

# Marionette Measurement: Measurement Support under the PacketLab Model

Tzu-Bin Yan<sup>1</sup>(✉)[0009-0002-0126-0026], Zesen Zhang<sup>2</sup>[0000-0002-8996-162X],  
Bradley Huffaker<sup>2,3</sup>[0009-0009-3468-1266], Ricky Mok<sup>2,3</sup>[0000-0003-3300-9514],  
kc claffy<sup>2,3</sup>[0000-0003-4824-3493], and Kirill Levchenko<sup>1</sup>[0000-0003-4527-9749]

<sup>1</sup> UIUC, Urbana, IL 61801, USA {tbyan2,klevchen}@illinois.edu

<sup>2</sup> UC San Diego, La Jolla, CA 92093, USA zez003@eng.ucsd.edu

<sup>3</sup> CAIDA, La Jolla, CA 92093, USA {bradley,cskpmok,kc}@caida.org

**Abstract.** The PacketLab Internet measurement framework is designed to facilitate vantage point (VP) sharing for active Internet measurements. The core idea behind PacketLab is to have experimenters instruct remote VPs to perform a series of monitored low-level network operations to conduct measurements, which would reduce costs and security concerns of VP sharing. Despite these benefits, PacketLab users have to update their existing tools to adapt to the new *measurement model*, where available VP capabilities and the method of access differ from traditional models such as shell access to VPs. This change in the measurement model introduces limitations in measurement feasibility that merit deeper analysis. We undertook this analysis, based on a survey of recent Internet measurement studies, followed by a result accuracy evaluation of PacketLab implementations of selected representative measurements. Our results showed the PacketLab measurement model allows the implementation of a major portion (40 out of 54 studies, 74%) of distributed active measurements in relevant studies in our survey. Further evaluation also showed that the PacketLab model not only accurately supports a diverse set of measurements ranging from latency, throughput, network path, to other non-timing data categories, but also measurement requiring precise spatial and temporal coordination. To assist with porting non-timing data measurements to PacketLab, we also introduce a new porting tool, `pktwrap`, which allows existing measurement executables to communicate over PacketLab without modification.

**Keywords:** Internet Measurement · Vantage Point Sharing · PacketLab

## 1 Introduction

A large class of Internet measurements depends on having multiple vantage points (VPs) in the network. In some cases, the location of VPs is an essential element of the study (e.g. censorship), while others only require a geographically diverse set of locations (e.g. anycast). To gain access to desired VPs, researchers

have deployed hardware—ranging from dongles to rack servers—at locations of interest to their study. The high cost of establishing and operating such infrastructure suggests amortizing this cost by *sharing VP access* across experiments and research groups.

Despite benefits, VP sharing is not without challenges. In the past, VP sharing has faced several key issues: compatibility, incentives, and trust [26]. The varying and often incompatible architectures of VPs have required experimenters to repeatedly port measurements to access appropriate VPs. Additionally, many sharing approaches necessitate VP operator assistance, such as deploying and vetting new measurements, which discourages operators from supporting more measurements due to limited resources. Lastly, VP operators who allow general use of their VPs (e.g. shell access) need to trust experimenters to behave responsibly, leading to access being granted only to a select group of trusted personnel. These challenges have thus hindered widespread, high measurement flexibility VP sharing in practice.

In an IMC '17 short paper, Levchenko *et al.* [26] proposed PacketLab, a novel Internet measurement framework. PacketLab presents a new direction in VP design for sharing—relocating almost all measurement (control) logic away from the VP and instead providing experimenters with an interface to the VP network access. Under PacketLab, experimenters use their own machines as *experiment controllers* that communicate with *measurement endpoints*, where the endpoint functions similarly to a VPN server by forwarding traffic (**Figure 1**). This VPN-like sharing approach contrasts with existing approaches, where most measurement logic resides on VPs, keeping PacketLab endpoints lightweight and simple. Levchenko *et al.* argued that this simplicity allows VP operators to more easily contribute PacketLab VPs. Such an interface approach also enables operators to support new measurements without incurring extra costs beyond maintaining the existing static interface.

In addition to exporting a lightweight interface, PacketLab offers flexible access control mechanisms to address VP operator security concerns. These mechanisms allow operators to define endpoint usage policies as monitors and ensure compliance by monitoring controller actions. By reducing the cost of adoption, maintenance, and security for VP operators, PacketLab aims to make VP sharing more accessible, which would, in turn, lower the measurement porting costs for experimenters due to the increasing availability of PacketLab endpoints and the interface approach allowing easy porting across endpoints. Finally, to enhance measurement flexibility, PacketLab endpoints support scheduling packets for future transmission and send back timestamps for all sent and received packets. These features enable many kinds of timing-based network measurements that would otherwise be impossible to do accurately using a VPN server.

Central to the PacketLab framework is the novel *measurement model* that it introduces. By measurement model, we mean the set of VP capabilities for performing Internet measurements and the method to access the capabilities, which characterizes the way of performing Internet measurements under various setups. The *PacketLab measurement model* exports remote VP capabilities

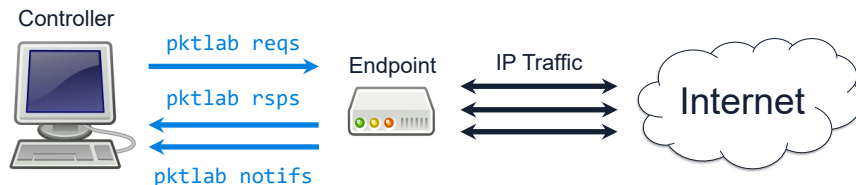


Fig. 1: Controller-endpoint interaction under the PacketLab model.

through a set of requests resembling BSD socket operations along with notifications to provide endpoint operation information across the network (Section 2.1). This model differs from the traditional *native model* where experimenters have access to VP hardware capabilities (e.g. via shell) to implement and conduct measurements directly on the VP.

In our view, the PacketLab proposal presumes what we call the *PacketLab assumption*:

- ◆ **PacketLab Assumption.** The PacketLab measurement model can support a significant fraction of—perhaps even most—distributed active measurements of interest to the Internet measurement community.

where we define *supporting a measurement under a measurement model* to be:

- ◆ **Model Measurement Support.** It is possible to develop an implementation of the measurement under the model (using the model-exported VP capabilities via the model-specified method of access) s.t. the implementation produces accurate measurement results.

Such an assumption is core to the applicability of the PacketLab framework, as despite the advantage of facilitating VP sharing, a substantially restricted measurement model supporting few measurements of interest benefits little. The assumption stands unsubstantiated in the original proposal, which is concerning as under the PacketLab model there is an inevitable delay between when the experimenter sees traffic and when the VP sees traffic, potentially preventing sufficiently timely reactions by the experimenter. In addition, the set of exported capabilities in the PacketLab model is a strict subset of those in the native model, indicating potential missing capabilities for some measurements. Without further analyses, it remains uncertain the extent to which these two limitations would affect measurement support, and in turn, whether the PacketLab assumption holds true.

The aim of this work is a first attempt to answer the research question of whether the PacketLab assumption is valid. We took an analytical and empirical approach in our investigation. We surveyed 284 measurement studies published in recent top networking and measurement conferences—SIGCOMM, IMC, PAM, CoNEXT, and TMA—to identify distributed active measurements relevant to the Internet measurement community. Our survey identified 54 stud-

ies that conducted distributed active measurements. As an initial assessment, we examined the methodology for each study to determine whether their distributed active measurements could, in principle, be *implemented* under the PacketLab model based on criteria derived from the two inherent limitations of the model. We determined that such measurements in 40 of the 54 studies (74%) could be implemented under the model.

We then implemented selected measurements under the PacketLab model using the published framework implementation [60,64]. We directly ported most measurements to PacketLab, while for the web content retrieval measurement, we developed a new general-purpose tool for porting non-timing data measurements: `pktwrap`. `pktwrap` enables existing measurement executables to communicate over PacketLab endpoints without modification. Using `pktwrap`, we were able to port with minimal effort the web content retrieval measurement by leveraging `curl`'s logic to perform HTTP and TLS operations, which demonstrates `pktwrap`'s potential to facilitate HTTP(S) and other non-timing data measurements over PacketLab.

We then performed result accuracy evaluation using the measurement implementations. As our experiments were directly conducted on the Internet, where measurement ground truth is unavailable, we define accuracy based on the results of a native implementation of the same measurement. The intuition is that because the native-implementation approach is the traditional and common way to collect measurement data, so long as the PacketLab implementation yields results close to the native implementation, the PacketLab implementation provides sufficiently accurate data.

Overall, we found that our PacketLab implementations yielded accurate results in most cases for four measurements: traceroute, TCP pipe saturation, web content retrieval, and MIDAR (Monotonic ID-based Alias Resolution [20]).<sup>4</sup> We found results from our PacketLab implementations were less than 5.8% off in mean for throughput and hop round-trip time, and had Jaccard indices no less than 0.92 for network path, HTTPS response bodies, and IP address alias pairs (MIDAR) when comparing with results given by native implementations of the same measurement. Such accuracy results combined with our survey outcome serve as strong evidence supporting the PacketLab assumption.

