Evaluation of Centralised vs Distributed Collaborative Intrusion Detection Systems in Multi-Access Edge Computing

Rahul Sharma[1], Chien Aun Chan[2], Christopher Leckie[1]
[1]School of Computing and Information Systems
[2]Department of Electrical and Electronic Engineering
The University of Melbourne, Australia
Email: {rahuls2, chienac, caleckie}@unimelb.edu.au

***Abstract*—With the rapid adoption of next generation networking architectures in 5G, like *Multi-Access Edge Computing* (MEC), there is a shift in the core processing capabilities to the edge of the network. This helps facilitate higher bandwidth and ultra-low latency responses, but can increase the attack surface for cyber-attacks like *Denial of Service* (DoS) and *Distributed Denial of Service* (DDoS). To safeguard these architectures without degrading performance, we require mechanisms capable of detecting these attacks in near-real time. *Collaborative Intrusion Detection Systems* (CIDS) are a popular choice for detecting sophisticated coordinated attacks in large complex networks, and a prime candidate for use in this context. However, finding the right CIDS deployment model is not straightforward, with each model presenting its own set of challenges like accuracy as well as network and computational overhead. In this paper, we focus on evaluating two CIDS models - a centralised vs a distributed approach using *Distributed Hash Tables* (DHTs), based on their detection accuracy, CPU and memory usage, and data transmission overhead. We assess each approach through experimentation with a real-world worm dataset to understand the complexities involved in developing an efficient intrusion detection solution for MEC.**

***Index Terms*—5G, Multi-Access Edge Computing, Cybersecurity, Intrusion Detection, DHT**

I. Introduction

The growth in data from Internet of Things (IoT) sensors and controllers has created demand for high-performance compute and storage environments to offload data processing from centralised servers. Cloud computing is a primary candidate for these use cases, but falls short in real-time applications demanding ultra-low latency access, high-bandwidth, context and location awareness [1].

These requirements have motivated the introduction of MEC, deemed capable of meeting the demands of these real-time applications. Edge Computing covers a broad range of techniques, designed to move compute and storage capabilities from centralised data centers (public and/or private) to the edge of the mobile network [2] [3]. This architecture facilitates data processing closer to end users, reducing response times and leveraging a heterogeneous set of edge servers. Major cloud and telecommunication providers are starting to rollout these service offerings as part of their portfolio, for instance Google provides Edge TPU [4], Microsoft has Azure IoT Edge [5], and Amazon Web Services (AWS) launched AWS Wavelength [6] for edge computing.

ISBN 978-3-903176-28-7 © 2020 IFIP

However, attacks targeting edge computing infrastructure have also risen dramatically in recent years [7]–[10]. For instance, the Mirai botnet attack took control of over 65,000 IoT devices within the first 20 hours of its release in August 2016 [7]. These compromised devices were used as part of a botnet to launch DDoS attacks on edge servers, affecting over 178,000 domains [8]. Following this, variations of Mirai such as IoTReaper and Hajime were identified, which reportedly affected more than 378 million devices in 2017 [7].

To prevent exposure to such attacks, it becomes crucial for MEC to leverage intrusion detection mechanisms for identifying anomalous behavior in near real-time [1] [11]. Most of the current IDS solutions propose deployment in the core backbone network, with edge-focused solutions still being in their infancy. Looking at the complex attack surface exhibited by MEC, it is important to identify trends between malicious activities occurring at multiple locations, like a CIDS. For instance, [12] proposed a Distributed Edge-based IDS for analysing IoT application traffic from multiple edge devices using CIDS. However, there are a range of CIDS deployment models to choose from, ranging from a centralised processing focus to distributed load sharing architectures. This presents a challenge in identifying the best CIDS approach in an MEC environment, making it crucial to understand the complexities involved in each mechanism. Vasilomanolakis et al. [13] outlined some of the requirements needed when choosing between different CIDS deployment models, such as accuracy, minimal overhead, scalability, resilience, privacy, self-configuration and interoperability. We use accuracy and overhead (computational and network) as our comparison parameters due to their importance in MEC to ensure strong security patterns without degrading network performance. Our novel contribution is to evaluate two CIDS mechanisms, namely a Centralised CIDS architecture versus a purely Distributed CIDS architecture to

detect coordinated attacks using a real-world worm dataset, addressing the following questions:

- How does a Centralised CIDS compare against a purely Distributed CIDS in evaluating intrusion detection accuracy in MEC?
- How do these architectures compare in utilising CPU and memory usage?
- Which of the two CIDS architectures incur lower data transmission overhead to share relevant information between peers in practice?

To address these questions, we implement a Centralised CIDS architecture having edge processing units, that filters and forwards alerts to a central correlation unit. This central unit tracks the frequencies of requests arising from a specific source within a given time frame to verify attacks. For our distributed CIDS architecture, we leverage DHTs for their fault tolerance through Peer-to-peer (P2P) communication and focused data sharing capability using a publish-subscribe model. These nodes form an overlay network to reduce the data transmission overhead. Each node runs a DHT instance in a trusted environment, tracking the frequencies of requests from specific sources by ingesting identified alerts as a single dimensional correlation query entry for sharing amongst peers. Nodes can identify malicious traffic from the same source and attain a global view of other nodes who observe malicious traffic from the same source through their subscription capability, thus validating the identified attack.

To evaluate the detection accuracy and performance of these models, we perform an empirical analysis using a large real-world worm dataset, namely - "Three days of Conficker" [14]. This dataset has been collected by the Center for Applied Internet Data Analysis (CAIDA), tracing the presence of Conficker A and B attacking specific services that have been anonymised. Our contribution demonstrates the effects on detection accuracy, CPU and memory use, and network overhead in detecting Conficker A using different CIDS architectures for alert correlation in an MEC environment.

This paper is structured as follows. Section II addresses related work of intrusion detection starting from traditional networks, and its progression towards edge computing. Section III discusses our implementation methodology for Centralised and Distributed CIDS in MEC. In Section IV, we present the results of our evaluation of the two approaches through deployment on virtual machines, and conclude in Section V.

