On the Unreliability of Network Simulation Results FROM Mininet and iPerf

Benjamin Hardin*, Douglas Comer, and Adib Rastegarnia

***Abstract*—The networking research community and industry both use the Mininet network simulator to assess new network architectures and predict how new designs will perform. Mininet employs an emulation approach that uses concurrent processes running in a single computer instead of physical hardware. A user specifies a network consisting of network switches, routers, and host computers that have network interface connections and links connecting all the pieces. Mininet creates software artifacts to represent each of the network devices and allows application programs to send packets across the resulting network. Researchers often use the iPerf application to measure network performance. Many research papers report results from Mininet and iPerf and use the results to validate new designs for Software-Defined Networks (SDNs). However, there has been little investigation into the scenarios where these emulations can perform different than intended.**

The goal of this paper is to understand the edge cases of these emulation methods and understand the severity of these scenarios. This paper reports surprising anomalies in the results of Mininet and iPerf. We show that the choice of apparently valid configuration options can make the reported throughput completely invalid. Our initial discoveries focused on a complex simulation of a data center network. However, we were able to show that Mininet produces completely invalid results for a basic case: network traffic traveling across a single emulated link between two switches with no other network traffic. The paper makes recommendations for ways to configure Mininet to avoid some of the anomalies.

***Index Terms*—Software-defined network, mininet, simulation, iPerf, modeling, performance analysis.**

I. Introduction

Researchers and engineers in industry and academia commonly use the Mininet network simulation tool to evaluate new network architecture and designs [1–10]. Mininet simulates a computer ethernet network by using a computer CPU rather than the physical hardware of switches, routers, links, and network cards. Simulation can be advantageous because it allows a user to assess both the feasibility and performance of a design before attempting large-scale investments in hardware. The networking research community and industry both use the Mininet network simulator to assess new network architectures and predict how new designs will perform. Mininet employs an emulation approach that uses concurrent processes running in a single computer instead of equipment. In particular, simulation allows one to evaluate how a design performs as the size of a network scales.

To use Mininet, one specifies the equipment to be simulated along with the exact interconnection of equipment to form a network. Equipment includes network switches, routers, and host computers that each have network interfaces used to attach them to the network. Specifically, Mininet can simulate the Ethernet switches commonly used in corporate networks and cloud datacenter networks and allows a user to specify the throughput of the links (commonly called capacity, speed, or bandwidth).

Mininet uses an emulation approach in which concurrent processes run on a single computer to emulate all the pieces of hardware in the network. The Mininet approach has the advantage of allowing the user to run conventional network management and measurement applications, such as controllers used for Software-Defined Networking (SDN) [1] and the iPerf application used to measure network performance [2-11]. The combination of Mininet and iPerf have been used to simulate a variety of network aspects, including load balancers [8-12], routing [13], Quality-of-Services (QoS) schemes [14], and firewall designs [15]. This paper evaluates the measurement results obtained from a combination of iPerf and Mininet.

However, despite the reliance on Mininet and iPerf for many research results, little attention has been paid to how Mininet network results such as latency and throughput can be affected by a myriad of configuration and operating system factors. This paper shows that the latency and throughput of a simulated network can vary greatly depending on the simulator configuration and the operating system on which the simulator runs. In particular, we demonstrate that Mininet and iPerf can report invalid measurements for even the most basic topology consisting of two switches connected by a single link. Our most significant finding is that the sender may report a sustained throughput higher than the throughput available on the simulated link.

II. Related Work

Several studies have considered the performance of SDN tools. In terms of robustness, one study has used Mininet and iPerf to determine whether emulated SDNs are susceptible to Denial of Service (DoS) attacks [16]. Mininet has been used to demonstrate that SDNs can have less latency than traditional networking systems because packets can stay in the data plane without causing an exception or being sent to an external network controller [17]. One study used Mininet and OpenFlow to find a maximal network utilization that

Manuscript received October 25, 2022; accepted November 20, 2022.

Benjamin Hardin was with Purdue University, West Lafayette, IN 47907 USA. He is now with the Department of Computer Science, University of Oxford, Oxford, Oxfordshire OX1 3QG UK

Douglas Comer is with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA.

Adib Rastegarnia is publishing as an independent author.

Correspondence: hardin30@purdue.edu

doi: 10.18178/ijfcc.2023.12.1.596

minimized packet drop and understand how CPU load affected delay [18]. Other studies focused specifically on evaluating controllers running on top of Mininet [1, 19].

More specifically, the performance of Mininet on a resource constrained system has been evaluated in prior work [20]. In particular, the authors recommend that to keep the simulation results from being compromised, Mininet should be run on isolated CPU cores that are not simultaneously used for OS-related tasks. In addition, baseline measurements should be conducted to adjust for memory bandwidth variations on the system running the simulation. Ultimately the paper claims that despite small variations that can be caused by these factors, Mininet provides a robust platform to evaluate network performance without the need of physical hardware.

One study has compared Mininet to the OPNET network simulator and found that Mininet has difficulty accurately reporting delay and jitter compared to OPNET [21]. Additionally, Mininet was found to be unsuitable for testing network controller placement due to a lack of a physical port between the network controller and a switch.

These prior works conclude that Mininet is a robust platform to test network architectures. However, we have experienced throughput anomalies that no prior work reports. This paper investigates these unexpected scenarios, documents them, and provides suggestions for their avoidance.

III. Experiment Setup

Our experiment evaluated two simulated network architectures. The experiments were performed on a Purdue University server that had the following specifications: 8x32 GiB of RAM, 2 x Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz 20core (80 threads) running Ubuntu 18.04 LTS OS, version 2.6.0 of the ONOS SDN controller [22], Mininet version 2.2.2 [23], and iPerf version 3.1.3 [24]. Each experiment trial was run 30 times on the two network architectures, and the means of the experiments were reported below.

