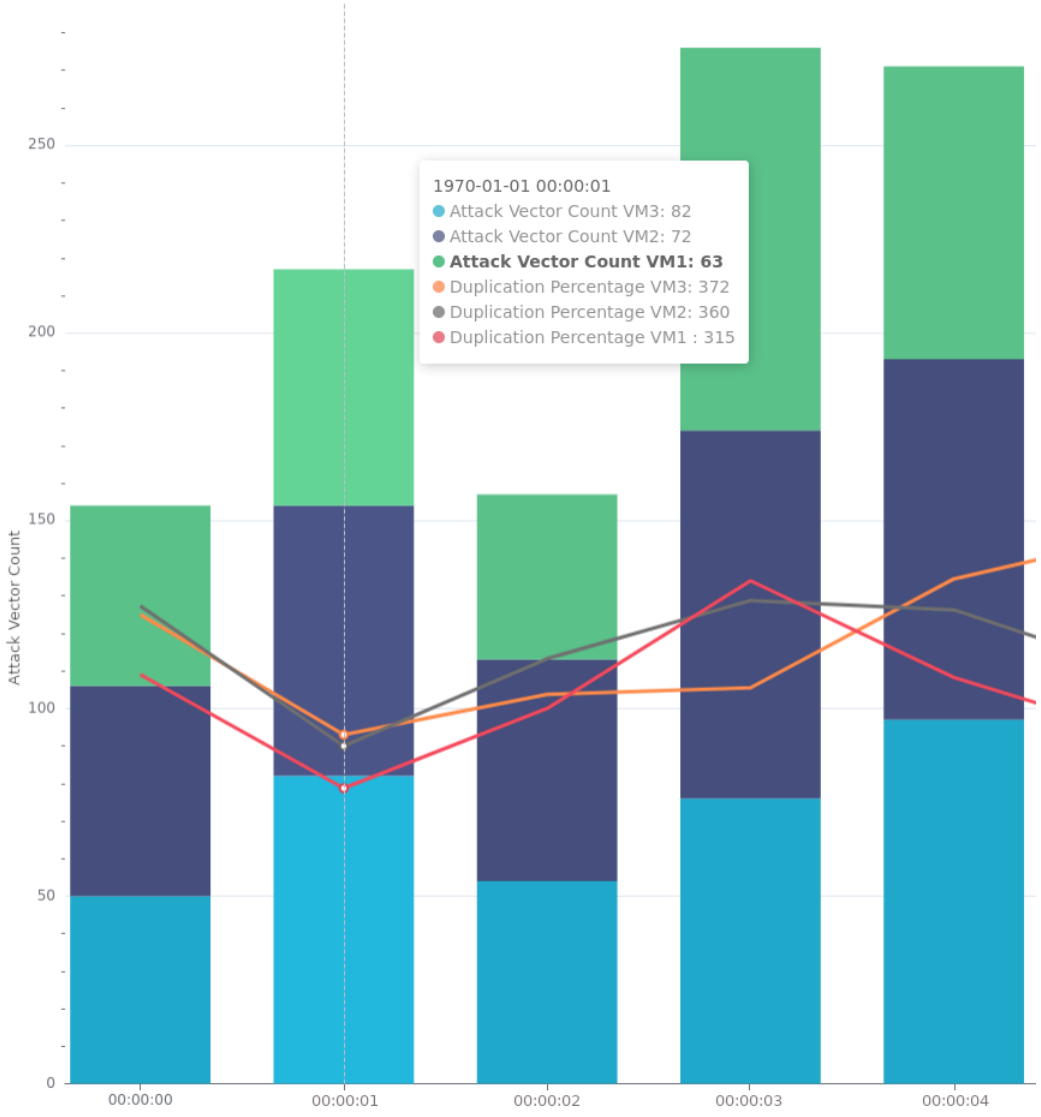

Figure 5.2: Close up: Attack Vector Counts and Duplication Percentages per VM over the Attack Duration

detection thresholds compare between the domain teams. Unusually high or low detection thresholds can be seen at a quick glance.

5.2.4 Query Performance

In this section, we evaluate the query performance for the queries specified earlier. To gain insights into the overall performance of the architecture, we measured several metrics of the queries. The dataset outlined in section 5.2.1 consists of 2051 fingerprints and collectively amounts to a total data size of 8.924 megabytes. The distribution of these fingerprints, as detailed in Section 5.2.1, resulted in the following data distribution: 2.612 megabytes on VM1, 3.064 megabytes on VM2 and 3.248 megabytes on VM3. Table 5.2 shows the evaluation results. All data has been read from the Trino cluster dashboard.

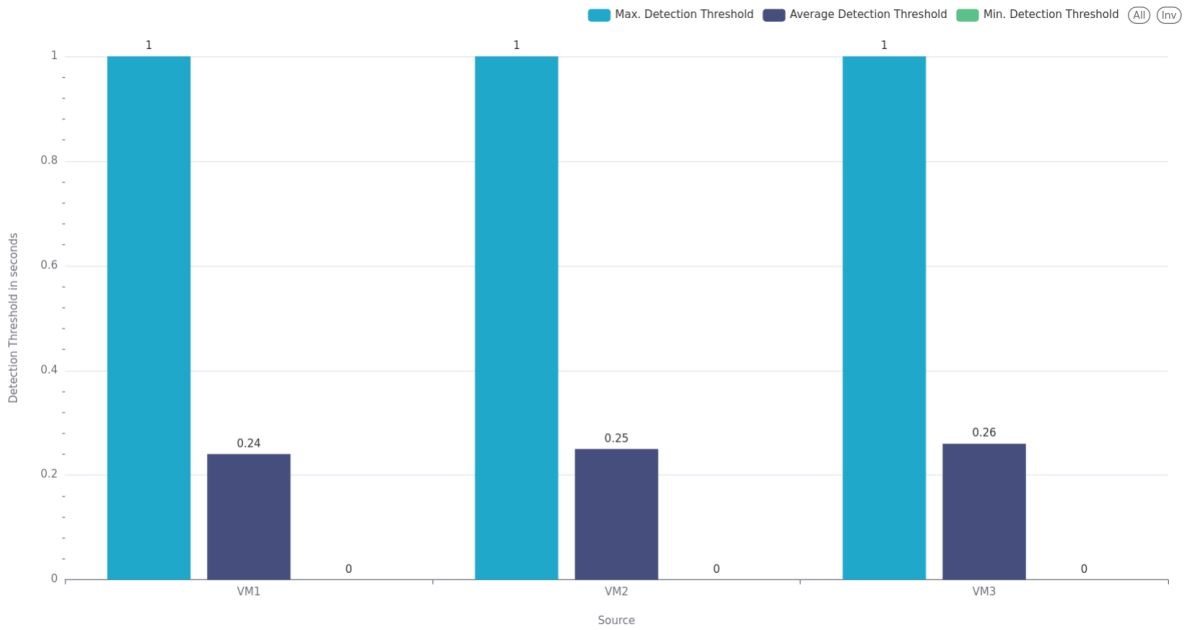

Figure 5.3: Detection Thresholds by Source

Each query was run five times to ensure accuracy and reliability. The time measurements were then averaged over these five runs. The input and output row measurements remained consistent across all the runs. For all three queries, we measured the following metrics:

- **Elapsed Time**: Total time of the query, from when it was submitted to when the final results were returned.
- **Queued Time**: The time the query spent waiting in a queue before it started processing. This time depends on the availability of worker nodes and the overall load the coordinator node faces.
- **Analysis Time**: The time required for analyzing the query, including tasks such as parsing the SQL query and doing semantic analysis.
- **Planning Time**: The time taken to formulate an execution plan for the query.
- **Execution Time**: The time taken to execute the query based on the formulated execution plan. This includes retrieving data from decentralized data repositories, processing it and aggregating the results.
- **No. Input Rows**: Represents the number of rows processed for the aggregation operations within the query. It is not the total number of rows read from the data sources. For example, in *Query 3*, the aggregation operations involve grouping the attack vectors by VM and calculating the average, minimum and maximum detection thresholds for each VM. The number of input rows counts the number of rows involved in this grouping and aggregation process, corresponding to the number of unique VMs in the data mesh.

