Using a columnar data structure to analyse network telescope data

THOM KASTELEIN, University of Twente, The Netherlands

Network telescopes offer valuable insights into unsolicited internet traffic, but their data is often unwieldy due to its size and the inefficiency of traditional storage formats. This research investigates the use of a columnar data structure as a more efficient alternative to sequential packet storage. A prototype system was developed to convert and analyse PCAP-formatted data, with performance assessed through typical analysis tasks. The results highlight notable improvements in aggregation task speed and long-term storage, while revealing shortcomings for optimisation in packet filtering operations.

Additional Key Words and Phrases: Network traffic archival, Network traffic analysis, Network telescope

1 INTRODUCTION

As specified in RFC 791 [1], an IP address as found in the Internet Protocol version 4 (IPv4) header is constructed from a series of exactly 32 bits. In total, there are approximately 4.3 billion possible ways to arrange these 32 ones and zeros. This, in turn, allows for the same number of valid IPv4 addresses. Due to their limited quantity, ICANN (more specifically, their IANA subdivision) is in charge of assigning IPv4 addresses to other entities [11]. Even though all such addresses have been allocated for over a decade, not all of them are actively in use. Given that, by its definition, an unused address is not being used by any host machine, one would expect for these addresses to not receive any traffic. However, this is in fact not true due to a phenomenon called "internet background radiation" [13]. By this phenomenon, any IP address could be addressed by unsolicited traffic for several reasons:

(1) Misconfigurations – traffic sent erroneously due to faulty software
(2) Denial-of-service backscatter – traffic resulting from IP spoofing used in DoS attacks
(3) IP scanning – traffic from widespread probing of host machines in search of specific services and/or vulnerabilities

While most of this background radiation goes unnoticed, if one were to set up a mechanism to capture the packets that are sent to unused addresses, it would provide valuable insight into the causes of these traffic categories.

TScIT 43, July 4, 2025, Enschede, The Netherlands

This is where network telescopes come in. These machines are configured to store any of the unsolicited traffic without responding. Later, the captured data is analysed by researchers. A problem arises, however, when trying to efficiently analyse the telescope's data. The problem stems from the fact that a network telescope generates a great quantity of data. As an example, the UCSD network telescope, which is comprised of a */9* and */10* subset, generates around four terabytes of data every day [9]. Captured data is most often stored using either the PCAP [10] or PcapNG [16] file formats. Both are sequential formats: the captured data for each packet is stored one after the other. When analysing network telescope data, not all fields of a packet may be used, as is exemplified by M. Zakroum et al. [20]. Their research was focused on the *destination port* field found in the TCP protocol's header. Performing this analysis using the current storage method requires looking over irrelevant fields of the TCP header. Worse yet, packets which do not contain a TCP header are looked over as well. Would it be possible to somehow read only the fields in which we are interested, thereby speeding up analysis, and potentially reducing storage demands?

This paper proposes a new data structure that is organized within columns, to solve precisely this problem. To assess the effects of this structure on the analysis of network telescope data, the following research questions are posed:

- **Main RQ** – How feasible is the use of a columnar structure for the analysis of network telescope data, as an alternative to using PCAP files?
- **Sub-RQ 1** – What are the effects of a columnar data structure on the compute time of typical network telescope analysis tasks?
- **Sub-RQ 2** – What are the effects of the columnar structure on the data storage requirements?

To find answers to these questions, we created prototype software that is capable of converting data encoded using the existing PCAP file format to the newly proposed data structure. Next to this, the software can read the data from this new structure, and perform four different evaluation tasks that are typical for the analysis of network telescope data. To answer the first sub-research question, the mean compute time of these tasks was measured, both using PCAP files and the new format. In order to answer the second sub-question, the total disk usage of both formats were measured. These results from these measurements

were then compared, and evaluated together to answer the main research question.

This research paper will first cover the requirements that the new data structure would need to satisfy for it to be applicable to the analysis of network telescope captures. Next, it outlines what potential solutions to this problem already exist, as well as why these do not satisfy the given requirements. Then, the new, columnar data structure is presented, after which it is verified whether it does satisfy the requirements. Lastly, the conclusion provides answers to the research questions stated above, as well as a discussion of potential further research and improvements.

2 REQUIREMENTS

While the previous section only loosely mentions that the new data structure should be adept at typical network telescope analysis tasks, how does this goal affect its design? This section lays out several concrete requirements that the design has to adhere.

2.1 Reading single fields

Firstly, as is written above, it is assumed that slowdown during the analysis of network telescope data originates from the process of separating relevant from irrelevant data. This process involves iterating over each individual packet, and then disregarding it if deemed unrelated to the research purposes. Put simply, processing time is being spent reading data that will not be used. Therefore, a solution which does not read this thrown-away data, would in theory be more efficient. A new structure would address this issue by allowing a program to only scan through the necessary fields.

