IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 979-8-3503-8447-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/INFOCOMWKSHPS61880.2024.10620664

P4-based In-Network Telemetry for FPGAs in the Open Cloud Testbed and FABRIC

Sandeep Bal[†], Zhaoyang Han[*], Suranga Handagala[‡], Mert Cevik[§] Michael Zink[†], Miriam Leeser[*]
Northeastern University *, University of Massachusetts Amherst †
Email: *zhhan,mel@coe.neu.edu, †sbal,mzink@umass.edu, ‡s.handagala@northeastern.edu, §mcevik@renci.org

***Abstract*—In recent years, Field Programmable Gate Arrays (FPGAs) have gained prominence in cloud computing data centers, driven by their capacity to offload compute-intensive tasks and contribute to the ongoing trend of data center disaggregation, as well as their ability to be directly connected to the network. While FPGAs offer numerous advantages, they also pose challenges in terms of configuration, programmability, and monitoring, particularly in the absence of an operating system with essential features like the TCP/IP networking stack. This paper introduces an In-band Network Telemetry (INT) approach based on the P4 language for FPGA data plane programming. The goal is to facilitate monitoring and network performance analysis by providing one-way packet delay information. The approach is demonstrated in the Open Cloud Testbed (OCT) and FABRIC testbeds, both offering open access to the research community with greater FPGA availability than commercial clouds. The workflow enables researchers to create custom P4 programs and bitstreams for installation on FPGAs. The paper presents a multi-step approach allowing experimentation within the New England Research Cloud (NERC), testing in OCT, and final deployment in FABRIC, well-suited for one-way delay measurements due to synchronized clocks via GPS time signals. Contributions include the provision of a P4 workflow for FPGAs in a research cloud, a novel FPGA clock-based INT approach, and a comprehensive evaluation through simulation and experiments in the Open Cloud and FABRIC testbeds.**

***Index Terms*—FPGA, Testbeds, P4, INT**

I. Introduction

In recent years, Field Programmable Gate Arrays (FPGA) have made an entry into data centers that are used for cloud computing. Examples of FPGAs in cloud computing data centers are offerings by Azure [17] and Amazon F1 [1]. The recent increase in popularity of FPGAs in compute clouds is driven by their ability to reduce compute load on their host servers. For example, FPGAs can be used to handle computationally intensive workloads in security, image and video processing, machine learning, and data analysis. Furthermore, FPGAs contribute to the current data center disaggregation trend, wherein server components such as memory, CPU, storage, and accelerators can be independently combined at a component level, departing from the traditional allocation on a per-server basis. FPGAs support data center disaggregation well since they often include onboard network interfaces that allow communication that is independent of the host system.

While FPGAs offer many benefits concerning data center disaggregation, they also pose challenges for configuration, programmability, and monitoring. One of the major challenges is the lack of the equivalent of an operating system that provides basic features like the TCP/IP networking stack.

In this paper, we introduce an In-band Network Telemetry (INT) approach which is based on the P4 language for network data plane programming and can be executed on FPGAs. This P4-based INT approach has the goal of supporting monitoring and network performance analysis by providing one-way packet delay information. Reasons for such delays often include the physical distance between two processing nodes, processing time, buffer queue fill, and jitter between individual data packets. In this paper, we implement a prototype INT method using P4. In the future, we plan to incorporate the Precision Time Protocol (PTP) for accurate one-way delay computations. We aim to implement an approach that supports 100 Gbps line rate to identify performance bottlenecks in the network. In addition, we plan to train ML models on the INT data to enable efficient source routing. We demonstrate our INT approach in the Open Cloud Testbed (OCT) [18] and FABRIC [3]. Both testbeds provide open access to the research community and offer considerably greater accessibility of FPGAs compared to what is provided in commercial clouds [1], [17]. FABRIC also expands to multiple sites across the US and a few international sites enabling large-scale experiments. Additionally, FABRIC provides GPS-synchronized PTP, which is an important component for clock synchronization. In addition, we developed a workflow that allows researchers to create their own P4 programs and bitstreams that can be installed on FPGAs in OCT and FABRIC [6].

We present a two-step approach that allows experimenters to develop a bitstream with the workflow we have set up within the New England Research Cloud (NERC), test the resulting bitstream on FPGAs in the Open Cloud Testbed (OCT), before finally deploying and using it in FABRIC. The latter is well suited for one-way delay measurements in networks since the clocks in FABRIC nodes at different locations are synchronized via GPS time signals.

