IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 978-1-6654-9427-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOMWKSHPS57453.2023.10225817

Experiments on Network Services for Video Transmission using FABRIC Instrument Resources

Alicia Esquivel Morel‡, Durbek Gafurov‡, Prasad Calyam
University of Missouri - Columbia
{ace6qv, dgvkh}@mail.missouri.edu, pcalyam@missouri.edu

Cong Wang, Komal Thareja, Anirban Mandal
RENCI, University of North Carolina at Chapel Hill
{cwang, kthare10, anirban}@renci.org

Eric Lyons, Michael Zink
University of Massachusetts Amherst
{elyons, mzink}elyons@umass.edu

George Papadimitriou, Ewa Deelman
University of Southern California
{georgpap, deelman}@isi.edu

***Abstract*—Video-based applications form one of the most popular applications on the Internet that is continually evolving. There is a need to develop novel network services that enable reliable video transmission over network paths with dynamic cross-traffic, as well as services that utilize programmable data planes enabled by Protocol-independent Packet Processors (P4). In this paper, we describe experiences in developing network services for reliable video transmission using resources from the FABRIC network instrument, which supports high-performance edge/cloud as well as programmable networking infrastructure. Specifically, we deploy a programmable network using local Ethernet (Layer 2) sites, as well as geographically distributed wide-area network (LAN extension) sites in FABRIC, in order to experiment with visual cloud computing application use cases. Our experiment results provide insights into benefits of data plane programmability (i.e., port forwarding) on improving video streaming quality and compare CPU vs. GPU processing times while completing object detection pipeline processing to obtain visual situational awareness.**

***Index Terms*—FABRIC Instrument, Programmable Data Planes, Video Delivery Testbed, Traffic Engineering Service**

I. Introduction

Video-based applications continue to grow in popularity on the Internet, and are evolving to use latest advances in the areas such as e.g., networking, artificial intelligence, and cybersecurity. To enable reliable video transmission, it is critical to perform experimentation of novel network services in realistic testbeds before deploying them on at-scale production environments with potentially millions of users. Particularly, the testbeds need to support traffic realism that causes dynamic cross-traffic patterns [1], [2], provide access to powerful cloud/edge hardware resources (e.g., CPU, GPU) and also use advances in programmable data planes enabled by Protocol-independent Packet Processors (P4) [3], [4]. Furthermore, the testbeds have to support repeatable experimentation to allow other researchers to replicate findings or extend the testbed capabilities for more sophisticated experimentation.

Fortunately, there have been significant investments by Federal funding agencies such as the National Science Foundation (NSF) to enable a wide range of research instruments to enable at-scale experimentation within realistic testbeds. For example, the Global Environment for Network Innovations (GENI) [5] started in 2007, and the US Ignite Initiative [6], linking cities and regions with ultra-high-speed bandwidth - were part of the initial efforts to provide infrastructures for experimentation with e.g., software-defined networking and edge computing. Chameleon [7] is another example of a community-scale infrastructure that supports bare metal reconfiguration systems and offers users full control of the software stack, including kernel customization, and console access. Using these research instruments, a number of works have developed testbed methodologies and novel network services that characterize and improve applications such as virtual desktop clouds [8], QUIC and HTTP/2 [1] protocols and wireless resource matchmaking [9].

In this paper, we present our experimentation on developing network services for reliable video transmission using a programmable network testbed in the latest NSF-funded FABRIC [10] network instrument. FABRIC infrastructure features state-of-the-art high-speed interconnections with dedicated optical links and access to powerful cloud/edge hardware resources. Specifically, we describe experiences in deploying video processing pipelines related to use cases of Visual Cloud Computing (VCC) applications, and perform video transmission application analysis using programmable data planes as well as object detection tools with dedicated computing nodes, all possible by use of FABRIC resources. Our experiments involve a series of steps that begin with allocation of resources to deploy a programmable network featuring an isolated Ethernet (Layer 2) setup in a site, and then use geographically distributed sites for a wide-area network (LAN extension) setup. The application traffic involves data transfers on FABRIC resources that are controlled using a programmable data plane that supports network services that: (a) use a port forwarding algorithm to avoid congested links in order to enhance video streaming quality, (b) enable fast processing times when deploying an object detection pipeline processing with GPU nodes. The experiment results provide insights into benefits of data plane programmability to improve video streaming quality and also show how visual

‡ These authors contributed equally to this work.

