Network Analysis through Edge Computing using Queries

by
Quangtri thai

A dissertation submitted to the Department of Computer Science,

College of Natural Sciences and Mathematics

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Science

Chair of Committee: Carlos Ordonez

Committee Member: Omprakash Gnawali

Committee Member: Driss Benhaddou

University of Houston
August 2021

ACKNOWLEDGMENTS

I would like to start off by thanking my advisor and committee chair, Dr. Carlos Ordonez, for his invaluable help and guidance through my undergraduate and graduate academic career at the University of Houston. I would also like to thank the committee members, Dr. Omprakash Gnawali and Dr. Driss Benhaddou, for taking the time to review and attend my presentation to offer their valuable feedback. I would like to again thank Dr. Carlos Ordonez and Dr. Omprakash Gnawali for their advice and help with my research, which made this all possible. To the peers who have been in my research group, I would like to give thanks to Xiantian Zhou, Sikder Tashin Al Amin, Steve Aigbe, and Siva Uday Sampreeth Chebolu for their help and feedback in the group. For the Department of Computer Science, University of Houston, I thank them for their help and acceptance. For my family who has been with me since the beginning, I would like to give thanks for their support and care for me over the years.

ABSTRACT

Monitoring networks requires two things, efficiently detecting abnormal events and summarizing connection information in big volumes of packet-level data. Some of these tasks can be accomplished with network and operating system utilities, but the questions should be relatively simple and each tool is designed to provide specific analysis. Being able to process data both in a centralized and decentralized manner, given the diversity in instrumentation and vantage points is also another requirement to monitoring the network. On the other hand, database systems can answer complex questions phrased as queries, provided data is in the right format and is quickly loaded. Having such motivation in mind, we propose to monitor a network with queries, running on a traditional DBMS (i.e. not a custom-built system programmed in C or C++). Thus, queries can be processed in a central manner in a traditional database server or in a distributed fashion, with edge computing. Our experimental evaluation shows queries can indeed be used to monitor the network with low latency and reasonable delay on a low-resource device like the Raspberry Pi. We explain some interesting findings in a local network. In addition, we show queries can be efficiently evaluated in a small computing device capturing local traffic.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

1.1 Motivation

Over the last decade, researchers have increasingly adopted the use of testbeds to bring realism in their network experiments or used instrumentation in real networks to understand the network packet patterns. The testbeds allow researchers to capture a large volume of somewhat realistic data traces from the network, but many of these testbeds do not provide a powerful, flexible, and practical network data analysis tool. Such lack of tools have led the researchers to using either basic network diagnostics tools (e.g., derivatives of ping, traceroute) or roll out their own custom tools leading to inefficiency in testbed based wireless networking research.

The availability of testbeds and instrumentation data from realistic environments are evidence of progress the research community has made, but we also observe a general lack of standard and flexible data analysis capability to fully take advantage of these powerful testbeds. For example, tools such as ping and traceroute are adequate for certain near-real time network data analysis, but these tools may find historical data processing a bit harder and is often not as flexible apart from their given task. While there are more sophisticated tools developed by researchers [15, 34], they are unable to provide a flexible way to allow custom, real-time queries that the user may want to run.

Our solution to this problem is informed by two principles. First, leverage the common architecture used by these testbeds and networks to export network packet data to the researchers. Second, rather than reinvent yet another data processing framework to process the packet data, leverage existing technologies that are known to be effective in other domains with similar data properties.

Combining the use of SQL to store and analyze network streams from a tool like Tshark is a fast and flexible option when compared to the standard tools provided by Linux, allowing for historical data processing. Using SQL will provide more custom and complex queries, while keeping it simple, flexible, and scalable [24]. Streams will be captured in small batches with Tshark, then added and analyzed in Postgres with SQL to give a historical analysis of the data. While using custom Python/R code for data analysis can provide further specialized queries, this comes with the price of being much harder to implement and is much more of a hassle to change and specialize to a different task. Our solution will allow more interactive and flexible exploration of network data by researchers. Furthermore, our evaluation shows that it is feasible to run this system on modern edge devices like the Raspberry Pi thereby allowing the researchers to use the same analytical tools on the testbed nodes or on their desktops/laptops leading to significant reuse of analysis code.

1.2 Our Contribution

We study how viable it is to monitor and analyze a network stream through the use of SQL in both a centralized and distributed manner. This involves tackling two of the five V's in big data, volume and velocity. We then study the feasibility of implementing a query system directly on the edge device, such as the Raspberry Pi, and run it reasonably efficiently for both near real-time and historical data analysis. The edge device will bring along complications from its limited hardware, causing possible inconsistencies or limitations. We will look into how to overcome these complications as we move forward. The network stream will be continuously captured and processed on the Raspberry Pi, while measuring how long each step takes. We analyzed the captured data using the tools we developed and measured the processing latency to see how viable it is on a low-resource device such as the Raspberry Pi. We also try to understand how well the system scales with increasing stream size and devices.

Our contributions are: (1) Modeling the network instrumentation as streaming data processing using SQL. (2) Formulating different streaming queries that are relevant for network monitoring. (3) Implementing and evaluating a light-weight DB and SQL-based network instrumentation system on an edge device and overcoming the complications that arise from using a low-resource device.

1.3 Thesis Organization

We will discuss the different literature and research that revolves around monitoring the network in Chapter 2. Chapter 3 will go over the definitions and background of network monitoring. In Chapter 4, we will specify the queries used and go into detail their uses and benefits. We will also look at how the network stream is processed and loaded on the device and the architectures used. Chapter 5 will focus on setting up and running these queries and experiments to examine their run time and performance on different devices and using both summarized and non-summarized streams. After discussing the experiments, Chapter 6 will conclude our findings.

2 Related Work

In this chapter, we will talk about the research done to monitor the network. The different ways the packets are captured from the network to understand the benefits provided from each method. We will also look at research done with different system architectures, giving an idea of the assistance and problems they pose. Since we are using a database to analyze our network, we will also look at some network monitoring research also done using database systems.

2.1 Network Monitoring

The motive of network monitoring is to detect potential problems from slow or failing components or from an attack on the network itself to make sure the network is operating as intended [21]. Traditionally, Linux tools such as ping, traceroute, netstat, and iperf are used to diagnose or understand the network performance. Researchers have developed additional tools such as tcpdump and more sophisticated tools for deep packet inspection [11]. The huge volume of data produced by the network is a huge challenge when monitoring the network. Regular network traffic is already large and attacks on the network can cause a greater influx of data on the network. The job of the network monitor is to capture and analyze this data as efficiently and accurately as possible, as such, systems like Sonata has been designed to capture large amounts of this traffic, 95%, and reduce the overall data rate by factors of about 400 [15]. Sonata does this by taking advantage of the advances made in programmable switches to capture subsets of the traffic needed through a set of rules and advances in streaming data analysis to more efficiently process these streams. To monitor these networks, you often have to create network policies to enforce certain quantitative rules on the network. As such, Declarative languages and systems [34, 6] have been built to make it easier for network engineers and researchers to query the network system data. For example, NetQRE looks to simplify this by integrating regular-expression-like pattern matching with aggregation operations to specify and implement quantitative network policies [34]. Research has also been done in reducing these large data volumes to not only analyze them, but also the store then for future analysis. As such, building a data synopsis has also been a topic of research that focuses on allowing researchers to access and query past data days to years in the past [9]. Utilizing the cloud to create a scalable network monitoring system is also a recent subject that looks to give a cost-effective approach that has a low learning curve [7].

