Patchwork: A Traffic Capture and Analysis Platform for Network Experiments on a Federated Testbed

Nishanth Shyamkumar
Illinois Institute of Technology
Chicago, USA

Hyunsuk Bang
Illinois Institute of Technology
Chicago, USA

Bjoern Sagstad
Illinois Institute of Technology
Chicago, USA

Prajwal Somendyapanahalli Venkateshmurthy
Illinois Institute of Technology
Chicago, USA

Sean Cummings
Illinois Institute of Technology
Chicago, USA

Nik Sultana
Illinois Institute of Technology
Chicago, USA

Abstract

Today's federated network testbeds enable experiments of unprecedented scale and detail, but only provide rudimentary primitives to capture an experiment's network traffic. Capturing and analyzing traffic is important for diagnosing and debugging research prototypes and for evaluating research, but today's testbed users divert and duplicate effort to craft custom solutions for their experiments. Building a general, reusable system is made more challenging by the autonomous structure of federated testbeds and by their high capacity links (requiring accelerated processing).

This paper describes the design, implementation, and evaluation of Patchwork: a user-provided, open-source, capture and analysis platform that runs on the state-of-the-art FABRIC testbed. Patchwork works both for *individual* experiments and also for *all* experiments occurring simultaneously on FABRIC. To attain a general design, Patchwork *itself* runs as an experiment on FABRIC and did not require modifications to FABRIC. For scalability, Patchwork offloads logic to FPGA NICs on the testbed and uses DPDK. Patchwork has been used by individual FABRIC users and has been running on FABRIC for over a year to produce a testbed-wide analysis of how researchers are collectively using the testbed's network. This paper also presents that analysis and discusses implications for future research on measurement.

IMC '25, Madison, WI, USA

ACM ISBN 979-8-4007-1860-1/2025/10
https://doi.org/10.1145/3730567.3764462

CCS Concepts

• **Networks** → **Network measurement**; **Network experimentation**.

Keywords

Network Testbeds; Traffic Profiling

ACM Reference Format:
Nishanth Shyamkumar, Hyunsuk Bang, Bjoern Sagstad, Prajwal Somendyapanahalli Venkateshmurthy, Sean Cummings, and Nik Sultana. 2025. Patchwork: A Traffic Capture and Analysis Platform for Network Experiments on a Federated Testbed. In *Proceedings of the 2025 ACM Internet Measurement Conference (IMC '25), October 28–31, 2025, Madison, WI, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3730567.3764462

1 Introduction

> "The ARPANET began operation in 1969 with four nodes as an experiment in resource sharing among computers."
>
> *Denning [21]*

Shared network testbeds are important platforms for evaluating research ideas at scales that exceed what a typical university testbed provides [11, 17, 29, 31]. Successive generations of testbeds—such as PlanetLab [19], Emulab [42], GENI [16], and CloudLab [37]—raised the standards for providing their users with scalable, configurable, and observable environments for research and learning [24, 39].

FABRIC [13] is a state of the art, international network testbed that is intended for research on high-performance and software-defined networking. It includes terabit-scale links [10] and researcher-programmable infrastructure [4]. FABRIC became operational in October 2023 after several years of development.

Providing control over experiment resources is a key feature of a network testbed, and researchers exploit this control to gather data about their experiments. For example, researchers evaluating WAN-oriented congestion control algorithms would observe completion times at hosts and buffer utilization in switches across a wide geographical area and under different types of transfer workloads and background traffic. On FABRIC, such a transfer could be set up between FABRIC sites in Amsterdam and Tokyo, and use links that cross the Atlantic Ocean, continental US, and Pacific Ocean.

As an improvement over its predecessors, FABRIC's Measurement Framework (MF) [2] collects measurements about the utilization of the testbed's resources—including CPU cores and memory from hosts and port counters from switches. Continuing the previous example, MF can be used to poll the counters of switches along the experiment's network path.

However, there remains a need for more observability of the network's *data plane* to obtain detailed measurements about network traffic. Continuing on the earlier example, switch counter information is coarse—it does not distinguish the traffic of one experiment from that of another, nor does it provide a granular breakdown of different traffic types.

For more fine-grained data plane monitoring, FABRIC's API provides a *port mirroring* primitive that supports full traffic capture. This low-level primitive simply clones frames to a second switch port. Using this primitive, researchers can sample the traffic on a subset of ports at each switch along an experiment's path. Using port mirroring, researchers can obtain a traffic sample that reflects the composition of their traffic stream. Continuing on the example, this capture allows researchers to inspect headers to understand the behavior of congestion control through the values of header fields.

But FABRIC's port mirroring primitive is a building block, not a solution. It necessitates substantial user-supplied logic to be usable. This logic would need to handle: **(1)** filtering to exclude unwanted traffic; **(2)** varying parameters for sampling, such as sampling frequency and duration; **(3)** truncating frames to a researcher-specified size (if researchers wish to discard payloads); **(4)** scaling to line rate; **(5)** detecting incomplete samples because of congestion in the switch. Specifically, since port mirroring clones packets from the Transmit (Tx) and Receive (Rx) channels of the mirrored port to a single egress port,[1] the mirroring might overflow the egress port's queue if "Tx rate + Rx rate > line rate". Researchers must devise a mechanism to detect or mitigate this behavior. **(6)** Researchers often must carry out close-to-source traffic processing—such as anonymization [33].

[1]Each FABRIC site consists of an Ethernet that uses point-to-point, duplex links between the site's switch and its servers. Section 3 provides more background on FABRIC's architecture.

It is impractical for each FABRIC user to fully implement this logic for themself. Having a general implementation of this logic would serve the needs of different FABRIC users. In addition to serving individual FABRIC users, a system for testbed traffic capture and analysis would serve the research community collectively by **(1)** providing testbed-wide analysis on the workloads that are utilizing the testbed's network, to study its current operation and investigate ideas for future upgrades—similar to how other networks are analyzed (Section 2)—and **(2)** providing anonymized samples from the testbed. Unlike other intercontinental-scale communications infrastructure (such as private backbones, telecom networks or inter-datacenter links) which provide infrequent and redacted traces, federated testbeds could serve as regular sources of high-fidelity network traces.

This paper describes *Patchwork*,[2] a network *profiler* that captures and analyzes FABRIC's testbed traffic. Patchwork is not part of FABRIC—it runs as an experiment. Patchwork was built by FABRIC users and it relies on existing FABRIC features to provide a general, configurable, and scalable network profiler for testbed users. For scalability, Patchwork offloads logic to Alveo FPGA NICs in FABRIC and uses DPDK.

To our knowledge, Patchwork is the first user-deployed extension to FABRIC. Patchwork can sample the network on any FABRIC site and provides FABRIC users with a simple interface for data plane observability. In addition to presenting Patchwork's design, this paper also describes a profile of FABRIC network traffic gathered using Patchwork during FABRIC's first year of operation (between December 2023 and December 2024). Patchwork now runs weekly to create a profile of FABRIC's network traffic,[3] in addition to being directly usable (at any time) by FABRIC's users.

Summary of Findings. Section 8 details two types of findings: **(A) Deploying a Profiling Tool on a Shared Testbed: (A1)** The testbed's federated structure provided a natural decomposition for Patchwork: every site runs its own Patchwork instance. **(A2)** The profiler must gracefully accommodate the varying number of resources that are available at each site. Patchwork implements 3 ideas: bounded resource usage, lowering resource requests through back-off, and using fewer NICs to capture traffic by cycling between mirrored switch ports. **(A3)** FABRIC's testbed resources can be used to offer testbed *extension* services. Patchwork uses port mirroring and smart NICs to capture traffic, SNMP telemetry to detect loss at the switch, and FPGA NICs for packet processing. **(A4)** In FABRIC's current host architecture, small frames can sometimes be captured and processed faster than they can be stored. **(A5)** The testbed is a convenient distribution platform, and it was easy for FABRIC users to

[2]https://packetfilters.cs.iit.edu/patchwork/

[3]https://packetfilters.cs.iit.edu/patchwork/dashboard/

start using Patchwork. **(B) FABRIC's Network Profile: (B1)** FABRIC sites have diverse traffic characteristics, suggesting diverse yet persistent workloads in those sites. We hypothesize that this is because of researchers wanting to use pools of equipment at those sites, and because of those sites' network connectivity—FABRIC's sites are not identical to each other. **(B2)** Most FABRIC sites exhibit a low variety of protocols in their traffic, but some sites use many types of protocols. **(B3)** FABRIC link utilization is often low, but it sometimes spikes to capacity. Background network activity is highly variable. We discuss how this has implications for testbed research on high-performance networking and its reproducibility. **(B4)** Following from the previous point, for high-performance network experiments, a QoS or resource scheduling policy will be needed to ensure faithful experiment results. **(B5)** Jumbo frames are highly prevalent in FABRIC's network in contrast with other networks such as datacenters and telecom backbones. **(B6)** IPv4 is the dominant network protocol used on FABRIC—surprisingly, IPv6 traffic makes up less than 2% of FABRIC traffic we sampled.

Contributions. This paper contibutes the following:

- A study of FABRIC's infrastructure and resource utilization (**Section 5**). This study informs the design of Patchwork but also conveys to the research community new details about how FABRIC is used.
- Design of the Patchwork network profiler (**Section 6**).
- Working implementation of Patchwork (**Section 7**).
- A profile of network traffic gathered using Patchwork during FABRIC's first year of operation (**Section 8**).
- Patchwork's code and documentation are freely available, and have been used outside the Patchwork team.

This paper substantially extends earlier work [40] that described a much shorter profile of FABRIC's network traffic (3 weeks compared to 13 months). The earlier work did not describe a reusable tool for other FABRIC users, and used a software-only setup that did not scale to line rate.

2 Related Work

Compared to related work, this paper describes (1) the most recent analysis of a shared testbed's resources and utilization, (2) the first network profiling tool for a federated testbed, (3) an analysis of the network profile of FABRIC's first year of operation, and (4) the first user-deployed service on FABRIC for the benefit of other researchers.