This paper makes the following contributions:

- We implement a P4 workflow that allows the deployment of bitstreams of FPGAs in FABRIC.
- We develop an INT approach that uses the (synchronized) FPGA clock for timestamping. In FABRIC, this approach can be used for one-way delay measurements.
- We evaluate our approach through simulation and experiments in Open Cloud and FABRIC testbeds and present results from these evaluations.

II. Background and Related Work

A. Open Cloud Testbed

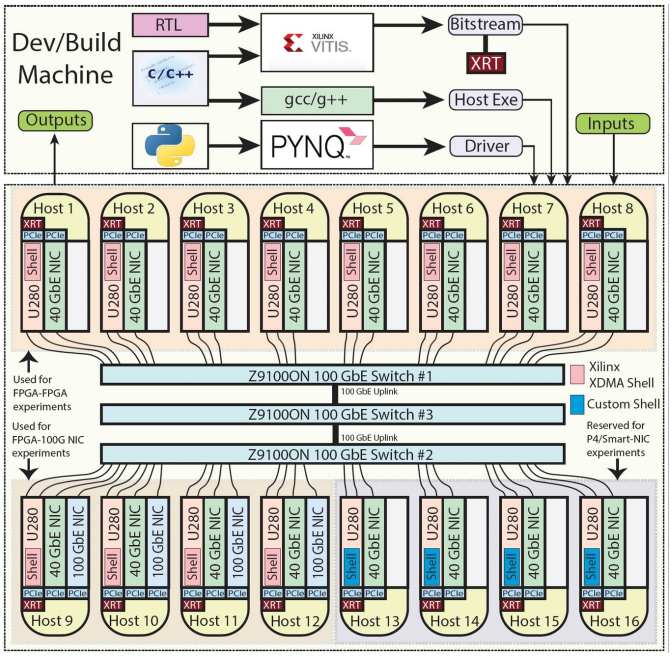

Fig. 1. Overview of OCT FPGA development and workflow [18]

The Open Cloud Testbed (OCT) [18] provides a research-oriented experimentation testbed for systems researchers who focus on cloud platforms. Testbeds like OCT deliver the necessary hardware and software on top of bare metal services to researchers in both the cloud and system communities, enabling more experimental-based research opportunities.

16 FPGAs, namely AMD/Xilinx's Alveo U280 data center cards with High Bandwidth Memory to support data-heavy tasks, are currently provided in OCT along with a toolchain that supports the development of bitstreams [8].

The OCT workflow consists of two primary stages which are illustrated in Fig. 1. **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.

Deployment Stage: After creating the bitstreams and host executables/drivers, users transfer them to the deployment machine(s) on CloudLab, where the FPGAs are installed. The subsequent process involves programming the FPGAs, executing the host executables, and optionally fetching the results back to the development machine.

Currently, OCT consists of 16 high-end servers each equipped with an Alveo U280 accelerator card. Besides the PCIe link connecting the server's CPU and the FPGA, each FPGA is directly connected to the 100 GbE network using QSFP28 direct attach copper (DAC) cables. Among the sixteen accelerators, four are designated for applications involving Xilinx custom flow, such as the AMD OpenNIC shell and P4-based approach discussed in this paper, while the remaining twelve follow the standard Xilinx flow with Xilinx Runtime and an XDMA-based shell. By accommodating both flows, our system meets different user needs, making it flexible and effective across various situations. Currently, we are advancing the FPGA capabilities of OCT by integrating 8 more Alveo U280s, along with 4 VCK5000s and 4 V70s. This expansion aims to broaden the applications of OCT, specifically by incorporating AI-based functionalities.

B. FABRIC

The NSF FABRIC [3] project has the goal to create a versatile experimentation environment that allows researchers to conduct experiments across a wide range of networking and computing scenarios. Its infrastructure provides advanced networking capabilities, enabling researchers to explore innovative networking architectures, protocols, and technologies. Through flexible resource allocation, researchers can configure components of FABRIC according to their needs. To achieve this goal, FABRIC has deployed nodes at almost 40 sites across the US and at some international locations.

Recently, a series of FPGAs were deployed in FABRIC nodes and are now available to researchers to include them in their experiment infrastructures. We have created a P4 workflow and collaborated with the FABRIC team to enable the resulting bitstreams on FPGAs in FABRIC. More details on P4 in FABRIC are provided in Sect. III.

The combination of a testbed that spans the US and beyond and P4-enabled FPGAs, in addition to the provision of a highly accurate time signal, makes FABRIC a suitable infrastructure for research in advanced networking, including areas like INT.

