Extracting TCPIP Headers at High Speed for the Anonymized Network Traffic Graph Challenge

Zhaoyang Han*, Andrew Briasco-Stewart‡, Michael Zink†, Miriam Leeser*
Northeastern University *‡, University of Massachusetts Amherst †
Email: *zhhan,mel@coe.neu.edu, ‡briasco-stewart.a@northeastern.edu
†mzink@umass.edu,

***Abstract*—**
Field Programmable Gate Arrays (FPGAs) play a significant role in computationally intensive network processing due to their flexibility and efficiency. Particularly with the high-level abstraction of the P4 network programming model, FPGA shows a powerful potential for packet processing. By supporting the P4 language with FPGA processing, network researchers can create customized FPGA-based network functions and execute network tasks on accelerators directly connected to the network. A feature of the P4 language is that it is stateless; however, the FPGA implementation in this research requires state information. This is accomplished using P4 externs to describe the stateful portions of the design and to implement them on the FPGA using High-Level Synthesis (HLS). This paper demonstrates using an FPGA-based SmartNIC to efficiently extract source-destination IP address information from network packets and construct anonymized network traffic matrices for further analysis. The implementation is the first example of the combination of using P4 and HLS in developing network functions on the latest AMD FPGAs. Our design achieves a processing rate of approximately 95 Gbps with the combined use of P4 and High-level Synthesis and is able to keep up with 100 Gbps traffic received directly from the network.

***Index Terms*—FPGA, P4, packet capture, in-network processing, anonymized network traffic**

I. Introduction

Large-scale network problems represent some of the most pressing challenges in our increasingly connected world. These problems encompass a variety of complex issues related to the design, analysis, and optimization of vast networks that facilitate global communication and data exchange. As networks continue to grow in size and complexity, addressing these problems becomes crucial for ensuring efficient, reliable, and secure operations. A significant bottleneck in the current network infrastructure is its inability to process large network traffic while maintaining the flexibility needed to adapt to varying demands and conditions.

The MIT Graph Challenge [15] is an important platform for tackling large-scale network problems. By providing a structured competition focused on innovative graph analytics, the challenge encourages researchers and practitioners to develop cutting-edge solutions. Participants engage with real-world datasets and complex graph problems, driving advances in algorithms and technologies that can be applied to optimize large-scale networks. In particular, network researchers use a data product of anonymized source-to-destination traffic matrices derived from billions of real network packets to analyze and solve real large-scale network problems.

This work was funded by National Science Foundation (NSF) grants CNS-1925464, CNS-1925658, CNS-2130891, and 2130907. All opinions and statements in the above publication are of the authors and do not represent NSF positions.

The Anonymized Network Sensing Graph Challenge [8] is part of the Graph Challenge that aims to find optimized and highly efficient approaches in the construction of anonymized traffic matrices from network traffic. As part of the Anonymized Network Sensing Graph Challenge, we proposed a novel approach based on the Programming Protocol-independent Packet Processors (P4) language combined with the implementation on Field Programmable Gate Arrays (FPGAs).

FPGAs have demonstrated significant potential in computationally intensive network processing, especially when they are directly connected to the network [14]. Their ability to offload compute-intensive tasks and support the disaggregation of data centers makes them valuable in addressing the demands of modern network infrastructures. FPGAs can be directly connected to networks, enhancing their role in processing and managing large volumes of data efficiently. In addition, the reconfigurability of FPGAs allows them to be programmed and reprogrammed to perform specific tasks. This flexibility is particularly beneficial for network processing, where requirements can change rapidly. FPGA-based network devices can be tailored to accelerate various networking functions, such as packet filtering, encryption, and deep packet inspection, and can be updated as new protocols and standards emerge. The P4 language is a high-level abstraction for network devices that allows the programmer to describe how network packets should be parsed, processed, and forwarded. The combination of the P4 language and FPGA implementation allows network researchers to easily develop their own FPGA-based network functions [3], [10].

In this research we focus on constructing the anonymized data structure of network traffic as described in the Anonymized Network Sensing Graph Challenge. This innovative solution makes use of FPGAs programmed with P4 to form a powerful approach to match the line rate of the high-speed network. Specifically, we demonstrate the use of an FPGA-based SmartNIC that is able to perform tasks for the Anonymized Network Sensing Graph Challenge at approximately 95Gbps rate.

