Video Streaming

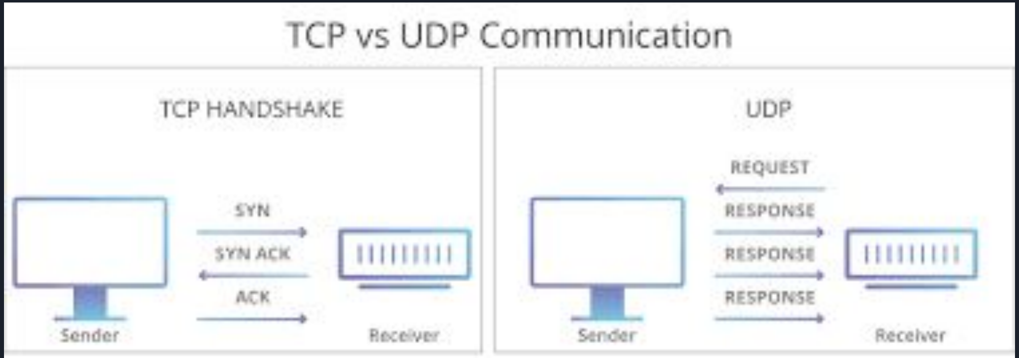
ECE 4400/6400 Project
Andrew Eubanks, Eric Mitchell, Josh Silva, Sam Quan

Motivation

- Video streaming is a popular service that depends highly on network speed
- We want to explore what causes load times, buffering, and quality adjustment
- We use these services ourselves

Background

- Streaming services send data to users using transport protocols
- Netflix, Hulu, and Youtube use TCP (Transmission Control Protocol)
- Zoom and Twitch use UDP (User Datagram Protocol)



Project Description

- Video streaming is affected by the following:
  - Bandwidth
  - Latency
  - Throughput

- It may also be affected by other network conditions

Project Description (2)

- Our first experiment focuses on how client bandwidth affects video delay (buffering)
- We will simulate video delay with packet loss and delay
- We will stream video in attempt to show network effects in real time

Resolution	Name	Quality
3840 x 2160	2160p	4K
2560 x 1440	1440p	2K
1920 x 1080	1080p	Full HD maximum resolution
1280 x 720	720p	HD minimum resolution
854 x 480	480p	Standard definition
640 x 360	360p	Normal website resolution
426 x 240	240p	YouTube minimum video size

INTERNET ADVANTAGE
100 Mbps Internet
High-speed Internet at a budget-friendly price.
$30 /mo
for 1 year

INTERNET PREMIER
500 Mbps Internet
Internet speeds that can handle it all.
$50 /mo
for 1 year

INTERNET GIG
1 Gig Internet
Get the ultimate experience ideal for serious gamers, streamers and large homes.
$70 /mo
for 1 year

Modeling Video Streaming

- Most video streaming services use TCP for reliability
- Service streams packets for video, which increase in bandwidth requirement as streaming quality increases
- Reviewed papers analyzing Youtube traffic to approximate needed rates

Bandwidth Requirements for Different Video Resolutions

Here are general bandwidth guidelines for various video resolutions and bitrates:

- **480p (SD):** 1.5 – 2.5 Mbps
- **720p (HD):** 3 – 5 Mbps
- **1080p (Full HD):** 5 – 8 Mbps
- **1440p (2K):** 10 – 16 Mbps
- **2160p (4K):** 20 – 35 Mbps

Original Teammate Responsibilities

- AJ: Experiment with queue sizes to determine which sizes we will use in our final analysis
- Eric: Configure Jupyter notebook with the server topology and help define network limits using tc
- Josh: Parse through output and create figures with Pandas and Matplotlib
- Sam: Determine which tc rates will create the best representation of the concepts we are exploring

Actual Teammate Responsibilities

- AJ: Experimented with varying rates for each of the clients using UDP. Created a python script to run the experiment all at once and collected results for varying bandwidths
- Eric: Wrote new presentation proposal doc and slideshow, configured Jupyter notebook with the network topology, completed troubleshooting with the original TCP plan, experimented with TCP
- Josh: Assisted with Network Setup (IP Routing, IPv4 Forwarding & IPTables), TCP Experiments (limit rates and bandwidth)
- Sam: Researched and ran real video-streaming with FFMPEG for TCP/UDP experiment and demonstration, using libcaca to view video as ascii