A. Leaf-Spine Data Center Architecture (Folded Clos)

Our work began by assessing a complex topology taken from cloud datacenters. The network consists of two datacenters, each with a folded Clos topology. Each datacenter contains 3 pods, 6 leaf switches, and 27 hosts. Except where otherwise specified, the datacenters were connected with a single 1 Gbps link with a delay of 50 ms. The delay was selected to simulate a real-world scenario where the datacenters are geographically separated (e.g., on separate continents). Throughout this paper, this architecture will be referenced as the "complex" topology. A diagram of this architecture is presented in Appendix A.

B. Simple Architecture

When results from the complex datacenter topology did not seem reasonable, we postulated that the large number of concurrent processes might affect Mininet. However, we found that anomalous behavior also occurred on the simplest possible topology: two switches connected by a single network link, each with one host attached to each switch. The hosts ran iPerf, and all traffic passed across the simulated link connecting the switches, which is configured with a 50 ms delay to simulate geographic separation between the switches. We speculated that the simplified scenario would allow the hosts to communicate at almost the full data rate of the simulated link (1 Gbps) because no other sources of traffic would exist. Additionally, we speculated that this would provide simpler routing and avoid any variations that could arise from a route intermittently changing based on network utilization. A diagram of this architecture is presented in Appendix B.

C. iPerf Configuration

To measure throughput between sending and receiving hosts, we invoked iPerf using the flags specified in Table I. The argument -M 9216 specifies the use of jumbo TCP packets, as commonly used on modern, high-capacity networks. Jumbo packets help reduce unnecessary overhead on the simulator by reducing the number of packets that must be processed [25, [26]. A following section discusses the choice of window size, as specified by the -w flag. Experiments use the Ubuntu default CUBIC TCP congestion control algorithm.

TABLE I: Sender iPerf Flags

Flag	Description
-M 9216	Use jumbo TCP packets
-V	Verbose output
-w *Window Size*	Set TCP receive window size
-t 10	Use 10 second trial
-p 5250	Use port 5250
-c *Receiver IP Address*	Specifies receiver
-logfile *Log File Location*	Where to log output

IV. Results

A. Unexpected Throughput

Our most significant finding is that when measured on the sending size, the combination of Mininet and iPerf can report throughput values that lie outside the possible range. In particular, the sender may report a sustained throughput higher than the throughput available on the simulated link. For the first second of the iPerf test, the sender may report a throughput that is several times higher than the link's capacity. The anomalous throughput values also occur when the capacity of the link is decreased. However, the average reported value does scale linearly with the link capacity. Ultimately, we found that these unexpected throughput values are dependent on the choice of TCP window size.

1) Reported goodput as a function of TCP window size

When the TCP receive window size is low, initial throughput on a connection will be limited because a sender must wait for an acknowledgment before sending more data. To ensure that window size was not a limiting factor in simulations, we increased the receiver's window size to 30 MB. Using a window size of 30 MB, which is near the maximum allowed on our system, revealed a strange phenomenon. As Fig. 1 shown, the initial reported throughput on the link exceeds the simulated network capacity. In our

testing, we were able to exceed the expected throughput by up to 422%, measuring a peak throughput of 3.98 Gbps. In short, with a sufficiently large receive window size, reported initial throughput soars well above the link's capacity. To assess the effect of window size, this trial measured throughput between two hosts connected to the same switch. Each link is configured with a 1 ms delay for a total delay of 2 ms in a single direction (i.e., a round trip delay of 4 ms).

We also measured sending across a link with a 50 ms delay between the two datacenters to simulate communication between two geographically separated hosts. This gives a roundtrip time (RTT) of 116 ms for the complex architecture and 108 ms for the simple architecture. With the complex topology, we experience 3.25 Gbps initial throughput and 940 Mbps average throughput for the remainder of the trial. The first second of the measured time exhibits a lower spike than the previous cross-switch test's spike (3.98 Gbps) because the higher link delay (50 ms) causes TCP to wait for acknowledgements, limiting the throughput. Although the mean throughput of 940 Mbps appears to be less than the link capacity, the results are misleading. iPerf reports goodput (actual data transferred) which does not include packet header overhead. Thus, a valid simulation must report a goodput of less than 1 Gbps. When packet headers are added to the goodput values reported by iPerf, the total throughput exceeds the link capacity of 1 Gbps.

Since the simulated link capacity is unable to reliably limit throughput, we wondered if TCP receive window size would be able to constrain the throughput. Our results indicate that receive window size is accurately able to constrain throughput. We ran a test with a window size of 2 MB across a 1 Gbps link with a delay of 50 ms, simulating a link between datacenters. The goodput reported during the first tenth of a second is 209 Mbps, and the average goodput for the remaining seconds is 132 Mbps, much lower than near-gigabit link capacity. Fig. 1 shows that initial throughput increases with increasing sizes of the receiver window. Because window size is able to effectively limit excessive throughput, we can use it as a method for ensuring results fall within an acceptable range.

2) Measuring throughput on lower link capacity

So far, we have learned that window size affects throughput, but link capacity should have also constrained throughput. To understand the relationship between window size, link capacity, and reported throughput, we repeated the previous tests using a 10 Mbps link. Our results indicate that lowering link capacity lowers average throughput but still demonstrates above-capacity throughput both initially and at points throughout the trial. The initial throughput spike on the sending side that we experienced previously still exists with the 10 Mbps link. With a 30 MB window size, the initial spike is reported at 3.24 Gbps. However, the average goodput during the test does scale to the link capacity. The goodput spikes throughout the test are 105 Mbps with one second gaps between them of 0 Mbps throughput. This leads to average second-by-second reported statistics of 10.5 Mbps, which is above the link capacity, especially because this is a measurement of goodput rather than total throughput.

Meanwhile, the receiver reports a maximum goodput of 9.75 Mbps and minimum goodput of 9.08 Mbps for the trial. This discrepancy between sender and receiver statistics shows that the sender does not report an accurate measure of the data traveling across the network. Packet loss is not responsible for the discrepancy between sender and receiver because there is negligible packet loss. Note that the test used the simplistic architecture with only one link, meaning that even in the simplest scenario with a low-speed link, the sender reports unreasonable results.

