# Towards Testbed-Wide Traffic Profiling for FABRIC

Nishanth Shyamkumar *Illinois Inst. of Technology*
Sean Cummings *Illinois Inst. of Technology*
Hyunsuk Bang *Illinois Inst. of Technology*
Nik Sultana *Illinois Inst. of Technology*

***Abstract*—Network testbeds are important tools in modern research, and they require substantial *observability* by users and operators. Observability of a testbed helps to capture transient conditions of the testbed that might be difficult to reproduce, and that affect the testbed's performance or the outcomes of user experiments. Having a *programmable network profiler* helps provide this observability for the testbed's *dataplane* by summarizing the network activity on the testbed. But building a testbed-wide profiler is non-trivial since it involves making various kinds of technical trade-offs.**

**This paper introduces Patchwork: a programmable network profiler for the FABRIC testbed. Patchwork programmatically samples the dataplane to create a trace of its contents over time. Unlike features like NetFlow or sFlow, Patchwork places minimal burden on operators to provide a configurable interface for flexible traffic sampling to the testbed's users.**

**Using Patchwork, we present the first ever profile of FABRIC's network traffic among its sites in two continents. We discuss both testbed-wide and site-specific traffic characteristics. We also describe ongoing work to enrich and extend this profile to obtain a deeper understanding of FABRIC's traffic profile, and enable researchers to form more detailed insights when using FABRIC.**

***Index Terms*—FABRIC, Traffic Profiling, Network Monitoring**

## I. Introduction

Large network testbeds are vital tools for research because they provide a variety and quantity of professionally-managed resources at a scale that far exceeds what a typical university testbed affords. Through their size and constituent hardware, network testbeds enable research that can be evaluated more rigorously, and which can thus be more impactful. FABRIC [1] is an example of such a testbed. It is organized into a federation of testbeds that are hosted by member institutions that span a wide geographical area.

In addition to its size and hardware composition, the usefulness of a research testbed also increases proportionally to its *observability*. A testbed is typically used for running experiments and taking measurements, and increased observability enables a wider variety of measurements to be taken at improved accuracy. In turn, observability supports the quality of the research that is done on a testbed.

FABRIC provides an excellent observability framework—the Measurement Framework [2]—that is focused on recording measurements across a wide variety of resources with high accuracy. Observability in this framework spans virtually any host-based resource—including the utilization of CPU cores, memory, and custom, experiment-specific quantities that testbed users can choose to gather. The Measurement Framework can also be used to capture traffic related to a specific experiment.

Observing the testbed's *dataplane* complements the existing observability framework in FABRIC. The testbed's dataplane is the composition of network interfaces across the testbed, and it can be implicitly shared by simultaneous experiments and their resources. Gathering packets that are related to multiple experiments provides an understanding of the type and frequency of traffic that is crossing the shared network.

Providing dataplane observability requires substantial technical development effort. The testbed's *operator* must provide testbed users with an interface into network equipment. Testbed *users* would then need a system that uses this interface and orchestrates the gathering, processing and visualizing of traffic-related observations. Both the operator and users need to navigate difficult technical choices and trade-offs—including how to balance accuracy and overhead, and how to manage different users' competing demands on the testbed.

This paper described *Patchwork*, a programmable network profiler for the FABRIC testbed. Patchwork programmatically samples the dataplane to create a trace of its contents over time. Unlike features like NetFlow or sFlow (§III), Patchwork places minimal burden on operators to provide a configurable interface for traffic sampling to testbed users. Patchwork complements FABRIC's Measurement Framework and builds on FABRIC's PortMirroring feature. Although we focus on FABRIC (§IV) for practical prototyping, the ideas in Patchwork could be adapted to other testbeds.

After describing Patchwork's design and implementation (§V), we present the first ever profile of FABRIC's network traffic among all its available sites (§VI). At the time the profile was made, we observed the following: **(1)** FABRIC traffic mostly consisted of IPv4 packets, many of which were related to SSH streams; **(2)** traffic at most FABRIC sites exhibit a low variety of protocol headers in their traffic, but some sites use many types of headers; **(3)** the distribution of frame sizes at different FABRIC sites suggests a variety of workloads across FABRIC's network, and some sites carry notably higher-throughput streams.

We also describe ongoing work (§VII) to enrich and extend this profile to obtain a deeper understanding of FABRIC's traffic profile, and enable researchers to form more detailed insights when using FABRIC.