978-1-6654-9427-4/23/$31.00 ©2023 IEEE

situational awareness is improved using GPU vs. CPU nodes for video pipeline processing. The remainder of the paper is summarized as follows: in Section II we discuss the main configuration setup in the FABRIC instrument and motivate the VCC application use cases. In Section III, we describe our experiment environment by first providing details in setting up a local Ethernet, and then describe a network slice setup for traffic analysis with video streaming applications. Section IV details our methodology to experiment using programmable data planes and computation nodes, and also presents the experiment results and salient findings for development of network services. Lastly, Section V concludes the paper.

II. Background

In this section, we first present our high-level FABRIC instrument configuration that leverages unique high-performance network instrument resources. Following this, we motivate the real-world scenarios in VCC applications that require novel network services experimentation involving large volumes of video transmission in a reliable and scalable manner.

A. FABRIC Instrument Configuration

The FABRIC instrument is an adaptive, programmable research infrastructure for supporting computer science research, as well as domain science application research involving cutting-edge network experimentation at scale [11]. Our high-level configuration relies on a centralized management and control framework, which is integrated with the FABRIC instrument. We use the Jupyter Notebooks within the FABRIC portal for orchestration, allocation and experimentation. Specifically, the Jupyter Notebooks offer a friendly interface to configure slice parameters, create and get slices, configuring nodes, switches and switch tables. Also, they provide comprehensive information about resources allocation and experimentation details. In our experiments, we perform bandwidth testing and setup of different network layers using FABRIC capabilities to demonstrate node connections in an isolated Ethernet within a local area network in same sites, and in different geographically distributed wide-area network sites. We also instantiate port forwarding to transfer packets from one network to another and enable end-to-end network communication across multi-site network segments.

B. Visual Cloud Computing (VCC) Application

We consider VCC application use case scenarios as they demand highly efficient solutions in video data processing pipelines on edge-to-cloud computing infrastructures. In addition, VCC applications can leverage artificial intelligence, and machine learning to provide visual situational awareness in cases of public safety response during disaster incidents [12]. It is critical in VCC applications to deliver video streaming while ensuring reliable performance, even in the presence of congestion bottlenecks as shown in [13]. Figure 1 illustrates a VCC application scenario where a set of Unmanned Aerial Vehicles (UAVs) are used as the source of video streams that need to be transferred using network services over multi-site network segments to provide visual situational awareness in applications such as e.g., search-and-rescue, aerial surveillance. The UAVs have on-board cameras that collect video from multiple perspectives at an incident scene, and transfer them to a Ground Control Station (GCS) that manages the command control for UAV guidance, and initiates the processing of the video data sets using available edge/cloud computing resources to satisfy the user requirements (e.g., to stream a high or a low definition video with satisfactory video quality) [14].

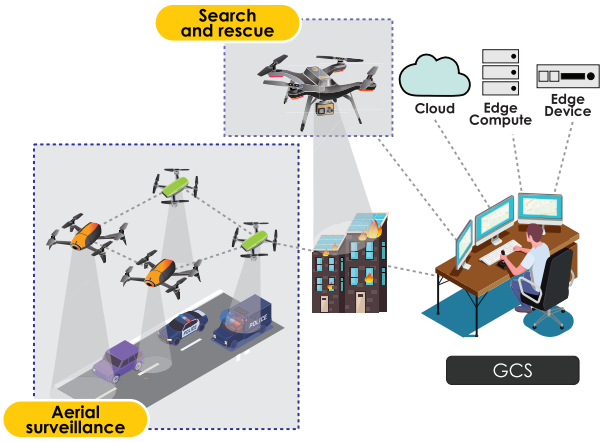

Figure 1: Unmanned Aerial Vehicles can be utilized for a wide variety of applications such as e.g., search-and-rescue, aerial surveillance. Challenges for network services management need to be overcome to guarantee satisfactory performance of network-edge based applications such as video delivery.