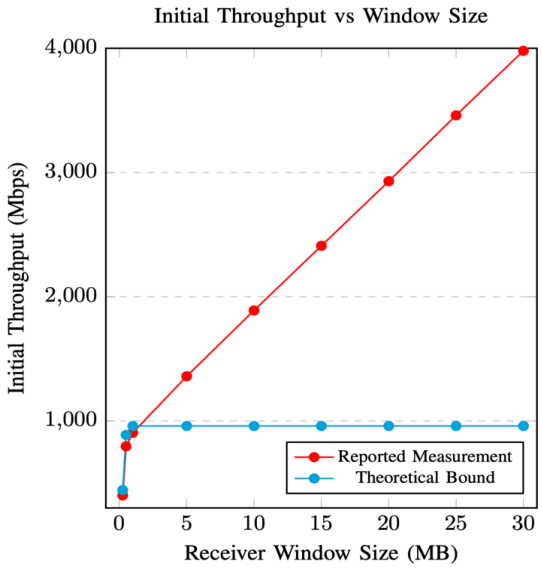


Fig. 1. Sender report of initial throughput soaring above the link capacity (complex data center setup).

3) Understanding these results

Our iPerf setup used a single stream TCP connection to ensure that these high throughput values were not occurring because of multiple TCP streams utilizing multiple links of the datacenter. To explain why iPerf reports excessive throughput when the receiving window is large, we hypothesized that the simulation might be splitting the single TCP stream across multiple links, causing reported measurements to exceed the capacity of a single link. If the hypothesis is correct, a switch connected to the destination host might temporarily act like a buffer, holding packets until they can all can enter the host across a single connection. To evaluate the hypothesis, we tested transmission on a simplistic architecture with two switches connected by a single 1 Gbps link along the path from sender to receiver. Using a receive window size of 30 MB and an RTT of 108 ms resulted in an initial goodput of 3.25 Gbps. After the initial spike, the simulator reported goodput of 945 Mbps. We conclude that because we observe excessive throughput with only a single link in the path, the excess cannot be attributed to data being split across multiple links.

When the link between the two switches is configured to have a latency of 1 ms, iPerf reports an even higher initial goodput spike of 3.56 Gbps and a steady goodput of 997 Mbps, higher than what should be allowed on the link. Ultimately, even in the simplest case, the combination of Mininet and iPerf produces unreasonably high throughput of more than 3.5 times the emulated link capacity.

Why The Sender and Receiver Report Differing Throughput: As described above, the sender and receiver report differing throughput, with the receiver reporting more accurate values especially with large window sizes. The iPerf

source code [27] explains the discrepancy: the sender does not have a global view of the system, but instead only reports data that has been injected into the network. In particular, the sender does not wait for data to be received and acknowledged.

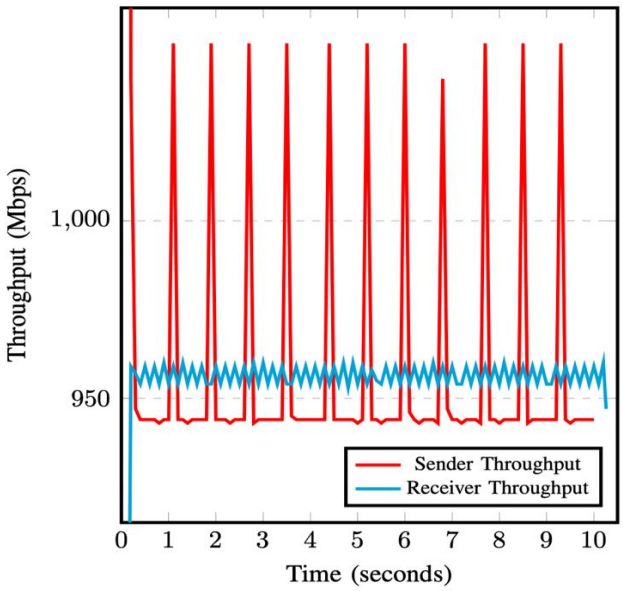


Fig. 2. Sender and receiver reports of throughput averaged over one second intervals; note the exaggerated spikes in sender reports.

Fig. 2 illustrates the difference between sender and receiver reports during a 10 second trial. The figure shows measurements on the simplistic topology with all traffic traversing a single 1 Gbps link between the endpoints and a round-trip latency of 4 ms. The scaling of the graph omits the goodput during the first tenth of a second. These values are 3.98 Gbps and 758 Mbps for the sender and receiver respectively. In addition, the sender reports spikes in throughput during the test, which do not correlate to spikes on the receiving end. The spikes result in unrealistically high average throughput values compared to the values reported on the receiving side: 997 Mbps vs. 955 Mbps.

4) Recommendations for measurements

Even on the receiving side, early reports of average throughput do not provide an accurate estimate of the overall sustained throughput. In particular, throughput may be lower during the first-time interval of a simulation because it takes time for the sending side to propagate packets into the network. Thus, when performing throughput measurements, it may be best to discard values reported during the first second (or more depending on latency) of a simulation. In addition, although the receiving side appears to report fairly accurate measurements when the window size is high, using a lower window size is recommended because artificially large window sizes can cause other issues, as described in later sections of this paper.

These discrepancies between sender and receiver demonstrate the need to configure iPerf to report statistics over short intervals instead of allowing iPerf to average values over an entire simulation run. Consider the case of a host sending to another host on the same switch with a TCP receiving window size of 30 MB. When it computes an average over the entire trial, iPerf reports an average throughput of 987 Mbps. However, when the report interval is set to 5 seconds, iPerf generates two values: averages of 1.02 Gbps and 956 Mbps. Using two reports shows a large discrepancy that is concealed in a single report. Finer granularity reveals additional detail. For example, requesting a report every second demonstrates that the average throughput on the first second is 1.26 Gbps with successive seconds averaging around 955 Mbps. Requesting a report every tenth of a second reveals an initial spike of 3.89 Gbps and subsequent spikes 1.05 Gbps throughout the test; these spikes remain hidden with larger granularity. Therefore, using a single average as a measure of throughput can inflate throughput inaccurately, hide important details, and lead to an inaccurate assessment.