## II. Motivation and Problem Definition

The dataplane of a network testbed is a specialized, shared resource that interconnects the resources in the testbed. In-

cluding the dataplane in a testbed's observability infrastructure provides an important service to the testbed's operators (who can better manage the testbed) and its users (who can better understand the testbed's operation when running experiments).

Use cases for dataplane observability include: **(1)** Understanding traffic composition to support diagnostics and post-mortems—e.g., to understand problems with resource allocation or why an experiment is not being adequately reproduced. **(2)** Getting a picture of flows across FABRIC to detect short-term and long-term anomalies in the network. **(3)** Understanding the most network-active nodes, and what their activity consists of, to plan for increased capacity. **(4)** Making inference from background traffic [3]. **(5)** Validating traffic-generation systems [4].

Observing the dataplane requires an adequate *technical capability* and also a *management infrastructure*, and this paper describes both. For technical capability, this paper leverages the *port mirroring* feature that the FABRIC testbed provides. There are alternative approaches that could also work, and they are described in §III.

Developing a management infrastructure requires navigating trade-offs on how to gather and process observations. This includes deciding whether to observe *all* or *some* traffic. And if observing only a subset of traffic, deciding on: which specific *sites* to observe, which *types* of traffic to observe, and at what *frequency*. Further, this management infrastructure must be accessible to users of the testbed and its existence must not place extra burden on the operators of the testbed.

## III. Why not use NetFlow, sFlow, IPFIX, etc?

Patchwork currently relies on port mirroring as the sole primitive for sampling the testbed's dataplane. Our experience so far is that this primitive is adequate for dataplane observability of a testbed's network.

Alternative primitives include **(1)** using switch-supported features such as NetFlow, sFlow, IPFIX—these an differ across versions, vendors, and implementations—or **(2)** using network taps to mirror this traffic—which would come at increased expense to purchase the taps and dedicate ports on receiving equipment. The port mirroring approach offers raw access to the dataplane and affords plenty of control over filtering and processing by using vendor-neutral tools.

If the testbed's switches are upgraded, then as long as the new switch supports port mirroring, then Patchwork will continue working. Alternative primitives could be added to Patchwork—including NetFlow and taps—-but that would expand the features that the testbed operator or specific FABRIC sites must expose to users, potentially increasing their burden.

## IV. Background: FABRIC sites

Before describing how Patchwork works in §V, this section outlines FABRIC's design.

FABRIC is structured into connected *sites*. A site consists of a cluster of equipment, sketched in Fig. 1. This cluster contains a switch that interlinks a number of worker machines. Each machine hosts one or more Virtual Machines (VMs) and is

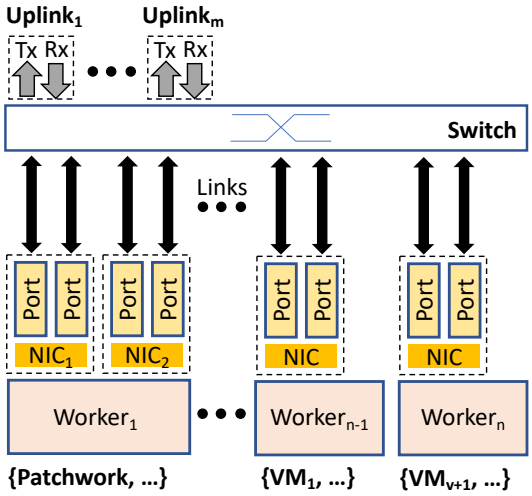


Fig. 1: Composition of a typical FABRIC site.

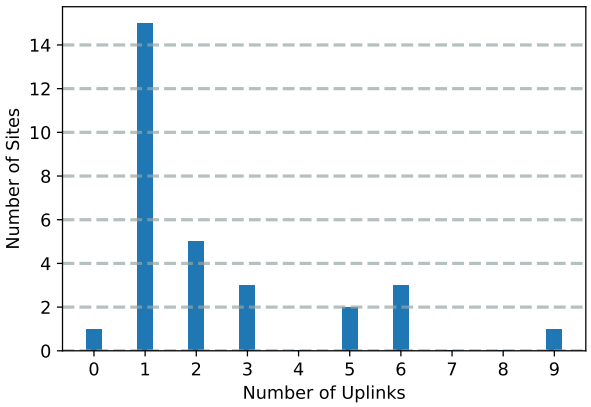


Fig. 2: Histogram from 12/11/2023 showing the distribution of uplink ports across all FABRIC sites. Most sites have a single uplink. The STAR site has 9 uplinks, making it an important hub. One site was in maintenance mode, and had zero uplinks.

equipped with one or more network interface cards (NICs). Each NIC has one or more physical ports. Each port is linked to a dedicated port on the switch.