- **No. Output Rows**: Represents the number of rows that are effectively returned from the query.

It is important to note that the elapsed time can be shorter than the sum of the other times. This is because some of these stages can happen concurrently. For instance, while the query is being analyzed (Analysis Time), the system might already start planning (Planning Time) for the parts of the query that have already been analyzed. Similarly, some parts of the query might start executing (Execution Time) while others are still being planned [53]. Additionally, we took the averages over five runs which may also lead to the times not adding up to the total elapsed time. The average of sums is not necessarily equal to the sum of averages.

	Elapsed Time:	**Queued Time**	**Analysis Time**	**Planning Time**	**Execution Time**	**No. Input Rows**	**No. Output Rows**
Query 1	5.11s	0.00676s	3.17s	1.20s	1.94s	2'050	2
Query 2	6.82s	0.00127s	3.13s	1.33s	3.68s	4'100	11
Query 3	4.97s	0.00163s	3.16s	1.14s	1.81s	3	3

Table 5.2: Query Performance Evaluation

The above table offers several interesting insights into the overall performance of the architecture:

- The total execution times for all queries were remarkably short, considering that this decentralized architecture fetches data from three different data sources within a single query.
- The short execution times can largely be attributed to the relatively small size of the dataset. Despite the complexity of the queries, the small data size allows for fast processing. *Query 2*, the most complex query with the highest execution time, also shows the largest number of input rows.
- The analysis and planning times are consistent across all queries. These times are directly influenced by the layout and architecture of the data mesh.
- The execution time consistently remained lower than the combined analysis and planning times. This indicates that the overhead associated with querying different data sources and the planning costs inherent to Trino running as a cluster may surpass the execution time itself. This suggests potential room for optimization, especially for small queries where the cluster handling overhead might not be justifiable.

In conclusion, the evaluation highlights the tradeoffs involved in utilizing a decentralized data mesh architecture with a Trino cluster for query processing. While the overall execution times are short, there exists a balance between the benefits of cluster performance and the overhead incurred. This may be more favorable for larger, more complex datasets and clusters. As with many things in system design this is a tradeoff. You trade a substantial overhead for small datasets with the potential performance benefits that a cluster offers for larger datasets.

5.2.5 Data Discovery Discussion

The data discovery and visualization sections demonstrate the data discovery capabilities of this architecture. *Query 2* shows that we can specify complex Queries that retrieve multidimensional datasets. With Superset, we are then able to visualize these complex datasets. Superset also offers powerful interactive features that can help with the data discovery. However, we have also noticed that we started to reach the limits of Superset with the visualization for *Query 2*. It seemed that Superset does not always offer the degree of flexibility one might need when working with complex datasets. While multidimensional visualizations are possible, we got the impression that this is not what the tool was built for. However, this is also where the modularity of the architecture comes into play. If a user thinks that the visualization possibilities of Superset are not enough, the tool can simply be used as a SQL IDE to run queries against the data mesh. Superset allows you to get a quick overview over the results of the queries. Once the user is satisfied with the resulting data, the data can be exported as a .csv or a .json file. This then allows you to use whatever visualization tool or library that fits the needs of the user. It is important to note that the queries and visualizations presented here demonstrate the data discovery possibilities within the architecture. More complex visualizations and interactive features can be developed with a more complex dataset to extract even deeper insights from the data.

5.3 Analysis

In this section, we aim to provide an in-depth analysis of the general performance, security and deployment aspects of the DDoS data mesh service. Many of these aspects are configuration- and deployment-specific. By discussing them, we aim to address aspects that are not necessarily important in our use case, but are important to consider if the design is replicated with a different use case at hand.

5.3.1 Performance Analysis

In this part of the thesis, we will discuss the performance of the architecture based on the implementation and deployment outlined in the previous chapters. Our goal is to provide an overview of configuration-based performance factors that may influence the overall performance of the architecture.

5.3.1.1 Maximum Query Size

The evaluation of query times in section 5.2.4 shows that the overall performance for data retrieval is reasonably good. Since DDoS fingerprints are already a condensed representation of network traffic data, these datasets should always be reasonably sized. Nonetheless, it is important to test the limits of the architecture to rule out potential shortcomings.

Trino supports a connector for the TPC-H benchmark dataset. The connector offers a collection of schemas specifically designed to facilitate the TPC-H benchmark. When querying a TPC-H schema, the connector dynamically generates data in real-time through a deterministic algorithm [58]. TPC-H serves as a benchmark for decision support. The selection of queries and database content aims for wide applicability across industries. This benchmark showcases the capabilities of decision support systems that handle substantial data volumes, execute intricate queries and provide solutions to vital business inquiries [48]. An overview of the tables contained in the TPC-H benchmark dataset can be found in the appendix of this thesis A.6

The TPC-H connector allows us to test the limits of our architecture without having to generate extremely large datasets ourselves. Instead of measuring the query times and comparing them with other architectures, we can use the dataset to find out at which point the architecture struggles. Each TPC-H schema consists of a standardized set of tables. The size of the schemas can be scaled arbitrarily by including the scaling factor in the schema name. For example, the default TPC-H schema can be queried using `tpch.sf1`. The same dataset scaled by a factor of two can be queried using `tpch.sf2`. For the evaluation, we used the below query. The query has been taken from a pre-defined collection of sample queries [16]. It joins the three tables `customer`, `orders` and `lineitem` and retrieves the shipping priority and potential revenue. This query has been selected, as it is similar in complexity to queries that might be performed with DDoS data. Further, the query has been adapted to fit the syntax of Trino.

```
SELECT
    l.orderkey,
    sum(l.extendedprice * (1 - l.discount)) as revenue,
    o.orderdate,
    o.shippriority
FROM
    tpch.sf2.customer c
JOIN
    tpch.sf2.orders o ON c.custkey = o.custkey
JOIN
    tpch.sf2.lineitem l ON l.orderkey = o.orderkey
WHERE
    c.mktsegment = 'BUILDING'
    AND o.orderdate < date '1995-03-15'
    AND l.shipdate > date '1995-03-15'
GROUP BY
    l.orderkey,
    o.orderdate,
    o.shippriority
ORDER BY
    revenue desc,
    o.orderdate;
```

Listing 5.4: TPC-H Example Query

Table 5.3 depicts the time taken to execute the query and the total number of input rows for each schema size queried. The queries were run 5 times and the time measures were

averaged across the five runs. The time and the number of input rows were taken from the Trino cluster dashboard.

	No. Input Rows	**Time Executed**
tpch.sf1	7'710'000	8.48s
tpch.sf2	15'600'000	17.84s
tpch.sf3	22'900'000	N/A

Table 5.3: Performance Comparison of TPC-H Schemas in Terms of Number of Input Rows and Execution Time

Based on the performance measurements of the TPC-H schemas, we observed that queries for *tpch.sf1* and *tpch.sf2* executed within a reasonable time, given the input size. However, when executing the query for *tpch.sf3*, Trino encountered an error due to insufficient memory allocation on the worker nodes. The current configuration limits the worker nodes to allocate a maximum of 614.40 MB of memory for query execution. This is insufficient for the query using *tpch.sf3*. Therefore, our proposed implementation can comfortably handle an input size of up to 15.6 million rows. It is important to consider that memory allocation directly affects the capability to handle larger queries. Adjusting the `-xmx` field in the *jvm.config* file of every Trino instance, allows you to directly adjust how much memory Trino uses. If your hardware allows for more memory to be used, you can increase this value. This allows the cluster to handle larger queries. As a reference, Trino recommends this value to be between 70 and 85 percent of the total available memory. In addition, Trino recommends memory allocations beyond 32GB for large production clusters [55].

