A Lightweight Persistence System for Network Traffic Measurements

Michael Mente 01634435, a01634435@unet.univie.ac.at

University Vienna, Faculty of Computer Science

1 Introduction

Introduction copied from Exposee for context, still somewhat redundant with related work
As never before, the internet plays a central role in our society, driven on the one hand by technological advances and on the other by social conditions such as the COVID19 pandemic. All of these things have led to services supported by cloud computing providers being more heavily utilized than ever before, be it by the increased use of video conferencing tools or by online streaming providers. As a result network traffic is not only increasing in volume, but also in velocity. However, due to the growing amount of network traffic, it is becoming more and more challenging to analyze this data. Currently, traffic is mostly processed live by means of special applications to detect possible issues. Despite these sophisticated algorithms, it is still possible that a security or performance issue cannot be detected in real time. Nevertheless, there are only limited possibilities to analyze a disturbance after it has already occured [1]. One possibility to analyze a fault at a later point in time is a database system that stores all network traffic. However, this approach is very storage intensive and existing solutions do not scale to meet these high traffic demands. In addition, these systems use, among other things, mostly complex indexes that require a lot of computational effort due to which the ingestion rate is comparatively low. There are two existing approaches to store the data: On the one hand, classic SQL databases have the advantage of straighforward data analysis due to their natural structure. Already existing SQL databases optimized for time-series-data have a comparatively high computational overhead and are therefore not fast enough to meet the requirements. One of the fastest relational databases for this type of data is Timescale DB. However, it has been shown that the maximum ingestion rate of about 400K packets per second is not sufficient to log the traffic of a top of rack switch [1]. On the other hand, NoSQL databases offer a potentially higher ingest rate which is one of the most important factors for a project of this kind. Nevertheless, even NoSQL systems are not nearly performant enough to meet the requirements.

2 Related Work

2.1 Network Monitoring & Analytics

While many network monitoring systems exist that address different challenges and implement different solutions, they almost all have in common that traffic is only monitored in a live manner. In terms of functionality, many implementations perform similar tasks and differ mainly in the level of granularity of the telemetry data collection or specializations, be it pre-processing in the data plane or customizations for a specific purpose. Network telemetry can be done in two ways, on the one hand there is flow-based sampling, where metadata is collected directly, and on the other hand there is packet-based sampling, which offers potential for best possible analysis but may require specialized hardware such as programmable switches [2, 3] to mirror the network traffic to the monitoring devices.

Sonata [3] is an example of a network telemetry system with a wide range of functionality at the packet level, and a focus on scalability and a unified query interface. This means that the queries are made without the network operator having to worry about technical details about the network interface device or the way of execution. An example of the usage of the expressive query language is shown in Figure 1 which detects newly created TCP Connections.

```
packetStream ( W )
  .filter ( p => p. tcp . flags == 2 )
  .map ( p => ( p . dIP, 1 ) )
  .reduce ( keys =( dIP, ), f=sum )
  .filter ( ( dIP, count) => count > Th)
```

Fig. 1. Sonata packet stream detecting new TCP connections [3]

Other solutions [4] focus on similarity to existing database languages and have developed their own SQL derivative. In the case of Gigascope, GSQL was developed that implements known syntax with new functions like stream-merge:

```
DEFINE{query name tcpDest;}
     Merge tcpDest0.time : tcpDest 1.time
     From tcpDest0, tcpDest1
```

Fig. 2. GSQL stream merge statement [4]

This (Figure 2) will combine streams from two different devices into one stream, with the intention that this way also simplex links can be turned into proper network traffic flows and thus more flexible and meaningful analysis can be created.

2.2 Record Persistence & Retrospective Queries

Although the idea of retrospective network traffic analysis has existed for a considerable time, the requirements for such solutions have changed dramatically. One of the first solutions for storing network traffic was Gigascope – a database for network applications – in 2003. Apart from the pure database functionality, Gigascope provides a whole suite of network monitoring tools for common traffic analysis as well as more advanced tools for specific analysis, e.g., router configuration analysis. Gigascope was installed on backbone links of the us network operator AT&T, OC48 routers to be precise. A large part of this paper focuses on the development of a domain-specific query language similar to SQL. However, the challenges encountered relate to the nature of stream databases. Cranor, Chuck, et al state, since the system is deployed on a router, it is necessary to query streams from multiple interfaces simultaneously in case the router uses optical directed simplex links. Another important point in their work was the performance insights. At that time (2003) the traffic rates were as high as about 650 megabits on a backbone link from AT&T, which corresponds to a throughput of 1.2 million packets per second resulting in 500 GB of generated data sets per day. Nevertheless, it is remarkable that such performance values could be achieved with commodity hardware on a 2.4 GHz dual CPU server. The most important findings regarding the performance of such a system were that, in general, the earlier the reduction of the data volume, the higher the performance. However, it should be noted that the focus of the performance of stream databases is not on the speed of query resolution but rather on the maximum ingestion rate. In order to achieve such values, it was necessary to build complex hardware setups to achieve the high write rates to the hard disks, in particular the authors used disk striping, i.e., a RAID-0 hard disk array. Limited by the technology of the time - SSD technology was still very new and, above all, too expensive for large-scale deployment - such a hard disk array tends to lead to an increased failure rate, as further sources of error are introduced.

