

# Network Test and Monitoring Tools

Author: A.J.Gillette  
Date: December 6, 2012  
Revision: 1.3

# Table of Contents

Network Test and Monitoring Tools .....	1
Introduction .....	3
Link Characterization .....	4
Summary .....	12
Appendix A: NUTTCP .....	13
Installing NUTTCP .....	13
Using NUTTCP .....	13
NUTTCP Examples .....	14
Appendix B: IPERF .....	17
Installing IPERF .....	17
Using IPERF .....	17
IPERF Examples .....	17
Appendix C: Traceroute & Tracepath .....	22
Appendix D: Ping .....	24

# Introduction

This document makes recommendations for testing and monitoring networks that will be used to transport audio and video media. It is important to understand the amount of bandwidth available to video end points, but it is also important to understand the characteristics of that bandwidth. In many cases bandwidth can:

- be shared between tenants.
- be limited by shapers and policers.
- be subject to packet loss.
- contain out of order packets.

This document will outline how to characterize links so that we can anticipate how they will perform. The target audience for this document would include system installers, network service providers and at a high level, IT professionals responsible for network provisioning and maintenance.

Each tool that is used in the characterization of links has been assigned an appendix. Each appendix contains information about how to acquire, install and use the tool.

Readers should review the examples in each appendix to familiarize themselves with the tools before attempting to use them to characterize a link.

# Link Characterization

## Network Impairments:

There are a number of settings on routers and other equipment involved in the local loop that can affect the service being delivered. It is important to test each network link to verify link performance before adding equipment that may mask some of the symptoms. The following is a list network impairments that have occurred during installations.

- Fiber transceivers set to auto 10/100.
  - Cause a low-level packet loss.
  - The packet loss occurs on a regular interval.
- Routers set to half duplex.
  - Cause early network congestion, by reducing the effective real time bandwidth. Half-duplex data transmission means that data can be transmitted in both directions on a signal carrier, but not at the same time.
- Policers and shapers limiting the upstream or downstream bandwidth.
  - Can cause latency or packet loss.
- Routers or switches that have reached their packet processing limit.
  - Cause irregular packet loss.
- Bad fiber or wiring.
  - Can cause intermittent packet loss.
  - Can cause bandwidth reduction.
- Lossy wireless links.
  - Can cause intermittent packet loss.
  - Can cause bandwidth reduction.
- ISP's with MTU limits at or below 1500 bytes.

Running the “iperf or nuttcp” tools and observing the results can detect each potential problem. In some cases the tests may have to run in parallel or over an extended period of time.

The tests described in the following section are performed between two Linux devices connected over a VPN. The VPN effectively removes problems that could be

caused by NAT / Firewalls. The systems used in these tests are the video end points, but any Linux system would be sufficient. If the tests are to be performed using a windows system, an emulator like Cygwin should be used.

Testing maximum throughput does assume that the system at the far end of the link has sufficient bandwidth to not skew the results. In our examples the remote system is connected to a 100Mbps link and the systems under test are connected to 10Mbps links. It would not be practical to test a 100Mbps access link by connecting to a remote 10Mbps site, as the remote site does not have sufficient bandwidth.

## Recommended Tests:

The tests in the following section will measure bandwidth in each direction. In order to demonstrate some of the typical network impairments we may repeat several of the tests on different network circuits. The results will help installers recognize the different link characteristics associated with policers and shapers.

### Test#1 (Downstream bandwidth):

The object of this test is to determine the amount of available bandwidth inbound to the site under test. The secondary objective is to determine the failure mechanism when the bandwidth limit is reached. For this test we will use IPERF to run a UDP test over a 10 second interval. We will start with a 2Mbps stream and increase in 2Mbps intervals until we start to see packet loss or a significant increase in network latency. The command on the client side to initiate the test will be:

```
iperf -c 192.168.97.10 -u -b 2m -M 1500
```

The commands on the server side to prepare for the test will be:

```
root@fred-laptop:~# iptables -A INPUT -p tcp --dport 5001 -j ACCEPT
root@fred-laptop:~# iptables -A INPUT -p udp --dport 5001 -j ACCEPT
root@fred-laptop:~# iperf -s -u -i 1
```

Note that the ip address should be replaced with the ip address of the server and the 2m replaced with the target test bandwidth. The MTU size is fixed at 1500 bytes.

### Test#1 Results:

```
Results for 2Mbps:
[ 4] local 192.168.97.10 port 5001 connected with 192.168.217.53 port 36083
[ 4] 0.0- 1.0 sec    244 KBytes  2.00 Mbits/sec  0.667 ms    0/ 170 (0%)
[ 4] 1.0- 2.0 sec    243 KBytes  1.99 Mbits/sec  0.769 ms    0/ 169 (0%)
[ 4] 2.0- 3.0 sec    245 KBytes  2.01 Mbits/sec  0.491 ms    0/ 171 (0%)
[ 4] 3.0- 4.0 sec    244 KBytes  2.00 Mbits/sec  0.412 ms    0/ 170 (0%)
[ 4] 4.0- 5.0 sec    244 KBytes  2.00 Mbits/sec  0.493 ms    0/ 170 (0%)
[ 4] 5.0- 6.0 sec    244 KBytes  2.00 Mbits/sec  0.401 ms    0/ 170 (0%)
[ 4] 6.0- 7.0 sec    244 KBytes  2.00 Mbits/sec  0.522 ms    0/ 170 (0%)
[ 4] 7.0- 8.0 sec    244 KBytes  2.00 Mbits/sec  0.450 ms    0/ 170 (0%)
[ 4] 8.0- 9.0 sec    244 KBytes  2.00 Mbits/sec  0.363 ms    0/ 170 (0%)
[ 4] 9.0-10.0 sec   244 KBytes  2.00 Mbits/sec  0.328 ms    0/ 170 (0%)
[ 4] 0.0-10.0 sec   2.39 MBytes  2.00 Mbits/sec  0.296 ms    0/ 1702 (0%)

Results for 4Mbps:
```

```

[ 3] local 192.168.97.10 port 5001 connected with 192.168.217.53 port 46772
[ 3] 0.0- 1.0 sec    488 KBytes  4.00 Mbits/sec  0.403 ms    0/ 340 (0%)
[ 3] 1.0- 2.0 sec    488 KBytes  4.00 Mbits/sec  0.287 ms    0/ 340 (0%)
[ 3] 2.0- 3.0 sec    488 KBytes  4.00 Mbits/sec  0.365 ms    0/ 340 (0%)
[ 3] 3.0- 4.0 sec    488 KBytes  4.00 Mbits/sec  0.413 ms    0/ 340 (0%)
[ 3] 4.0- 5.0 sec    490 KBytes  4.01 Mbits/sec  0.513 ms    0/ 341 (0%)
[ 3] 5.0- 6.0 sec    488 KBytes  4.00 Mbits/sec  0.362 ms    0/ 340 (0%)
[ 3] 6.0- 7.0 sec    488 KBytes  4.00 Mbits/sec  0.707 ms    0/ 340 (0%)
[ 3] 7.0- 8.0 sec    488 KBytes  4.00 Mbits/sec  0.578 ms    0/ 340 (0%)
[ 3] 8.0- 9.0 sec    490 KBytes  4.01 Mbits/sec  0.288 ms    0/ 341 (0%)
[ 3] 9.0-10.0 sec    488 KBytes  4.00 Mbits/sec  0.270 ms    0/ 340 (0%)
[ 3] 0.0-10.0 sec   4.77 MBytes  4.00 Mbits/sec  0.269 ms    0/ 3402 (0%)
[ 3] 0.0-10.0 sec    1 datagrams received out-of-order