II. Related Work

In this section, we review related work on CIDS and its current outlook from the perspective of MEC.

A. Traditional Collaborative Intrusion Detection Deployment

Collaborative Intrusion Detection Systems have been an active area of research for addressing coordinated attacks across multiple heterogeneous networks [15]–[17]. Where traditional IDSs are limited to the scope of their own network, a CIDS leverages a global view across multiple networks by scaling and sharing context between peers. Hence, a CIDS can extend its detection range across multiple networks, enabling them to make better decisions. There are different classifications of CIDS based on their communication architecture [16], such as:

- ***Centralised* CIDS**: These comprise multiple monitoring nodes that forward alerts to a central processing unit for aggregation and analysis.
- ***Hierarchical* CIDS**: These maintain a multi-layered architecture, with each layer filtering alerts, and transmitting relevant data only to the layer above. This data moves up the hierarchy until it reaches the central processing node for aggregation and processing.
- ***Distributed* CIDS**: Data filtering, detection, and processing tasks are shared between all peers in a decentralised manner.

LarSID [17] is a general-purpose Collaborative Detection scheme to share suspicious evidence with peers. It focuses on identifying worm outbreaks and detection of large scale stealthy coordinated scans across multiple nodes, using a modified Pastry DHT [18] [19] as its decentralised data sharing mechanism. It leverages a publish-subscribe mechanism where each node acts as a monitor that filters and evaluates traffic. Every node maintains a watchlist for its local subnetwork, subscribing and correlating messages that are shared by other nodes, and generating notifications for peers to describe any identified malicious traffic. Through experimentation, they demonstrated that a distributed architecture fares better than a centralised one, as it alleviates potential bottlenecks through decentralisation of the consensus-based decision-making process.

Split-and-Merge [16] uses a Centralised CIDS solution to track network trends and note changes in port usage using a modified Z-score that has outlier resistant properties [20]. It maintains local detection modules that are distributed across networks to analyse traffic and forward alerts to a central controller for aggregation. This central controller analyses all incoming packets to confirm malicious traffic and notifies the local detection modules to update their baseline evaluation of the port specific features used for anomaly detection. This setup works well in identifying attacks with high accuracy, but the central module still presents a bottleneck as its absence decouples the local detection modules, thus losing its global view of the network.

B. Intrusion Detection in Edge Networks

The authors of [9] proposed a six-layer edge computing intrusion detection model, with each layer undertaking specific roles in the architecture. It utilises detection modules in the edge nodes, which are responsible for identifying attacks and monitoring the underlying state of its host. The logs and metrics generated at this layer are forwarded to a cloud server for analysis to generate reports and derive insights. The cloud layer is also responsible for orchestration and management of edge nodes to facilitate intrusion forensic investigation. This deployment setup uses a hybrid approach to intrusion

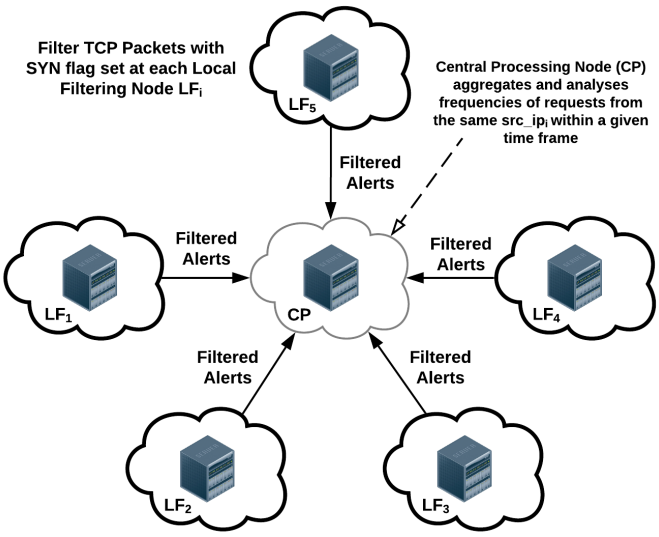

Fig. 1. Centralised CIDS Architecture

detection, where dependence on a cloud data center is key for orchestration and data sharing.

In [21], the authors proposed a three-layered framework for Distributed Intrusion Detection Systems using Edge Computing, coordinating between edge nodes and a cloud data center. They use a filter layer for an IDS to inspect traffic and reduce false alarms while coordinating with peers. These IDS further communicate with the Edge layer to analyse shared data using appropriate algorithms selected locally. The edge nodes sift through all the data and forward computationally expensive operations to a cloud data center for analysis if needed. This architecture results in sharing large volumes of data between the edge layer and the cloud data center, which adds an additional communication burden that introduces delays in response based on the geolocation of these individual layers.

Looking at these hybrid IDS models in edge computing, we can observe a trend in utilising both edge nodes and cloud data centers. However, it also presents an increase in the communication overhead to share information back and forth between these layers, while increasing response time [22] [23]. This becomes an issue when trying to perform real-time intrusion detection. To address these issues, we focus on removing the cloud data center dependency for orchestration and load sharing by evaluating the detection process in the MEC directly.

III. CIDS Architectures

In this section, we give an overview of the reference architectures (i.e., Centralised and Distributed CIDS) that are the basis of our study.

A. Centralised CIDS Architecture

Our Centralised CIDS comprises two components - local filtering nodes (LF_i) and a centralised processing node (CP) as seen in Figure 1. Local filtering nodes are deployed across multiple MEC nodes, and used as a pre-filtering module to scan network traffic in the local subnet of where it is deployed. They filter network traffic, extracting TCP packets containing the SYN flag (packets initiating connection) and forward that information to the central processing node.