C. In-band Network Telemetry

In-band Network Telemetry (INT) [16] represents an approach to network monitoring and visibility, which integrates telemetry data directly into the data plane of the network, providing real-time insights into the performance, quality, and behavior of network traffic. The INT approach is different to traditional out-of-band monitoring methods. By embedding telemetry information within the packets themselves, INT enables granular monitoring and analysis, facilitating the identification of bottlenecks, latency issues, and other network anomalies. INT offers new insight in the performance of distributed systems resulting in better resource management and troubleshooting. For example, INT allows the probing of the performance of individual segments of an end-to-end path by inserting and removing probes direclty into data packets.

In the context of INT, P4 plays a crucial role in defining how telemetry information is embedded in the packets as they traverse the network. With INT, P4 programs can be designed to capture specific information about the packets, such as ingress and egress ports, latency metrics, or other relevant data. This is achieved through the insertion of instructions in the P4 code that define how telemetry data is collected. In Sect. IV, we explain how P4 is used to collect timestamp information and embedded in data packets.

III. P4 in OCT and FABRIC

P4 is a domain-specific language for packet processing and forwarding. It offers software control over data plane processing within network devices like routers, switches, and NICs. P4 can be used for parsing packet headers, configuring match tables, forwarding packets, and In-band Network Telemetry.

The interest in instantiating P4 applications on FPGAs has grown since the inception of the P4 language [11] due to the excellent programmability and reconfigurability characteristics of FPGAs. For example, the P4-NetFPGA flow [7] employs the Xilinx SDNet (now AMD/Xilinx's VitisNet) toolchain, which translates the P4 code to a hardware description language. Additional research endeavors [13]–[15] exploring various networking challenges build on this foundational workflow. Yet, researchers have had to build their own P4-FPGA toolchains from scratch in order to fully utilize P4.

Recently, two open-source toolchains [4], [6] using AMD/Xilinx's VitisNet were developed. These works have the goal to relieve researchers from having to create their own toolchains and accelerate the development of new P4 codes that can run on FPGAs. Both workflows are supported on the FPGAs that are offered in OCT and FABRIC.

One major difference between OCT and FABRIC is the way researchers can access the FPGAs. In OCT the FPGAs are offered on bare metal servers, which provides a very flexible approach. In OCT, through the support of [6] and its bare metal servers, the P4 applications become runtime programmable. The FPGA and server do not need to reboot while swapping different P4 applications. This feature provides extra flexibility for researchers. On FABRIC, the FPGA can only be accessed through VMs, which offers as an extra layer of protection. This needs to be considered by experimenters since it adds ease of use at the cost of an extra layer of complexity.

IV. P4-based Timestamping on FPGAs

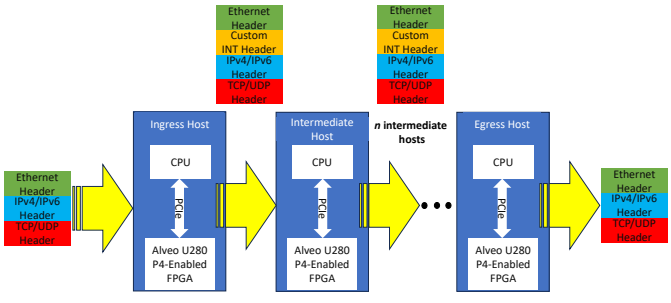

Fig. 2. P4-based INT timestamping for one-way delay measurement.

This section discusses the implementation of the In-band Network Telemetry (INT) functionality within a partially reconfigurable P4 program deployed on the FPGAs of the Open Cloud and FABRIC testbeds. The specific INT functionality we target involves the measurement of the one-way delay between two arbitrary nodes. This is achieved by timestamping data packets at the sender and comparing this timestamp with the receiver's local clock. This requires a very precise synchronization of the sender's and receiver's clocks.

The INT approach presented in this paper has been implemented utilizing the AMD/Xilinx OpenNIC shell, an open-source FPGA-based 100G Network Interface Card (NIC) platform. The INT functionality of the P4 hardware Intellectual Property (IP) block is generated through the utilization of AMD/Xilinx's Vitis Networking P4 (VitisNetP4) tools [6]. This toolchain generates the custom hardware IP from the P4 block required for the implementation of the INT functionality. Packets containing INT information are transmitted between an ingress and an egress node with P4-enabled FPGAs, which can be a part of a larger network as shown in Figure 2. Along with a generic Network Interface Card (NIC), the ingress and egress nodes also host AMD/Xilinx's Alveo U280 FPGAs as P4-enabled SmartNICs capable of processing the INT information from the packets (see Sects. II and III for further details). P4-capable nodes along the path can process the INT data if the goal is to measure one-way delay at several locations, not only the ingress and egress nodes. Clock synchronization requirements as outlined above also apply to intermediate nodes if they are involved in the INT process.