```

Results for 6Mbps:

```

[ 4] local 192.168.97.10 port 5001 connected with 192.168.217.53 port 49178
[ 4] 0.0- 1.0 sec    732 KBytes  6.00 Mbits/sec  0.461 ms    0/ 510 (0%)
[ 4] 1.0- 2.0 sec    732 KBytes  6.00 Mbits/sec  0.356 ms    0/ 510 (0%)
[ 4] 2.0- 3.0 sec    731 KBytes  5.99 Mbits/sec  0.467 ms    0/ 509 (0%)
[ 4] 3.0- 4.0 sec    735 KBytes  6.02 Mbits/sec  0.347 ms    0/ 512 (0%)
[ 4] 4.0- 5.0 sec    732 KBytes  6.00 Mbits/sec  0.500 ms    0/ 510 (0%)
[ 4] 5.0- 6.0 sec    731 KBytes  5.99 Mbits/sec  0.443 ms    1/ 510 (0.2%)
[ 4] 6.0- 7.0 sec    734 KBytes  6.01 Mbits/sec  0.597 ms    0/ 511 (0%)
[ 4] 7.0- 8.0 sec    731 KBytes  5.99 Mbits/sec  0.721 ms    0/ 509 (0%)
[ 4] 8.0- 9.0 sec    734 KBytes  6.01 Mbits/sec  0.266 ms    0/ 511 (0%)
[ 4] 9.0-10.0 sec    732 KBytes  6.00 Mbits/sec  0.372 ms    0/ 510 (0%)
[ 4] 0.0-10.0 sec   7.15 MBytes  6.00 Mbits/sec  0.368 ms    1/ 5104 (0.02%)

```

Results for 8Mbps:

```

[ 3] local 192.168.97.10 port 5001 connected with 192.168.217.53 port 50987
[ 3] 0.0- 1.0 sec    976 KBytes  8.00 Mbits/sec  0.332 ms    0/ 680 (0%)
[ 3] 1.0- 2.0 sec    973 KBytes  7.97 Mbits/sec  0.351 ms    0/ 678 (0%)
[ 3] 2.0- 3.0 sec    980 KBytes  8.03 Mbits/sec  0.407 ms    0/ 683 (0%)
[ 3] 3.0- 4.0 sec    843 KBytes  6.90 Mbits/sec  0.367 ms    93/ 680 (14%)
[ 3] 4.0- 5.0 sec    561 KBytes  4.60 Mbits/sec  0.691 ms   289/ 680 (42%)
[ 3] 5.0- 6.0 sec    481 KBytes  3.94 Mbits/sec  0.673 ms   345/ 680 (51%)
[ 3] 6.0- 7.0 sec    359 KBytes  2.94 Mbits/sec  0.698 ms   429/ 679 (63%)
[ 3] 7.0- 8.0 sec    360 KBytes  2.95 Mbits/sec  0.790 ms   431/ 682 (63%)
[ 3] 8.0- 9.0 sec    359 KBytes  2.94 Mbits/sec  0.951 ms   428/ 678 (63%)
[ 3] 9.0-10.0 sec    359 KBytes  2.94 Mbits/sec  0.607 ms   432/ 682 (63%)
[ 3] 0.0-10.3 sec    6.11 MBytes  5.00 Mbits/sec  16.206 ms  2448/ 6804 (36%)

```

The test on this link ran clean from 2Mbps and 4Mbps but started to show loss at 6Mbps. At 8Mbps you can clearly see the policer force the inbound stream down to 3Mbps.

The link under test was purchased with a 6Mbps cap.

## Test#2 (upstream bandwidth):

The object of this test is to determine the amount of available bandwidth outbound from the site under test. The secondary objective is to determine the failure mechanism when the bandwidth limit is reached. For this test we will use IPERF to run a UDP test over a one minute interval. We will start with a 1Mbps stream and increase in 1Mbps intervals until we start to see packet loss or a significant increase in network latency.

The command on the client side to initiate the test will be:

```
iperf -c 192.168.97.10 -u -b 1m -M 1500
```

## Test#2 Results:

Results for 4Mbps:

```
[ 4] local 192.168.217.53 port 5001 connected with 192.168.97.10 port 57857
[ 4] 0.0- 1.0 sec 488 KBytes 4.00 Mbits/sec 0.692 ms 0/ 340 (0%)
[ 4] 1.0- 2.0 sec 488 KBytes 4.00 Mbits/sec 0.644 ms 0/ 340 (0%)
[ 4] 2.0- 3.0 sec 488 KBytes 4.00 Mbits/sec 0.631 ms 0/ 340 (0%)
[ 4] 3.0- 4.0 sec 488 KBytes 4.00 Mbits/sec 0.290 ms 0/ 340 (0%)
[ 4] 4.0- 5.0 sec 488 KBytes 4.00 Mbits/sec 0.329 ms 0/ 340 (0%)
[ 4] 5.0- 6.0 sec 488 KBytes 4.00 Mbits/sec 0.430 ms 0/ 340 (0%)
[ 4] 6.0- 7.0 sec 488 KBytes 4.00 Mbits/sec 0.502 ms 0/ 340 (0%)
[ 4] 7.0- 8.0 sec 488 KBytes 4.00 Mbits/sec 0.237 ms 0/ 340 (0%)
[ 4] 8.0- 9.0 sec 488 KBytes 4.00 Mbits/sec 0.483 ms 0/ 340 (0%)
[ 4] 9.0-10.0 sec 488 KBytes 4.00 Mbits/sec 0.484 ms 0/ 340 (0%)
[ 4] 0.0-10.0 sec 4.77 MBytes 4.00 Mbits/sec 0.762 ms 0/ 3402 (0%)
[ 4] 0.0-10.0 sec 1 datagrams received out-of-order
```

