Round-Trip Time Measurement in FABRIC

Vaibhav Sharan

School of Computing and Augmented Intelligence

Arizona State University

vsharan1@asu.edu

***Abstract*— This paper presents a comprehensive study of Round-Trip Time (RTT) within the intricate framework of the FABRIC testbed network. RTT, a fundamental metric of network performance, serves as a crucial determinant of user experience and the overall quality of a network. This study explores the facets of RTT, delving into its impact on user satisfaction and network efficiency. To achieve these objectives, various RTT measurement methods are compared and employed. These methods include One-way Latency (OWL) utilizing Precision Time Protocol (PTP), a novel algorithm implemented in the data-plane using P4 programmable high-performance BMv2 switches, continuous TCP traffic analysis, and the traditional UNIX "ping" command. By leveraging these diverse techniques, this study provides a nuanced understanding of RTT measurement behaviors.**

Keywords—Network Monitoring, TCP RTT, Data Plane, Latency, P4, FABRIC

I. Introduction and Motivation

This paper presents a study of Round-Trip Time (RTT) for the FABRIC [1] testbed network. Round-Trip Time is one of the basic metrics of a network that directly affects the user experience and quality of the network. It is also an indicator of performance issues in the network like congestion, server throttling, and changes in the routing. The factors that impact RTT the most are distance, number of network hops, traffic levels, transmission medium, and server response time. RTT stands as a pivotal metric shaping user experience and network efficiency [5].

The motivation behind this study stems from the crucial role RTT plays in shaping the digital experience of millions of users worldwide. In an era where real-time applications and services are ubiquitous, users expect instant responses. High RTT values directly translate to delays in data transmission, leading to frustrated users and diminished user experience. By comprehensively studying the methods of measuring RTT, we aim to help other studies uncover strategies to minimize latency, thereby elevating user experience across various digital platforms [7, 8].

Understanding RTT variations is key to identifying and rectifying performance bottlenecks within the network infrastructure [4]. Congestion, server throttling, or changes in routing can significantly degrade RTT, impacting the network's overall efficiency. This study aims to measure RTT in the specific context of the FABRIC testbed network.

Moreover, this research contributes to the broader domain of networking technologies. Insights gained from this study can inform the design and implementation of future networks. By unraveling the complexities of RTT measurement dynamics, we pave the way for innovations in network protocols, routing algorithms, and transmission mediums. This, in turn, supports the development of next-generation networks capable of handling the demands of emerging technologies such as the Internet of Things (IoT), 5G, and beyond.

In essence, the motivation behind this study is to help researchers who are performing experiments on FABRIC to be more efficient and successful in creating experiments for the harmonious integration of technology into our daily lives.

II. RTT Measurement Methods

This paper studies and makes a comparison between different methods of measuring RTT. The following methods will be employed:

A. Using One-way Latency (OWL)

The FABRIC [1] testbed provides a Measurement Framework library (MFLib) that enables the experiments to go deeper into the network slices by adding monitoring and measurement services of network nodes and traffic. I will be using one of these services called One-way Latency (OWL). It is a network latency measurement tool that leverages Precision Time Protocol (PTP) on the FABRIC testbed.

It is different from the traditional method for calculating RTT because it timestamps a UDP packet when it leaves the source node and again when it is received at the destination node. The reason that it is more accurate than the traditional method is because of the difficulty associated with clock synchronization across devices in the network. The Precision Time Protocol (PTP) is a protocol used to synchronize clocks throughout a computer network. On a local area network, it achieves clock accuracy in the sub-microsecond range, making it suitable for measurement and control systems [9].

This method provides accurate one-way latency measurements, enabling the assessment of individual segments of the RTT journey. At the time of writing of this paper, only 10 sites in FABRIC are PTP-compatible. Therefore, we can use this method for those sites only.