The contributions of the work presented in this paper are to:

- Describe the design flows of FPGA-based SmartNICs in developing high-throughput network functions using P4. In particular, this paper shows the first example of the combination of using P4 and High-level Synthesis in developing network functions on the latest AMD FPGAs.
- Demonstrate innovative FPGA-based SmartNIC solutions for the Anonymized Network Sensing Traffic Graph Challenge.
- Implement and test the solution with the CAIDA dataset on the public Open Cloud Testbed (OCT), achieving high-throughput processing at 95Gbps.

The rest of this paper is organized as follows. In Sec. II we introduce the Anonymized Network Sensing Traffic Graph Challenge problem, discuss using the P4 language for programming FPGAs, and describe the Open Cloud Testbed (OCT) that is used for these experiments. Our design is described in Sec. III and results are presented is Sec. IV. We present a discussion of future directions for incorporating this design into the rest of the graph challenge problem in Sec. V and conclude the paper with a summary of our design, evaluation and future directions.

II. Background

In this section, we introduce the details of the Anonymized Network Traffic Graph Challenge problem, the FPGA-based P4 SmartNICs used in our implementation, the Open Cloud Testbed (OCT) as well as discussing related research.

A. Anonymized Network Sensing Challenge

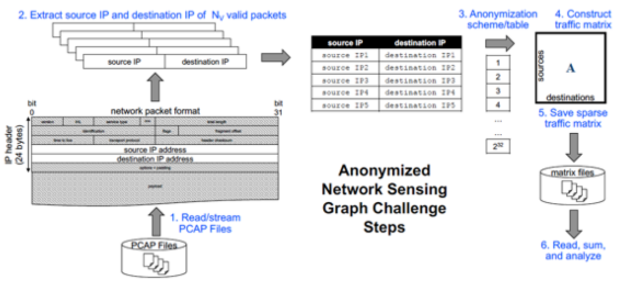


Fig. 1. Anonymized Network Traffic Graph Challenge Steps. Figure adapted from [8].

The Anonymized Network Traffic Graph Challenge aims to extract source and destination information from real network traffic, as shown in Fig. 1. Our solution focuses on the first two steps. To simulate real network traffic, participants are encouraged to use network traces captured in real networks for the evaluation. To test the performance of the flow, we use the realistic network traffic from the Center for Applied Internet Data Analysis (CAIDA) [5].

B. P4

The P4 language is a domain-specific language to describe how network devices like switches, Network Interface Cards (NICs), routers, filters, etc. should process packets. This device-agnostic high-level abstraction provides network researchers an opportunity to design their network functions without considering hardware devices.

The evolution of the P4 language [4] and the advent of programmable network devices have significantly enhanced the user's ability to program both control and data planes. This progress has enabled extensive research in areas such as data plane disaggregation, cryptography, and data plane machine learning. Several P4 programmable devices are available including AMD Pensando, Intel Tofino Switch, NVIDIA Bluefield2 SmartNICs, etc. FPGAs are a key technology to empower the performance of packet processing in the data plane [17]–[19], and can be programmed with the P4 language, as described in the next section.

C. FPGAs as Network Devices

There is a trend to attach more devices directly to the network to improve data access and decrease latency. Modern applications, including machine learning [6], require a large amount of data that can be more efficiently accessed directly from the network. SmartNICs are gaining popularity as a packet processing platform and can be programmed using the P4 programming model.

FPGA-based SmartNICs offer superior programmability and customization compared to dedicated P4 SmartNICs. Compared to AMD's Pensando Distributed Services Card (DSC-200), which has a fixed data flow and lacks flexibility, SmartNICs based around FPGA programming can be customized and scaled to address specific requirements. Firestone et al. [9] demonstrated this advantage by developing and deploying FPGA-based SmartNICs on Azure, effectively offloading network functions. Their results show that FPGAs excel at dynamically managing network traffic, alleviating the main CPU processing burden.

We have created a framework for developing and testing network functions on P4-based FPGA SmartNICs using the Open Cloud Testbed, a public research platform described in Sec. II-D. This P4-based framework offers a more efficient method for describing network behaviors on FPGAs than traditional Hardware Description Languages (HDLs) such as Verilog and VHDL. While FPGAs are ideal for developing P4 applications due to their customizable pipelines, the P4 model does not fully leverage the advantages of FPGA parallel programming. Our framework extends the original P4 model with the extra ability for concurrent processing using the `extern` function, a P4 language construct. Such extern functions can be described in HDL or C++ and translated to the FPGA fabric using either a standard tool flow or High-Level Synthesis (HLS). We utilize the P4 framework in OCT to develop our solution for the Graph Challenge due to its ease of use and high performance.