2.2 Streaming Data Analysis

2.2.1 Packet Analysis

Packet analysis allows a more in-depth look into the network by capturing and interpreting network traffic to understand parts of the network [28]. This often involves using a packet sniffer to capture and store the packets as they move through the network to then perform analysis, which can be a computationally heavy task. To reduce this load, researchers have looked into many different methods to deal with this load, including methods like the exploitation of multi-core systems [10]. Packet analysis is used by many researchers to monitor and analyze the network [27, 5, 1]. This can be used for a variety of things, such as defending against cyber warfare at a national level, defending the public against cyber attacks, or to discover problems with network performance [27, 5, 1].

2.2.2 Flow Analysis

Recently, more research has been done on flow monitoring [16], a prevalent method used to monitor high-speed traffic using protocols such as NetFlow and IPFIX. This method involves analyzing flows rather than streams of packets, leading to much lower overhead. To do this, “packets are aggregated into flows” where each flow is defined by “a set of IP packets” that contains “a set of common properties” [16]. This benefit comes at the cost of losing some detail from the packet, but allows the large volume of data to become easier to manage and the loss of detail can be offset by performing packet analysis alongside flow analysis. Use of flow analysis can be seen in many systems that monitor the network, such as intrusion detection systems (IDS) called flow-based intrusion detection [31, 17, 29, 25]. IDS is a tool used to protect IP networks by analyzing “network traffic and system logs to detect an attack,” however this method is computationally costly and can become a bottleneck in high-speed IP networks [31]. With flow-based intrusion detection, this network load can be greatly reduced into network flow records that can be inputted and analyzed in a more efficient manner [31].

2.3 Database Systems applied in Network Monitoring

Researchers have developed more sophisticated tools for network monitoring over the last decades. For example, researchers have designed a new relational database to ingest network data stream [26]. The authors claimed they could achieve throughput up to 50% of the throughput of the hard drive itself. In order to achieve this speed, the system must sacrifice consistency and durability characteristic of a DBMS. Stream databases like GigaScope [8] has also been developed for network applications to store and analyze their traffic, but it has many limitations with storing streaming data, not taking advantage of the parallel file system, and not being able to correlate streaming data with stored historical data. Overtime, as sorting summarized historical stream data and supporting standard SQL became needed, a new system was needed. In this system, queries would have arbitrary joins (natural, outer, time band) and diverse aggregations (distributive, algebraic, holistic). While storing, managing, and querying stream data is more difficult than analyzing packet-level data, it enables advanced analytics to monitor the network. To fulfill these requirements, the DataDepot Warehouse system [12] was created. The DataDepot featured a POSIX-compliant parallel file system, standard SQL, and extensibility via UDFs [23, 22] (which enabled mathematical analytics). Given the common wisdom that one-size-does-not-fit-all [30] and the difficulty of changing the source code of a large existing system, TidalRace [18] was developed to support new and more demanding networking monitoring and maintenance applications.

3 Background

3.1 OSI Model and Packet Layers

Alani, 2014 & Kumar et al., 2014 explains that the Open System Interconnection (OSI) model is composed of 7 layers [3, 19]. This model consists of the application, presentation, session, transport, network, data link, and physical layer. These layers help to characterize and standardize the communications between devices, allowing network-aware devices to communicate with each other. At the lowest layer is the physical layer where strings of ones and zeros are transmitted and received through a physical medium such as a wire or NIC cards. Layer 2, the data link layer, defines the data transfer between two adjacent nodes. Layer 3, the network layer, handles routing data sequences (datagrams) from one network to another. This layer will manage, transfer, and reassembly the data, possibly routing the data in fragments and reporting the delivery errors. Layer 4, the transport layer, is a layer that is similar to the data link layer. This layer also provides a way for different machines to communicate with each other, but also maintains a certain level of reliability. This reliability is gained from powerful routing protocols that use or detect acknowledgements, packet drops, misrouting, re-transmission, etc. Layer 5 is the session layer where two machines can hold a session across the network where data on either end can be exchanged for as long as the session lasts. For layer 6, the presentation layer is where the data is packed or unpacked for the application. As Kumar et al. explains, data conversion, compression, translation, and encryption can occur in this layer. For layer 7, the application layer, this layer is the closest to the user where the data is shown to the end-user as a website, program, etc. This layer also contains many protocols such as HTTP, SSH, etc.

Figure 1 shows the typical layers of a packet which are data link layer frames [4]. Each layer acts as a filter to get the packet from one destination to another, whether it is traveling

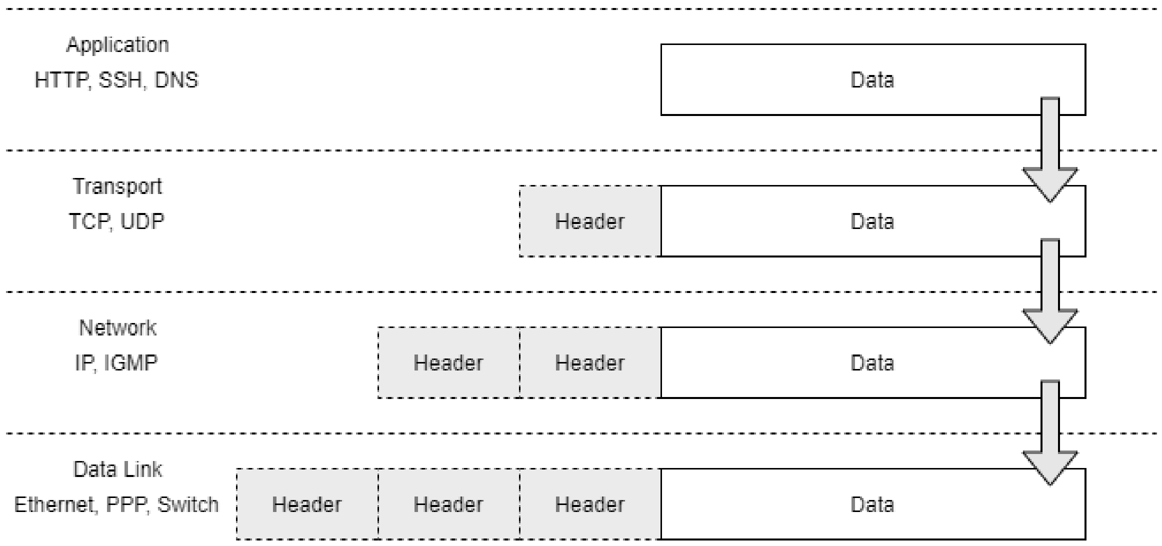

Figure 1: Packet.

through the physical layer or getting pushed up to the application layer. The headers assigned for each layer will contain info such as addresses, ports, flags, etc. to help the packet get to its destination. Knowing this, we can extract useful information from the packet layers to allow us to monitor the network.

