CSC466 Bi-Weekly Update 2

March 7, 2025

Ryan Dreher, Ben Howland, Parris Mook-Sang-Forbes

I. Progress Summary

Since our previous bi-weekly update, we have focused on collecting data for future analysis using the Mininet model. Final test-environment configuration items that were unfinished before the previous update were completed, including the adjustment of the Mininet architecture to follow the proposed network model more precisely. Data on key FTP performance metrics was collected using many of the tools we outlined in the previous update. These metrics were obtained for our three chosen FTPs (SFTP, FTPS, SCP) across varying network conditions, for both singular large-file and numerous small-file transfers, and with varying key encryption standards. Lastly, this data was recorded to spreadsheets for our project's upcoming data analysis phase.

II. Data Collection: Method and Challenges

As outlined two weeks ago, the test environment is established on a virtual machine (VM) with a fresh Ubuntu 24.04.2 LTS Server, outfitted with a standalone installation of Mininet and an array of benchmarking packages (iperf3, curl, iftop, nload, tcpdump, iotop, htop, and Wireshark). Mininet's Python API enables us to build our desired architecture (see figure below) in a Python environment, where we can automate our tests via scripting.

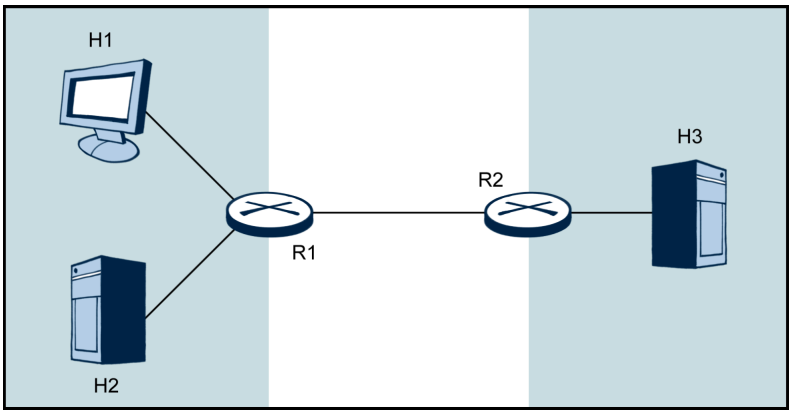

Figure 1: Mininet Topology

Our custom topography is described in a class, which is defined by the network's endpoints (hosts), routers (switches), and the connections (links) between them. The figure above is slightly simplified as each router needs to be connected to a switch, which then connects all devices within a domain. The level of jitter and delay assigned to each link can be adjusted. However, our Mininet is only masquerading as a network of servers; in reality, it is just one server. Therefore, we must rely on Open vSwitch (OVS) – Mininet's pre-installed default switch implementation – to create virtual network interfaces and simulate traffic routing.

In the initial attempts to acquire our data, we encountered an issue which took some time to overcome. The client reported no route to the destination server despite the server receiving at least some of the data from the client.

The cause for this was determined to be the configuration of IP addresses. Initially, the link between the two networks (between the routers) was not correctly configured using known public and private IPs, so there was no route to exit from a given domain. This was eventually corrected to add a proper route utilizing the commands shown below in Figure 2.

```
r1.setIP('10.0.0.254/24', intf=r1.intf('r1-eth0'))
r1.setIP('10.1.0.1/24', intf=r1.intf('r1-eth1'))

r2.setIP('10.0.1.254/24', intf=r2.intf('r2-eth0'))
r2.setIP('10.1.0.2/24', intf=r2.intf('r2-eth1'))
```

Figure 2: IP configuration required to connect two simulated network domains.

Another challenge we faced in trying to collect the required data was related to using the FTPS protocol. In the existing script, we must destroy the network structure and change the configuration/parameters each time we run a test. With SFTP and SCP, which use SSH, which works by the client providing the private key in the command and only requires opening a port for OpenSSH, this does not cause any problems. FTPS, however, utilizes SSL certificates for authentication in implicit mode, with the server doing the authentication. Because the server is the one serving the SSL certificate, this causes problems with reachability and transfer completion each time a new test is run.