## 2 Background

PacketLab is an Internet measurement framework proposed to facilitate Internet measurement VP sharing. An implementation of the framework is available [60, 64]. Here, we summarize elements relevant to the current work, as well as the benefits PacketLab brings to VP sharing compared to existing approaches to

---

<sup>4</sup> We also evaluated a fifth measurement, the send gap microbenchmark, to test the model's ability to achieve arbitrary packet spacing, though we omit the details here due to space constraints. In summary, our PacketLab implementation matched the performance of the native implementation down to 7.3  $\mu$ s packet spacing. We invite interested readers to request further details from the authors.

motivate this study. We refer readers to the original proposal and project website [26, 60] for further details.

## 2.1 The PacketLab Framework and Model

The core idea behind PacketLab is to move almost all measurement logic for a distributed active measurement from the endpoint that sends packets to a separate host operated by the experimenter, turning the endpoint into something like an enhanced VPN server. To perform a measurement in PacketLab, an experiment controller that implements the measurement logic sends a measurement endpoint requests to open a socket, send packets, and so on, where the endpoint serves the requests (Figure 1). The set of requests mostly mimics the BSD socket API, with one notable difference: the send request (`nsend`) takes in a schedule-send time parameter where the controller can request the send to happen at some later specific time. An endpoint may also send asynchronous notifications to the controller.<sup>5</sup> These are used to deliver received network data as well as timestamps for network data receive time and queued data send time to the controller. These asynchronous notifications can be buffered at the endpoint up to available memory, configurable via separate requests. **Table 1** lists the core set of network requests and notifications defined in the PacketLab framework. These requests and available information propagated from the endpoint back to the controller characterize the VP capabilities available to perform measurements, which combined with their over-the-network method of access defines the PacketLab measurement model.

## 2.2 PacketLab Advantages over Other Sharing Approaches

A main advantage of PacketLab over existing VP sharing approaches, as highlighted in the original proposal, is its low adoption cost on existing VPs, achieved by maintaining a simple and lightweight endpoint design.

Another main advantage PacketLab offers over existing approaches is its ability to reduce VP operator costs during operation, particularly in supporting new measurements at low cost and preventing abuse, while at the same time maintaining some level of measurement flexibility for experimenters. A key way PacketLab achieves this cost reduction is by minimizing operator involvement in the sharing process. Many existing measurement platforms, such as CAIDA Ark [3], M-Lab [30], FCC MBA [15], BISmark [59], and Dasu [52], require VP operator to vet and/or deploy any custom measurements, which limits the support for additional measurements due to budget constraints. In contrast, supporting new measurements within PacketLab does not require operator involvement, as experimenters rely on the same VP interface for any new measurement, with existing endpoint usage policies still enforced.

---

<sup>5</sup> Asynchronous notification is a modification to the framework, which supersedes `npoll` requests in the original proposal.

Table 1: Excerpt of defined controller-endpoint messages in the PacketLab framework

---

<b>nopen</b> (sktid,prfram,proto,rbufsz,locaddr,...)	Open a socket on the endpoint with the given protocol family (IPv4 or IPv6) and protocol (UDP, TCP, or raw IP).
<b>nsend</b> (sktid,prfram,proto,sndtime,tag,...,data)	Schedule data to be sent at some specified time (0 sends immediately). Result (including actual endpoint send time) is returned in <b>ntag</b> notifications containing the same <b>tag</b> identifier.
<b>ncap</b> (sktid,prfam,proto,endtime,recvfilter)	Capture packets from a raw IP socket until some specified time. The <b>recvfilter</b> parameter is a BPF program that defines which packets to receive.
<b>nclose</b> (sktid)	Close a socket on the endpoint.
<b>ndata</b> (sktid,prfram,proto,rcvtime,...,data)	Asynchronous notification sent by endpoint (to controller) with received data and timestamp.
<b>ntag</b> (sktid,tag,time,...)	Asynchronous notification sent by endpoint with send timestamp (see <b>nsend</b> ).

---

Alternatively, measurement platforms like the PlanetLab testbed [55] and EdgeNet [5], which provided shell access to VPs, also reduced VP operator involvement by granting experimenters significant freedom. However, these platforms rely heavily on trust that experimenters will comply with the acceptable use policies [13] or simply behave responsibly. Scriptroute [56] similarly minimized operator involvement by allowing unvetted measurement script execution on VPs. They also supported limited VP security policy specifications, such as rate limiting and packet field checks, to address security concerns. In comparison, PacketLab enables operators to actively prevent abuse by supporting endpoint monitors that could enforce more complex, stateful endpoint usage policies.

Another common approach to reducing VP operator costs is to limit VPs to a static set of vetted, rate-limited measurements, as seen in platforms like RIPE Atlas [49], perfSONAR [16], and others [22, 39]. This approach minimizes both operator involvement and security concerns, and has led to widespread adoption in some cases [42, 50]. Nevertheless, it comes at the cost of low measurement flexibility. PacketLab offers greater flexibility by allowing experimenters to create arbitrary measurements using PacketLab requests. However, the extent of PacketLab’s capability to support measurements relevant to the Internet measurement community has not been explored in prior work, which we address in this study.

### 3 Methodology

The PacketLab assumption is that the PacketLab model can effectively support the deployment of a significant fraction of distributed active measurements. A

natural way to validate this assumption would be to implement such measurements and evaluate the implementations’ ability to yield accurate measurement results in experiments. Accordingly, our method has two phases: (1) a survey to identify prevalent measurements and (2) an implementation evaluation of selected measurements either commonly used in the surveyed studies or of special characteristics.

### 3.1 Internet Measurement Literature Survey

Our first phase requires identifying the set of measurements to use to test the PacketLab assumption. The difficulty we face is the sheer number and diversity of distributed active measurements performed throughout the years by the Internet measurement community; there are simply too many measurements for us to evaluate given limited resources and time. We addressed this problem by focusing on measurements that are more relevant to the community, which we argue are measurements more frequently performed in recent measurement studies. Accordingly, we surveyed recent editions of top networking and measurement conferences—SIGCOMM, IMC, PAM, CoNEXT, and TMA. We employed a twofold strategy to enhance the diversity of measurement studies: breadth, by covering the most recent editions of the top conferences to capture diversity across venues, and depth, by examining editions of SIGCOMM and IMC since 2020 to identify long-standing measurement trends over time.

During our survey, we first manually identified studies that conducted one or more distributed active measurement experiments, which we define as:

- ◆ A data collection process involving orchestration of one or more active measurements at one or more VPs, and where the collected data are VP location sensitive.

These studies would benefit the most from easier access to distributed VPs, which is the primary motivation for PacketLab. We determined data *location sensitivity* by applying one of the following two criteria:

- ◆ The study stated, explicitly or implicitly (by using multiple distributed VPs), that the data collected differ or could differ based on VP locations.
- ◆ The experiment contained an Internet-wide HTTP/HTTPS/SSH service scan, which Wan *et al.* [63] found varies based on VP locations.

For studies that performed distributed active measurement experiments, we further manually examined the measurements in the experiments as an initial assessment to determine if they are *implementable* under PacketLab by requiring both criteria:

- ◆ **IMPL1.** The measurement does not require timely reactions (less than one controller-endpoint RTT reaction time) to endpoint events (e.g., data reception).
- ◆ **IMPL2.** The measurement does not require VP capabilities not exported by the PacketLab model (Section 2.1).

We then categorized the measurements in the experiments based on measured metric type and selected representative measurements covering all major

categories for evaluation: *traceroute* (measuring hop RTT and network path), *TCP pipe saturation* (measuring TCP throughput), and *web content retrieval* (measuring HTTPS response). We also included another measurement based on the special characteristic of requiring VP coordination: *MIDAR* [20] (measuring IP address alias pairs). **Section 4** summarizes our survey results regarding the prevalence of distributed active measurements, study measurement implementability, observed major measurement categories, and representative measurement selection.

**Cross-validation.** Our study labeling process was initially performed by the first author only. To reduce human bias, we had a separate author independently verify these results using the same criteria. The two authors agreed on the 54 distributed active measurement experiment studies but had four disagreements on implementability labeling. To be conservative, our implementability result in Section 4 is an intersection of results by both authors, where we only report a study as implementable when both authors reported it implementable.

### 3.2 Result Accuracy Evaluation

After selecting the representative measurements, the next step was to evaluate result accuracy of the chosen measurements. To ensure a realistic testing environment, we performed our experiments directly over the Internet using globally distributed VPs. Such a setup meant that we could not obtain the underlying measurement ground truth. We overcame this issue by performing *native implementation result comparison*: we compared the results of a native implementation and a PacketLab implementation of the same measurement. A close match between the two would suggest that the PacketLab implementation produces results *congruent* with its native counterpart, which is the traditional and common way to collect measurement data. We used the following *congruence criteria* to determine if the implementation results were sufficiently close.

**Numerical Results.** For traceroute hop RTT and TCP throughput, we devised two congruence criteria. Denoting the sample mean of the PacketLab-implementation data to be  $\bar{x}_{PL}$ , and the sample mean of the native-implementation data to be  $\bar{x}_{NT}$ , our two criteria are as follows:

- ◆ **C1.** Over all samples,  $|1 - \frac{\bar{x}_{PL}}{\bar{x}_{NT}}| \leq c$  where  $c \geq 0$  is some small constant specifying the allowed error margin between implementations based on criteria in past works.
- ◆ **C2.** Over all samples, we cannot find statistical significance for a two-sample Kolmogorov-Smirnov (KS) test with significance level  $\alpha = 0.05$  between the PacketLab and native results.