Sites are connected through dedicated uplink ports on the switch—such as Uplink$_1$ in Fig. 1. Uplink ports are linked to infrastructure that connects FABRIC sites. As with other links, uplinks are physically structured into two uni-directional channels: one for transmitting (Tx) and another for receiving (Rx) frames. Mirroring a switch port necessitates choosing whether to mirror one or both channels of that port.

Different FABRIC sites can have a different number of resources—workers, NICs, but also uplinks. Fig. 2 shows the current distribution of uplinks at FABRIC sites.

Note that Fig. 2 provides a static picture of a system that

changes over time. FABRIC changes as more sites are added and as resources are added or upgraded in sites that form part of FABRIC. In addition to the problem formulation described in §II and in §III, tolerance to FABRIC's change and resource variety is another constraint that Patchwork seeks to satisfy.

## V. Design and Implementation of Patchwork

Patchwork is a tool for observing FABRIC's dataplane through programmable traffic capture and processing. This section describes how Patchwork was designed and built in the context of FABRIC's design.

### A. Overview

To FABRIC, Patchwork appears like any other experiment. Patchwork runs on FABRIC using the resources that the testbed makes available to all experiments. In particular, Patchwork creates slices that include VMs for traffic gathering and processing, smart NICs for receiving traffic for those VMs, and uses port mirroring at the site switch as the technical capability (§II) that directs traffic to those NICs. In the sketch shown in Fig. 1, Patchwork runs on $\mathsf{Worker}_1$ and uses one or both of $\mathsf{NIC}_1$ and $\mathsf{NIC}_2$ for traffic gathering.

Patchwork provides a configurable management infrastructure (§II) for capturing traffic that crosses FABRIC's dataplanes. Arbitrary ports on the switch can be mirrored, but we currently focus on mirroring uplinks to capture inter-site traffic. Patchwork's configurability enables programmable capture and processing—this includes specifying sites where to capture, which uplinks to mirror, what types of traffic to capture, how frequently, and how to process it. Processing can include truncating payloads and blanking addresses.

Since a FABRIC site may have multiple uplinks (§IV), Patchwork by default discovers the number of uplinks and can be set to sample all of them or a subset. Constrained by FABRIC's design, currently we need a separate NIC to handle each uplink. Since each NIC has two physical ports, if $u$ is the number of uplinks we wish to sample then we evaluate $\lceil u/2 \rceil$ to calculate the number of NICs needed. Patchwork's management infrastructure helps users navigate and make these choices. With regards to multi-tenancy, since the profiler uses its own NIC ports, it does not impact the network traffic of other users. Each VM used for sampling uses 2 cores which is 0.4% of average CPU availability at a site. Similarly the RAM and secondary memory used is minimal.

### B. Patchwork Implementation

FABRIC provides a Jupyter Hub environment for interacting with the testbed and running experiments. Each user is provided a Jupyter Hub server within which users can create notebooks with cells that run Python code, parse Markdown format, and run shell commands.

Patchwork provides a Jupyter notebook in which users can set parameters for traffic sampling. The notebook then carries out a number of steps that conclude with the submission of an experiment to FABRIC. Once the experiment starts, dataplane traffic starts to get sampled. These steps involves querying FABRIC to discover the available sites, enumerate their uplinks, and identify suitable NICs at each site. Based on available resources, VMs are started to receive, sample, and process mirrored traffic from each uplink. Captured traffic is ultimately saved as a pcap file, which is later downloaded for offline processing.

Once pcap files are downloaded, they are run through another series of scripts that create a traffic profile. This profile is based on analysis that is done using the mature protocol dissection implementations in tshark/Wireshark, which is integrated with Patchwork's tooling for automated analysis.

### C. Using Patchwork

We now describe the typical workflow when using Patchwork. This workflow is structured into four phases.