In addition to the previously discussed tests of each protocol in varying network conditions with either small or large files, we have added a new test to determine the performance effect three cryptographic hash functions on the same SSH key have on overall transfer time. This was added to increase thoroughness in our testing and better determine what kinds of security and performance tradeoffs we can expect in different conditions and what we can tolerate based on the application. The key types/standards used are listed below:

SFTP, SCP: ED25519, RSA-2048, and RSA-4096

FTPS: ECDSA P256, RSA-2048, and RSA-4096

These were selected because RSA-2048 is the most commonly used key type currently and provides moderate security, RSA-4096 provides enhanced security with a noticeable increase in time to generate, and the last key type represents an older, depreciated, but third-most popular option for each protocol type. From testing, these algorithm types make a significant difference for transfers, where the older algorithms are noticeably slower.

To measure encryption performance on FTPS, which is not based on SSH, we have created a self-signed SSL certificate for each standard using OpenSSL. Because we ultimately control the network, we do not need to worry about the real-world security risks associated with self-signed certificates. By changing the vsftpd (Very Secure FTP Daemon) config, we can select which of these certificates to use.

Key parts of the Python script used for Mininet configuration and data collection are provided below, which, while still subject to potential revisions, has provided us with our initial data for analysis. Data will be stored in CSV format. The project report will give the complete code (in the interest of maintaining the readability of this update).

Defines specific parameters for differing network types:

```
client_connection_types = [
    {'type': 'Ethernet', 'latency': 1, 'jitter': 0.1, 'loss': 0},
    {'type': 'Coax', 'latency': 5, 'jitter': 1, 'loss': 0.1},
    {'type': 'WiFi', 'latency': 10, 'jitter': 5, 'loss': 1},
    {'type': 'Fibre', 'latency': 1, 'jitter': 0.05, 'loss': 0},
]

router_connection_types = [
    {'type': 'Fibre', 'latency': 65, 'jitter': 1, 'loss': 0},
    {'type': 'Wireless', 'latency': 100, 'jitter': 10, 'loss': 0.5},
]
```

Building Mininet topology:

```
    def build(self):
        # just base vals, dont stay. based on for loops with connection types.
        r1 = self.addNode('r1', cls=Router)
        r2 = self.addNode('r2', cls=Router)
        c1 = self.addHost('c1', ip='10.0.0.1/24', defaultRoute='via 10.0.0.254')
        c2 = self.addHost('c2', ip='10.0.0.2/24', defaultRoute='via 10.0.0.254')
        c3 = self.addHost('c3', ip='10.0.1.3/24', defaultRoute='via 10.0.1.254')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')

        self.addLink(c1, s1, cls=TCLink, delay='5ms', jitter='1ms')
        self.addLink(c2, s1, cls=TCLink, delay='5ms', jitter='1ms')
        self.addLink(c3, s2, cls=TCLink, delay='5ms', jitter='1ms')
        self.addLink(s1, r1, cls=TCLink, delay='5ms', jitter='1ms')
        self.addLink(s2, r2, cls=TCLink, delay='5ms', jitter='1ms')
        self.addLink(r1, r2, cls=TCLink, delay='30ms', jitter='5ms')
```

Metric gathering:

```
def gather_metrics(duration, results, label):
    start_time = time.time()
    while time.time() - start_time < duration:
        cpu, memory, io_counters, net_counters = measure_system_metrics()
        results.append({
            'label': label,
            'time': time.time(),
            'cpu': cpu,
            'memory': memory,
            'io_read_count': io_counters.read_count,
            'io_write_count': io_counters.write_count,
            'io_read_bytes': io_counters.read_bytes,
            'io_write_bytes': io_counters.write_bytes,
            'net_bytes_sent': net_counters.bytes_sent,
            'net_bytes_recv': net_counters.bytes_recv,
            'net_packets_sent': net_counters.packets_sent,
            'net_packets_recv': net_counters.packets_recv
        })
        time.sleep(0.5)
```

Modifying network conditions:

```
def apply_network_conditions(net, client_condition, router_condition):
    for link in net.links:
        if ('c1' in str(link.intf1) and 's1' in str(link.intf2)) or ('s1' in str(link.intf1) and 'c1' in str(link.intf2)):
            link.intf1.config(delay=f"{client_condition['latency']}ms", jitter=f"{client_condition['jitter']}ms", loss=client_condition['loss'])
            link.intf2.config(delay=f"{client_condition['latency']}ms", jitter=f"{client_condition['jitter']}ms", loss=client_condition['loss'])

    for link in net.links:
        if ('r1' in str(link.intf1) and 'r2' in str(link.intf2)) or ('r2' in str(link.intf1) and 'r1' in str(link.intf2)):
            link.intf1.config(delay=f"{router_condition['latency']}ms", jitter=f"{router_condition['jitter']}ms", loss=router_condition['loss'])
            link.intf2.config(delay=f"{router_condition['latency']}ms", jitter=f"{router_condition['jitter']}ms", loss=router_condition['loss'])
```