B. Using P4 Programmable Data Plane Switch

An algorithm [2] has been developed by researchers in Princeton which passively and continuously monitors the RTT of TCP traffic by calculating the time difference between packets and their acknowledgments. This algorithm is implemented in the data-plane with the help of the P4 programmable high-performance BMv2 switch. The algorithm uses a multi-stage hash table data structure to maintain records of TCP packets for matching their ACK packets. The use of P4 programmable switches enhances the experiment's flexibility and allows for real-time, granular

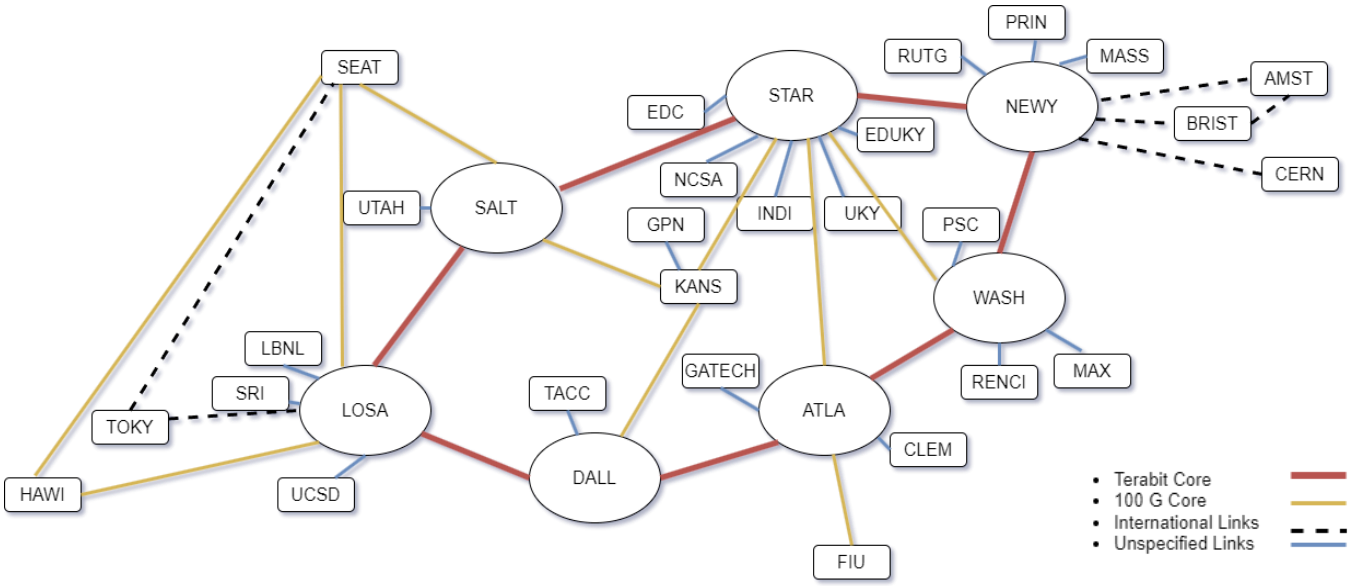

Figure 1: FABRIC Network sites and their connections across the globe.

analysis of RTT behavior. The code for implementation of this work is available publicly for others to utilize in their studies [2].

C. *Using packet capture of a continuous TCP connection*

A stream of continuous TCP traffic will be sent between two nodes of the FABRIC network and the packets will be captured. This allows for measurement and analysis of the RTT behavior of the complete duration of traffic by matching the TCP packets and their corresponding acknowledgments. This is different from the previous method since this will measure the RTT on the application layer whereas the former used the data plane. The importance of this method lies in the fact that quite a few applications like online video games and interactive media players utilize similar methods to measure RTTs for their purposes.

D. *Using UNIX "ping" command*

Finally, as a benchmark, the traditional UNIX "ping" command utilizing the Internet Control Message Protocol (ICMP) is employed. This method provides a standard for comparison, validating the results obtained through specialized techniques against a widely recognized RTT measurement approach.

III. Experiment Description

The experiment detailed in this study harnesses the robust resources provided by the FABRIC testbed, an interconnected network comprising 33 sites [10], each endowed with substantial computing and storage capabilities. These sites are seamlessly linked by shared and dedicated high-speed optical connections, forming the backdrop for our exploration into the intricacies of RTT dynamics.

Given the extensive scale of the experiment involving 33 sites, crafting a full factorial experiment design for every possible pair of locations becomes a laborious and time-consuming endeavor. To optimize this process, I will strategically utilize the terabit core infrastructure to select specific node pairs. These chosen pairs will serve as focal points for conducting experiments on significant links, ensuring a targeted and efficient approach to the study.

Below is a more in-depth overview of the experiment's methodology and scope.

A. *Using One-way Latency (OWL)*

In this method, I create nodes on different sites, with basic NICs connected to FABRIC's FABnetv4 internet. The Current list of the 10 PTP-compatible sites: STAR, MAX, MICH, MASS, UTAH, NCSA, UCSD, FIU, CLEM, CERN.