Algorithm 1: Centralised CIDS Processing

```
// Local Filtering Module
while Incoming Network Traffic do
    if pkt_i.hasTCP && pkt_i.flags == "SYN" then
        filtered_data ⟵ pkt_i;
        send_data(filtered_data);
    end
end

// Central Processing Module
while Incoming Filtered Data do
    Queue ⟵ filtered_data;
end

// Aggregation Logic Running In Separate Thread
Function data_aggregator():
    pkt_i ⟵ Poll(Queue);
    if pkt_i.src_ip != watch_list {src_ip_i} then
        watch_list {src_ip} = 1 ;
    else if pkt_i.src_ip == watch_list {src_ip_i} then
        watch_list {src_ip} += 1 ;
    end
    if watch_list {src_ip} % threshold == 0 then
        Log ⟵ pkt_i.src_ip
    end
End Function
```

The central processing node inserts received packets into a queue, popping them sequentially for aggregation. It first checks if the source IP address of that packet exists in the *watch list* of the central processing node. If it is present, we increment a counter against that source IP in the watch list, indicating the amount of traffic received from that source. If the source IP address is not present, we add it to the watch list. We use different thresholds in our experiments, namely 2, 3, or 5, to keep track of the frequency of packets incoming from the same source IP address. If the number of incoming packets is equal to a multiple of the threshold then we log the traffic as an alert for intrusion prevention or other mitigation actions. Our watch list is empty initially and is refreshed every 10 mins, thus, reducing the overall memory requirements of the centralised architecture, while keeping focus on current traffic patterns.

B. Distributed CIDS Architecture

For our distributed architecture, we use an Opensource DHT called OpenDHT [24] based on Kademlia [25] similar to the model used by [17]. Here, each edge server runs a DHT node in a trusted environment, connecting with peers through the formation of an overlay network using P2P communication as seen in Figure 2.

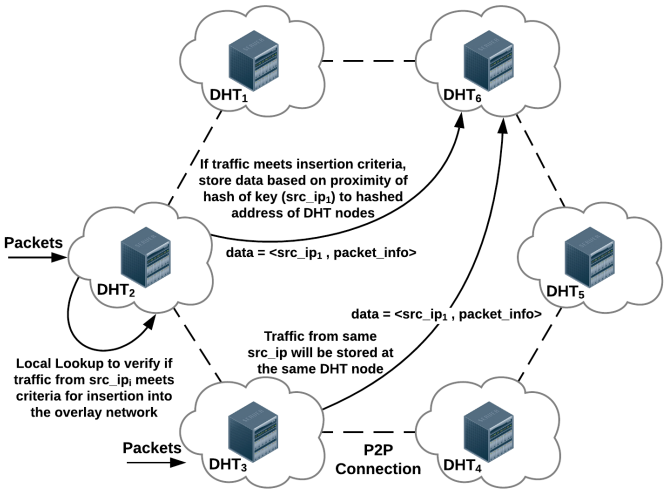

Fig. 2. Distributed CIDS Architecture

Similar to our Centralised CIDS, each DHT node scans the network traffic in the local subnet of its deployment, extracting TCP packets containing the SYN flag. Each node checks if the source IP address of that packet exists in its local watch list. If the source address is not present, the edge node creates an entry in its watch list and inserts data about the packet as a single dimensional key <Key,Value> format in our DHT network. Each key can record multiple value entries for a fixed period of time (default of 10 minutes in our tests), presenting a global view of any and all updates of a referenced correlation query key. Some of the main benefits of using a DHT are that it transmits referenced data only, reducing the network overhead. However, these references are based solely on a single dimensional key in OpenDHT. Hence, we use source IP address as this key, similar to the approach taken by [17]. OpenDHT provides a callback functionality called *LISTEN* which allows nodes to trigger an automatic callback against a key whenever an update is registered against it. This enables a publish-subscribe model, where nodes directly subscribe on certain keys and the DHTs publish all updates to the subscribers of a key. In our experiments, every time a packet from a source is observed for the first time within a 10 minute interval, we set a callback on that source IP in the DHT network.

We use different thresholds in our Distributed CIDS architecture, namely, 2 or 3. These thresholds are used to identify relevant source IP addresses to insert into the DHT, and also form a consensus to validate a source IP address to be malicious. For instance, if the source IP of the packet being analysed exists in the watch list of our edge node, we update a counter against that source IP address in the watch list and verify if the frequency of packets observed from this source is equal to a multiple of our threshold. If so, we insert data about this packet into the DHT network using the source IP as our single dimensional key.

Algorithm 2: Distributed CIDS Processing

```
// Local DHT Node Entry Point
while Incoming Network Traffic do
    if pkt_i.hasTCP && pkt_i.flags == "SYN" then
        Queue ⟵ pkt_i;
    end
end

// Aggregation Logic Running In Separate Thread
Function data_filter():
    while true do
        pkt_i ⟵ Poll(Queue);
        if pkt_i.src_ip != watch_list {src_ip_i} then
            watch_list {src_ip} = 1 ;
            DHT ← PUT(pkt_i.src_ip, {Msg})
            LISTEN(src_ip)
        else if pkt_i.src_ip == watch_list {src_ip_i} then
            watch_list {src_ip} += 1 ;
        end
        if watch_list{src_ip} % threshold==0 then
            DHT ← PUT(pkt_i.src_ip, {Msg})
        end
    end
End Function

// Track Consensus Using Callback
Function listen_callback():
    while true do
        p_i ⟵ callback_response
        if "expired"! = p_i.status then
            if node! = p_i.current_node then
                if p_i.src_ip! = peers[src_ip] then
                    if len(peers) >= threshold then
                        Notify(peers[src_ip])
                    end
                end
            end
        end
    end
End Function
```

IV. Evaluation

The experiments in our evaluation aim to compare the data transmission overhead, detection accuracy, and CPU and memory usage of our two CIDS models in an MEC environment. The evaluations are performed using different detection thresholds to measure the frequency of packets from the same source IP arriving at edge nodes and capture node performance metrics across multiple time frames.

A. Experimental Dataset

We compare the performance of our two approaches outlined in Section 3 using a real-world worm dataset called "Three Days of Conficker" [14] made available by the Center for Applied Internet Data Analysis (CAIDA). It is composed

of a network telescope having a /8 globally routed network space carrying versions A and B of the Conficker worm across 3 days. The first day (i.e. 21st November 2008) captures the initial presence of Conficker A, the second day (on 21st December 2008) reflects the presence of only Conficker A, and on the third day on 21st January 2009, both Conficker A and Conficker B are active.