This INT approach to measure one-way delay is divided into a two-step process. In the initial phase, the incorporation of the INT data into the incoming data packets is performed on the P4-enabled FPGA. This operation can be performed at any FPGA within a network. For the remainder of this paper and without loss of generality we assume that the INT timestamping is executed at the ingress node. In the second step, packets including the INT header that has been added at an upstream node have the timestamp information extracted, which is subsequently used to calculate the one-way delay between the two nodes. In addition, the INT header is removed from the packets, which allows the forwarding of them via standard, non-P4-capable nodes to their destination.

The following subsections provide a detailed explanation of this two-step process.

A. Adding the Custom Header with INT Information

Fig. 3 represents the format of the custom header that we implement at the ingress P4-enabled FPGA node.[1] The custom header has a fixed size of 10 bytes. Out of the 10 bytes, the first 8 bytes in the custom header store the actual timestamp value in hexadecimal format. The timestamp, which represents the ingress time of a captured packet, is extracted from an in-built library provided by Xilinx/AMD in the standard metadata, and its size is predefined. The trailing 2 bytes of them are allocated to store the protocol identifier for the next header to facilitate its parsing in the P4 program. This protocol identifier is copied from the EtherType field of the Ethernet header, which is also 2 bytes in size. Consequently, we store a custom protocol identifier in the EtherType field to identify and parse the custom header at the egress node. The custom protocol identifier is chosen to be a reserved value for experimental purposes following the Internet Assigned Numbers Authority

[1]While P4 provides a full-fledged standard for the inclusion of INT data in packets [12], we decided on a much simpler initial approach. We will consider using the official INT standard in future work.

(IANA). This ensures that our custom protocol does not interfere with other existing protocols in use. Insertion of the custom header changes the total size of the packet by 10 bytes and this is updated in the user metadata. This custom header is inserted between the Ethernet and the Network Layer, see Fig. 4. For simplicity, we only consider the ingress and egress nodes without intermediate nodes. Once inserted, the packet is forwarded to its next destination.

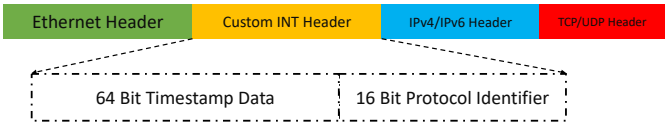

Fig. 3. Custom INT Header for timestamping.

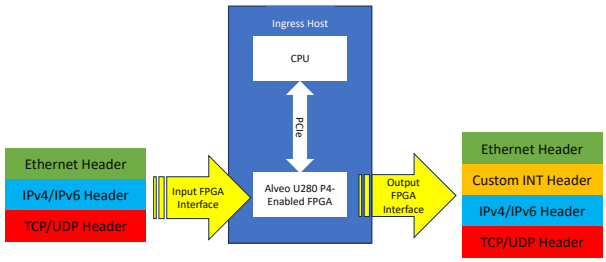

Fig. 4. Process for adding the custom INT header to an incoming packet.

B. Custom Header Removal and INT Processing

Before the custom header is removed, the timestamp data is recorded at the egress node. A new, local timestamp is recorded at the arrival of every incoming packet that includes the custom INT header. This is triggered by the P4 code implemented in the FPGA. The local timestamp and the one carried in the INT header are used to compute the one-way delay between ingress and egress nodes. Then, the protocol identifier of the network layer is restored to the EtherType field. Since the removal of the custom header reduces the size of the packet by 10 bytes, the user metadata is updated. Finally, the original packet is forwarded to the next hop.

V. Evaluation

In this section, we demonstrate the implementation of our INT application in two different testbeds: OCT and FABRIC. Since the experiments involve running P4 programs on FPGAs, we first simulate the design and the environment and verify its correctness. We then run the application on the OCT testbed on two separate nodes containing P4-enabled FPGAs. After that, we run the same application on the FABRIC testbed where the FPGAs are located in different physical locations. The details of the above processes are discussed below.