2.2 Storing relevant fields

Second, a new structure should have the ability to store the fields that are relevant to the analysis in question. To have the design be useful in a broader context, the structure should be flexible enough to not only facilitate tasks related to network telescope analysis, but those used in the more general field of network traffic analysis as well. Fields that are typically used during network traffic analysis are found in Table 1. If the new data structure stores only these fields, a substantial amount of disk space could be saved. Such a consideration is not applicable to the analysis of telescope data however. Nefarious actors do use the padding and unused bits found in some protocol's headers to encode actual information [21]. Given that one of their main traffic classes is denial-of-service backscatter, it is crucial for the analysis of network telescope data to be able to store such fields. Therefore, it is imperative that any new data format should contain all information found within a packet.

2.3 No wasted space

Third, while figuring out packet relevancy is a challenge, it is certainly not the only one faced during network telescope data analysis. A second issue stems from the sheer volume of data that is generated by some telescopes. Smaller telescopes, such as the SURF-NT telescope, covering a */16* and three */24* subnets, generate a manageable amount of data. However, others like UCSD's telescope, utilising */9* and */10* subnets, capture a far greater amount. This problem only increases with the size of the network telescope's underlying subnet, as well as with the general increase in traffic over time [9]. Clearly, more input data will always lead to longer compute times. Therefore, it is crucial that the new design does not use more disk space than the existing PCAP format, as this would likely counteract any processing speed improvements arising from its structure.

2.4 Parallel processing

Lastly, another bottleneck faced when processing PCAP files is that this is mainly done on a single CPU core. Even though it is true that solutions exist for the parallel processing of PCAP files [2, 19], these are not viable for the analysis of network telescope data, as they do not consider the normally unused bits. Next to this, given that the PCAP standard was created in the late 1980's, parallel processing was likely not one of its design considerations, since multi-core processors only became popular around 1995 [12]. For this reason, a new data format could, and should take advantage of the capabilities of modern hardware.

3 EXISTING SOLUTIONS

When the new data structure is combined with a program that is able to interface with it, it creates what is known as an internet traffic archival system (ITAS). Many such systems do already exist. However, these are not suitable in relation to network telescopes. This section goes over the systems mentioned by Chen et al. [4], and explains the reasons for their unsuitability.

3.1 Telegraph CQ and other flow-based systems

Telegraph CQ, as well as similar flow-based archival systems, derive their record structure from the NetFlow format [5]. Usually, such records are composed of only five fields, together defining a flow: the source and destination IP addresses, the source and destination port number, and the transport layer protocol identifier. This means that, without specialised settings, most of the

Table 1. Typical fields used in network traffic analysis

Field	Protocol	Details
Timestamp	N/A	Date and time of when the packet was received
Packet Length	N/A	Number of bytes contained within the packet
Source IP	IPv4	IP address of the packet's origin
Destination IP	IPv4	IP address of packet's target host
Protocol Number	IPv4	Identifies what transport layer protocol was used
Time-to-Live	IPv4	Can hint at operating system [14]
Source Port	TCP/UDP	Identifies process on the origin's machine
Destination Port	TCP/UDP	Indicates a service on the host machine [20]
Window Size	TCP	Can hint at operating system [14]
Flags (SYN, ACK, etc.)	TCP	Used to distinguish port scanning, backscatter traffic and misconfigurations [3]

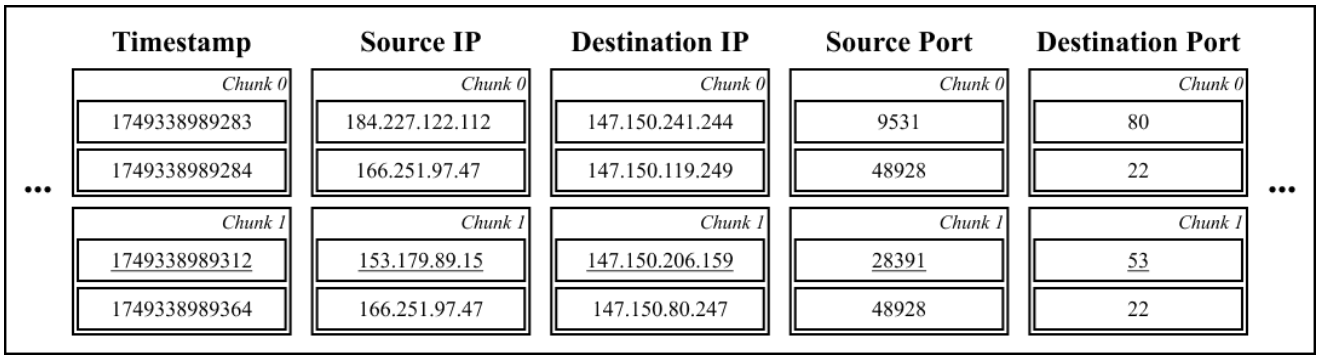

Fig. 1. Structure of an IPv4 and TCP table with a chunk size of two (as an example, the underlined values belong to the same packet)