The packets contain anonymised destination IP addresses, with the majority of the traffic targeting 445/tcp [26] [27], both in terms of the number of packets and the source IP addresses observed. In our experiments, we use a sample of the first day containing the initiating phase of the Conficker Worm A to understand the effectiveness of both our CIDS mechanisms in identifying its outbreak. Our data comprises 16,358,338 raw packets, out of which 4,204,248 packets are used to initiate a TCP connection. From these, we extract the TCP SYN messages sent as part of the TCP 3-way handshake to focus the scope of our attack pattern. During analysis of these packets, we capture the corresponding 5 tuples for aggregation purposes within our respective approaches.

B. Centralised CIDS Setup

We simulate our MEC setup using virtual machines (VMs) running in a single data center. For the Centralised architecture, we deployed our local filtering module on 5 VMs and used 1 VM as our central processing node. The local filtering nodes read data from the Conficker worm dataset, and filter traffic for TCP SYN messages. We recover the 5 tuples from these filtered SYN packets and forward this payload to the central processing node for aggregation.

The central node inserts incoming http packets into a queue and polls it in a separate thread to ingest and filter data concurrently. The filtered information is forwarded to the aggregation logic to track all source IPs observed. The results of the analysis are logged in a MySQL database for the first and n^{th} multiple of occurrence of a source IP within a given timeframe (we use a 10 minute window), constantly refreshing memory after this time.

If the source IPs are observed only once within a specific timeframe, we treat it as suspicious, but if multiple requests are aggregated and it crosses the threshold on requests observed as alerts - those source IPs are mapped as malicious. This is used to validate malicious traffic originating from specific sources, and an alert notification can be created to initiate intrusion prevention and other mitigation procedures.

C. Distributed CIDS Setup

Similar to the Centralised setup, we deploy instances of our DHT on 6 VMs, with each VM running our processing logic. The DHTs form an overlay network routed via a private namespace, dynamically registered via peers. We start the process by reading the network data from our worm dataset, filtering on the TCP SYN messages. Similar to the centralised approach, we use the first and n^{th} multiple of occurrence of our source IPs in each 10 minute window, inserting them into the local DHT node.

The DHT creates a hash of the entry based on the Key (source IP address in our case) used to decide the placement of our <Key, Value> pair across one of the nodes in the overlay network. Any subsequent values published for this Key will be hashed and placed in the same location as the original entry using the Key as the single dimension namespace across the entire overlay network.

For our workflow, we set a callback for every new source IP inserted into the local DHT node. This enables each local node to track updates on that source IP as flagged by other peers, leveraging the distributed nature of the architecture. It also allows nodes to gain a global context on that source IP address with minimal data transmission and delay.

If multiple nodes identify the same source IP address to be suspicious then we can validate the malicious nature of traffic originating from that source. However, picking the right threshold value to enable this is not an easy problem to address, so we evaluate with varying thresholds to observe the affects on the detection process for validating malicious source IPs. The consensus based detection process can be used to initiate global and local mitigation procedures to limit the impact of the identified malicious traffic on our services/destinations. The distributed nature of the DHTs allows us to share our load across multiple heterogeneous nodes, irrespective of their physical configuration, which is a key aspect of MEC.

Currently, the DHT is configured to store data for a limited period of time only to save space as edge nodes are often resource constrained. When a value expires due to the retention time limit, it is promptly deleted from the overlay network. Nodes subscribed to this key will receive a notification with an expired tag in the payload, enabling us to keep track of the data ingested into the network and log its entire lifecycle.

During callback responses, we filter out messages inserted into the DHTs by the same nodes to avoid false aggregations and keep track of all MEC nodes in the network reporting the same source IP to be suspicious. If we reach a consensus (equal to or above a threshold) from peers about a source IP address being malicious, we trigger custom notifications to notify peers (both new and old) confirming the source IP to be malicious. This provides a more fine-grained control towards coordinating the sharing of information between peers.

D. Data Transmission Overhead

In our experiment, we tracked the amount of data transmitted across our components to identify the total network overhead involved. Each transmitted message is composed of 732 Bytes including its HTTP headers, with the Centralised CIDS transmitting a total of 3,520,892 messages and each filter node transmitting an average of 704,178.4 messages to the central processing node. Similarly, the Distributed approach transmitted a total of 1,000,191 messages with each node transmitting an average of 166,698.5 messages for suspicious IP addresses for ingestion into the DHT, with approximately 27,624 messages sent out as notifications to peers to validate malicious source IP addresses when using a threshold 3, and 134,567 messages when using a threshold 2.

TABLE I
DATA TRANSMISSION BETWEEN DIFFERENT CIDS MODELS (IN GB)

CIDS Mechanism	*Data Transmitted (in GB)*
Centralised CIDS	2.5
Distributed CIDS (Threshold of 2)	1.5
Distributed CIDS (Threshold of 3)	1.4

As shown in Table 1, the Distributed CIDS requires far less data transfer (i.e., a 40% reduction compared to the Centralised CIDS), as it only transmits data when ingesting it into the DHT or when receiving updates via callbacks. Using its publish-subscribe mechanism, only subscribed nodes receive updates, hence reducing unnecessary publish messages to all peers. If an attack is distributed across multiple nodes the DHT helps validate malicious traffic with less network overhead, enabling edge nodes to perform intrusion prevention/mitigation procedures faster. The Centralised CIDS, however, requires all alerts to be forwarded to a centralised processing node, resulting in significantly more data transfer.

In private networks, organisations may focus more on security as opposed to network overhead. This makes the Centralised CIDS an ideal choice due to the ease in securing such architectures, albeit at the cost of fault tolerance, as the centralised processing node is required to be online at all times for this to work. The Distributed CIDS, on the other hand, provides a more fault tolerant approach with less network overhead, but may require additional measures in place to secure the distributed deployment.