3.2 Edge Devices

Edge devices process data close to the edge, the source where data is produced or consumed. With the introduction of 5G wireless and deployment of Internet of Things (IoT) devices, using edge devices to compute or analyze the data close to the edge is becoming a more appealing approach. This is because there are many benefits provided by processing the data near the source instead of through the cloud. For one, edge devices would allow for quick data processing and storing that can translate to more efficient real-time applications. Another reason would be the limited bandwidth posed by using the cloud as high bandwidth usage can become expensive. Edge devices would allow for better bandwidth as the data would not be travelling as far and only relevant or processed data would be sent to the cloud. This would give edge devices a role in on-demand and real-time applications, working to overcome the problems found in cloud computing. Because of the bandwidth limitation of cloud computing from problems such as real-time video analytics, research has been done to make use of the edge to overcome these problems. One such system is the Latency Aware Video Edge Analytics (LAVEA) system where tasks are either done locally in the device or offloaded remotely to an edge computing node [33].

3.3 Database

As Foster, 2014 explains, a database is a collection of data stored in a computer system. It is explained that these databases can be further managed by a database management system (DBMS). The DBMS consist of software used to store, update, retrieve, as well as other functions used to manage the database. Along with these critical functions are primary and secondary objectives that a DBMS aims to achieve. This would include objectives like security, reliability, flexibility, consistency, integrity controls, etc. These objectives for the DBMS bring along many advantages such as reduced redundancy and inconsistencies, integrity, improved performance, etc. We choose to use a DBMS to manage and store our data because of these advantages provided and supported by the DBMS. We needed a system that would maintain data integrity, have good performance, is flexible, and is easy to maintain and retrieve data from.

3.4 Packet Sniffing

Network monitoring involves tracking and monitoring various aspects of the network to better understand and analyze the network and its operations. As Asrodia and Patel, 2012 explained, packet sniffing is a network monitoring technique used to monitor every packet that passes through the network. To do this a packet sniffer is used which is a “piece of software or hardware that monitors all network traffic” [4]. There are many packet sniffers available for use, with each focusing on different aspects of the network [4]. This would include sniffers like Wireshark, Tcpdump, Etherape, Capsa, etc. [4]. Wireshark is a popular tool used for network troubleshooting with features that allow users to view traffic on a specific interface, capture packets, and filter and analyze the network traffic [4, 13]. It has a user friendly GUI, but does not have a detection system to alert strange activities [4]. On the other hand, Tcpdump is a lightweight analyzer that “runs under the command line” to “intercept and display TCP/IP and other packets being transmitted or received over a network” in a raw format [4, 13]. It can be used to analyze various parts of the network to help the user isolate the problem or display “communications of another user or computer” [4]. This goes to show that packet sniffers can focus on different aspects of the network and have different goals for the application of the software or hardware. With this in mind, we choose to use Wireshark for its ability to capture the network, along with the many features provided by Wireshark. As the network grows, more research is done to improve network monitoring performance and new techniques are created to monitor the network [2, 14]. Al-Yaseen et al., 2016 improved the performance of a real-time intrusion detection system (IDS) by using a Multi-agent System (MAS-IDS) to reduce processing time by 81% [2]. This reduction is from dividing the traffic data among a set of hosts (computers) where each host would create agents to process the data subsets in parallel [2].

3.5 Network Data Stream

We first describe our model of data stream as it pertains to network monitoring. A stream is an unbounded timestamped sequence of data points $< ts, t >$, where ts is the timestamp and t is the tuple of data [32]. As such, the stream will give us at least a data point of

$< ts, src, src\_port, dst, dst\_port, protocol, len, info >$ to sufficiently answer the queries used in Chapter 5. This data point will be obtained from filtering the stream using Tshark, then processed and stored in a database where the queries can make use of the data. The data can then be further summarized, allowing us to improve the efficiency of these queries.

3.6 Basic Queries

There are two types of queries to analyze stream data: *one-time queries* and *continuous queries* (materialized views). The *one-time queries* are the queries being used in traditional DBMS, where the query is computed from scratch every time. *Continuous queries* are used in modern stream database systems that update the results as new data arrives. In our experiments, we use continuous queries to replicate real situations.

As shown in Figure 3, Specific data will be extracted and queried from the stream using the Raspberry Pi. This would include timestamps, source, destination, length, and info. From these fields, we can obtain various info on the stream by grouping, summarizing, and joining them in different ways.

Time window, Figure 2: In a data stream, data continuously arrives as the old data is still being processed. Hence a time window is required on every query to purge the records falling outside the range. It is worth noting that once the data resides within the database, it can be queried multiple times on different time windows. In other words, data is loaded once and queried multiple times.

A time window is a concept to help us understand the range of data being viewed. It represents a duration of time that we can use to understand the range of data being analyzed and stored in our database. As such the time window can be adjusted to a wide range of values to fit your needs.

Figure 3 shows an example of how the Raspberry Pi will monitor the network on a 10

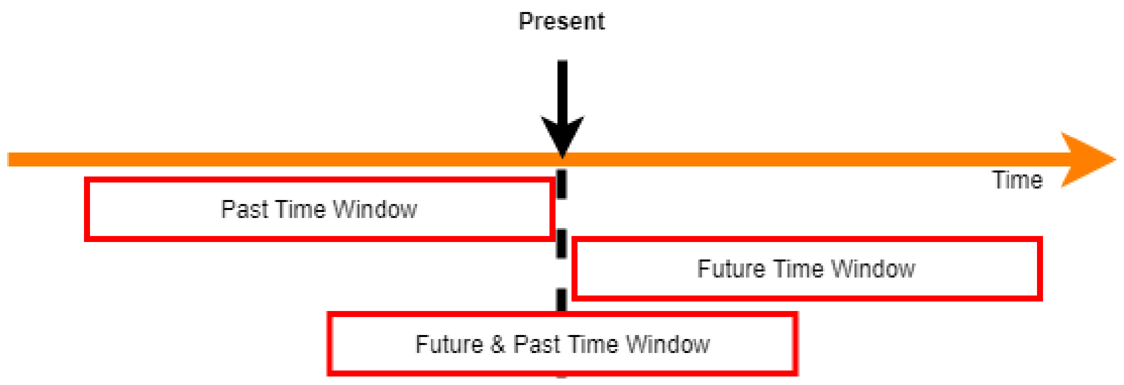

Figure 2: Time Window.

minute time window, capturing packets traveling between devices and routers. Tshark will be the tool used to capture these packets with Postgres being used to store and analyze these streams.