flow-based systems will not save the fields we are interested in. On the other hand, if one were to configure these systems to save all relevant fields, this would result in a significantly larger record size. This is because every field of a NetFlow record is stored using at least one byte. Thus fields that occupy the same bytes within protocol headers (e.g., IPv4's flags and fragment offset), take up more space in a flow record. Additionally, flow-based systems are not capable of storing the reserved/padding bits, as these are assumed to follow the respective protocol's specification.

3.2 Hyperion and other row-based systems

Next, Hyperion [6] and other row-based systems exhibit a different issue. For this type of system, their records are a combination of several fields next to each other. By this, similar to using PCAP files, in order to extract relevant data, one must sift through data which is irrelevant to their research goal. Due to this extraction step, the performance of row-based archival systems would be sub-optimal for the analysis of network telescope data.

3.3 PCAPIndex

Out of all previously existing systems covered by the research of Chen et al. [4], only PCAPIndex has both a columnar structure, as well as packet-level granularity. However, PCAPIndex builds indexes to speed up sparse queries, for which related results are spaced far apart in the data. Simply put, these indexes take up space, and grow in size proportionally to the total number of records. Despite their applications in general network traffic analysis, these indexes serve little purpose when it comes to telescope data analysis, where research is mostly focussed on the aggregate statistics [14].

4 NEW ARCHITECTURE

With the motivations behind the creation of the new data format in place, it is time to discuss its structure in detail. It should be noted that this structure (and the terminology thereof) is inspired by Apache Parquet [18].

4.1 Database Structure

A packet as it is stored within a PCAP file is made up of three parts:

(1) Packet header – This contains general packet information, namely a timestamp and the packet's length in bytes.
(2) Series of protocol headers – Headers contain information about different networking protocols. They serve as a collection of fields. For example, the IPv4 header is a combination of a source address field, a destination address field, etc.
(3) Payload – This is application-layer data of a variable length.

Even though not all packets use the same protocols, there are only so many ways to combine them. Taken with the fact that each header always contains the same set of fields, this gives that packets with the same collection of protocol headers will also have the same collection of fields. These are used as schemas in a similar manner to other database systems. Another similarity is that the database is divided into several tables. Each table is dedicated to a different combination of protocols. This is done so that every packet stored within a certain table has the same collection of field, i.e., they can be represented by a combination of the same fields. Thus, a table's schema is derived from the union of the sets of fields from the protocol headers contained within its packets. Additionally, a table stores the timestamp and length of a packet.

4.2 Columns and chunks

Within a table, each field is assigned a column to store the corresponding values which are extracted from packets. The division into columns is made so that it is possible to read values of a specific field, without having to sift through unrelated data, as was specified in Section 2.1. Each column is subsequently divided into a series files, each one storing a fixed number of values (see Figure 1). This number is the *chunk size*. Since (most) field values always have the same length, it is simple to calculate exactly where a specific value is stored. If one were to recreate a packet's data, they would have to retrieve values from every column, all at the same position. There are two distinct kinds of columns: ones with a fixed size, and ones with a variable size. These will now be covered in detail.

4.2.1 Fixed-sized Columns. First are fixed-size columns. The values in this type of column are always represented using the same number of bits, the *column width*. Most fields in protocol headers are represented by some multiple of eight bits, i.e., whole bytes. There are some fields however, that do not nicely align to this convention. The fragment offset found in the IPv4 header, for example, is thirteen bits wide. To preserve space within chunk files, these values are densely packed, and do not follow byte

