2024 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS) | 979-8-3503-9172-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ANTS63515.2024.10898859

Computing in Transit to Identify Rare Events in Streaming Scientific Data

Ganesh C. Sankaran
Information Sciences Institute
Los Angeles, CA, USA
gsankara@isi.edu

Joaquin Chung
The University of Chicago
Chicago, IL, USA
chungmiranda@uchicago.edu

Rajkumar Kettimuthu
Argonne National Laboratory
Lemont, IL, USA
kettimut@anl.gov

***Abstract*—Programmable networks, aside from carrying out their core network functions, can look deep into the data stream and perform application layer processing. But, expect for a few demonstrations, this capability remains largely under explored and under utilized. Currently, scientific computing leverages networks only for communication and not for computation. We propose Computing in Transit to unleash the potential of network computing for scientific workflows. Specifically, we investigate computing in transit in the context of light source experiments. Researchers using light sources are interested in rare events and we intend to leverage computing in transit to solve this problem. As the compute and memory resources available within the network are scarce, we must use these resources prudently without sacrificing on performance metrics. Computing within the network can support significantly higher throughput at low latency but it may be less accurate as there are limitations to how deep a network can inspect the payload. We propose a neutralized checksum that takes in TCP checksum as an input to avoid processing the entire payload. We evaluate this approach to identify rare events by introducing random perturbations to reference frames. We measure the effectiveness of neutralized checksum to identify changes. We see that neutralized checksum identifies all changes and is a very promising approach to rare event detection.**

I. Introduction

Scientific workflows continuously monitor a phenomenon and generate huge amount of raw data. Researchers on the other hand are interested in streams that carry regions of interest. For instance, light source facilities such as Argonne's Advanced Photon Source (APS) [1], encompass a diverse set of scientific workflow in which researchers are more interested in changes in data rather than the raw data. Current state-of-the-art in scientific workflow (see left hand side of Fig. 1) involves moving data to edge resources for preprocessing, and then to remote Cloud / High-performance Computing (HPC) resources for large-scale processing and storage. We envision that scientific workflows will be able to compute and distribute data at optimal points along the network path (see right hand side of Fig. 1) by leveraging programmable networks.

Programmable networks provide benefits in terms of processing huge volumes of data, while in-network computing is the process of offloading operations from end hosts into networking devices (e.g., switches, routers, or smart NICs) [2]. When near-network computing resources are also used, we can call this paradigm in-transit computing (see Figure 2). Since the initial work on in-network computing [3], there has been a lot of research interest in this area. Still, realising a scientific workflow is highly challenging. There are only a handful of closely related works at the intersection of scientific computing and in-network computing. Relevant work in this space attempted to solve two important problems: (i) providing an unified interface across heterogeneous network resources [4] and (ii) supporting floating point and logarithmic math function on programmable switch hardware [5]. Recently, Patel et al. [6] realized a richer set of math functions using Taylor series approximation.

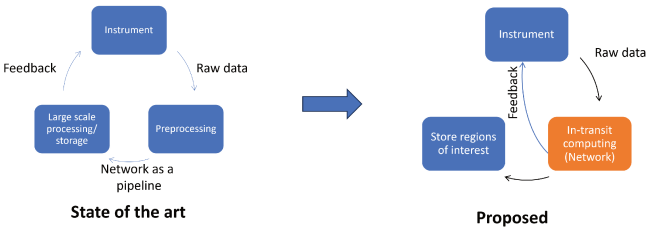

Fig. 1. Scientific Workflow computing In-Transit: Network computes on streaming data in the proposed system.

In this paper, we take another step towards in-transit computing by solving the rare event detection problem for streaming scientific data. We present a preliminary study to evaluate the TCP checksum as change detection mechanism. Checksums have been used to ensure integrity and for change detection. In this case, we propose the use of transport layer checksums for change detection. The primary advantage of this approach is that this solution can be programmed within a network switch. Due to the computing and memory resource limitations, we compute neutralized checksums from transport layer checksums. Our initial results look promising and we propose to carry out a detailed evaluation of the proposed checksum with possible use cases where it is suitable and where it is not.