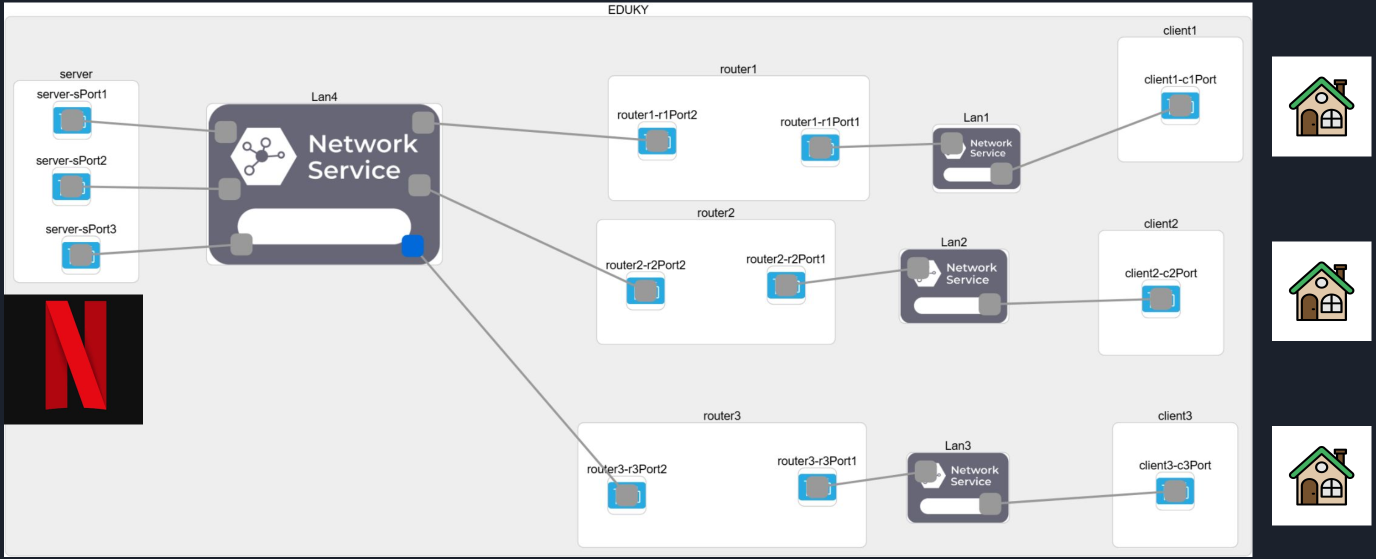
Project Challenges and Changes

- Challenges with iPerf and TCP: auto rate adjustment
- Can’t measure packet loss because iPerf avoids it with TCP
- Modified TCP experiment to show how the data transmission rate declines with heavier concurrent use
- New UDP experiment to show stress test data with packet loss and delay

Network Architecture

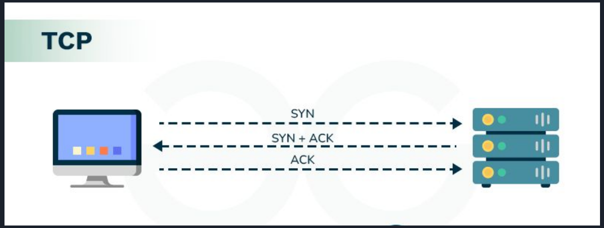
- Streaming platform server connected to 3 clients via routers

- Star configuration



TCP Experiment

TCP Experiment



- As mentioned earlier, most Video Streaming services make use of Transmission Control Protocol, or TCP.
- TCP is meant to have no data loss, meaning no packet loss unlike UDP. However if Transfer Rate cannot catch up, video buffering or lower quality is expected.
- Experiments will test the effects of limit rate on the router, and bandwidth from the server.

TCP Experiment Parameters

- To validate our simulated Network, need to test what data it can take in.

- Test out various "download speeds" for various "Video Resolutions" suggestions from Streaming Services.

- Make use of iPerf and tc-tbf to simulate video streaming. Provides easy access to data such as Transfer Rate.

Internet connection speed recommendations

To watch TV shows and movies on Netflix, we recommended having a stable internet connection with a download speed shown below in megabits per second (Mbps).