*1) Setup phase:* The Patchwork VM requests 2 CPU cores, 8GB of RAM and 35G of storage—all these parameters are easy to change in the Patchwork notebook. Patchwork partitions the experiment into three slices. Each slice covers a fraction of the uplink sites. This partitioning is done to avoid the overhead of requesting resources for all the sites in a single operation, since that can cause a timeout in the FABRIC backend and thus fail to allocate the resources.

*2) Sampling phase:* We use tcpdump to capture a sample of the packets that arrive at the NIC port at configurable intervals. To lessen the risk of packet drops, we set the capture buffer to be 32MB; this can be tuned as needed. Packets are truncated to lower capture overhead and protect user privacy. Patchwork does not require payloads for its analysis. The truncation size is a parameter that can be set in Patchwork.

Patchwork also creates logs at every VM to capture a variety of network- and host-related statistics that can help us notice and diagnose any problems. These logs are included in the final compressed file that is gathered for offline processing.

*3) Gathering phase:* In parallel to the sampling phase, a watchdog process checks for both successful and unsuccessful termination—e.g., in case the VM ran out of storage. Upon termination, the saved pcap files are collected and downloaded. The time taken for the gathering phase depends on the amount of data that is written to the pcap files on the listener VMs during sampling, as well as on the duration of the experiment. For runs lasting 24 hours, it took around 20 minutes to download the tarred pcap files from the VMs.

*4) Analysis phase:* Once downloaded, the data is processed by a series of scripts that abstract field details to produce abstract representations of headers. This data is then process by other scripts to graph the data. Fig. 3, 4 and 5, described in §VI-B, were produced by these scripts.

## VI. Evaluation and Findings

In the experiments described in this section we ran Patchwork to capture at all FABRIC sites, using as many uplinks as possible.[1] Captures consisted of the 200-byte prefix of each

[1] In some FABRIC sites we were unable to sample all uplinks because there were insufficient NICs (§V-A).

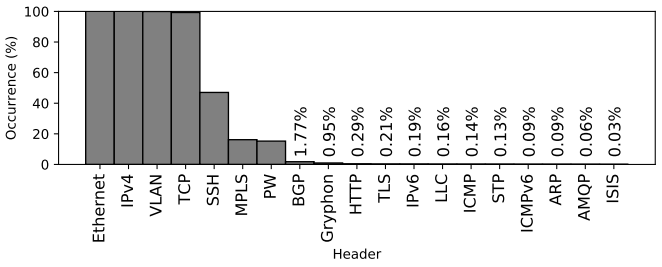


Fig. 3: Occurrence of protocol headers across FABRIC sites. We observe that most traffic consists of Ethernet frames that carry IPv4 packets, that in turn carry TCP segments. Most traffic is overlaid by using VLAN, MPLS, or both.

frame to capture most standard header stacks. Uplinks were sampled for 20 seconds at 5-minute intervals.

### A. Performance and Overhead

Patchwork currently uses tcpdump to capture packets, and the overhead from kernel-to-userspace transitions can constrain the throughput performance of Patchwork. We characterized this upper bound by conducting an experiment on FABRIC during which we generated traffic using iperf3 [5]. At the listening host, we run tcpdump which listens for incoming frames, truncates the length to 64 Bytes and has a buffer memory of 32MB. With this experiment setup, we were able to generate 11 Gbps of throughput between the iperf3 client and server. tcpdump was able to capture packets without packet loss until about 8.5 Gbps of throughput for minimum-size frames. Thus we use this as our listening node's throughput upper bound.

### B. FABRIC's traffic profile

We created a profile of FABRIC's dataplane by sampling all sites during the second half of December 2023. This section presents a profile based on a subset of that data. Because of when we carried out this sampling, FABRIC was quieter than normal.[2] Thus we consider the current profile to be a potential baseline and we plan to create more profiles at a different times of the year, to observe FABRIC's dataplane activity over time.

Fig. 3 shows the occurrence of the most prevalent headers in traffic across all sampled sites—e.g., the graph shows that ICMP shows up in 0.14% of all frames. We observe that observed traffic mostly consists of IPv4 packets, and only 0.19% of frames had IPv6 headers. Most of the observed traffic consists of TCP streams. Approximately half of those streams during the observation period related to SSH connections. Underlying that profile, we observe overlays that rely on VLAN and MPLS. Around 2% of traffic consists of signalling for routing (BGP and ISIS) or loop-avoidance (STP).