Analysis of a shared testbed's resources and utilization. Chun and Vahdat [20] studied the utilization of PlanetLab for three months in 2003, focusing on end-host resources such as CPU load, memory, and the utilization of host network interfaces (in terms of traffic sent and received over those interfaces). Kim et al. [28] studied the utilization of PlanetLab between 2005 and 2010. They describe the number of slices running on PlanetLab during that period and the slices' utilization of CPU, memory, and network utilization on end-hosts. More recently, Ricci et al. [36] and Hermenier et al. [26] studied the utilization of the Emulab testbed and its responsiveness—such as the time taken to provision experiments of different sizes and that span different sites. Yi and Fei [43] characterize the links of the GENI testbed in terms of latency and throughput. In contrast, this paper does not focus on the responsiveness of a testbed's infrastructure or the types of experiments that run on it—this paper focuses on the network dataplane of the testbed. In comparison with earlier work, the present paper describes the design of a reusable and scalable network monitoring tool for a federated testbed (FABRIC), and uses that tool to study the utilization and profile of the testbed's network. Network-related data in earlier work consisted of counter values from end-host NICs. In contrast, this paper samples the traffic across the sites of a federated dataplane and builds a picture of the traffic on the network—such as the distribution of frame sizes and the composition of flows. The INSTOOLS system by Griffioen et al. [25] supports the flexible creation of measurement resources for GENI experiments that can sample the dataplane. In comparison, the system described in this paper can work across all experiments currently running on the testbed (as well as sample traffic for individual experiments), can use heterogeneous hardware, and is able to run at line rate.

Profiling other networks. Network profiles have been carried out for networks that form part of the Internet backbone [27], residential broadband access [30], campus [12], datacenter [15, 38] and satellite networks [35]. Given the different purposes for which networks are used, it is unsurprising that the network profile presented in this paper differs from that of other networks. A key, general difference with other networks is that FABRIC's traffic makes heavy use of jumbo frames. It also has high traffic variation (even across different FABRIC sites) owing to the different research experiments that are taking place on the network.

3 Background: FABRIC Testbed

This section outlines FABRIC's design to establish background for the sections that follow. FABRIC is structured into a *federation* of *sites*. As a federated testbed [14], FABRIC consists of racks that are embedded in the networks of different institutions (*sites*). Such institutions include universities and Internet exchange points (IXPs). FABRIC racks are managed centrally but span a wide geographical area—there are FABRIC sites in Asia, Europe, and USA.

A FABRIC rack contains an Ethernet switch that links a number of worker machines. Each machine hosts one or more Virtual Machines (VMs) and is equipped with several network interface cards (NICs). Researchers who use FABRIC

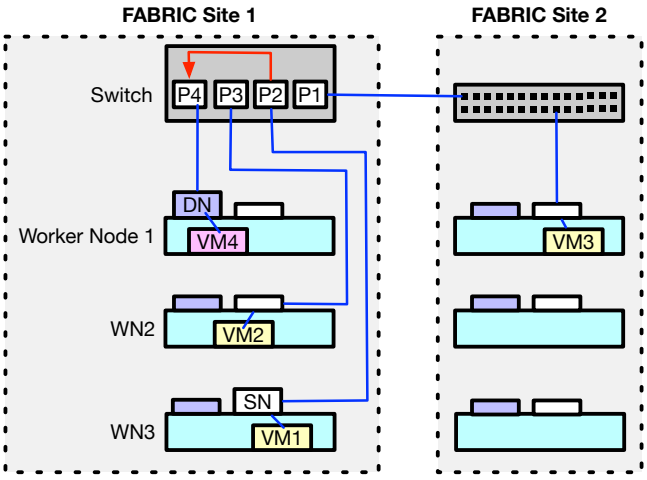

Figure 1: Port mirroring on FABRIC. On the switch at FABRIC Site 1, port P2 is mirrored to port P4 . Port P4 is connected to a *dedicated NIC* (DN) on Worker Node 1, on which a FABRIC user runs VM4 to receive traffic. The traffic consists of what is being exchanged by (possibly a different FABRIC user's) VM1 on WN3, which is using a *shared NIC* (SN) to access the network. This example is described further in Section 3.

can directly reserve VMs and NICs. In the standard terminology used in network testbeds, researchers create a *slice* that reserves resources for their experiments. Reservable resources are called *slivers*. On FABRIC, slivers can include VMs, different types of GPUs, Mellanox ConnectX NICs, Alveo FPGA NICs, and Tofino switches. P4 [18] is used in several research projects on FABRIC, often targeting DPDK, Alveo FPGA NICs, and Tofino switches.

A FABRIC site is connected to one or more other sites through dedicated uplink ports on that site's switch. All links consist of two uni-directional channels: one for transmitting (Tx) and another for receiving (Rx) frames.

FABRIC sites differ in the types and quantities of resources they offer. For example, FABRIC's NCSA site has 10 single-user ConnectX NICs, a ConnectX NIC that can be shared among 381 users, an Alveo FPGA card, five GPUs and an aggregate of hundreds of cores and terabytes of storage [5]. Inter-site links have different capacities, and their availability is not guaranteed—all links are shared with other FABRIC users, and some links are shared with non-FABRIC users.

FABRIC sites use Cisco 5700 Series or Ciena 8190 as top-of-rack (ToR) switches. Switch configuration and query interfaces are abstracted by an API provided by FABRIC, which in turn provides network configuration, port-level telemetry, and port mirroring functionality to FABRIC users. The research described in this paper uses telemetry and port mirroring. Telemetry consists of SNMP-polled readings that are stored in a Prometheus database and queried through the MFlib [2] front-end API. Both SNMP and port mirroring are commonly-supported features on managed switch platforms.

Port mirroring is an important primitive for network profiling, but it is low-level and therefore requires code to manage it. Port mirroring involves picking a ToR port and choosing whether to mirror either or both of Rx and Tx for that port, and which ToR port to mirror it to. The management code must determine ports of interest (to be mirrored), allocate a NIC and VM to received mirrored traffic, and manage those resources for the duration of the mirroring.

Fig. 1 sketches port mirroring on FABRIC. In that example, a FABRIC user has a slice consisting of VM1, VM2 and VM3. In that slice, that user's VMs are sharing NICs with other FABRIC users. This slice is spread across two FABRIC sites, and port P1 on Site 1 links to Site 2. A—possibly different—FABRIC user set up port mirroring on P2 to collect traffic in VM4 through P4. That user specifies whether to receive P2's Rx or Tx traffic, or both. By default, VM4 will receive copies of *all* the frames that cross P2—not only those of VM2.

4 Motivation and Requirements

This section motivates this paper's research and compiles the requirements for a testbed network profiler. Some requirements depend on prevailing usage patterns of FABRIC's resources. Understanding those usage patterns required carrying out a dedicated study. This section formulates research questions for that study and Section 5 presents answers.

A network **profile** describes how a network is being used. A profile is created by observing traffic composition over time. It captures a range of characteristics about a network—including header types, encapsulation patterns, and various flow characteristics. Such characteristics include the sizes and durations of flows, distribution of packet sizes in a flow, and the presence of important control information (such as RST-flagged packets in TCP flows).

Scope. A shared network research testbed is a high-churn environment. At any one time, several researchers are starting experiments while other researchers' experiments are already running. Each experiment is likely to invoke a research prototype. Different prototypes induce different workload shapes over the testbed's network. These prototypes usually evolve to maturity over several months, during which time their network activity can change.

Asymmetry in general profiling. Today's network profiling techniques are inadequate for shared testbed networks because they are designed to provide information to a network's *operator*, not to the network's *users*. Today's approaches include obtaining information from network switches using standards like NetFlow, sFlow, IPFIX, and SNMP. This information does not distinguish between testbed users and provides coarse statistics. In previous work, we set up NetFlow

generation and collection within a single FABRIC experiment to assess the detail we could obtain.

How is network profiling done on research testbeds today? Testbed users typically use tcpdump to capture a packet stream for later analysis, and possibly mutate their experiment's topology to introduce bump-in-the-wire nodes to collect traffic. FABRIC offers a port mirroring primitive, but this is unwieldy to use directly: the researcher must find out which switch ports their experiment is using (potentially at several FABRIC sites), and then set up the resources to receive and record mirrored traffic. FABRIC also provides a query interface to MFlib (see Section 3) which continuously gathers switch information polled using SNMP from all FABRIC switches. This provides a time-series of summary information about the network's state at a high level that complements the profile from data-plane sampling.

Goal. A network profiler for a research testbed would passively gather data related to its users' research experiments. That data can be used to evaluate, diagnose, debug or reproduce testbed experiments, and to study the operation of the testbed and carry out new research on testbeds. The testbed's operator can use this same profiler to help monitor their network. When activated, the profiler would either monitor a *single experiment* (**single-experiment mode**) or *all* experiments in a site or across the testbed (**all-experiment mode**),[4] depending on the access tokens of the invoking user. Testbed users cannot see the traffic of other users unless they are granted access tokens by the operator.

Requirements and Open Questions

This section compiles a set of requirements for a testbed profiler. Some requirements depend on understanding testbed characteristics for which there is no published or public information. A study is therefore needed to answer these questions and complete the requirements. The study's results are presented in Section 5 to set context for Patchwork's design. Requirements are labeled as **(R*number* : *name*)** and an open question for a requirement is labeled as **(R*n*.Q*m*)**.

(R1: Resource Sensitivity and Scheduling) Should the profiler use resources on one site to profile another site, or should each site provide for its own profiling needs? Sites might have different quantities of resources available—in which case, what network ports should the profiler prioritize? Before picking a direction for the profiler's design, we must first understand FABRIC's resources and how researchers use them. **(R1.Q1)** What is the ratio of uplinks to downlinks? Should we focus on sampling uplinks only, downlinks only, or both? **(R1.Q2)** How distributed are users' slices across different sites? Using a profiler requires navigating trade-offs on how to gather and process network traffic. 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*. **(R1.Q3)** When should the profiler sample traffic, and for how long?