Our C1 criterion captures the common approach in measurement studies that tolerates result deviation below a small threshold across methods for practical purposes. For latency measurements, Chhabra *et al.* [8] validated their DoH latency measurement method by comparing the median of their VPN-based

method results with results collected by directly performing DoH resolution at the VP. They reported that their method result medians were off by no more than 10 ms for all test cases, based on which they claimed high consistency and, thus, validity. Translating their reported absolute errors to relative errors, in the worst case their medians were off by approximately 10.6% from direct measurements. We therefore used 10% as the  $c$  for latency measurements. For throughput measurements, MacMillan *et al.* [28] showed that major speedtest implementations—Ookla [38] and M-Lab NDT7 [31]—gave increasingly deviating throughput results when increasing client-server RTT. Such findings combined with the accepted approach of combining throughput results for different speedtest implementations [33] suggest that the community is willing to tolerate the degree of throughput deviation caused by varying speedtest implementations. Accordingly, we used the reported relative error—6%<sup>6</sup>—by MacMillan *et al.* for the 100 ms RTT test case as the  $c$  for throughput measurements. We chose to use the reported relative error for the 100 ms RTT case as throughput results in Mok *et al.* [33] commonly had corresponding RTTs around 100 ms.

Our C2 criterion checks the similarity between two result distributions by checking if we could find any distinctive difference under one hundred samples. This criterion captures an arguably stronger idea of result congruence, requiring the whole distribution to appear close. We highlight that by no means do we intend to claim that the two distributions are the same if they satisfy C2, as the lack of statistical significance may be simply due to insufficient sample size.

**Non-numerical Results.** For traceroute hop addresses, HTTPS responses, and MIDAR alias pairs collected in our evaluation experiment, which are non-numerical measurement data, our congruence criterion is based on the well-known set similarity metric—Jaccard index<sup>7</sup>:

- ◆ **C3.** Over all samples,  $j \geq 0.9$ , where  $j$  is the Jaccard index for the two measurement result sets by the two implementations.

Ideally, we would select a Jaccard index threshold based on relevant criteria in prior work similar to C1, but we were unable to find such information. Therefore, we chose to use 0.9 as the threshold, as we believe a  $\geq 0.9$  Jaccard index is high enough for the two result sets to be considered congruent. To illustrate the strictness of C3, consider two traceroute results from the same source to the same destination. Suppose both identified 15 network hops, with the hop addresses only differing for one intermediate hop. Despite this minor difference, the Jaccard index for these two results would be only 0.875<sup>8</sup>, already violating the C3 criterion.

**Section 5** summarizes our experiment outcome.

<sup>6</sup> From MacMillan *et al.* Figure 2b.

<sup>7</sup> Also known as intersection over union or IoU.

<sup>8</sup> 14 shared hops out of 16 distinct observed hops, computed using the same approach in Section 5.3.

## 4 Survey of Measurement Studies

We surveyed measurement studies presented at top networking and measurement conferences, including SIGCOMM (2020-2024), IMC (2020-2023), PAM (2024), CoNEXT (2023), and TMA (2024).<sup>9</sup> Due to space restrictions, we summarize our findings in the following text and omit the full list of identified distributed active measurement experiment studies and per-study labeling results. We invite interested readers to request the full list and labeling results from the authors. Starting with 284 papers, we identified 54 papers, about 1 in 5, that performed one or more distributed active measurement experiments. Further examining the 54 studies, we found 14 (26%) contained experiments that could not be implemented under PacketLab, mainly due to their need for non-exported VP computation capabilities (violating criterion IMPL2). In particular, many studies measured client-side computation performance such as browser page load time [19, 21, 23] and video conferencing performance [7, 62]. Some studies also performed measurements requiring other non-exported capabilities such as access to physical layer information (mobile signal strength [35]) and OS congestion control algorithm modification [19]. A few studies were unimplementable because the measurements required timely endpoint event reaction (violating criterion IMPL1) for QUIC congestion control [21] and SSH honeypot interactive behavior [34].

For the 54 identified distributed active measurement experiment papers, we further categorized the measurements into four main categories: *latency*, *throughput*, *network path*, and *other non-timing data*.

**Latency Measurement.** Latency measurements capture time intervals between events, including query latency and network hop round-trip time. 19 of the 54 (35%) studies we surveyed included latency measurement, e.g., DNS query latency [8, 21], hop latency [11, 12, 19, 24, 62] and ping latency [7, 11, 12, 24, 32, 51, 54]. Studies of web performance [19, 21, 23, 32, 61] frequently used browser native mechanisms or browser extensions to collect latency metrics such as page transit time, page load time, speed index, and first contentful print. Such browser-originated metrics usually also factor in VP local computational power due to the inclusion of delays in browser parsing and rendering web content.

**Throughput Measurement.** Throughput measurements capture the amount of data transferred in a unit of time. Nine of the 54 (17%) papers estimated throughput from VPs using two main methods: TCP pipe saturation [19, 32, 33, 35, 62] and packet trace collection and inference [7]. TCP pipe saturation measurements were either performed using the HTTP protocol against public speedtest services such as LibreSpeed, Ookla Speedtest, Netflix Fast.com, M-Lab NDT, and Comcast Xfinity Speed Test [19, 32, 33, 62] or via iPerf [19, 35].

---

<sup>9</sup> For SIGCOMM and CoNEXT, we only looked at the 21+7 works in the measurement/telemetry sessions for relevance.

**Network Path Measurement.** Twelve of the 54 (22%) studies measured the network path taken by traffic, all using traceroute. Five of them also used the hop RTT of the traceroute output; as such, we also consider them to include latency measurement.

**Other Non-timing Data Measurement.** Most (38 of the 54, 70%) of the studies collected and analyzed non-path non-timing data retrieved over the network, such as DNS resource records [18, 21, 43, 44] and web content [14, 23, 27, 45, 63].

Based on the observed measurement prevalence, we chose the *traceroute*, *TCP pipe saturation*, and *web content retrieval* measurements for further evaluation. Other than being popular methods used in a major portion of studies (traceroute: 12/54, 22%; TCP pipe saturation: 6/54, 11%; web content retrieval: 11/54, 20%), these three measurements cover all four of our measurement categories. Regarding measurement with special characteristics, we picked another extra measurement for evaluation: *MIDAR* [20], which requires VP coordination to spread probing load for improving efficiency and to perform synchronized VP actions for accurate data collection.

## 5 Implementation Evaluation

We performed implementation evaluation on the selected measurements. As described in Section 3.2, our analysis adopted criteria C1~3 depending on the data characteristics. This section we describe our evaluation setup, including PacketLab and native implementation details, and result comparison findings.

### 5.1 Vantage Points

We deployed 6 VPs for our experiments, aiming to maximize geographic and network diversity. One VP was a local residential fiber connection in the same metro area as UIUC. The provider advertised the connection as 250 Mbps in both directions. The hardware machine was an Intel NUC7i7BNH, 2 cores (4 threads) at 3.5 GHz, 32 GB RAM, with a 1 Gb Ethernet interface, running Ubuntu 22.04 with Linux kernel 5.15.0. Five VPs were `t2-medium` AWS VM instances (2 vCPUs, 4 GB RAM, network performance class “Low to Moderate”) running Ubuntu 22.04 with Linux kernel 6.2.0, located in the following data centers: `us-west-2` (Oregon), `ap-northeast-1` (Tokyo), `ca-central-1` (Montréal), `eu-central-1` (Frankfurt), and `sa-east-1` (São Paulo). We deployed the PacketLab endpoint daemon in the published framework implementation on these VPs to run our PacketLab implementations of the measurements. For native implementations, we ran them as a native OS process directly on the VPs.

## 5.2 TCP Pipe Saturation

Our first experiment tested model support for the throughput measurement, TCP pipe saturation, used by 6 of the 54 identified studies in our survey.

**Overview.** We conducted a TCP throughput measurement experiment between each of the 6 VPs and a test server located at UIUC. We transferred a 25 MiB payload of random bytes over a TCP connection from the VP to the test server (upload) and from the test server to the VP (download). We ignored the first 1 MiB of data to mitigate TCP slow start effects and measured the time to transfer the remaining 24 MiB. We implemented the measurement natively using the VP OS interface and in PacketLab using the published framework implementation. We ran the native and PacketLab implementations back-to-back in alternating order 100 times each, which we repeated for all VPs. This process yielded four sets of 100 TCP throughput results per VP (upload/PacketLab, upload/native, download/PacketLab, and download/native). We applied the C1~2 criteria to evaluate the congruence between the two upload sets and the two download sets.

**Native Implementation.** Our native implementation consists of two programs: a client and a server, both written in C. The client program runs on the VPs and opens a TCP connection to the server. The server program runs on the test server and sends or receives 25 MiB random bytes as requested by the client, and reports upload statistics to the client. The receiver records the first timestamp  $t_0$  and byte count  $b$  after receiving 1 MiB, and the second timestamp  $t_1$  once it receives all 25 MiB. The throughput is then calculated, in bits per second, as  $8 \times (25 \times 2^{20} - b)/(t_1 - t_0)$ .