E. Accuracy Model

We deployed our two models using different threshold values to identify changes in detection accuracy. For instance, we ran our Centralised CIDS implementation with 2, 3 or 5 as our thresholds, i.e., whenever it observed frequencies of alerts originating from the same source IP more than these set thresholds within a given timeframe (10 mins) - it flagged the source IP to be malicious. Similarly, we used a threshold of 2 or 3 in our distributed architecture to decide when to insert data into the DHT network and also to form consensus between peers to validate a source IP address to be malicious.

Based on the results in Figures 3-7, we observe that the Centralised CIDS model flagged significantly more traffic as part of its analysis, the majority of which is targeting port 445 used by the Conficker worm based on its attack model [26] [27]. Similarly, the Distributed CIDS also flagged port 445 to be the predominant port number being attacked, although the amount of traffic captured is far less. This is due to attacks happening on a small number of edge nodes, less than the threshold for forming consensus to validate malicious source IP addresses. In the Centralised CIDS, since the traffic analysed is aggregated centrally, attacks localised on one or more destination IP addresses on specific nodes are identified easily. However, our distributed approach flags suspicious IP addresses every first and the n^{th} multiple of when

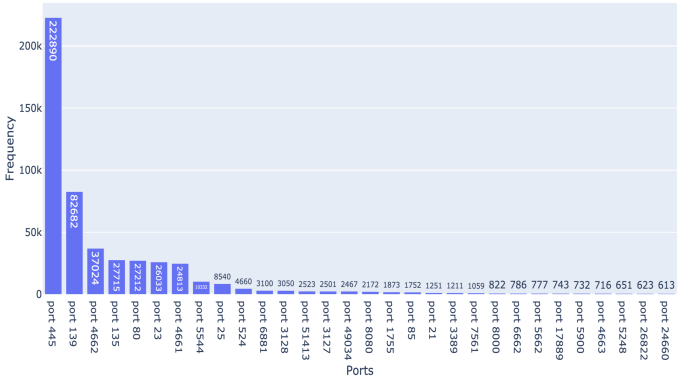

Fig. 3. Top Ports attacked as identified in Centralised CIDS (Threshold of 5)

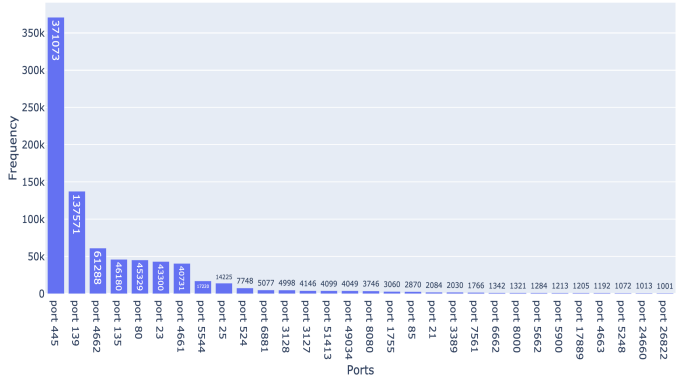

Fig. 4. Top Ports attacked as identified in Centralised CIDS (Threshold of 3)

it is observed on a specific edge node, but these source IPs are only considered to be malicious based on a consensus. Hence, attacks localised to a single or few nodes (less than the threshold for the consensus) remain undetected. This resulted in the identification of fewer alerts, thus affecting accuracy. To overcome this drawback, we need a mechanism to cater not only for attacks that are distributed/observed in different parts of the network, but also on localised edge nodes, while sharing context in the form of source IPs or attack patterns (port centric, etc) between peers.

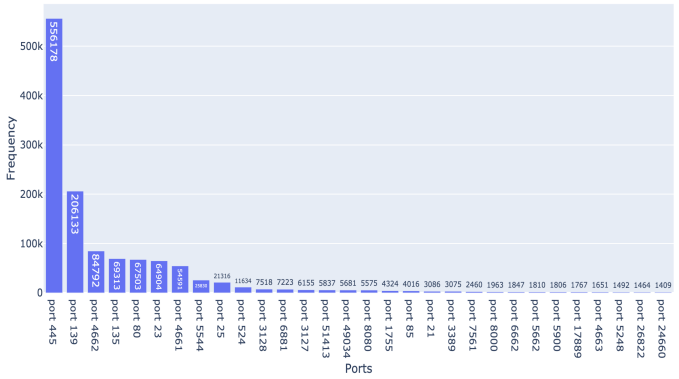

Fig. 5. Top Ports attacked as identified in Centralised CIDS (Threshold of 2)

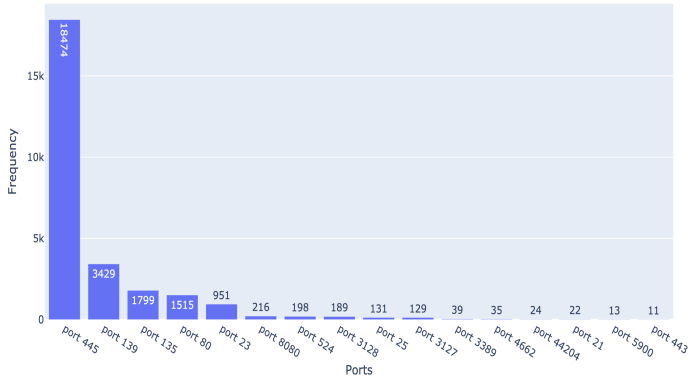

Fig. 6. Top Ports attacked as identified in Distributed CIDS (Threshold of 3)