(R2: Testbed service overlay) The profiler must not require additional resources to be added to the testbed, or changes to the testbed's API. Adding resources would incur cost and management burden for the testbed's operator. Thus, the profiler must work within the resources that already exist on the testbed. **(R3: Resilience and Detecting Failures)** The profiler must work in a federated system and survive partial outages and transient disconnections. The profiler must therefore implement failure and recovery modes, and gather information for diagnosing runtime problems. **(R3.Q1)** Based on historical or telemetry information, should we rank sites in terms of their likelihood of being network-active? Which sites should we sample? **(R4: Performance)** The profiler must be able to ingest a high volume of traffic from across the federated testbed. **(R4.Q1)** What is the peak transmission rate across all port switches on FABRIC? **(R5: Tunable Fidelity)** users of the profiler must have a simple interface for configuring its behavior and level of detail.

5 Resource & Infrastructure Study of the FABRIC Testbed

This section presents a study that answers the questions from Section 4 to complete the technical requirements for a network testbed profiler. The information for this study came from three sources: (1) FABRIC's *information model* [3] encodes the testbed network's topology (similar to Google's MALT [32]). (2) Anonymized slice creation statistics were shared with us by the FABRIC operator. (3) Telemetry information about every switch on the network testbed was obtained by querying FABRIC's MFlib (described in Section 3).

Uplink distribution on FABRIC. The distribution of uplinks on FABRIC provides information about the connectedness of each FABRIC site. In turn, that can inform decisions into which sites to prioritize for profiling and whether to focus on uplinks, downlinks, or both—answering (R1.Q1). We analyzed FABRIC's information model to count ports at each site. We found that most sites have a similar number of uplinks, and all sites have many more downlinks than uplinks. The results are graphed in Fig. 2. The utilization of uplinks and downlinks depends on whether slices tend to be located in a single site (which would utilize downlinks more) or distributed across sites (which would utilize uplinks), as asked in (R1.Q2). This is analyzed next.

Slice activity on FABRIC. Fig. 3 answers (R1.Q2): we found that 66.5% of slices use resources in a single site—therefore

[4] An instance of the "zero-one-infinity rule" [9].

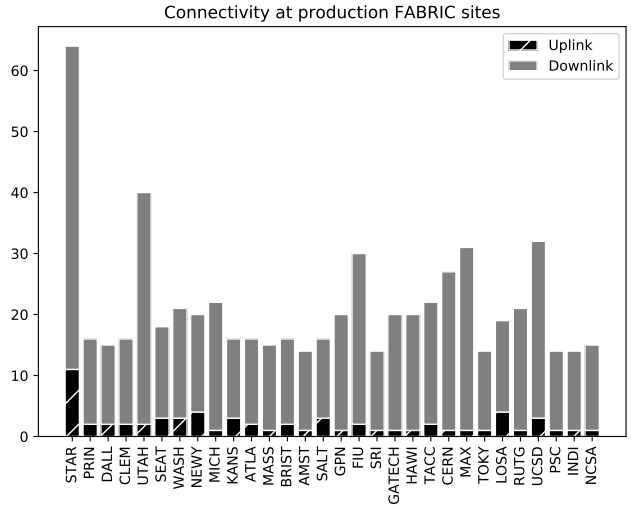

Figure 2: Distribution of ports across all production FABRIC sites. Downlinked ports are connected to FABRIC servers at the same site. Uplinked ports are connected to other FABRIC sites' switches.

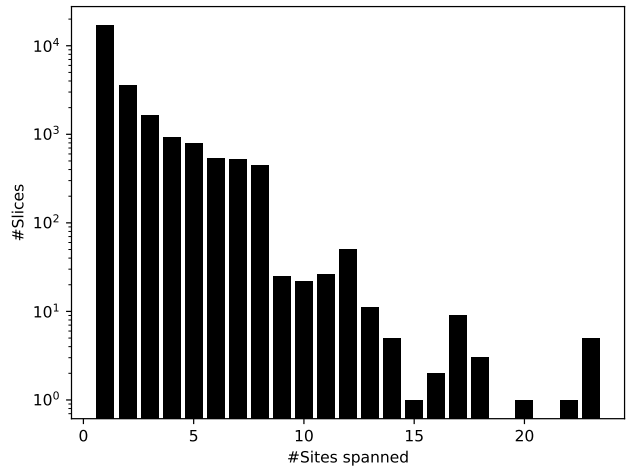

Figure 3: This graph shows that FABRIC slices tend to use resources that are spread across few FABRIC sites. 66.5% of all FABRIC slices use a single site.

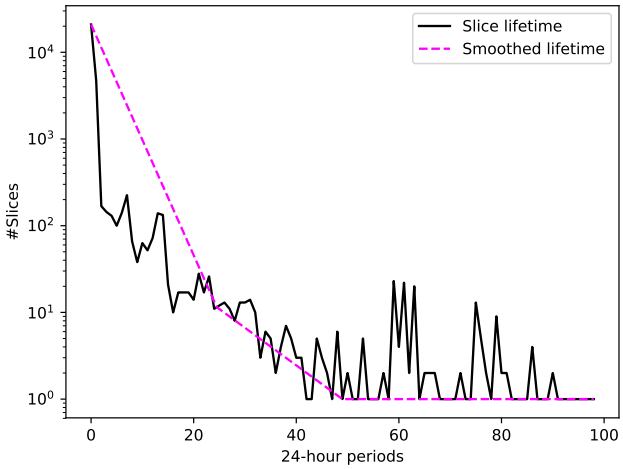

Figure 4: Duration of slices on FABRIC. 75% of slices last for 24 hours. The smoothed graph is produced using linear interpolation after sparsing the data points.

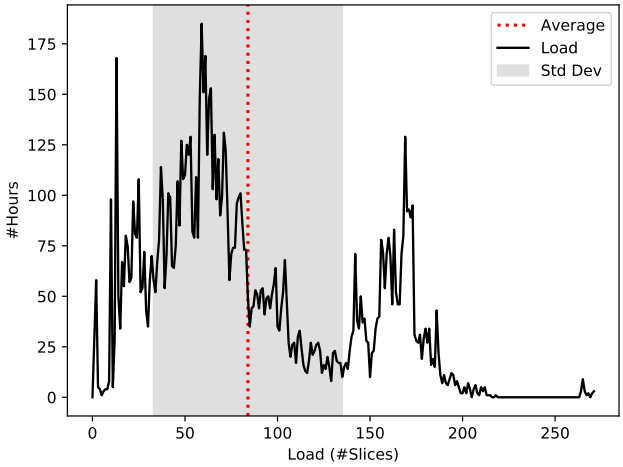

Figure 5: Average number of slices on FABRIC is 85, with a standard deviation of 52. At most, we saw 272 simultaneous slices on FABRIC.

the network activity of most slices is unlikely to cross the uplinks. Notwithstanding, a substantial subset of slices (43.5%) use resources across different sites, therefore the profiler must be able to sample *both* uplinks and downlinks. Precisely *which* links to sample depends on the goal of the profile—for example, we could sample links at random, or we could concentrate on the links that have highest traffic rates. The design must let the user of the profiler choose. From analyzing slice lifetimes in Fig. 4, we see that 75% of slices last for 24 hours. Therefore in answer to (R1.Q3), in all-experiment mode, it appears that 12-24 hours seems adequate to get a good sample of the testbed network's behavior. Further, in Fig. 5 we see that FABRIC has an average of 85 slices active at any one time, with a standard deviation of 52. Thus, the testbed is always active—but we will soon see that it becomes especially active in the run-up to important deadlines!

Network activity on FABRIC. Not all testbed experiments use the network heavily. We now turn to analyzing the network's utilization in time and space. This analysis is based on 5-minute samples of Tx and Rx rates for all switch ports

at every FABRIC rack. We extracted time-series information from MFlib and characterized the activity across FABRIC switch ports. We found which ports are more active than others, and that there are occasional workloads on FABRIC that generate plenty of traffic both within and across sites. Continuing on our analysis for (R1.Q3), if we cannot sample from all ports (because insufficient resources are available to the profiler) then the profiler must accept ordering heuristics to avoid "starving" less-active ports. If we only sample from the busiest ports, we might miss important traffic.

Referencing (R3.Q1): FABRIC users can choose the sites in which to place their resources, and can choose the sites with the lowest loads, *but* those sites would become more loaded if other users also choose them. Therefore, rather than seeking to *avoid* runtime problems (that are beyond our control) it is a better policy to be able to *detect* them.

Fig. 6 uses the data obtained from MFlib to graph the data-transfer activity in FABRIC's network during most of 2024. The network's activity peaked the week before the Supercomputing'24 conference [6], a venue where several FABRIC users were presenting talks and demonstrations of their research. During that week, an average of 3.968Tbps crossed FABRIC's network. The main take-away for (R1.Q3) is that the network activity appears to be correlated with key deadlines, and we can observe a "ramp-up" period to April and November.

Turning to (R4.Q1), we found that 50% of switch ports have utilization ≤38%, but there are ports that run at line rate (i.e., they are 100% utilized). It is therefore prudent to expect to need to capture traffic at line rate.

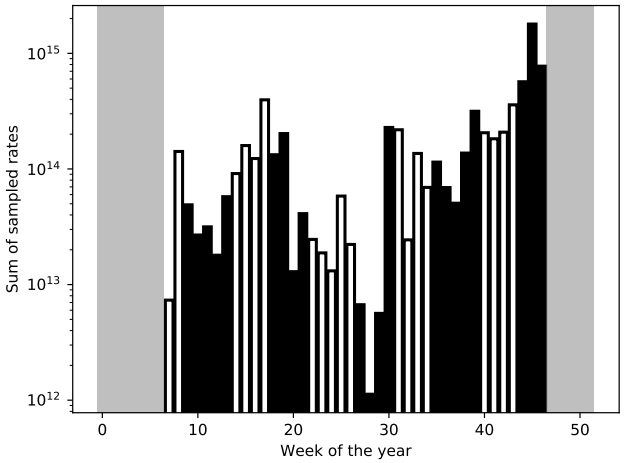

Figure 6: Utilization of FABRIC's network over each week of 2024. The gray bands are intervals for which we have no information. Months alternate between black and white sequences of bars, starting with the last two weeks of February. For each week, the y-value consists of sums of 5-minute rate (bytes-per-second) samples from each switch port during that week.