5) Why does window affect the simulated THROUGHPUT?

We have been unable to determine exactly why a large window size results in iPerf reporting excessive throughput. This problem occurs with both iPerf2 and iPerf3. One possibility lies in an equation that relates the TCP receive window size, the round-trip time, and the underlying link capacity (i.e., throughput). According to the equation:

$$TCP\ Receive\ Window\ Size = Bandwidth * RTT$$

where RTT is round trip time measured in seconds and bandwidth is the maximum throughput capacity of the link measured in bits per second. Rearranging terms results in an equation that specifies the maximum throughput in terms of the window size:

$$Throughput = TCP\ Receive\ Window\ Size / RTT \quad (1)$$

Recall that for our first scenario, iPerf reported up to 3.98 Gbps of initial throughput. Using the values configured for the simulator, the equation gives the following theoretical maximum throughput:

$$\begin{aligned} Throughput &= 30{,}000{,}000\ bytes / 0.116\ seconds \\ &= 258{,}620{,}689.655\ bytes / second \\ &= 2{,}068{,}965{,}517.24\ bits/second \\ &= 2.069\ Gbps \end{aligned}$$

The maximum throughput reported by iPerf is higher than theoretical maximum specified by the formula, meaning this formula does not explain the results. However, using end-to-end delay or 1/2 of RTT gives a throughput of 4.138 Gbps, which is just above our reported goodput. It is possible that iPerf is using single direction delay in the calculation rather than RTT. Note that the formula only specifies an upper-bound on throughput for a given window size and round-trip time; it does not include a constraint for the physical link capacity. If the window size is sufficiently large, the maximum throughput can be greater than the simulated network capacity.

It is clear that Mininet does not use the link capacity as an upper limit when calculating the rate at which data can be sent over a link. The layers of code make it difficult to find the exact source of the problem. It seems likely that the Mininet uses the formula above to calculate the maximum allowed rate and then uses the calculated rate to deposit outgoing data into an output buffer, leading iPerf to assume the data has been sent and to calculate the data rate accordingly. Once the simulation starts delivering packets and the receive window size shrinks, the available receive

window limits the flow of data, causing iPerf to report data rates closer to the link capacity. The rate at which the receive-side window absorbs data limits the rate at which packets travel across the link.

As further support for our hypothesis, we observe that link delay seems to impose a bound-on throughput that cannot be overcome merely by increasing the receive window size. Thus, it seems likely that Mininet uses the link delay to determine how quickly the receiver's window will fill and limits traffic transmission accordingly. Consequently, a high link delay causes the data rate to slow, regardless of the emulated link capacity.

Congestion control algorithms can limit the flow rate of packets onto a network, and the question arises whether the congestion control algorithm might also affect reported throughput. Recall that our tests use the default TCP congestion control algorithm in Ubuntu, known as *CUBIC*. To check the effect of congestion control, we reran tests with the TCP Reno congestion control algorithm. The change in congestion control made no difference in results, leading us to conclude that the choice of congestion control algorithm does not contribute to the observed behavior.

B. *Unequal Link Sharing*

The experiments thus far have considered only a single sender. In most situations, however, multiple senders share a given network. Therefore, it is important to understand how Mininet and iPerf behave when multiple flows share a link. All flows on a network should be balanced according the policy being used. Unless a specific policy is configured, network switches employ statistical multiplexing, which gives each flow an equal share of the capacity. When n flows share a link, each flow should receive 1/n of the link capacity.

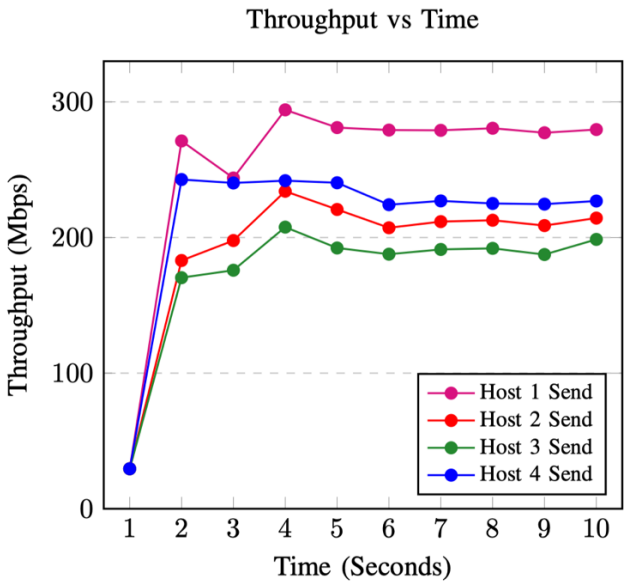


Fig. 3. Folded Clos Architecture: Unequal link sharing on a 1 Gbps link with a 30 MB receiving window size.

The question arises: how does the size of the receive-side TCP window affect throughput when multiple flows share a link? This section reports experiments that show how a large TCP receive-side window can cause unequal link sharing among flows.

The first link-sharing experiments use a folded Clos architecture with two datacenters. Four senders in one datacenter send to four receivers in a second datacenter. Fig. 3 illustrates that with a window size of 30 MB, the link capacity between the datacenters is not shared equally among the four senders. The largest difference among throughput on the four flows is 73%. Interestingly, a dominant flow emerges and remains dominant throughout the simulation. Furthermore, the pattern of flow dominance is repeatable, but it is difficult to predict which flows will dominate in a given run.

We hypothesized that the order in which senders start would determine how flows dominate. However, tests show that the order in which senders start does not guarantee which flows dominate. In particular, the first host to start sending does not gain an advantage by sending over an empty network or by using the network the longest. In fact, although they start sequentially, the iPerf tests all start within a few milliseconds of each other. Thus, no sender-receiver pair has significant time to gain an advantage by transmitting at full link capacity without other traffic.