II. Motivation and Challenges

Broadly, scientific workflows involving physical phenomenon with a sense of continuity have additional clues embedded in raw data. For instance, a perturbation of a point on the surface of an object impacts nearby space and is to be seen in subsequent monitoring cycles. This change is continuously visible both in time and space until it disappears. We will

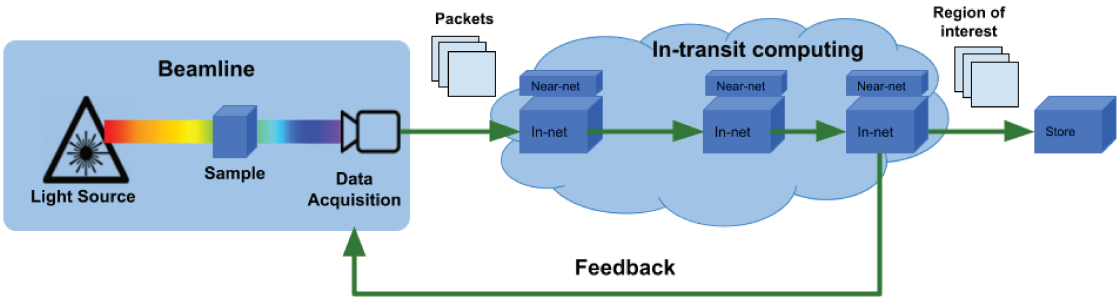

Fig. 2. In-transit system for a light source facility use case.

exploit these characteristics for solving the change detection problem in transit. We consider a light source facility use case to motivate this work, specifically the High-Energy Diffraction Microscopy (HEDM) type of experiment. HEDM experiments could be described as follows: a typical experiment comprises 16 layers, with each layer containing four scans. A scan refers to the collection of angular projections of an object, while a projection refers to an image snapshot obtained by illuminating the object at a specific angle. Each scan captures 1440 frames, where every frame is an image with a resolution of 2048×2048 pixels. In the analysis phase of HEDM data, the primary focus is on extracting patches that feature a singular diffraction peak, crucial for the 3D microstructural mapping of materials. Typically, each diffraction peak spans across 2-3 frames within a single scan. Following the acquisition phase, where each scan takes approximately 6-7 minutes to produce a raw image, the patch extraction process is completed within 1-2 minutes. It is within these extracted patches that the singular peak (persisting for a few frames) provides vital insights into the material's microstructure.

A programmable switch provides a high-speed, stateful processing pipeline that can operate on Gbps to Tbps of data. Packet data and its metadata is available throughout the processing pipeline. While, the pipeline operates on the packet headers and metadata, it is not capable of operating on the complete packet payload. Depending on the OEM vendor design, a small portion (e.g., 200 to 500 bytes) of payload can be parsed out as a header and processed by the pipeline. In comparison with the Jumbo packet size (i.e., 8000+ bytes), this is significantly small.

One of the key challenges when realizing a scientific workflow is the limited pipeline resources. The number of pipeline stages and resources available per pipeline stage are limited. Flight Plan [7] and DINC [8] extended the pipeline beyond a single network switch to address this limitation. Moreover, Kim et al. [9] outline the challenges in processing high-dimensional data used in scientific computing. Finally, transport is another key challenge. Traditionally, TCP based transport was defined in an immutable payload regime and it is sensitive to deflation or compression of payloads as a result of processing as pointed out by Stephens et al. [10].

III. Approach

In this paper, we use the terms payload and application data interchangeably. In the current use case, a checksum on payload is desired. While both IP and TCP headers have checksum fields, the IP header checksum is not a function of the packet payload, and does not indicate changes in payload. On the contrary, the TCP checksum takes the payload as its input. However, this checksum also takes additional inputs such as source and destination IP addresses, source and destination TCP ports, sequence numbers, and other TCP fields to compute its checksum. These header fields change significantly across connections and within the same connection, leaving an undesired impact on the TCP checksum for our change detection objective.

