ICT Express xxx (xxxx) xxx

Contents lists available at ScienceDirect

ICT Express

journal homepage: www.elsevier.com/locate/icte

LatScope: End-to-end latency decomposition across the cloud network stack

Bongwon Lee [a,1], Yunseo Jeong [b,1], Woongsub Shin [c], Sangtae Ha [c], Youngbin Im [b,*]

[a] MangoBoost, Seoul, Republic of Korea
[b] Department of Computer Science and Engineering, Ulsan National Institute of Science and Technology, Ulsan, Republic of Korea
[c] University of Colorado Boulder, Boulder, CO, USA

ARTICLE INFO

Keywords:
Multi-layer latency
Latency decomposition
Measurement tool

ABSTRACT

Data centers are rapidly scaling and becoming more complex, making it critical to pinpoint where latency arises across the network protocol stack. Existing tools primarily measure RTT or single-layer delays, while multi-layer approaches are often invasive or limited. We present LatScope, a practical multi-layer latency analyzer that (i) matches packets across layers to compute accurate inter-layer delays despite retransmissions and out-of-order delivery, (ii) synchronizes server clocks using XDP to enable precise inter-server delay breakdowns, and (iii) controls overhead through selective data extraction. We validate LatScope across diverse environments and show how its insights can be applied effectively in cloud deployments.

1. Introduction

Today, the scale of data centers is increasing rapidly due to the growth of artificial intelligence (AI), cloud computing, and big data analytics. In particular, data center network stacks are becoming more complex because of advances in cloud service models, virtualization and networking techniques, and the adoption of serverless computing. In these large-scale systems, latency has become a key performance indicator that determines quality of service (QoS) [1,2]. In addition, system performance degradation often arises not only from a single layer but also from bottlenecks across multiple layers of the network protocol stack. Therefore, it is critical to analyze detailed latencies across the components of the network protocol stack.

However, existing latency measurement tools are limited. Previous latency measurement studies [3–9] focus on RTT or specific-layer latency and target different environments, such as end-to-end connections, virtualized network systems, the Internet, the cloud, and programmable switches. However, most of these works do not provide comprehensive delay information across multiple layers of the network stack. [10] provides multi-layer information but requires modifications to the kernel and TCP/UDP headers, which creates a significant barrier to deployment. Recent eBPF-based system observability tools [11–16] provide a rich set of network-related information, but they are typically limited to certain layers.

To fill the gap between the need for comprehensive latency measurement and the limited measurement tools available, we propose LatScope. LatScope offers the following benefits over existing tools. (1) It provides multi-layer delay information, from the device layer to the socket layer, and also measures delays on virtual interfaces used for network virtualization. It computes inter-layer delays by matching packets across layers using cumulative byte counts derived from TCP sequence numbers. This approach accurately measures delays under various network conditions, including packet retransmissions and out-of-order packet delivery. (2) It provides a method to synchronize each server's clock with that of a central management server to compute precise inter-server delays. It uses XDP [17] to minimize synchronization error caused by variation in the transmission times of time-synchronization control packets within the network stack. (3) It provides a method to control overhead during delay analysis by selectively executing data extraction, balancing latency accuracy against CPU overhead. System administrators can choose the desired accuracy based on administrative requirements and the network and system environment.

To validate LatScope's versatility, we conduct a variety of experiments. (1) We run LatScope across different networks, virtualization environments, realistic applications, and contention scenarios. Our experiments show that, although the sender-side TCP buffer delay is the largest component of the end-to-end delay [3], delays in other layers can also be significant. For example, under competition among multiple

* Corresponding author.
E-mail addresses: bongwon.lee@mangoboost.io (B. Lee), jeong.ys@unist.ac.kr (Y. Jeong), wosh6271@colorado.edu (W. Shin), sangtae.ha@colorado.edu (S. Ha), ybim@unist.ac.kr (Y. Im).
[1] These authors contributed equally to this work.

https://doi.org/10.1016/j.icte.2026.04.006
Received 29 December 2025; Received in revised form 4 April 2026; Accepted 14 April 2026
Available online 16 April 2026