The first step is to create a slice in FABRIC with the nodes on these sites and then set up the PTP. The Measurement Framework Git repository has all the required steps and scripts for setting up PTP. The OWL framework works in an isolated environment with the help of Docker.

In this experiment, there will be one "owl-capturer" container and multiple "owl-sender" containers running on each site. For each link, sender and capturer containers will start. If one node is a destination for more than 1 sender, only 1 capturer container instance will be created. The "owl-capturer" runs in a docker container in the destination node and the "owl-sender" is running in the source node. After running the experiment for 600 seconds for each link, we obtained a PCAP file at each "owl-capturer" instance consisting of around 7,000 UDP packets. We analyze the captured PCAP files using the OwlDataAnalyzer provided by the OWL framework.

The median of the approximately 14,000 observations for each source-destination pair is considered as the result. The results are summarized in Table 1.

Table 1: Results of experiments using One-Way Latency framework

Source	Destination	One-way Latency (ms)	Standard Deviation (ms)
STAR	CLEM	15.25	0.033
CLEM	STAR	15.26	0.028
STAR	CERN	51.79	4.237

CERN	STAR	61.54	0.029
CLEM	CERN	62.67	0.021
CERN	CLEM	52.85	4.632
CERN	MASS	50.88	4.510
MASS	CERN	56.00	4.842
CERN	MAX	55.91	4.613
MAX	CERN	51.31	4.845
CERN	MICH	64.11	4.502
MICH	CERN	59.39	4.927
CERN	NCSA	62.75	0.020
NCSA	CERN	57.98	4.928
CERN	UTAH	67.39	0.075
UTAH	CERN	72.29	4.843
MASS	MAX	7.70	0.074
MAX	MASS	7.69	0.080
MASS	MICH	15.90	0.017
MICH	MASS	15.94	0.014
MASS	NCSA	14.53	0.019
NCSA	MASS	14.53	0.016
MASS	STAR	13.31	0.018
STAR	MASS	13.33	0.020
MASS	UTAH	28.86	0.015
UTAH	MASS	28.72	0.021
MAX	MICH	11.18	0.081
MICH	MAX	11.14	0.085
MAX	NCSA	9.70	0.076
NCSA	MAX	9.71	0.079
MAX	STAR	8.48	0.077
STAR	MAX	8.54	0.080
MAX	UTAH	24.04	0.078
UTAH	MAX	23.89	0.021
MICH	NCSA	3.85	0.016
NCSA	MICH	3.82	0.018
MICH	STAR	2.64	0.024
STAR	MICH	2.61	0.017
MICH	UTAH	18.18	0.008
UTAH	MICH	18.00	0.014
NCSA	STAR	1.23	0.018
STAR	NCSA	1.24	0.028
NCSA	UTAH	16.78	0.023
UTAH	NCSA	16.63	0.015
STAR	UTAH	15.57	0.018
UTAH	STAR	15.41	0.028

Analyzing these findings, it appears that CERN behaves unusually in terms of significant variations in one-way latencies compared to other sites. The underlying cause for this behavior remains unclear. One speculation I have is that this anomaly could be attributed to CERN being an international site with distinct routing paths configured for inbound and outbound traffic.

Another observation is that there are unusually high standard deviation values associated with some of the results of CERN. The reason for the variability of one-way latencies could be because of multiple network routes for traffic to and from CERN in the experiment.

These results are one-way latencies only as the name suggests. To calculate the RTT we can add these for corresponding each source-destination pair. RTT calculated this way can be seen in Table 2.

Table 2: RTT using One-Way Latency Framework

Node A	Node B	RTT (ms)
STAR	CLEM	30.51
CLEM	CERN	115.52
CERN	STAR	113.33
CERN	MASS	106.88
CERN	MAX	107.22
CERN	MICH	123.50
CERN	NCSA	120.73
CERN	UTAH	139.68
MASS	MAX	15.39
MASS	MICH	31.84
MASS	NCSA	29.06
MASS	STAR	26.64
MASS	UTAH	57.58
MAX	MICH	22.32
MAX	NCSA	19.41
MAX	STAR	17.02
MAX	UTAH	47.93
MICH	NCSA	7.67
MICH	STAR	5.25
MICH	UTAH	36.18
NCSA	STAR	2.47
NCSA	UTAH	33.41
STAR	UTAH	30.98

B. *Using P4 Programmable Data Plane Switch*

In this method, we create the BMv2 switches along with their respective node and implement the algorithm developed by Chen et al [2]. We have implemented this algorithm on a commodity programmable switch BMv2 using the P4 language [11].

