Port Forwarder

COMP 8005 – Final Project - Discussion

**Danny Lieu**

**Mike Zobac**

**Feb 27, 2018**

Table of Contents

[PROCEDURE 3](#_Toc509772769)

[TEST DETAIL 3](#_Toc509772770)

[Configuration 3](#_Toc509772771)

[Port-Forwarding 3](#_Toc509772772)

[SSH 3](#_Toc509772773)

[HTTP 3](#_Toc509772774)

[Scalability 5](#_Toc509772775)

[Load Balancing 6](#_Toc509772776)

# PROCEDURE

We tested the capability of our port-forwarder on two fronts. First, we tested the port-forwarding functionality by connecting a single machine to a single remote service. We then tested the scalability of the program by forwarding many simultaneous connections.

# TEST DETAIL

## Configuration

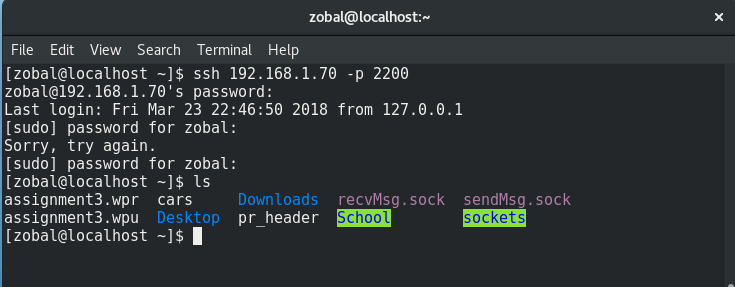
The forwarder uses a configuration file to map services to remote and local ports, as well as to remote host addresses. When a client requests a service on the local port, the forwarder opens a new connection to that service’s remote port on the mapped remote host. Each remote service in the configuration file is on a unique host.

## Port-Forwarding

We tested the port-forwarding functionality by connecting a single client to remote services. We chose to use a browser to connect to port 80 on a remote host, and SSH to connect to port 22 on another remote host.