5.3.1.2 Increasing Availability

As mentioned in the discussion on performance in Chapter 4.11, a single coordinator node introduces a single point of failure for the whole architecture. Nonetheless, there are approaches to avoid this single point of failure. Multiple coordinator nodes can be deployed inside the same cluster. Coordinator nodes do not interfere with the queries of other coordinator nodes. At any point, they can allocate available worker nodes for their query. Once a worker node is assigned a query, it becomes unavailable for other queries. This allows you to deploy multiple coordinator nodes to mitigate the single point of failure. However, it is important to note that you must keep track of which domain team accesses which coordinator. In our implementation, the URL of the coordinator node is configured in the Superset instance. If you have multiple coordinator nodes, you have multiple URLs that you can configure (one per coordinator node). In Superset, you can configure multiple database connections, allowing you to configure multiple coordinator nodes. For each SQL Lab query, one can select which database connection you want to use. If a query fails due to an unavailable coordinator node, you can switch to another database connection and re-run the query.

An alternative approach would be to deploy a load balancer in front of the coordinator nodes. The load balancer can then be configured to dynamically distribute incoming

queries across the coordinator nodes based on availability. This would minimize the possibility of receiving a query error due to the unavailability of a coordinator node.

As is often the case when designing system architecture, performance is a tradeoff accompanied by other considerations such as deployment complexity, security, or high availability. The design taken in this thesis offers the flexibility to address these tradeoffs and adapt to specific performance requirements. Increasing the number of worker or coordinator nodes increases the performance of the cluster at the cost of deployment complexity and security. Placing the cluster behind a load balancer increases the availability at the cost of deployment complexity.

5.3.1.3 Performance and Resource Management Considerations

In the decentralized and multi-party environment of our data mesh architecture, careful consideration of resource management and accountability is crucial. As discussed in the section on Trino concepts (4.1), worker nodes are allocated to queries based on their availability. If there are not enough workers available, a query is held in a queue until enough worker nodes are available. Worker nodes can therefore be considered as a shared resource across the data mesh. This opens up the mesh for potential denial of service, if this shared resource is intentionally, or unintentionally, wasted. For example, if some domain teams submit inefficient queries that occupy worker nodes longer than necessary, they block the allocation for other incoming queries. An example of such an inefficient query would be cross joining the largest tables from all data sources as depicted below. If each `attackvector` table had one million rows, the result of this query would have a staggering quintillion (10^{18}) rows. We did not manage to come up with an infinite query, as recursions quickly hit recursion depth-limits and for-loops are not supported by Trino. Nonetheless, such resource-intensive queries can still cause significant strain on the cluster.

```
SELECT a.*, b.*, c.*
FROM vm1_data.vm1_ddos_data.attackvector AS a
CROSS JOIN vm2_data.vm2_ddos_data.attackvector AS b
CROSS JOIN vm3_data.vm3_ddos_data.attackvector AS c
```

Listing 5.5: Example of an inefficient SQL Query

To address these concerns, the Trino dashboard becomes an essential tool for tracing accountability. It records queued, running and finished queries alongside their performance metrics, offering transparency into the cluster usage. This information may be leveraged in a business setting to enforce payment based on query runtimes. This, in return, discourages the (un)intentional wasting of shared resources. From a technical perspective, it is also possible to configure the maximum total runtime of queries on the cluster. However, Trino is designed to run longer queries, so overly restrictive settings might not be the most suitable solution, as wasteful queries can be restarted once they time out. Therefore, balancing performance and resource accountability becomes a crucial consideration for optimizing the effectiveness and efficiency of the architecture.

5.3.2 Security Analysis

As mentioned earlier, there is always a tradeoff in system design. With decentralization, security concerns increase due to the larger number of components within the system. The deployment in this thesis maximizes openness, allowing all domain teams to access all data sources. Consequently, all database access credentials must be configured in the catalog configurations of all Trino instances. While this approach maximizes decentralization and performance, it also increases the overall vulnerability of the system. In our proof of concept implementation, we mitigated this tradeoff by implementing a zero-trust network to restrict unwanted external access. Generally, with the design proposed in this thesis, the following trade-off about security can be stated:

- The higher the degree of decentralization, the higher the number of components in the architecture and the higher the overall vulnerability of the architecture.
- The lower the degree of decentralization, the lower the number of components in the architecture, and the lower the overall vulnerability of the architecture.

By default, Trino runs with no security measures. However, you can improve security by forcing communication via HTTPS. Trino coordinator and worker nodes can be configured to use TLS certificates. This adds a layer of security to cluster-internal communication. If you expand the implementation to use a load balancer in front of the cluster, as explained previously, you may also terminate TLS at the load balancer. This introduces a layer of security between clients and the load balancer and removes the need to add complexity to the configurations of the Trino instances. For this proof of concept deployment, we decided against the implementation of TLS. This is mainly because we aim to keep the configuration of all services as small and replicable as possible. Since this is not a production deployment, security is not a priority of the architecture. Nonetheless, it is important to address how security shortcomings might be mitigated.

Alternatively, access to the Trino cluster can be made more restrictive. Trino accounts can be created for each domain team that only grant access to the data sources that the respecting domain team requires. The username of the account can then be passed as an argument in the Trino connection URL used to query the cluster.

A further approach to enhancing security and limiting access is to divide the cluster into subparts. By configuring only a share of all catalogs (data sources) on worker nodes, you can restrict access to specific data sources. For example, dividing the cluster by geographical region ensures that worker nodes in each region only have access to the data sources from that same region. This approach eliminates the need to configure the access credentials of all data sources on all Trino instances. Trino supports this capability as the coordinator knows which worker node can access which data source. The query optimizer then creates a plan based on data access and geographical location (see 4.11).

It is important to note that trust plays a crucial role among the participants of the collaborative DDoS defense. The nature of a decentralized system involves granting a significant degree of autonomy to the involved parties. While certain aspects like tracing accountability, resource allocation or data privacy and security can be controlled with the precautions

mentioned above, the topic of trust extends beyond these measures. Specifically data integrity is a critical consideration. There is a risk that some domain teams may not store accurate or appropriate data, whether due to human error, deliberate manipulation or technical storage issues. Additionally, non-compliance with agreed-upon security protocols or best practices by some domain teams could expose vulnerabilities in the entire architecture.

These are inherent concerns that come with decentralized systems and must be carefully addressed based on the level of trust established among the participants. Trust is a fundamental aspect that underpins the effectiveness and reliability of the architecture proposed in this thesis.

5.3.3 Deployment Considerations

For this proof of concept implementation, we chose to deploy Trino and Superset in a containerized environment. While we have already touched on the motivations behind this choice, below we list further deployment considerations that highlight some of the deployment possibilities of the architecture and the effects that might come with them:

- The architecture relies entirely on open-source tools. This reduces costs but may lead to potential maintenance challenges, as the maintenance for the tools is community-driven.
- Not every domain team needs to run their own Trino instance. While each domain team requires to run its own data storage solution, deploying a Trino cluster with coordinator and worker nodes should be carefully evaluated. The number of worker nodes directly impacts deployment complexity and query overhead. Assessing the appropriate number of worker nodes is crucial for achieving the desired degree of decentralization while optimizing performance.
- The number of components, particularly Trino instances, should be determined based on the performance requirements. This decision is a tradeoff specific to the use case of the architecture.
- Similarly, not every domain team needs to run a Superset instance (see 4.4). Instances can be shared across domain teams.
- Any of the services within the architecture may also be installed from scratch. The configurations proposed in chapter 4 can also be used if the services are installed from scratch.
- The entire architecture can be deployed on a Kubernetes cluster. This may offer increased security, performance and availability. However, this approach also requires a significant initial investment to set the cluster up and configure network policies.
- It is important to consider that this collaborative DDoS defense architecture using a data mesh is most effective when implemented in a domain-driven environment with independent domain teams. Otherwise, alternative centralized data architectures or other solutions might be more suitable.