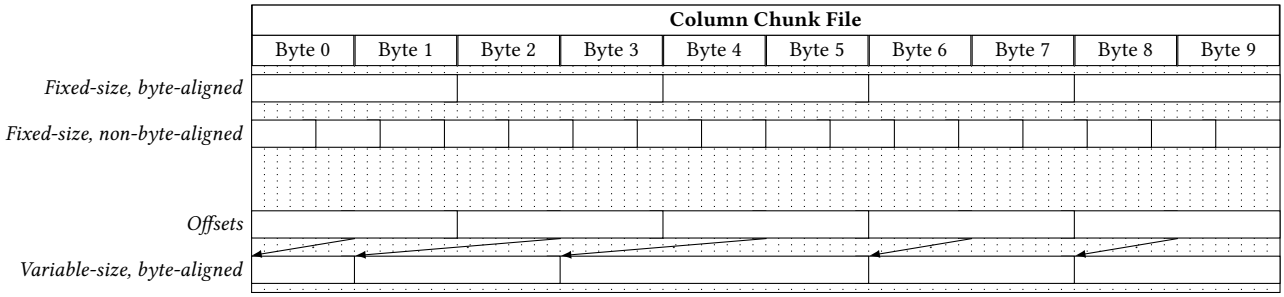

Fig. 2. Value alignment within a column chunk file

boundaries. Specialised reading and writing algorithms are required to work with this. Both of these algorithms are slightly less efficient than those for byte-aligned columns. With a column width of w bits, to read a single value, the reading algorithm takes at most $\lceil \frac{w}{8} \rceil + 1$ byte read operation. Similarly, to write a single value, the writing algorithm requires at most $\lceil \frac{w}{8} \rceil + 1$ write operations, and possibly a read operation. Pseudocode for both is given in Appendix B. When working with byte-aligned values, these algorithms greatly simplify so a sequence of byte read/writes.

4.2.2 Variable-sized Columns. Next to fixed-size fields, some protocol headers contain fields that are variable in length. An example of these are TCP options [7]. Due to their variable length, the regular calculation to figure out a value's position does not work. Instead, for these columns, an extra columns is constructed. The purpose of this extra column is to store the starting positions of the variable-length values (see Figure 2). Thus, to retrieve a value, first both its and the next value's starting positions are retrieved. Then from the regular chunk file, the data between both starting positions is read. Since packet payloads also vary in length, they are stored using the same mechanism.

5 EVALUATION

Whether the proposed, columnar data structure indeed meets the requirement specified in Section 2 was experimentally verified. To make sure that this was performed under realistic conditions, a random sample of five PCAP files captured (in 2025) by the SURF-NT network telescope were used. Each file is around 430 MB in size, for an approximate total of 2.2 GB. These were then sequentially combined into larger PCAP files, i.e., one with only the first PCAP file, one with the first and second, another with the first, second and third, and so on. For all evaluation tests, a virtual machine was used. This virtual machine was equipped with an AMD EPYC 9534 processor running 2.45 GHz. Any memory read and write operations were performed using a 500 GB HDD with an average read speed of 540 MB/s.

5.1 Task run time comparison

To measure the processing speed of a system using the columnar data structure, the mean compute time of a set of four tasks was measured using both PCAP files, and the new structure. These tasks are typical for the analysis of network telescope data:

(1) Find the top ten most common source IP addresses. [14]
(2) Find the top 10 most common destination ports, across both TCP and UDP. [20]
(3) Retrieve every packet which uses UDP, with 53 (DNS) as its destination port. [15]
(4) Calculate the percentage of packets using TCP with destination port 80, which contain an HTTP GET request. [17]

The runtime of these tasks was measured using chunk sizes of *4096*, *8192*, *16384*, *32768*, and *65536*. For each combination of task and chunk size, ten iterations were performed to reduce the variance of the sample means. To make sure measurements were independent on the others, all cache memory was cleared before every test iteration. By comparing the results from tests that used different chunk sizes, it was found that these do not differ significantly. Therefore, to increase readability, only results obtained by using a chunk size of *65536* are presented below. We have chosen to not include the time required to convert PCAP files to the new structure, because of the assumption that data would be stored using the new structure in the long term.

5.1.1 Task 1 – Top 10 source IP addresses. The results for task 1 show a clear distinction in compute time between the old and new structure. This is what was expected, as utilising the new format, a program would have to traverse around 95% less data than if if were to use PCAP files. Theoretically, the counting step should also be faster for the new format, as it can deal with raw bytes, while *tshark* can only output specific fields using text-based formats. In practice however, only 0.4% percent of the time taken by the old method is used to count IP address frequencies, making this difference negligible. Therefore, the differences between the two data structures seen in Figure 3 are primarily a result of method of extracting the IP addresses.

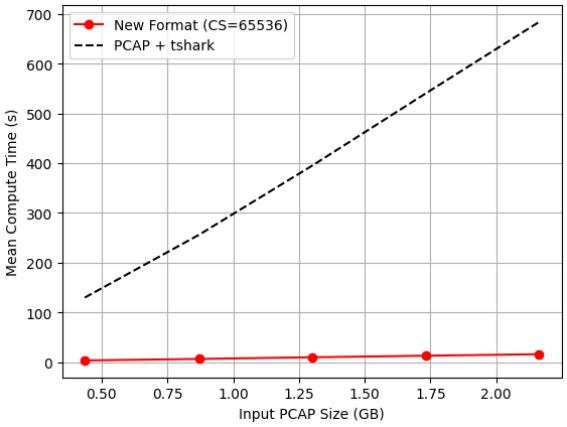

Fig. 3. Relation between input size and task 1 compute time

5.1.2 Task 2 – Top 10 destination ports. Task 2 produces an extremely similar result (see Figure 4) to that of task 1. It is therefore

likely that task 2 is faster when using the new format for the same reasons. On top of the reduced reading that arises from the columnar structure, the new method should, in theory, also be faster because of the initial division into tables based on protocol usage. In reality however, TCP and UDP traffic makes up 98.9% of the testing data, therefore only causing a marginal difference.