Next, we argue that the option of computing a checksum on payload within the data plane is infeasible. Programmable switches have limitations in terms of resources and payload visibility as mentioned in Section II. A switch data plane has lookup tables, limited pipeline stages, registers and arithmetic operation support. In short, the resources are limited for computing a checksum on the payload.

When arguing about resource limitation, the question on whether any checksum can be computed at all within the data plane arises. TCP checksum involves the entire payload whereas IPv4 checksum does not. IPv4 header gets updated at every router hop. This update involves decrementing TTL, thus IPv4 header checksum is recomputed on IPv4 header fields. This operation is limited to 20-byte header. On the other hand, a TCP checksum involves the entire payload, so the data plane has the capability to compute checksums such as IP header checksum that involve handful header fields but it is not capable of computing checksums on the entire payload such as TCP checksum or a custom checksum on payload as we desire.

A. Neutralized TCP Checksum

Having presented the challenges, here, we present a neutralized checksum computation algorithm $\mathcal{N}(p)$. We know that TCP checksum takes payload along with pseudo IP header fields and entire TCP header as its input. We observe two key properties of this checksum: it is conditionally commutative

and reversible. A commutative operation does not depend on the permutation of input but on the combination of input.

TCP checksum operation is not commutative on fields that are less than 2 bytes. TCP header contains TCP flags, reserved and data offset fields that are less than 2 bytes. When, we change the permutation of these fields, the resulting TCP checksum is not the same as the checksum computed by the IP/TCP protocol stack. In short, we cannot change the permutation of these fields, they must be rolled up into a 2-byte virtual field in the same sequence as they appear in the TCP header. However, all fields that align to a 2-byte boundary are commutative. For instance, TCP checksum computed considering IP source before IP destination is equal to the checksum computed when IP destination is considered before IP source.

Next, we observe that TCP checksum is reversible. We demystify the TCP checksum computation to reinforce this observation. TCP checksum computation breaks down the entire TCP header, pseudo IP header, and payload into 2 byte chunks. It then adds all these chunks with carry. Then reduces the final value to 2 bytes by adding the carry to the 2 byte least significant bits. Then, a one's complement is computed on this and taken as TCP checksum. Since this involves add operations and one's complement, TCP checksum computation is reversible.

The proposed algorithm uses commutative and reversible properties to neutralize the impact of IP/TCP header fields on a packet's TCP checksum. The neutralized checksum on payload, $\mathcal{N}(p) = f(c, i, t)$, where p is the payload, c is the current TCP checksum, i is the pseudo IP header, t is the TCP header and $f(*)$ is the function of c, i, and t. Neutralization involves more number of fields as compared to an IPv4 header checksum but is still within the resource limitations of a switch's data plane.

IV. Evaluation

In this section, we evaluate the effectiveness of using a neutralized TCP checksum implemented in P4 [11]. P4 is a domain specific language that provides a handle on parser, deparser, processing pipeline of a network switch. We use the processing pipeline resources to implement the neutralized TCP checksum. After forwarding decision is taken on the packet, we check whether this packet has a TCP header. Then compute the neutralized checksum on this packet. Neutralized TCP checksum implementation involves a series of add operations on the header fields and then the final operation involves current TCP checksum.

The number of reference packets (r), packet length (l), number of patches to be introduced in the packet (n), patch size (s) are the evaluation parameters. Here, r and l are integers, whereas $n \in [0, 1, 2, 3]$ and $s \in [0, 3, 4, 5]$ are chosen randomly. The domain for n and s were chosen from the HEDM use case where patch size is anywhere between 3×3 and 5×5 pixels. We generate random reference packets of specified length l that carry the background data. Subsequent packets are generated from these reference packets based on n and s. For instance, when $n = 3$, three random patch sizes of s are generated. For every patch, a random patch position is chosen. At this random position, a randomly generated patch of size s is applied. This results in a series of modifications to the reference packet at random positions. Finally, the modified packet is sent over. When zero is chosen as patch size or patch number, no patch is applied on the packet.