III. Experimentation Environment in FABRIC

In this section, we first describe the experimentation environment by defining the steps to create an isolated Ethernet network, and allocate nodes in FABRIC. Next, we characterize the different computing nodes, and assess their performance metrics for different sub-nets. Lastly, we detail the setup of a video streaming application for VCC experiments.

A. Isolated Ethernet Network Setup

We setup a Layer 2 network topology in FABRIC in GPN site as shown in Figure 2 for data transfer between two computing nodes, and use two corresponding network interface card (NIC) components that are connected via an isolated local Ethernet configuration. We also allocate resources in different sites in a wide-area network setup to compare performance/bottlenecks for resource-intensive applications. FABRIC offers various NIC components with models that include NIC_Basic, NIC_ConnectX_5, and NIC_Connect_X6 respectively. In this experiment we are leveraged the capabilities of a *NIC_Basic component*. This NIC_Basic is a 100 Gbps Mellanox ConnectX-6 SR-IOV VF (1 Port). The allocation and settings are made via a centralized management and control framework in FABRIC [15]. The Layer 2 network is added

Table I: Sub-net connectivity statistics to show characteristics of round-trip times.

Measurement	Subnet_1	Subnet_2	Subnet_11	Subnet_12
$count$	25.00000	25.00000	25.00000	25.00000
$mean$	0.07028	23.156000	0.255720	23.328000
std	0.01724	0.050662	0.057024	0.089069
min	0.05300	23.100000	0.168000	23.200000
max	0.05300	23.100000	0.168000	23.200000

Table II: Packet transfer throughput comparison using Iperf for parallel connections between same site and different sites for a 10 second period of test showing improved performance in the same site configuration.

Connections	Size (GBytes)		Throughput (Gbits/sec)	
	Same site	Diff. sites	Same site	Diff. sites
10	20.7	3.25	17.5	2.77
20	21.3	5.31	18.0	4.52
30	20.4	6.97	17.3	5.91
40	14.3	7.71	11.9	6.53
50	19.1	5.49	16.0	4.60

to the slice, and then this passes the list of interfaces that can be connected to the Ethernet setup.

An essential step to setup LAN extension between GPN and SALT sites as shown in Figure 2 is to configure the IP addresses. FABRIC provides the API FABlib2 [16] that facilitates this step, with easy-to-use methods. Sub-net and list of IP addresses can be queried with the respective descriptions. After specifying the IP addresses and picking the sub-nets, nodes are ready to be configured by printing out the interface details. The methods to accomplish involve the use of `node.get interface` to get the interface, and `iface.ipaddr add` to set the IP address and sub-net. Once the deployment is ready (which includes both network and computing nodes), users can log in to the nodes and run experiments. `Iperf3` is installed in the computing nodes and produces standardized performance measurements for our network setup.

Table I shows the sub-net connectivity statistics in terms of round-trip times. Specifically, in *Subnet_1*, we performed Ping connectivity testing to node 2 from node 1 (both nodes were in the same GPN site), with default packet sizes. Similarly, in *Subnet_2*, we performed Ping connectivity testing to node 3 from node 2 (with node 2 in the GPN site and node 3 in the different SALT site), with default packet sizes. In *Subnet_11*, we performed Ping connectivity testing from node 1 with max (65000kb) packet size (with both nodes in the same GPN site). Lastly, in *Subnet_12*, we performed Ping connectivity testing to node 2 from node 3 with max (65000kb) packet size (with nodes in different sites of GPN and SALT, respectively). Table II shows the packet transfer throughput comparison obtained using the Iperf tool with parallel connections (between 10 to 50) used between the same site and different sites for a 10 second period of test. The key finding is that we see improved performance in the same site nodes configuration case, versus when the nodes are in different sites.

B. Video Streaming Application Setup

To validate the experiment environment suitability for VCC application experimentation, we analyzed a video streaming application. Specifically, we relied on an open-source software viz., FFmpeg [17] to emulate real-world video transmission for an application scenario that uses cloud/edge resources. Video data was transferred from *node 1* to *node 2* and *node 3* to *node 2*. The `ffprobe` and `tcpdump` tools were utilized to output information about our video input, including duration, frame size, rate, and analyze the overall network performance, including the jitter measure in the video stream. We began an `ffmpeg` process on a source node that streamed the contents of an input video file over UDP to a target node. We then ran the `ffprobe` utility on the target node to analyze the video stream and extract frame data. We repeated this procedure for two different video streams, one from the same site as the target node, and one from a different site.