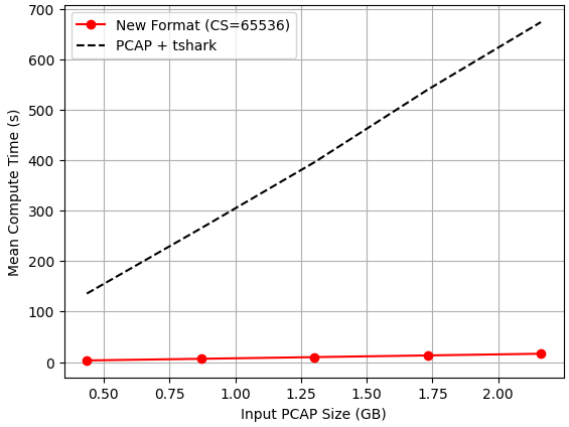

Fig. 4. Relation between input size and task 2 compute time

5.1.3 Task 3 – Retrieving DNS packets. For task 3 and 4, two existing methods of packet filtering were tested: *tshark* and *tcpdump*. While tshark is general a general purpose tool for traffic and packet analysis, tcpdump is specifically optimised to filter packets. This is also reflected in the resulting graphs (Figures 5 and 6). By comparing these existing methods with the new structure, it can be seen that the prototype is again significantly faster than tshark. On the other hand, the prototype fails to outperform tcpdump. It should be noted that the prototype used to perform these tests is by no means fully optimised. Firstly, there are very likely to be much faster read and write algorithms than those shown in Appendix B. Also, the prototype does not leverage the potential for parallelism offered by dividing columns into chunks. It is possible that, if these improvements were to be made, the new format would be comparable to (or even surpass) tcpdump in performance.

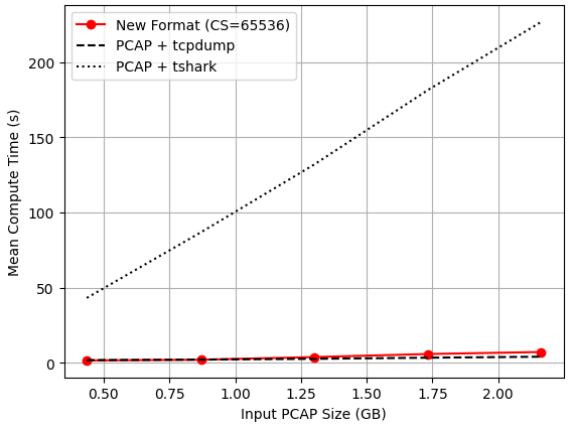

Fig. 5. Relation between input size and task 3 compute time

5.1.4 Task 4 – Calculating HTTP GET percentage. For task 4, the new format is again slower than tcpdump. This time however, the contrast between the two is much larger. Since this task deals with packet payloads, which is a variable-length field, it is possible that this discrepancy is the result of having to read extra data from its *offset column*.

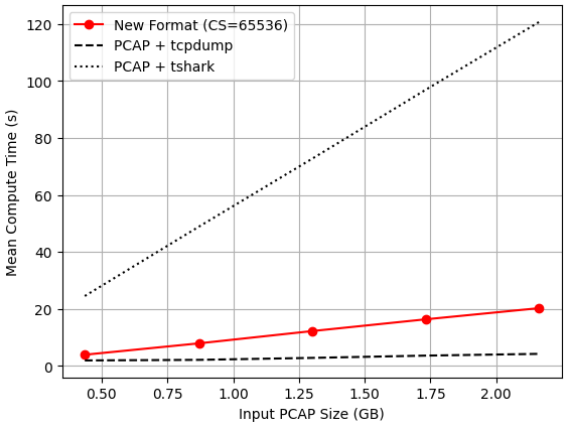

Fig. 6. Relation between input size and task 4 compute time

5.2 Disk space comparison

When considering the disk space usage of the new format, there is a clear contrast between compressed and uncompressed storage methods. First, regarding the unzipped method, the columnar format takes up significantly more disk space than the old format. Even though the data in the chunk files is the exact same as that in the PCAP files, just rearranged, the number of chunk files has a significant impact on the disk space usage. This is also evident from the graph in Figure 7, as is shows that using larger chunk sizes, which result in fewer files, reduce the overall disk usage.