D. Open Cloud Testbed

The Open Cloud Testbed (OCT) [20] provides a research-oriented experimentation testbed for systems researchers who focus on cloud platforms. Testbeds like OCT deliver the

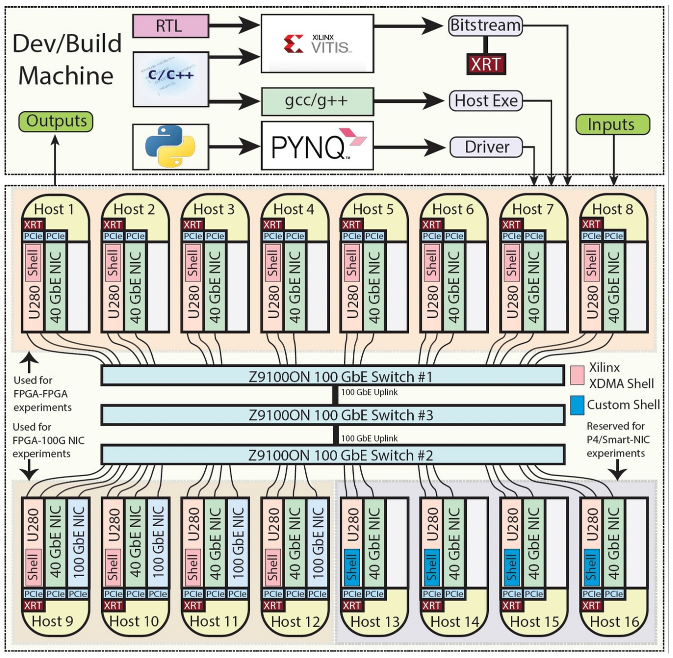


Fig. 2. Overview of OCT FPGA development and workflow [20]

necessary hardware and software on top of bare metal services to researchers in both the cloud and system communities, enabling more experimental-based research.

OCT currently offers 32 FPGAs to the research community: 24 AMD Xilinx Alveo U280, 4 AMD Xilinx VCK5000, and 4 AMD Xilinx V70. Each of these FPGAs is housed in a host server that an experimenter can allocate as a bare metal machine. 28 (all U280s and VCK5000s) of the 32 FPGAs have two direct network links, which connect both of their QSFP28 (100GbE) interfaces to a switch for a combined maximum bandwidth of 200 Gbps [11]. In tandem, OCT offers a toolchain that supports the development of bitstreams that can be deployed on the FPGAs [14].

The OCT workflow consists of two primary stages which are illustrated in Fig. 2. We provide a series of tutorials, example applications, and scripts and profiles for the setup and execution of experiments [16].

Development Stage: OCT development tools are hosted on a virtual machine (VM) within the New England Research Cloud (NERC). OCT users can remotely log into this VM to create FPGA bitstreams, host executables, and drivers using the provided tools. In addition, licenses required for certain Xilinx IPs are hosted on a separate license server. OCT provides several different FPGA configurations that can be used for different research directions and are well-suited to this research.

This basic framework is available on OCT as the HLS acceleration flow. Users can use HLS to develop their compute-intensive accelerators and test them on OCT. Recently, P4-based FPGA support has been made available on OCT, including tutorials and several demonstration examples [10]. The P4 design further supports the potential of FPGAs as programmable network devices.

In this paper, we make use of the P4-based tool flow as the basic framework and combine it with HLS. Specifically, this paper demonstrates the first example of the combination of P4+HLS on the FPGA-based SmartNICs.

Deployment Stage: After creating the bitstreams and host executables/drivers, users transfer them to bare metal servers that host the FPGAs. The subsequent process involves programming the FPGAs, executing the host executables, and optionally fetching the results back to the development machine.