Results for 6Mbps:

```
[ 3] local 192.168.217.53 port 5001 connected with 192.168.97.10 port 53764
[ 3] 0.0- 1.0 sec 734 KBytes 6.01 Mbits/sec 1.279 ms 0/ 511 (0%)
[ 3] 1.0- 2.0 sec 731 KBytes 5.99 Mbits/sec 1.036 ms 0/ 509 (0%)
[ 3] 2.0- 3.0 sec 734 KBytes 6.01 Mbits/sec 1.104 ms 0/ 511 (0%)
[ 3] 3.0- 4.0 sec 732 KBytes 6.00 Mbits/sec 1.156 ms 0/ 510 (0%)
[ 3] 4.0- 5.0 sec 732 KBytes 6.00 Mbits/sec 1.021 ms 0/ 510 (0%)
[ 3] 5.0- 6.0 sec 732 KBytes 6.00 Mbits/sec 1.181 ms 0/ 510 (0%)
[ 3] 6.0- 7.0 sec 732 KBytes 6.00 Mbits/sec 1.226 ms 0/ 510 (0%)
[ 3] 7.0- 8.0 sec 732 KBytes 6.00 Mbits/sec 1.089 ms 0/ 510 (0%)
[ 3] 8.0- 9.0 sec 732 KBytes 6.00 Mbits/sec 1.263 ms 0/ 510 (0%)
[ 3] 9.0-10.0 sec 732 KBytes 6.00 Mbits/sec 0.972 ms 0/ 510 (0%)
[ 3] 0.0-10.0 sec 7.16 MBytes 6.00 Mbits/sec 1.001 ms 0/ 5103 (0%)
[ 3] 0.0-10.0 sec 1 datagrams received out-of-order
```

Results for 8Mbps:

```
[ 4] local 192.168.217.53 port 5001 connected with 192.168.97.10 port 45830
[ 4] 0.0- 1.0 sec 976 KBytes 8.00 Mbits/sec 1.247 ms 0/ 680 (0%)
[ 4] 1.0- 2.0 sec 976 KBytes 8.00 Mbits/sec 1.175 ms 0/ 680 (0%)
[ 4] 2.0- 3.0 sec 978 KBytes 8.01 Mbits/sec 1.294 ms 0/ 681 (0%)
[ 4] 3.0- 4.0 sec 804 KBytes 6.59 Mbits/sec 0.808 ms 121/ 681 (18%)
[ 4] 4.0- 5.0 sec 533 KBytes 4.36 Mbits/sec 0.455 ms 305/ 676 (45%)
[ 4] 5.0- 6.0 sec 435 KBytes 3.56 Mbits/sec 1.348 ms 378/ 681 (56%)
[ 4] 6.0- 7.0 sec 345 KBytes 2.82 Mbits/sec 0.921 ms 442/ 682 (65%)
[ 4] 7.0- 8.0 sec 343 KBytes 2.81 Mbits/sec 1.258 ms 440/ 679 (65%)
```



```

[ 4] 8.0- 9.0 sec    343 KBytes  2.81 Mbits/sec  1.171 ms  442/ 681 (65%)
[ 4] 9.0-10.0 sec   343 KBytes  2.81 Mbits/sec  0.962 ms  440/ 679 (65%)
[ 4] 0.0-10.5 sec   5.94 MBytes  4.74 Mbits/sec  32.193 ms 2569/ 6803 (38%)
[ 4] 0.0-10.5 sec    1 datagrams received out-of-order

```

The test on this link ran clean from 2Mbps through 6Mbps. At 8Mbps you can clearly see the policer force the outbound stream down to 3Mbps.

The link under test was purchased with a 6Mbps cap.

### Test#3 (upstream bandwidth):

This is a repeat of test #2 but the result will be quite different. In this case we are going to be dealing with a shaper instead of a policer.

The command on the client side to initiate the test will be:

```
iperf -c 192.168.53.11 -u -b 1m -M 1500
```

### Test#3 Results:

Results for 4Mbps:

```

[ 3] local 192.168.217.53 port 5001 connected with 192.168.53.11 port 44700
[ ID] Interval      Transfer      Bandwidth      Jitter    Lost/Total Datagrams
[ 3] 0.0- 1.0 sec    274 KBytes    2.25 Mbits/sec  4.739 ms   0/ 191 (0%)
[ 3] 1.0- 2.0 sec    240 KBytes    1.96 Mbits/sec  4.434 ms   0/ 167 (0%)
[ 3] 2.0- 3.0 sec    241 KBytes    1.98 Mbits/sec  4.424 ms   0/ 168 (0%)
[ 3] 3.0- 4.0 sec    223 KBytes    1.82 Mbits/sec  5.067 ms   0/ 155 (0%)
[ 3] 4.0- 5.0 sec    227 KBytes    1.86 Mbits/sec  4.500 ms   0/ 158 (0%)
[ 3] 5.0- 6.0 sec    230 KBytes    1.88 Mbits/sec  4.803 ms   0/ 160 (0%)
[ 3] 6.0- 7.0 sec    215 KBytes    1.76 Mbits/sec  5.216 ms   0/ 150 (0%)
[ 3] 7.0- 8.0 sec    223 KBytes    1.82 Mbits/sec  5.739 ms   0/ 155 (0%)
[ 3] 8.0- 9.0 sec    225 KBytes    1.85 Mbits/sec  5.363 ms   0/ 157 (0%)
[ 3] 9.0-10.0 sec    231 KBytes    1.89 Mbits/sec  4.555 ms   0/ 161 (0%)
[ 3] 10.0-11.0 sec   231 KBytes    1.89 Mbits/sec  4.409 ms   0/ 161 (0%)
[ 3] 11.0-12.0 sec   238 KBytes    1.95 Mbits/sec  4.839 ms   0/ 166 (0%)
[ 3] 12.0-13.0 sec   228 KBytes    1.87 Mbits/sec  4.685 ms   0/ 159 (0%)
[ 3] 13.0-14.0 sec   230 KBytes    1.88 Mbits/sec  4.688 ms   0/ 160 (0%)
[ 3] 14.0-15.0 sec   230 KBytes    1.88 Mbits/sec  4.287 ms   0/ 160 (0%)
[ 3] 15.0-16.0 sec   247 KBytes    2.02 Mbits/sec  5.189 ms   0/ 172 (0%)
[ 3] 16.0-17.0 sec   234 KBytes    1.92 Mbits/sec  4.568 ms   0/ 163 (0%)
[ 3] 17.0-18.0 sec   266 KBytes    2.18 Mbits/sec  4.167 ms   0/ 185 (0%)
[ 3] 18.0-19.0 sec   273 KBytes    2.23 Mbits/sec  4.100 ms   0/ 190 (0%)
[ 3] 19.0-20.0 sec   286 KBytes    2.34 Mbits/sec  4.278 ms   0/ 199 (0%)
[ 3] 0.0-20.3 sec   4.77 MBytes    1.97 Mbits/sec  3.859 ms   0/ 3402 (0%)