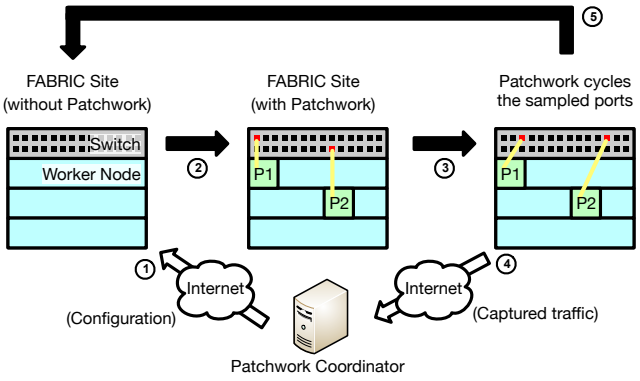

Figure 7: Outline of how Patchwork works. Port cycling changes the mirrored port while keeping fixed the NICs and VMs. Cycled ports take turns to be sampled.

6 Design of Patchwork

This section describes the design of Patchwork, how it meets the requirements listed earlier, and the design's limitations.

Patchwork is a network profiler that works across FABRIC's federation of sites. This profiler samples ports on FABRIC switches and manages the entire profiling process end-to-end. Managing the profiling process involves provisioning testbed resources, deciding which ports to sample, and processing and analyzing the sampled traffic.

Patchwork decouples traffic sampling from analysis. Sampling and pre-processing (e.g., truncation) of traffic is done on FABRIC. Analysis takes place outside of FABRIC, and processes the packet captures to produce statistics and graphs.

6.1 Overview

To FABRIC, Patchwork appears like any other experiment. It runs on FABRIC using the resources that the testbed makes available to its users. Patchwork relies on FABRIC for access control to the testbed's resources. These resources include VMs, dedicated NICs, and port mirrors. FABRIC's access control and virtualization systems preserve the isolation between slices, privacy of the testbed's users and the security of the overall system. To run in all-experiment mode, a special, discretionary permission is required from the FABRIC team. This permission allows the mirroring of traffic that belongs to other testbed users. In Appendix A we describe the process that we followed to obtain this permission. Patchwork's access and use of resources is completely encapsulated by FABRIC's management interfaces, and that is how Patchwork satisfies requirement (R2: Testbed service overlay) from Section 4.

Patchwork creates slices on FABRIC that include VMs for traffic gathering and pre-processing, dedicated NICs for receiving traffic for those VMs, and uses port mirroring at the site's switch to direct traffic to those NICs from ports of interest. Patchwork also uses MFlib's switch telemetry at runtime to detect congestion at mirrored ports, and logs this as part of the profile.

Fig. 7 outlines Patchwork's operation. A *coordinator* ① running outside FABRIC configures and ② starts Patchwork in sites across FABRIC. If run in all-experiment mode, Patchwork is started on all FABRIC sites; if run in single-experiment mode, then Patchwork is started on sites on which that slice is using resources. Within each site, Patchwork can start fewer sampling instances than there are ports to sample. To sample all ports of interest, Patchwork ③ *cycles* between ports to build its profile. Cycling simply changes the mirroring at the switch as sketched in Fig. 7—the Patchwork VMs P1 and P2 persist throughout the profiling. Finally, the samples are ④ downloaded to the coordinator, then Patchwork's resources are ⑤ yielded back to FABRIC. The workflow description below outlines the resource selection methods used by Patchwork to meet requirement (R1: Resource Sensitivity and Scheduling).

6.2 Patchwork's workflow

Patchwork's workflow is structured into four phases.

6.2.1 Setup phase. The coordinator shown in Fig. 7 determines which sites to profile and formulates a set of resources that will be requested from the testbed to run the profiler at those sites. In all-experiment mode, Patchwork usually profiles all FABRIC sites but it can be focused to run on specific sites. In single-experiment mode, Patchwork is run on the sites and slices that the FABRIC user can already access. Within each site, the coordinator can run several Patchwork profiling instances—the number of instances is commensurate with the number of switch ports to be mirrored.

Related to (R1: Resource Sensitivity and Scheduling), for testbed-wide profiling, Patchwork acquires as many resources as needed. In the default case, each Patchwork's listening node requests 2 CPU cores, 8GB of RAM, 100GB of storage and 1 dedicated, dual port NIC. Of these, the most scarce resource is the dedicated NIC since each site usually has only around 2-6 available. Patchwork discovers the resources that are available at each site by querying FABRIC's APIs. To meet requirement (R3: Resilience and Detecting Failures) from Section 4, each instance of Patchwork is independent from the others. While running, instances do not require coordination. Patchwork uses iterative back-off during resource acquisition at each site: that is, if the requested resources are not available, then Patchwork will scale down its request. For example, if the site has 4 dedicated NICs available, then there are 8 physical ports available. Thus, Patchwork can profile 8 switch ports at any given time. However, while setting up the system, if there is no storage available, then Patchwork will reduce its request by 1 VM and 1 dedicated NIC. On that site, if the slice request succeeds after 2 iterative back-offs, then Patchwork can monitor 4 switch ports (since at each back-off, a dedicated NIC with 2 ports is reduced from Patchwork's request). Scaled-down operation involves starting fewer instances that monitor the same number of ports—trading off resources for sample quality. To meet requirement (R4: Performance), Patchwork can offload operations like sampling, truncation, filtering, and pre-processing to Alveo FPGA cards, and uses a custom DPDK application to receive and store traffic. DPDK [22] is a framework for running high speed networking applications on a CPU. It bypasses the Linux network stack, and provides a user-space library and APIs optimized for networking performance.

6.2.2 Sampling phase. During this phase, Patchwork captures traffic from mirrored ports. This capture is controlled by user-provided parameters to meet (R5: Tunable Fidelity), and its behavior is sketched in Fig. 8. The raw data that forms a *profile* consists of traffic captures carried out during a series of *runs*, where each run consists of a series of *samples*. The user sets the duration of each sample, number of samples in each run, and the number of runs between cycles. The user also configures packet truncation size and capture pre-processing—such as blanking or transforming addresses.

Frames are captured using one of three methods: (1) tcpdump with the capture buffer raised to 32MB, (2) a custom DPDK application, and (3) preprocessing on an Alveo FPGA NIC then serialization to storage by the custom DPDK application. All three methods produce pcap files.

During the sampling phase, a watchdog process checks for both successful and unsuccessful termination of the Patchwork instance—e.g., in case the FABRIC VM hosting a Patchwork instance ran out of storage. To meet (R3: Resilience and Detecting Failures), Patchwork creates logs at every instance to capture a variety of network- and host-related statistics that can help users notice problems. (For example, this information helped us notice and dismiss a quirk of how the Linux kernel accounts for "dropped" packets when associated kernel modules are missing [40].) These logs are included in the final compressed file that is later gathered by the coordinator for offline processing—this is shown as step ④ in Fig. 7.

Port cycling. To meet (R1: Resource Sensitivity and Scheduling), Patchwork supports *cycling* between mirrored ports in order to use fewer FABRIC resources. If we were constrained to only running a single Patchwork instance on a site, then that instance could cycle between all the ports of that site's switch. If we could run two instances, then each

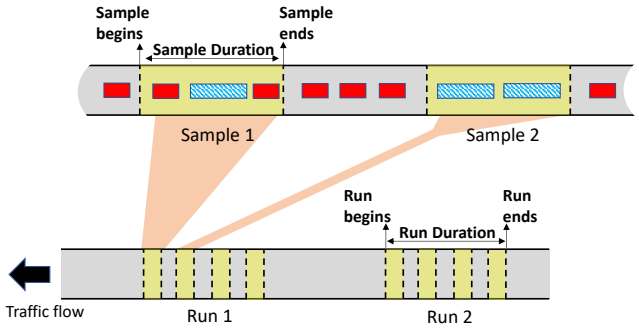

Figure 8: Outline of Patchwork's sampling procedure, described in Section 6.2.2. Traffic flows to a NIC that is dedicated to Patchwork through the mirrored switch port.

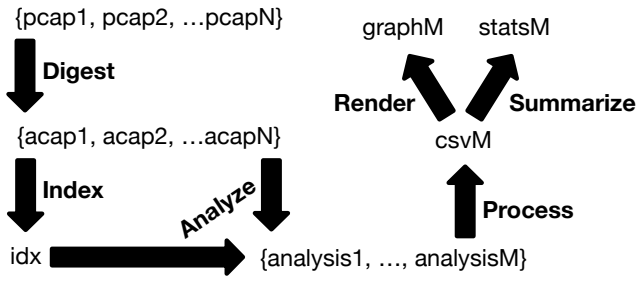

Figure 9: Packet captures are preprocessed at source, then put through this analysis pipeline to produce graphs and statistics, as explained in Section 6.2.4.

switch port would be twice as likely to be sampled. The cycle duration and port selection method are user-supplied. By default, Patchwork uses a "busiest ports bias, $\frac{1}{n}$ other non-idle port" heuristic—that is, during every $n-1$ cycles it picks a random non-idle port, and during the other cycles it picks the busiest port that has not been sampled during the last n cycles. The busiest port has the highest Tx and Rx rates during a recent (and configurable) time interval; this information is obtained from MFlib at runtime. This heuristic was designed to provide fair sampling across all non-idle ports. Other selection methods Patchwork supports include: (1) sampling fixed ports (no cycling), (2) sampling only uplink ports, (3) cycling between all ports, including idle ones. Users can also add their own heuristics.