E. Related Work

Several studies have explored the potential of FPGAs in-network processing due to their customizable pipelines and high throughput. Since the inception of P4 [4], [12], various compilers and frameworks have been developed for FPGAs. In 2023, ESNet introduced a framework that integrates the latest AMD FPGA with their VitisNetP4 compiler [7]. Similarly, [10] presented a framework that offers additional features such as support for `extern` functions and partial reconfiguration. In this work, we extend the framework from [10] by utilizing simplified C-based HLS functions as extern functions. For the graph challenge, others have investigated the performance of deploying a solution on the edge with two Accolade Technology ANIC-200Kq dual port 100 gigabit NICs [13]. Their work demonstrates an example of the anonymized networking sensing challenge with high throughput. Our design achieves a comparable throughput as [13]. Similar to this work, we deploy our solution on an FPGA-based NIC. Our work demonstrates comparative results that saturate the 100Gbps link on the FPGA and also showcases a development with state-of-the-art domain-specific language P4.

III. Proposed Design

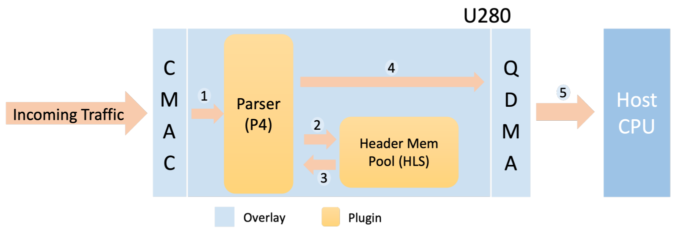


Fig. 3. Design illustration: the numbers shows the order of data flows.

We proposed a design solution for the Anonymized Network Sensing Challenge that focuses on the first two steps shown in Fig. 1. This optimized hardware design provides a high-performance solution to in-network packet processing and header extraction. In this section, we explore the utilization of our P4-based FPGA framework for implementing high-performance packet header extraction. The design consists of an overlay and a plug in. The overlay establishes a shell that facilitates basic connections between physical interfaces and host CPUs, and is common among P4 designs. The plugin houses the P4 application including the extern. The P4 parts handles the packet parsing and deparsing and the HLS parts are used to save the extracted headers.. The framework consisting of the the overlay and plugin is shown in Fig. 3.

A. Overlay Design

The overlay is built on AMD/Xilinx's OpenNIC shell [1], an open-source, FPGA-based 100G NIC platform. The OpenNIC shell features a Queue-based Direct Memory Access (QDMA) block for transferring packets between the NIC shell and the host CPU via the PCIe bus, and a 100G MAC block (CMAC) for Ethernet communication. The user plugin connects the QDMA and CMAC components.

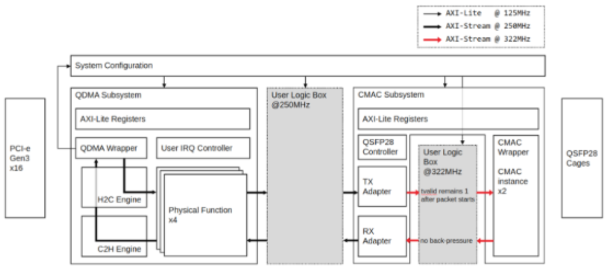


Fig. 4. OpenNIC shell structure. Figure adapted from the OpenNIC shell documentation [1]

As shown in Figure 4, we populate the block labeled *User Plugin@250MHz* with the P4 hardware IP block produced by AMD/Xilinx's Vitis Networking P4 (VitisNetP4) toolchain [2]. This toolchain generates hardware IP from P4 source code. The OpenNIC shell delivers NIC functionality capable of supporting 100Gb/s throughput. Our framework establishes the required packet and control logic pathways between the OpenNIC shell and the outputs from VitisNetP4.

With this approach, users only needs to provide P4 code as input, and can quickly generate a P4-enabled NIC and achieve fast deployment on OCT using this tool flow. It provides an easy way for network researchers unfamiliar with FPGAs or hardware design to conduct their research on OCT. Under this framework, users can easily focus on network functions while the OpenNIC shell provides the development of the NIC design.

B. Header Extraction

The key to achieving the challenge lies in utilizing the plugin. This plugin is a combination of P4 and HLS blocks. The P4 block consists of a parser and a deparser. The parser is employed to extract source and destination IP addresses, while the deparser is used to reconstruct the packets and forward them to the host. A limitation of the P4 processing model is its stateless nature; the extracted header information cannot be stored under the P4 description. For this reason, we use an HLS block as a P4 `extern struct` to describe the behaviors of storing and processing this information on the FPGA. The HLS-based extern function is instantiated in the processing block between the parser and deparser of the P4 code.