```

Results for 6Mbps:

```

[ 4] local 192.168.217.53 port 5001 connected with 192.168.53.11 port 34340
[ 4] 0.0- 1.0 sec    281 KBytes    2.30 Mbits/sec  5.023 ms   1/ 197 (0.51%)
[ 4] 1.0- 2.0 sec    234 KBytes    1.92 Mbits/sec  4.980 ms   0/ 163 (0%)
[ 4] 2.0- 3.0 sec    221 KBytes    1.81 Mbits/sec  5.498 ms   0/ 154 (0%)

```

```

[ 4] 3.0- 4.0 sec    223 KBytes  1.82 Mbits/sec  5.318 ms    0/ 155 (0%)
[ 4] 4.0- 5.0 sec    231 KBytes  1.89 Mbits/sec  4.875 ms    0/ 161 (0%)
[ 4] 5.0- 6.0 sec    250 KBytes  2.05 Mbits/sec  4.630 ms    0/ 174 (0%)
[ 4] 6.0- 7.0 sec    297 KBytes  2.43 Mbits/sec  2.776 ms    0/ 207 (0%)
[ 4] 7.0- 8.0 sec    392 KBytes  3.21 Mbits/sec  2.418 ms    0/ 273 (0%)
[ 4] 8.0- 9.0 sec    398 KBytes  3.26 Mbits/sec  2.342 ms    0/ 277 (0%)
[ 4] 9.0-10.0 sec    416 KBytes  3.41 Mbits/sec  2.748 ms    0/ 290 (0%)
[ 4] 10.0-11.0 sec   423 KBytes  3.47 Mbits/sec  2.410 ms    0/ 295 (0%)
[ 4] 11.0-12.0 sec   415 KBytes  3.40 Mbits/sec  2.393 ms    0/ 289 (0%)
[ 4] 12.0-13.0 sec   422 KBytes  3.46 Mbits/sec  2.242 ms    0/ 294 (0%)
[ 4] 13.0-14.0 sec   441 KBytes  3.61 Mbits/sec  2.042 ms    0/ 307 (0%)
[ 4] 14.0-15.0 sec   438 KBytes  3.59 Mbits/sec  2.329 ms    0/ 305 (0%)
[ 4] 15.0-16.0 sec   434 KBytes  3.55 Mbits/sec  2.406 ms    0/ 302 (0%)
[ 4] 16.0-17.0 sec   423 KBytes  3.47 Mbits/sec  2.428 ms    0/ 295 (0%)
[ 4] 17.0-18.0 sec   454 KBytes  3.72 Mbits/sec  2.515 ms    0/ 316 (0%)
[ 4] 18.0-19.0 sec   435 KBytes  3.56 Mbits/sec  2.400 ms    0/ 303 (0%)
[ 4] 19.0-20.0 sec   423 KBytes  3.47 Mbits/sec  2.152 ms    0/ 295 (0%)
[ 4] 0.0-20.2 sec   7.15 MBytes 2.98 Mbits/sec  2.084 ms    1/ 5104 (0.02%)

```

Results for 8Mbps:

```

[ 3] local 192.168.217.53 port 5001 connected with 192.168.53.11 port 32886
[ 3] 0.0- 1.0 sec    586 KBytes  4.80 Mbits/sec  2.469 ms    8/ 416 (1.9%)
[ 3] 1.0- 2.0 sec    438 KBytes  3.59 Mbits/sec  2.376 ms    0/ 305 (0%)
[ 3] 2.0- 3.0 sec    438 KBytes  3.59 Mbits/sec  2.445 ms    0/ 305 (0%)
[ 3] 3.0- 4.0 sec    438 KBytes  3.59 Mbits/sec  2.402 ms    0/ 305 (0%)
[ 3] 4.0- 5.0 sec    436 KBytes  3.58 Mbits/sec  2.738 ms    0/ 304 (0%)
[ 3] 5.0- 6.0 sec    439 KBytes  3.60 Mbits/sec  2.372 ms    0/ 306 (0%)
[ 3] 6.0- 7.0 sec    438 KBytes  3.59 Mbits/sec  2.337 ms    0/ 305 (0%)
[ 3] 7.0- 8.0 sec    438 KBytes  3.59 Mbits/sec  2.189 ms    0/ 305 (0%)
[ 3] 8.0- 9.0 sec    438 KBytes  3.59 Mbits/sec  2.532 ms    0/ 305 (0%)
[ 3] 9.0-10.0 sec    438 KBytes  3.59 Mbits/sec  2.316 ms    0/ 305 (0%)
[ 3] 10.0-11.0 sec   438 KBytes  3.59 Mbits/sec  2.304 ms    0/ 305 (0%)
[ 3] 11.0-12.0 sec   436 KBytes  3.58 Mbits/sec  2.493 ms    0/ 304 (0%)
[ 3] 12.0-13.0 sec   438 KBytes  3.59 Mbits/sec  2.459 ms    0/ 305 (0%)
[ 3] 13.0-14.0 sec   438 KBytes  3.59 Mbits/sec  2.254 ms    0/ 305 (0%)
[ 3] 14.0-15.0 sec   438 KBytes  3.59 Mbits/sec  2.366 ms    0/ 305 (0%)
[ 3] 15.0-16.0 sec   438 KBytes  3.59 Mbits/sec  2.350 ms    0/ 305 (0%)
[ 3] 16.0-17.0 sec   438 KBytes  3.59 Mbits/sec  2.345 ms    0/ 305 (0%)
[ 3] 17.0-18.0 sec   438 KBytes  3.59 Mbits/sec  2.542 ms    0/ 305 (0%)
[ 3] 18.0-19.0 sec   395 KBytes  3.23 Mbits/sec  4.136 ms   360/ 635 (57%)
[ 3] 0.0-19.9 sec   8.59 MBytes 3.62 Mbits/sec  3.489 ms   673/ 6801 (9.9%)