Table III shows DataFrame measurements obtained for the number of frames transferred and frame size comparison for parallel connections between same site and different sites, in order to assess the effects of network location of the nodes on the video streaming application. We found no apparent differences for a simple video transmission experiment in an ideal scenario as seen from the measurements. The same site DataFrame contained 3872 rows, representing 3872 frames in the video stream. On the other hand, the different sites DataFrame contained 3884 rows, representing 3884 frames in the video stream. The column size (bytes) in both DataFrames represents the size of each frame in bytes, the mean size of the frames in the same site stream was 3957.89 bytes, while the mean size of the frames in the different sizes was 3956.77 bytes. The standard deviation of the frame sizes was relatively high in both streams, with a value of 6794.92 bytes for the same site stream and a value of 6791.73 bytes for the different sites stream as also illustrated in Figure 3. The minimum and maximum values for the frame sizes were similar in both streams, with a minimum value of 468 bytes and a maximum value of 31438 bytes. The size (bytes) column in both DataFrames had a similar distribution, with the 25^{th}, 50^{th}, and 75^{th} percentile values, all within a few hundred bytes of each other.

Table III: Number of frames transferred and frame size comparison for parallel connections between same site and different sites showing no apparent differences for simple video transmission in an ideal scenario.

Measure	Same Site		Different Sites	
	Frame ind.	Size (bytes)	Frame ind.	Size (bytes)
$count$	3872.00	3872.00	3884.00	3884.00
$mean$	1935.50	3957.89	1941.50	3956.77
std	1117.89	6794.92	1121.35	6791.73
min	0.00	468.00	0.00	468.00
25%	967.75	815.00	970.75	816.75
50%	1935.50	1621.00	1941.50	1623.00
75%	2903.25	3127.50	2912.25	3125.00
max	3871.00	31438.00	3883.00	31438.00

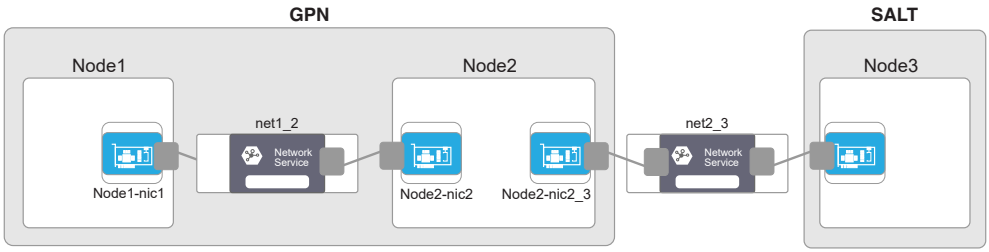

Figure 2: FABRIC deployment with slices allocated in two different sites i.e., GPN and SALT. Two nodes were allocated in GPN, whereas one node in SALT as part of the LAN extension. Network interfaces connect the computing nodes at the sites.

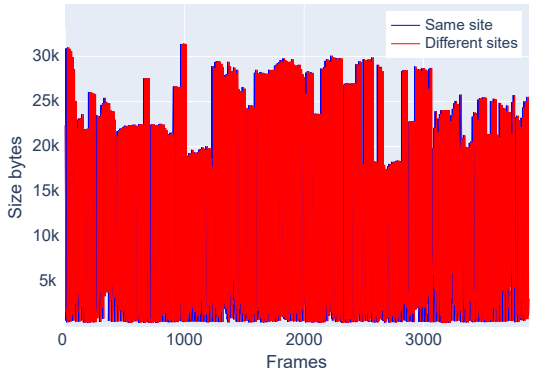

Figure 3: The frame size was relatively high in both same and different sites, resulting in minimum effects of network location on the video streams.

Table IV: Jitter measurement comparison between same and different sites showing no apparent differences for simple video transmission in an ideal scenario.