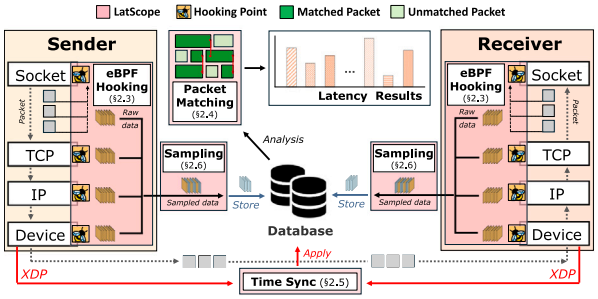


Fig. 1. Overview of LatScope.

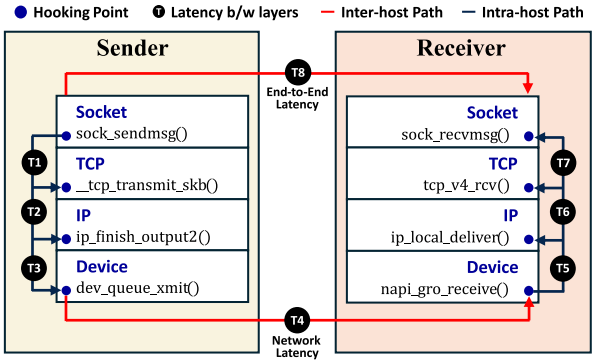


Fig. 2. Linux network protocol stack and hooking points of LatScope. Numbers are assigned to identify each delay segment uniquely, and used in Section 3.

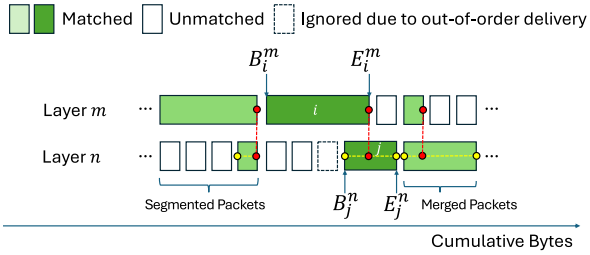


Fig. 3. Packet matching between two layers. jth packet at layer n is matched with ith packet at layer m if the cumulative byte count of i's last byte lies between the cumulative byte counts of j's first and last bytes.

flows, the delay between the TCP and IP layers at the sender and the delay between the TCP and socket layers at the receiver can increase substantially. (2) We also demonstrate how LatScope can be used to improve users' QoS. To guide the selection of system parameters for target flows, we examine how per-layer delays vary with flow priority and TCP buffer size. In addition, to guide the choice of a cloud virtual machine type for achieving a target QoS level, we compare delays across three virtual machine types on a commercial cloud service.

2. LatScope design

Below, we outline the key design challenges for building LatScope and our corresponding design components to overcome each challenge.

2.1. Design challenges

C1. Selecting the correct hooking points across the network stack. To accurately observe when packets are processed at each layer of the network stack, appropriate hook points should be selected. In addition, to facilitate deployment, network function hooking should be possible using existing kernel features without additional kernel modifications.

C2. Packet granularity mismatch across layers. Since different layers process packets of different sizes, simple one-to-one matching between packets is not applicable for measuring delay. This problem is more pronounced when mechanisms such as TSO (TCP Segment Offload) or GRO (Generic Receive Offload) are used, because a single large packet is divided or multiple packets are merged at different layers.

C3. Lack of reliable global time synchronization. It is difficult to obtain accurate absolute times for packet events across servers without setting up separate time synchronization protocols, such as NTP, on each server. In addition, it is known that the NTP-based time synchronization can be highly inaccurate depending on the network environment.

C4. Measurement overhead at scale. Unlike existing tools, if the delay is measured across all layers of the network stack simultaneously, the number of measurement points and events increases significantly, which can lead to substantial CPU overhead.

2.2. Design overview

Fig. 1 shows the overview of LatScope. eBPF hooks collect packet events at each layer of the network protocol stack, and adaptive sampling is applied to reduce measurement overhead. The resulting raw measurements are stored in the database. Using XDP-based time synchronization, timestamps across hosts are aligned, after which packet matching is performed to compute per-layer latency results.