**PacketLab Implementation.** Our PacketLab implementation works with the native implementation server program by performing actions similar to the native implementation client program through the PacketLab endpoint. For the download measurement, the experiment controller disables `ndata` events (causing the endpoint to buffer all received data; see Table 1 for event description), schedules the endpoint to send a request to the server to request 25 MiB of data, and waits 10 seconds for the download to finish. The 10-second delay avoids link contention between download traffic to the VP and `ndata` messages from the VP to the controller. After waiting 10 seconds, the controller re-enables `ndata` events and receives the data from the endpoint. The received `ndata` events contain the data the endpoint received and a timestamp taken by the endpoint when `recv()` returned, signifying the receive time for the data. The controller uses these timestamps to calculate the download throughput after the first 1 MiB of data, in the same way as the native implementation.

To measure the upload throughput, the controller disables `ntag` (Table 1) events to avoid link contention between `ntag` messages and upload traffic. It then queues 25 MiB of data to be sent 10 seconds from the current time using

Table 2: Throughput and RTT measured using native and PacketLab implementations of the TCP pipe saturation (throughput) and traceroute (RTT) measurements. The *VP* column shows the vantage point/endpoint. The remainder of the table shows three columns: download throughput (*DL Tput*), upload throughput (*UL Tput*), and *RTT*. The *NT* subcolumn shows the sample mean measured using the native implementation. The *PL err.* subcolumn shows the difference between the PacketLab and the native sample mean in absolute and relative terms. The *C1* subcolumn shows the C1 criterion (relative difference between the PacketLab and native sample mean  $\leq 6\%$  for throughput and  $\leq 10\%$  for RTT) satisfaction for the case, where a  $\bullet$  indicates satisfaction and a  $\circ$  otherwise. For all measurements  $n = 100$ . **We found that the throughput and RTT results given by both implementations of the TCP pipe saturation and traceroute measurements are similar for all VP cases, satisfying congruence criterion C1.**

<i>VP</i>	<i>DL Tput (Mibps)</i>			<i>UL Tput (Mibps)</i>			<i>RTT (ms)</i>		
	<i>NT</i>	<i>PL err.</i>	<i>C1</i>	<i>NT</i>	<i>PL err.</i>	<i>C1</i>	<i>NT</i>	<i>PL err.</i>	<i>C1</i>
Local res. fiber	203.0	+1.0 (+0.5%)	$\bullet$	237.6	-0.9 (-0.4%)	$\bullet$	17.3	-0.9 (-5.4%)	$\bullet$
AWS Frankfurt	35.3	-0.8 (-2.3%)	$\bullet$	74.1	-2.8 (-3.8%)	$\bullet$	108.1	-0.1 (-0.1%)	$\bullet$
AWS São Paulo	32.0	-0.3 (-0.9%)	$\bullet$	77.2	-4.4 (-5.7%)	$\bullet$	138.8	-0.0 (-0.0%)	$\bullet$
AWS Montréal	75.4	-1.1 (-1.4%)	$\bullet$	187.5	+0.4 (+0.2%)	$\bullet$	25.9	-0.0 (-0.1%)	$\bullet$
AWS Tokyo	31.0	-0.5 (-1.6%)	$\bullet$	68.5	-3.5 (-5.1%)	$\bullet$	167.1	-0.0 (-0.0%)	$\bullet$
AWS Oregon	64.7	+2.7 (+4.2%)	$\bullet$	109.6	+1.2 (+1.1%)	$\bullet$	59.5	-0.0 (-0.1%)	$\bullet$

*nsend* messages. Similar to *ndata*, the 10-second delay avoids link contention between the *nsend* messages and upload traffic. When the upload finishes, the test server sends the measured values  $b$ ,  $t_0$ , and  $t_1$  to the endpoint, and the endpoint forwards them to the controller for calculation.

**Throughput Results.** The throughput results given by both implementations were similar for all VPs (**Column *DL Tput* and *UL Tput* in Table 2 and Figure 2**), satisfying congruence criteria C1 and C2. For C1, the relative errors between the PacketLab and native result sample means were always no larger than 5.8%, which falls below the C1  $c$  for throughput (6%). For C2, the KS tests did not reveal significant differences between the two distributions (see Appendix A for test details), which is consistent with the similarity of the CDF curves per VP in Figure 2. Based on the satisfaction of the C1 and C2 criteria, we consider the results given by the two implementations to be congruent with each other.

**Takeaway.** With our congruence results, we consider the PacketLab model to support the TCP pipe saturation measurement, and potentially other throughput measurements as well.

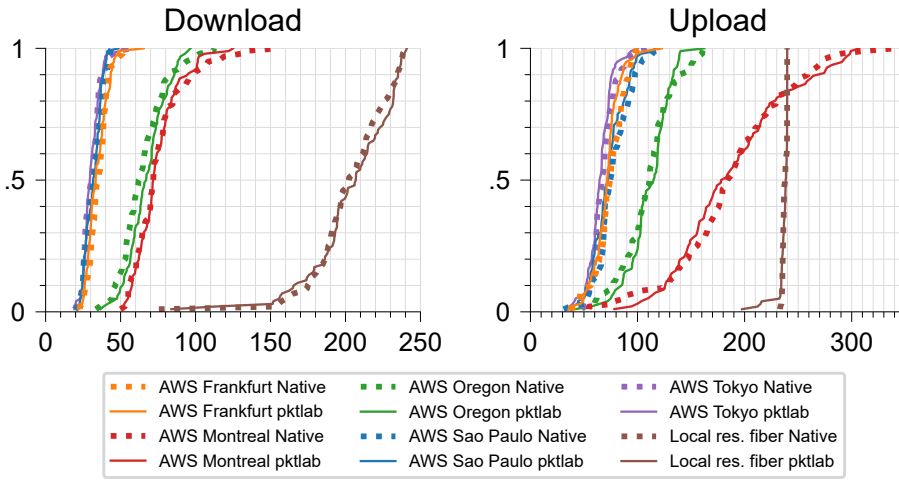


Fig. 2: Empirical CDFs of VP - test server TCP throughput (in Mibps) measured using native and PacketLab implementations of the TCP pipe saturation measurement. **We found that the throughput results given by both implementations have high distribution similarity for all VP upload and download cases, satisfying congruence criterion C2** (see Appendix A for C2 KS test details).

### 5.3 Traceroute

Our second experiment tested model support for the common hop latency and network path measurement, traceroute, used in 12 of the 54 distributed active measurement studies in our survey.

**Overview.** We conducted a traceroute measurement experiment from our 6 VPs to a test server at UIUC. We followed the well-known traceroute algorithm—sending TTL-limited UDP probes from the VP to the test server and receiving ICMP error responses for the probes, where we limited ourselves to one UDP probe per hop. We implemented the PacketLab version using the published framework implementation, while for the native implementation, we employed the popular version by Butskoy [1] for Linux. To minimize routing variation caused by packet content, we ensured that the probes of both implementations were as similar as possible, including the UDP ports and the packet payload. We could not craft the probes to be exactly the same. Notably, the kernel fills the packets’ IP ID field (and consequently IP header checksum) with random numbers based on the kernel internal state for Butskoy’s implementation, whereas in PacketLab the experiment controller decides the field. Similar to the TCP pipe saturation experiment, this experiment ran the two implementations back-to-back in alternating order for 100 times each on all VPs. The outcome of the experiment was two sets of traceroute data per VP. We compared the network

path part of the measurement data with the C3 criterion and the hop RTT part of the measurement data with the C1 and C2 criteria.

**Native Implementation.** We used Butskoy’s traceroute [1] (version 2.1.2) based on its wide availability in major Linux distributions. Butskoy’s traceroute by default sends UDP probes with their destination ports starting from 33434 (incremented per probe sent) and source port selected by the kernel at random (`bind(0)`) per probe. To maximize the similarity between implementations, we used the `--sport` option to set the source port to 20556. We reduced the per-hop probe count from the default three to one (with `-q 1`), increased the maximum hop count from 30 to 64 (with `-m 64`), and disabled reverse DNS lookup (with `-n`). We left the remaining options to their default values. For each probe, Butskoy’s traceroute obtains the send timestamp by calling `gettimeofday()` immediately before the probe `send()` call, while the kernel supplies the ICMP response receive timestamp via an accompanying `SO_TIMESTAMP` control message. The implementation then takes the difference between the send and receive timestamps as the hop RTT.

**PacketLab Implementation.** We implemented the traceroute algorithm under PacketLab by iteratively probing and receiving ICMP responses per hop. To transmit a probe, the experiment controller schedules one TTL-limited UDP probe to be sent immediately (with the `nsend` message) through an endpoint raw IP socket. We used a raw IP socket rather than a UDP socket on the endpoint for probing, as the PacketLab framework only supports TTL customization for raw IP sockets. We designed the probe to be as close as possible to Butskoy’s implementation, where for IP ID we attempted to mimic the kernel’s behavior by picking a value uniformly at random within the possible field range. To collect ICMP responses, the controller issues an `ncap` (Table 1) request with no end time to capture ICMP TTL exceeded and destination unreachable packets at the endpoint, which the endpoint then sends back to the controller via `ndata` notifications. For hop RTT, the controller gets probe send timestamps from the endpoint `ntag` notifications, which the endpoint daemon records *after* the `sendto()` call with `clock_gettime`. The ICMP response receive timestamp is available in the `ndata` message. For comparable accuracy with the native implementation, we also modified the endpoint daemon to use kernel receive timestamps (`SO_TIMESTAMPNS`) for this experiment. The controller then calculates the difference between the send and receive timestamps as the hop RTT.