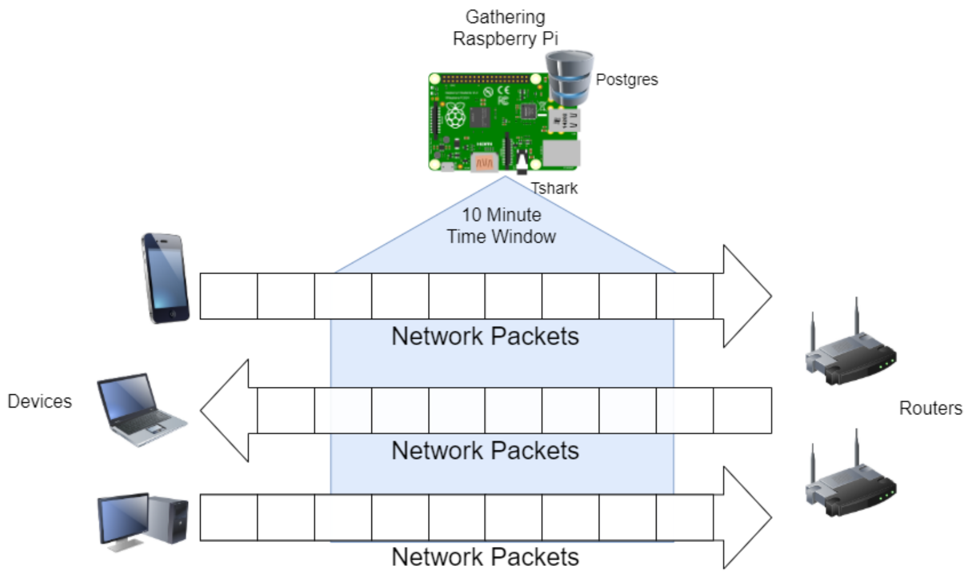

Figure 3: Monitoring.

4 Network Streams Processing Insight

4.1 Approaches to Processing Data Streams

There are two approaches to processing data streams inside a DBMS. One approach is to continuously load and process a stream, while the other is to periodically load and summarize a stream in batches. By summarizing before you process a stream, you can reduce the data size greatly at the cost of a bit of overhead.

4.1.1 Continuous Processing

Continuously processing a stream involves keeping every individual packet, demanding greater storage size and keeping more details of the stream. The greater volume imposed may exceed the RAM capacity and often contains redundant data. However, having greater detail of the stream will allow you to dig deeper into the stream to find patterns and information hidden by being summarized.

4.1.2 Summarizing in Batches

Continuously loading a stream uses a lot of space and is inefficient, as a lot of the data is redundant from a network stream. This large amount of data will lead to slower analysis time when accessing and filtering through the data. This is especially apparent in a small device like the Raspberry Pi, that would have more trouble with larger data sets from its limited components when compared to larger systems. As such, it makes sense to summarize the data after loading it into the database. This will lead to a striking decrease in data size, while retaining important aspects of the data, although losing a bit of detail as a result. This will not only lessen the load on the Raspberry Pi, but greatly cut the time it takes to analyze the data.

4.2 Pre-processing and Loading Data

4.2.1 Pre-processing

In general, data streams have significant redundancy. Stream data sets have the form of a log file where records may have variable length. To accelerate processing, we truncate string columns with variable length such that 90% of information is preserved. On the other hand, contiguous records that have the same information (e.g. network packets) are aggregated into one record. Finally, we project important attributes for analysis, leaving out detailed information (e.g. network packet content). Pre-processing is a necessary step to allow the DBMS to store the packet into a relational table.

4.2.2 Batch Loading

Loading data is a time consuming task for two reasons: the input text file needs to be parsed and data needs to be transformed and stored into a specific binary format on disk. Hence, catching data records and loading them in batches yields better performance as the SQL statement is parsed and optimized once for the whole batch. However, this introduces a delay from the time the stream data arrives to the time it becomes available in the database. In stream processing, a lower latency is better. There is an I/O bottleneck introduced in this step where the data is read from the text file then written to disk. We want to continuously update the DBMS with new batches of data so that the DBMS stays close to real time. Here we update the DBMS with many small batches instead of one giant batch of data. This method has a lower performance than loading large batches, since the SQL statement is parsed and optimized multiple times, but benefits from keeping the data up to date.

4.3 Summarizing the Data

After the data is loaded into the DBMS, it is summarized using a group by on the src, dst, and protocol of the packet from a temporary table. This method is called flow analysis where a set of identifiers identifies a flow and a new flow is defined when an identifier is changed. The flows are stored at set time intervals.

Summarizing query:

```
\begin{verbatim}
INSERT INTO L1(min_ts, max_ts, src, src_port, dst, dst_port, protocol,
min_len, max_len, avg_len, summ_size)
SELECT
  min(ts), max(ts),
  src, src_port,
  dst, dst_port,
  protocol,
  min(len), max(len), avg(len),
  count(*)
FROM L2
GROUP BY src, src_port, dst, dst_port, protocol;
```

4.4 Centralized and Distributed Processing

Centralized processing involves moving the data and processing the stream through one device. This can be intensive for one device to handle. The benefit of this method comes from its simple implementation, only needing to worry about one device. However, this does not cover a wide area and would have more trouble analyzing large amounts data. To help alleviate this, techniques can be used to reduce the load such as summarizing the data to help accelerate data ingestion.

Distributed processing can be used to further reduce this load and cover a wider area, but is a bit more complicated to implement. This method would require an addition step of merging the data, allowing for the heavy task of processing and collecting to be split between multiple devices. These devices would capture and process the stream, send and merge the data to a single device that can then be monitored and analyzed.

We plan to use a distributed architecture because our devices will not have as much processing power individually, but together should allow for more complex analysis. We do this by having the DBMS running on all Raspberry Pis with each Raspberry Pi monitoring a partition of the network as shown in Figure 4. The packets will be summarized at each Raspberry Pi, then sent downstream to the monitoring device to eliminate bottlenecks. For now, experiments will be run on a single device to test the feasibility of a distributed system.

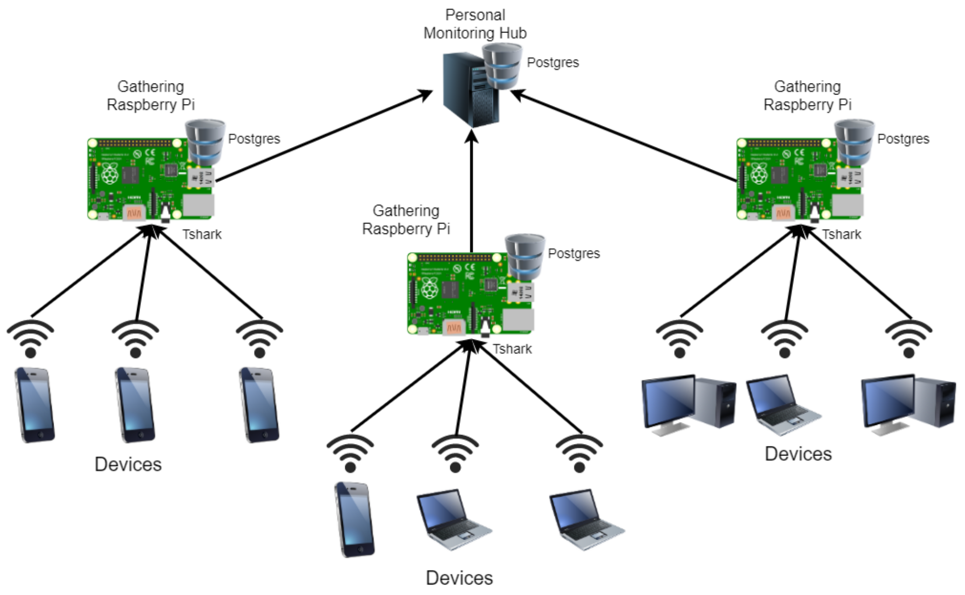

Figure 4: Architecture.