```

The test on this link seems to run clean from 2Mbps through 8Mbps. The printout below is from a ping test running in parallel with the 8Mbps outbound iperf test. Notice how the ping delay increases to over 9 seconds when we stress the link. The shaper is buffering and delaying the extra data.

```

root@fred-laptop:~# ping 192.168.53.11
PING 192.168.53.11 (192.168.53.11) 56(84) bytes of data.
64 bytes from 192.168.53.11: icmp_seq=1 ttl=62 time=108 ms
From 192.168.216.1: icmp_seq=2 Redirect Host(New nexthop: 192.168.217.8)
64 bytes from 192.168.53.11: icmp_seq=2 ttl=62 time=108 ms
64 bytes from 192.168.53.11: icmp_seq=3 ttl=62 time=109 ms
64 bytes from 192.168.53.11: icmp_seq=4 ttl=62 time=110 ms
64 bytes from 192.168.53.11: icmp_seq=5 ttl=62 time=109 ms
64 bytes from 192.168.53.11: icmp_seq=6 ttl=62 time=829 ms
64 bytes from 192.168.53.11: icmp_seq=7 ttl=62 time=2059 ms
64 bytes from 192.168.53.11: icmp_seq=8 ttl=62 time=3286 ms
64 bytes from 192.168.53.11: icmp_seq=9 ttl=62 time=4527 ms
64 bytes from 192.168.53.11: icmp_seq=10 ttl=62 time=5755 ms
64 bytes from 192.168.53.11: icmp_seq=11 ttl=62 time=6999 ms
64 bytes from 192.168.53.11: icmp_seq=12 ttl=62 time=8227 ms

```

```
64 bytes from 192.168.53.11: icmp_seq=13 ttl=62 time=9456 ms
64 bytes from 192.168.53.11: icmp_seq=16 ttl=62 time=9204 ms
64 bytes from 192.168.53.11: icmp_seq=17 ttl=62 time=8216 ms
64 bytes from 192.168.53.11: icmp_seq=18 ttl=62 time=7228 ms
64 bytes from 192.168.53.11: icmp_seq=19 ttl=62 time=6227 ms
64 bytes from 192.168.53.11: icmp_seq=20 ttl=62 time=5227 ms
64 bytes from 192.168.53.11: icmp_seq=21 ttl=62 time=4226 ms
64 bytes from 192.168.53.11: icmp_seq=22 ttl=62 time=3222 ms
64 bytes from 192.168.53.11: icmp_seq=23 ttl=62 time=2221 ms
64 bytes from 192.168.53.11: icmp_seq=24 ttl=62 time=1213 ms
64 bytes from 192.168.53.11: icmp_seq=25 ttl=62 time=212 ms
64 bytes from 192.168.53.11: icmp_seq=26 ttl=62 time=108 ms
64 bytes from 192.168.53.11: icmp_seq=27 ttl=62 time=110 ms
64 bytes from 192.168.53.11: icmp_seq=28 ttl=62 time=111 ms
64 bytes from 192.168.53.11: icmp_seq=29 ttl=62 time=111 ms
```

The link under test was purchased with a 4Mbps cap. In this case the shaper is causing more trouble by introducing the latency than a policer would by just throwing packets away. In the diagram below “TCS” stands for “TeleCollaborationSystem” and “VPN” refers to the VPN router that connects the two network segments over the internet.



TCS-B is connected via a known good 100 Mbps link. TCS-A is connected to the link we will be testing. Note that TCS-A and TCS-B could be replaced by any Linux system with the appropriate tools installed. Depending on the type of VPN router installed, the tools may also be located there eliminating the need for additional test systems.

## Summary

The objective of this document was to provide reader with a way to characterize links. We have seen how links can be limited using shapers and policers. We learned how to detect the presence of these devices and how to measure their effects. We also demonstrated how to detect and measure packet loss.

The appendices that follow contain tutorials on the tools that were used in the preceding examples. Reviewing these tutorials is highly recommended.

## Appendix A: NUTTCP

NUTTCP is available in source and binary form. Building the source is only necessary if a binary version is not available for your operating system. This should not be the case for Linux based systems.

### Installing NUTTCP

To install NUTTCP on an Ubuntu system, you must have a route to the repositories on the Internet. If you can ping “www.google.com” you should be able to proceed.

The command to install the binary will be:

```
sudo apt-get install nuttcp
```

Note that although this works for systems running Ubuntu 10.04, nuttcp is not available for systems running older operating systems like ubuntu 8.04. In those cases you will have to download and build the source.

### Using NUTTCP

NUTTCP is made up of two parts, a client and a server. It is good practice to run the server on all end-points. To start the server type the following case sensitive command.

```
nuttcp -S
```

All Magor systems ship with a firewall that must be told to allow the nuttcp packets through. The following commands temporarily open up the ports on the firewall required to do our testing. When the system is restarted this ports will no longer be open. There is one command for TCP testing and one for UDP testing.

```
iptables -A INPUT -p tcp --dport 5000 -j ACCEPT
iptables -A INPUT -p udp --dport 5001 -j ACCEPT
```

Now that the firewall has been prepared and the server has been started, we are ready to test. NUTTCP allows us to make measurements in one direction at a time and this is important because most links we encounter will not be symmetrical. In order to keep things simple we will not review the entire list of options for the NUTTCP command. Variations on the following command will be all you need to do your testing.

```
nuttcp -u -I -Ri2m 192.168.217.80
```

Lets take this command apart to understand it a bit better.

“nuttcp” launches the client.

“-u” tells the client that we want the transmission to be UDP.

The only other option here is “-t” for TCP.

“-l” tells the client that we want interval reports every second. The default for interval reports is one second so it is not necessary to specify a number. If you wanted 2 second interval reports you would replace the “-l” with “-l2”.

“-R2M” tells the client that the transmit Rate will be 2Mbps. To increase the rate to 10Mbps the command would be “-R10m”

“192.168.217.80” tells the client the IP address of the remote server we want it to send the test stream to.

One of the nice things about NUTTCP is that all of the commands and reports are available at the same end. Once you start the server at the far end there is no reason to remain connected or monitor the remote node. The following examples will illustrate some of the most useful commands.

## NUTTCP Examples

For all of the examples that follow it is assumed that all of the remote locations being tested are already running the nuttcp server and the firewall has been configured to allow the appropriate connections.