To meet (R3: Resilience and Detecting Failures), Patchwork detects congestion at the switch, which can lead to incomplete samples. That is, if both Mirrored(Tx) and Mirrored(Rx) channels are mirrored to the Patchwork(Tx) direction of another switch port (to be transmitted to a dedicated NIC controlled by Patchwork), then samples will be incomplete if Mirrored(Tx) + Mirrored(Rx) exceeds the capacity of the channel Patchwork(Tx)—that is, frames will simply be dropped at the switch before they are transmitted from Patchwork(Tx). Patchwork queries the switch for the rates of Mirrored(Tx) and Mirrored(Rx), to infer whether frames are likely being dropped.

6.2.3 Gathering phase. When the sampling phase ends, the captured traffic (as pcap files) and logs are compressed and downloaded to the coordinator as sketched in Fig. 7.

6.2.4 Analysis phase. Once downloaded, the data is processed offline to extract high-level meaning. All the data and graphs in Section 8.2 were produced directly by this phase.

Fig. 9 outlines the Analysis phase. The *Digest* step takes raw pcap files and applies the protocol dissectors used by Wireshark [8] to extract information about each header, discarding unneeded information. Wireshark's protocol dissectors were chosen since they are mature and widely used. Using the dissectors' output, for each frame prefix this analysis produces an abstract stack of headers ("acap"). Since a single profile often produces dozens of gigabytes of data, an *Index* step is carried out to allow subsequent analyses to more quickly locate the acap files needed. The *Analyze* step runs a diverse set of analyses—for example, to characterize frame sizes, the types of headers observed in the captures, and classify flows (represented as sequences of abstract header stacks). Flows are classified by using the virtualization tags (MPLS and VLAN) and network- and transport-layer fields—thus even if the same 10/8 addresses are used in different slices, they are treated as different flows. Timing and frame size metadata is retained from the original pcap. From the results of analyses, the *Process* step produces CSV files that describe different aspects of the profile—such as the distribution of different types of frames across FABRIC sites, and the composition of flows. Finally, this information is processed by other scripts to produce graphs or summary statistics.

6.3 Design Limitations

Patchwork's limitations include the following, which are topics for future work: (1) Resources cannot be shared across Patchwork instances, leading to some underutilization of profiling since, for example, only a single FABRIC user at a time can mirror a specific switch port. Sharing could be achieved by having an intermediate layer that schedules the use of mirrored ports on behalf of more than one FABRIC user. (2) While Patchwork has some dynamic behavior—i.e., port cycling that was described in Section 6.2.2—it does not support dynamic scaling of resources. Except for mirrored ports, all of the resources used by Patchwork are reserved at start-up time. Adding dynamic scaling could improve Patchwork's performance (e.g., by taking advantage of offloading

opportunities that become available at runtime) and flexibility (e.g., by having a "nice" factor [34] for the profiler to scale down its use of resources if the testbed is being highly utilized by other researchers.)

7 Implementation

Patchwork runs on FABRIC as a regular experiment, and its implementation automates the end-to-end workflow described in Section 6. Patchwork is implemented in Python, C, and P4. The system that cycles ports and captures frames on FABRIC consists of 3.2KLOC of Python supported by 0.7KLOC of shell scripts, the DPDK application consists of 0.5KLOC of C, and the P4 application (that is compiled to the Alveo FPGA NIC) consists of 0.6KLOC. The offline analysis is done by around 1.1KLOC of Python, supported by around 350LOC of shell scripts. The visualization code—also used for this paper's graphs—is around 2KLOC in size.

Where possible, Patchwork was implemented to be portable. Currently it uses a FABRIC-specific API to acquire resources, transfer files, and obtain telemetry. Some other testbeds have counterparts for some of FABRIC's features, albeit they have far fewer network resources than FABRIC. Porting Patchwork to other testbeds is potential future work.

8 Evaluation and Findings

This section evaluates Patchwork (Section 8.1), analyzes FABRIC's network profile that was gathered using Patchwork over a period of 13 months (Section 8.2), and discusses lessons learnt from the deployment of Patchwork on FABRIC (Section 8.3). Although we focus on the all-experiment mode here, the considerations are the same for single-experiment mode, with the additional constraint that it can only monitor ports that are involved in the single user experiment.

8.1 Behavior and Performance

Patchwork is evaluated in terms of its behavior on FABRIC's federation of sites and its overhead sensitivity when capturing sequences of frames of different sizes.

8.1.1 Patchwork behavior on FABRIC's federation. Patchwork was run on all FABRIC sites but one—EDUKY was omitted since it is restricted for teaching use and it lacks dedicated NICs. We analyzed the logs generated by Patchwork as described in Section 6.2 to determine its success rate and reasons for failing. During this interval, Patchwork succeeded in profiling all FABRIC sites in 79% of cases. In around 20% of cases, a FABRIC site lacked resources to run Patchwork, and in the remaining cases there was a bug in Patchwork that has since been fixed. The time-series is graphed in Fig. 10. In this graph, "Failed" is due to transient problems with FABRIC's back-end or unavailable resources in a FABRIC site (e.g. no dedicated NICs available). For example, on the days of 10 Sept and 11 Sept most of the failures were due to an issue in FABRIC's back-end system which we were discussing with the FABRIC team and testing their patches. Failures on 15 Sept were due to a different backend error. These were transient backend issues that eventually were resolved. "Degraded" is due to low resources available in a FABRIC site, requiring the scaling-down of requests through back-off. "Incomplete" refers to when Patchwork crashed.

8.1.2 Software-based capture. The default configuration of Patchwork uses tcpdump to capture frames. tcpdump is a mature software program that is widely tested and requires fewer system requirements to deploy. It also provides filtering and truncation through its configuration parameters which make learning and using it simple. For these reasons, we choose this to be the default over the DPDK implementation. We characterized the upper bound of this approach by conducting an experiment on FABRIC during which we generated traffic using iperf3 [23]. The listening host ran tcpdump with a buffer memory of 32MB. tcpdump captured incoming frames and truncated their length to 64 bytes. This setup was able to sustain 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 1500B frames.

8.1.3 Accelerator- and bypass-assisted capture. For higher performance, Patchwork offloads filtering, truncation, sampling, and packet editing logic to an Alveo FPGA card on the FABRIC site being profiled. It also uses a custom DPDK application that serializes frames into pcap files. This introduced a new bottleneck: that of storing information arriving at a line rate of 100Gbps into secondary storage. Frames that bypassed the kernel's network stack must be written to storage by using the kernel's file system. This section reports on the host system's behavior and performance when storing frames into a pcap file at line rate.

Since IO access to storage takes many cycles [7], the Linux kernel employs page caching—also called filesystem caching to reduce the increased latency for applications. Filesystem caching involves representing a secondary storage file into page-sized (4KB) blocks of data. These blocks act just like regular pages in RAM, as part of the virtual memory abstraction provided by Linux, but they are backed by a file residing in secondary storage. For persistence, these pages must be written back to the secondary storage at some point in time.

Linux has default system threshold parameters that dictate when write-backs to the secondary storage from the page cache begin and end. Two such important parameters are the `dirty_background_ratio` threshold and `dirty_ratio` threshold. These thresholds are defined as a percentage of total free cache memory. We observed that the page caching

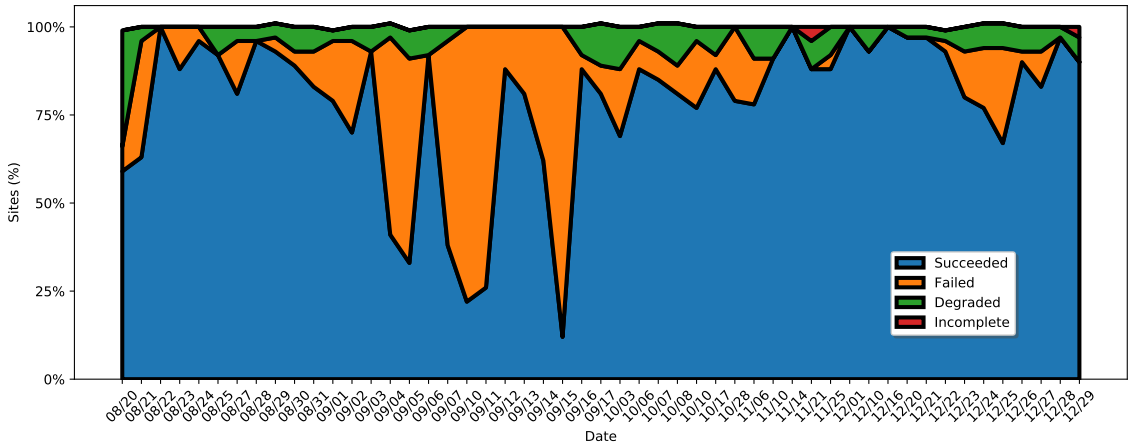

Figure 10: Behavior of Patchwork on FABRIC over an ordinary 4-month period in 2024. This is described in Section 8.1.

mechanism is overwhelmed when frames are arriving at a line rate of 100Gbps.

When the percentage of data in the page cache exceeds the `vm.dirty_background_ratio` threshold, the kernel begins to flush dirty page caches back to the secondary storage asynchronously. When the RAM exceeds the `dirty_ratio` threshold, the kernel blocks processes that are writing to the page cache while the kernel flushes dirty pages back to the secondary storage. Since the processes are blocked by the kernel, performance is degraded since frames are lost. This is evaluated in more detailed in Appendix B.

8.1.4 Scaling packet capture. As mentioned above, when the system memory pressure crosses the midpoint between `vm.dirty_background_ratio` and `vm.dirty_ratio` thresholds, the filesystem cache becomes the major bottleneck. Prior to this midpoint, the performance can be improved by scaling the cores on the system. Another important workaround is the truncation of the packet before writing it to the pcap file. Patchwork's DPDK program writes the first 200 bytes of the frame to the pcap file. There is a minimum write latency associated with this size, and therefore the more data written per packet, the greater is this minimum latency.

We show an experiment measuring the throughput performance when having a truncation length set to 200 bytes and 64 bytes. The thresholds are set to (60:80) and the Rx queue depth is configured to 4096. From Table 1 and Table 2, we observe the performance improves for 64 bytes truncation, requiring fewer cores to achieve the same throughput performance as the 200 bytes truncation.