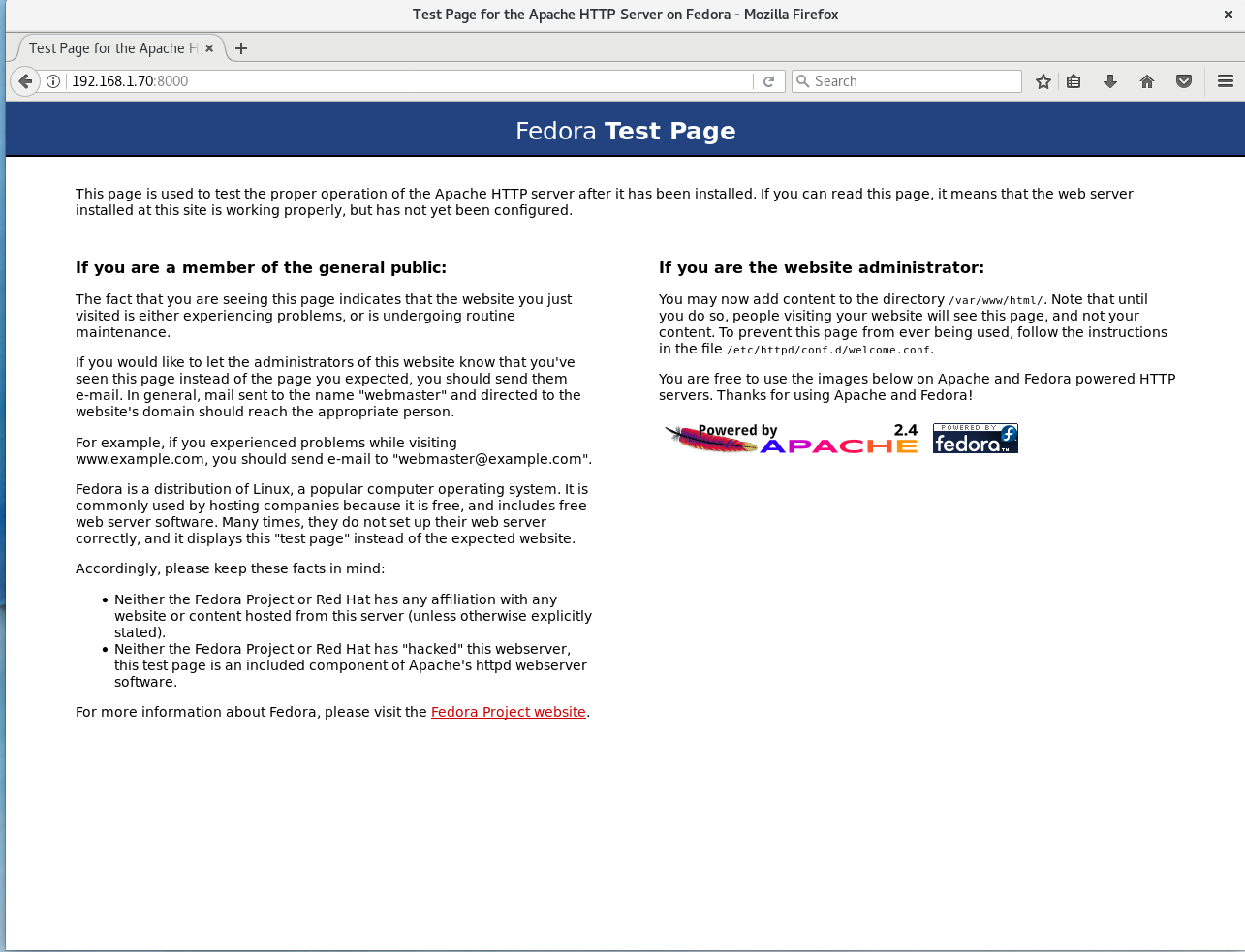
### SSH

SSH is mapped to well-known port 22 on the remote host, and port 2200 locally. We ran the SSH program on a bash shell and connected to port 2200 on the port-forwarder, which then forwarded to port 22 on the remote machine.



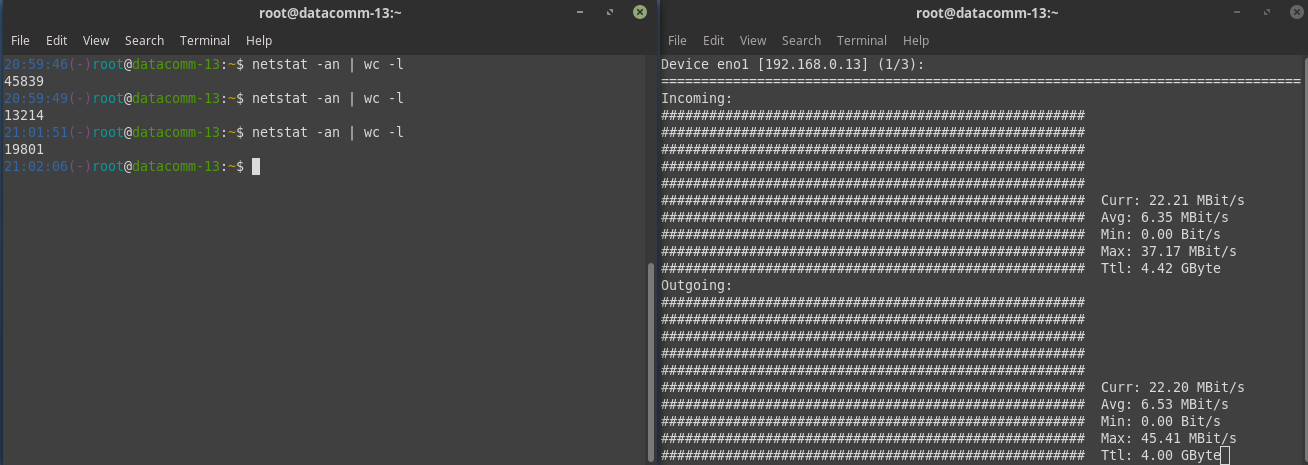
### HTTP

HTTP is mapped to well-known port 80 on the remote host, and port 8000 locally. By running a browser on a remote client machine, and entering 192.168.1.70:8000 we were able to connect to port 8000 on the port-forwarder, which then forwarded to port 80 on a machine running at 192.168.1.82