A. Simulation

Here we describe the simulation process in detail. To begin with, we run our P4 program on a BMv2-based software switch restricted by the Xilinx/AMD VitisNetP4 tools. Some of these restrictions include the usage of the xsa.p4 architecture instead of the v1model architecture, the fields in the standard metadata, and various extern functions, among others [2]. The simulation process includes behavioral level verification of the P4 program along with the Register Transfer Level (RTL) simulation to verify the correctness of the P4 hardware logic generated by the VitisNetP4 tools.

```
000000 11 11 11 11 11 12 aa aa aa aa aa ab 08 00 45 03
000010 00 4b 43 57 00 00 39 11 1d 1f 7c f6 cd 9c cc 93
000020 0a 03 21 1e 5b 98 00 37 aa 9a 6b fa a4 95 d2 54
000030 47 71 92 29 8b 0f 8d e7 e2 99 08 f0 13 0b ef 64
000040 07 3b fe e0 d4 6a ad 3f 5b 3e fd 58 33 49 fc 8f
000050 86 00 1c 4f 00 a0 d0 6d 70
```

Fig. 5. Sample input packet used to evaluate the addition of the INT header in behavioral simulation.

```
000000 11 11 11 11 11 12 aa aa aa aa aa ab 88 b6 00 00
000010 00 00 00 00 00 00 08 00 45 03 00 4b 43 57 00 00
000020 39 11 1d 1f 7c f6 cd 9c cc 93 0a 03 21 1e 5b 98
000030 00 37 aa 9a 6b fa a4 95 d2 54 47 71 92 29 8b 0f
000040 8d e7 e2 99 08 f0 13 0b ef 64 07 3b fe e0 d4 6a
000050 ad 3f 5b 3e fd 58 33 49 fc 8f 86 00 1c 4f 00 a0
000060 d0 6d 70
```

Fig. 6. Sample packet entering the egress node, indicating the added timestamp information in red, in behavioral simulation

```
000000 11 11 11 11 11 12 aa aa aa aa aa ab 08 00 45 03
000010 00 4b 43 57 00 00 39 11 1d 1f 7c f6 cd 9c cc 93
000020 0a 03 21 1e 5b 98 00 37 aa 9a 6b fa a4 95 d2 54
000030 47 71 92 29 8b 0f 8d e7 e2 99 08 f0 13 0b ef 64
000040 07 3b fe e0 d4 6a ad 3f 5b 3e fd 58 33 49 fc 8f
000050 86 00 1c 4f 00 a0 d0 6d 70
```

Fig. 7. Sample packet leaving the egress node, indicating that the timestamp has been removed.

To verify the correctness of the P4 program, we run the header addition and removal of the INT process and verify the correctness of the output packets. We used 8 packets as input for adding the custom header in .pcap format provided by the vendor as part of the VitisNetP4 workflow. Fig. 5 shows one such packet converted from .pcap to .text format. Fig. 6 shows the same packet after the addition of the timestamp (shown in red) between the Ethernet and the Network header. Note that the timestamp is all 0s in this case since this is a simulation and hence, there is no actual timestamp. (As we will show in Sect. V-B, a timestamp that is based on the FPGA cycle counter is added.) The output of the custom header addition simulation is utilized at the input of the custom header removal simulation. Fig. 7 represents the output of the custom header removal simulation which is the same as in Fig.5, thus verifying the correctness of the removal.

B. Open Cloud Testbed

```
0x0000:  4503 004b 4357 0000 3911 1d1f 7cf6 cd9c
0x0010:  cc93 0a03 211e 5b98 0037 aa9a 6bfa a495
0x0020:  d254 4771 9229 8b0f 8de7 e299 08f0 130b
0x0030:  ef64 073b fee0 d46a ad3f 5b3e fd58 3349
0x0040:  fc8f 8600 1c4f 00a0 d06d 70
```

Fig. 8. Sample input packet used to evaluate the addition of the INT header in OCT

```
0x0000:  0000 007f d154 7f28 0800 4503 004b 4357
0x0010:  0000 3911 1d1f 7cf6 cd9c cc93 0a03 211e
0x0020:  5b98 0037 aa9a 6bfa a495 d254 4771 9229
0x0030:  8b0f 8de7 e299 08f0 130b ef64 073b fee0
0x0040:  d46a ad3f 5b3e fd58 3349 fc8f 8600 1c4f
0x0050:  00a0 d06d 70
```

Fig. 9. Sample packet entering the egress node, indicating the added timestamp information in red, in OCT