Fig. 4 shows site-specific diversity of observed headers. For example, in the traffic from S11.2 we only observed

[2]The average number of active slices per day on FABRIC is 118, with a std.dev of 61. In comparison, during the days of this research the average active slices were 101 with a std.dev of 29.

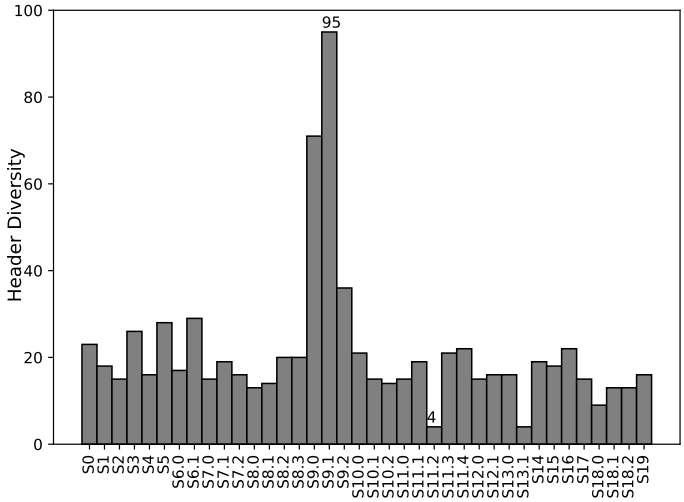


Fig. 4: Diversity of headers across FABRIC sites. S11.2 and S13.1 have the least diversity (4 headers), while in traffic at S9.1 we observed 95 different classifications of headers.

4 headers being used (e.g., Ethernet+VLAN+IPv4+TCP). The occurrence of other headers—e.g., IPv6, FTP, etc—contributes to a higher diversity. We observe that the observed traffic in most FABRIC sites exhibits low header diversity, but S9 has high header diversity across all three of its uplinks. Further analysis is needed to understand the composition of that traffic, and what workloads are contributing to its diversity.

We also observed significant variety in frame sizes across FABRIC sites, which is indicative of different types of workloads. Fig. 5 shows the size profile for a subset of sites. We observe that most sites carry a proportion of smaller packets—e.g., up to 128 bytes. S5 is notable because the majority of its traffic is in that category. Many sites carry minimum-sized frames—e.g.S8.0. Often these frames consist of payload-free ACKs in a TCP stream. For example, we observed one such frame consisting of: Ethernet+VLAN+MPLS+IPv4+TCP. Some sites are also notable for carrying larger frames—and in the case of S0, S8.2 and S9.0 they carry extremely large frames. Further analysis is needed to understand the composition of that traffic and its workloads.

### C. Linux interface counters

A key part of the construction of Patchwork consisted of careful load-scaling and interpretation of OS resources, to optimize packet capture. During the packet capture we noticed reports of packets being dropped by the interface, despite the data rate being much lower than 8.5 Gbps (§VI-A). We were able to correlate this to the rx_dropped counter displayed by the ifconfig and iproute2 utilities.

Since we were not expecting packet drops, we invested effort to understand the cause of this problem in the Linux kernel, and this produced valuable take-aways for our future work. To understand at which level the packets were being dropped, we developed an XDP program [6] that copies the

