Monitoring Networks with Insightful Queries

Quangtri Thai
University of Houston

Carlos Ordonez
University of Houston

Omprakash Gnawali
University of Houston

ABSTRACT

Monitoring networks requires 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 questions should be relatively simple and data pre-processing should be kept at a minimum. Another requirement is to be able to process data, both in a centralized and decentralized manner given the dynamic nature of TCP-IP packet flow. 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. A brief experimental evaluation shows queries can indeed be used to monitor the network with low latency and reasonable delay.

1 INTRODUCTION

Over the last decade, researchers have increasingly adopted the use of testbeds to bring realism in their wireless network experiments. The use of testbeds have led to the development of technologies that are more feasible in real world settings. Roofnet was an early example of such a city-scale testbed and is followed by numerous wireless testbeds including a large number of low power wireless testbeds [2]. Lately, the rise of low cost computing devices such as Raspberry Pi has allowed these testbeds to be built at a large scale in many places around the world. Such testbeds allow administrators 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 to the researchers 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 [1, 4], 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 model used by these testbeds to export data to the researchers. We do not want to require the testbeds to significantly change the export data format or require the researchers to perform complex data pre-processing. Second, rather than reinvent yet another data processing framework, 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 [3]. Streams will be captured in small batches, then added and analyzed in the database to give a historical analysis of the data. While using custom Python/R code for data analysis can provide further specialized queries, this come with the price of being much harder to implement and is much more of a hassle to change and specialize to different tasks. Our solution will allow more interactive and flexible exploration of network data by the 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.

In this paper, we study how viable it is to monitor and analyze a network stream through the use of SQL. We then study the feasibility of implementing a query system directly on the edge device, such as a Raspberry Pi, and run it reasonably efficiently for both near real time and historical data

WiNTECH'20, September 21, 2020, London, United Kingdom

ACM ISBN 978-1-4503-8082-9/20/09.
https://doi.org/10.1145/3411276.3414695

analysis. We will continuously capture the network stream and process it on the Raspberry Pi, while measuring how long each step takes. This poses a challenge of having to handle the large volume and velocity from the stream. This will be done in both a centralized and distributed manner to compare and contrast the different methods. Intensive analysis will be ran on the data captured and timed to see how viable it is on such a small device. We look to understand how much resource and time is being required to store and analyze these large streams of data, how usable these analysis are, and how well it 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 within a time window. (3) Implementing and evaluating a light-weight DB and the SQL-based network instrumentation system on an edge device in centralized and distributed configurations.

2 TECHNICAL CONTRIBUTION

We seek to monitor and analyze the network using SQL commands on the edge devices in both a centralized and distributed manner. We follow this approach because of the many advantages and flexibility SQL would offer. This would include practically unlimited storage, querying, and scalability. 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 your 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.

We want a data stream that has an intensive load, as well as multiple connections from many users. This is to mimic crowded, real world situations to try and push the limits of our monitoring device in a realistic scenario. The stream is loaded into a relational table L defined as L(ts,src,src_prt,dst, dst_prt,prot,len,info), with the timestamp, ts, being the index.

Q1: Request/response protocol: The request/response protocol is a useful protocol that can be observed to find network failures, checking if there is a response to all requests or vice versa. The query works by selecting all rows where the source and destination are switched, representing a response. This is done on a time window between $[a, b)$.

```
SELECT src,dst FROM L   /*Q1*/
WHERE (src,dst) NOT IN (
  SELECT dst,src FROM L
  WHERE a <= ts and ts <= b
) AND a <= ts and ts <= b;
```

Q2: Detecting co-occurring events: To detect all co-occurring events within the table, we use a band join, a theta join with inequalities. Band joins will allow us to determine what happens around the same time between multiple streams, a complicated task made simple with SQL. Our query give an alias to table L as L1 and L2, then performs a band join on both tables with a time window between $[a, b)$ and constant $\pm c$. From there we can use an aggregation to give us more information about the stream. This is shown as aggr() and can represent sum(), max(), count(), or any other aggregate function.

```
SELECT L1.src,L1.dst,aggr() FROM (    /*Q2*/
  L AS L1 INNER JOIN L AS L2
  ON L1.ts-c <= L2.ts AND L2.ts <= L1.ts+c
  AND L1.src != L2.src)
WHERE a <= L1.ts AND L1.ts <= b
  AND a <= L2.ts AND L2.ts <= b
GROUP BY L1.src,L1.dst;
```

Preliminary experiments: To validate our ideas, we performed preliminary experiments. We measured the delay required to load and process the network and time it takes for both queries to analyze the data. From our experiments, we found the query to detect co-occurring events (band join) and the query to analyze the response protocol (set containment) could return results in a few seconds for short time windows (e.g. one minute or less).

Table 1: Query Times

Query	Window (secs)	Records	Server (secs)	Raspberry Pi (secs)
Q1	10	6639	0.03	0.07
	20	13422	0.06	0.14
Q2	10	9220818	38.47	15.11
	20	18744282	167.47	30.96

REFERENCES

[1] Arpit Gupta, Rüdiger Birkner, Marco Canini, Nick Feamster, Chris Mac-Stoker, and Walter Willinger. 2016. Network monitoring as a streaming analytics problem. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. 106–112.

[2] David Johnson, Tim Stack, Russ Fish, Daniel Montrallo Flickinger, Leigh Stoller, Robert Ricci, and Jay Lepreau. 2006. Mobile emulab: A robotic wireless and sensor network testbed. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. IEEE, 1–12.

[3] Carlos Ordonez, Theodore Johnson, Divesh Srivastava, and Simon Urbanek. 2017. A Tool for Statistical Analysis on Network Big Data. IEEE, Lyon, France, 32–36. https://doi.org/10.1109/DEXA.2017.23

[4] Yifei Yuan, Dong Lin, Ankit Mishra, Sajal Marwaha, Rajeev Alur, and Boon Thau Loo. 2017. Quantitative network monitoring with NetQRE. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 99–112.