The following subsections describe the key design components that enable LatScope to perform accurate, multi-layer latency measurement while addressing the design challenges described above.

2.3. eBPF hooking points for multi-layer probing

eBPF [18] has certain limitations in its usage, such as the available stack size and the number of probing points. Therefore, we carefully choose the probing points in the network stack to efficiently collect the appropriate network data, as shown in Fig. 2. We strike a balance between granularity and overhead when selecting probing points. Hooking eBPF code into multiple functions within each layer provides higher-granularity performance metrics but could impact each server's performance. We apply eBPF code to each layer's representative function to mitigate the performance impact. When selecting representative functions, we assess whether all information required for each layer is available within the function. We check the function's return value (to determine the number of bytes transmitted) and verify that the packet headers are fully formed (to extract the information we need from the headers), etc. We use Linux kernel 5.14.0.

2.4. Multi-layer packet matching

LatScope saves the initial TCP sequence number for each flow and utilizes the difference between subsequent sequence numbers and the initial sequence number to calculate the total cumulative bytes sent or received by each layer. The TCP and lower layers may experience packet loss or out-of-order delivery, leading to reverse sequence numbers. To ensure efficient packet matching, LatScope ignores packets with sequence numbers below the largest sequence number observed for a flow up to the current time. In the case of socket layers, there is no TCP sequence number information. However, since the data is delivered to or from the socket layer without loss or out-of-order delivery, we can determine the exact cumulative bytes for the socket layer. We ignore control packets without data, such as ACKs, SYNs, FINs, RSTs.

The collected cumulative byte information is stored in the database along with the corresponding timestamps. As shown in Fig. 3, to calculate the delay between two layers, LatScope uses each transmission and

Algorithm 1: Algorithm for server time synchronization and data timestamp calibration.

Input : $MST_{1,\ldots,N}$ // $MST_k \leftarrow$Management server send time of sync packet k
$MRT_{1,\ldots,N}$ // $MRT_k \leftarrow$Management server receive time of sync packet k
$ST_{1,\ldots,N}$ // $ST_k \leftarrow$Server time when receiving sync packet k
Output : $TD \leftarrow$Time difference b/w server and management server
Function `SyncServerTime`$(MST_{1,\ldots,N}, MRT_{1,\ldots,N}, ST_{1,\ldots,N})$:
  $K \leftarrow \arg\min_k (MRT_k - MST_k)$
  $One\_Way\_Delay \leftarrow \frac{(MRT_K - MST_K)}{2}$
  // Server time corresponding to the time when the management server sends the sync packet
  $STC \leftarrow ST_K - One\_Way\_Delay$
  $TD \leftarrow MST_K - STC$
  return TD

Input : $ST_i \leftarrow$Server time in measured data i
Output : $STC_i \leftarrow$Server time converted to the time of management server for measured data i
Function `CalibrateMeasuredDataTime`(ST_i, TD):
  $STC_i \leftarrow ST_i + TD$
  return STC_i

reception at one layer as matching points with another layer. Let B_i^m and E_i^m be the cumulative bytes for the first byte and last byte of the ith packet at layer m. jth packet at layer n is matched with ith packet at layer m if $B_j^n \leq E_i^m \leq E_j^n$. LatScope calculates the delay by comparing the timestamps of matched packets. This approach is also applied to calculate the delay between probed servers.

2.5. Cross-server time synchronization

When measuring latencies between servers or between the host and its virtual machines, it is necessary to synchronize time because each server or virtual machine uses its own time. Time synchronization is achieved by synchronizing each server's time with the management server's. Before starting delay measurement, the management server creates UDP-based time-synchronization packets for all probed servers and sends them to its *localhost* interface. An XDP program intercepts the packets and uses the metadata contained in those packets to redirect them to each probed server by setting *dst_addr*, *dst_port*, and the checksum. Afterward, the packets are forwarded to a physical interface for transmission. At this point, the management server stores the timestamp of each packet's transmission. When the XDP program on each server receives synchronization packets, it records the packet-reception time in the packet. Then, it exchanges the *src_addr, src_port* and *dst_addr, dst_port* and sends the packets back to the management server. When the management server receives packets, it records the reception times. It then uses Algorithm 1 to synchronize each server's time with the management server by calculating the time difference between them. Whenever measurement data is collected from the server, it is calibrated to the management server time. Note that we use XDP to measure RTT at the lowest possible layer of the network stack to exclude kernel-induced delay variation. To mitigate the effect of clock drift, we run the synchronization operation periodically, with a 64-second interval.