We hypothesized that a large window size allowed senders to flood the network with packets. In particular, a window size of 30 MB is sufficient to allow any of the four flows to use the entire link capacity. To understand whether the window size affects sharing, we ran the same throughput experiments with a window size of 4 MB, a value small enough to allow each of the four flows to use 1/4 of the link capacity. As Fig. 4 illustrates, although differences in throughput persist, the gap between the highest and lowest is significantly smaller. The differences in throughput are small enough so that minor variations allow the plots from multiple flows to overlap. The maximum difference among throughput for the four flows is now only 15%. Table II outlines the window sizes used based on the number of flows sharing a single link. Note that the values depend on the topology, link speed, and packet round-trip time. The value of an optimal window size may be calculated using 1 in the previous section.

TABLE II: TCP WINDOW SIZE BASED ON NUMBER OF FLOWS

Number of Hosts Sharing Link	Window Size (MB)
1	15
2	7.5
3	5
4	4
5	4

One possible explanation of unequal throughput arises because the leaf-spine architecture provides multiple paths along which data can travel. If the system does not distribute traffic from multiple flows across paths equally, congestion could occur on some paths and not others. To rule out this possibility, we also ran multi-flow simulations using the simplistic topology with only one path between two switches. Four hosts were added to each switch, allowing for a total of four simultaneous flows. We used a TCP receive window size of 30 MB, and measured the throughput on each flow. Fig. 5 illustrates that unequal sharing occurs even when the network consists of a single path. The maximum difference in throughput is 94%.

Furthermore, the figure shows that the throughput for each

flow exhibits high variability. Lowering the receive window size reduces the variability, just as it does for the complex topology. Thus, it is important to set an appropriate receive window size even in the simplest of scenarios.

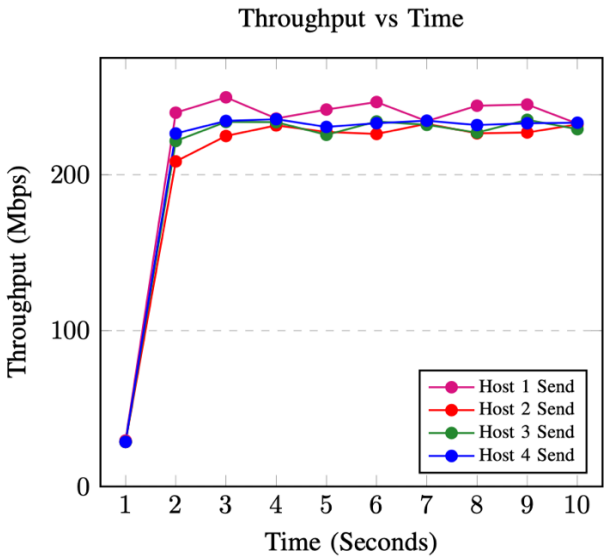


Fig. 4. Folded Clos Architecture: Unequal link sharing on a 1 Gbps link with a 4 MB receiving window size.

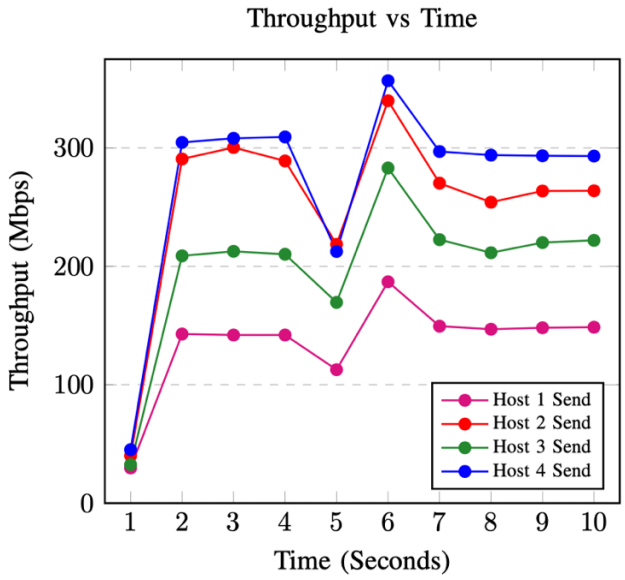


Fig. 5. Simple Architecture with unequal link sharing over a 1 Gbps link with a 30 MB receive window size.

C. Bidirectional Traffic and Link Sharing

Most networks support two-way traffic, and data flowing in the reverse direction may delay acknowledgements. With a full-duplex link, such as an Ethernet, we expect simultaneous traffic flows in the forward and reverse directions to each achieve throughput that approaches the link capacity. We conducted experiments to test the assumption, and found that when using Mininet and iPerf, simultaneous traffic in two directions does not approach the link capacity.

We used two methods to assess simultaneous traffic. First, we used the iPerf option for bidirectional traffic, where the host at each end acts as both a sender and receiver. As Fig. 6 illustrated, an iPerf bidirectional test does not allow throughput in either direction to reach the full link capacity. Moreover, the throughput of the two flows is unequal, with the two trading off for various window sizes. As the figure shows, when the window size is 10 MB, throughput in one direction falls below half of the link capacity. Smaller window sizes do not result in equal throughput unless the window size becomes small enough to limit throughput severely. The test was performed on the complex topology with a single host pair sending between two data centers across a link with a delay of 1 ms.

The graph in Fig. 6 shows data for a single trial, which illustrates how the throughput vary. Fig. 7 shows that averaging the throughput measurements over 30 trials produces a smoother, and more consistent pattern. The smoothing occurs because the two directions randomly become either the faster or slower direction for a given window size. In Figure 6, for example, the traffic sent by Host 1 dominates at each value of window size except for 20 MB. The important point is that even when averaged over 30 trials, Mininet only allows traffic in each direction to reach approximately 80% of the link capacity.