Table 1: 200B truncation, 60:80 threshold

Frame Size (B)	Rate (Gbps)	Cores	Loss (%)
1514	100	5	0.67
1024	100	10	0.13
512	60	15	0.03
128	15	15	0.1

Table 2: 64B truncation, 60:80 threshold

Frame Size (B)	Rate (Gbps)	Cores	Loss (%)
1514	100	3	0.17
1024	100	5	0.32
512	100	15	0.07
128	28	15	0.13

8.2 FABRIC's traffic profile

Patchwork was run on 69 occasions between December 2023 and December 2024 to gather data from across FABRIC. To gather headers for analysis, the first 200 bytes was captured for each sampled frame. Switch ports were sampled for 20 seconds at 5-minute intervals over a period of 12-24 hours.

Headers. Reading the left y-axis of Fig. 11, we see that FABRIC sites exhibit a range of different dissected protocol headers. Since this graph is produced by using tshark's output, layer-4 ports are often used to classify the payload that follows, and which is counted as a separate header in Fig. 11. This suggests that different FABRIC sites have different network workload characteristics—for example, some sites are more likely to be used for simple throughput experiments while other sites involve experiments that feature different application-layer headers. The duration of the profile suggests that these characteristics are persistent in time across sites. We hypothesize that this is because of researchers wanting to use specific pools of resource at specific sites that are

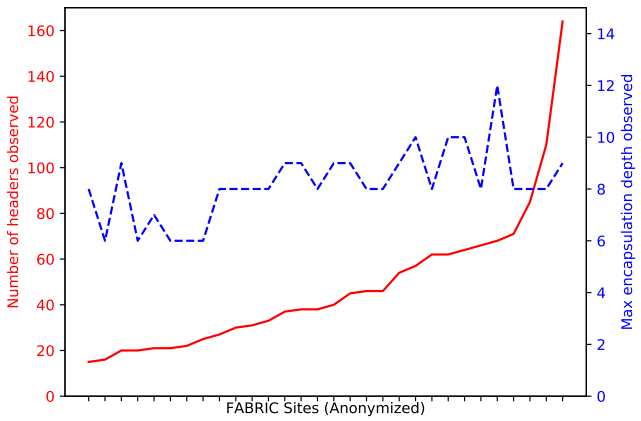

Figure 11: Across all (anonymized) FABRIC sites, this shows (y1-axis) the number of distinct headers observed, and (y2-axis) deepest stack of headers observed. The sites are in the same order between the two curves.

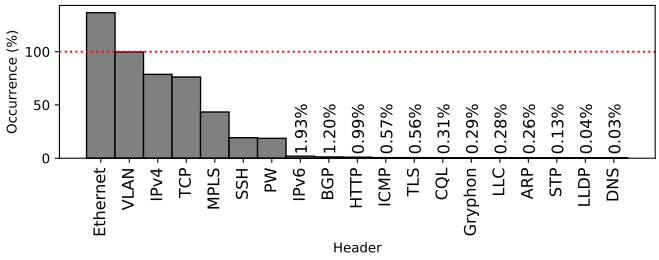

Figure 12: Occurrence of protocol headers in FABRIC traffic. Most traffic consists of Ethernet frames that carry IPv4 packets, that in turn carry TCP segments. Most traffic is tagged using VLAN, MPLS, or both.

more suitable for their resource—for instance, some sites have more NICs or GPUs than others.

Reading the right y-axis, we see that frames across all FABRIC sites have a maximal header prefix consisting of between 6 and 12 headers. Some of these headers are provided by FABRIC to isolate different researchers' traffic. Examples of typical encapsulations include "Ethernet / VLAN / MPLS / MPLS / PseudoWire / Ethernet / IPv4 / TCP / TLS" and "Ethernet / VLAN / MPLS / PseudoWire / Ethernet / IPv6 / SSH." The implication for researchers is that they do not fully control their utilization of interface MTUs in the network, since the FABRIC underlay uses some headers for tagging. We expect that this tagging has minimal impact on MTU usage, since FABRIC switch interfaces are configured to support jumbo frames throughout the network.

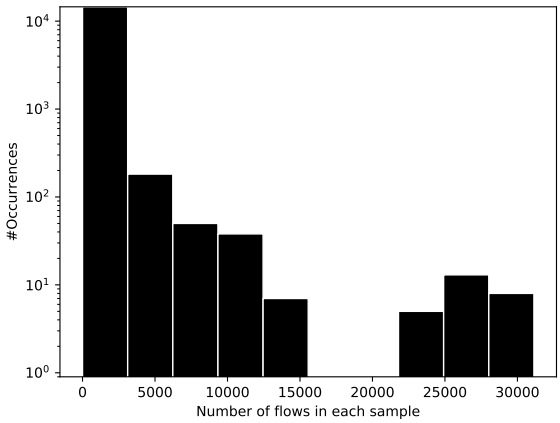

Figure 13: Frequency of encountering different numbers of flows in each 20s traffic sample.

Fig. 12 shows the occurrence of the most prevalent headers across all sampled sites. Ethernet exceeds 100% because, as we saw in the example above, Ethernet frames often carry other Ethernet frames. We observe that network traffic mostly consists of IPv4 packets, and only 1.93% of frames contained IPv6 headers. Most of the observed traffic consists of TCP streams.

Frame sizes. Across FABRIC, the most frequently-occurring frame sizes fall into the range of: 1519-2047 bytes (74.7%), 65-127 bytes (14.15%), and 128-255 bytes (5.79%). Across different sites, we observed a substantial variation in frame sizes, which is indicative of different types of network workloads. Often, minimum-size frames consist of payload-free ACKs in a TCP stream: "Ethernet / VLAN / MPLS / IPv4 / TCP." A per-site graph of frame-size distribution is presented in Appendix C.

Flow sizes. Recall from the start of Section 8.2 that each sample is 20s long. Fig. 13 shows that most samples (aggregated across all FABRIC sites) have packets belonging to fewer than 3,000 distinct flows, and a handful of samples had snippets of more than 20,000 flows. (Because of the length of the sampling, it is unlikely that all samples captured entire flows.) This suggests that the level of network activity of FABRIC can vary significantly, and that this variation can affect the results of experiments and their reproducibility. We also analyzed *across* samples to piece together flow snippets and aggregate their packets. In this aggregation, we found that most flows are short—less than 10^2B—but some flows were around 100GB in size. Vishwanath and Vahdat [41] discussed the importance of faithful background traffic when evaluating systems, but in this case the background consists

of traffic that the researchers cannot control. This suggests the importance of stronger isolation and scheduling features in shared network testbeds.

8.3 Deploying Patchwork on FABRIC

This section discusses the lessons learnt from deploying a user-developed tool to a federated testbed. This tool is deployed in two senses: **(1)** testbed users can use and extend Patchwork to profile their experiments on FABRIC, **(2)** we run Patchwork weekly to study the evolution of FABRIC's network profile to share with the community.

For the success of this research and for the uptake of Patchwork, it was important to use as few resources as possible on FABRIC. Otherwise, Patchwork would impede other experiments from starting—and thus have less to observe. This need for frugality motivated the development of iterative back-off (Section 6.2.1) and port cycling (Section 6.2.2).

Continuing on the previous point, the decoupling of capture and analysis phases in Patchwork (described in Section 6) was designed to shorten its resource leases on FABRIC. A capture lasting 12 hours can generate tens of gigabytes. Analyzing this data can take several days—most of this time is taken up by Wireshark's protocol dissectors to produce an abstracted capture (Section 6.2.4).

This work benefited greatly from FABRIC's resources and interfaces. Access control was entirely left to the testbed, and testbed resources were leveraged for this work—particularly the port mirroring primitive, switch port statistics, and FPGA NICs. Finally, system tuning enabled the studying and tuning of bottlenecks, as described in Section 8.1.3.

We noticed that FABRIC's slice allocator often struggled when handling large slices. Specifically, it took a long time to allocate slices in all-experiment mode. Parts of Patchwork's design react to and mitigate the behavior of the host testbed—for instance, Patchwork prefers smaller slices (since they are handled more quickly by the testbed's allocator) and carries out its own allocation simulations to ensure that resource requests can always be satisfied by the testbed.

Finally, FABRIC served as a very convenient distribution platform, and reduced the friction for other users to try Patchwork. In past research projects, we observed that it was difficult to reliably package research prototypes for external use, since users could encounter a wide variety of problems—for example, users might lack dependencies, lack the version of the compiler needed, or lack experience with building software. Because of FABRIC's provision of common VM images, and the culture of scripting-up FABRIC experiments for reproducibility, it was easy for FABRIC users to use Patchwork without having to compile or install anything locally.

9 Conclusion and Future Work

Federated testbeds represent huge investments in infrastructure that supports research and teaching. Currently, such testbeds only provide rudimentary primitives to capture and analyze an experiment's network traffic. This paper presented the first platform that profiles (captures and analyzes) traffic in a federated network testbed. This profiler was developed by users of the FABRIC research testbed, and it has been running on FABRIC for over a year. Using data from that deployment, this paper presented a study of the network profile and utilization of FABRIC.

As testbeds continue to evolve, network profiling might one day become a standard feature in federated testbeds, and Patchwork provides a first study of how such a profiler can be built and used in that environment.

As an initiative to benefit the research community, it would be useful to produce regular updates to the analysis of FABRIC's network profile (Section 8.2), and periodically release anonymized traffic traces to the research community.

Future Work. Directions for future research include overcoming the design limitations described in Section 6.3, and porting Patchwork to run on other testbeds. Overcoming the design limitations would involve devising a controller that scales Patchwork resources at runtime according to the resources available on FABRIC. Currently Patchwork is assigned fixed resources at start-up, and runtime scaling would be useful during long runs if Patchwork starts with small allocations. While it is clear how to scale up, it is not clear what signal Patchwork could use to scale-down—it would require a new control channel from users or FABRIC's allocator. Porting Patchwork to run on other testbeds would involve designing an abstraction layer to interface with APIs from different testbeds, in order to acquire and manage testbed resources for Patchwork.