Chapter 6

Final Considerations

In the subsequent sections, we draw conclusions from our achievements, evaluate the fulfillment of our goals and outline potential directions for future research and improvement.

6.1 Summary

This thesis offers insights into the research of collaborative DDoS defense, focusing on data-centric and decentralized approaches. The initial phase involved creating an overview on DDoS attack and defense mechanisms, data mesh networks and collaborative defenses. This knowledge formed the foundation for the subsequent tasks. During the research on related literature, we focused on existing tools that could be used to design and implement our DDoS data mesh. However, we discovered a lack of suitable solutions for our specific use case. This challenge led us to build a custom stack tailored to our needs. With the insights on existing tools, we designed an architecture according to what may be possible with the tools evaluated in the related works part. Trino quickly emerged as the central component of the architecture, allowing for the decentralized querying of DDoS attack data. Using Trino as a core component, we iteratively assembled the rest of the stack, creating a fully functional DDoS data mesh that met our requirements. During the design of the architecture, we were able to test the functionality of single components of the design. However, it was in the implementation part where we got to see how these components interact with each other for the first time. This step would decide whether adjustments to the design were necessary to continue with the evaluation of the architecture. In the evaluation part, we aimed to demonstrate the functionality and performance of the architecture. The data discovery phase involved exploring various query iterations, ultimately selecting three representative queries to showcase the capabilities of the architecture.

Throughout the thesis, we focused on the expandability and modularity of the architecture. This allowed us to create a solution according to the specified requirements, which can also be adapted to suit different requirements.

6.2 Conclusions

This thesis focused on a specific use case where parts of a collaborative DDoS defense architecture already existed. The main goals of this thesis were to:

- Give an overview of DDoS attack basics and associated related work. Produce a basis for the comparison of a DDoS data mesh approach, listing its associated advantages and drawbacks.
- Design and implement a data mesh architecture, including a discovery service to find information about the nature and characteristics of DDoS attacks.

We have successfully met these goals. Through the background and related works section, we explored key topics and tools relevant to designing and implementing DDoS data mesh architectures. The design and implementation outlined in this thesis offer a viable solution for our goals. With the ability to query and join DDoS fingerprints stored in decentralized repositories over our data mesh network, we provide the baseline functionality that was required. With Trino, we provide a standardized way to retrieve data from multiple decentralized data sources in a single statement. The integration of Superset on top of the data mesh provides data discovery capabilities, allowing seamless access to the data mesh within the discovery service.

Over the course of the thesis, we encountered the following challenges and difficulties:

- Trade-off Situations: We encountered various trade-off situations while designing and implementing the final solution. These trade-offs involved the performance and the overall complexity of the architecture. In some cases, we knew where to position ourselves in the trade-off based on the requirements and overall prototype-setting of the thesis. However, oftentimes there was no right or wrong regarding these situations. The only approach then was to weigh the implications carefully, and then reason about our choice.
- Position in Existing Collaborative DDoS Defense: Another difficulty was the position of this work in an already existing collaborative DDoS defense setting. This meant we had to thoroughly consider which aspects of the architecture may be assumed and where the relevance of the work of this thesis lies.
- Edge-Cases and What-if Scenarios: A typical difficulty when designing system architecture is to think about edge cases and ”what-if” scenarios. When designing or implementing system architecture, it is easy to be satisfied once the main functionality is given. However, a thorough examination of the scenarios where certain parts may not be available, potential vulnerabilities, or performance implications required extensive consideration.

Throughout the design and implementation of the architecture, performance and usability were central considerations. Alongside achieving the main goals of the thesis, we continuously considered ways to enhance performance and usability. This focus significantly

influenced the overall outcome, as the provided solution achieves the set goals while being performant and usable within the testing scope. During the process of designing, implementing and evaluating the architecture, we encountered several key takeaways that shape how we approach similar projects in the future:

- Data Mesh Potential: Exploring data mesh architectures as decentralized alternatives for collaborative DDoS defenses allowed us to recognize the potential of data-centric solutions. Such solutions can be effective in sharing DDoS-related information. This realization opens up new possibilities for future research in data mesh networks and their applications in cybersecurity and collaborative defense strategies.
- Thorough Evaluations and Trade-offs: We undertook a comprehensive evaluation, meticulously assessing different trade-offs, evaluating performance metrics, and considering the implications of various architectural choices. These evaluations provided important insights and empowered us to refine the system, ensuring its adaptability and robustness.
- Challenges of Heterogeneous Environments: Working on the thesis highlighted the complexities and challenges of designing a service within a highly heterogeneous environment. For instance, in line with the heterogeneity of the environment, we envisioned the domain teams to have a large degree of autonomy in how they store their DDoS attack data. However, this comes with a high price in architectural complexity. Addressing this challenge required careful consideration in selecting the right tools and technologies to streamline the architecture effectively.

We carefully followed the proposed schedule at the beginning of the thesis. Regular interactions with my supervisor and staying on schedule were essential in maintaining focus on relevant topics and keeping the project on track.

6.3 Future Work

One potential area for future work is to use larger and more realistic datasets with the architecture proposed in this thesis. This will enable a more comprehensive evaluation of data discovery and performance of the architecture. The evaluation conducted in this thesis was limited to the use case presented by the generated DDoS Fingerprint dataset. This may not accurately represent real-world scenarios. By incorporating more data and diverse use cases, a more thorough evaluation of the performance and data discovery capabilities can be achieved.

The work done in line with this thesis is a first step into the research of collaborative DDoS defense architectures utilizing data mesh networks. In order to motivate future research, we have created a complementary GitHub repository. The repository contains all necessary files and instructions for replicating the architecture proposed in this thesis.

This effort reflects our commitment to initiating further progress in the field of collaborative DDoS defense architectures. The repository can be accessed under the following URL: `https://github.com/tportmann-uzh/ddos-data-mesh-network`

There is a noticeable lack of research on Data Mesh networks specifically used in collaborative DDoS defense architectures. Particularly, research is scarce addressing the possibilities of generating DDoS defense information in decentralized systems. This area of research may not only focus on how relevant data can be retrieved from the parties of the collaborative defense architecture. It can also explore how that data can be structured or which analyses offer the most useful insights into DDoS attacks. Moreover, the related works chapter revealed a scarcity of single-tool solutions or suitable tools for quick and straightforward deployments of data mesh networks. Although this work shed light on this research area, it goes beyond the scope of just DDoS defense. Data mesh networks represent a novel approach that counters large centralized data architectures. Further research could explore ways to provide the necessary knowledge and tools for implementing such networks. Establishing this foundation with existing research would significantly lower the barrier to entry for decentralized architectures, making data mesh networks more prevalent and suitable for various applications.

Bibliography

[1] Neha Agrawal and Shashikala 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 (2019), pp. 3769–3795.

[2] Muhammad Ejaz Ahmed, Saeed Ullah, and Hyoungshick Kim. "Statistical application fingerprinting for DDoS attack mitigation". In: *IEEE Transactions on Information Forensics and Security* 14.6 (2018), pp. 1471–1484.