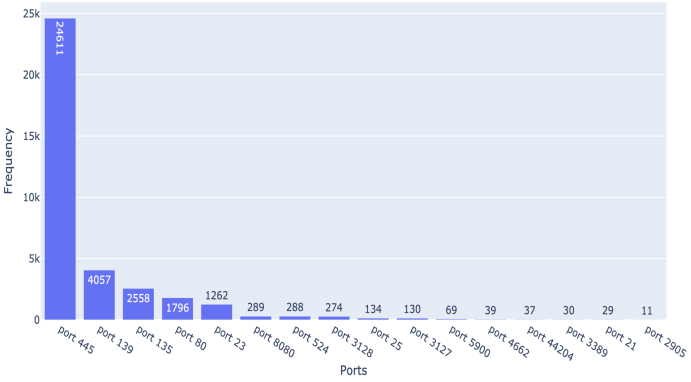

Fig. 7. Top Ports attacked as identified in Distributed CIDS (Threshold of 2)

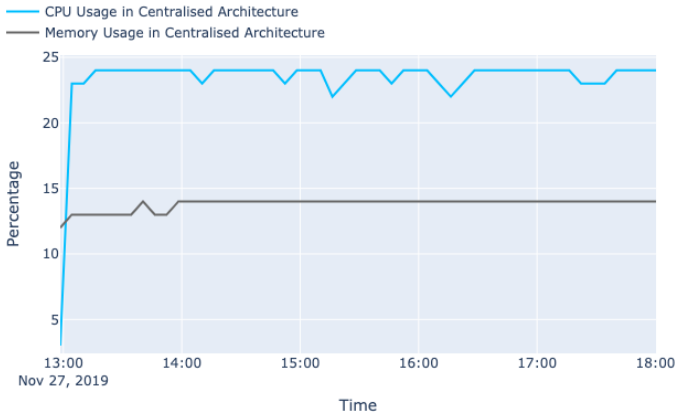

Fig. 8. CPU and Memory Usage in Centralised CIDS (Threshold of 5)

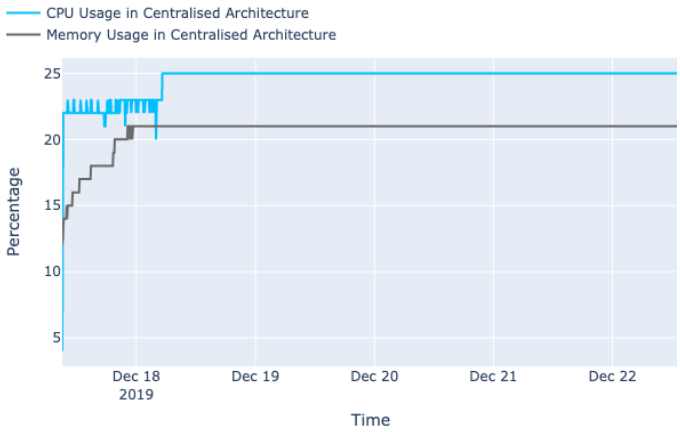

Fig. 9. CPU and Memory Usage in Centralised CIDS (Threshold of 3)

Similarly, we can observe that as the threshold value decreases for both the Centralised CIDS (from 5 to 3 and then 2) and the Distributed CIDS (from 3 to 2), the amount of traffic identified as malicious increases, providing finer granular detection ranges. However, this comes at a cost of computing memory and latency, which were not major issues in our setup given the small cluster size used.

F. CPU and Memory Requirements

For the Centralised CIDS architecture, we observe that the CPU requirements can vary considerably based on the threshold parameter set for confirming malicious traffic. For instance, the centralised node has a sharp increase in CPU and Memory usage initially and then becomes stable with few fluctuations depending on the processing needs of traffic crossing the set threshold of 5 in Fig 8. However, for thresholds of 3 and 2 we observe a higher level of fluctuations, indicating that as the threshold decreases, the CPU and Memory load needed for processing on the underlying host increases.

Similarly, the memory usage of the central node shows a small amount of fluctuation in the initial phase of the analysis, followed by a stable load throughout its processing lifetime. We also notice that the higher the threshold value used to map malicious traffic, the faster is the saturation point, as a threshold of 5 is more stable as opposed to threshold values of 3 and or (especially) 2. One would expect the memory usage to fluctuate further due to the queueing mechanism used for the staging and aggregation process. However, due to the limited amount of data forwarded by the 5 local monitor nodes and the faster processing capabilities present in modern architectures, the queueing of data was a trivial compute task, completing on an average 0.00017 msec per alert for a threshold of 5, in 0.00029 msec for a threshold of 3, and 0.00054 msec for a threshold of 2.

For our Distributed architecture, we capture CPU and Memory usage metrics across all our DHT nodes for threshold values 3 and 2. As seen in Figures 11-12 for a threshold value of 3, the CPU utilisation has a steep increase initially but reaches a saturation level at around 33-35%. Similarly, the memory utilisation displays an initial steep increase following which the usage across all nodes is stable. These metrics indicate that the resources across all edge nodes are evenly used with data processing and storage being offloaded in a similar fashion. However, for a threshold value of 2, we note that the CPU utilisation of our DHT nodes increases slowly until it reaches a saturation point due to the constant hashing of the single dimensional correlation queries invoked by the ingestion and retrieval actions of our data. Since we separate these actions into their own threads, the only compute

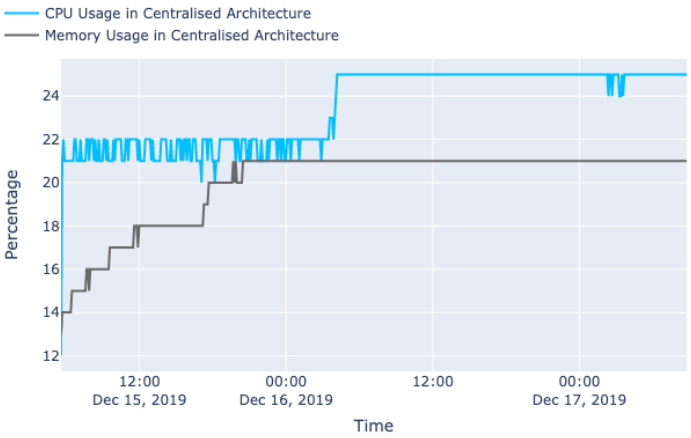

Fig. 10. CPU and Memory Usage in Centralised CIDS (Threshold of 2)