Figure 3 shows a sample of the terminal output, which indicates the type of transfer occurring, source and destination network nodes, encryption key in use, current simulated network architecture (i.e., which type of network are we attempting to emulate), file size, and overall throughput rate and time it took to complete the transfer.

```
d_ed25519 from c1 to c2
Client Type: Ethernet, Router Type: Fibre, File Size: 100M
Testing scp transfer with key id_ed25519 from c1 to c2
file_100M                                     100%  100MB  98.3MB/s   00:01

Deleted file /home/ryan/CSC446/landing/file_100M on destination host.
Testing scp transfer with key id_ed25519 from c1 to c2
Client Type: Ethernet, Router Type: Fibre, File Size: 100M
Testing scp transfer with key id_ed25519 from c1 to c2
file_100M                                     100%  100MB  81.0MB/s   00:01

Deleted file /home/ryan/CSC446/landing/file_100M on destination host.
Testing scp transfer with key id_ed25519 from c1 to c2
Client Type: Ethernet, Router Type: Fibre, File Size: 100M
Testing scp transfer with key id_ed25519 from c1 to c2
file_100M                                     100%  100MB  99.3MB/s   00:01
```

Figure 3: A sample of terminal output from the data collection.

Finally, Figures 4 and 5 below show an example of some of the data obtained through the testing, which we will use to conclude the following stages.

A	B	C	D	E	F	G	H	I	J	K	L	M
client_type	router_typ	type	key	ssl_cert	size	direction	stdout	label	time	cpu	memory	io_read_count
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transf	1741398344	0.5	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transf	1741398346	1.2	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transf	1741398347	1.5	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transf	1741398349	0.1	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_tra	1741398350	0.1	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_tra	1741398352	0.2	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_tra	1741398353	0.4	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_tra	1741398355	0	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_tra	1741398356	0.4	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_tra	1741398358	0.1	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_tra	1741398359	0.1	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_trans	1741398363	0.2	7.9	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_trans	1741398365	0.1	7.9	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_trans	1741398366	0.2	7.8	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_trans	1741398368	0.8	7.7	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transf	1741398369	0	7.7	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transf	1741398371	0.4	7.6	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transf	1741398372	0.1	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transf	1741398374	0	7.5	38102
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_tra	1741398375	0.1	7.5	38102

Figure 4: Sample of some collected data for a single file transfer.

A	B	C	D	E	F	G	H	I	J	
client_type	router_type	type	key	ssl_cert	size	direction	stdout	label	time	cp
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398344	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398346	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398347	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398349	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398350	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398352	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398353	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398355	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398356	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398358	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398359	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_transfer	1741398363	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_transfer	1741398365	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_transfer	1741398366	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_transfer	1741398368	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398369	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398371	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398372	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398374	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398375	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398377	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398378	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398380	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398381	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398383	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398384	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_transfer	1741398388	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_transfer	1741398389	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_transfer	1741398391	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	post_transfer	1741398392	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398394	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398395	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398397	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	pre_transfer	1741398398	
Ethernet	Fibre	scp	id_ed25519		100M	c1 to c2	file_100M	during_transfer	1741398400	

Figure 5: Sample of collected data showcasing collection before, during, and after transfer.

III. Next Steps

As outlined by the project proposal, the project's next phase will mainly consist of data analysis. The first step in this analysis will be aggregating and partitioning data for all three protocols that correspond to a given network test condition, which will allow us to generate data visualizations more easily.

Once complete, an initial check of the data across test runs will be made to ensure no extreme outliers or inconsistent values are found. This can be done through standard deviation or variance calculations. Additional test runs may be performed if outliers exist to ensure sufficiently accurate data. After this, we will be able to compare the values numerically and graphically for each performance metric collected in each network condition category. From this, we can determine the tradeoffs for each file transfer protocol and whether these differences changed or are magnified under different network conditions. Testing each protocol with different key encryption standards will shed further light on performance vs security tradeoffs that may impact scalability. Based on our findings, we can suggest each protocol's strengths, weaknesses, and optimal use cases.

IV: References

[1] Mininet Project Contributors. (2022). Mininet walkthrough. Mininet. https://mininet.org/walkthrough/

[2] SpaceX. (2025). Starlink specifications. Starlink. https://www.starlink.com/legal/documents/DOC-1400-28829-70