In this subsection, we evaluate the workflow through which we run the INT application in the Open Cloud Testbed. We reserve two nodes with a standard NIC (Intel XL710) and a P4-enabled FPGA on each of them. At first, we program the FPGA on one of the nodes to add the custom header on the transmission path. Subsequently, we forward the same set of packets that we utilized in Sect. V-A from the ingress node to the egress node. Fig. 8 shows one such packet from the set. Once we forward the packets from the ingress node, we capture the packets at the egress node. Fig. 9 shows one of the captured packets with the ingress timestamp (shown in red). This confirms the addition of the ingress timestamp to the packets. Subsequently, we program the egress node with the P4 code that removes the custom header after the one-way delay is computed internally on the FPGA. Finally, we send the packets again from the ingress node to the egress node via the P4-enabled FPGAs. We attempt to verify the functional correctness of the INT application by forwarding the set of 8 packets from the ingress node where the custom header is added to the egress node where the one-way delay is computed and the custom header is removed. Fig. 10 depicts the INT application operating end-to-end between the ingress and the egress nodes. In the scenario where the INT application functions correctly, the output packets from the egress node after the removal of the custom header should match the input packets at the ingress node before the custom header is added. As we can observe from Fig. 8 and Fig. 10, the packet forwarded out of the egress node matches the packet input at the ingress node, while the delay is computed between the two nodes, hence verifying the INT functionality. We also test the INT timestamping with an actual application in addition to the replay of previously captured packets. In this case, we use SSH to connect from the host OS of the ingress node to the host OS of the egress node. In this case, the SSH session between both nodes was successfully established.

```
22:14:10.869834 IP gp-cm68-0412.lcv.ne.jp.8478 > 204.147.10.3.23448: UDP, length 47
        0x0000:  4503 004b 4357 0000 3911 1d1f 7cf6 cd9c  E..KCW..9...|...
        0x0010:  cc93 0a03 211e 5b98 0037 aa9a 6bfa a495  ....!.[..7..k...
        0x0020:  d254 4771 9229 8b0f 8de7 e299 08f0 130b  .TGq.).........
        0x0030:  ef64 073b fee0 d46a ad3f 5b3e fd58 3349  .d.;...j.?[>.X3I
        0x0040:  fc8f 8600 1c4f 00a0 d06d 70              .....O...mp
```

Fig. 10. Sample packet leaving the egress node, indicating that the timestamp has been removed and the packet matches the one that was originally transmitted.

C. FABRIC

Fig. 11 depicts the slice we use in the FABRIC testbed for validating the intended functionality of the INT application. In line with the OCT experiments, we allocate two nodes: the ingress node featuring a P4-enabled FPGA and the egress node equipped with a standard NIC. All resources used for this experiment are housed at the MASS site.

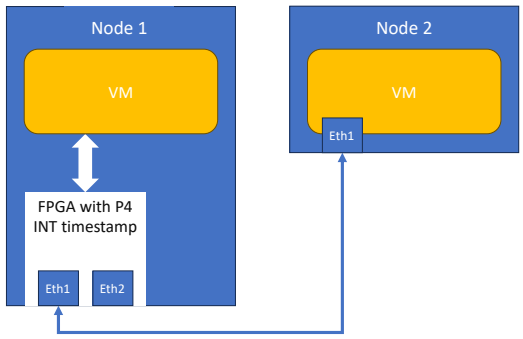

Fig. 11. Experiment setup for the evaluation in FABRIC.

```
ubuntu@fpga-node:~$ sudo tcpdump -r pcap/behav_addhdr_in.pcap -X
reading from file pcap/behav_addhdr_in.pcap, link-type EN10MB (Ethernet)
18:43:19.000000 IP 253.95.11.200.9233 > 154.170.32.16.27841: UDP, length 106
        0x0000:  4fc9 00ae 50fa 0000 b311 d386 fd5f 0bc8  O...P........_..
        0x0010:  9aaa 2010 0101 0101 0101 0101 0101 0101  ................
        0x0020:  0101 0101 0101 0101 0101 0101 0101 0101  ................
        0x0030:  0101 0101 0101 0101 0101 0100 2411 6cc1  ............$.l.
        0x0040:  0072 99c3 7972 5bcf c66e 5ffa b92f 38cb  .r..yr[..n_../8.
        0x0050:  5039 3091 9198 2788 edfa be03 08b9 bff3  P90...'.........
        0x0060:  d0e9 c001 5b13 ac7e 629a 2636 06ae 5d96  ....[..~b.&6..].
        0x0070:  ec71 2da8 66be 4602 6589 8766 c19b cac2  .q-.f.F.e..f....
        0x0080:  4b89 19ba 0eed f4af 84d3 e43c 34d6 da25  K..........<4..%
        0x0090:  1d8f adff afe1 bbb9 374f 644b 06a2 0ee9  ........7OdK....
        0x00a0:  365d c187 1126 20a6 0fc9 5be5 4ffd       6]...&....[.O.
```