5 Monitoring Network Streams with Queries

We seek to monitor and analyze the network using SQL commands on the edge devices. We follow this approach because of the many advantages and flexibility SQL would offer. This would include practically unlimited storage, flexibility in querying, and scalability. Relational databases, like Postgres, have built-in mechanisms for keeping data integrity to eliminate duplicate data and allow more flexible queries to be carried out. While languages like Python/R can meet these needs, it comes with the price of increased time and complexity to implement and change these features to our needs. As such we look at SQL to compress the data stream, summarize the data, and analyze the data stream. SQL allows the user to answer many complex questions with little programming effort, providing insight not available in traditional network tools and OS commands.

5.1 Query Processing

Query processing is the process of interpreting or translating higher level SQL command into lower level programming language. It is the process a query goes through to extract data from a database. This would include being scanned, parsed, and validated; then translated into a query tree with many options for execution strategies. This process is further optimized through a process called *query optimization* where the DBMS finds the most efficient execution strategy. As such, we believe that using queries is a flexible and efficient approach to processing our data.

5.1.1 Understanding Traffic Between Source and Destination

The query with group by aggregation will be used to understand the traffic passing through our data stream. Our query groups the source and destination to give us a count of the packets passing through the stream between a set time window. This can easily be modified to give us other information about the stream.

```
SELECT src,dst,aggr()
FROM L
WHERE start<=ts AND ts<end
GROUP BY src,dst;
```

Since the data is summarized so that each row may contain info for a range of packets and as such a range of timestamps. The query will be changed to take into account this timestamp range. Instead of timestamp being a single point, ts, it will be a range, min_ts to max_ts. As such, there would be a slight change in the WHERE clause, so that the query will only consider data strictly in the time window, start to end. The WHERE clause would simply be changed to: WHERE start<=max_ts AND min_ts<end

src, *dst* is a set of columns to be grouped because we want to understand the traffic between src-dst pairs.

5.1.2 Query for Co-ocurring Events

Co-ocurring events are a good way to discover distinct or peculiar events that happen in the network. Say for example you want to find when there are more than 10 streams running in the network. To do this you will first need to group the streams then count them. Our query does this using a band join [18] and an aggregate function. Our query builds a temporary table L1 and L2 from L, then performs a band join on both tables with a time window between $[start, end)$. From there we can use an aggregation to give us more information about the streams. This is shown as aggr() and can represent sum(), max(), count(), or any other aggregate function.

```
SELECT L1.src,L1.dst,aggr()
FROM L AS L1 INNER JOIN L AS L2
  ON L1.ts-c<=L2.ts AND L2.ts<=L1.ts+c
WHERE start<=L1.ts AND L1.ts<end
  AND start<=L2.ts AND L2.ts<end
GROUP BY L1.src,L1.dst;
```

With the summarized data, each row will again have a time range from min_ts to max_ts. The query will perform a join with the data where the timestamps are strictly within a bound, start to end. By using band joins, we can determine what happens around the same time between multiple streams. This can be a complicated task, but is made easy using SQL.

5.1.3 Query for Session Duration

Session duration is useful to know because it tells us how long a user may use a particular website or service. It may also be useful to discover network failure or network attacks because session lengths corresponding to those traffic may be different from session duration for regular application traffic. The query works by simply selecting a stream's earliest and latest timestamp. The stream is filtered by using src, dst from table L.

```
SELECT src,src_port,dst,max(ts)-min(ts)
FROM L
WHERE start<=ts AND ts<end
GROUP BY src,src_port,dst;
```

5.1.4 Query for Request/Response Protocol

The request/response protocol is a useful protocol that can be observed to find network failure, checking if there is a response to all requests or vice versa. The query works by selecting all rows that does not have a response, represented as the inverse of src, dst.

```
WITH CTE AS (
  SELECT src,dst
  FROM L
  WHERE start<=ts AND ts<end)
SELECT src,dst
FROM CTE L1
WHERE NOT EXISTS (
  SELECT 1
  FROM CTE L2
  WHERE L1.src=L2.dst AND L1.dst=L2.src);
```

It is also important to note that this protocol can also be done using an outer join, but will be less efficient as the above solution will be using a common table expression(CTE), a temporary result set that can be used multiple times in the query.

6 Experimental Evaluation

6.1 System, Software, and Hardware

We implement our system on the Raspberry Pi 3, a single board computer with CPU ARM7 Quad-Core 900MHz, 1GB RAM, and 64GB SD Card. These devices are deployed in many wireless testbeds today and represent a low-cost and low-resource node that has enabled scaling of many wireless testbeds. A DBMS server with CPU Intel Pentium N3700 1.6GHz, 8GB RAM, and 2TB HDD was also used to implement and compare with the Raspberry Pi. In terms of software, we installed Postgres and T-shark, a program utilizing the on-board network interface to capture network packets.

6.2 Network Data Collection, Pre-processing, and Summarization

An instance of T-shark (a packet monitoring program) is run on the micro-computer (Raspberry Pi) to capture WIFI packets in a busy common room. As the T-shark program captures a large number of fields from a packet, for simplicity, we ignore many of those fields while retaining the essence of streaming data processing for network monitoring. As a result, these columns are chosen: timestamp, source address, source port, destination address, destination port, size of the packet in bytes, protocol, and extra information that may contain important information about the packet. These packets are then summarized using group by on source, source port, destination, destination port and protocol. The summarized table rows contain columns on minimum timestamp, maximum timestamp, source, source port, destination, destination port, protocol, minimum packet length, maximum packet length, average packet length, and the number of rows summarized for each group queried by the group by.

Column's Name	Type
min_timestamp	DOUBLE PRECISION
max_timestamp	DOUBLE PRECISION
source	TEXT
source_port	INTEGER
destination	TEXT
destination_port	INTEGER
protocol	TEXT
min_length	INTEGER
max_length	INTEGER
avg_length	NUMERIC
summ_size	INTEGER

Table 1: Summarized Data Schema.

6.3 DBMS-based Analysis system Performance

To understand how well these processes perform on the Raspberry Pi, we first look at how much data is being captured at various time windows. To do this, we simply capture a large amount of data, then count how many rows of data there are at various time windows using a query.

Time Window(secs)	Total Stream Records
10	6940
20	13923
30	20863
60	41311
120	86723
1800	1595385

Table 2: Non-Summarized Records.

For the summarized data in Table 3, from a different stream, we recorded the total records at different time windows as well as how much those records were compressed in the summarized table.