On the other hand, there are the compressed versions of the new format. These show a general trend of using less disk space than a compressed PCAP file, with larger chunk sizes again using less disk space. This can be explained by the fact that run-length encoding schemes, which gzip is based on, are more space-efficient when contiguous data is homogeneous [8]. Since the new format divides its columns into chunks, it also makes it possible to gather specific records by selectively decompressing chunk files. In summary, when uncompressed, the new format uses 100 bytes per packet on average, compared to 79.2 bytes with PCAP files. When gzip compressed, these average to 26.7 and 30.2 bytes, respectively.

6 CONCLUSION

To conclude, this paper covers the concept of an internet traffic archival system designed specifically for the analysis of network telescope data. This design divides its structure into columns, which enables a reduction in the amount of data that has to be read to perform a query. This final section serves to provide answers to the research questions posed in the introduction, regarding the new structure's impact on processing speed, storage efficiency and viability for research.

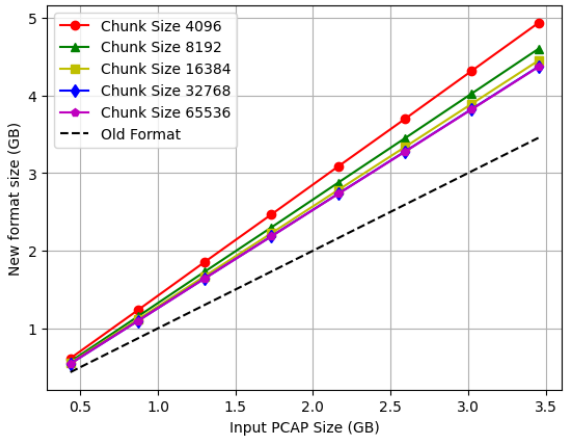

Fig. 7. Relation between PCAP size and the new format's disk usage (uncompressed)

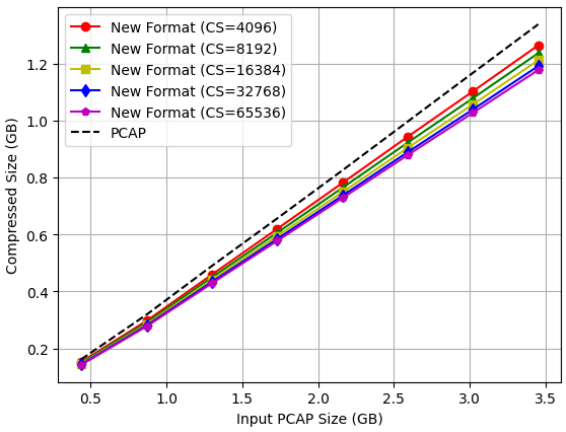

Fig. 8. Relation between PCAP size and the new format's disk usage (gzip compressed)

6.1 On sub-research question 1

Firstly, what are the effects of the new data structure on the processing speed of typical analysis tasks? As is evident from Figures 3 and 4, the new structure poses significant benefits when it comes to *aggregation tasks*, where all values of a single protocol's field are used to compute a combined result. This is because for these task, tshark is most often used, which is outperformed by the new format. On the other hand, as is shown by Figures 5 and 6, the prototype performs worse than the existing methods on *filtering tasks*, where packets are selected based on a predicate. This is to be expected tcpdump is optimised specifically to perform this kind of task efficiently.

6.2 On sub-research question 2

Second, what are the effects of the new structure on data storage? In order to work with the network telescope data, it has to be in an uncompressed format. From what is shown in Figure 7, it might seem that if one were to use the columnar data structure, more disk space would be used whilst processing the captured data. However, due to the column's division into chunks, smaller subsets of packets could be decompressed at a given time, thereby reducing overall system memory requirements. As can be seen in Figure 8, the new structure uses approximately 12% fewer bytes per packet on average when compressed using the gzip compression algorithm. When taken over the terabytes of data that are usually captured by network telescope, even this 12% presents a significant increase in storage efficiency.

6.3 On the main research question

Third, how feasible is the use of the proposed data structure for use in network telescope analysis? Taking into consideration the answers to the two sub-research questions, it is safe to say that new structure definitely has applications for certain research topics. Specifically, research topics which deal mostly with aggregation tasks. Next to this, there are also applications when it comes to long-term packet storage, as is illustrated by the size reduction when the data is compressed using run-length encoding schemes. Despite these benefits, the structure is currently not ready for research, as it does not integrate with existing analytical tools.