Due to the constantly advancing development of hardware, completely new possibilities for improving such a system have opened up since the development of Gigascope. On the one hand, the trend towards multi core CPUs gives the possibility to process a continuous data flow in parallel more efficiently, on the other hand, the storage possibilities have improved due to the continuous research on SSD storage technologies, be it in speed, durability, or price. As a result of these advances, research has been conducted to determine whether general purpose databases could permanently store network traffic [1]. Analogous to the technological progress, the requirements for persistence systems have also grown, such database solutions should be able to handle even hundreds of millions of packets [1]. Modern database solutions potentially offer sufficiently high ingestion rates, but since some structure is required for subsequent analysis of network traffic for high-expression queries, NoSQL are not suitable for these tasks. Therefore, Michel et al. chose TimescaleDB as a promising candidate, since it provides an SQL interface and optimizations regarding to time scale data, to measure its suitability for storing network packets. In the course of the research, it was shown that the query performance is ideal for this application area of interactive data analysis. Despite the promising results in query performance, SQL database systems are not particularly well suited for write-intensive tasks such as network packet storage. Another factor that reduces the insertion rate is the amount of additional computation required to provide the indexes for the SQL interface. The measured insertion rate was only about 400k packets per second, which means that if the link is fully utilized, only a fraction of the network traffic could be stored.

To avoid this problem there is the possibility to process the packets in advance to keep the insertion rate lower. For this the grouped packet vector format (GPV) was used which was developed before [5]. GPV is a method to group packets in a way that a composite of a packet record and a flow record is created which has already been processed efficiently on hardware [5]. This has the additional advantage that the required storage volume is reduced since several packets are stored in one GPV.

However, if a system is to be used for permanent storage of network traffic, such methods are not sufficient to utilize the available storage as efficiently as possible. The problem especially with network traffic is the high rate of entries that potentially allocates a lot of storage to the timestamp. This problem has also been encountered previously in other areas where high ingestion rates are required [6]. To counteract this problem modern databases such as TimescaleDB use sophisticated encoding mechanisms to reduce the size of individual entries.

2.3 Compression Techniques

It has been shown that even specialized databases that have been optimized for so-called timescale data sets still have a major impact on storage efficiency [7]. Such database types are optimized for repetitive data sets, such as a timestamp, which accounts for a relatively large proportion of the storage volume, depending on the accuracy of the data. However, there are also further possibilities for increasing storage efficiency through optimization for the domain specific application scenario. A common possibility for incrementing data is the reduction of the length of the data types, by only storing the difference to the previous value, also called delta encoding. However, it is also possible to apply multi-stage delta compression processes in succession in order to achieve even higher memory efficiency at the expense of the computational effort [8, 9]. In general, knowledge of the data has the potential to save more storage space as opposed to data agnostic compression techniques. This means, for example, in the context of network telemetry, that one can reduce delta values of timestamps to a small 16-bit or even 8-bit data type if one knows that a certain value will not be exceeded due to a certain packet rate [10, 9]. Another technique is dictionary encoding which stores a few repetitive values in a dictionary in a different location and exchanges the value with the index in the entries. However, this requires that the storage space of the index is smaller than the actual value to achieve an advantage [10].

In the domain of network telemetry, it is important to note that mostly only metadata, i.e., packet headers, are analyzed [4, 1, 5]. Apart from the IP 5 tuple (source & destination IP address, source & destination port, protocol) the timestamp makes up a large part of the storage volume depending on the resolution. However, further advantages can be achieved by also compressing the data points as shown by Shi, Xuanhua, et al [7].

Bibliography

[1] Oliver Michel, John Sonchack, Eric Keller, and Jonathan M Smith. Piq: Persistent interactive queries for network security analytics. In *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 17–22, 2019.

[2] Oliver Michel. *Packet-Level Network Telemetry and Analytics.* PhD thesis, University of Colorado at Boulder, 2019.

[3] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 357–371, 2018.

[4] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 647–651, 2003.

[5] John Sonchack, Oliver Michel, Adam J Aviv, Eric Keller, and Jonathan M Smith. Scaling hardware accelerated network monitoring to concurrent and dynamic queries with* flow. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 823–835, 2018.

[6] Frank Eichinger, Pavel Efros, Stamatis Karnouskos, and Klemens Böhm. A time-series compression technique and its application to the smart grid. *The VLDB Journal*, 24(2):193–218, 2015.

[7] Xuanhua Shi, Zezhao Feng, Kaixi Li, Yongluan Zhou, Hai Jin, Yan Jiang, Bingsheng He, Zhijun Ling, and Xin Li. Byteseries: an in-memory time series database for large-scale monitoring systems. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 60–73, 2020.

[8] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015.

[9] Torsten Suel. Delta compression techniques. *Encyclopedia of Big Data Technologies*, 63, 2019.

[10] Joshua Lockerman. Time-series compression algorithms, explained, Mar 2021.