Time Window(mins)	Total Stream Records	Total Summarized Records
1	10152	513
5	57774	2998
10	120837	6161
15	180471	9154
20	243212	12163
25	307717	15400
30	360470	18343
40	474112	24408
50	585721	30537
60	693976	36591
240	2800527	147096

Table 3: Summarized Records.

Notice in Table 3 how much compression greatly reduces the data, with some reaching up to 19 times reduction for smaller time windows. With a greatly reduced data set, we hope to see this reflected in our experiments.

6.3.1 Processing

With a general idea of how many records there are at each time window, we look at how long it takes to process and store these records. Measuring these times will give us an idea of how well these same experiments will perform on live data and what kind of delay we can expect as we monitor the network.

At each step of the process in Table 4, we measure the time it takes to complete. The total rows represents how many rows of data was captured in that interval and summarized rows represents how many rows it got compressed down to. From the Table, we can determine that the bulk of the time is used in converting from binary to CSV. Converting is the job of the networking tool and not the DBMS, as such it leaves a lot of room to improve on, greatly lowering the delay required to monitor the network if optimized. The DBMS on the

	Time(secs)					
Time Window	1	5	10	15	30	60
Bin to CSV	1.44	1.96	2.36	2.93	7.52	10.77
Pre-process	0.01	0.03	0.03	0.06	0.17	0.33
Load	0.04	0.07	0.08	0.09	0.34	1.82
Summarizing	0.02	0.04	0.04	0.05	0.08	0.18
Total rows	165	84	1139	1967	5777	11518
Summ rows	30	12	143	178	375	601

Table 4: Summarized Processing.

other hand was able to quickly load and summarize the data. Summarizing can reduce the rows by a lot because there can be a lot of redundant data in streams. This can translate to performing queries hundreds of times faster than on non-summarized data.

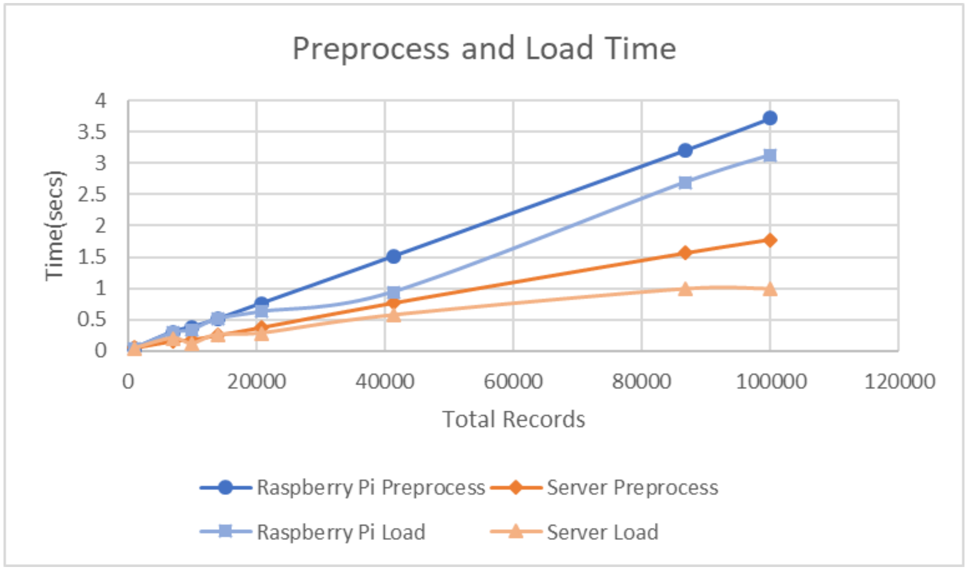

Figure 5: Non-Summ Preprocess and Load Time.

In Figure 5, we do a similar test but on both the Raspberry Pi and Server, comparing the two. From the Figure, we can see that the Raspberry Pi times are still linear, but lags a bit behind the Server's speed.

6.3.2 Queries

Next, we will look at how well the queries performed on both the summarized and non-summarized data streams. The queries we will be looking at are group by and band join for summarized data and additionally session duration and request/response protocol for non-summarized data. Quantiles is also another query we can look at in future experiments, but is currently too slow and complicated to be implemented. By using a DBMS to query and analyze your data, you can flexibly change your queries to your need without having to make major changes to your program. These tests are done on static data and on a single Raspberry Pi/Server, but should give us an idea on how they will perform in real-time and how well they will do in a distributed fashion.

Time Window(mins)	Total Stream Records	Time(secs)
1	10152	2.65
5	57774	2.78
10	120837	2.93
15	180471	3.10

Table 5: Group By on Non-Summarized Data.

Time Window(mins)	Total Stream Records	Time(secs)
1	513	0.17
5	2998	0.14
10	6161	0.15
15	9154	0.16

Table 6: Group By on Summarized Data.

As you can see in Table 6, group by finished in well under a second. The query also scales well as the number of rows queried increases, still staying under a second with a 15 minute time window. When compared to the times measured in Table 5, it still outperforms even when compared to a lower row count. This is because of how compressed the data was in the summarized table, allowing for faster processing times.

In Figure 6 and 7, a band join was done on two selected close streams from captured static data. The band join was done on a variety of band ranges and time windows. Timestamps are very precise measurements, as such we use Band Ranges to represent how much tolerance we gave for each data point when joining the data. This way, if data points are close enough within the tolerance margin, they will be considered to happen at the same time and joined in the query. Depending on these ranges, the band join can perform reasonably well at lower time windows, but quickly escalates as you widen the time window. On the other hand, increasing the band range did not affect the times as much, except on wider time windows. When comparing the two figures, you will notice that the time to complete the queries were significantly less for higher time windows and band ranges. Band ranges also have a bigger effect on the summarized data when compared to non-summarized data, being a major factor to the time spent to finish a query. These results show that it is possible to perform heavy queries on these devices, but it can take a long time.

In Figure 8, we compare some simple queries on both the Raspberry Pi and Server with non-summarized data. For these queries that are much less intensive than band join, they were able to perform much faster and had a more linear increase in time. As such, running these less intensive queries directly on the Raspberry Pi before funneling the results to the monitoring device becomes a feasible option when we later look to implement a distributed system.