[3] Aditya Akella et al. "Detecting DDoS attacks on ISP networks". In: *Proceedings of the Workshop on Management and Processing of Data Streams*. 2003, pp. 1–2.

[4] Amazon Amazon. *Amazon Athena*. 2023. URL: `https://docs.aws.amazon.com/athena/latest/ug/what-is.html` (visited on June 20, 2023).

[5] Amazon Amazon. *Amazon S3*. 2023. URL: `https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html` (visited on June 20, 2023).

[6] Apache Hadoop Apache Hadoop. *Apache Hadoop*. 2023. URL: `https://hadoop.apache.org/` (visited on June 20, 2023).

[7] Apache Superset Apache Superset. *Apache Superset*. 2023. URL: `https://superset.apache.org/docs/intro` (visited on June 24, 2023).

[8] Mark W Beall and Mark S Shephard. "A general topology-based mesh data structure". In: *International Journal for Numerical Methods in Engineering* 40.9 (1997), pp. 1573–1596.

[9] Robert E Calem. "New York's panix service is crippled by hacker attack". In: *Accessed: Dec* (2018).

[10] Jochen Christ, Larysa Visengeriyeva, and Simon Harrer. 2022. URL: `https://www.datamesh-architecture.com/` (visited on May 21, 2023).

[11] Confluent Confluent. *Data Mesh Demo*. 2023. URL: `https://github.com/confluentinc/data-mesh-demo` (visited on June 20, 2023).

[12] DDoSDissector DDoSDissector. *DDoSDissector*. 2023. URL: `https://github.com/ddos-clearing-house/ddos_dissector` (visited on June 26, 2023).

[13] Edith De Leeuw and William Nicholls. "Technological innovations in data collection: Acceptance, data quality and costs". In: *Sociological Research Online* 1.4 (1996), pp. 23–37.

[14] Z. Dehghani and M. Fowler. *Data Mesh: Delivering Data-driven Value at Scale*. 2022.

[15] Zhamak Dehghani. *Data Mesh Principles and logical architecture*. 2020. URL: `https://martinfowler.com/articles/data-mesh-principles.html`.

[16] Deistercloud. *Deistercloud Sample TPCH Queries*. 2023. URL: https://docs.deistercloud.com/content/Databases.30/TPCH%20Benchmark.90/Sample%20querys.20.xml?embedded=true (visited on July 12, 2023).

[17] Calvin Falter. *Design and Implementation of a Commit Evaluation Engine for an Open Source Donation Platform*. Communication Systems Group, Department of Informatics, 2020. URL: https://files.ifi.uzh.ch/CSG/staff/scheid/extern/theses/BA-C-Falter.pdf.

[18] Paul Gardner-Stephen et al. "MeshMS: Ad Hoc Data Transfer within Mesh Network". In: *International Journal of Communications, Network and System Sciences* 05.08 (2012), pp. 496–504.

[19] Thomer M Gil and Massimiliano Poletto. "{MULTOPS}: A {Data-Structure} for bandwidth attack detection". In: *10th USENIX Security Symposium (USENIX Security 01)*. 2001.

[20] Google Google. *Dataplex Overview*. 2023. URL: https://cloud.google.com/dataplex/docs/introduction (visited on June 20, 2023).

[21] Google Google. *Google Cloud*. 2023. URL: https://cloud.google.com/docs?hl=de (visited on June 20, 2023).

[22] Google Google. *Google Cloud*. 2023. URL: https://cloud.google.com/bigquery/docs (visited on June 20, 2023).

[23] Grafana Grafana. *Grafana*. 2023. URL: https://grafana.com/docs/grafana/latest/ (visited on June 24, 2023).

[24] Seyed Milad Helalat. *An Investigation of the Impact of the Slow HTTP DOS and DDOS attacks on the Cloud environment*. 2017.

[25] IPFS IPFS. *IPFS*. 2023. URL: https://ipfs.tech/#how (visited on June 20, 2023).

[26] Sean Kandel et al. "Enterprise data analysis and visualization: An interview study". In: *IEEE transactions on visualization and computer graphics* 18.12 (2012), pp. 2917–2926.

[27] Angelos D Keromytis, Vishal Misra, and Dan Rubenstein. "SOS: An architecture for mitigating DDoS attacks". In: *IEEE Journal on selected areas in communications* 22.1 (2004), pp. 176–188.

[28] Alexander Kniesz and Rainer Jaspert. *MinIO and Trino*. 2022. URL: https://www.datamesh-architecture.com/tech-stacks/minio-trino.

[29] Daniel Kopp, Christoph Dietzel, and Oliver Hohlfeld. "DDoS never dies? An IXP perspective on DDoS amplification attacks". In: *Passive and Active Measurement: 22nd International Conference, PAM 2021, Virtual Event, March 29–April 1, 2021, Proceedings 22*. Springer. 2021, pp. 284–301.

[30] LakeFS LakeFS. *LakeFS*. 2023. URL: https://github.com/treeverse/lakeFS (visited on June 20, 2023).

[31] Neil Long and Rob Thomas. "Trends in denial of service attack technology". In: *CERT Coordination Center* 648 (2001), p. 651.

[32] Tasnuva Mahjabin et al. "A survey of distributed denial-of-service attack, prevention, and mitigation techniques". In: *International Journal of Distributed Sensor Networks* 13.12 (2017), p. 1550147717741463.

[33] J. Majchrzak, S. Balnojan, and M. Siwiak. *Data Mesh in Action*. 2023.

[34] Vivien Marx. "The big challenges of big data". In: *Nature* 498.7453 (June 2013), pp. 255–260.

[35] MinIO MinIO. *MinIO*. 2023. URL: `https://github.com/minio/minio` (visited on June 20, 2023).

[36] Jelena Mirkovic, Gregory Prier, and Peter Reiher. "Attacking DDoS at the Source". In: *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* IEEE. 2002, pp. 312–321.

[37] Jelena Mirkovic and Peter Reiher. "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms". In: *SIGCOMM Comput. Commun. Rev.* 34.2 (2004), 39–53.

[38] Jelena Mirkovic and Peter Reiher. "D-WARD: a source-end defense against flooding denial-of-service attacks". In: *IEEE transactions on Dependable and Secure Computing* 2.3 (2005), pp. 216–232.

[39] MySQL MySQL. *MySQL*. 2023. URL: `https://dev.mysql.com/doc/` (visited on Aug. 8, 2023).

[40] George Oikonomou et al. "A framework for a collaborative DDoS defense". In: *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE. 2006, pp. 33–42.

[41] Christos Papadopoulos et al. "Cossack: Coordinated suppression of simultaneous attacks". In: *Proceedings DARPA information survivability conference and exposition.* Vol. 1. IEEE. 2003, pp. 2–13.

[42] Kihong Park and Heejo Lee. "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets". In: *ACM SIGCOMM computer communication review* 31.4 (2001), pp. 15–26.

[43] J. Rejina Parvin. "An Overview of Wireless Mesh Networks". In: *Wireless Mesh Networks - Security, Architectures and Protocols*. May 2020.

[44] Vern Paxson. "An analysis of using reflectors for distributed denial-of-service attacks". In: *ACM SIGCOMM Computer Communication Review* 31.3 (2001), pp. 38–47.

[45] Franck Ravat and Yan Zhao. "Data lakes: Trends and perspectives". In: *Database and Expert Systems Applications: 30th International Conference, DEXA 2019, Linz, Austria, August 26–29, 2019, Proceedings, Part I 30*. Springer. 2019, pp. 304–313.

[46] Bruno Rodrigues et al. "Blockchain Signaling System (BloSS): Cooperative Signaling of Distributed Denial-of-Service Attacks". In: *Journal of Network and Systems Management* 28.4 (Aug. 2020), pp. 953–989.