Video quality	Resolution	Recommended speed
High definition (HD)	720p	3 Mbps or higher
Full high definition (FHD)	1080p	5 Mbps or higher
Ultra high definition (UHD)	4K	15 Mbps or higher

Netflix Download Speed Recommendations

Bandwidth Requirements for Different Video Resolutions

Here are general bandwidth guidelines for various video resolutions and bitrates:

- **480p (SD):** 1.5 – 2.5 Mbps
- **720p (HD):** 3 – 5 Mbps
- **1080p (Full HD):** 5 – 8 Mbps
- **1440p (2K):** 10 – 16 Mbps
- **2160p (4K):** 20 – 35 Mbps

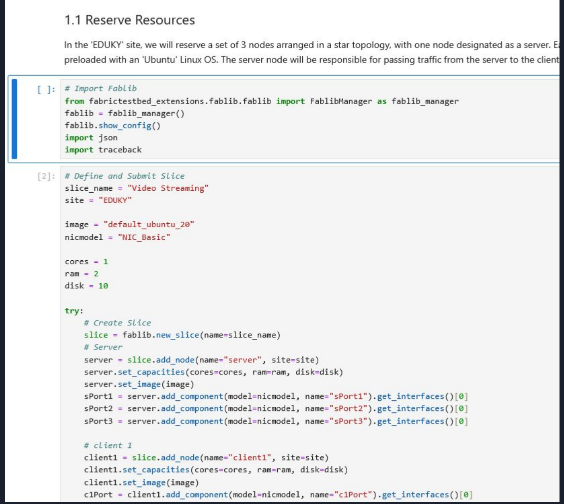
VODLIX Guidelines for Bandwidth

TCP Experiment Setup

- To start the experiment, CreateSlice is run, like UDP. Allocates resources & establishes IP Route

- After Network Setup, various router parameters are set. Limit Rate is first tested, all other variables kept constant

- After testing valid Limit Rates, bandwidths are altered to simulate different video resolutions.



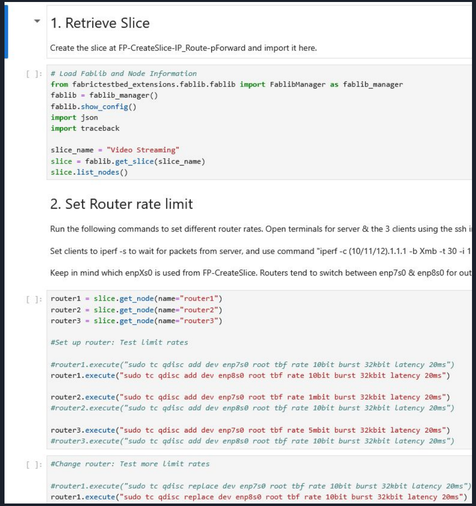
1.1 Reserve Resources

In the 'EDUKY' site, we will reserve a set of 3 nodes arranged in a star topology, with one node designated as a server. E
preloaded with an 'Ubuntu' Linux OS. The server node will be responsible for passing traffic from the server to the client

```
# Import Fablib
from fabrictestbed_extensions.fablib.fablib import FablibManager as fablib_manager
fablib = fablib_manager()
fablib.show_config()
import json
import traceback
```

```
# Define and Submit Slice
slice_name = "Video Streaming"
site = "EDUKY"

image = "default_ubuntu_20"
nicmodel = "NIC_Basic"

cores = 1
ram = 2
disk = 10

try:
    # Create Slice
    slice = fablib.new_slice(name=slice_name)
    # Server
    server = slice.add_node(name="server", site=site)
    server.set_capacities(cores=cores, ram=ram, disk=disk)
    server.set_image(image)
    sPort1 = server.add_component(model=nicmodel, name="sPort1").get_interfaces()[0]
    sPort2 = server.add_component(model=nicmodel, name="sPort2").get_interfaces()[0]
    sPort3 = server.add_component(model=nicmodel, name="sPort3").get_interfaces()[0]

    # client 1
    client1 = slice.add_node(name="client1", site=site)
    client1.set_capacities(cores=cores, ram=ram, disk=disk)
    client1.set_image(image)
    c1Port = client1.add_component(model=nicmodel, name="c1Port").get_interfaces()[0]
```

CreateSlice Snippet



1. Retrieve Slice

Create the slice at FP-CreateSlice-IP_Route-pForward and import it here.

```
# Load Fablib and Node Information
from fabrictestbed_extensions.fablib.fablib import FablibManager as fablib_manager
fablib = fablib_manager()
fablib.show_config()
import json
import traceback

slice_name = "Video Streaming"
slice = fablib.get_slice(slice_name)
slice.list_nodes()
```

2. Set Router rate limit

Run the following commands to set different router rates. Open terminals for server & the 3 clients using the ssh i