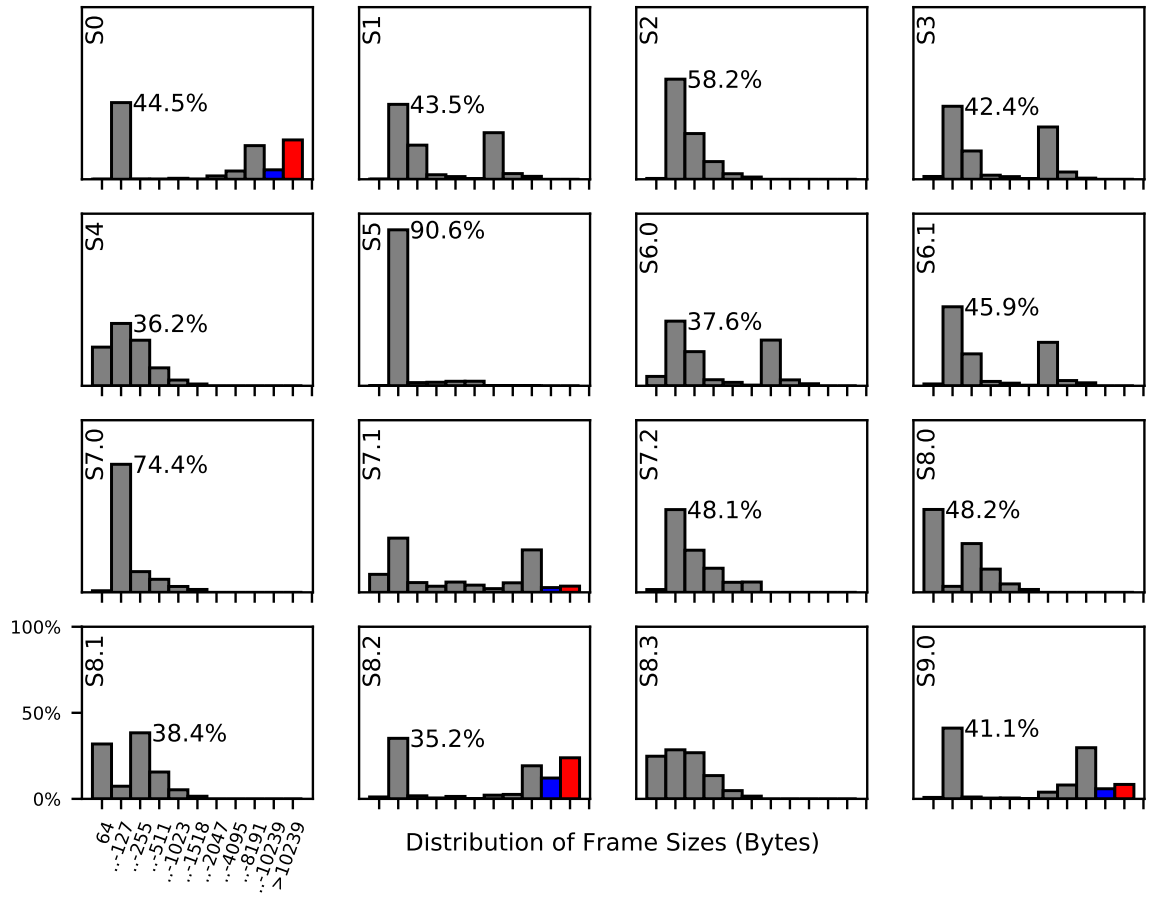


Fig. 5: Histogram showing the frame-size profile of different sites. Sites S7 and S8 have three uplinks each, indicated by their .0-.2 suffixes. The red columns show frame sizes that are extremely small or large. The blue column indicates jumbo frames.

raw packet and writes it to a BPF map perfbuffer that is shared with userspace. We then compare the packet stream as seen by the XDP program with that seen by tcpdump. We noticed that both streams had the same packets: thus packets were not being dropped between the driver and the network stack.

We went through the network driver code and the documentation for Intel and Mellanox NICs, to infer the possible reasons for rx_dropped counters being incremented. In the case of Intel, the rx_dropped counter is set to the rx_discards field as reported by the hardware. rx_discards is incremented generally when packets are dropped due to lack of free rx descriptors available. However since FABRIC uses Mellanox ConnectX NICs as their network cards, we needed to rely on Mellanox documentation for this specific case. Unfortunately the documentation of ConnectX5/ConnectX6 NICs is not public, and as a result we could not understand the hardware register that sets the rx_dropped counter.

Instead we relied on the Linux kernel's interface statistics documentation [7]. From it we learnt that rx_dropped counter increments when (1) the interface runs out of resources or (2) encounters unsupported protocols or (3) when packets are filtered out by the layer-2 handler.

Through further tests we dismissed causes (1) and (3), this only left cause (2). In modern Linux kernels, each layer-3 handler registers itself with the kernel beforehand, so that this operation can take place seamlessly. However if a specific protocol is not registered, then it falls under "unsupported protocols"—which causes the rx_dropped counter to increment. One such protocol was MPLS.

We verified this finding on our local testbed by observing the counter values when the MPLS module in the Linux kernel image is enabled, and comparing them to the counter values observed when the module is disabled. Since we were not relying on the kernel network stack to handle the packets that it was receiving—since we were only intending to sample the traffic—then we concluded that it was safe to ignore the packet drop counter for the low-throughput scenarios.

## VII. Conclusion, Limitations, and Future Work

Patchwork provide programmable profiling of FABRIC's dataplane and relies on FABRIC's simple but effective port mirroring primitive. It complements FABRIC's powerful and flexible Measurement Framework (§I) by focusing on observing the dataplane while experiments are taking place.