Measure	Same Site	Different Sites
count	22209.000000	22209.000000
mean	0.000930	0.000907
std	0.002962	0.003675
min	0.000000	0.000000
25%	0.000322	0.000305
50%	0.000387	0.000369
75%	0.000592	0.000554
max	0.240147	0.467062

In addition, we ran experiments with the `tcpdump` utility to capture packets from the video streams, and also used the `scapy` library to extract data from the packets. More specifically, latency and jitter metrics were analyzed in the video streams as important indicators of the video quality. Through a `ffmpeg` process on a source node, we streamed a video file over UDP to a target node. We used `tcpdump` on the target node to capture packets from the video stream and save them as a `PCAP` file. This file was read with `scapy` library and timestamp, latency and jitter information was extracted for each packet.

Again, we repeated this process for two different video streams, one from the same site as the target node, and one from a different site. We compared the resulting data to assess the effects of network location on the video streams. The resulting data was stored in Pandas DataFrames and summarized using statistical measures listed in Table IV. The measures include information about the mean, standard deviation, minimum and maximum values, and percentiles for the latency and jitter data for the two video streams. Through this experimentation, we again found no apparent differences in the measurements for simple video transmission in an ideal scenario. This further indicates the high-performance nature of both network settings in the FABRIC resources.

IV. Resource Adaptation Experiments

In this section, we first explain the experiment testbed for solution implementation of a VCC application deployed using FABRIC resources in order to demonstrate the benefits of the programmable network capability for resource adaptation. Following this, we explain the implementation and execution of experiments involving a high-performance object detection pipeline for video stream data processing using CPU/GPU node configurations.

A. Video Streaming Application and Programmable Data Planes

Our adapted experiment testbed is comprised of a set of P4-programmable components with corresponding data plane interfaces between them. The current revision of the P4 [18] language $P4_{16}$ is instantiated, and the BMV_2 *(Behavioral model version 2)* is installed in the switches. We benefit from the P4Runtime [19] tool to programmatically install rules into each switch. Our core network is setup to emulate VCC application servers and end-user sites, in addition to basic NICs (Virtual NICs implemented as Single Root I/O Virtualization, on top of ConnectX-6 physical cards). The disk of the switches is 100 GB each, whereas each of the hosts has a 10 GB disk. For our experiments, we relied on a topology (see Figure 4) comprised of: five hosts which act as servers of the UAVs' video streams, end-users of the video streams [h1, h2, h3, h11, h22] corresponding to decision makers, and three P4 programmable switches [s1, s2, s3] of the video content delivery network. Our P4 algorithms are implemented within the switches [s1, s2, s3]. Specifically, we rely on a port forwarding algorithm, creating a control system that knows the entire topology, being aware of any congestion that might occur, and taking the appropriate steps to overcome it. Our

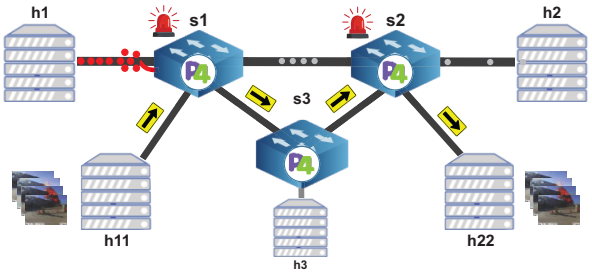

Figure 4: Experiment testbed for solution implementation of a VCC application deployed in a FABRIC testbed with port forwarding algorithm to reroute the packets through a link with no congestion.

system is also able to diagnose congestion issues at the ingress level, thus solving the issue before it reaches the egress level.

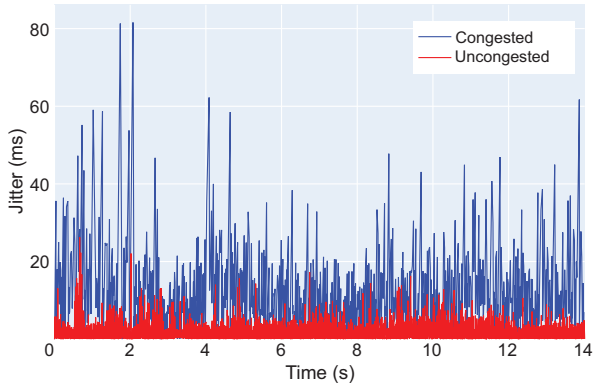

Figure 5: Jitter measurements between two paths in our realistic custom topology to compare traditional shortest path method (that introduces congestion) and a port forwarding approach that uses programmable switches (that makes the setup congestion-free).