Set clients to iperf -s to wait for packets from server, and use command "iperf -c (10/11/12).1.1.1 -b Xmb -t 30 -i 1

Keep in mind which enpXs0 is used from FP-CreateSlice. Routers tend to switch between enp7s0 & enp8s0 for out

```
router1 = slice.get_node(name="router1")
router2 = slice.get_node(name="router2")
router3 = slice.get_node(name="router3")

#Set up router: Test limit rates

#router1.execute("sudo tc qdisc add dev enp7s0 root tbf rate 10bit burst 32kbit latency 20ms")
router1.execute("sudo tc qdisc add dev enp8s0 root tbf rate 10bit burst 32kbit latency 20ms")

router2.execute("sudo tc qdisc add dev enp7s0 root tbf rate 1mbit burst 32kbit latency 20ms")
#router2.execute("sudo tc qdisc add dev enp8s0 root tbf rate 10bit burst 32kbit latency 20ms")

router3.execute("sudo tc qdisc add dev enp7s0 root tbf rate 5mbit burst 32kbit latency 20ms")
#router3.execute("sudo tc qdisc add dev enp8s0 root tbf rate 10bit burst 32kbit latency 20ms")
```

```
#Change router: Test more limit rates

#router1.execute("sudo tc qdisc replace dev enp7s0 root tbf rate 10bit burst 32kbit latency 20ms")
router1.execute("sudo tc qdisc replace dev enp8s0 root tbf rate 10bit burst 32kbit latency 20ms")
```

TCP_Experiment Snippet

TCP Experiment Results

- Higher Limit Rates
  - Less Transfer Rate Loss
- Higher Bandwidths
  - More Transfer Rate Loss
- For each bandwidth increase
  - Min Limit Rate increases by 5-10 Mb/s

TCP Experiment Data

TCP Experiment Conclusion

- Network functions as expected. Able to handle incoming Packets.
- Mostly matches online recommendations. Might need slightly higher download speeds than recommended.
- Ready for more tests with FFmpeg. No “skipping” expected for videos but buffering is possible.

Internet connection speed recommendations

To watch TV shows and movies on Netflix, we recommended having a stable internet connection with a download speed shown below in megabits per second (Mbps).

Video quality	Resolution	Recommended speed
High definition (HD)	720p	3 Mbps or higher
Full high definition (FHD)	1080p	5 Mbps or higher
Ultra high definition (UHD)	4K	15 Mbps or higher

Netflix Download Speed Recommendations

```
------------------------------------------------------------
Client connecting to 10.1.1.1, TCP port 5001
TCP window size: 93.5 KByte (default)
------------------------------------------------------------
[  3] local 13.1.1.1 port 59252 connected with 10.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  2.62 MBytes  22.0 Mbits/sec
[  3]  1.0- 2.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3]  2.0- 3.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3]  3.0- 4.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3]  4.0- 5.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3]  5.0- 6.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3]  6.0- 7.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3]  7.0- 8.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3]  8.0- 9.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3]  9.0-10.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 10.0-11.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 11.0-12.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 12.0-13.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 13.0-14.0 sec  2.50 MBytes  21.0 Mbits/sec
[  3] 14.0-15.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 15.0-16.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 16.0-17.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 17.0-18.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 18.0-19.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 19.0-20.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 20.0-21.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 21.0-22.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 22.0-23.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 23.0-24.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 24.0-25.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 25.0-26.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 26.0-27.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3] 27.0-28.0 sec  2.50 MBytes  21.0 Mbits/sec
[  3] 28.0-29.0 sec  2.38 MBytes  19.9 Mbits/sec
[  3]  0.0-30.0 sec  71.6 MBytes  20.0 Mbits/sec
```

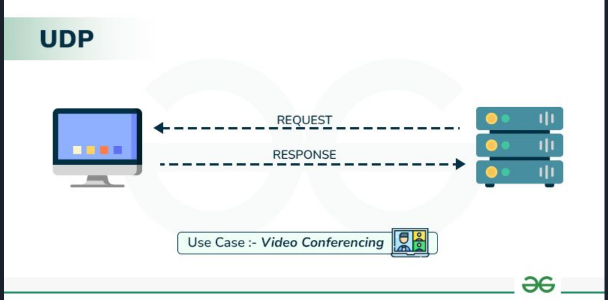
```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
[  4] local 10.1.1.1 port 5001 connected with 13.1.1.1 port 59252
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-30.0 sec  71.6 MBytes  20.0 Mbits/sec
```

