Mininet /Netem Emulation Pitfalls
A Multipath TCP Scheduling Experience

Alexander Frömmgen

November 17, 2017

Abstract

Do you use Mininet for network experiments? This article summarizes our experience with notable differences between Mininet and real-world measurements. During our work on Multipath TCP scheduling, we experienced systematic and reproducible differences between real-world measurements and Mininet experiments. Our investigation shows that these differences are caused by the interplay of Mininet, netem and the queue management in the networking stack, i.e., TSQ.

1 Introduction

Multipath TCP [3] is a recent TCP evolution which enables the usage of multiple paths for a single logical Multipath TCP connection. During our work on flexible Multipath TCP scheduling [4], we used Mininet [5] and the Linux Kernel Multipath TCP implementation [1] for development and systematic evaluations. We chose Mininet, as it is convenient to run a real network stack in user defined topologies. Mininet is widely used in current efforts to reproduce network research results [1]. We were surprised when our Mininet experiments and real-world experiments showed significantly different behavior [2].

2 A Simple Example

For a basic MPTCP measurement example, let’s consider the following topology with two paths (Figure 1). The first path has a round-trip time of 40ms, the second path has a round-trip time of 80ms. The topology in Figure 1 can be easily specified with the Mininet Python API as shown in Listing 1.

Now, let’s run an application on top of this topology. Our example application starts with a short handshake and eventually sends 10kb of data. At this point, Multipath TCP managed to establish one subflow per path, so we have two subflows available. In a typical Internet setup, 10kb of data require roughly 8 packets. Thus, MPTCP has to schedule 8 packets on two subflows, and both subflows have an initial congestion window of 10 packets [2].

1 https://reproducingnetworkresearch.wordpress.com
2 This experience report is available online at https://progmp.net/mininetPitfalls.html
1

Figure 1: Basic example Mininet topology with two paths.

Listing 1: Python code snippet for the topology in Figure 1.

```python
class StaticTopo(Topo):
    def build(self):
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')

        /* first path */
        s1 = self.addSwitch('s1')
        self.addLink(h1, s1, bw=100, delay="10ms")
        self.addLink(h2, s1, bw=100, delay="10ms")

        /* second path */
        s2 = self.addSwitch('s2')
        self.addLink(h1, s2, bw=100, delay="20ms")
        self.addLink(h2, s2, bw=100, delay="20ms")
```

Figure 2: Sending queue with 8 packets in the Mininet example topology with two paths.

2.1 Mininet Evaluation

Using Mininet and our example application, we found that the default Multipath TCP scheduler schedules all 8 packets on the lowest round-trip time subflow (Figure 3). We used wireshark to analyze the sent packets (Figure 4).

\[3\] The source of the evaluation is available at https://progmp.net/mininet_tsq.zip
Figure 3: In the Mininet setup, all 8 packets are scheduled on the first subflow.

Figure 4: Wireshark trace for the Mininet example of Figure 3. Note that all packets are scheduled on the first path (IP 11.0.0.2).

2.2 Real-World Evaluation

When we used the same application in a comparable real-world setup, we experienced different behavior. In the real-world experiments, packets were distributed over multiple paths, although the round-trip times showed the same differences as in our emulation setup (Figure 5).

We repeated the experiment multiple times, showing reproducible results. So why do our Mininet experiments and the real-world measurements show significantly different behavior?

3 Explanation

Digging into the Multipath TCP implementation, we found that the scheduler stops using the first path due to TCP small queues (TSQ)\(^4\). TSQ is an effort to reduce queue sizes in the network stack. Large queues in the network stack are known to increase latency due to queuing delays. During the development of Multipath TCP, a commit was added to avoid sending packets on subflows

---

\(^4\)https://lwn.net/Articles/507065
Figure 5: In the real-world environment, packets are split on both subflow.

which are considered throttled by TSQ (Listing 2) For many application scenarios, this is a reasonable design decision for Multipath TCP.

Listing 2: The Multipath TCP default scheduler checks if a subflow is already throttled by TSQ.

```c
if (test_bit(TSQ_THROTTLED, &tp->tsq_flags))
    return 0;
```

But why don’t we see this behavior in our Mininet emulation? This requires a more detailed analysis of the interactions of Mininet, TSQ and Multipath TCP. Mininet relies on netem for delay emulation. Checking the implementation of netem we found that packets which are delayed by netem are not considered queued by TSQ (Listing 3).

Listing 3: The netem implementation removes delayed packets from the sending queue.

```c
if (q->latency || q->jitter || q->rate)
    skb_orphan_partial(skb);
```

Thus, Multipath TCP never experiences a TSQ throttled subflow in Mininet setups with delayed links at the sender. To substantiate our claim, we repeated
the previous Mininet emulation with a slightly modified topology (Listing 4). In this topology, the first link at the sender does not use netem.

Listing 4: Python code snippet for a modified topology which has no netem delay on the sender's link.

```python
class StaticTopo(Topo):
    def build(self):
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')

        # first path
        s1 = self.addSwitch('s1')
        self.addLink(h1, s1, bw=100)
        self.addLink(h2, s1, bw=100, delay="20ms")

        # second path
        s2 = self.addSwitch('s2')
        self.addLink(h1, s2, bw=100)
        self.addLink(h2, s2, bw=100, delay="40ms")
```

The wireshark trace in Figure 6 shows the expected behavior, which is comparable to our real-world measurements.

![Wireshark trace](image)

Figure 6: Mininet experiment without netem on the sender's links. This Mininet experiment shows the same behavior as our real world measurements, i.e., packets are send from both source IP addresses 11.0.0.2 and 10.0.0.2.

4 Conclusion

Our experiments show that we have to be careful when transferring emulation results to real-world applications. Evaluation results should always be (at least) supported by real-world measurements. We are still convinced that Mininet is great for network experiments and reproducible network research, but we feel confirmed that sanity checks are essential.

Did we misuse Mininet? As our results show that our initial Mininet setup does not reproduce real-world situations, we probably used it wrongly. However, we just followed established online examples here, so chances are high that others will fall into the same pitfalls as well. Note that this problem is not specific to Mininet, but might appear in comparable netem setups as well.
Are there more pitfalls? Yes. We are preparing some more posts. If you struggled with Mininet pitfalls, feel free to contact us, we might prepare a collection :-) 

Talking about MPTCP. How can I get the behavior MPTCP showed in Mininet in the real-world? There are a lot of options. You might change the queue sizes considered by TSQ or change the MPTCP scheduler, e.g., using ProgMP.

5 Acknowledgments

This work has been funded by the German Research Foundation (DFG) as part of the project C2 in the Collaborative Research Center (SFB) 1053 MAKI. We would like to thank the Mininet Mailing list replier who supported our analysis.

References


https://progmp.net
https://mailman.stanford.edu/pipermail/mininet-discuss/2017-April/007425.html