We evaluate the impact of clock drift on time-synchronization accuracy. Since the clock drift cannot be measured without devices to provide absolute time, such as a GPS receiver, we measured the variations of TD, the time difference between one server and the management server, as shown in Fig. 4. We measured the time difference across 100 synchronization operations and plotted the CDF of the differences between the measured time difference values and their minimum. We observe that the values are within 0.25 msec for all data points, indicating that the clock drift is not severe within each synchronization period.

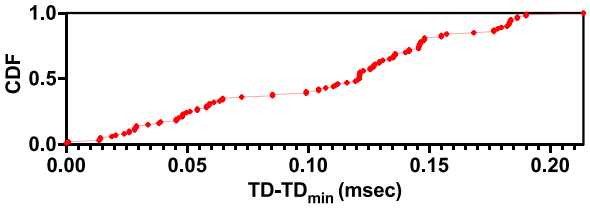


Fig. 4. CDF of variations of measured time differences between the server and management server.

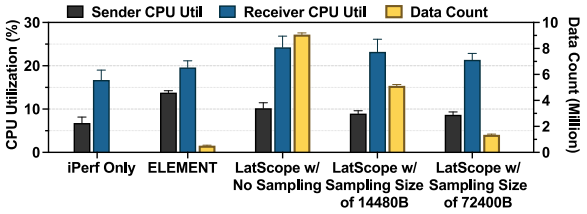


Fig. 5. CPU utilization and recorded data count for different sampling sizes.

2.6. Adaptive sampling for overhead control

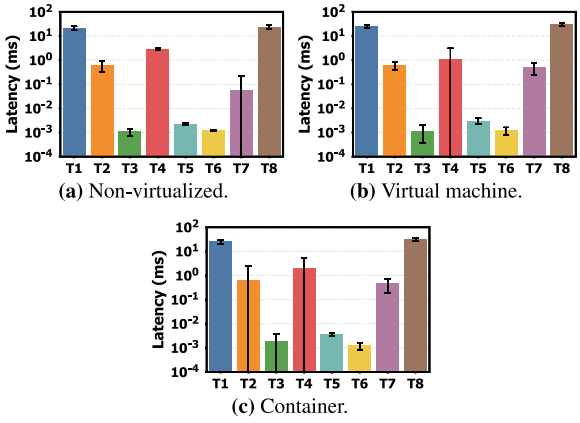
When a function hooked by LatScope is called, eBPF executes code to extract raw data that serves as the basis for delay analysis. Executing the data extraction code on every packet transmission or reception can improve the accuracy of measured delay, but it may impose high overhead on the server. The server can reduce overhead by selectively executing the data-extraction code on a subset of packets or at specified time intervals. This approach balances the accuracy in latency measurement with server CPU overhead as shown in Fig. 5. LatScope consumes 10.2% and 24.2% of CPU cycles at the sender and receiver, without sampling, respectively. If we use sampling with an interval of 72400 bytes (corresponding to 50 MTU-sized packets in Ethernet), the CPU consumption is reduced to 8.6% at the sender and 21.3% at the receiver while lowering the total number of data counts to 15%. We also observe that, for the sender, LatScope's CPU utilization is lower than that of ELEMENT for all sampling configurations. For the receiver, LatScope uses slightly more CPU cycles than ELEMENT. On the other hand, the number of recorded data points is much larger for LatScope than for ELEMENT, indicating that LatScope provides richer delay information at diverse layers. We run *iperf* throughput measurement tool over the Ethernet link for this test.