“Client/Server” Data

UDP Experiment

UDP Experiment

- Less common, some platforms use UDP, which does not retransmit lost data like TCP
- This is common for services that want to prioritize speed
- Experiment tests clients with different connection speeds trying to stream different levels of Bandwidths
- Experiment measures packet loss as the main metric and tries to relate each Bandwidth to a corresponding video streaming quality
- Examples: Zoom, Twitch



UDP Experiment With Varying Rates - Code

```
from fabrictestbed_extensions.fablib.fablib import FablibManager as fablib_manager
import time

# Function to apply rate limit, delay, and buffer limit on server's interface to each client
def apply_netem_on_server(server, iface, rate, delay, limit):
    server.execute(f"sudo tc qdisc del dev {iface} root || true")
    server.execute(f"sudo tc qdisc add dev {iface} root netem rate {rate} delay {delay} limit {limit}")

# Function to run iperf UDP stream from server to client
def run_udp_stream(server, client_ip, rate, duration=10):
    server.execute(f"iperf -u -c {client_ip} -b {rate} -t {duration} -i 5")

# Initialize slice
fablib = fablib_manager()
slice = fablib.get_slice("Video Streaming")

# Get server and clients
server = slice.get_node("server")
client1 = slice.get_node("client1")
client2 = slice.get_node("client2")
client3 = slice.get_node("client3")

iface1 = server.get_interface(network_name="Lan1").get_device_name()
iface2 = server.get_interface(network_name="Lan2").get_device_name()
iface3 = server.get_interface(network_name="Lan3").get_device_name()

# Apply different network limitations
apply_netem_on_server(server, iface1, rate="10mbit", delay="20ms", limit=2)   # Poor connection
apply_netem_on_server(server, iface2, rate="50mbit", delay="20ms", limit=10)  # Moderate connection
apply_netem_on_server(server, iface3, rate="200mbit", delay="20ms", limit=20)  # Good connection
```

```
# Start UDP servers on clients
for i, client in enumerate([client1, client2, client3], 1):
    client.execute("pkill iperf || true")
    client.execute(f"iperf -s -u > udp_client{i}.txt 2>&1 &")
    print(f"Client {i} listening")

time.sleep(3)

bandwidths = ["2M", "4M", "6M", "8M", "10M"]

# Stream to each client at each bandwidth
for bw in bandwidths:
    run_udp_stream(server, "10.1.1.1", bw)
    run_udp_stream(server, "11.1.1.1", bw)
    run_udp_stream(server, "12.1.1.1", bw)

    # Show client-side iperf results
    for i, client in enumerate([client1, client2, client3], 1):
        stdout, _ = client.execute(f"cat udp_client{i}.txt")
        print(f"\nClient {i} Output for Bandwidth {bw}\n{stdout}")
```