### Example 1:

This first example is a 10 second transfer of TCP data. The numbers associated with %TX and %RX are the CPU utilization for the transmitter and receiver. “retrans” indicates how many re-transmissions were required and msRTT indicates the round trip delay in milliseconds.

```
root@fred-laptop:~# nuttcp 192.168.217.27
q = 85.2588 Mbps 2 %TX 2 %RX 0 retrans 0.20 msRTT
```

### Example 2:

This next example uses the “-w” option to set the window size. Note that 6m is actually  $6 * 1024 * 1024 = 6291456$  bytes. Notice how the throughput increases with the larger window size.

```

root@fred-laptop:~# nuttcp -w6m 192.168.217.27
nuttcp-t: Warning: send window size 131071 < requested window size 6291456
nuttcp-r: Warning: receive window size 131071 < requested window size 6291456
112.3339 MB / 10.01 sec = 94.1070 Mbps 2 %TX 3 %RX 0 retrans 0.25 msRTT

```

### Example 3:

This next example uses the “-u” option to change the data mode to use UDP. The “-i” requests interim reports every second and the “-Ri4m” sets and limits the instantaneous data to 4mbps. If we had selected “R4m” the system the average rate would be 4mbps but it may peak above that. The UDP transport means that there will be no retransmissions making it easy to record lost packets. By default this test will use 8192 byte buffers. This is perhaps the most useful test for determining the upstream and downstream bandwidth.

```

root@fred-laptop:~# nuttcp -u -i -Ri4m 192.168.217.27
0.4766 MB / 1.00 sec = 3.9976 Mbps 0 / 488 ~drop/pkt 0.00 ~%loss
0.4766 MB / 1.00 sec = 3.9977 Mbps 0 / 488 ~drop/pkt 0.00 ~%loss
0.4766 MB / 1.00 sec = 3.9977 Mbps 0 / 488 ~drop/pkt 0.00 ~%loss
0.4775 MB / 1.00 sec = 4.0059 Mbps 0 / 489 ~drop/pkt 0.00 ~%loss
0.4766 MB / 1.00 sec = 3.9977 Mbps 0 / 488 ~drop/pkt 0.00 ~%loss
0.4766 MB / 1.00 sec = 3.9977 Mbps 0 / 488 ~drop/pkt 0.00 ~%loss
0.4775 MB / 1.00 sec = 4.0059 Mbps 0 / 489 ~drop/pkt 0.00 ~%loss
0.4766 MB / 1.00 sec = 3.9977 Mbps 0 / 488 ~drop/pkt 0.00 ~%loss
0.4775 MB / 1.00 sec = 4.0059 Mbps 0 / 489 ~drop/pkt 0.00 ~%loss

4.7676 MB / 10.00 sec = 4.0000 Mbps 99 %TX 0 %RX 0 / 4882 drop/pkt 0.00 %loss

```

### Example 4:

This last example is good for determining the ability of the server to deliver packets. The local address of 127.0.0.1 means the packets generated will not actually go out on the wire. This means that bus speed and NIC hardware will not be part of the test, but the CPU performance will be indicated.

```

root@fred-laptop:~# nuttcp -wlm 127.0.0.1
nuttcp-t: Warning: send window size 131071 < requested window size 1048576
nuttcp-r: Warning: receive window size 131071 < requested window size 1048576
33944.1250 MB / 10.05 sec = 28338.0050 Mbps 97 %TX 75 %RX 0 retrans 0.05 msRTT

```

### Other nuttcp Options:

The following is a list of useful options that can be added to any nuttcp command.

- r runs the stream in the other direction, ie. From the server to the client
- v turns on verbose reporting (the old default style) with such things as the used window size
- F this flips the connect, which you don't need to understand. What you need to know is that if you are behind a NAT, or a port block, this allows you to do a receive by doing -r -F.
- R[i]<rate> this limits the rate of the transmitter. This is important if you really don't want to use the maximum bandwidth that a host can

produce, especially when running udp transfers. The optional extra 'i' is to have the rate limit apply instantaneously as opposed to an average which may go above and below the limit.

## Server Options:

Most of the time the server will be run using the single “-S” option. The server does have a couple of options however to change the port number and/or run using IP version 6. If you wanted to run two servers on the same system you would have to start the second server with a different port. The “-6” option starts the server with IPV6. The “-P5008” option starts a server on port 5008.

```
nuttcp -S -P5008  
nuttcp -S -6 -P5008
```



## Appendix B: IPERF

IPERF is another network testing tool that is available in binary form for a number of operating systems. It is particularly attractive to us because it is available for Ubuntu but also for OpenWRT which is a popular OS for small VPN routers.

By default IPERF tests are run using TCP and clients connect to servers on port 5001 and the test results are for the link from the client to the server. For bi-directional tests use the “-d” and “-r” options.

### Installing IPERF

To install IPERF on an Ubuntu based system, you must have a route to the repositories on the Internet. If you can ping “www.google.com” you should be able to proceed.

The command to install the binary will be:

```
sudo apt-get install iperf
```

### Using IPERF

IPERF has some features that make it very useful for testing jitter and out of order packets. Ports on the server side firewall must be opened to accept connections from clients. Use the following command to setup the firewall.

```
iptables -A INPUT -p tcp --dport 5001 -j ACCEPT
iptables -A INPUT -p udp --dport 5001 -j ACCEPT
```

### IPERF Examples

#### Example 1:

This first example is a basic client server TCP network link test. It shows the basic syntax for the iperf commands and the most basic form of output. This test was done between two systems connected by a 100Mbps switch. The measured throughput was 94 Mbps.

#### Client side:

```
root@fred-laptop:~# iperf -c 192.168.217.27
-----
Client connecting to 192.168.217.27, TCP port 5001
TCP window size: 16.0 KByte (default)
-----
```

```
[ 3] local 192.168.217.53 port 53773 connected with 192.168.217.27 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-10.0 sec    112 MBytes  94.2 Mbits/sec
```

### Server side:

```
root@spiderman:/tmp# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 4] local 192.168.217.27 port 5001 connected with 192.168.217.53 port 53773
[ 4] 0.0-10.0 sec    112 MBytes  94.0 Mbits/sec
```

### Example 2:

In this example we will use the “-r” switch to force a bi-directional test. Using “-r” causes the tests to be run sequentially where a “-d” argument would have caused the tests to be run at the same time.

### Client side:

```
root@fred-laptop:~# iperf -c 192.168.217.27 -r
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
-----
Client connecting to 192.168.217.27, TCP port 5001
TCP window size: 161 KByte (default)
-----
[ 3] local 192.168.217.53 port 41355 connected with 192.168.217.27 port 5001
Waiting for server threads to complete. Interrupt again to force quit.
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-10.0 sec    112 MBytes  94.2 Mbits/sec
```

### Server side:

```
root@spiderman:/tmp# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
-----
Client connecting to 192.168.217.53, TCP port 5001
TCP window size: 16.0 KByte (default)
-----
[ 4] local 192.168.217.27 port 56498 connected with 192.168.217.53 port 5001
[ 4] 0.0- 0.0 sec    15.6 KBytes  57.6 Mbits/sec
```

### Example 3:

In this example we use the “-p” command to change the communication port, the “-t” command to change the test duration and the “-i” command to set the test interval. Note that the firewall on the server side will have to be setup to allow the connection on the new port.