Fig. 12. Sample input packet used to evaluate the addition of the INT header in FABRIC

We deploy the same bitstream we used for the evaluation in OCT (see Sect. V-B) on the FPGA in FABRIC. Subsequently, we again proceed to forward the identical set of packets, as utilized in Section V-A, from the ingress node to the egress node. One of the input packets from the sample is shown in Fig. 12. These set of sample packets are then captured at the egress node along with the added INT header. Fig. 13 shows one of the captured packets with the ingress timestamp (shown in red). This confirms the inclusion of the ingress timestamp in the packets. In future work, we would also like to implement the INT functionality between two P4-enabled FPGA nodes from two different FABRIC sites. Achieving this goal requires the allocation of FPGAs from different sites in our experiment slice since each FABRIC site only contains a single FPGA.

VI. Discussion

Currently, our approach uses the cycle counter of the FPGA as the clock. This is not a well-synchronized clock since every local oscillator drifts over time. Mitigating this issue and enabling the synchronization of local computer clocks in distributed systems is achieved through protocols like the Network Time Protocol (NTP) [9] and the Precision Time Protocol (PTP) [10]. In this section, we outline an approach to how a local, synchronized clock can be implemented on the FPGAs. As shown in Fig. 14, the FPGAs installed in OCT and FABRIC can host a PTP client. This client will be able to

```
ubuntu@cx-6-node:~$ sudo tcpdump -i enp7s0 -X
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes


19:38:45.368056 aa:aa:aa:aa:aa:aa (oui Unknown) > 11:11:11:11:11:11 (oui Unknown),
 ethertype Unknown (0x88b6), length 198:
        0x0000:  0000 01b5 cc73 d753 0800 4fc9 00ae 50fa  .....s.S..O...P.
        0x0010:  0000 b311 d386 fd5f 0bc8 9aaa 2010 0101  ........_........
        0x0020:  0101 0101 0101 0101 0101 0101 0101 0101  ................
        0x0030:  0101 0101 0101 0101 0101 0101 0101 0101  ................
        0x0040:  0101 0101 0100 2411 6cc1 0072 99c3 7972  ......$.l..r..yr
        0x0050:  5bcf c66e 5ffa b92f 38cb 5039 3091 9198  [..n_../8.P90...
        0x0060:  2788 edfa be03 08b9 bff3 d0e9 c001 5b13  '.............[.
        0x0070:  ac7e 629a 2636 06ae 5d96 ec71 2da8 66be  .~b.&6..]..q-.f.
        0x0080:  4602 6589 8766 c19b cac2 4b89 19ba 0eed  F.e..f....K.....
        0x0090:  f4af 84d3 e43c 34d6 da25 1d8f adff afe1  .....<4..%......
        0x00a0:  bbb9 374f 644b 06a2 0ee9 365d c187 1126  ..7OdK....6]...&
        0x00b0:  20a6 0fc9 5be5 4ffd                      ....[.O.
```

Fig. 13. Sample packet entering the egress node, indicating the added timestamp information in red, in FABRIC

communicate with a GPS-synchronized PTP server, allowing for accurate clock synchronization.

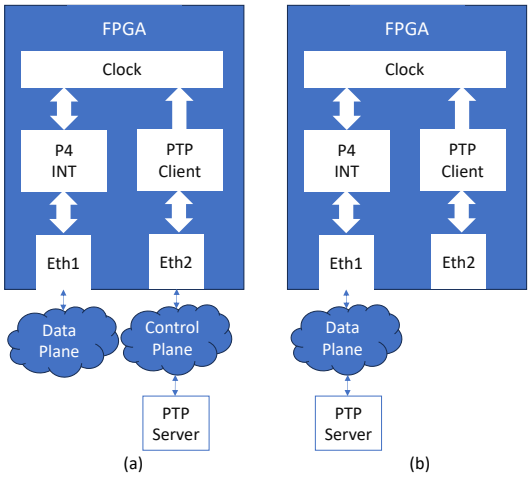

Fig. 14. Two clock synchronization approaches via PTP

As shown in Figure 14, FABRIC nodes provide a GPS-synchronized PTP server with which the PTP client residing on the FPGA can communicate [5]. This communication can happen in two different ways. In the first case (shown in Fig. 14 (a)), both of the FPGA's network interfaces are used. One of the interfaces is connected to the data plane and the other one to the control network. This approach has the advantage that the NTP traffic will not interfere with the actual experiment traffic that is transmitted on the data plane.

