Project 3: Mininet

Colin Sanders

CS 538

# OVERVIEW

Simulating real environments computer in networking is difficult given the complexity of real networks. One tool that emulates networks in called Mininet. Mininet creates a network of virtual hosts, switches, controllers, and links on which real network mechanics can be simulated on complete experimental networks. Mininet also runs standard Linux networking software and its virtual switches support OpenFlow for custom routing and Software-Defined Networking, just as real switches would in a network. It is significantly cheaper and more configurable than networked hardware testbeds, runs faster and scales larger than virtualization approaches, and can easily connect to real networks, making Mininet one of the best tools for supporting network-related research, prototyping, and development. [1]

# MININET INTRODUCTION

The first part of the project was to install Mininet and perform some basic functions on a simple network topology example. There are two potential ways to install Mininet – either by installing from source or running a pre-built Linux virtual machine. The version chosen was the most recent Ubuntu virtual machine with Mininet preinstalled, ran on VMWare Workstation Pro virtualization software. Also installed per the recommendations of the Mininet website is the lxde graphical interface toolkit, which ended up being very useful for having individual terminals open for different hosts, along with viewing the results of packets sent through a network using the Wireshark software suite.

After installing Mininet, a basic topology was created by running the “mn” command with no arguments. This created a simple network of two hosts and one switch, along with a controller for the switch. The command “net” shows this network topology printed to the command line. Additionally, in preparation for the later experiments, the TCP Vegas protocol had to be loaded using the “modprobe” command.

# TCP VEGAS VS. RENO

The TCP transport protocol has many “flavors” that provide mechanisms for managing congestion in a network, both to benefit the throughput of the connection and to improve the throughput of other connections in a network. TCP Reno is the main congestion-management protocol that is used on the Internet today, while research is going on to develop newer and better approaches to managing congestion, such as TCP Vegas. Before diving into a competition comparing TCP Reno and Vegas, the mechanisms of each will be described and contrasted.

A core mechanism of reducing congestion in a TCP connection is to change the window size – that is, the number of packets being delivered at one time. All TCP congestion protocols reduce the window size when congestion occurs and decrease it when the congestion is alleviated. Congestion can be identified at different parts of the packet life cycle. One such time is when the number of packets in a queue starts to grow (sometimes known as a “knee” strategy), while another is once a packet is dropped from a queue (known as a “cliff” strategy). It is the goal of individual TCP connections to monitor congestion in the Internet and reduce their own footprint when the congestion is recognized, despite it being detrimental to the throughput of the individual TCP connection.

# TOPOLOGY

The Windows desktop computer (IP address: 10.200.129.19

# PROCESS

The three primary metrics that will be measured and examined are average round-trip time of a packet, maximum round-trip

is tricky to measure the number of hops on mobile, so the metric is restricted to the North American Amazon hosts only.

# RESULTS

CLI Commands used through Windows Command Line (similar actions were done through iOS measurement apps) (host names are listed in Appendix A):

*$ ping HOST\_NAME*

*$ tracert -h 50 HOST\_NAME*

Table 1. Internet Measurement Results for Windows PC via Wi-Fi Interface

|  |  |  |  |
| --- | --- | --- | --- |
| **AWS Region** | **Avg RTT** | **Max RTT** | **Num Hops** |
| Virginia | 23 ms | 26 ms | 10 |
| Ohio | 39 ms | 100 ms | 15 |
| California | 58 ms | 60 ms | 18 |
| Oregon | 74 ms | 78 ms | 14 |
| Central Canada | 35 ms | 38 ms | 12 |
| Ireland | 123 ms | 195 ms | 6 |
| Tokyo | 167 ms | 168 ms | 22 |

Table 2. Internet Measurement Results for Apple iPhone via Verizon 4G LTE Cellular Data

|  |  |  |  |
| --- | --- | --- | --- |
| **AWS Region** | **Avg RTT** | **Max RTT** | **Num Hops** |
| Virginia | 88 ms | 264 ms | 13 |
| Ohio | 71 ms | 92 ms | 20 |
| California | 131 ms | 348 ms | 15 |
| Oregon | 142 ms | 353 ms | 17 |
| Central Canada | 94 ms | 277 ms | 15 |
| Ireland | 217 ms | 354 ms | N.A. |
| Tokyo | 212 ms | 487 ms | N.A. |