For this experiment, we take the same sites as the previous experiment for the sake of comparison of data and by keeping the time constraints in mind.

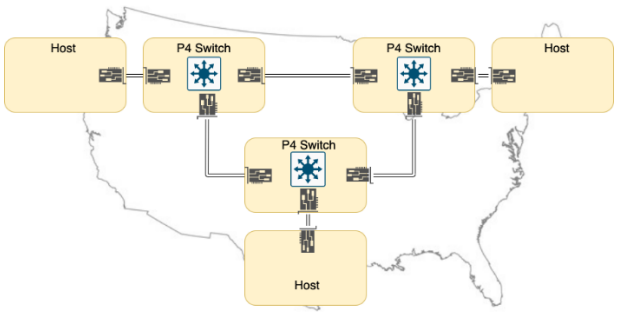

Figure 2: An example of FABRIC slice topology employed in this method.

We use a data-plane algorithm [2] designed for the continuous and passive monitoring of the Round-Trip Time (RTT) of TCP traffic. This algorithm operates by

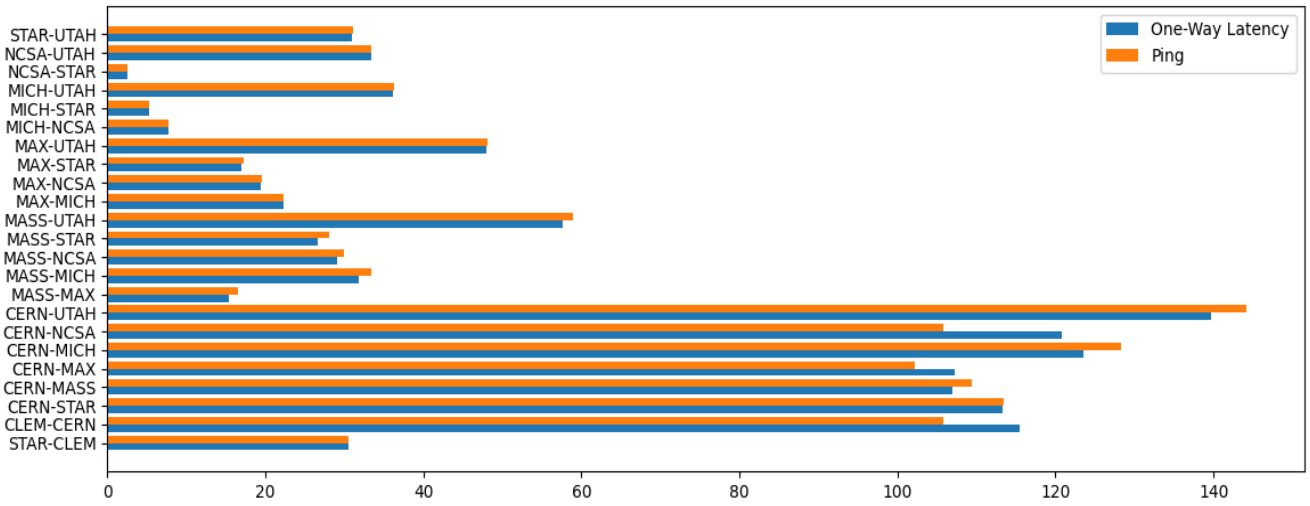

Figure 3: Bar plot comparing the RTTs observed using OWL and using ping command. (values in ms)

correlating data packets with their corresponding acknowledgments and computing the time difference. In contrast to conventional measurement systems that rely on active probing or focus solely on SYN/ACK packets, this algorithm passively generates numerous samples for prolonged connections. This feature allows network operators to promptly detect abnormal RTT increases, serving as an early indicator of potential security or performance issues within the network in real time. The algorithm employs a multi-stage hash table data structure to manage records for in-flight packets. Records that do not receive their acknowledgments are lazily expired and overwritten efficiently.