The development of Patchwork has revealed interesting insights about **(1)** how to best leverage FABRIC's resources for dataplane observation (§V-A), **(2)** current limits of the gathering system (§VI-A), **(3)** a snapshot of FABRIC's traffic profile (§VI-B), and **(4)** in what conditions the kernel's network-related error indicators provide a false positive (§VI-C).

Although it can already produce results, Patchwork is "work in progress". The rest of this section describes the current limitations of Patchwork and outlines some work to improve the system. We are working to implement these improvements and make Patchwork available for others to use.

One key current limitation is that Patchwork cannot sample at full line rate (§VI-A). Our listening port's link is 100 Gbps, but it can only receive at around 8.5 Gbps. If we mirror packets from an uplink port that operates at a higher throughput, then packets will be dropped at the receiving interface. The loss of packets can lead to some imbalance in the sample. In order to achieve a throughput closer to line rate, we are considering as a possible solution, a sampler built atop DPDK to bypass the kernel network stack.

Even if we can receive at 100 Gbps, since the port mirror service can mirror both Rx and Tx traffic on the target uplink port (§IV), at full utilization we would have 200 Gbps of traffic being mirrored to the Rx channel of a listening port that can at most handle 100 Gbps. In that case, packets will be dropped at the switch. There is no way for the listening node to identify this potential scenario without querying the counters on the switch. Patchwork is currently unable to retrieve information about switch resource constraints, which would help it validate user-chosen parameters—for instance, by ensuring that Patchwork is not under-resourced to sample at the full line rate. If FABRIC switch counters are made available, then we can use them to provide additional configuration-time and runtime checks in Patchwork. Further the scalability of Patchwork is currently constrained by the number of available SmartNIC ports at each site.

In the current setup Patchwork only mirrors uplinks. Thus it observes cross-site traffic, but does not observe traffic travelling between workers in the same FABRIC site (§IV). Changing Patchwork to mirror intra-site ports is straightforward, and we plan to explore this in the future.

Patchwork currently does not export or import data with other systems. In the future, we hope to explore the integration of Patchwork with FABRIC's Measurement Framework to enable FABRIC users to observe both host resources and dataplane resources more easily. We also hope to explore the integration with NetSage and perfSONAR for Patchwork to consume or produce diagnostic information to better support the observability of FABRIC users and operators into the network infrastructure.

A current limitation of Patchwork is that it still involves some manual steps, but we plan to automate these when the system is more mature. The Jupyter Hub server provided to the FABRIC user is a container instance with a volume-mounted file system for local storage. This file system's size is limited to 1 GB, and this creates a bottleneck for long-duration experiments (1 day and above) across multiple FABRIC sites using Patchwork, since the tarred pcap files from each VM goes well above this storage limit. In our work so far, this necessitated a manual step of transferring a pcap download script onto the local system, as part of the Patchwork gathering phase. Therefore, instead of downloading the pcap files from each VM onto the Jupyter Hub server filesystem, we download it onto our local computer which has abundant storage.

*Acknowledgement*

We thank the anonymous reviewers, Ilya Baldin, Jim Griffioen, and Alexander Wolosewicz for feedback. This work was possible thanks to technical support from the FABRIC operators. This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0106. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funders.

## References

[1] 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.

[2] "FABRIC Experiment Measurement APIs (MFlib)," https://learn.fabric-testbed.net/article-categories/mflib-api/.

[3] K. V. Vishwanath and A. Vahdat, "Evaluating distributed systems: Does background traffic matter?" in *2008 USENIX Annual Technical Conference (USENIX ATC 08)*, 2008.

[4] O. A. Adeleke, N. Bastin, and D. Gurkan, "Network Traffic Generation: A Survey and Methodology," *ACM Comput. Surv.*, vol. 55, no. 2, jan 2022. [Online]. Available: https://doi.org/10.1145/3488375

[5] ESnet, "iperf3 at 40Gbps and above," https://fasterdata.es.net/performance-testing/network-troubleshooting-tools/iperf/multi-stream-iperf3/.

[6] Matt Fleming, "A thorough introduction to eBPF," https://lwn.net/Articles/740157/.

[7] The Linux kernel development community, "Interface statistics," https://www.kernel.org/doc/html/latest/networking/statistics.html.