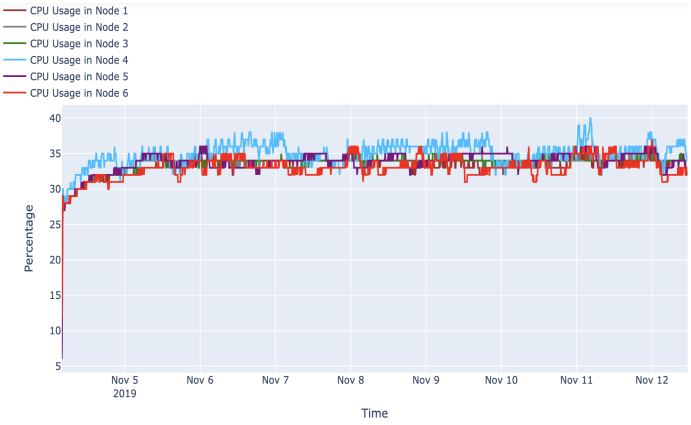

Fig. 11. CPU Usage in Distributed CIDS (Threshold of 3)

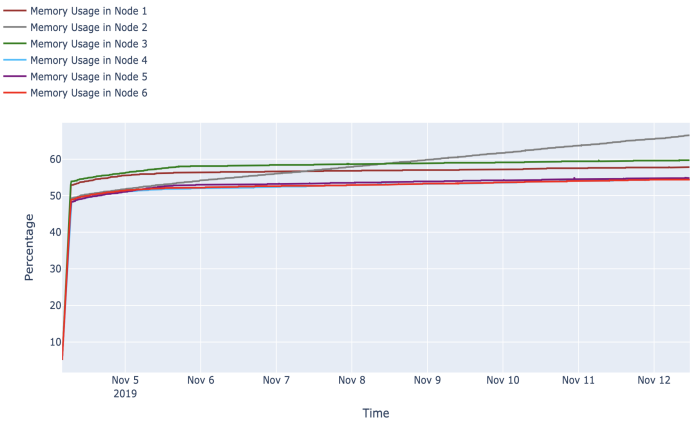

Fig. 12. Memory Usage in Distributed CIDS (Threshold of 3)

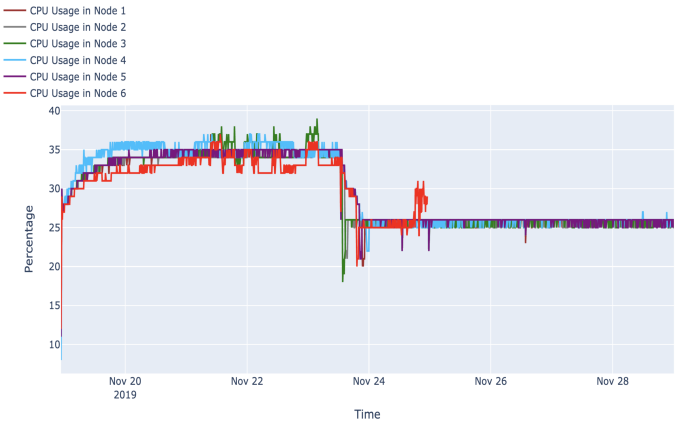

Fig. 13. CPU Usage in Distributed CIDS (Threshold of 2)

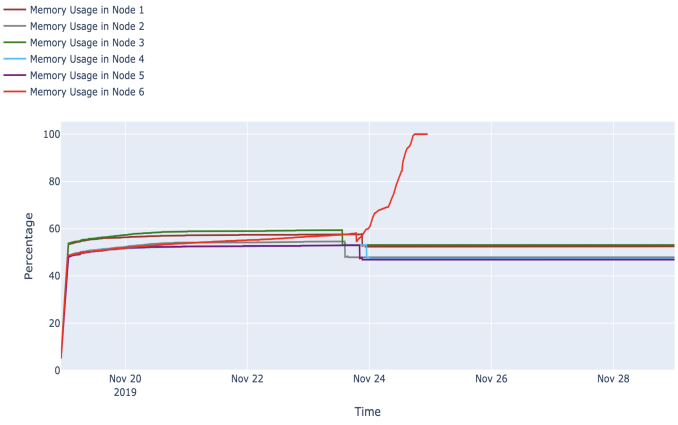

Fig. 14. Memory Usage in Distributed CIDS (Threshold of 2)

overhead involved on each node is the aggregation through queueing and the DHT based processing, which is trivial from a computing overhead perspective.

In contrast, the memory utilisation can vary quite significantly as observed in Figure 14. Although most nodes store data in a balanced fashion across the overlay network, we observe a spike in memory usage on node 6. This is due to constant updates being made on a particular source IP address, prompting the DHT to position the entry on the same node each time, resulting in additional memory usage. This could potentially affect heterogeneous edge nodes having less storage space by overwhelming them due to constant data storage.

Also, we observed that the data filtering and ingestion calls made to the DHT were completed in a few hours, but it took the DHT days to process, store and validate information, presenting additional overhead on resource utilisation. This presents a major bottleneck for the scalability of this distributed solution, as the amount of data that edge nodes process may be significantly more. This resulted in a steep increase in processing overhead and response latency for the DHTs.

In order to reflect a more realistic setup for modern net-

works, it is important to observe these metrics when adding additional nodes for both setups – Filtering nodes for the Centralised and DHT nodes for the Distributed approach, along with an increased data sample size, which we will aim to address in future work. Since the queuing mechanism in both approaches worked quickly with our current sample dataset, it would be interesting to observe how these architectures would behave under a heavier influx of data.