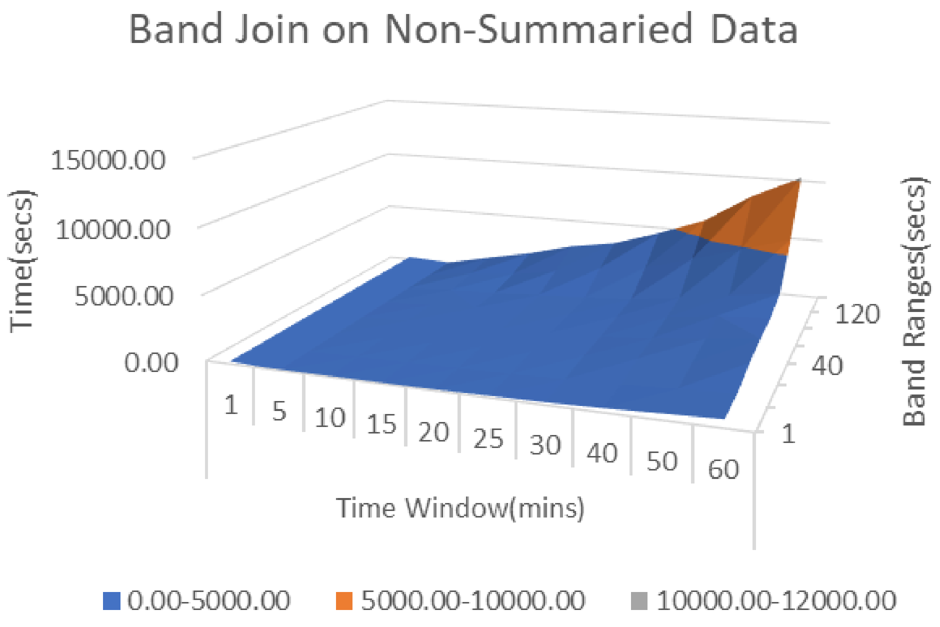

Figure 6: Band Join Performance - Time Window Records(1min:10k, 5min:57k, 10min:120k, 30min:360k).

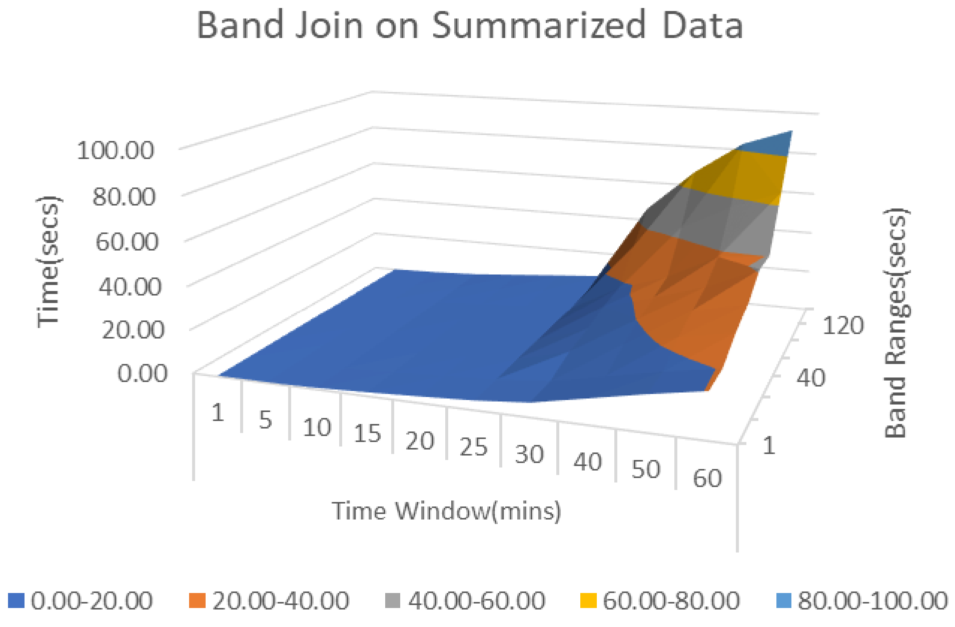

Figure 7: Band Join Performance - Time Window Records(1min:513, 5min:2k, 10min:6k, 30min:18k).

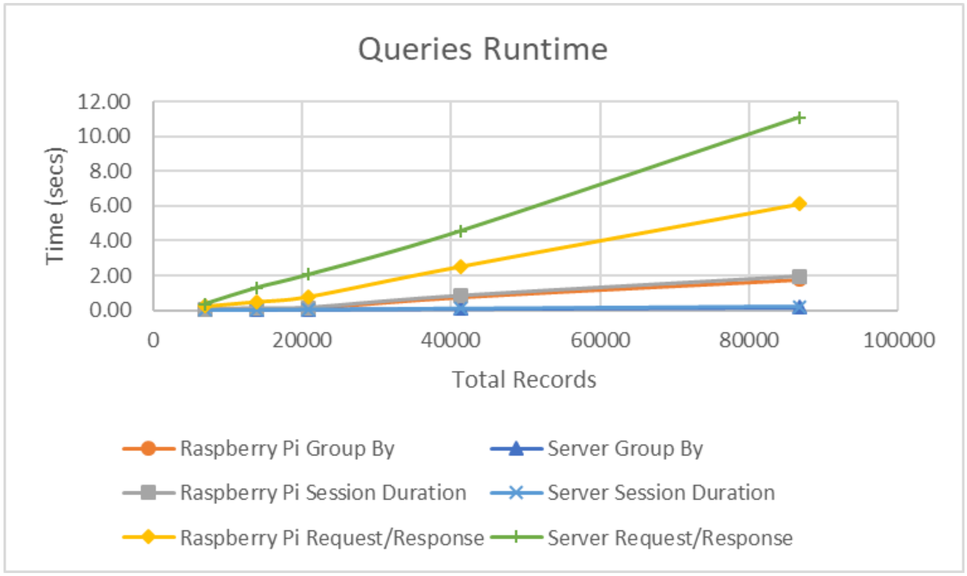

Figure 8: Queries Runtime.

6.4 Running the DBMS inside the Network Monitoring Tool

We present experiments monitoring a real local network during busy hours from 10am to 2pm in a building at our university using a small device. This time window involved about 100 network devices including servers, personal computers, and phones. We emphasize we analyzed packet-level data at the network monitoring device, which resulted in a challenging workload. We would also like to point out there were no privacy issues because we analyzed packet characteristics, not packet content and also because TCP/IP packets contain fragmented information. We ran the experiments presented in Section **??**, simulating a network administrator. The goal was to capture and periodically load and summarize big data inside a small device to later analyze, while still keeping up with the flow of data.

6.5 Experimental Discussion

In short, the DBMS does well processing and loading the data, while the bottleneck lies in converting the binary file into a CSV file, which is not the responsibility of the DBMS. To solve this, we can look into ways to directly insert the binary data into the DBMS. Our goal is to quickly load and summarize a large stream of data, so that we can analyze it in an efficient manner inside a device like the Raspberry Pi. By summarizing the data, we can reduce its size, leading to faster queries. We experimented on two queries for summarized data, group by and band join, and additionally session duration and request/response protocol for non-summarized data. Session duration and request/response protocol queries were able to perform very well, making it feasible to implement directly on the Raspberry Pis when we implement a distributed system. The group by was able to perform its task in under a second and scales well with the growing time window. The band join was able to do reasonably well at lower time windows, but quickly increased as we expanded the time and band range. When experimenting with bigger data, the device was able to handle the summarized data well. On the other hand, it can be a bit unpredictable how the device handles the non-summarized data, having odd jumps in the data or taking very long to complete. This can be caused from query optimization or the data itself, but will be an interesting issue to look into.