3. Evaluation

We validate the effectiveness of LatScope by presenting results from network delay analyses across various network environments. Specifically, we are interested in how delays across system components vary across different virtualization environments, network types, applications, and in contention scenarios. If not specified, we use *iperf* for each experiment.

3.1. Different virtualization environments

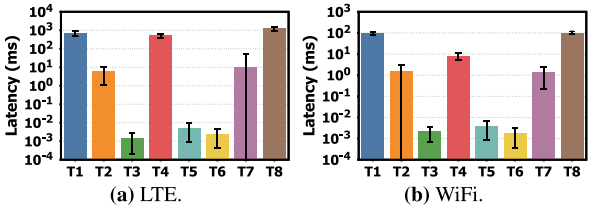
We compare the non-virtualized environment with commonly used virtualization configurations, such as virtual machines and containers, in Fig. 6. In the non-virtualized environment, within the end-to-end delay (T8), the delay between the socket and TCP at the sender (T1) is the largest. This observation is consistent across all virtualization environments. Due to virtualization overhead, the average delay in the virtual machine environment is higher than in the non-virtualized environment. In addition, the receiving server shows increased delay between TCP and socket (T7). This could be attributed to the



(a) Non-virtualized. (b) Virtual machine.

(c) Container.

Fig. 6. Delays in different virtualization environments.



(a) LTE. (b) WiFi.

Fig. 7. Delays in different network environments.

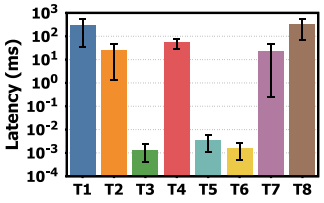
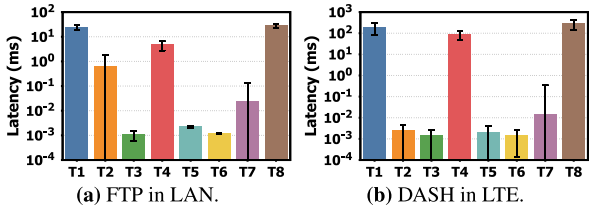


Fig. 8. Delays of the contention scenario in WiFi.

virtual machine's overhead in handling TCP packet reordering. With containers, the end-to-end delay (T8) is also slightly higher than in a non-virtualized environment. One difference from other environments is that the delay between TCP and IP at the sender (T2) and the network delay (T4) show larger variations.

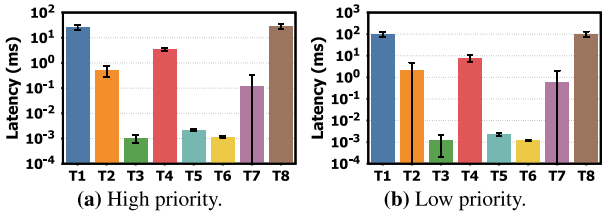
3.2. Different network environments

We additionally run LatScope across the LTE and WiFi networks. We utilize an LTE femtocell testbed for the LTE experiment that consists of DAUZ LDW931 for the UE, USRP B210 for the eNB, Open5GS [19] for LTE core software, and srsRAN [20] for eNB software. We also use ASUS RT-AX55 as the WiFi access point and ipTIME A1000UA-4DBI as the WiFi modem. As in the LAN experiment, in the LTE experiment, we observe that the delay between the socket and TCP layers (T1) accounts for most of the overall delay, as shown in Fig. 7(a). The significant differences are that the average of T1 is much higher and that the variation is slower. This is due to increased buffer bloat caused by the lower physical bandwidth and higher RTT of the LTE network. The network delay (T4) also exhibits large-scale variation. The average end-to-end delay (T8) in the WiFi is lower than that of LTE but higher than that of LAN, as shown in Fig. 7(b). This is consistent with the observed differences in bandwidth and RTT among the three networks. One notable difference compared to other networks is that the network delay (T4) is more unstable.



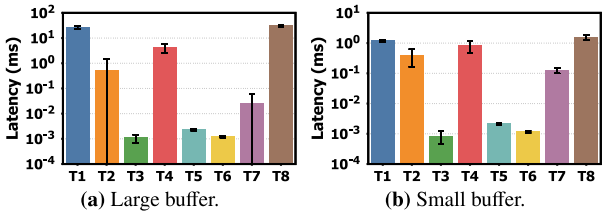
(a) FTP in LAN. (b) DASH in LTE.