At the P4 switch, a neutralized TCP checksum $\mathcal{N}(p)$ is computed. This indicates whether the data is changed or not. When there is a change in data, rare event is identified. This will help the scientific workflow in identifying rare events while at the same time discarding uninteresting events from the streaming data.

A. Metrics and objective of evaluation

Checksum collisions are possible and there is chance that a rare event goes unnoticed. The primary objective is to measure the false negatives associated with the neutralized TCP checksum. In addition, false positives increase the noise in the scientific data stream. The secondary objective is to minimize the false positives.

B. P4 Implementation

We implemented the neutralized checksum on P4 BMv2 as shown in code listing 1.

Listing 1. P4 neutralized checksum source code.

```
action neutralize() {
    bit<32> isum =   ((bit<32>)(hdr.ipv4.srcAddr >> 16 & 0xFFFF))
        + ((bit<32>)(hdr.ipv4.srcAddr & 0xFFFF))
        + ((bit<32>)(hdr.ipv4.dstAddr >> 16 & 0xFFFF))
        + ((bit<32>)(hdr.ipv4.dstAddr & 0xFFFF))
        + ((bit<32>)(hdr.tcp.seqNo >> 16 & 0xFFFF))
        + ((bit<32>)(hdr.tcp.seqNo & 0xFFFF))
        + ((bit<32>)(hdr.tcp.ackNo >> 16 & 0xFFFF))
        + ((bit<32>)(hdr.tcp.ackNo & 0xFFFF))
        + ((bit<32>)hdr.tcp.srcPort
        + (bit<32>)hdr.tcp.dstPort
        + (bit<32>)hdr.tcp.others)
        + (bit<32>)(hdr.tcp.window + hdr.tcp.urgentPtr)
        + ((bit<32>)(hdr.ipv4.protocol & 0x00FF))
        + ((bit<32>)hdr.ipv4.totalLen - ((bit<32>)hdr.ipv4.ihl<<2));
    bit<16> sum = hdr.tcp.checksum
        + (bit<16>)((isum >> 16) & 0xFFFF)
        + (bit<16>)(isum & 0xFFFF);
}
```

A set of ten r randomly generated reference packets with different lengths $l = 500, 1500, 8000$ were used. These lengths correspond to average packet size on the Internet, MTU of Ethernet and Jumbo packet size respectively. Then, these reference packets were used to generate more than 3000 random packets with and without patches based on the randomly chosen number of patches n, patch size s and patch positions. The parameters used to create the packet such as total patch size (shown in Fig. 3) and the number of patches used to modify the packet were encoded into the packet to compute false positives and false negatives.

C. Results and discussion

Out of 3030 packets, 2024 packets were modified and the rest unmodified. This is more than the number of rare events that takes place in our scientific use case and evaluation must be treated as a worst case analysis.

All patches and hence all rare events were identified, So, Fig. 3 also presents the distribution of patches by total size identified by the neutralized checksum algorithm.

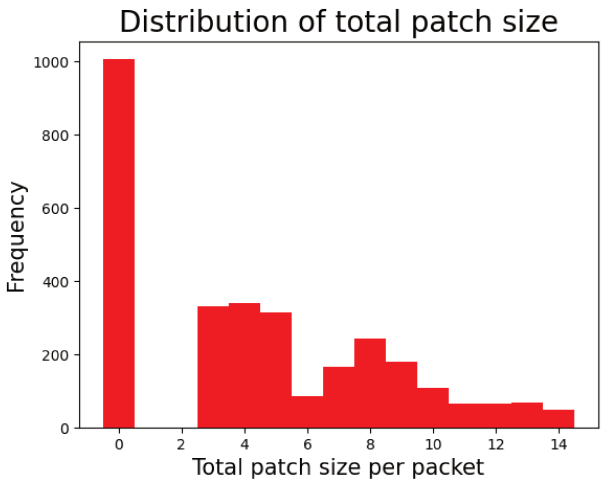

Fig. 3. Distribution of total patch size per packet.