## Client side:

```
root@fred-laptop:~# iperf -c 192.168.217.27 -p 14000 -t 20 -i 2
-----
Client connecting to 192.168.217.27, TCP port 14000
TCP window size: 16.0 KByte (default)
-----
[ 3] local 192.168.217.53 port 52907 connected with 192.168.217.27 port 14000
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0- 2.0 sec  22.6 MBytes  94.7 Mbits/sec
[ 3] 2.0- 4.0 sec  22.5 MBytes  94.6 Mbits/sec
[ 3] 4.0- 6.0 sec  22.4 MBytes  93.8 Mbits/sec
[ 3] 6.0- 8.0 sec  22.5 MBytes  94.4 Mbits/sec
[ 3] 8.0-10.0 sec  22.4 MBytes  93.9 Mbits/sec
[ 3] 10.0-12.0 sec 22.4 MBytes  94.0 Mbits/sec
[ 3] 12.0-14.0 sec 22.4 MBytes  93.8 Mbits/sec
[ 3] 14.0-16.0 sec 22.5 MBytes  94.4 Mbits/sec
[ 3] 16.0-18.0 sec 22.4 MBytes  93.9 Mbits/sec
[ 3] 18.0-20.0 sec 22.4 MBytes  94.1 Mbits/sec
[ 3] 0.0-20.0 sec   224 MBytes  94.1 Mbits/sec
```

## Server side:

```
root@spiderman:/tmp# iptables -A INPUT -p tcp --dport 14000 -j ACCEPT
root@spiderman:/tmp# iperf -s -p 14000
-----
Server listening on TCP port 14000
TCP window size: 85.3 KByte (default)
-----
[ 4] local 192.168.217.27 port 14000 connected with 192.168.217.53 port 52907
[ 4] 0.0-20.0 sec   224 MBytes  94.1 Mbits/sec
```

## Example 4:

In this example we use the “-u” command to change the communication protocol to UDP. UDP does not have a mechanism for the retransmission of packets so packet loss is more of an issue. The “-b” command allows us to tell the client how much bandwidth we want it to consume. The “-i” argument on the server side sets the reporting interval to 1 second.

## Client side:

```
root@fred-laptop:~# iperf -c 192.168.217.27 -u -b 10m
-----
Client connecting to 192.168.217.27, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 122 KByte (default)
-----
[ 3] local 192.168.217.53 port 50480 connected with 192.168.217.27 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec
[ 3] Sent 8505 datagrams
[ 3] Server Report:
[ 3] 0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec  0.002 ms    0/ 8504 (0%)
[ 3] 0.0-10.0 sec   1 datagrams received out-of-order
```

### Server side:

```
root@spiderman:/tmp# iperf -s -u -i 1
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 122 KByte (default)
-----
[ 3] local 192.168.217.27 port 5001 connected with 192.168.217.53 port 50480
[ 3] 0.0- 1.0 sec 1.19 MBytes 10.0 Mbits/sec 0.003 ms 0/ 850 (0%)
[ 3] 1.0- 2.0 sec 1.19 MBytes 10.0 Mbits/sec 0.004 ms 0/ 851 (0%)
[ 3] 2.0- 3.0 sec 1.19 MBytes 10.0 Mbits/sec 0.003 ms 0/ 850 (0%)
[ 3] 3.0- 4.0 sec 1.19 MBytes 10.0 Mbits/sec 0.003 ms 0/ 850 (0%)
[ 3] 4.0- 5.0 sec 1.19 MBytes 10.0 Mbits/sec 0.003 ms 0/ 851 (0%)
[ 3] 5.0- 6.0 sec 1.19 MBytes 10.0 Mbits/sec 0.181 ms 0/ 850 (0%)
[ 3] 6.0- 7.0 sec 1.19 MBytes 10.0 Mbits/sec 0.002 ms 0/ 851 (0%)
[ 3] 7.0- 8.0 sec 1.19 MBytes 10.0 Mbits/sec 0.002 ms 0/ 850 (0%)
[ 3] 8.0- 9.0 sec 1.19 MBytes 10.0 Mbits/sec 0.003 ms 0/ 851 (0%)
[ 3] 9.0-10.0 sec 1.19 MBytes 10.0 Mbits/sec 0.002 ms 0/ 850 (0%)
[ 3] 0.0-10.0 sec 11.9 MBytes 10.0 Mbits/sec 0.002 ms 0/ 8504 (0%)
[ 3] 0.0-10.0 sec 1 datagrams received out-of-order
```

### Example 5:

In this example we use the “-m” command to test the network for its ability to support large packet sizes. IPREF defines a new metric called “Maximum Segment Size” which is the MTU size minus the TCP/IP headers. The maximum default MTU size for ethernet networks is 1500 bytes. TCP/IP headers are typically 40 bytes and timestamps can take 12 bytes, so the default MSS will be 1448 bytes. Making the MTU higher means less header information and more data throughput. We can use the “-M” option to set the MSS. Notice how the MTU size has been reduced in the second test below.

### Client side:

```
root@fred-laptop:~# iperf -c 192.168.217.27 -m
-----
Client connecting to 192.168.217.27, TCP port 5001
TCP window size: 16.0 KByte (default)
-----
[ 3] local 192.168.217.53 port 49858 connected with 192.168.217.27 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0-10.0 sec   113 MBytes   94.6 Mbits/sec
[ 3] MSS size 1448 bytes (MTU 1500 bytes, ethernet)
```

### Server side:

```
root@spiderman:/# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 4] local 192.168.217.27 port 5001 connected with 192.168.217.53 port 49858
[ 4] 0.0-10.1 sec   113 MBytes   94.1 Mbits/sec
```

### Client side:

```

root@fred-laptop:~# iperf -c 192.168.217.27 -M 1300 -m
WARNING: attempt to set TCP maximum segment size to 1300, but got 536
-----
Client connecting to 192.168.217.27, TCP port 5001
TCP window size: 16.0 KByte (default)
-----
[ 3] local 192.168.217.53 port 33749 connected with 192.168.217.27 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-10.0 sec   112 MBytes  93.9 Mbits/sec
[ 3] MSS size 1288 bytes (MTU 1328 bytes, unknown interface)

```

### Server side:

```

root@spiderman:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 4] local 192.168.217.27 port 5001 connected with 192.168.217.53 port 33749
[ 4] 0.0-10.1 sec   112 MBytes  93.4 Mbits/sec

```

### Example 6:

In this example we use the “-P” command to perform parallel tests. This provides us with a way to load the link with any number of tests where each test reports its results independently.