Fig. 9. Delays in realistic applications.



(a) High priority. (b) Low priority.

Fig. 10. Delays with different flow priority settings.



(a) Large buffer. (b) Small buffer.

Fig. 11. Delay according to different TCP buffer sizes.

3.3. Contention scenario

Fig. 8 shows the delays of a flow when 20 flows are running simultaneously in the WiFi network. The results show that significant delays occur not only between the socket and TCP layers (T1) at the sender but also in other parts of the network stack. Specifically, we can observe an increase in delay between the TCP and IP layers (T2) at the sender and between TCP and socket layers (T7) at the receiver. In addition, the range of variation in the end-to-end delay (T8) is much larger than in a single flow scenario. This is due to factors such as reduced per-flow bandwidth and interflow interference.

3.4. Realistic applications

We evaluate LatScope through practical applications, including FTP file downloads and DASH video streaming. Fig. 9(a) depicts the delay of FTP downloads. Because FTP application operates similarly to *iperf*, the delays at each layer are comparable to those observed in the *iperf* experiment, but with slightly larger delays and delay variability. Fig. 9(b) shows the delay of a DASH application in an LTE network. Due to DASH's periodic download pattern, we observe substantial variation in the delay between the socket and TCP at the sender (T1) and in the end-to-end delay (T8). Overall, we observe LatScope's ability to measure each layer's delay in detail under dynamic traffic patterns in real-world applications, a capability not possible with any existing delay measurement tool, which primarily focuses on RTT [4,6–9] or specific layers [3].

4. Use cases and applications of LatScope

LatScope can be used in various ways due to its ability to obtain detailed delay information across different layers and components of

Table 1
Latencies of different virtual machine types in Naver Cloud. (unit: milliseconds).

VM type	sock_tcp (T1)	tcp_ip (T2)	ip_driver (T3)	network (T4)	driver_ip (T5)	ip_tcp (T6)	tcp_sock (T7)
s4-g3	25.2472	0.5414	0.0024	0.8348	0.0088	0.0022	0.921
s8-g3	12.4788	0.2761	0.0023	0.6728	0.0067	0.0022	0.3863
s16-g3	12.4241	0.2734	0.0023	0.5856	0.0067	0.0021	0.3658

the end-to-end path with minimal overhead. First, it can provide information for developing algorithms to control latency based on the flow's QoS requirements. Fig. 10 represents the measurements taken while varying the priority of one flow when there are 10 *iperf* flows. We observe significant differences in delay across different priority values. In Fig. 11, we vary the TCP buffer size and observe that appropriately tuning the buffer size can mitigate the effects of bufferbloat and regulate delay.

Second, LatScope can be used to detect gray failures [21] in cloud systems. Since gray failures exhibit subtle abnormal symptoms, we expect that detecting slight variations in delay patterns within a specific component and analyzing them at the whole-system level will help identify gray failures. Inspired by Perseus [22], which combines machine learning with a scoring mechanism to detect gray failures, we plan to explore system-wide detection strategies using LatScope as our future work.

Third, LatScope can be exploited to select appropriate cloud configurations to help service providers meet their QoS requirements. Table 1 shows the average latency of each layer for three representative virtual machine types in Naver Cloud, a cloud service in South Korea. We run an *iperf* flow between two virtual machines, and observe a significant delay difference between s4-g3 and the other types, particularly between the socket and TCP at the sender (T1).

5. Discussion

Packet filtering. The main reason we adopt the approach of ignoring packets with sequence numbers less than the maximum observed up to the current time in packet matching is to reduce the algorithm's overhead. Suppose we run the packet-matching algorithm on n packets at both the sender and the receiver, and packets are ordered by arrival time. If we do not use packet filtering, the packet-matching complexity becomes $O(n^2)$. On the other hand, if we use packet filtering, we can maintain one index variable per node and increment it whenever matched packets are found or out-of-order packets are encountered. The complexity of the packet matching becomes $O(n)$. To allow obtaining delay information for retransmitted or out-of-order packets, we also provide an option of disabling this method.