This algorithm [2] achieves its functionality by correlating an outgoing TCP packet, identified by its sequence number, with an incoming packet that bears the corresponding acknowledgment number. As depicted in Figure 4, the algorithm captures Round-Trip Time (RTT) samples extending beyond the initial three-way handshake, enabling continuous monitoring throughout a TCP session. Operating within the data plane of commodity programmable switches provides the capability to measure per-packet RTT in real-time, at a higher linear rate.

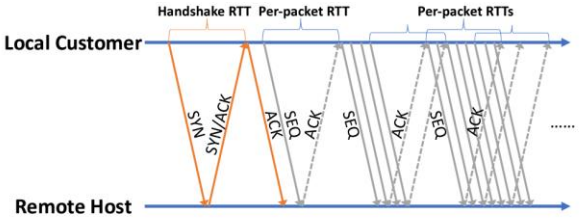

Figure 4: Match data and ACK packets to measure RTT.

We use the utility scripts provided by Chen et al [2] to analyze the PCAP files generated and convert them into CSV files with the RTTs. The median of 10,000 values from these CSV files are tabulated in Table 3.

Table 3: RTT using P4 Data Plane Switch

Node A	Node B	RTT (ms)
STAR	CLEM	28.164
CLEM	CERN	104.723
CERN	STAR	110.469
CERN	MASS	105.999
CERN	MAX	100.112
CERN	MICH	122.284
CERN	NCSA	102.937
CERN	UTAH	138.184
MASS	MAX	14.127
MASS	MICH	29.025
MASS	NCSA	27.008
MASS	STAR	23.695
MASS	UTAH	55.231
MAX	MICH	21.502
MAX	NCSA	19.317
MAX	STAR	14.849
MAX	UTAH	45.269
MICH	NCSA	6.666
MICH	STAR	3.324
MICH	UTAH	34.697
NCSA	STAR	2.278
NCSA	UTAH	32.618
STAR	UTAH	28.745

C. Using packet capture of a continuous TCP connection

This method will utilize the application layer to measure RTTs from a continuous stream of TCP traffic. The observations from this method are expected to be higher than the other methods since it goes through more layers of the network stack than the other methods.

Scapy [12] is a powerful Python library for crafting and sending packets. We use it to send TCP Request packets and measure the round-trip time (RTT). We send 1,000 TCP packets from each source to a destination and calculate the RTTs with the help of their replies.

We are limiting the use of sites to that of previous experiments again for the sake of effective comparison. The medians of the collected data are tabulated in Table 4.