### Client side:

```

root@fred-laptop:~# iperf -c 192.168.217.27 -P 2
-----
Client connecting to 192.168.217.27, TCP port 5001
TCP window size: 16.0 KByte (default)
-----
[ 4] local 192.168.217.53 port 55099 connected with 192.168.217.27 port 5001
[ 3] local 192.168.217.53 port 55098 connected with 192.168.217.27 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.0-10.0 sec   54.9 MBytes  46.0 Mbits/sec
[ 3] 0.0-10.0 sec   58.1 MBytes  48.6 Mbits/sec
[SUM] 0.0-10.0 sec   113 MBytes  94.4 Mbits/sec

```

### Server side:

```

root@spiderman:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 4] local 192.168.217.27 port 5001 connected with 192.168.217.53 port 55098
[ 5] local 192.168.217.27 port 5001 connected with 192.168.217.53 port 55099
[ 5] 0.0-10.0 sec   54.9 MBytes  45.8 Mbits/sec
[ 4] 0.0-10.1 sec   58.1 MBytes  48.4 Mbits/sec
[SUM] 0.0-10.1 sec   113 MBytes  94.1 Mbits/sec

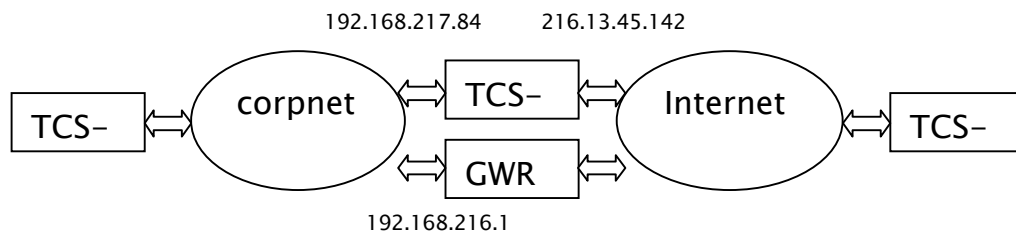
```

## Appendix C: Traceroute & Tracepath

It is sometimes necessary to use static routes to force network traffic to take a specific path. This is especially the case with some network topologies that use VPN's to create network tunnels. When static routes are used we need a tool that will allow us to verify the path that network traffic is taking. The names of the tools change from OS to OS, but the basic premise is the same. "traceroute" and "tracepath" will map out the routers between two IP addresses and print the result.

### Example 1:

In this first example we have a TeleCollaboration system (TCS-A) that has one NIC on the Internet and one on the corporate LAN. All systems will call this end point by using the same public URL, but systems on the internal network will send their media to the local NIC and systems on the Internet will send media to the external NIC.



In order to make this work, a static route is added to the corporate gateway router (GWR) to redirect internal traffic destined for the external URL .

```
route add -net 216.13.45.142 netmask 255.255.255.255 gw 192.168.217.84
```

Systems on the local LAN (TCS-B) trying to reach "216.13.45.142" will send their traffic to the gateway (GWR) "192.168.216.1" and it will forward the traffic to "192.168.217.84". If the static route is added correctly we will see the following:

```
videowall@dan:~$ tracepath 216.13.45.142
 1: dan.magor.local (192.168.217.44)           0.076ms pmtu 1500
 1: 192.168.216.1 (192.168.216.1)           1.009ms
 1: 192.168.216.1 (192.168.216.1)           0.968ms
 2: 216.13.45.142 (216.13.45.142)         3.151ms reached
Resume: pmtu 1500 hops 2 back 64

videowall@dan:~$ traceroute 216.13.45.142
traceroute to 216.13.45.142 (216.13.45.142), 30 hops max, 40 byte packets
 1 192.168.216.1 (192.168.216.1)  0.284 ms  0.327 ms  0.389 ms
 2 216.13.45.142 (216.13.45.142) 0.582 ms  0.626 ms  0.666 ms
```

Notice how the packets from "192.168.217.44" go to the gateway "192.168.216.1" and

then bounce back to reach their destination right away. If the packets were not being redirected we would get something like the printout below:

```
root@fred-laptop:~# tracepath 216.13.45.140
 0:  fredlaptop.magor.local (192.168.217.53)           0.120ms pmtu 1500
 1:  192.168.216.1 (192.168.216.1)                   1.296ms
 1:  192.168.216.1 (192.168.216.1)                   1.279ms
 2:  72-1-217.105.tel-ott.com (72.1.217.105)         2.056ms
 3:  tol-gw.hydroone.com (142.46.130.9)              2.210ms asymm 4
 4:  Router1.HydroOneTelecom.com (142.46.128.13)     7.658ms asymm 5
 5:  gw-allstream.torontointernetexchange.net (198.32.245.4) 9.186ms asymm 7
 6:  ge7-1.hcap4-ott.bb.allstream.net (199.212.172.34) 14.042ms asymm 9
 7:  216.13.62.226 (216.13.62.226)                  15.327ms asymm 9
```

This time the packets go to the gateway, but then go to external routers before they find their way back to the external address.

Both of these tools can also be used to find routing problems. Tracing a remote IP address will list the routers in the path up to where the route can't be resolved. You can then focus your attention on that device.

## Appendix D: Ping

The most used debugging command is most likely “ping”. The “ping” command sends ICMP echo requests to a remote IP address and then prints the statistics associated with the returned echo packet.

```
root@fred-laptop:~# ping 192.168.217.44
PING 192.168.217.44 (192.168.217.44) 56(84) bytes of data.
64 bytes from 192.168.217.44: icmp_seq=1 ttl=64 time=0.272 ms
64 bytes from 192.168.217.44: icmp_seq=2 ttl=64 time=0.215 ms
64 bytes from 192.168.217.44: icmp_seq=3 ttl=64 time=0.255 ms
64 bytes from 192.168.217.44: icmp_seq=4 ttl=64 time=0.255 ms
```

For a complete list of options available for the ping command refer to the man page. The most commonly used options would be :

```
ping -c 5 192.168.217.44 //Ping the remote address only 5 times
ping -s 2000 192.168.217.44 //Ping the remote address with packets of 2000 bytes
ping -R 192.168.217.44 //Ping the remote address and display the route
ping -I eth4 192.168.217.44 //Ping the remote address using the specified NIC
```

Note that ping packets with MTU greater than 1500 will most likely be fragmented. The “-M do” option will prevent fragmentation, but special NIC hardware is required to support the larger frames.