Support for high-speed networks. Since LatScope stores measurement data in the database, it can incur substantial overhead on high-speed networks such as 100 Gbps Ethernet. As future work, we aim to reduce the overhead by modifying LatScope to operate without raw data collection in the centralized database. This can be done by having the eBPF code running on the probed server compute the intra-host delay locally by storing recent packet transmission and reception information in the eBPF map, with sampling as needed. Once the measurement data for packet transmission and reception are used to compute the intra-host delay, they can be removed from the map, thereby limiting the total map size. End-to-end delays can be estimated by adding the network delay, measured separately by probing packets, to the intra-host delays on both sides.

Support for newer kernel versions. Since the kernel network stack evolves continuously, eBPF hook points may need to be updated with a new kernel version. In newer kernel versions, function names or arguments for each layer may change. We can support newer kernels by updating the LatScope hooking-point configuration file.

6. Related work

There has been a large body of work on network delay measurement. In this section, we focus on recent eBPF-based system observability tools and recent academic works on delay measurement. Comprehensive monitoring solutions, such as Netflix's Vector [23] and Performance Co-Pilot [24], provide high-resolution system metrics via agent-based collection. However, unlike LatScope, these tools are designed for broader system health rather than precise cross-layer delay analysis. Existing eBPF-based observability frameworks such as Retina [11], tcpdog [12], and netobserv-ebpf-agent [13] offer real-time flow-level monitoring, but their scope is limited to the network and transport layers. Kindling [14] and pwru [15] extend visibility to the application and kernel spaces, yet focus on debugging rather than precise latency decomposition.

Closer academic works also have significant limitations. vNetTracer [10] provides latency tracing in virtualized environments but requires kernel and header modifications. Pingmesh [9] offers large-scale active probing but cannot measure real application traffic. Other passive methods have critical constraints. PIRATE [4] could be inaccurate because of its estimation-based approach, which relies on request-to-request intervals as a proxy for response time. Furthermore, Torp [5] and other P4-based approaches [6–8] are not universally deployable, as they depend on programmable hardware. In contrast, LatScope employs eBPF-based probes across multiple layers — socket, TCP, IP, driver, and virtual devices — enabling *cross-layer latency decomposition* with low overhead.

7. Conclusion

We proposed LatScope, a practical multi-layer latency analyzer. Latscope matches packets across layers to compute the precise delay between layers. It synchronizes server times via XDP to provide accurate inter-server delays and controls measurement overhead through selective data extraction. The source code of LatScope is available at https://github.com/unist-n2sl/LatScope.

CRediT authorship contribution statement

Bongwon Lee: Writing – original draft, Visualization, Validation, Software, Methodology. **Yunseo Jeong:** Writing – review & editing, Software, Methodology, Data curation. **Woongsub Shin:** Writing – review & editing, Visualization, Validation, Conceptualization. **Sangtae Ha:** Writing – review & editing, Visualization, Validation, Supervision. **Youngbin Im:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Project administration, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP)-ITRC (Information Technology Research Center) grant funded by the Korea government(MSIT) (IITP-2026-RS-2021-II211817, 25%), the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2026-00349594, 25%), The Institute of Information & Communications Technology Planning & Evaluation (IITP) under Next-generation Cloud-native Cellular Network Leadership Program grant funded by the Korea government (MSIT) (RS-2026-00418784, 25%), and the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2026-00405128, 25%).

References

[1] S. Chen, S. Galón, C. Delimitrou, S.T. Manne, J.F. Martínez, Workload characterization of interactive cloud services on big and small server platforms, in: 2017 IEEE International Symposium on Workload Characterization, IISWC, IEEE, 2017, pp. 96–107.

[2] S. Chen, Y. Jiang, C. Delimitrou, J.F. Martínez, PIMCloud: QoS-aware resource management of latency-critical applications in clouds with processing-in-memory, in: 2022 IEEE International Symposium on High-Performance Computer Architecture, HPCA, IEEE, 2022, pp. 1086–1099.