Table 4: TCP RTT measurement in the Application Layer

Node A	Node B	RTT (ms)
STAR	CLEM	35.390
CLEM	CERN	118.716

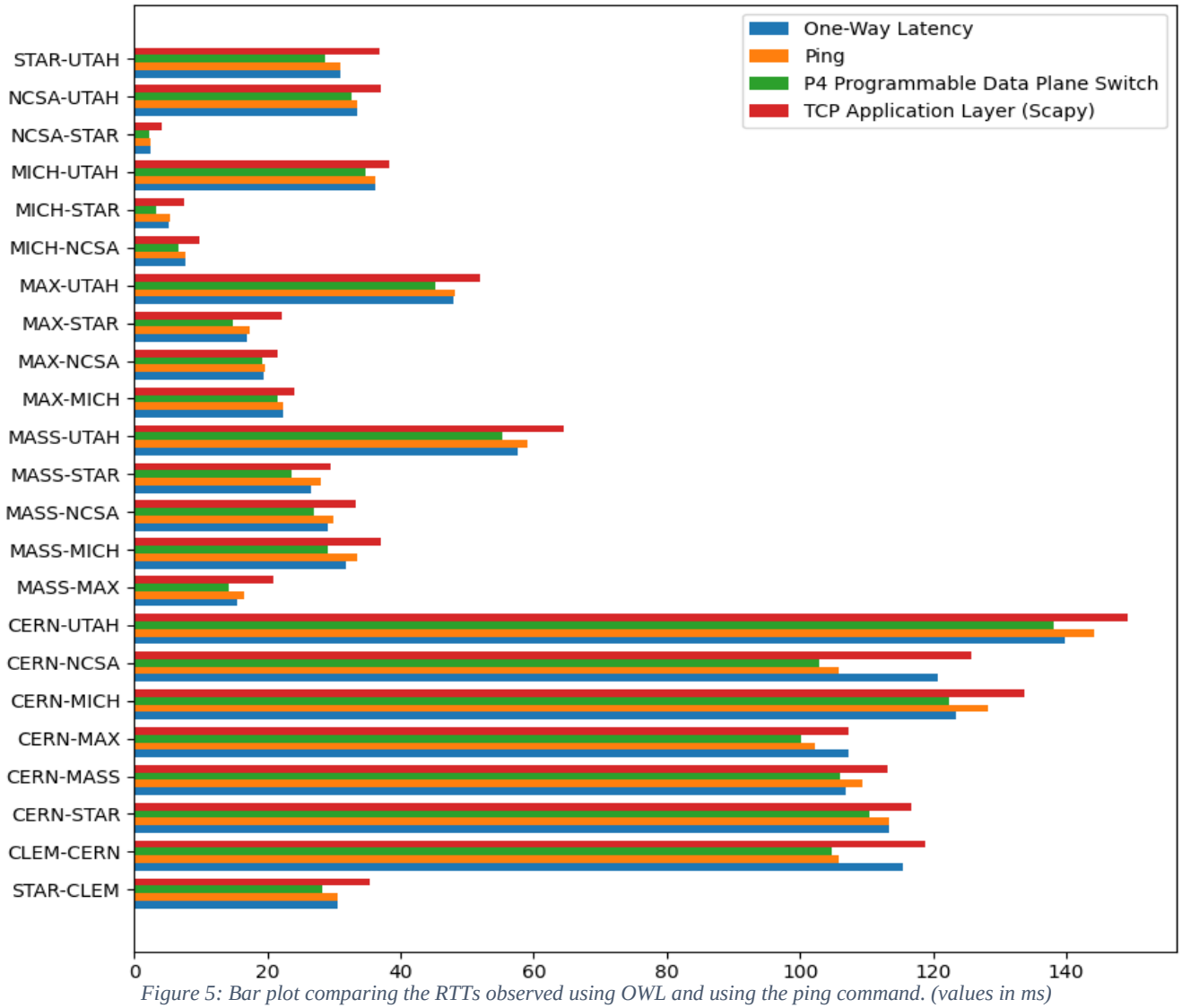

Figure 5: Bar plot comparing the RTTs observed using OWL and using the ping command. (values in ms)

CERN	STAR	116.754
CERN	MASS	113.139
CERN	MAX	107.337
CERN	MICH	133.599
CERN	NCSA	125.720
CERN	UTAH	149.098
MASS	MAX	20.906
MASS	MICH	36.959
MASS	NCSA	33.374
MASS	STAR	29.575
MASS	UTAH	64.566
MAX	MICH	23.991
MAX	NCSA	21.640
MAX	STAR	22.120
MAX	UTAH	52.016
MICH	NCSA	9.819
MICH	STAR	7.589
MICH	UTAH	38.384
NCSA	STAR	4.108
NCSA	UTAH	37.090
STAR	UTAH	36.876

D. Using the UNIX "ping" command

This method acts as the benchmark for comparison of the results obtained using the other methods. The values in Table 5 are medians of 15 observations. The standard deviations in this case were very small (of the order of 10^{-2} ms) compared to actual values, and considering the relatively small sample size of the collected data, it was omitted from the table.

Table 5: RTT using ping

Node A	Node B	RTT (ms)
STAR	CLEM	30.53
CLEM	CERN	105.83
CERN	STAR	113.41
CERN	MASS	109.40
CERN	MAX	102.16
CERN	MICH	128.31
CERN	NCSA	105.84
CERN	UTAH	144.23
MASS	MAX	16.55
MASS	MICH	33.44
MASS	NCSA	30.01
MASS	STAR	28.11

MASS	UTAH	58.99
MAX	MICH	22.37
MAX	NCSA	19.60
MAX	STAR	17.31
MAX	UTAH	48.12
MICH	NCSA	7.74
MICH	STAR	5.34
MICH	UTAH	36.26
NCSA	STAR	2.51
NCSA	UTAH	33.45
STAR	UTAH	31.05

IV. Measurement Results and Analysis

After the experiment is performed, the results from the different methods will be tabulated and compared, and medians of sufficiently large observations will be taken for visualization and analysis.

Figure 3 depicts a comparison between the observations taken. Initial observations point toward the fact that the RTT values from the OWL method are slightly smaller than that of ping. However, there are a few cases where the opposite is true. This finding might indicate OWL is a more accurate method for measuring RTTs.