Acknowledgements

We thank our anonymous reviewers and shepherd, and Vaneshi Ramdhony, Alexander Wolosewicz, and Cees de Laat for feedback. We thank Ilya Baldin, Paul Ruth, Komal Thareja, Mert Cevik, Xi Yang, Tom Lehman, Anita Nikolich, and Jim Griffioen for FABRIC-related help. We thank Charles Carpenter and Yongwook Song for help with MFlib, and Jonathan Sewter, Stacey Sheldon, Peter Bengough, and Yatish Kumar for help with the ESnet smart NIC framework. This work was supported in part by equipment from AMD/Xilinx, the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0106, and by the National Science Foundation (NSF) under award 2346499. 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] [n. d.]. bpftrace. https://bpftrace.org/. ([n. d.]). Accessed: 2025-05-11.

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

[3] [n. d.]. FABRIC Information Model. https://github.com/fabric-testbed/InformationModel. ([n. d.]). Last Update: 2025-01-6. Accessed: 2025-01-8.

[4] [n. d.]. FABRIC Programmable Networking . https://learn.fabric-testbed.net/article-categories/programmable-networking/. ([n. d.]). Accessed: 2024-07-13.

[5] [n. d.]. FABRIC Site: NCSA. https://portal.fabric-testbed.net/sites/NCSA. ([n. d.]). Accessed: 2025-01-8.

[6] [n. d.]. Homepage of SC24: The International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing). https://sc24.supercomputing.org/. ([n. d.]). Accessed: 2025-01-8.

[7] [n. d.]. These are the numbers every computer engineer should know. https://www.freecodecamp.org/news/must-know-numbers-for-every-computer-engineer/. ([n. d.]). Accessed: 2025-01-8.

[8] [n. d.]. Wireshark. https://www.wireshark.org/. ([n. d.]). Accessed: 2025-01-8.

[9] [n. d.]. Zero-One-Infinity Rule. http://www.catb.org/jargon/html/Z/Zero-One-Infinity-Rule.html. ([n. d.]). Accessed: 2025-05-11.

[10] 2022. NSF FABRIC project announces groundbreaking high-speed network infrastructure expansion. https://learn.fabric-testbed.net/knowledge-base/nsf-fabric-project-announces-groundbreaking\-high-speed-network-infrastructure-expansion/. (Oct. 2022). Accessed: 2024-01-26.

[11] Jay Aikat, Ilya Baldin, Mark Berman, Joe Breen, Richard Brooks, Prasad Calyam, Jeff Chase, Wallace Chase, Russ Clark, Chip Elliott, Jim Griffioen, Dijiang Huang, Julio Ibarra, Tom Lehman, Inder Monga, Abrahim Matta, Christos Papadopoulos, Mike Reiter, Dipankar Raychaudhuri, Glenn Ricart, Robert Ricci, Paul Ruth, Ivan Seskar, Jerry Sobieski, Kobus Van der Merwe, Kuang-Ching Wang, Tilman Wolf, and Michael Zink. 2018. The Future of CISE Distributed Research Infrastructure. *SIGCOMM Comput. Commun. Rev.* 48, 2 (April 2018), 46–51. https://doi.org/10.1145/3213232.3213239

[12] Martin Arlitt, Mehdi Karamollahi, and Carey Williamson. 2023. A Retrospective on Campus Network Traffic Monitoring. *SIGCOMM Comput. Commun. Rev.* 53, 2 (July 2023), 40–45. https://doi.org/10.1145/3610381.3610387

[13] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. 2019. FABRIC: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing* 23, 6 (2019), 38–47.

[14] Andy Bavier, Yvonne Coady, Tony Mack, Chris Matthews, Joe Mambretti, Rick McGeer, Paul Mueller, Alex Snoeren, and Marco Yuen. 2012. GENICloud and transcloud. In *Proceedings of the 2012 Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit (FederatedClouds '12)*. Association for Computing Machinery, New York, NY, USA, 13–18. https://doi.org/10.1145/2378975.2378980

[15] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. Association for Computing Machinery, New York, NY, USA, 267–280. https://doi.org/10.1145/1879141.1879175

[16] Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. 2014. GENI: A federated testbed for innovative network experiments. *Computer Networks* 61 (2014), 5–23. https://doi.org/10.1016/j.bjp.2013.12.037 Special issue on Future Internet Testbeds – Part I.

[17] Mark Berman, Piet Demeester, Jae Woo Lee, Kiran Nagaraja, Michael Zink, Didier Colle, Dilip Kumar Krishnappa, Dipankar Raychaudhuri, Henning Schulzrinne, Ivan Seskar, and Sachin Sharma. 2015. Future Internets Escape the Simulator. *Commun. ACM* 58, 6 (may 2015), 78–89. https://doi.org/10.1145/2699392

[18] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (jul 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[19] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. 2003. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.* 33, 3 (jul 2003), 3–12. https://doi.org/10.1145/956993.956995

[20] Brent Chun and Amin Vahdat. 2003. Workload and failure characterization on a large-scale federated testbed. *Intel Research Berkeley Technical Report IRB-TR-03-040* (2003).

[21] Peter J Denning. 1989. *The ARPANET after Twenty Years*. Technical Report RIACS TR-89.38. Research Institute for Advanced Computer Science (RIACS). 18 pages. https://ntrs.nasa.gov/api/citations/19920002477/downloads/19920002477.pdf

[22] DPDK 2025. Data Plane Development Kit. https://www.dpdk.org/. (2025).

[23] ESnet. [n. d.]. iperf3 at 40Gbps and above. https://fasterdata.es.net/performance-testing/network-troubleshooting-tools/iperf/multi-stream-iperf3/. ([n. d.]).

[24] Fraida Fund. [n. d.]. Teaching on Testbeds. https://teaching-on-testbeds.github.io/. ([n. d.]). Accessed: 2024-07-13.

[25] James Griffioen, Zongming Fei, Hussamuddin Nasir, Xiongqi Wu, Jeremy Reed, and Charles Carpenter. 2014. Measuring experiments in GENI. *Computer Networks* 63 (2014), 17–32. https://doi.org/10.1016/j.bjp.2013.10.016 Special issue on Future Internet Testbeds – Part II.

[26] Fabien Hermenier and Robert Ricci. 2012. How to Build a Better Testbed: Lessons from a Decade of Network Experiments on Emulab. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, Thanasis Korakis, Michael Zink, and Maximilian Ott (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 287–304.

[27] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. 2008. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference (CoNEXT '08)*. Association for Computing Machinery, New York, NY, USA, Article 11, 12 pages. https://doi.org/10.1145/1544012.1544023

[28] Wonho Kim, Ajay Roopakalu, Katherine Y. Li, and Vivek S. Pai. 2011. Understanding and characterizing PlanetLab resource usage for federated network testbeds. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC '11)*. Association for Computing Machinery, New York, NY, USA, 515–532. https://doi.org/10.1145/2068816.2068864

[29] Nikola Knežević, Simon Schubert, and Dejan Kostić. 2010. Towards a cost-effective networking testbed. *SIGOPS Oper. Syst. Rev.* 43, 4 (jan 2010), 66–71. https://doi.org/10.1145/1713254.1713269

[30] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. 2009. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement (IMC '09)*. Association for Computing Machinery, New York, NY, USA, 90–102. https://doi.org/10.1145/1644893.1644904

[31] Deep Medhi and Peter A. Freeman. 2009. Research challenges in future networks: a report from US-Japan workshop on future networks. *SIGCOMM Comput. Commun. Rev.* 39, 3 (jun 2009), 35–39. https://doi.org/10.1145/1568613.1568621

[32] Jeffrey C. Mogul, Drago Goricanec, Martin Pool, Anees Shaikh, Douglas Turk, Bikash Koley, and Xiaoxue Zhao. 2020. Experiences with Modeling Network Topologies at Multiple Levels of Abstraction. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 403–418. https://www.usenix.org/conference/nsdi20/presentation/mogul

[33] Meisam Mohammady, Lingyu Wang, Yuan Hong, Habib Louafi, Makan Pourzandi, and Mourad Debbabi. 2018. Preserving Both Privacy and Utility in Network Trace Anonymization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. Association for Computing Machinery, New York, NY, USA, 459–474. https://doi.org/10.1145/3243734.3243809

[34] Martin Ohlin and Martin Ansbjerg Kjær. 2007. Nice resource reservations in Linux. In *Second IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID07)*.

[35] Aravindh Raman, Matteo Varvello, Hyunseok Chang, Nishanth Sastry, and Yasir Zaki. 2023. Dissecting the Performance of Satellite Network Operators. *Proc. ACM Netw.* 1, CoNEXT3, Article 15 (Nov. 2023), 25 pages. https://doi.org/10.1145/3629137

[36] Robert Ricci, Jonathon Duerig, Leigh Stoller, Gary Wong, Srikanth Chikkulapelly, and Woojin Seok. 2012. Designing a Federated Testbed as a Distributed System. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, Thanasis Korakis, Michael Zink, and Maximilian Ott (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 321–337.

[37] Robert Ricci, Eric Eide, and CloudLab Team. 2014. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *; login:: the magazine of USENIX & SAGE* 39, 6 (2014), 36–38.

[38] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 123–137. https://doi.org/10.1145/2829988.2787472

[39] Jürgen Schönwälder, Timur Friedman, and Aiko Pras. 2017. Using Networks to Teach About Networks (Report on Dagstuhl Seminar #17112). *SIGCOMM Comput. Commun. Rev.* 47, 3 (sep 2017), 40–44. https://doi.org/10.1145/3138808.3138814

[40] Nishanth Shyamkumar, Sean Cummings, Hyunsuk Bang, and Nik Sultana. 2024. Towards Testbed-Wide Traffic Profiling for FABRIC. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops, Vancouver, BC, Canada, May 20, 2024*. IEEE, 1–6. https://doi.org/10.1109/INFOCOMWKSHPS61880.2024.10620844

[41] Kashi Venkatesh Vishwanath and Amin Vahdat. 2008. Evaluating Distributed Systems: Does Background Traffic Matter?. In *USENIX 2008 Annual Technical Conference (ATC'08)*. USENIX Association, USA, 227–240.