We store the extracted headers on the FPGA on-chip Block RAM. Since the traffic matrix is constructed on the host CPU, we pack the information from a fixed number of N_p packets received from the network into one packet and forward it to the host CPU. Similar to a network packet, we need eight bytes for the address of these information packets. Therefore, a fixed size $8 \times N_p$ bytes network packet is sent to the host. It will yield a theoretical packet drop rate $1 \div (N_p + 1)$. Although the larger N_p yields a lower drop rate, it also requires a wider data bus between the P4 block and HLS block to exchange information as $8 \times N_p$ data need to be transferred. A wide data bus will increase the hardware design difficulties and eventually cause a larger packet drop rate than expected. This design accommodates the current FPGA configuration. To further improve the performance and reduce the drop rate, instead of sending data as a network packet, we can establish a direct data link between the HLS block and the host as described in section V.

IV. Implementation and Evaluations

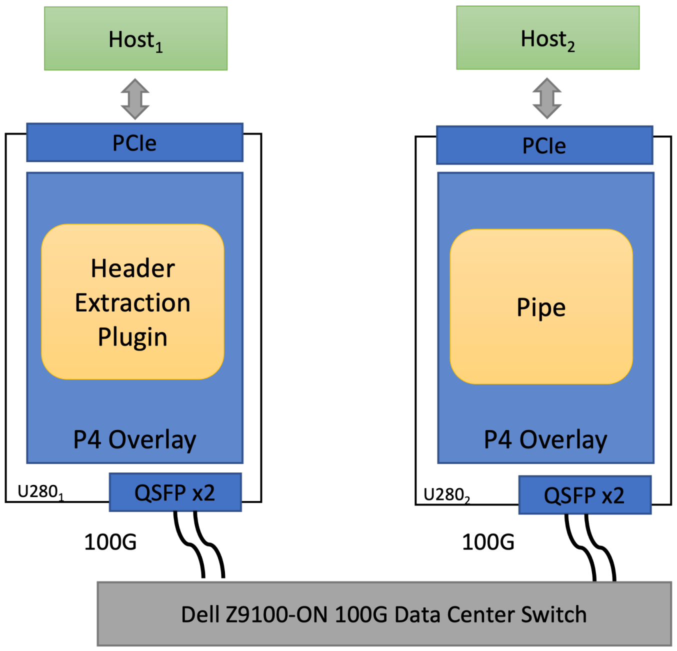


Fig. 5. Testbed Illustration

To evaluate the performance of our solution, we implemented it on OCT, as illustrated in Fig. 5. We deployed our solution on one FPGA node in the OCT as the receiver. To generate a sufficient number of packets for testing, we utilized another U280 FPGA node as the sender. Both nodes are capable of handling 100Gbps traffic through their network connections. On the sender side, with an empty plugin, packets are directly transmitted through the FPGA. To fully utilize the hardware's speed and minimize performance bottlenecks from the host CPU processing rate, we also employed DPDK on both hosts.

Table I shows the performance results of processing different size packets using our implementation. We use the CAIDA Anonymized Internet Traces Dataset [5]. In particular, we use the traces captured by the Equinix-Chicago monitor on

```
...
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t smeta) {
    state start {
        transition parse_eth;}
    state parse_eth {
        packet.extract(hdr.eth);
        transition select(hdr.eth.type) {
            IPV4_TYPE : parse_ipv4;
            default   : accept;}}
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        packet.extract(hdr.ipv4opt, (((bit<32>)hdr
            .ipv4.hdr_len - 5) * 32));
        transition select(hdr.ipv4.protocol) {
            TCP_PROT  : parse_tcp;
            UDP_PROT  : parse_udp;
            default   : accept;}}
    state parse_tcp {
        packet.extract(hdr.tcp);
        packet.extract(hdr.tcpopt, (((bit<32>)hdr.
            tcp.dataOffset - 5) * 32));
        transition accept;}
    state parse_udp {
        packet.extract(hdr.udp);
        transition accept;}
}
...
```