1) A second approach to bidirectional testing

The question arose: does the erratic performance and lower-than-expected average throughput result from the way iPerf performs bidirectional transfers? To discover whether iPerf causes the problem, we took a second approach of running two independent unidirectional iPerf tests simultaneously, one in each direction. One of the two uses TCP port 2050, and the other uses port 2051. The results are similar to the measurements obtained by using the built-in bidirectional command in iPerf. The throughput exhibits high variability, and one flow proceeds faster than the other. As reported for the iPerf bidirectional measurements, the host pair that has highest throughput varies randomly between trials, and does not depend on which pair of hosts starts first. Thus, we conclude that problems with bidirectional tests appear to arise from Mininet rather than the iPerf application.

2) Possible causes of bidirectional issues

We wondered whether the CPU of the system running Mininet and iPerf formed a bottleneck that led to lower performance. However, CPU utilization does not appear to be the culprit. Both the iPerf bidirectional tests and our custom solution use similar amounts of CPU, and both values are low. The overall processor utilization remains around 0.3% during the throughput measurements. With iPerf's bidirectional command, there are 2 iPerf instances (one for each direction), each instance incurred only 10% CPU utilization. With our custom bidirectional test, 4 iPerf instances run (two for each direction), each incurred 35% CPU utilization. Thus, both approaches incur similar CPU utilization overall, neither imposes a heavy load on the CPU.

We also wondered if the emulated switches or hosts that Mininet uses have small internal buffers that limited total throughput. In particular, we speculated that a small buffer might not allow an emulated device to fill a Gigabit link. To test whether the emulated devices could fill a link, we lowered the link speed from 1 Gbps to 10 Mbps and reran the bidirectional tests. Even with a lower link speed, the same problem arose: when sending bidirectional flows across a 10 Mbps link, each flow exhibited an average throughput of approximately 7 Mbps. The tests used the same packet sizes as the 1 Gbps trials, which means the packet processing time decreases because the number of packets decreases. We concluded that the limited throughput does not result from a 1 Gbps link overwhelming the emulated devices or filling their

input buffers. Some other factors must result in a consistent average link utilization of 80% or less.

As a final question, we asked whether the emulated link or emulated hosts might be the bottleneck. To test link performance, we arranged a bidirectional test with four hosts instead of two hosts. Hosts 1 and 3 were placed on one side of a link, and hosts 2 and 4 were placed on the other side. Host 1 sent a unidirectional flow to host 2 while host 4 sent a unidirectional flow to host 3 in the reverse direction. In essence, the only change from the previous bidirectional tests was the use of four hosts instead of two. Interestingly, we find that both flows achieve throughput of the full link capacity. Furthermore, tests show low variability. Thus, we can conclude that the issue with bidirectional tests arises from the emulated hosts and not from the emulated link.

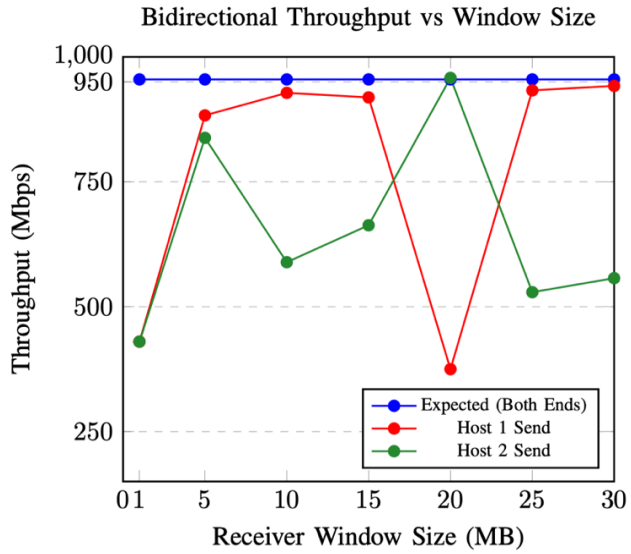


Fig. 6. Throughput using iPerf bidirectional tests of various window sizes across a link with a 1 ms latency.

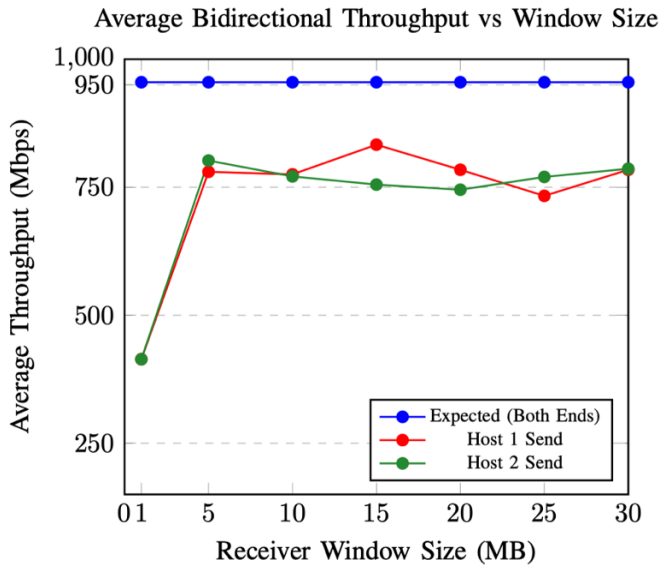


Fig. 7. Average throughput for 30 iPerf bidirectional tests of various window sizes across a link with a 1 ms latency.

V. Future Work

Several avenues of work can arise from these results. First, the community should investigate the underlying cause of anomalous simulation results and modify Mininet to eliminate the problem. It cannot be expected for every research to configure their simulation precisely, and robustness of the platform is the best solution. Second, our work focused on one version of Mininet and Linux; future work should investigate whether the same issues arise on other operating systems, including MacOS, Windows, and other versions of Linux. Third, it should be investigated whether our findings can be replicated when iPerf is used across a network simulation/emulation system other than Mininet. In limited testing, this is not an issue that occurs when using iPerf outside of the Mininet application. Finally, a logical next step is comparing these results to those obtained on a physical testbed. This comparison would quantify the errors we experience in this paper compared to real measurements rather than theoretical values.