**Hop RTT Results.** For our hop RTT results, we focused on the VP - test server (i.e. traceroute target) RTT as the representative set of hop RTT data for checking result accuracy, as such data are available for all VPs. Our hop RTT results are summarized in **column *RTT* in Table 2** and **Figure 3**. We found that our hop RTT results satisfied C1 of the congruence criteria, where the relative errors between the sample means were always smaller than

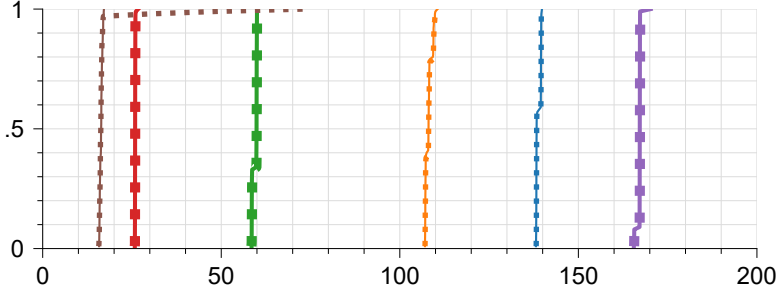


Fig. 3: Empirical CDFs of VP - test server RTT (in ms) measured using native and PacketLab implementations of the traceroute measurement. See Figure 2 for plot legend. VP cases where we obtained KS test significance for the C2 criterion are highlighted to have thicker curves (see Appendix A for C2 KS test details). **We found that the RTT results fail to achieve high distribution similarity (violating congruence criterion C2) for AWS Montréal, Tokyo, and Oregon, despite seemingly close CDFs for all VP cases. We believe this phenomenon is due to the small variance in RTT distributions for most VPs, which allows small distribution deviation to be sufficient for KS test significance.**

5.5% (recall C1  $c$  for latency measurements is 10%). Notably, our results were in most cases off by no more than 0.2% of the native sample mean. However, our hop RTT results violated C2 for AWS Montréal, Tokyo, and Oregon (see Appendix A), meaning that hop RTT distributions were quite different between the two implementations. We believe the reason for the violation was due to the rather small variance in hop RTT results (less than 1 ms standard deviation for most VPs except local residential fiber), implying that even small deviation in result distribution could be sufficient for KS test significance. The small-variance phenomenon could also be observed via the hop RTT empirical CDFs in Figure 3. We believe the root cause of the small deviation was the difference in send timestamp collection between the implementations: in Butskoy’s traceroute, the send timestamp is collected *before* the probe `send` call, while in the PacketLab endpoint implementation, the send timestamp is collected *after* the `sendto` call. Nevertheless, with our hop RTT results satisfying C1, we consider the PacketLab and native implementation results congruent for practical purposes.

**Network Path Results.** For network path results, we chose to perform an aggregation over all runs per implementation and VP due to potential route variation over time [46]. We performed our aggregation by first taking a traceroute run outcome, which consists of a list of (hop count, hop address, hop RTT) tuples, and extracting all unique (hop count, hop address) tuples within that outcome. We then combined all unique (hop count, hop address) tuples from all 100 runs of an implementation on a VP into a set. By performing the (hop count, hop address)-tuple extraction and aggregation per implementation and VP, we had at the end two (hop count, hop address)-tuple sets per VP, one for

each implementation. Our Jaccard index computations were then based on the two (hop count, hop address)-tuple sets per VP.

Overall, we found that our two implementations obtained highly similar network path results. For AWS VPs, our calculated Jaccard index was all 1, indicating an exact match of two sets. Although we did not have an exact match for the local residential fiber VP (the native implementation saw one extra unique address for hop 5, suspected due to infrequent route variations), the Jaccard index for the VP was still 0.93 ( $> 0.9$ ), satisfying C3 and indicating that the network path results from both implementations were congruent.

**Takeaway.** Based on our congruent results for both hop RTT and network path, we consider the PacketLab model to support the traceroute measurement and potentially also other latency measurements for practical purposes.

## 5.4 Web Content Retrieval

Our third experiment tested model support for the common non-timing data measurement, web content retrieval, used in 11 of the 54 distributed active measurement studies in our survey.

**Overview.** We conducted a web content retrieval measurement experiment from our Tokyo and Oregon AWS VPs as well as the local residential fiber VP. We did not use all AWS VPs due to losing access to some (funding issues). Our experiment involved querying the default webpage for the top 1K websites on the Tranco list [25], generated on August 5, 2024.<sup>10</sup> During our experiment, we used the same measurement executable to collect data directly at the VP (native implementation) and indirectly at the controller server via the PacketLab endpoint by running the executable with our new non-timing data measurement porting tool `pktwrap` (PacketLab implementation). For these 1K websites, we intentionally queried the HTTPS default port (443) instead of HTTP (80), because HTTPS querying is a more complex test case involving TLS’s multi-round interactions. This approach allows us to stress test `pktwrap` in terms of complex interaction support, which would provide more evidence on the general applicability of the `pktwrap` tool to other non-timing data measurements. The experiment produced two sets of HTTPS response data per VP. We then compared the HTTPS response bodies for each website, excluding the HTTPS response headers to avoid skewing the results with session-specific values (e.g., `Date` and cookies). In cases where one or both implementations failed to retrieve the HTTPS response, we considered it a match only if both implementations failed. Finally, we calculated the Jaccard index based on the comparison between the two data sets to apply the C3 criterion.

---

<sup>10</sup> Available at <https://tranco-list.eu/list/7X79X>.

**Native Implementation.** We used `curl` [10] (ver 8.7.1) for our native implementation. We primarily used the default options, except for the `-i` option, which instructs `curl` to include the HTTPS response headers in its output.

**PacketLab Implementation.** For our PacketLab implementation, we developed a new general-purpose tool called `pktwrap` to port non-timing data measurements. `pktwrap` enables repurposing existing measurement executables as PacketLab controllers *without requiring modification or recompilation*. It does so by intercepting executable socket function calls and bridging them through the PacketLab endpoint—a process we term as “wrapping” the measurement executable. Specifically, `pktwrap` incorporates a libc shim that intercepts function calls such as `bind`, `connect`, and `send`-variants, and translates them into PacketLab requests (e.g., `nopen` and `nsend`; see Table 1 for request description) that perform similar, if not equivalent, actions on the endpoint. In addition, as certain functions require information from PacketLab asynchronous notifications (e.g., `recv`-variants need data from `ndata` messages), `pktwrap` further incorporates a background communication manager. This manager runs on a dedicated thread to process and buffer notifications, which the libc shim then retrieves to fulfill wrapped executable operations.

We prototyped `pktwrap` on Linux using the published framework implementation, and used it to wrap `curl` to query the HTTPS web server through the VP PacketLab endpoint, using the same executable options and arguments. This approach significantly accelerated our evaluation process by reusing `curl`’s logic for TLS and HTTP.

**Web Content Results.** We computed the Jaccard index for the two data sets by first constructing a (web domain, HTTPS response body)-tuple for each response in each data set, and then aggregating the tuples for each set. In cases of query failure, we stored a special identifier as the HTTPS response body that will only match with itself. This process produced two sets of (web domain, HTTPS response body)-tuples per VP, which we used to compute the Jaccard index by checking whether the tuples matched exactly (byte-by-byte for non-failure HTTPS response bodies) between the two sets. Our initial results gave relatively low Jaccard indices: 0.84 for both Oregon and Tokyo and 0.85 for local residential fiber, which is below our C3 criterion threshold of 0.9.

Upon further investigation, we discovered that many differing websites embedded session-specific values in their response bodies, despite being otherwise identical. To prevent these websites from skewing our results, we collected another set of HTTPS responses for the 1K websites per VP using only the native implementation. We then compared this with our previous native implementation responses to identify outlier websites that gave differing bodies across native implementation queries. After removing these outlier websites (84 for Oregon, 83 for Tokyo, and 79 for local residential fiber) from our Jaccard index computation, we found significantly improved results: 0.99 for Oregon and local residential fiber, and 0.98 for Tokyo, all above the 0.9 threshold. This agreement indicates

that our results satisfy the C3 criterion, demonstrating that the web content retrieval results from both the native implementation (non-wrapped `curl`) and the PacketLab implementation (`pktwrap`-wrapped `curl`) were congruent.

**Takeaway.** Based on our congruence results, we consider the PacketLab model to support the web content retrieval measurement, and potentially also other non-timing data measurements.

When conducting the experiment on the local residential fiber VP, we also tested the PacketLab endpoint monitor mechanism, which we found could be leveraged to restrict controller access to selected website servers. Due to space limitations, we refer readers to Appendix B for further testing details.