Listing 1. P4-based front-end

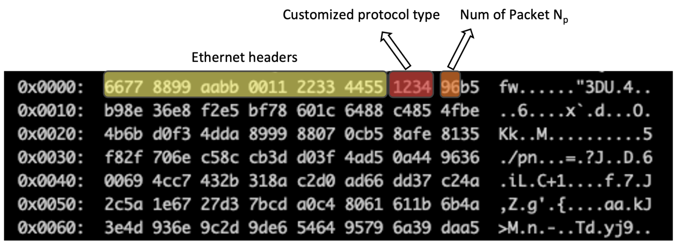
TABLE I
FPGA PROCESSING RATE

Packet Size (Byte)	Data Rate (Mbps)	Packet Rate (pps)
64	27,704	41,210,656
128	47,432	39,790,209
256	77,718	35,098,591
512	94,238	22,428,831
1024	95,759	11,447,680
1518	95,359	7,746,182

high-speed internet backbone links in 2016. These traces only contain the TCP/IP headers. To test on a real network, we pad these packets with zeros to simulate real traffic data. Table I shows that for 512-byte packets, we can fully saturate the 100Gbps link. The packet rate for small packets can be further improved as the theoretical performance of the OpenNIC shell overlay is 100 million packets per second.

Through the use of P4 and HLS, we were able to reduce the effort of developing hardware logic. The FPGA design consists of a few hundred lines of P4 codes in place of thousands of lines of HDL code. The shorter code base makes it easier to develop, debug, and improve our FPGA design. Listing. 1 shows the P4 parser code to extract the headers. If this code were developed in HLS, thousands of lines of detailed logic description would have been required to describe the parsing logic.

The packet that summarizes the information contains an Ethernet header with a customized protocol as shown in Fig. 6. With the help of the P4 deparser, we can easily define and reconstruct any customized protocol. The byte following the header contains the number of source-destination pairs included in the payload, i.e. N_p. In our implementation, we set N_p to 150 in order to transfer as much information as possible. Then we pack the source and destination IP pairs into the payload. For 150 packets, there will be a 1200-byte payload.



```
0x0000:  6677 8899 aabb 0011 2233 4455 1234 96b5  fw......"3DU.4..
0x0010:  b98e 36e8 f2e5 bf78 601c 6488 c485 4fbe  ..6....x`.d...O.
0x0020:  4b6b d0f3 4dda 8999 8807 0cb5 8afe 8135  Kk..M..........5
0x0030:  f82f 706e c58c cb3d d03f 4ad5 0a44 9636  ./pn...=.?J..D.6
0x0040:  0069 4cc7 432b 318a c2d0 ad66 dd37 c24a  .iL.C+1....f.7.J
0x0050:  2c5a 1e67 27d3 7bcd a0c4 8061 611b 6b4a  ,Z.g'.{....aa.kJ
0x0060:  3e4d 936e 9c2d 9de6 5464 9579 6a39 daa5  >M.n.-..Td.yj9..
```

Fig. 6. Received packet example showing the first hundred bytes of the packet.

V. Future Improvements

Our approach, as described in this paper, tackles the first two steps of the Anonymized Network Traffic Graph Challenge, namely extracting source and destination IP addresses at line speed. Our solution makes use of a network connected FPGA to receive and process packets and transmit the extracted information to the host.

For ease of implementation, we gathered source and destination IPs, stored them in memory on the FPGA, and used network packets to transfer this information to the host. We accomplish this by replacing one out of every 150 packets with the collected information. This solution did not require the addition of any additional mechanism to implement host-to-FPGA communication. It also results in one out of every 150 packets of original data being dropped and not forwarded to the host to accommodate the IP addresses.

In the future, we plan to investigate different mechanisms for transferring the information to the host machine, independent of the packet data bus. For example, it is possible to establish an independent data bus for the desired information through the QDMA, which will require modifications on both the FPGA and host sides.

Second, due to the limitations of the on-chip BRAM, we cannot store a significant amount of data on the FPGA. Our design assumes that steps 3 and 4 in Fig. 1 will be executed on the host computer and only IP address extraction is executed on the FPGA. However, the FPGA is equipped with an 8GB high-bandwidth memory (HBM) offering 460GBps bandwidth as well as an additional 32GB DDR4 memory with a lower bandwidth of around 30GBps. A potential improvement to the design is to utilize this memory to construct the data tables directly on the FPGA.

By incorporating these changes, a comprehensive solution for the Anonymized Network Sensing Challenge can be implemented based on the initial design presented in this paper.