V. Conclusion & Future Work

Progression towards MEC in 5G still requires significant planning to detect and mitigate large scale coordinated attacks that disrupt service provisioning and usage. To understand the key complexities involved in detecting these attacks, we studied two CIDS architectures, namely a Centralised CIDS and a Distributed CIDS. Through experimentation with a real-world dataset, we observed that a Centralised CIDS displays higher detection accuracy than a Distributed CIDS, especially when attacks are concentrated on one or a few edge nodes. However, accuracy may be overwhelmed by response latency in a Centralised CIDS as the number of edge detection nodes increases. Similarly, the Centralised CIDS has lower CPU and memory requirements than the Distributed CIDS at low workloads, but increases gradually as the workload increases as observed at lower detection threshold levels. Also, the Centralised CIDS has higher network overhead than the Distributed CIDS, even at different thresholds. For future work, we will evaluate the tolerance of our architectures through stress tests, and consider how to combine the strengths of both mechanisms to optimize the detection process in a fault tolerant, decentralised manner, especially when attacks are focused on one or a few nodes in an edge computing environment.

References

[1] H. Hui, C. Zhou, X. An, and F. Lin, "A New Resource Allocation Mechanism for Security of Mobile Edge Computing System," in *IEEE Access*, 2019, pp. 116886-116899, Aug 2019.

[2] ETSI (2014). *Mobile-edge computing — Introductory technical white paper*. [Online] Available at: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1\%2018-09-14.pdf

[3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher and V. Young, "Mobile edge computing: A key technology towards 5G," in *ETSI white paper*, 2015.

[4] Google Cloud. (2019). *Edge TPU - Run Inference at the Edge* [Online] Available at: https://cloud.google.com/edge-tpu/.

[5] Microsoft Azure (2019). *IoT Edge — Microsoft Azure* [Online] Available at: https://azure.microsoft.com/en-in/services/iot-edge/.

[6] Amazon Web Services, Inc. (2019). *AWS Wavelength - Amazon Web Services*. [Online] Available at: https://aws.amazon.com/wavelength/.

[7] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in Proc. 26th USENIX Conference on Security Symposium, Vancouver, BC, Canada: USENIX Association, 2017, pp. 1093–1110., Aug 2017.

[8] S. Weagle (2017, February 21). *Financial Impact of Mirai DDoS Attack on Dyn Revealed in New Data*. [Online]. Available at: https://www.corero.com/blog/797-financial-impact-of-mirai-ddos-attack-on-dyn-revealed-in-new-data.html.

[9] F. Lin, Y. Zhou, X. An, I. You, and K. R. Choo, "Fair Resource Allocation in an Intrusion-Detection System for Edge Computing: Ensuring the Security of Internet of Things Devices," in *IEEE Consumer Electronics Magazine*, vol. 7, no. 6, pp. 45-50, Nov 2018.

[10] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges," in *Future Generation Computer Systems*, 2018, pp. 680-698, Jan 2018.

[11] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile Edge Computing: A Survey," in *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450-465, Sep 2017.

[12] K. Sha, T.A. Yang, W. Wei, and S. Davari, "A survey of edge computing based designs for IoT security," in *Digital Communications and Networks*, 2019.

[13] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Taxonomy and Survey of Collaborative Intrusion Detection," in *Journal of ACM Computing Surveys (CSUR)*, vol. 47, no. 4, 55, July 2015.

[14] The CAIDA UCSD "Three Days of Conficker Traffic from the UCSD Network Telescope" Dataset: http://www.caida.org/data/passive/telescope-3days-conficker_dataset.xml

[15] C. V. Zhou, C. Leckie, and S. Karunasekara, "A survey of coordinated attacks and collaborative intrusion detection," in *Computers & Security*, vol. 29, no. 1, pp. 124-140, Feb 2010.

[16] A. Blaise, M. Bouet, S. Secci, and V. Conan, "Split-and-Merge: Detecting Unknown Botnets," in *the Sixteenth IFIP/IEEE International Symposium on Integrated Network Management (IM), USA*, pp. 153-161, Apr 2019.

[17] C. V. Zhou, S. Karunasekara, and C. Leckie, "Evaluation of a Decentralized Architecture for Large Scale Collaborative Intrusion Detection," in *the Tenth IFIP/IEEE International Symposium on Integrated Network Management (IM), Germany*, pp. 80-89, May 2007.

[18] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, pp. 10-10, May 2004.

[19] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenkar, I., Stoica, and H. Yu, "OpenDHT: a public DHT service and its uses," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '05)*, pp. 73-84, Aug 2005.

[20] B. Iglewicz and D. Hoaglin, "How to detect and handle outliers," in *The ASQC Basic References in Quality Control: Statistical Techniques*. [Online]. Available at: https://hwbdocuments.env.nm.gov/Los\%20Alamos\%20National\%20Labs/TA\%2054/11587.pdf

[21] W. Li, Z. Liu, J. Li, and C. W. Probst, "Enhancing Intelligent Alarm Reduction for Distributed Intrusion Detection Systems for Edge Computing," in *W. Susilo and G. Yang (eds) Information Security and Privacy*, ACISP 2018, Lecture Notes in Computer Science, vol 10946, Springer, Cham, June 2018.

[22] Y. Wang, L. Xie, W. Li, W. Meng, and J. Li, "A Privacy-Preserving Framework for Collaborative Intrusion Detection Networks Through Fog Computing," in *S. Wen, W. Wu, A. Castiglione (Eds.), International Symposium on Cyberspace Safety and Security*, CSS 2017, Lecture Notes in Computer Science, vol 10581. Springer, Cham, Oct 2017.

[23] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," in *IEEE Communications Surveys Tutorials*, pp. 2322 - 2358, Aug 2017.

[24] Savoirfairelinux (2014). *savoirfairelinux/opendht*. [Online] Available at: https://github.com/savoirfairelinux/opendht.

[25] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Druschel P., Kaashoek F., Rowstron A. (eds) Peer-to-Peer Systems*, IPTPS 2002, Lecture Notes in Computer Science, vol 2429. Springer, Berlin, Heidelberg, Oct 2002.

[26] P. Porras, H, Saidi, V. Yegneswaran, "A foray into Conficker's logic and rendezvous points," in *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, pp. 7-7, Apr 2009.

[27] E. Aben (2019). *Conficker/Conflicker/Downadup as seen from the UCSD Network Telescope*. [Online] CAIDA. Available at: https://www.caida.org/research/security/ms08-067/conficker.xml.