[42] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2002. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*. USENIX Association, Boston, MA, 255–270.

[43] Ping Yi and Zongming Fei. 2014. Characterizing the GENI Networks. In *2014 Third GENI Research and Educational Experiment Workshop*. 53–56. https://doi.org/10.1109/GREE.2014.8

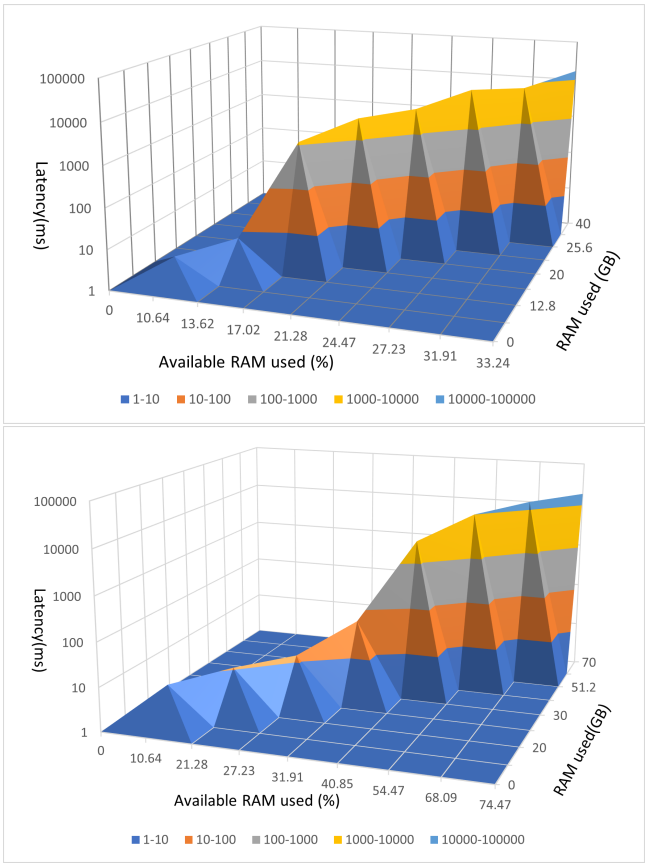

Figure 14: Summed latency observed during pcap storage for accelerator- and bypass-assisted Patchwork. The x-axis shows the percentage of free cache memory used by Patchwork's custom DPDK application; y-axis shows this percentage as RAM size; and z-axis shows the summed latencies for the `sys_writev()` calls. The top figure uses 10:20 threshold, while the bottom figure uses 20:50 threshold. These figures are explained in Appendix B.

A Ethics

Creating FABRIC's traffic profile involved sampling the traffic of other FABRIC users, and this requires a special permission on FABRIC. We obtained this permission from FABRIC leadership after discussing our research goals and data-gathering, storage, and access processes. We also submitted an IRB request to our institution and received an exemption.

B Evaluating the storage bottleneck

Building on the description in Section 8.1.3, Fig. 14 shows the result of measuring the summed latency of the `sys_writev()` calls made by the process that writes the frames to pcap files. For these graphs, the thresholds for the two parameters (`vm.dirty_background_ratio`:`vm.dirty_ratio`) were set to (10:20) and (20:50). We used the bpftrace tool [1] to profile the latency of the `sys_writev()` calls. These calls are made once for every batch of 128 frames. We used a BPF hook at the `sys_enter_writev()` to record the process making the call and measures the current time in nanoseconds. Another BPF hook at `sys_exit_writev()` invoked by the same process records the current time in nanoseconds and subtracts the earlier recorded time. This difference is the latency. We create a log-scaled histogram of these recorded values. For calculating the total latency in our graphs, we do not include the average case, and instead we focus more on the cases where the latency is higher. This is because these cases have a much greater influence in slowing down performance and causing frames to be dropped. Thus, if the latency of a `sys_writev()` falls in the [32K, 64K] ns range, then we use 64Kns (64us) as the latency value in our calculation.

This was evaluated by using DPDK Pktgen to transmit frames at 100 Gbps and capture them using Patchwork. The Alveo FPGA NIC runs a bitstream that was built using the ESnet smart NIC framework, and the receiving FABRIC node runs the DPDK pcap writer program. The host system has a single NUMA node, 16 cores, and 128GB of RAM.

The graphs show a steep increase in latency after threshold `vm.dirty_background_ratio` is exceeded. Surprisingly, this increase happened before exceeding `vm.dirty_ratio`. Looking into the kernel code, we found that at the midpoint of `vm.dirty_background_ratio` and `vm.dirty_ratio`, the writing process is throttled by the operating system. This code confirms what the graphs are showing. The graphs also show the importance of tuning the thresholds. When using 21% available RAM in the 10:20 threshold case, the summed latency is 3283ms; while for the same RAM usage in the 20:50 threshold case, it is 13ms, which is two orders of magnitude lower. To get an idea of the writing bottleneck, for a sustained 100Gbps network traffic throughput, this can be represented as 8.5GBps of traffic. For a 128GB RAM, the free cache memory by default will be around 100GB of RAM. With a 60:80 threshold, the bottleneck will happen at about 70% of free cache memory, i.e, at 70GB usage of free cache memory. This implies in about 8-9 seconds we will hit a page cache bottleneck, thus stalling the pcap writer application. Optimizing the capture of high-throughput traffic needs to work around this bottleneck, and generalizing this approach is a topic for future research.

C Frame size distribution

This section expands on the analysis of frame sizes in Section 8.2. Fig. 15 shows significant variety in frame sizes across FABRIC sites, which is indicative of different types of workloads. Most sites carry a proportion of smaller packets (e.g., S11 and S12) and several sites are notable for carrying jumbo frames (e.g., S3 and S7).

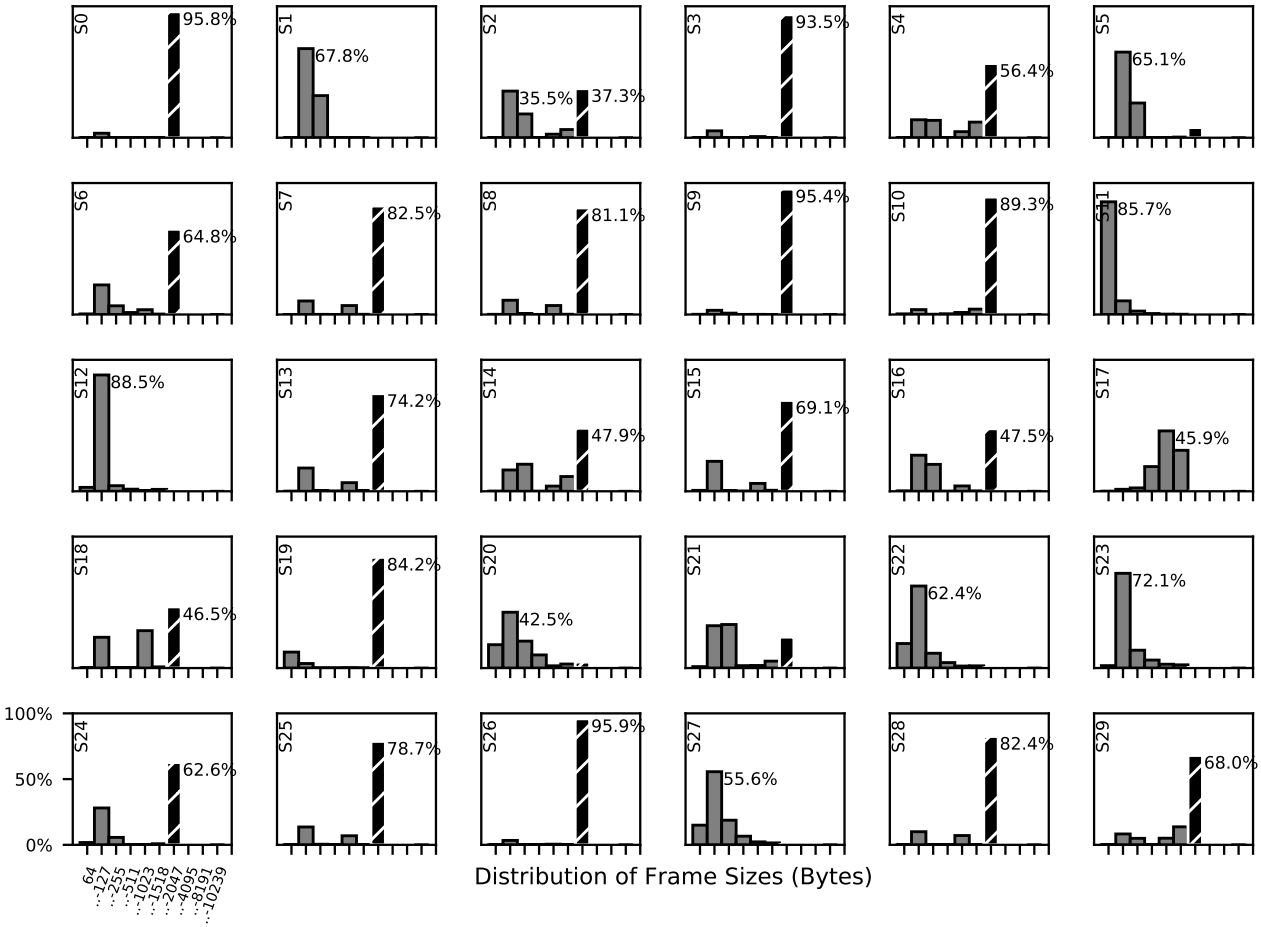

Figure 15: Distribution of frame sizes at different FABRIC sites. In this graph, site names are pseudonymized as S0-S29. Striped columns represent the portion of a site's frames that were jumbo size.