We consider a crucial scenario where high throughput and low latency are needed to support the VCC application, allowing reliable and timely transmission of video data. Our experiments are based on a real-world use case in which a UAV surveillance system a.k.a. swarm of UAVs connected via an ad-hoc network, are responsible for collecting video data for a VCC application. To mitigate congestion, P4 switches are utilized in conjunction with a port forwarding algorithm to avoid the presence of congested links. To validate the effectiveness of this approach, jitter measurements were taken in two connecting lines, one utilizing the traditional shortest path method and the other one utilizing the port forwarding approach. Figure 5 shows the jitter measurements are lower with the port forwarding algorithm that is implemented in the P4 switches (implies better video quality), and the jitter measurements are higher with the traditional shortest path that introduces congestion.

Table V: Frame processing times comparison using $\texttt{YOLO}_{V5}$ model with `CPU` and `CUDA` modes of operation on a node, showing min/max metrics for frame per second, and overall processing time that includes video pre-processing, objection detection and storing processed video.

Processor	FPS Max.	FPS Min.	Overall Processing Time
CPU	5.649718	0.017332	599.704482
CPU	5.617978	0.034234	599.704482
CPU	5.555556	0.994036	599.704482
CPU	5.681818	0.96432	599.704482
CPU	14.925373	0.017119	599.704482
CUDA	83.333333	0.766284	511.263838
CUDA	83.333333	0.87108	511.263838
CUDA	83.333333	0.743494	511.263838
CUDA	83.333333	0.714796	511.263838
CUDA	83.333333	0.034426	511.263838

B. *Video Analytics and Object Detection Model*

The goal of this set of experiments is to empirically show the feasibility of utilizing FABRIC in facilitating the implementation and execution of high-performance VCC applications. Video analytics, storage, and object detection models were performed using both CPU and GPU nodes. Specifically, we leverage a surveillance application that employs a swarm of UAVs, which stream and offload video data to the Ground Control Station (GCS). A popular object detection model is implemented with a class that performs object detection on the video file. The algorithm uses the $\texttt{YOLO}_{V5}$ pre-trained model to make inferences and `opencv2` to manage the frames. These inferences are utilized to plot boxes on objects (i.e. people), along with labels as observed in Figure 6.

Figure 6: Example of object detection results obtained using a video file that is streamed and offloaded at an edge Ground Control Station in experiments to compare performance of video processing using CPU and GPU resources in FABRIC.

As shown in Table V, $\texttt{YOLO}_{V5}$ model enabled the comparison of CPU and CUDA modes performance in terms of

min/max metrics for frame per second (FPS), and Overall Processing Time that includes video pre-processing, object detection and storing processed video. The CUDA mode involves a GPU worker with a storage instance (i.e., NVIDIA RTX6000 GPU node) that is directly connected to the head node in FABRIC. We can see from Table V that the performance of the CPU mode processing is slower in comparison with the CUDA mode processing of the compute-intensive tasks involved in the video streaming and object detection tasks. Additionally, as illustrated in Figure 7, we can observe that the GPU-based execution with CUDA mode outperforms CPU-based execution in terms of Overall Processing Times, even when the threat counts increase, and a linear increase in Overall Processing Times is observed in both the CPU and CUDA mode cases. This trend can be attributed to the occurrence of I/O operations, which become increasingly time-consuming as the number of threads increases.

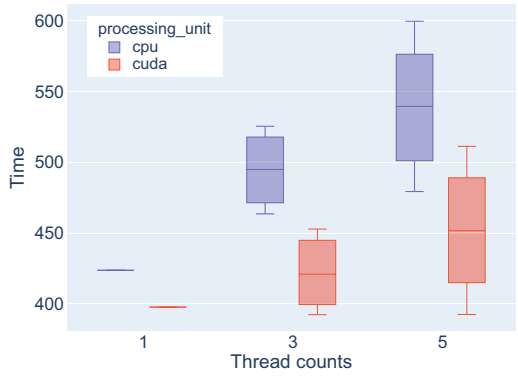

Figure 7: Overall Processing Time comparison when using 1, 3 and 5 thread counts in both the CPU and CUDA mode cases.