# ANALYSIS & CONCLUSION

The results provide intuitive data based off of a general understanding of packet switching. Propagation delay is directly related to the distance between the source and each hop until the destination is reached, so it makes sense that the AWS hosts further away from Tuscaloosa, AL, have greater round-trip time for both mobile cellular and desktop Wi-Fi networking. All around, the average and maximum round-trip times for mobile cellular were significantly lower than the desktop Wi-Fi data, and the variance of the round-trip times was significantly higher on mobile as indicated by the high maximum round-trip time values. This is likely due to how the home router can transmit data at a much faster rate due to how it does not need to support as many devices or have as far of a reach as mobile cellular towers.

There are a few peculiarities in the data gathered. The biggest surprise in the data is the performance on both mobile and desktop for sending packets to the Central Canada region, located near Montreal, Quebec. The distance between Tuscaloosa and Montreal is about 1200 miles, while the distance to Ohio is half of that at about 600 miles, but the performance to Canada was either very close to Ohio or outright better in many cases. Another interesting data point is how few hops packets took to get to the data center located in Ireland. The second-to-last link was a transatlantic link that took up the majority of the round-trip time spent, and then the next link was the link that connected to the Amazon hosts, which contrasts very much to the data center in Japan that had the highest number of hops at 22, including many transpacific links versus the singular transatlantic link the Ireland packet took.

As an application developer targeting a Tuscaloosa, Alabama audience, the logical choice would be to choose to deploy cloud resources into the AWS region located in Virginia in order to minimize the time spent on networking. However, it is rare for a software user base to be located primarily in one city or even one region, which is why it is important to gather Internet measurements for many different users in order for to make software users have the best possible experience, or as in the direction of cloud computing is headed, having distributed networked applications with resources in a variety of locations in order to minimize round-trip time of packets for as many users as possible.

# REFERENCES

1. Jake Frankenfield. 2021. How Cloud Computing Works. (May 2021). Retrieved October 22, 2021 from <https://www.investopedia.com/terms/c/cloud-computing.asp>
2. Amazon Web Services, Inc. What is AWS? (2021). Retrieved October 22, 2021 from <https://aws.amazon.com/what-is-aws/>
3. Amazon Web Services, Inc. Amazon DynamoDB Endpoints and Quotas (2021). Retrieved October 22, 2021 from <https://docs.aws.amazon.com/general/latest/gr/ddb.html>
4. Ping (networking utility). (October 2021). Retrieved October 22, 2021 from <https://en.wikipedia.org/wiki/Ping_(networking_utility)>
5. Traceroute. (October 2021). Retrieved October 22, 2021 from <https://en.wikipedia.org/wiki/Traceroute>
6. Apple App Store. iNetTools – Ping,DNS,Port Scan (2021). Retrieved October 22, 2021 from <https://apps.apple.com/us/app/inettools-ping-dns-port-scan/id561659975>
7. Apple App Store. Network Anaylzer (2021). Retrieved October 22, 2021 from <https://apps.apple.com/us/app/network-analyzer/id562315041>

# APPENDIX

A. AWS Region to DynamoDB Host Name

|  |  |
| --- | --- |
| **AWS Region** | **Dynamo Hostname** |
| Virginia | dynamodb.us-east-1.amazonaws.com |
| Ohio | dynamodb.us-east-2.amazonaws.com |
| California | dynamodb.us-west-1.amazonaws.com |
| Oregon | dynamodb.us-west-2.amazonaws.com |
| Central Canada | dynamodb.ca-central-1.amazonaws.com |
| Ireland | dynamodb.eu-west-1.amazonaws.com |
| Tokyo | dynamodb.ap-northeast-1.amazonaws.com |