## 5.5 MIDAR

Our final evaluated measurement was MIDAR for router alias resolution [20]. Our goal is to test the ability of PacketLab to support complex measurements where the experiment controller must coordinate actions from many global VPs.

**Overview.** MIDAR (Monotonic ID-based Alias Resolution) constructs a router-level Internet topology by inferring which IP (v4) addresses in observed traceroutes belong to the same physical router, i.e. are *IP address aliases*. The outcome of MIDAR is a set of IP address alias pairs. For the native implementation results, we directly used the alias data from CAIDA’s Macroscopic Internet Topology Data Kit [4] collected in late February 2023 (labeled 2023-03 on the website) using a native implementation of MIDAR on 55 CAIDA Ark [3] nodes. We implemented MIDAR in PacketLab and ran the implementation in May 2023 using 25 of the 55 Ark nodes along with 30 `t2-medium` AWS instances spread across all available regions as PacketLab endpoints to match the VP count in the native MIDAR run. We did not use all 55 original Ark nodes due to software incompatibilities and node instabilities. We applied the C3 criterion to evaluate congruence between the two alias pair sets.

**Vantage Point Coordination.** The MIDAR measurement uses a phased sequence of coordinated sub-measurements (estimation, discovery, elimination, and corroboration) from many VPs to many IP addresses (theoretically, all reachable ones) to infer whether interfaces belong to the same router. Specifically, the IP header ID field of responses from interfaces allows the inference of whether different addresses belong to the same router. We refer the readers to the original MIDAR study [20] for further details of the method, while we highlight here parts of the measurement that require VP coordination. With a large number of IP targets (2.6 million initially), all four of MIDAR’s phases split the probing load across all VPs for efficiency, which requires coordination among VPs on which set of targets to run per VP. In addition, the MIDAR discovery phase in particular adds an extra requirement among VPs where the probe transmission needs to be synchronized regardless of potential VP clock skew.

**PacketLab Implementation.** Each MIDAR phase has two parts: probing and analysis. In the native implementation, each Ark node probes a target list, performs initial analysis on probing results, and sends back the initial analysis outcome to a central machine for final analysis. The central machine then generates the target list for the next phase based on the final analysis outcome. For our PacketLab implementation, we ported the probing component to PacketLab using the published framework implementation and reused the analysis components of the native MIDAR implementation. The PacketLab centralized controller design made it easy to divide probing load among VPs. To synchronize the sending of probes, we implemented the NTP protocol in PacketLab to estimate clock skew between VPs and the controller server, which we then used to schedule simultaneous probe transmissions.

**Alias Pair Results.** Overall, we found a low Jaccard index (0.64) between CAIDA’s native MIDAR alias pair results (1.99M pairs) and our PacketLab MIDAR implementation results (2.03M pairs). The native MIDAR campaign discovered around 426K unique alias pairs not present in the PacketLab campaign, while the PacketLab campaign found about 465K unique alias pairs not present in the native campaign. Further examination revealed that these extreme disparities derived from slight differences in large alias sets, i.e., when one more IP appears in an alias set of a campaign but not the other, which we believe was caused by potential churn in responsive IP addresses during the two-month period along with differences in the VPs used for the two campaigns. For instance, PacketLab (but not native) MIDAR concluded that IP 38.23.192.18 belonged to the same router as 896 other addresses. This IP increased PacketLab MIDAR’s final alias pair count by 896 (recall MIDAR produces alias *pairs* as the final outcome). When we removed outlier IPs that only appeared in one campaign but not the other (a total of 108K IPs, where both campaigns gave around 1.58M pairs post-removal), the Jaccard index between native and PacketLab MIDAR increased to 0.98 ( $> 0.9$ ), satisfying the C3 criterion and suggesting that the results of both implementations are congruent.

**Takeaway.** This general agreement on a global alias resolution measurement campaign suggests the PacketLab model would successfully manage the complex MIDAR measurement process, and in turn potentially support other measurements requiring VP coordination.

## 6 Discussion

### 6.1 Model Limitations

In our survey (Section 4), we identified two classes of distributed active measurements that pose a challenge for the PacketLab model: those that include host computation (e.g., browser load times) and those that require a timely response (e.g., congestion control). Here we discuss these measurements in more detail.

**Host Computation.** Eight of the surveyed studies measured some element of end-host behavior. For example, Netravali *et al.* [23] experimentally evaluated JavaScript dead code elimination on web pages served to mobile clients. The authors measured page load time, which included both a network and browser component, requiring direct browser execution on the VP. The PacketLab model does not export such arbitrary computation capability of the endpoint, so such a measurement is not possible directly.

One alternative is to run the computation on the controller and forward traffic through the endpoint, treating the endpoint as a VPN server. The nature of the PacketLab model meant that the measured delay could then include the path from the controller to the endpoint, increasing the network overhead of the measurement. It may be possible to use the send and receive timestamps supported by the PacketLab model to compensate for the controller-endpoint portion of the network overhead. However, such trickery introduces inaccuracies in the results due to differences in computational power between the VP and the controller.

**Timely Response.** Another class of measurements where the PacketLab model stumbles are those that require a timely response. Congestion control is a notable example: the sending side of a TCP session will adjust its sending rate in response to observed real-time responses from the receiver. The PacketLab model does support native TCP sockets provided by the VP operating system, where the kernel would assist in real-time responses via the kernel network stack. Consequently, certain TCP measurements, such as throughput, are still doable via the model as demonstrated in Section 5.2. However, implementing custom congestion control support appears out of the question for the PacketLab model. This includes measuring QUIC performance [21] or evaluating different congestion control algorithms [19]. To support user-space congestion control and other measurements where measured behavior is sensitive to response timing, the endpoint needs to allow running measurement logic on the VP. The PacketLab model does not export such logic-running capability of the VP, indicating such measurements are not possible under the current model.

**Endpoint Measurement Logic Execution.** The identified limitations shed light on a framework improvement: adding native measurement logic execution support to the endpoint. This improvement would allow the framework to accommodate measurements requiring timely responsiveness that were not possible before and allow an arguably better estimation of endpoint host computation power. However, this improvement raises concerns about increasing the complexity of PacketLab endpoints, which could elevate adoption costs for endpoint operators, as well as the risk of abuse by malicious experimenters. Further research is thus needed to develop a lightweight and secure method for executing measurement logic on the endpoint.

## 6.2 Performance

When evaluating the traceroute measurement, we modified the endpoint daemon to use kernel receive timestamps (`SO_TIMESTAMPNS`) to achieve accuracy comparable to Butskoy’s traceroute. While this suggests that the published framework still has room for performance improvements, it also presents opportunities for experimenters to enhance measurement accuracy. By improving a shared endpoint implementation, PacketLab experimenters can directly benefit from these upgrades without needing to engineer native implementations incorporating mechanisms behind the improvement. For example, incorporating kernel timestamp support into the endpoint daemon implementation could allow more accurate timestamps for PacketLab measurements compared to the common `recv()` followed by `gettimeofday()` approach in naive measurement implementations. This would save other researchers the effort to learn esoteric system call parameters necessary for accurate network measurements.

## 6.3 Congruence Criteria

For implementation evaluation, we picked several result congruence criteria to represent potential experimenter standards on data similarity: sufficiently close for practical purposes based on past study criteria (C1), having high distribution similarity (C2), and subjective judgments as Internet measurement researchers (C3). Our criteria do not cover all cases, where an experimenter could have tighter or different requirements. In this manner, one could interpret our experiment results as evidence of potentially achievable result accuracy under the PacketLab model. Experimenters could use this information to decide whether or not to conduct their measurement over PacketLab, where it would be debatable if there is a stricter accuracy requirement than found in our results.

## 6.4 PacketLab Usage Insights

We summarize some of our insights when working with PacketLab during our evaluation experiments to assist future experimenters in their adoption efforts.

**Control Traffic Interference.** Our TCP pipe saturation PacketLab implementation temporarily disabled notifications during endpoint data download and upload, and scheduled endpoint upload to happen further in the future to prevent link contention between PacketLab control traffic and measurement traffic from distorting the measurement results. We performed this adjustment because we expected high measurement and control traffic for this measurement and our VPs had only a single Internet-accessible network interface, shared for both measurement and control purposes. Experimenters should be wary of similar link contention problems when designing their measurements, especially for heavy traffic ones, and employ workarounds such as suppressing endpoint notification and scheduling transmissions to avoid undesirable PacketLab control traffic interference on measurement outcome.

**Clock Desynchronization.** PacketLab notification timestamps and `nsend` schedule send times are based on the endpoint clock rather than the controller clock. For most of our evaluation, our VPs and controller server periodically synchronized with external NTP servers, keeping clock skew minimal (within a few milliseconds). This setup allowed us to schedule future sends easily using the controller clock by specifying a large schedule delay. As VPs in real-world scenarios may have significant clock skew due to lack of or delayed clock synchronization, experimenters should consider this possibility when designing measurements. This is particularly important when scheduling future sends, as an unhandled significant clock skew can result in undesirable delayed or premature transmissions. Experimenters could leverage PacketLab endpoints’ exported clock value to estimate clock skew with NTP and adjust the scheduling delay.