The second approach (shown in Fig. 14 (b)), is more general and can be used by FPGAs that have only one physical network interface. In this scenario, the NTP traffic can be transmitted on a different VLAN than the data plane traffic. With a slight modification of the P4 code, the PTP traffic can be filtered out and forwarded to the PTP client on the FPGA.

VII. Conclusions and Future Work

We present a P4-based approach for In-band Network Telemetry (INT) that can be executed on FPGAs. With INT gaining importance in the area of network monitoring and the increasing popularity of FPGAs in data centers and core networking nodes as smartNICs, FPGAs must support INT to avoid measurement gaps in the network. We propose a P4-based INT approach for FPGAs that is based on a workflow we have developed for the development of bitstreams that can be implemented on FPGAs. We demonstrate how this workflow can be used for INT one-way delay measurements through simulation and experiments in OCT and FABRIC.

We have two future work goals for this project. Our first goal is to implement the actual clock synchronization on the FPGAs as outlined in Sect. VI. This will require the implementation of a PTP client and the logic that will synchronize the local clock on the FPGA. Our second goal is to extend the evaluation of our INT approach to more elaborate topologies that contain more than two nodes. To achieve this goal, we will set up topologies across the FABRIC testbed that will span geographically distinct sites. If possible, we will expand these topologies to include international FABRIC sites. In the long term, it is our goal to expand the INT features of our P4-based approach supports. This will include the collection of local node data like throughput and queue size.

References

[1] Amazon. EC2 F1 Instances. https://aws.amazon.com/ec2/instance-types/f1/. Accessed: 2024-01-09.

[2] Vitis networking p4 user guide (ug1308). https://docs.xilinx.com/r/en-US/ug1308-vitis-p4-user-guide, 2023. [Accessed 12-01-2024].

[3] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K. Wang, T. Lehman, and P. Ruth. Fabric: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing*, 23(6):38–47, 2019.

[4] ESNet SmartNIC toolchain. https://github.com/esnet/esnet-smartnic-hw. Accessed: 2024-01-15.

[5] FABRIC. PTP Support in FABRIC Experiments. https://github.com/fabric-testbed/ptp. Accessed: 2024-01-09.

[6] Z. Han, S. Handagala, et al. A framework to enable runtime programmable p4-enabled fpgas in the open cloud testbed. In *IEEE INFOCOM Workshops*, pages 1–6, 2023.

[7] S. Ibanez, G. Brebner, et al. The p4-¿ netfpga workflow for line-rate packet processing. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 1–9, 2019.

[8] M. Leeser, S. Handagala, and M. Zink. FPGAs in the Cloud. *Computing in Science & Engineering*, 23(6):72–76, 2021.

[9] J. Martin, J. Burbank, W. Kasch, and P. D. L. Mills. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905, June 2010.

[10] A. S. Nagra, M. A. Pasha, and S. Masud. Fpga implementation of ieee 1588 protocol for bluetooth-based distributed wireless systems. In *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4, 2022.

[11] P4 Lang. https://p4.org, 2024. [Online; accessed 01-15-2024].

[12] In-band Network Telemetry (INT) Dataplane Specification. https://p4.org/p4-spec/docs/INT_v2_1.pdf. Accessed: 2024-01-09.

[13] M. Saquetti, G. Bueno, et al. P4VBox: Enabling P4-based switch virtualization. *IEEE Communications Letters*, 24(1):146–149, 2019.

[14] H. Soni, M. Rifai, et al. Composing dataplane programs with μP4. In *Proceedings of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 329–343, 2020.

[15] N. Sultana, J. Sonchack, et al. Flightplan: Dataplane disaggregation and placement for P4 programs. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 571–592, 2021.

[16] L. Tan, W. Su, et al. In-band network telemetry: A survey. *Computer Networks*, 2021.

[17] J. Zhang, Y. Xiong, et al. The Feniks FPGA Operating System for Cloud Computing. In *ACM APSys*, September 2017.

[18] M. Zink, D. Irwin, et al. The Open Cloud Testbed (OCT): A platform for research into new cloud technologies. In *10th International Conference on Cloud Networking (CloudNet)*, pages 140–147. IEEE, 2021.