VI. Conclusion

This paper proposes an FPGA-based hardware solution for the first few steps of the Anonymized Network Traffic Graph

Challenge. Our solution leverages in-network processing to extract and construct anonymized network traffic data structures, demonstrating high throughput that maximizes the potential of an FPGA-based NIC. The design is implemented through a P4-based framework deployed on the Open Cloud Testbed (OCT). `extern` functions in P4 and High Level Synthesis (HLS) are used to implement portions of the design that require state, as P4 is a stateless language. This paper describes the first example of the combination of using P4 and High-level Synthesis in developing network functions on the latest AMD FPGAs.

In the future we plan to investigate implementing more of the Anonymized Network Traffic Graph Challenge by improving the communication of information between FPGA and local host computer, as well as implementing more of the graph challenge on the FPGA itself.

References

[1] AMD OpenNIC Project. https://github.com/Xilinx/open-nic, 2022. [Online; accessed 01-01-2023].

[2] Vitis Networking P4. https://www.xilinx.com/products/intellectual-property/ef-di-vitisnetp4.html, 2022. [Online; accessed 01-01-2023].

[3] Sandeep Bal, Zhaoyang Han, Suranga Handagala, Mert Cevik, Michael Zink, and Miriam Leeser. P4-based in-network telemetry for fpgas in the open cloud testbed and fabric. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2024.

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[5] The caida anonymized internet traces. [Online; accessed 07-12-2024].

[6] Dana Diaconu, Yanyue Xie, Mehmet Gungor, Suranga Handagala, Xue Lin, and Miriam Leeser. Machine learning across network-connected fpgas. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2023.

[7] ESNet SmartNIC. https://github.com/esnet/esnet-smartnic-hw, 2022. [Online; accessed 01-01-2023].

[8] Jananthan et al. Anonymized network sensing graph challenge. In *2024 IEEE High Performance Extreme Computing Conference (HPEC) Submitted*, pages 1–8, 2024.

[9] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure Accelerated Networking:SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, 2018.

[10] Zhaoyang Han, Suranga Handagala, Kalyani Patle, Michael Zink, and Miriam Leeser. A framework to enable runtime programmable p4-enabled fpgas in the open cloud testbed. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2023.

[11] Suranga Handagala, Miriam Leeser, Kalyani Patle, and Michael Zink. Network Attached FPGAs in the Open Cloud Testbed (OCT). In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2022.

[12] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. The P4->NetFPGA workflow for line-rate packet processing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 1–9, 2019.

[13] Michael Jones, Jeremy Kepner, Daniel Andersen, Aydin Buluç, Chansup Byun, K Claffy, Timothy Davis, William Arcand, Jonathan Bernays, David Bestor, William Bergeron, Vijay Gadepally, Micheal Houle, Matthew Hubbell, Hayden Jananthan, Anna Klein, Chad Meiners, Lauren Milechin, Julie Mullen, Sandeep Pisharody, Andrew Prout, Albert Reuther, Antonio Rosa, Siddharth Samsi, Jon Sreekanth, Doug Stetson, Charles Yee, and Peter Michaleas. Graphblas on the edge: Anonymized high performance streaming of network traffic. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8, 2022.

[14] Miriam Leeser, Suranga Handagala, and Michael Zink. FPGAs in the Cloud. *Computing in Science & Engineering*, 23(6):72–76, 2021.

[15] MIT Graph Challenge. https://graphchallenge.mit.edu/, 2024. [Online; accessed 07-11-2024].

[16] S.Handagala. OCT FPGA Tutorial. https://github.com/OCT-FPGA, 2021. [Online; accessed 07-11-2024].

[17] Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, and Boon Thau Loo. Flightplan: Dataplane disaggregation and placement for p4 programs. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 571–592, 2021.

[18] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. Taurus: a data plane architecture for per-packet ml. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1099–1114, 2022.

[19] Abbas Yazdinejad, Reza M Parizi, Ali Dehghantanha, and Kim-Kwang Raymond Choo. P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking. *Computers & Security*, 88:101629, 2020.

[20] Michael Zink, David Irwin, Emmanuel Cecchet, Hakan Saplakoglu, Orran Krieger, Martin Herbordt, Michael Daitzman, Peter Desnoyers, Miriam Leeser, and Suranga Handagala. The Open Cloud Testbed (OCT): A platform for research into new cloud technologies. In *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, pages 140–147. IEEE, 2021.