UDP Experiment With Varying Rates - Results

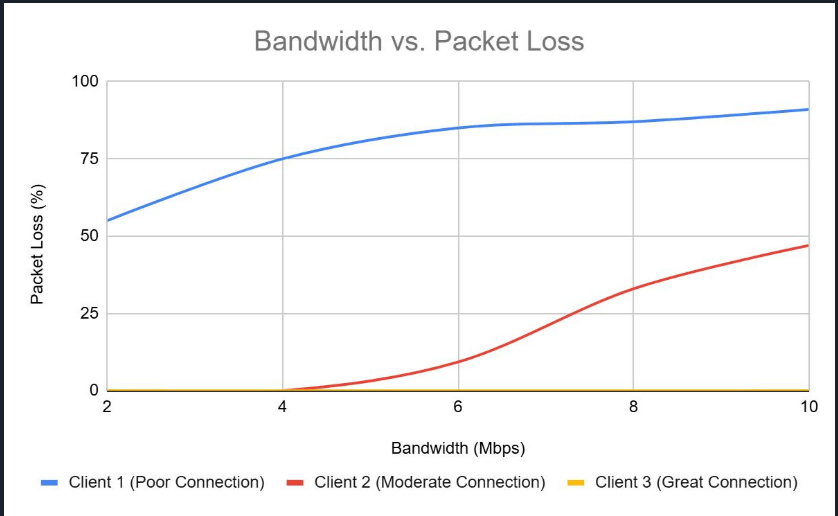
```
Client 1 Output for Bandwidth 10M
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 40148
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec  1.12 MBytes   940 Kbits/sec   1.441 ms  984/ 1784 (55%)
[  4] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 46123
[  4]  0.0-10.3 sec  1.25 MBytes  1.02 Mbits/sec  15.647 ms 2676/ 3568 (75%)
[  3] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 51074
[  3]  0.0-10.2 sec  1.12 MBytes   918 Kbits/sec  15.819 ms 4551/ 5351 (85%)
[  4] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 45688
[  4]  0.0-10.3 sec  1.25 MBytes  1.02 Mbits/sec  15.647 ms 6243/ 7135 (87%)
[  3] local 10.1.1.1 port 5001 connected with 10.1.1.2 port 33443
[  3]  0.0-10.3 sec  1.18 MBytes   966 Kbits/sec  15.800 ms 8076/ 8918 (91%)
```

```
Client 2 Output for Bandwidth 10M
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 11.1.1.1 port 5001 connected with 11.1.1.2 port 34276
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec  2.50 MBytes  2.10 Mbits/sec   0.003 ms    0/ 1784 (0%)
[  4] local 11.1.1.1 port 5001 connected with 11.1.1.2 port 54517
[  4]  0.0-10.0 sec  5.00 MBytes  4.19 Mbits/sec   0.003 ms    0/ 3567 (0%)
[  3] local 11.1.1.1 port 5001 connected with 11.1.1.2 port 41997
[  3]  0.0-10.0 sec  6.80 MBytes  5.71 Mbits/sec   0.004 ms  497/ 5350 (9.3%)
[  4] local 11.1.1.1 port 5001 connected with 11.1.1.2 port 38509
[  4]  0.0-10.0 sec  6.67 MBytes  5.59 Mbits/sec   0.003 ms 2377/ 7134 (33%)
[  3] local 11.1.1.1 port 5001 connected with 11.1.1.2 port 41108
[  3]  0.0-10.0 sec  6.58 MBytes  5.52 Mbits/sec   0.004 ms 4224/ 8917 (47%)
```

```
Client 3 Output for Bandwidth 10M
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 12.1.1.1 port 5001 connected with 12.1.1.2 port 33466
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec  2.50 MBytes  2.10 Mbits/sec   0.007 ms    0/ 1784 (0%)
[  4] local 12.1.1.1 port 5001 connected with 12.1.1.2 port 49864
[  4]  0.0-10.0 sec  5.00 MBytes  4.19 Mbits/sec   0.002 ms    0/ 3567 (0%)
[  3] local 12.1.1.1 port 5001 connected with 12.1.1.2 port 53044
[  3]  0.0-10.0 sec  7.50 MBytes  6.29 Mbits/sec   0.002 ms    0/ 5350 (0%)
[  4] local 12.1.1.1 port 5001 connected with 12.1.1.2 port 55195
[  4]  0.0-10.0 sec  10.0 MBytes  8.39 Mbits/sec   0.002 ms    0/ 7134 (0%)
[  3] local 12.1.1.1 port 5001 connected with 12.1.1.2 port 58345
[  3]  0.0-10.0 sec  12.5 MBytes  10.5 Mbits/sec   0.001 ms    0/ 8917 (0%)
```

UDP Experiment With Varying Rates - Results

		Packet Loss (%)		
Bandwidth (Mbps)	Associated Quality	Client 1 (Poor Connection)	Client 2 (Moderate Connection)	Client 3 (Great Connection)
2	480p	55	0	0
4	720p	75	0	0
6	1080p	85	9.3	0
8	1440p	87	33	0
10	2160p	91	47	0



UDP Experiment With Varying Rates - Conclusion

- A client with a great connection are able to stream at all qualities with no packet loss

- A client with a moderate connection is able to stream at some of the lower qualities with no packet loss, but is unable to stream at very high qualities

- A client with a poor connection is barely able to stream even at super low qualities
  - This may occur when multiple devices are trying to stream data on the same network, or if someone else on the network in downloading something that is using up all of the bandwidth

FFmpeg

FFmpeg Overview

- FFmpeg is open-source project of libraries/commands to handle media streaming
- Run an experiment on real video streaming where we observe effect of router throttling (network speed)
- Video device challenge: use libcaca to view video as ascii in terminal

Demo

Conclusion

- In our first two experiments, we confirmed what we expect from transport protocols
- TCP - reliability
- UDP - speed

- As we expected, today’s networks handle streaming well
- Can handle multiple streaming devices per network

- Platforms have the option to avoid buffering through software

Any Questions?