### 6.5 `pktwrap` Limitation

A limitation of `pktwrap` in its current form is that it does not preserve timing data fidelity, as `pktwrap` currently does not manipulate timing information observed by the executable. This limitation is similar to the distortion observed in timing data measurements performed over a VPN. Given the prevalence of timing data measurements in our survey, this limitation suggests an avenue for further research: enhancing `pktwrap` to maintain timing data fidelity. One potential approach is to implement a form of *timing information forgery* within `pktwrap`, where the forged timing information, while not accurate itself, would enable accurate estimation of measurement data by preserving network event delays as observed at the endpoint. We leave this as a direction for future work.

### 6.6 Adoption Cost Evaluation

In this work, we investigated the PacketLab model’s measurement support capability in an effort to understand the impact of PacketLab’s design choice on measurement logic relocation. We found mostly positive results. A separate issue we did not address in this work is the proposal-claimed low adoption cost of PacketLab for VPs due to its supposed simplicity and lightweightsness. Incorporation with existing measurement platforms may be more involved than simply running the reference endpoint daemon on the VP due to different platform operational models. In addition, the hardware resource requirements for running PacketLab endpoints also need further investigation to see if machines with limited hardware capabilities could function as endpoints. We leave this adoption cost evaluation as future work.

## 7 Related Work

There have been a number of attempts to organize a list of comprehensive network monitoring tools, most notably the now defunct “Internet tools taxonomy” page by CAIDA [2] and the currently active “Network Monitoring Tools” page

by Stanford SLAC [57]. The CAIDA list organized tools based on intent and was used by Scriptroute [56] to understand the need for existing measurements and demonstrate approach capability. The Stanford list contains a diverse range of (over 700) network monitoring tools organized into specific categories such as commercial/public domain, wireless, web etc., and serves as a reference for network administrators. For this study, we surveyed recent network measurement studies to understand the needs of the network measurement community and evaluated the PacketLab model in accordance with the identified needs.

Measurement platform studies often present case studies of selected measurements for evaluation [9, 22, 52, 56, 59] to demonstrate efficacy and familiarize readers with the system. The original PacketLab proposal [26] followed a similar strategy by giving examples of both bandwidth measurement and traceroute and only provided on-paper arguments on model limitations without any empirical evidence on impact. Our study extends this previous work by evaluating measurement support of the PacketLab measurement model through a survey of PacketLab-relevant distributed active measurement studies and performing result accuracy experiments on selected representative measurements.

For space reasons, we also discuss some further related work in Appendix C.

## 8 Conclusion

The goal of this work was a first attempt to validate the *PacketLab assumption* of the original PacketLab proposal, namely that the PacketLab measurement model can effectively support the deployment of a significant fraction of distributed active measurements. To validate this assumption, we surveyed recent measurement studies published at SIGCOMM, IMC, PAM, CoNEXT, and TMA that included relevant distributed active measurement experiments and assessed whether they could be implemented under the PacketLab model. We found that a majority (74%) of distributed active measurement studies could be implemented under PacketLab. Furthermore, for selected prevalent measurements and measurement of special characteristics, we found that we were able to implement measurements yielding congruent results for practical purposes under the PacketLab model using the published framework implementation, and in some cases even achieve congruent results with high distribution similarity. Based on our survey analysis and implementation evaluation, we believe that the PacketLab assumption holds despite the simplified VP design, making it a strong candidate for VP sharing within the Internet measurement community. Our new `pktwrap` measurement porting tool could also potentially assist Internet measurement experimenters in porting non-timing data measurements to the framework at minimal cost.

**Acknowledgments.** AWS results presented in this paper were obtained using CloudBank [37], which is supported by the National Science Foundation (NSF) under award #1925001. The PacketLab project was supported by NSF award #1764055/1903612/2131987 and a gift from Comcast. We would like to thank our shepherd Dr. Alessandro Finamore and anonymous reviewers for their valuable feedbacks.

This version of the contribution has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: <https://doi.org/10.1007/978-3-031-85960-1>. Use of this Accepted Version is subject to the publisher’s Accepted Manuscript terms of use <https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms>

## A Throughput and RTT KS Test Results

Table 3: Throughput and RTT KS test results between measurement results of native and PacketLab implementations of the TCP pipe saturation and traceroute measurements. Similar to Table 2, each row is divided into three columns, with each column showing the KS test results for download throughput, upload throughput, and RTT per VP. The *KS D* and *p* subcolumns show the KS test statistic and p-value, while the *C2* subcolumn shows C2 criterion satisfaction for the case. **We found that throughput result distributions were close between implementations, satisfying congruence criterion C2, while RTT result distributions already differ for AWS Montréal, Tokyo, and Oregon, violating C2.**

VP	DL Tput (Mibps)			UL Tput (Mibps)			RTT (ms)					
	KS	D	p	C2	KS	D	p	C2	KS	D	p	C2
Local res. fiber	0.090	0.815	•	•	0.130	0.368	•	•	0.160	0.155	•	•
AWS Frankfurt	0.134	0.300	•	•	0.187	0.052	•	•	0.080	0.908	•	•
AWS São Paulo	0.114	0.498	•	•	0.177	0.080	•	•	0.170	0.111	•	•
AWS Montréal	0.060	0.988	•	•	0.103	0.628	•	•	0.220	0.016	○	○
AWS Tokyo	0.106	0.578	•	•	0.144	0.229	•	•	0.470	0.000	○	○
AWS Oregon	0.163	0.122	•	•	0.119	0.437	•	•	0.270	0.001	○	○

## B Testing the PacketLab Endpoint Monitor Mechanism

When conducting the experiment on the local residential fiber VP, we also tested the PacketLab endpoint monitor mechanism to restrict access to selected website servers. This test was motivated by similar applied policies in real-world VP sharing scenarios, such as RIPE Atlas’s HTTP measurement target restriction toward RIPE Atlas anchors [47]. Given the legal and security concerns surrounding arbitrary HTTP queries [48], we expect PacketLab endpoint operators to enforce similar restrictions in practice. It is therefore valuable to investigate the feasibility of using endpoint monitors to implement these website server access policies.

The PacketLab framework provides access control mechanisms—the PacketLab public key system (PPKS) and WebAssembly-based monitors—to allow

operators to define and enforce endpoint usage policies. To enforce a policy, an operator first procures or implements a monitor program that statefully audits PacketLab message exchanges between the controller and the endpoint based on the policy. The operator then compiles the program into a monitor binary and signs a PacketLab *experiment privilege certificate*, embedding the binary’s hash, for some requesting experimenter. The certificate and monitor binary are then provided to the experimenter. At experiment time, the experimenter instructs the controller to supply the certificate and binary to the endpoint during measurement session setup, which the endpoint then verifies and applies the monitor throughout the measurement process to enforce the policy.

Our testing involved generating an experiment privilege certificate containing the hash of a custom monitor binary that only allowed TCP communication with the top 1K website IPs. The monitor enforced this policy by checking and permitting only TCP connection `nopen` requests targeting these specific IP addresses (embedded within the monitor) and allowing `nsend` requests and `ndata` notifications only for TCP communication. We used the certificate to access the local residential fiber endpoint for collecting the PacketLab implementation HTTPS response data for the top 1K websites, as well as another 100 websites ranked 1001 to 1100 to verify monitor efficacy. When collecting data, we used the `--resolve` option to force curl to query the specific web server IP addresses permitted by the monitor, which we also specify when collecting native implementation data.

In our blocking test, we found that the endpoint monitor successfully blocked attempts to access all websites ranked 1001 to 1100 while still allowing queries to the top 1K websites via the permitted addresses. This suggests that PacketLab endpoint operators can employ monitors to implement access control policies for regulating website server access similar to RIPE Atlas’s policy on HTTP measurement targets and potentially other similar access control policies, which highlights PacketLab’s capability to mitigate security concerns through its access control designs.

## C Further Related Work

In addition to measurement platforms mentioned in Section 2, there also exist(ed) other platforms that focus on collecting and sharing insightful data, including ICLab [36], GFWatch [17], Censored Planet [58], PingER [29], Censys [6], Shodan [53], OpenINTEL [40], and DScope [41]. In essence, these platforms are large-scale measurement experiments that could serve as evaluation targets for PacketLab. For this work, we instead opted to perform the evaluation based on recent measurement studies with the aim of evaluating the PacketLab model via prevalent measurements.

## D Ethical Considerations

This work does not raise any ethical issues.