TABLE I
EFFICIENCY OF NEUTRALIZED CHECKSUM

		Rare?	
		Y	N
Identified?	Y	2024	0
	N	0	1006

With random patches, neutralized TCP checksum indicated a change in all 2024/2024 cases and identified rare events. In the unmodified cases, it identified no rare events. No false positives or false negatives were observed. In 25/3030 cases, checksum collision was observed across different references. For the same reference, two checksum collisions were observed. When, collision is non-zero, there is a non-zero possibility that the light source moves from one random state to another but will go unnoticed. We need additional mechanisms to ensure that this situation does not occur.

To conclude, neutralized checksum in this initial study looks as a promising option for identifying rare events. We observe non-zero collisions and detailed research is required to reduce the collisions.

V. CONCLUSION

In this paper, we proposed the application of computing in transit to scientific workflows. We chose HEDM scientific use case where researchers are continuously looking for rare events. When data is still in transit, we proposed the use of programmable data plane switches to identify rare events. Since switches lack the capability to process entire payloads, we proposed a neutralized checksum to identify changes in data. We evaluated our approach in P4 and were able to find all change instances successfully. We also observed that checksum collision is a non-zero probability incident. This affects the accuracy of the approach. We conclude that the neutralized checksum approach is promising in terms of efficiency in identifying changes.

In future, we intend to study multiple checksums, leveraging limited payload visibility and processing payload by progressively chopping off a portion of data. In case of doubt, we intend to study the use of near-network resources to cross-verify.

TABLE II
NUMBER OF PATCHES APPLIED TO A PACKET AND HOW MANY WERE IDENTIFIED

n	Identified
0	1006/1006
1	624/624
2	656/656
3	744/744

REFERENCES

[1] "Advanced Photon Source," https://www.aps.anl.gov.
[2] N. Zilberman, "In-network computing," https://www.sigarch.org/in-network-computing-draft/.
[3] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, pp. 150–156.
[4] G. C. Sankaran, J. Chung, and R. Kettimuthu, "App2net: Moving application functions to network & a case study on low-latency feedback," in *2022 IEEE/ACM International Workshop on Innovating the Network for Data-Intensive Science (INDIS)*, 2022, pp. 1–8.
[5] ——, "Leveraging in-network computing and programmable switches for streaming analysis of scientific data," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE, 2021, pp. 293–297.
[6] S. Patel, R. Atsatsang, K. M. Tichauer, M. H. Wang, J. B. Kowalkowski, and N. Sultana, "In-network fractional calculations using p4 for scientific computing workloads," in *Proceedings of the 5th International Workshop on P4 in Europe*, 2022, pp. 33–38.
[7] N. Sultana, "Leveraging in-network application awareness," in *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*, 2021, pp. 63–67.
[8] C. Zheng, H. Tang, M. Zang, X. Hong, A. Feng, L. Tassiulas, and N. Zilberman, "Dinc: Toward distributed in-network computing," *Proceedings of the ACM on Networking*, vol. 1, no. CoNEXT3, pp. 1–25, 2023.
[9] D. Kim, A. Jain, Z. Liu, G. Amvrosiadis, D. Hazen, B. Settlemyer, and V. Sekar, "Unleashing in-network computing on scientific workloads," *arXiv preprint arXiv:2009.02457*, 2020.
[10] B. E. Stephens, D. Grassi, H. Almasi, T. Ji, B. Vamanan, and A. Akella, "Tcp is harmful to in-network computing: designing a message transport protocol (mtp)," in *Proceedings of the 20th ACM Workshop on Hot Topics in Networks*, 2021, pp. 61–68.
[11] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, July 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890