7 Conclusions

Availability of wireless testbeds has led to rapid advance of technologies. Adoption of technologies such as SQL will lead to faster and more flexible iterations of analysis and wider sharing of analysis techniques. Our work shows that using a DBMS on an edge device for network monitoring is viable with the use of summarizing the data using SQL. By summarizing the data, we were able to greatly reduce its redundancy, allowing the use of more intensive analysis in a reasonable amount of time. Another aspect worth mentioning is that even a low-cost and low-resource device such as the Raspberry Pi, which is widely used in wireless testbeds, is capable of processing data, arriving at high speed, but with medium volume, allowing the researchers to obtain useful network monitoring insights both for near real-time data and historical data. One of the powerful aspects of our solution is that the same queries can be used on different devices or on more powerful devices in the future for more extensive analysis.

Even though we have shown DBMSs are viable to analyze pre-processed streams for network monitoring, there are many research issues. It can be slow to load the stream into Postgres and SQL can still be slower than lower level languages like C/C++. For now, nothing is done with the info column and is mostly ignored. We need faster and more accurate queries to aggregate and summarize streams. Aiming towards a distributed system, we still need to work out the global state of the system to understand the order of occurrence in our system [20]. From the edge computing point of view, more research is needed to fully understand the storage and processing capability of small devices in the rapidly changing environment of the Internet of Things. From a database perspective, we need to study how to efficiently load new data into the database. Although plenty of work in both research and practice exist in fast database loading, we need further evaluation of their applicability in low-cost and low-resource settings.

Bibliography

[1] Akhshabi, S., Anantakrishnan, L., Begen, A. C., and Dovrolis, C. What happens when http adaptive streaming players compete for bandwidth? In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video* (2012), pp. 9–14.

[2] Al-Yaseen, W. L., Othman, Z. A., and Nazri, M. Z. A. Real-time intrusion detection system using multi-agent system. *IAENG International Journal of Computer Science 43*, 1 (2016), 80–90.

[3] Alani, M. M. Osi model. In *Guide to OSI and TCP/IP Models*. Springer, 2014, pp. 5–17.

[4] Asrodia, P., and Patel, H. Analysis of various packet sniffing tools for network monitoring and analysis. *International Journal of Electrical, Electronics and Computer Engineering 1*, 1 (2012), 55–58.

[5] Bachupally, Y. R., Yuan, X., and Roy, K. Network security analysis using big data technology. In *SoutheastCon 2016* (2016), IEEE, pp. 1–4.

[6] Borders, K., Springer, J., and Burnside, M. Chimera: A declarative language for streaming network traffic analysis. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)* (2012), pp. 365–379.

[7] Brattstrom, M., and Morreale, P. Scalable agentless cloud network monitoring. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)* (2017), pp. 171–176.

[8] Cranor, C., Johnson, T., Spataschek, O., and Shkapenyuk, V. Gigascope: a stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (2003), pp. 647–651.

[9] Duan, Q., Wang, P., Wu, M., Wang, W., and Huang, S. Approximate query on historical stream data. In *Database and Expert Systems Applications* (2011), DEXA.

[10] Fusco, F., and Deri, L. H. High speed network traffic analysis with commodity multi-core systems. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (Nov 2010).

[11] Ghafir, I., Prenosil, V., Svoboda, J., and Hammoudeh, M. A survey on network security monitoring systems. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)* (Aug 2016), pp. 77–82.

[12] Golab, L., Johnson, T., Seidel, J. S., and Shkapenyuk, V. Stream warehousing with DataDepot. In *Proc. ACM SIGMOD* (2009), pp. 847–854.

[13] Goyal, P., and Goyal, A. Comparative study of two most popular packet sniffing tools-tcpdump and wireshark. In *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)* (2017), IEEE, pp. 77–81.

[14] Guezzaz, A., Asimi, A., Sadqi, Y., Asimi, Y., and Tbatou, Z. A new hybrid network sniffer model based on pcap language and sockets (pcapsocks). *International Journal of Advanced Computer Science and Applications 7*, 2 (2016).

[15] Gupta, A., Birkner, R., Canini, M., Feamster, N., Mac-Stoker, C., and Willinger, W. Network monitoring as a streaming analytics problem. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (2016), pp. 106–112.

[16] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., and Pras, A. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Comm. Surveys & Tutorials 16*, 4 (May 2014).

[17] Jirsik, T., Cermak, M., Tovarnak, D., and Celeda, P. Toward stream-based ip flow analysis. *IEEE Communications Magazine 55*, 7 (2017), 70–76.

[18] Johnson, T., and Shkapenyuk, V. Data stream warehousing in Tidalrace. In *CIDR* (2015).

[19] Kumar, S., Dalal, S., and Dixit, V. The osi model: Overview on the seven layers of computer networks. *International Journal of Computer Science and Information Technology Research 2*, 3 (2014), 461–466.

[20] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (1978).

[21] Lee, S., Levanti, K., and Kim, H. S. Network monitoring: Present and future. *Computer Networks 65*, 2 (2014).

[22] Ordonez, C. Building statistical models and scoring with UDFs. In *Proc. ACM SIGMOD Conference* (NY, USA, 2007), ACM Press, pp. 1005–1016.

[23] Ordonez, C., and García-García, J. Vector and matrix operations programmed with UDFs in a relational DBMS. In *Proc. ACM CIKM Conference* (2006), pp. 503–512.

[24] Ordonez, C., Johnson, T., Srivastava, D., and Urbanek, S. A tool for statistical analysis on network big data. IEEE, pp. 32–36.

[25] Pacheco, F., Exposito, E., Gineste, M., Baudoin, C., and Aguilar, J. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys Tutorials 21*, 2 (2019), 1988–2014.

[26] Rhea, S., Wang, E., Wong, E., Atkins, E., and Storer, N. Littletable: A time-series database and its uses. In *Proceedings of the 2017 ACM International Conference on Management of Data* (New York, NY, USA, 2017), SIGMOD '17, ACM, pp. 125–138.

[27] Robinson, M., Jones, K., Janicke, H., and Maglaras, L. Developing cyber peacekeeping: Observation, monitoring and reporting. *Government Information Quarterly 36*, 2 (2019), 276–293.

[28] Sanders, C. *Practical packet analysis: Using Wireshark to solve real-world network problems.* No Starch Press, 2017.

[29] Sonchack, J., Aviv, A. J., Keller, E., and Smith, J. M. Turboflow: Information rich flow record generation on commodity switches. In *Proceedings of the Thirteenth EuroSys Conference* (New York, NY, USA, 2018), EuroSys '18, Association for Computing Machinery.

[30] Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., and Helland, P. The end of an architectural era: (it's time for a complete rewrite). In *VLDB* (2007), pp. 1150–1160.

[31] Umer, M. F., Sher, M., and Bi, Y. Flow-based intrusion detection: Techniques and challenges. *Computers & Security 70* (2017), 238–254.

[32] Xie, J., and Yang, J. *A Survey of Join Processing in Data Streams.* Springer US, Boston, MA, 2007, pp. 209–236.

[33] Yi, S., Hao, Z., Zhang, Q., Zhang, Q., Shi, W., and Li, Q. Lavea: Latency-aware video analytics on edge computing platform. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (2017), pp. 2573–2574.

[34] Yuan, Y., Lin, D., Mishra, A., Marwaha, S., Alur, R., and Loo, B. T. Quantitative network monitoring with netqre. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), pp. 99–112.