## References

1. Butskoy, D.: Traceroute for Linux (2024), <https://traceroute.sourceforge.net/>
2. CAIDA: Internet Tools Taxonomy (2004), <https://web.archive.org/web/20210422112114/https://www.caida.org/tools/taxonomy/>
3. CAIDA: Ark (2024), <https://catalog.caida.org/software/archipelago>
4. CAIDA: ITDK: Internet Topology Data Kit (2024), [https://catalog.caida.org/dataset/ark\\_itdk](https://catalog.caida.org/dataset/ark_itdk)
5. Cappos, J., Hemmings, M., McGeer, R., Rafetseder, A., Ricart, G.: EdgeNet: A Global Cloud That Spreads by Local Action. In: IEEE/ACM SEC '18
6. Censys: Censys Search (2024), <https://search.censys.io/>
7. Chang, H., Varvello, M., Hao, F., Mukherjee, S.: Can You See Me Now? A Measurement Study of Zoom, Webex, and Meet. In: Proceedings of ACM IMC '21
8. Chhabra, R., Murley, P., Kumar, D., Bailey, M., Wang, G.: Measuring DNS-over-HTTPS Performance around the World. In: Proceedings of ACM IMC '21
9. claffy, k., Hyun, Y., Keys, K., Fomenkov, M., Krioukov, D.: Internet Mapping: From Art to Science. In: IEEE CATCH '09
10. curl: curl (2024), <https://curl.se/>
11. Dang, T.K., Mohan, N., Corneo, L., Zavodovski, A., Ott, J., Kangasharju, J.: Cloudy with a Chance of Short RTTs: Analyzing Cloud Connectivity in the Internet. In: Proceedings of ACM IMC '21
12. Darwich, O., Rimlinger, H., Dreyfus, M., Gouel, M., Vermeulen, K.: Replication: Towards a Publicly Available Internet Scale IP Geolocation Dataset. In: Proceedings of ACM IMC '23
13. EdgeNet: Acceptable Use Policy | EdgeNet (2020), <https://www.edge-net.org/OMTpages/usage-policy.html>
14. Elmenhorst, K., Schütz, B., Aschenbruck, N., Basso, S.: Web Censorship Measurements of HTTP/3 over QUIC. In: Proceedings of ACM IMC '21
15. FCC: Measuring Broadband America | Federal Communications Commission (2024), <https://www.fcc.gov/general/measuring-broadband-america>
16. Hanemann, A., et al.: PerfSONAR: a service oriented architecture for multi-domain network monitoring. In: Proceedings of ICSOC '05
17. Hoang, N.P., et al.: How Great is the Great Firewall? Measuring China's DNS Censorship. In: USENIX Security '21
18. Hsu, A., Li, F., Pearce, P., Gasser, O.: A First Look at NAT64 Deployment In-The-Wild. In: Proceedings of PAM '24, Part I. Springer-Verlag (2024)
19. Kassem, M.M., Raman, A., Perino, D., Sastry, N.: A Browser-Side View of Starlink Connectivity. In: Proceedings of ACM IMC '22
20. Keys, K., Hyun, Y., Luckie, M., Claffy, K.: Internet-Scale IPv4 Alias Resolution With MIDAR. *IEEE/ACM Transactions on Networking* **21**(2) (2013)
21. Kosek, M., Schumann, L., Marx, R., Doan, T.V., Bajpai, V.: DNS Privacy with Speed? Evaluating DNS over QUIC and Its Impact on Web Performance. In: Proceedings of ACM IMC '22
22. Kreibich, C., Weaver, N., Nechaev, B., Paxson, V.: Netalyzr: Illuminating the Edge Network. In: Proceedings of ACM IMC '10
23. Kupoluyi, J., et al.: Muzeel: Assessing the Impact of JavaScript Dead Code Elimination on Mobile Web Performance. In: Proceedings of ACM IMC '22
24. Laniewski, D., Lanfer, E., Meijerink, B., van Rijswijk-Deij, R., Aschenbruck, N.: WetLinks: A Large-Scale Longitudinal Starlink Dataset with Contiguous Weather Data. In: Proceedings of TMA '24

25. Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M., Joosen, W.: Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In: Proceedings of NDSS '19
26. Levchenko, K., Dhamdhere, A., Huffaker, B., claffy, k., Allman, M., Paxson, V.: Packetlab: A Universal Measurement Endpoint Interface. In: Proceedings of ACM IMC '17
27. Liu, S., Bischof, Z.S., Madan, I., Chan, P.K., Bustamante, F.E.: Out of Sight, Not Out of Mind: A User-View on the Criticality of the Submarine Cable Network. In: Proceedings of ACM IMC '20
28. MacMillan, K., Mangla, T., Saxon, J., Marwell, N.P., Feamster, N.: A Comparative Analysis of Ookla Speedtest and Measurement Labs Network Diagnostic Test (NDT7). Proc. ACM Meas. Anal. Comput. Syst. **7**(1) (2023)
29. Matthews, W., Cottrell, L.: The PingER project: active Internet performance monitoring for the HENP community. IEEE Communications Magazine **38**(5) (2000)
30. Measurement Lab: Home - M-Lab (2024), <https://www.measurementlab.net/>
31. Measurement Lab: NDT (Network Diagnostic Tool) - Tests (2024), <https://www.measurementlab.net/tests/ndt/>
32. Michel, F., Trevisan, M., Giordano, D., Bonaventure, O.: A First Look at Starlink Performance. In: Proceedings of ACM IMC '22
33. Mok, R.K.P., Zou, H., Yang, R., Koch, T., Katz-Bassett, E., Claffy, K.C.: Measuring the Network Performance of Google Cloud Platform. In: Proceedings of ACM IMC '21
34. Munteanu, C., Saidi, S.J., Gasser, O., Smaragdakis, G., Feldmann, A.: Fifteen Months in the Life of a Honeyfarm. In: Proceedings of ACM IMC '23
35. Narayanan, A., et al.: Lumos5G: Mapping and Predicting Commercial MmWave 5G Throughput. In: Proceedings of ACM IMC '20
36. Niaki, A.A., et al.: ICLab: A Global, Longitudinal Internet Censorship Measurement Platform. In: IEEE S&P '20
37. Norman, M., et al.: CloudBank: Managed Services to Simplify Cloud Access for Computer Science Research and Education. In: Proceedings of ACM PEARC '21
38. Ookla: Speedtest by Ookla - The Global Broadband Speed Test (2024), <https://www.speedtest.net/>
39. OONI: Open Observatory of Network Interference (2024), <https://ooni.org/>
40. OpenINTEL: OpenINTEL: Active DNS Measurement Project (2024), <https://openintel.nl/>
41. Pauley, E., Barford, P., McDaniel, P.: DScope: A Cloud-Native Internet Telescope. In: USENIX Security '23
42. perfSONAR: View panel - perfSONAR Public - Dashboards - Grafana (2024), <https://stats.perfsonar.net/goto/Ia218mzNR?orgId=2>
43. Randall, A., et al.: Home is Where the Hijacking is: Understanding DNS Interception by Residential Routers. In: Proceedings of ACM IMC '21
44. Randall, A., et al.: Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers. In: Proceedings of ACM IMC '20
45. Rasaii, A., Gosain, D., Gasser, O.: Thou Shalt Not Reject: Analyzing Accept-Or-Pay Cookie Banners on the Web. In: Proceedings of ACM IMC '23
46. Reda, W., et al.: Path persistence in the cloud: A study of the effects of inter-region traffic engineering in a large cloud provider's network. SIGCOMM Comput. Commun. Rev. (2020)
47. RIPE NCC: HTTP Measurements with RIPE Atlas | RIPE Labs (2015), <https://labs.ripe.net/author/kistel/http-measurements-with-ripe-atlas/>

48. RIPE NCC: Ethics of RIPE Atlas Measurements | RIPE Labs (2016), <https://labs.ripe.net/author/kistel/ethics-of-ripe-atlas-measurements/>
49. RIPE NCC: RIPE Atlas (2024), <https://atlas.ripe.net/>
50. RIPE NCC: RIPE Atlas - Coverage (2024), <https://atlas.ripe.net/coverage/>
51. Rizvi, A.S.M., Huang, T., Esrefoglu, R., Heidemann, J.: Anycast Polarization in the Wild. In: Proceedings of PAM '24. Springer-Verlag
52. Sánchez, M.A., et al.: Dasu: Pushing Experiments to the Internet's Edge. In: Proceedings of NSDI '13
53. Shodan: Shodan Search Engine (2024), <https://www.shodan.io/>
54. Sommese, R., et al.: MANycast2: Using Anycast to Measure Anycast. In: Proceedings of ACM IMC '20
55. Spring, N., Peterson, L., Bavier, A., Pai, V.: Using PlanetLab for Network Research: Myths, Realities, and Best Practices. SIGOPS Oper. Syst. Rev. (2006)
56. Spring, N., Wetherall, D., Anderson, T.: Scriptroute: A Public Internet Measurement Facility. In: Proceedings of USITS '03
57. Stanford SLAC: Network Monitoring Tools (2023), <https://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html>
58. Sundara Raman, R., Shenoy, P., Kohls, K., Ensafi, R.: Censored Planet: An Internet-wide, Longitudinal Censorship Observatory. In: Proceedings of CCS '20
59. Sundaresan, S., Burnett, S., Feamster, N., de Donato, W.: BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks. In: Proceedings of USENIX ATC '14
60. The PacketLab Team: Home | PacketLab (2024), <https://packetlab.github.io/>
61. Umayya, Z., Malik, D., Gosain, D., Kumar Sharma, P.: PTPerf: On the Performance Evaluation of Tor Pluggable Transports. In: Proceedings of ACM IMC '23
62. Varvello, M., Chang, H., Zaki, Y.: Performance Characterization of Videoconferencing in the Wild. In: Proceedings of ACM IMC '22
63. Wan, G., et al.: On the Origin of Scanning: The Impact of Location on Internet-Wide Scans. In: Proceedings of ACM IMC '20
64. Yan, T.B., et al.: PacketLab: Tools Alpha Release and Demo. In: Proceedings of ACM IMC '22