6.4 Further work

6.4.1 Prototype optimisations. As shown in Figure 5, methods using the new structure are approximately twice as slow as the ones that use PCAP files. While this is a substantial difference, it may be overcome by further optimisations in the reading and writing algorithms. Next to this, as was mentioned in Section 5.1.3, the prototype used for evaluation does not take advantage of the potential parallelism offered by dividing columns into several chunk files.

6.4.2 Better compression algorithms. Alongside possible improvements to processing speed, ones which target file sizes are important as well. Even though using gzip compression on the new format results in a 12% reduction in disk usage, one may question if it truly the best option for this purpose. This is because the characteristics of fields may differ. For example, the IPv4 header's *identification* field exhibits higher entropy than its *version* field. Therefore it may be better to determine specifically which fields should be compressed, and which should not. However, the gzip algorithm produces variable-length encodings, which are not compatible with the concept of accessing arbitrary values. Therefore, a solution using fixed-length Huffman coding may be able to greatly reduce file sizes, while preserving random-access capabilities.

6.4.3 Applications in network analysis. While this paper has only explored the new structure's effect on the analysis of network telescope data, there may display similar effects when applied to the analysis of general network traffic. This idea stems from the fact that general network traffic analysis is subject to similar problems, such as efficient data storage. Although it is out of the scope of this research, the applicability of the new structure in this broader field warrants further exploration.

REFERENCES

[1] 1981. Internet Protocol. https://datatracker.ietf.org/doc/html/rfc791. https://doi.org/10.17487/RFC0791
[2] Ruo Ando. 2018. Asura: A huge PCAP file analyzer for anomaly packets detection using massive multithreading. DEF CON. https://doi.org/10.5446/39733 https://doi.org/10.5446/39733.
[3] Eray Balkanli and A. Nur Zincir-Heywood. 2014. On the analysis of backscatter traffic. In *39th Annual IEEE Conference on Local Computer Networks Workshops*. 671–678. https://doi.org/10.1109/LCNW.2014.6927719
[4] Zhen Chen, Lin-yun Ruan, Jun Li, Shuai Ding, Fu-ye Han, Hang Li, and Wen-yu Dong. 2014. A Survey on Internet Traffic Archival Systems. (2014).

[5] Benoît Claise. 2004. Cisco Systems NetFlow Services Export Version 9. RFC 3954. https://doi.org/10.17487/RFC3954
[6] Peter Desnoyers and Prashant Shenoy. 2007. Hyperion: High Volume Stream Archival for Retrospective Querying. 45–58.
[7] Wesley Eddy. 2022. Transmission Control Protocol (TCP). RFC 9293. https://doi.org/10.17487/RFC9293
[8] Sven Fiergolla and Petra Wolf. 2021. Improving Run Length Encoding by Preprocessing. arXiv:2101.05329 [cs.DS] https://arxiv.org/abs/2101.05329
[9] Max Gao, Ricky Mok, Esteban Carisimo, Eric Li, Shubham Kulkarni, and kc claffy. 2024. DarkSim: A similarity-based time-series analytic framework for darknet traffic. In *Proceedings of the 2024 ACM on Internet Measurement Conference*. 241–258.
[10] Guy Harris and Michael Richardson. 2025. *PCAP Capture File Format.* Internet-Draft draft-ietf-opsawg-pcap-05. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-opsawg-pcap/05/ Work in Progress.
[11] Russ Housley, John Curran, Geoff Huston, and David R. Conrad. 2013. The Internet Numbers Registry System. RFC 7020. https://doi.org/10.17487/RFC7020
[12] Goran Nikolic, Bojan Dimitrijević, Tatjana Nikolic, and Mile Stojcev. 2022. Fifty years of microprocessor evolution: from single CPU to multicore and manycore systems. *Facta universitatis - series: Electronics and Energetics* 35 (01 2022), 155–186. https://doi.org/10.2298/FUEE2202155N
[13] Ruoming Pang, Vinod Yegneswaran, Paul Barford, Vern Paxson, and Larry Peterson. 2004. Characteristics of internet background radiation. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement* (Taormina, Sicily, Italy) *(IMC '04)*. Association for Computing Machinery, New York, NY, USA, 27–40. https://doi.org/10.1145/1028788.1028794
[14] Urban Sedlar. 2022. Network telescope: insights from a decade of observations. *Electrotechnical Review/Elektrotehniski Vestnik* 89, 4 (2022).
[15] Raffaele Sommese, KC Claffy, Roland van Rijswijk-Deij, Arnab Chattopadhyay, Alberto Dainotti, Anna Sperotto, and Mattijs Jonker. 2022. Investigating the impact of DDoS attacks on DNS infrastructure. In *proceedings of the 22nd ACM Internet Measurement Conference*. 51–64.
[16] Michael Tüxen, Fulvio Risso, Jasper Bongertz, Gerald Combs, Guy Harris, Eelco Chaudron, and Michael Richardson. 2025. *PCAP Next Generation (pcapng) Capture File Format.* Internet-Draft draft-ietf-opsawg-pcapng-03. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-opsawg-pcapng/03/ Work in Progress.
[17] Jean-Pierre van Riel and Barry Irwin. 2006. Identifying and Investigating Intrusive Scanning Patterns by Visualizing Network Telescope Traffic in a 3-D Scatter-plot. 1–12.
[18] Deepak Vohra. 2016. *Apache Parquet.* Apress, Berkeley, CA, 325–335. https://doi.org/10.1007/978-1-4842-2199-0_8
[19] Syed Wali, Yasir Farrukh, Irfan Khan, and Nathaniel Bastian. 2024. Meta: Toward a Unified, Multimodal Dataset for Network Intrusion Detection Systems. *IEEE Data Descriptions* PP (01 2024), 1–8. https://doi.org/10.1109/IEEEDATA.2024.3482286
[20] Mehdi Zakroum, Abdellah Houmz, Mounir Ghogho, Ghita Mezzour, Abdelkader Lahmadi, Jérôme FranÇois, and Mohammed El Koutbi. 2018. Exploratory Data Analysis of a Network Telescope Traffic and Prediction of Port Probing Rates. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 175–180. https://doi.org/10.1109/ISI.2018.8587323
[21] Sebastian Zander, Grenville Armitage, and Philip Branch. 2007. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys Tutorials* 9, 3 (2007), 44–57. https://doi.org/10.1109/COMST.2007.4317620