V. Conclusion

In this paper, we detailed experiments of network services for supporting Visual Cloud Computing (VCC) applications using FABRIC instrument resources. We experimented with high-speed and dedicated optical links, deploying use-case scenarios at-scale and testing video streaming performance to adapt resources for reliable video transmission. We leveraged a centralized management and control framework that is suitable for reproducible experimentation. We successfully deployed P4-based switches for avoiding congestion and managing cross-traffic dynamics in programmable network settings. We also showed how Overall Processing Times for compute-intensive tasks involving object detection of video streams obtained from UAV swarms can be more efficient when using GPU resources in FABRIC. Our experimentation with network services in FABRIC are important to deploy real-world scenarios that involve video transmission in network-edge locations that leverage high-performance edge/cloud resources as well as programmable network infrastructure.

Future work can leverage In-band Network Telemetry (INT) capabilities in FABRIC for more fine-grained adaptation of resources to further improve video transmission quality.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Award Numbers: CNS-1950873, CNS-1647182 and OAC-2018074. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] D. Bhat, J. Anderson, P. Ruth, M. Zink, and K. Keahey, "Application-based qoe support with p4 and openflow," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 817–823, IEEE, 2019.

[2] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 edge node enabling stateful traffic engineering and cyber security," *Journal of Optical Communications and Networking*, vol. 11, no. 1, pp. A84–A95, 2019.

[3] S. Kaur, K. Kumar, and N. Aggarwal, "A review on p4-programmable data planes: Architecture, research efforts, and future directions," *Computer Communications*, vol. 170, pp. 109–129, 2021.

[4] F. Cugini, P. Gunning, F. Paolucci, P. Castoldi, and A. Lord, "P4 in-band telemetry (int) for latency-aware vnf in metro networks," in *Optical Fiber Communication Conference*, pp. M3Z–6, Optica Publishing Group, 2019.

[5] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5–23, 2014.

[6] G. Ricart, "Us ignite testbeds: Advanced testbeds enable next-generation applications," in *2014 26th International Teletraffic Congress (ITC)*, pp. 1–4, IEEE, 2014.

[7] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Mambretti, P. Rad, and P. Ruth, "Chameleon: a scalable production testbed for computer science research," in *Contemporary High Performance Computing*, pp. 123–148, CRC Press, 2019.

[8] P. Calyam, A. Venkataraman, A. Berryman, and M. Faerman, "Experiences from virtual desktop cloud experiments in geni," in *GENI Research & Educational Experiment Workshop (GREE)*, 2012.

[9] K. Webb, S. K. Kasera, N. Patwari, and J. Van der Merwe, "Wimatch: Wireless resource matchmaking," in *IEEE INFOCOM WKSHPS*.

[10] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "Fabric: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.

[11] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "Fabric: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.

[12] R. Gargees, B. Morago, R. Pelapur, D. Chemodanov, P. Calyam, Z. Oraibi, Y. Duan, G. Seetharaman, and K. Palaniappan, "Incident-supporting visual cloud computing utilizing software-defined networking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 1, pp. 182–197, 2016.

[13] A. E. Morel, P. Calyam, C. Qu, D. Gafurov, *et al.*, "Network services management using programmable data planes for visual cloud computing," *International Conference on Computing, Networking and Communications (ICNC 2023)*, 2023.

[14] G. A. Q. Marrogy, "Enhancing video streaming transmission in 5 ghz fanet drones parameters," *Telecommunications and Radio Engineering*, vol. 79, no. 11, 2020.

[15] FABRIC, "https://jupyter.fabric-testbed.net," 2023. Last accessed January 2023.

[16] FABlib2, "https://github.com/fabric-testbed/fabrictestbed-extensions/," 2023. Last accessed January 2023.

[17] S. Tomar, "Converting video formats with ffmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

[18] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[19] P4Runtime, "https://p4.org/p4-spec/p4runtime/main/p4runtime-spec.html," 2021. Last accessed January 2023.