[47] Ajit Singh. "METHODS FOR DATA LAKES IMPLEMENTATION." In: *Intercathedra* 48.3 (2021).

[48] Snowflake Snowflake. *TPCH*. 2023. URL: `https://docs.snowflake.com/en/user-guide/sample-data-tpch` (visited on Aug. 4, 2023).

[49] Starburst Starburst. *Starburst*. 2023. URL: `https://www.starburst.io/platform/starburst-enterprise/` (visited on June 20, 2023).

[50] Brian Stein and Alan Morrison. "The enterprise data lake: Better integration and deeper analytics". In: *PwC Technology Forecast: Rethinking integration* 1.1-9 (2014), p. 18.

[51] Jessica Steinberger et al. "Collaborative DDoS defense using flow-based security event information". In: *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2016, pp. 516–522.

[52] Trino Trino. *Trino Docs*. 2023. URL: `https://trino.io/docs/current/index.html` (visited on June 20, 2023).

[53] Trino Concepts Trino Concepts. *Trino Concepts*. 2023. URL: `https://trino.io/docs/current/overview/concepts.html` (visited on July 12, 2023).

[54] Trino Connectors Trino Connectors. *Trino Connectors*. 2023. URL: `https://trino.io/docs/current/connector.html` (visited on July 12, 2023).

[55] Trino Deployment Trino Deployment. *Trino Deployment*. 2023. URL: `https://trino.io/docs/current/installation/deployment.html` (visited on July 12, 2023).

[56] Trino Docker Trino Docker. *Trino Docker*. 2023. URL: `https://trino.io/docs/current/installation/containers.html` (visited on July 12, 2023).

[57] Trino Git Trino Git. *Trino Git*. 2023. URL: `https://github.com/trinodb/trinol` (visited on July 12, 2023).

[58] Trino TPCH Trino TPCH. *Trino TPCH*. 2023. URL: `https://trino.io/docs/current/connector/tpch.html` (visited on July 12, 2023).

[59] Trino Worker Properties Trino Worker Properties. *Trino Worker Properties*. 2023. URL: `https://trino.io/docs/current/admin/properties-task.html` (visited on July 12, 2023).

[60] Nikhil Tripathi and Neminath Hubballi. "Application layer denial-of-service attacks and defense mechanisms: a survey". In: *ACM Computing Surveys (CSUR)* 54.4 (2021), pp. 1–33.

[61] Coral Walker and Hassan Alrehamy. "Personal Data Lake with Data Gravity Pull". In: *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*. 2015, pp. 160–167.

[62] Xiaowei Yang, David Wetherall, and Thomas Anderson. "A DoS-limiting network architecture". In: *ACM SIGCOMM Computer Communication Review* 35.4 (2005), pp. 241–252.

Abbreviations

ABI	Application Binary Interface
AITF	Active Internet Traffic Filtering
AWS	Amazon Web Service
BI	Business Intelligence
BloSS	Blockchain Signaling System
CIA	Confidentiality, Integrity and Availability
CSIRT	Computer Security Incident Response Team
DDoS	Distributed Denial of Service
DoS	Denial of Service
DNS	Domain Name System
DOTS	Distributed-Denial-of-Service Open Threat Signaling
ETH	Ether (Cryptocurrency)
EVM	Ethereum Virtual machine
IaaS	Infrastructure as a Service
IETF	Internet Engineering Task Force
IoT	Internet of Things
IPFS	Inter Planetary File System
ISP	Internet Service Provider
NFV	Network Function Virtualization
P2P	Peer to Peer
PoA	Proof-of-Authority
PoW	Proof-of-Work
REST	Representational State Transfer
RTT	Round Trip Time
SDN	Software-Defined Networking
SPI	Service Provider Interface
SLA	Service Level Agreement
VNF	Virtualized Network Function

List of Figures

List of Tables

Listings

Appendix A

Supplementary Contents

Trino Dashboard:

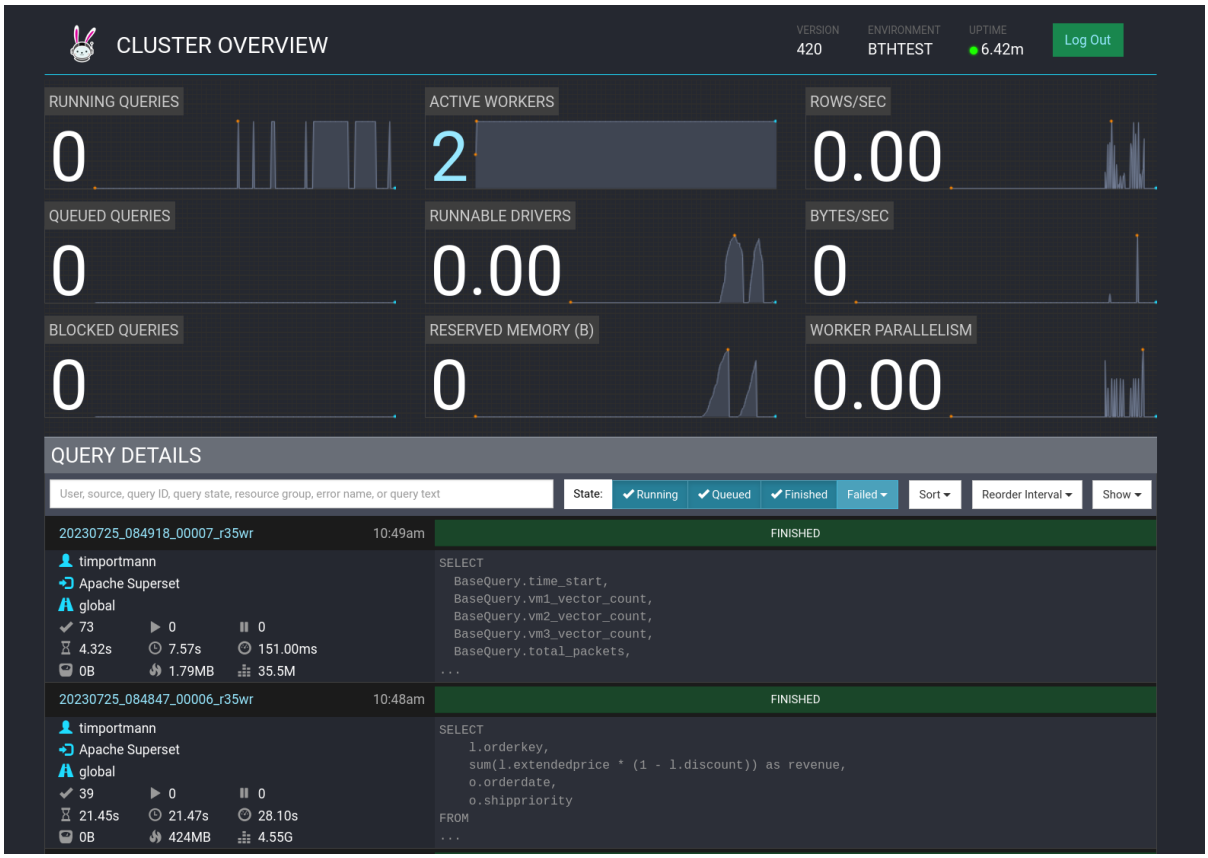

Figure A.1: Trino Dashboard Overview

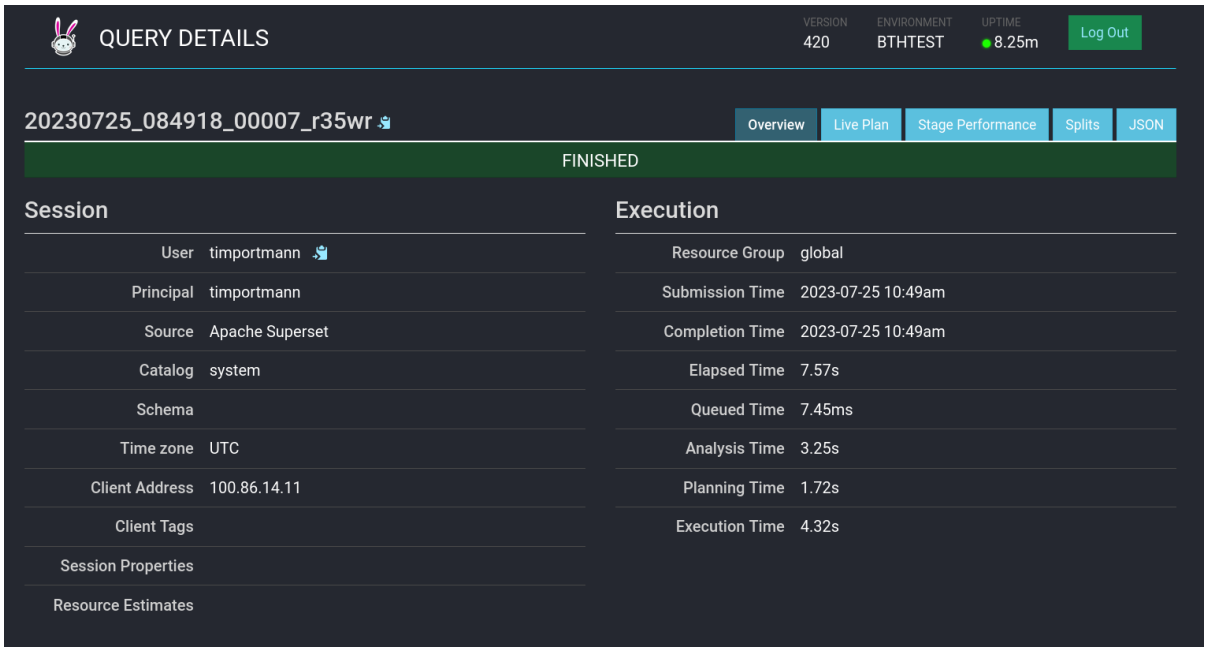

Figure A.2: Trino Query Details (1)

Figure A.3: Trino Query Details (2)

Example DDoS Fingerprint:


```
{
  "attack_vectors":[
    {
      "service":null,
      "protocol":"ICMP",
      "source_ips":[
        "110.30.152.226",
        "112.167.110.73",
        "128.0.203.96",
        "128.235.229.87"
      ],
      "source_ips_real":{
        "128.0.203.96":"10.0.17.165",
        "128.235.229.87":"10.0.17.165",
        "110.30.152.226":"10.0.17.165",
        "112.167.110.73":"10.0.17.165"
      },
      "ttl":{
        "128":1
      },
      "ttl_by_source":{
        "128.0.203.96":[
          128
        ],
        "128.235.229.87":[
          128
        ],
        "110.30.152.226":[
          128
        ],
        "112.167.110.73":[
          128
        ]
      },
      "time_start":"1970-01-01T00:00:03.171650",
      "duration_seconds":35.15932,
      "nr_packets_by_source":{
        "128.0.203.96":29752.58823529412,
        "128.235.229.87":29752.58823529412,
        "110.30.152.226":29752.58823529412,
        "112.167.110.73":29752.58823529412
      },
      "nr_packets":505794,
      "nr_megabytes":411.28,
      "detection_threshold":1
    }
  ],
  "target":"75.220.17.56",
  "location":"10.0.17.165",
  "key":"066eff111e1b0833abbd340f52557f2d"
}
```


Listing A.1: Example DDoS Fingerprint

Superset SQL Lab Environment

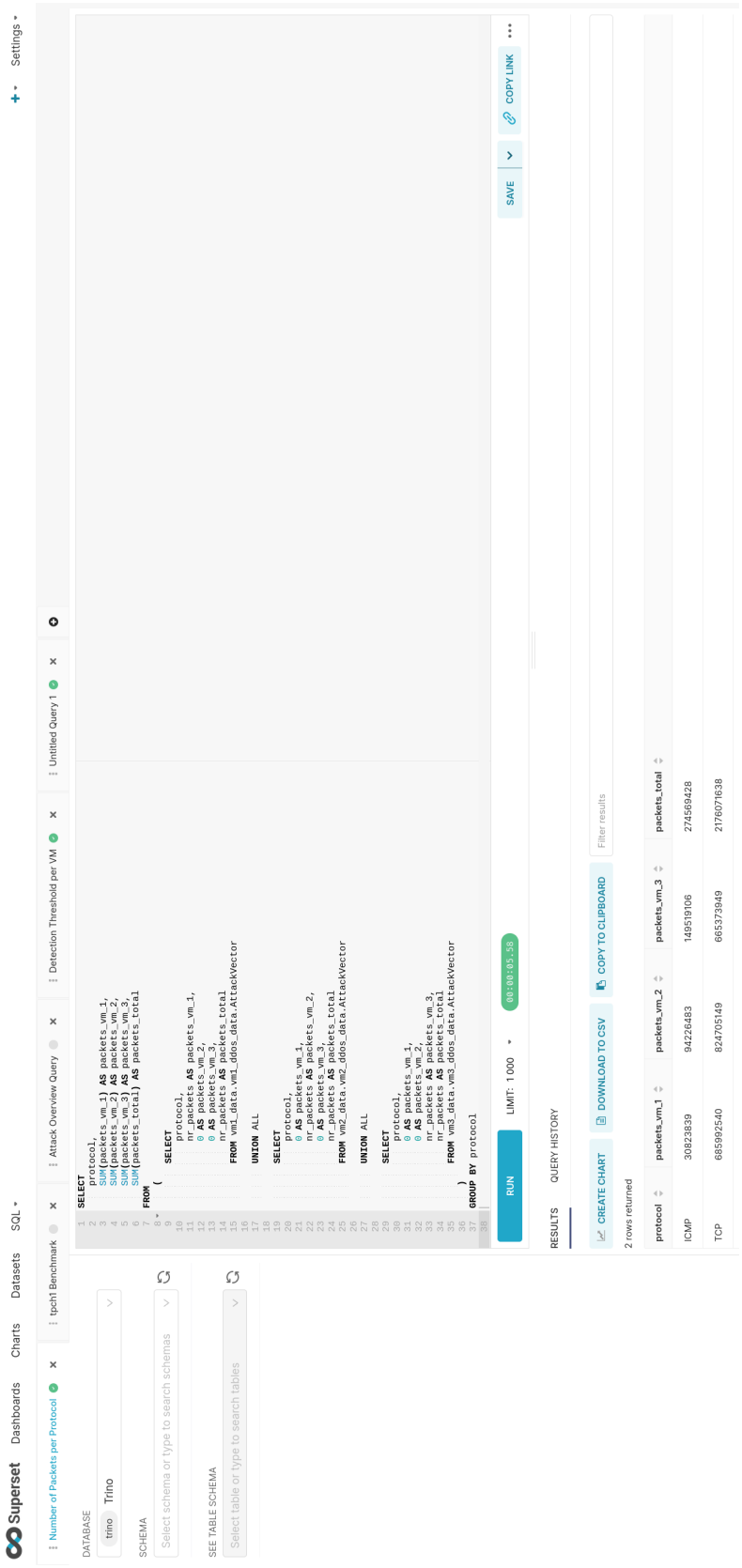

Figure A.4: Superset SQL Lab Environment

Complete Visualization for *Query 2* (5.2)