VI. Conclusions

The flexibility and relatively low hardware requirements of Mininet and iPerf mean that they remain a popular combination when developing and testing new network architectures and applications of SDN technology [1–5]. Despite their popularity, we have uncovered evidence that Mininet and iPerf can report misleading and invalid measurements. In particular, throughput reported by iPerf depends on the TCP receive-side window, and a large window size can result in reported throughput that exceeds the emulated network capacity. During small time periods near the beginning of a simulation, the reported throughput can exceed the link capacity by over 422%.

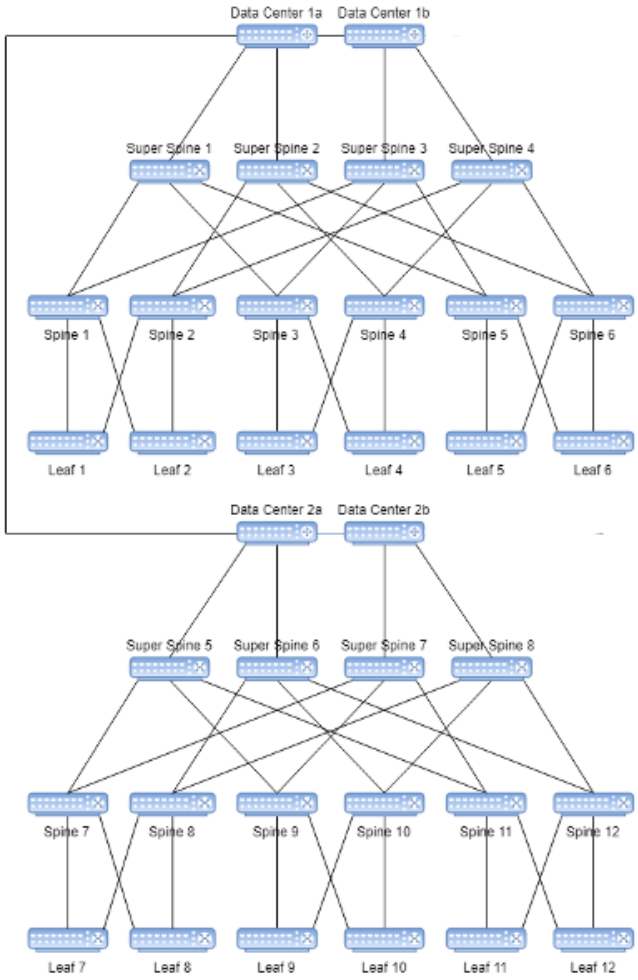
As a short-term solution, we provided three recommendations for researchers using Mininet and iPerf:

- Use iPerf window size flags to limit throughput to match link capacity.
- Measure iPerf results over small intervals to monitor throughput variation.
- Choose an appropriate window size given the number of flows sharing the link to avoid unequal link sharing.
- Avoid bidirectional iPerf tests and instead use independent pairs of hosts to fill the link.

Our findings have significant implications for researchers and others who use Mininet and iPerf to assess network architectures and network performance. Minor configuration changes can result in significantly invalid output. In particular, we demonstrated that Mininet and iPerf can report invalid measurements for even our simple topology consisting of two switches connected by a single link. We also found that the sender may report a sustained throughput higher than the simulated link capacity. Therefore, it is important to follow our configuration guidelines above to avoid misleading results. Previous work that uses Mininet and iPerf should be reevaluated in light of our findings to determine whether the results remain valid when appropriate configuration parameters are used. Future research utilizing these tools should pay close attention to the reported values to ensure results that align with physical hardware. We have demonstrated that emulations can, in fact, fail and researchers must carefully evaluate the quality of output to ensure results are logical.

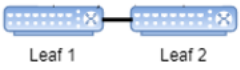
VII. Appendix

A. Leaf-Spine Datacenter Architecture (Folded Clos)



Complex Scenario: Two datacenters connected with a high-speed link

B. Simple Architecture



Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

BH conducted the experiments and wrote the paper, DC guided the research and helped significantly with paper edits, and AR served as a technical consultant.

Acknowledgment

We would like to thank Assistant Professor Pedro Fonseca of Purdue University for his advice and discussions that helped guide this research.

References

[1] D. Lunagariya and B. Goswami, "A comparative performance analysis of stellar sdn controllers using emulators," in *Proc. 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, 2021, pp. 1–9.

[2] S. Kunal, P. Gandhi, R. Sutariya, and H. Tarpara, "A secure software defined networking for distributed environment," *Security and Privacy*, vol. 3, no. 6, 2020.

[3] U. Noring, "Investigating the possibility of speeding up mininet by using netmap, an alternative linux packet i/o framework," *Procedia Computer Science*, vol. 126, pp. 1885–1894, 2018.

[4] T. Abdullah, "Testing of floodlight controller with mininet in sdn topology," *ScienceRise*, vol. 5, no. 2, pp. 68–73, 2014.

[5] S. Shamim, M. Miah, A. Sarker, A. Bahar, and A. Sarker, "Simulation of minimum path estimation in software defined networking using mininet emulator," *British Journal of Mathematics and Computer Science*, vol. 21, no. 3, pp. 1–8, 2017.

[6] R. Dos Reis Fontes, C. Campolo, C. E. Rothenberg, and A. Molinaro, "From theory to experimental evaluation: Resource management in software-defined vehicular networks," *IEEE Access*, vol. 5, 2017.

[7] C. Campolol, R. D. R. Fontes, A. Molinaro, C. E. Rothenberg, and A. Iera, "Slicing on the road: Enabling the automotive vertical through 5g network softwarization," *Sensors (Basel, Switzerland)*, vol. 18, no. 12, pp. 4435–, 2018.