A AI STATEMENT

During the preparation of this work, I used ChatGPT to answer questions about unfamiliar tools (the Rust programming language and the Linux operating system), to partially generate the abstract, and to perform spell/grammar checking. After using this tool/service, I thoroughly reviewed and edited the content as needed, taking full responsibility for the final outcome.

B READ AND WRITE ALGORITHMS

```
def read_value(i: int, col: Column, chunk_size: int):
    chunk_id = i % chunk_size # the relevant chunk's number
    chunk_file = open_file(col.name + chunk_id) # opens the relevant chunk file

    start_byte = i % chunk_size * col.width // 8 # position of the first byte of the value
    start_bit = i % chunk_size * col.width % 8 # position of the first bit of the value

    out = 0 # value to be returned
    bits_left = col.width # number of bits left to read

    # errant_bits represents the number of bits of the first byte of the read value
    # that are shared with the value before it.
    errant_bits = (8 - start_bit) % 8

    if errant_bits != 0: # checks whether the value shares its first byte with the last byte of the previous value.
        # The read byte is truncated to 'errant_bits' bits, then shifted into place, and added to the output value.
        out |= chunk_file.read_byte() & ((1 << errant_bits) - 1)
        bits_left -= errant_bits

    while bits_left > 0: # repeats until the entire value has been read
        # The read byte is truncated to 'errant_bits' bits, then shifted into place, and added to the output value.
        # signed_shl shifts the first argument to the left by a number of bits that is determined by the
        # second argument. If the second argument is negative, a shift to the right is performed instead.
        out |= signed_shl(chunk_file.read_byte(), bits_left - 8)
        bits_left -= 8

    return out

def write_value(value: int, col: Column, chunk_size: int):
    # it is assumed that value is within [0, 2^col.width - 1]

    chunk_id = i % chunk_size # the relevant chunk's number
    chunk_file = open_file(col.name + chunk_id) # opens the relevant chunk file

    start_bit = i % chunk_size * col.width % 8 # position of the first bit of the value
    bits_left = col.width # number of bits of the value left to be written

    if start_bit != 0: # checks whether the new value shares its first byte with the previous value

        errant_bits = 8 - start_bit # number of bits available in the last byte

        # The last byte is read, and the value's first 'errant_bits' bits are added to the end of it.
        last_byte = chunk_file.read_last_byte()
        last_byte |= signed_shr(value, bits_left - errant_bits) & ((1 << errant_bits) - 1)
        chunk_file.write_last_byte(last_byte)

        bits_left -= errant_bits

    file.go_to_end() # sets the write pointer to the file's end
    while bits_left > 0: # repeats until the entire value has been writen
        # The correct eight bits of the value are shifted into place, and are then appended to the file.
        # signed_shr shifts the first argument to the right by a number of bits that is determined by the
        # second argument. If the second argument is negative, a shift to the left is performed instead.
        chunk_file.write(signed_shr(value, bits_left - 8) % 255)
        bits_left -= 8

```

Fig. 9. Algorithms for value reading and writing