Figure 5 visualizes the results of this whole experiment. We now try to conclude the various methods used based on these results with the help of our knowledge about the methods and underlying technologies.

The RTT values collected from the P4 switches using the algorithm developed by Chen et al [2] are consistently lower than all the other methods. This pattern can be attributed to the fact that the calculation is being done on the switches instead of the nodes. This method is the true measure of the round-trip time of a network between two switches. Since the medium of communication is not known from a switch to the node in a network, the only common part of a traffic routing path between two nodes on different sites is the connection of a switch to another switch.

The last method of calculating RTT was a Python script using Scapy. It was running on the application layer of the nodes in FABRIC. As expected, these values are consistently higher than the rest of the data. This behavior can be explained by the logic that the packets must pass through the kernel space to the Python program execution runtime for TCP acknowledgment in addition to the rest of the network.

The GitHub repository https://github.com/vaibharn/RTT-Measurement-FABRIC contains all the code and files related to this experiment.

V. Conclusion

With the help of this study, we try to gain insights into the difference between different RTT measuring methods and the accuracy of various methods. This paper offers a comprehensive analysis of Round-Trip Time (RTT) within the intricate network landscape of the FABRIC testbed. Through a meticulous exploration of diverse RTT measurement methods, this research will provide valuable insights for future work into the factors influencing network latency and user experience.

This study serves as a foundational resource for network engineers, researchers, and policymakers, providing valuable insights into the complexities of RTT measurement dynamics.

References

[1] I. Baldin, K. Wang, A. Nikolich, T. Lehman, J. Griffioen, P. Ruth and I. S. Monga, “FABRIC: A National-Scale Programmable Experimental Network Infrastructure” IEEE Internet Computing, Volume: 23, Issue: 6, 01 Nov.-Dec. 2019, doi 10.1109/MIC.2019.2958545.

[2] Xiaoqi Chen, Hyojoon Kim, Javed M. Aman, Willie Chang, Mack Lee, and Jennifer Rexford. “Measuring TCP Round-Trip Time in the Data Plane”. Proceedings of the Workshop on Secure Programmable Network Infrastructure (SPIN '20). Association for Computing Machinery, New York, NY, USA, 35–41, 2020, doi: 10.1145/3405669.3405823.

[3] E. F. Kfoury, J. Crichigno and E. Bou-Harb, "An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends," in IEEE Access, vol. 9, pp. 87094-87155, 2021, doi: 10.1109/ACCESS.2021.3086704.

[4] Jay Aikat, Jasleen Kaur, F. Donelson Smith, Kevin Jeffay. “Variability in TCP round-trip times”. IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement. October 2003, 279–284, doi: 10.1145/948205.948241

[5] Richard Cziva, Christopher Lorier, and Dimitrios P. Pezaros. 2017. Ruru: High-speed, Flow-level Latency Measurement and Visualization of Live Internet Traffic. In Proceedings of the SIGCOMM Posters and Demos (SIGCOMM Posters and Demos '17). Association for Computing Machinery, New York, NY, USA, 46–47. doi: 10.1145/3123878.3131981

[6] Hao Ding and Michael Rabinovich. 2015. TCP Stretch Acknowledgements and Timestamps: Findings and Implications for Passive RTT Measurement. SIGCOMM Comput. Commun. Rev. 45, 3 (July 2015), 20–27. Doi: 10.1145/2805789.2805793

[7] Amogh Dhamdhere, David D. Clark, Alexander Gamero-Garrido, Matthew Luckie, Ricky K. P. Mok, Gautam Akiwate, Kabir Gogia, Vaibhav Bajpai, Alex C. Snoeren, and Kc Claffy. Inferring persistent interdomain congestion. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18). 1–15. doi: 10.1145/3230543.3230549

[8] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2019. Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience. Proc. ACM Meas. Anal. Comput. Syst. 3, 3, Article 56 (December 2019), 25 pages. doi: 10.1145/3366704

[9] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," in IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002), vol., no., pp.1-269, 24 July 2008, doi: 10.1109/IEEESTD.2008.4579760.

[10] About FABRIC: https://portal.fabric-testbed.net/about/about-fab

[11] The P4 Language Consortium. 2018. P416 Language Specification. https://p4.org/ p4-spec/docs/P4-16-v1.1.0-spec.html. (Nov.2018).

[12] Scapy: https://github.com/secdev/scapy