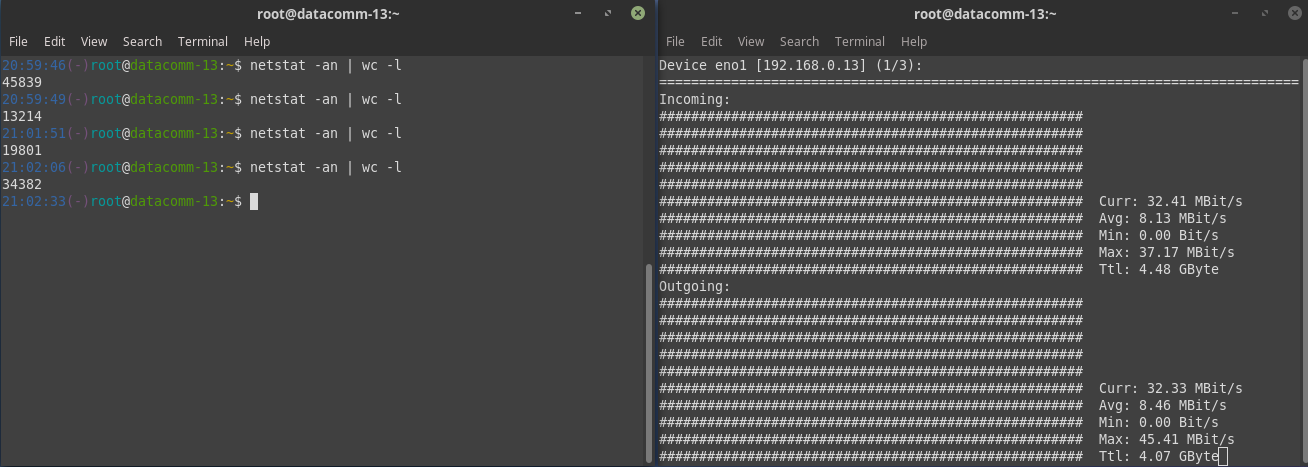


## Scalability

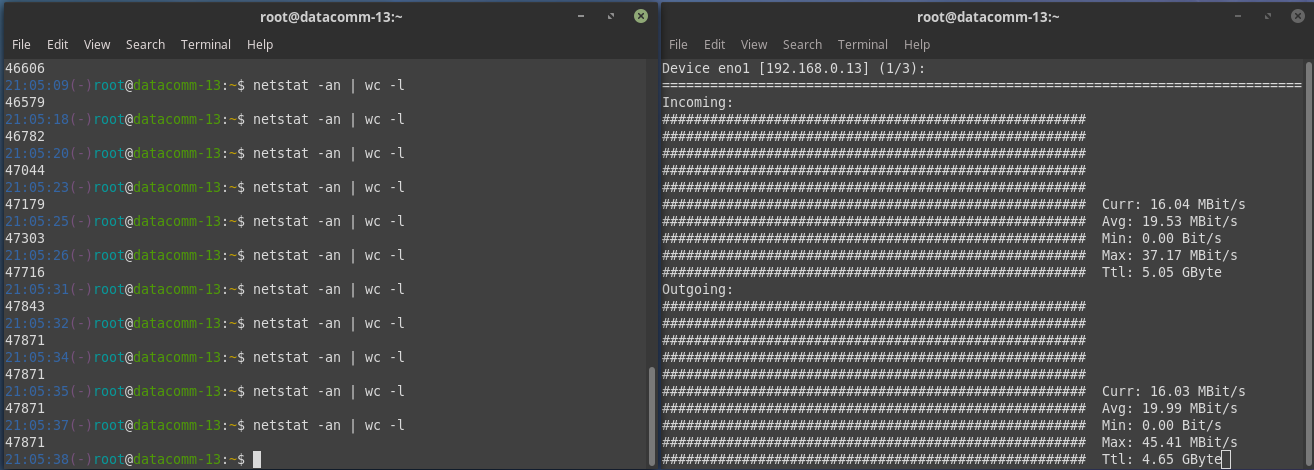
To test scalability, we connected many clients simultaneously the same remote service though the port forwarder. The remote service was a simple echo-server, built from the same code this port forwarder is based on. Connected clients exchange messages with the server at the rate of one-per-second. To create many connections at once, we designed a client program which creates one process for each CPU on the machine. Each of these processes creates 1000 connections to the server, and each connection sends a message and is echoed a response. This design allowed us to create many thousand connections in very little time. A few seconds after restarting the clients we had almost 20000 connections transferring at 22.21 MBps. This speed slowed as the number of connection increased.



A few seconds later we had 15000 more connections with an impressive current speed, and an increasing average speed.



After three and a half minutes the number of connections plateaued at 47871, with an impressive average throughput of 19.53 Mbps, and near matching statistics for both incoming and outgoing data. This indicates that all messages are being forwarded properly. It should be noted that though the number of connections is 47870 (the extra connection is the listening socket on the server) the actual number of clients being served is 23965. This is because two connections are required for each client.



## Load Balancing

We made very little effort to load balance the port-forwarder. When a new client connection is accepted, the main thread simply passes the new socket off to the next worker thread in line. When it gets to the end of the line, it loops back to the front. Because of an unreliability in the client program, some worker threads ended up serving more clients that others.