[3] Y. Im, P. Rahimzadeh, B. Shouse, S. Park, C. Joe-Wong, K. Lee, S. Ha, I sent it: Where does slow data go to wait? in: Proceedings of the Fourteenth EuroSys Conference 2019, 2019, pp. 1–15.

[4] B.V. Shobhana, Y. lin Chien, J. Diamant, B. Nath, S.L. Feibish, S. Narayana, Measuring round-trip response latencies under asymmetric routing, 2025, arXiv:2505.14358. URL: https://arxiv.org/abs/2505.14358.

[5] X. Chen, H. Liu, J. Guo, X. Jiang, Q. Huang, D. Zhang, C. Wu, H. Zhou, Torp: Full-coverage and low-overhead profiling of host-side latency, in: IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, 2022, pp. 1349–1358, http://dx.doi.org/10.1109/INFOCOM48880.2022.9796758.

[6] X. Chen, H. Kim, J.M. Aman, W. Chang, M. Lee, J. Rexford, Measuring TCP Round-Trip Time in the Data Plane, SPIN '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 35–41, http://dx.doi.org/10.1145/3405669.3405823.

[7] J. Gomez, E.F. Kfoury, J. Crichigno, G. Srivastava, Reducing the impact of RTT unfairness using P4-programmable data planes, in: Proceedings of the IEEE International Conference on Communications, ICC, IEEE, Denver, CO, USA, 2024, pp. 427–432, http://dx.doi.org/10.1109/ICC51166.2024.10622372.

[8] F. Ihle, E. Zink, M. Menth, Enhancements to P4TG: Histogram-based RTT monitoring in the data plane, 2025, arXiv preprint arXiv:2507.15382.

[9] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, V. Kurien, Pingmesh: A large-scale system for data center network latency measurement and analysis, in: SIGCOMM, pp. 139–152.

[10] K. Suo, Y. Zhao, W. Chen, J. Rao, vNetTracer: Efficient and Programmable Packet Tracing in Virtualized Networks, IEEE.

[11] Microsoft, Retina, 2025, URL: https://github.com/microsoft/retina.

[12] Gridgentoo, Tcpdog, 2025, URL: https://github.com/gridgentoo/tcpdog.

[13] netobserv, Netobserv-ebpf-agent, 2025, URL: https://github.com/netobserv/netobserv-ebpf-agent.

[14] KindlingProject, Kindling, 2024, URL: https://github.com/KindlingProject/kindling.

[15] Cilium, Packet, where are you? 2025, URL: https://github.com/cilium/pwru.

[16] hengyoush, Kyanos, 2025, URL: https://github.com/hengyoush/kyanos.

[17] T. Høiland-Jørgensen, J.D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, D. Miller, The express data path: Fast programmable packet processing in the operating system kernel, in: Proceedings of the 14th International Conference on Emerging Networking Experiments and Technologies, 2018, pp. 54–66.

[18] Matt Fleming, A thorough introduction to eBPF, 2017, https://lwn.net/Articles/740157/.

[19] Open5GS, Open5GS, 2025, URL: https://open5gs.org/.

[20] srsRAN Project, srsRAN project - open source RAN, 2025, https://www.srslte.com/.

[21] P. Huang, C. Guo, L. Zhou, J. R.Lorch, Y. Dang, M. Chintalapati, R. Yao, Gray failure: The achilles'heel of cloud-scale systems, in: HOTOS, pp. 150–155.

[22] R. Lu, E. Xu, Y. Zhang, F. Zhu, Z. Zhu, M. Wang, Z. Zhu, G. Xue, J. Shu, M. Li, et al., Perseus: A $\{Fail - Slow\}$ detection framework for cloud storage systems, in: 21st USENIX Conference on File and Storage Technologies, FAST 23, 2023, pp. 49–64.

[23] M. Spier, A. Ather, B. Gregg, Introducing vector: Netflix's on-host performance monitoring tool, 2015, https://netflixtechblog.com/introducing-vector-netflixs-on-host-performance-monitoring-tool-c0d3058c3f6f.

[24] Red Hat, Performance co-pilot, 2025, https://pcp.io/.