[8] Y. A. H. Omer, A. B. A. Mustafa, and A. G. Abdalla, "Performance analysis of round robin load balancing in sdn," in P*roc. 2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, 2021, pp. 1–5.

[9] I. Z. Bholebawa, R. K. Jha, and U. D. Dalal, "Performance analysis of proposed openflow-based network architecture using mininet," *Wireless Personal Communications*, vol. 86, no. 2, pp. 943–958, 2016.

[10] V. Kumar, S. Jangir, and D. G. Patanvariya, "Traffic load balancing in sdn using round-robin and dijkstra based methodology," in *Proc. 2022 International Conference for Advancement in Technology (ICONAT)*, 2022, pp. 1–4.

[11] M. Z. Ibrahim and R. Hassan, "The implementation of internet of things using test bed in the ukmnet environment," *Asia-Pacific Journal of Information Technology Multimedia*, vol. 8, no. 2, pp. 1–17, 2019.

[12] V. D. Chakravarthy and B. Amutha, "A novel software-defined networking approach for load balancing in data center networks," *International Journal of Communication Systems*, vol. 35, no. 2, 2022.

[13] O. Fares, A. Dandoush, and N. Aitsaadi, "Sdn-based platform enabling intelligent routing within transit autonomous system networks," in *Proc. 2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, 2022, pp. 909–912.

[14] R. Carreras Ramirez, Q. T. Vien, R. Trestian, L. Mostarda, and P. Shah, "Multi-path routing for mission critical applications in software-defined networks," in *Proc. Industrial Networks and Intelligent Systems, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 2019, pp. 38–48.

[15] A. O. Adedayo and B. Twala, "Testing the functionality of firewall in software-defined networking," in *Proc. Artificial Intelligence and Evolutionary Computations in Engineering Systems, ser. Advances in Intelligent Systems and Computing*, 2018, pp. 1–14.

[16] A. F. Abdullah, F. M. Salem, A. Tammam, and M. H. A. Azeem, "Performance analysis and evaluation of software defined networking controllers against denial-of-service attacks," *Journal of Physics*, vol. 1447, no. 1, 2020.

[17] C. N. Shivayogimath and N. V. U. Reddy, "Performance analysis of a software defined network using mininet," *Artificial Intelligence and Evolutionary Computations in Engineering Systems, ser. Advances in Intelligent Systems and Computing*, 2016, pp. 391–398.

[18] H. M. Noman and M. N. Jasim, "Pox controller and open flow performance evaluation in software defined networks (sdn) using mininet emulator," in *Proc. IOP conference series. Materials Science and Engineering*, vol. 881, no. 1. 2020.

[19] N. M. Kazi, S. R. Suralkar, and U. S. Bhadade, "Evaluating the performance of pox and ryu sdn controllers using mininet," *Data Science and Computational Intelligence, ser. Communications in Computer and Information Science*, 2022, pp. 181–191.

[20] D. Muelas, J. Ramos, and J. E. L. D. Vergara, "Assessing the limits of mininet-based environments for network experimentation," *IEEE Network*, vol. 32, no. 6, pp. 168–176, 2018.

[21] S. Lee, J. Ali, and B. hee Roh, "Performance comparison of software defined networking simulators for tactical network: Mininet vs. opnet," in *Proc. the Institute of Electrical and Electronics Engineers*, 2019.

[22] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos: Towards an open, distributed sdn os," in *Proc. the Third Workshop on Hot Topics in Software Defined Networking*, 2014, p. 1–6.

[23] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.
[24] Iperf — The ultimate speed test tool for tcp, udp and sctp. [Online]. Available: https://iperf.fr/
[25] Y. Wu, S. Kumar, and S. J. Park, "On transport protocol performance measurement over 10gbps high speed optical networks," in *Proc. 2009 Proceedings of 18th International Conference on Computer Communications and Networks*, 2009, pp. 1–6.
[26] A. Das and S. Debbarma, "Performance of jumbo sized data on jumbo frame and ethernet frame using udp over ipv4/ipv6," in *Proc. 2013 2nd International Conference on Advanced Computing*, 2013, pp. 204–207.
[27] ESnet. iperf3: A tcp, udp, and sctp network bandwidth measurement tool. [Online]. Available: https://github.com/esnet/iperf

Benjamin Hardin received his bachelor's in computer science honors from Purdue University in 2022. During his studies at Purdue, he focused on systems programming and software engineering, leading him to research emulations and protocols of software-defined networking. In October 2022, Benjamin began his PhD in computer science at the University of Oxford studying explainable and trustworthy AI interfaces for autonomous vehicles.

Douglas E. Comer is a distinguished professor of CS and ECE (courtesy) at Purdue University. He formerly served as the inaugural VP of Research at Cisco Systems. He is internationally recognized as an authority on TCP/IP Internet protocols, and his 3-volume Internetworking series is cited as an authoritative work on Internet protocols and the scientific principles that underlie the Internet. Comer's books have been translated into 16 languages, and are used in industry and academia in many countries. A former editor-in-chief of the journal Software – Practice and Experience, he is a Fellow of the ACM, the recipient of numerous teaching awards, and a member of the Internet Hall of Fame.

Adib Rastegarnia received his masters and the PhD in computer science from Purdue University in 2019 under the supervision of Prof. Douglas Comer. During his studies at Purdue, he focused on different aspects of computer networks and systems including control plane disaggregation in software defined networking (SDN), internet of things, and operating systems. As the results of his work at Purdue, he designed and implemented multiple working prototypes of large and complex systems and published publications in well known conferences and journals. In 2020, Adib Joined Open Networking Foundation (ONF) to pursue his research interests in a broader context. He joined the micro-onos project team at ONF to design and implement a cloud native architecture for disaggregating control plane services and functions. In addition, he has contributed on designing and implementing ONF SD-RAN project subsystems including a near-real-time RAN Intelligent Controller (RIC) and xApps for controlling RAN. In 2022, Adib joined Intel as a Cloud Software Development Engineer. Currently at Intel, Adib is working on micro-onos and Intel SDN related projects.