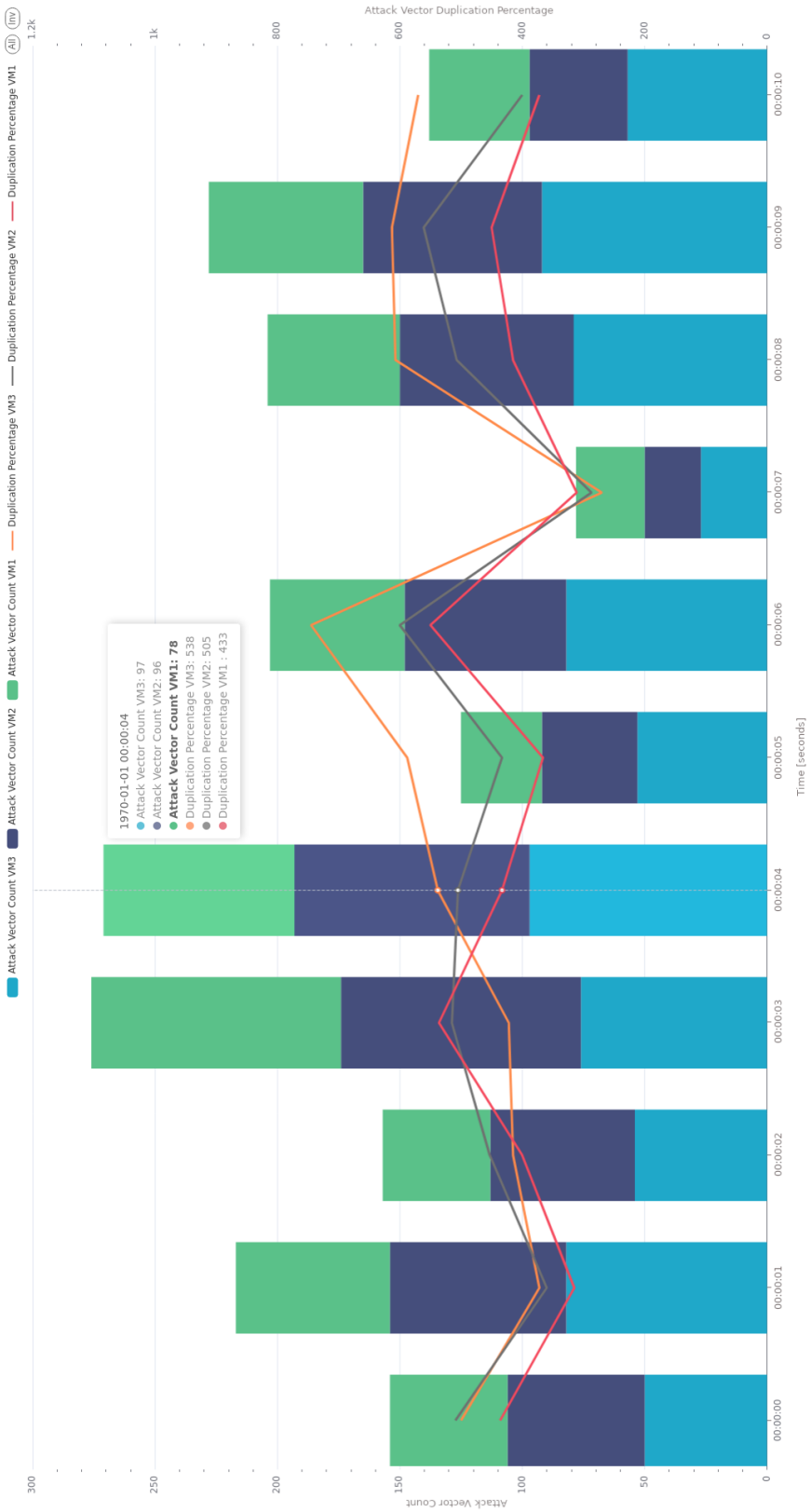

Figure A.5: Complete: Attack Vector Counts and Duplication Percentages per VM over the Attack Duration

TPC-H Benchmark Schema:

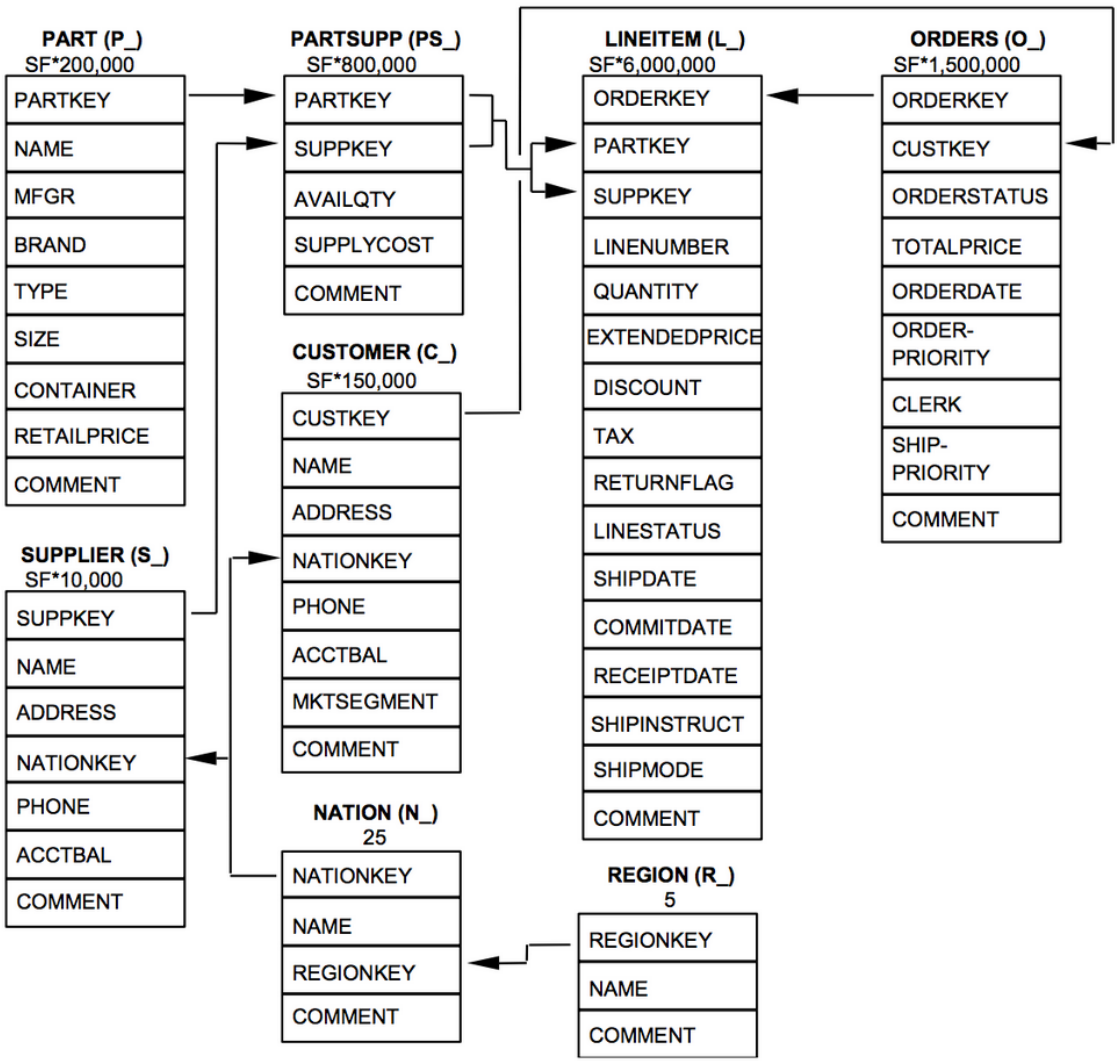

Figure A.6: TPC-H Benchmark Schema [48]