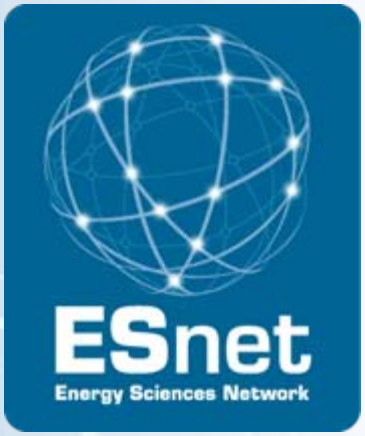


Iperf Tutorial

Jon Dugan <jdugan@es.net>

Summer JointTechs 2010, Columbus, OH

Outline

What are we measuring?

TCP Measurements

UDP Measurements

Useful tricks

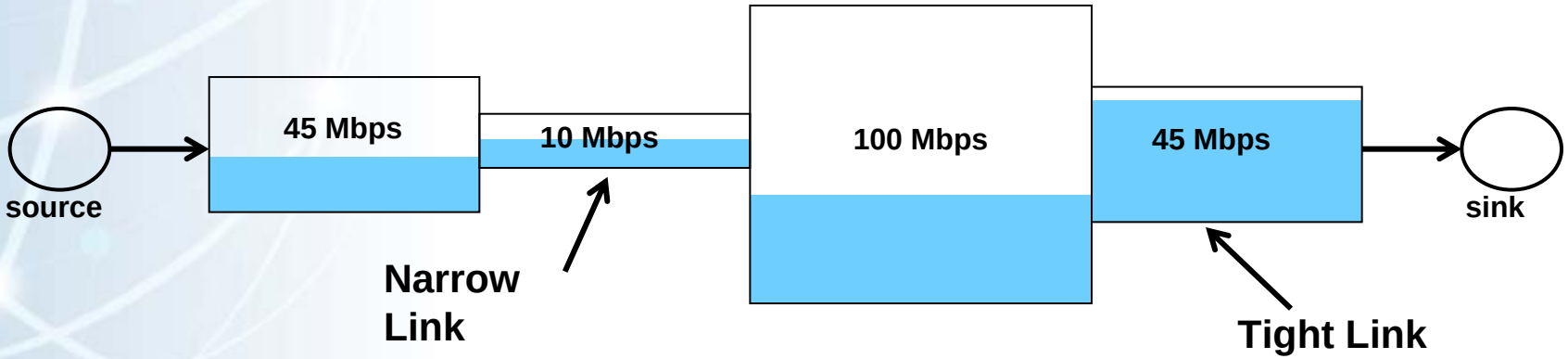
Iperf Development

What are we measuring?

Throughput? Bandwidth? What?

The term “throughput” is vague

- Capacity: link speed
  - Narrow Link: link with the lowest capacity along a path
  - Capacity of the end-to-end path = capacity of the narrow link
- Utilized bandwidth: current traffic load
- Available bandwidth: capacity – utilized bandwidth
  - Tight Link: link with the least available bandwidth in a path
- Achievable bandwidth: includes protocol and host issues



(Shaded portion shows background traffic)

Iperf data flow

Client is the sender
(data source)

Server is the receiver
(data sink)

Iperf discards
the data

TCP Measurements

TCP Measurements

Measures TCP Achievable Bandwidth

- Measurement includes the end system
- Sometimes called “memory-to-memory” tests
- Set expectations for well coded application

Limits of what we can measure

- TCP hides details
- In hiding the details it can obscure what is causing errors

Many things can limit TCP throughput

- Loss
- Congestion
- Buffer Starvation
- Out of order delivery

Example Iperf TCP Invocation

Server (receiver):

```
$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.0.1.5 port 5001 connected with 10.0.1.10 port 60830
[  4]  0.0-10.0 sec  1.09 GBytes    933 Mbits/sec
[  4] local 10.0.1.5 port 5001 connected with 10.0.1.10 port 60831
[  4]  0.0-10.0 sec  1.08 GBytes    931 Mbits/sec
```

Client (sender):

```
$ iperf -c 10.0.1.5
------------------------------------------------------------
Client connecting to 10.0.1.5, TCP port 5001
TCP window size:  129 KByte (default)
------------------------------------------------------------
[  3] local 10.0.1.10 port 60830 connected with 10.0.1.5 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.2 sec  1.09 GBytes   913 Mbits/sec
```

TCP performance: window size

Use TCP auto tuning if possible

- Linux 2.6, Mac OS X 10.5, FreeBSD 7.x, and Windows Vista

The –w option for Iperf can be used to request a particular buffer size.

- Use this if your OS doesn't have TCP auto tuning
- This sets both send and receive buffer size.
- The OS may need to be tweaked to allow buffers of sufficient size.
- See http://fasterdata.es.net/tuning.html for more details

Parallel transfers may help as well, the –P option can be used for this

To get full TCP performance the TCP window needs to be large enough to accommodate the Bandwidth Delay Product

TCP performance: Bandwidth Delay Product

The amount of “in flight” data allowed for a TCP connection

BDP = bandwidth * round trip time

Example: 1Gb/s cross country, ~100ms

1,000,000,000 b/s * .1 s = 100,000,000 bits

100,000,000 / 8 = 12,500,000 bytes

12,500,000 bytes / (1024*1024) ~ 12MB

TCP performance: read/write buffer size

TCP breaks the stream into pieces transparently

Longer writes often improve performance

- Let TCP “do it’s thing”
- Fewer system calls

How?

- -l <size> (lower case ell)
- Example –l 128K

UDP doesn’t break up writes, don’t exceed Path MTU

TCP performance: parallel streams

Parallel streams can help in some situations

TCP attempts to be “fair” and conservative

- Sensitive to loss, but more streams hedge bet
- Circumventing fairness mechanism
  - 1 Iperf stream vs. n background: Iperf gets 1/(n+1)
  - x Iperf streams vs. n background: Iperf gets x/(n+x)
  - Example: 2 background, 1 Iperf stream: 1/3 = 33%
  - Example: 2 background, 8 Iperf streams: 8/10 = 80%

How?

- The –P option sets the number of streams to use
- There is a point of diminishing returns

TCP performance: congestion control algorithm selection

Classic TCP (aka TCP Reno) is very conservative

Linux supports several different algorithms

- http://en.wikipedia.org/wiki/TCP_congestion_avoidance_algorithm
- CUBIC seems to work well for RE&E traffic flows

How?

- -Z allows the selection of a congestion control algorithm

UDP Measurements

UDP Measurements

UDP provides greater transparency

We can directly measure some things TCP hides

- Loss
- Jitter
- Out of order delivery

Use -b to specify target bandwidth

- Default is 1M
- Two sets of multipliers
  - K. m, g multipliers are 1000, 1000^2,1000^3
  - K, M, G multipliers are 1024, 1024^2,1024^3
- Eg, -b 1m is 1,000,000 bits per second

Example Iperf UDP Invocation

Server (receiver):

```
$ iperf -u -s
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:   107 KByte (default)
------------------------------------------------------------
[  3] local 10.0.1.5 port 5001 connected with 10.0.1.10 port 65299
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec  0.008 ms    0/  893 (0%)
```

Client (sender):

```
$ iperf -u -c 10.0.1.5 -b 1M
------------------------------------------------------------
Client connecting to 10.0.1.5, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 9.00 KByte (default)
------------------------------------------------------------
[  3] local 10.0.1.10 port 65300 connected with 10.0.1.5 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec  0.003 ms    0/  893 (0%)

[  3] Sent 893 datagrams
```

Useful tricks

Using Iperf to generate high rate streams

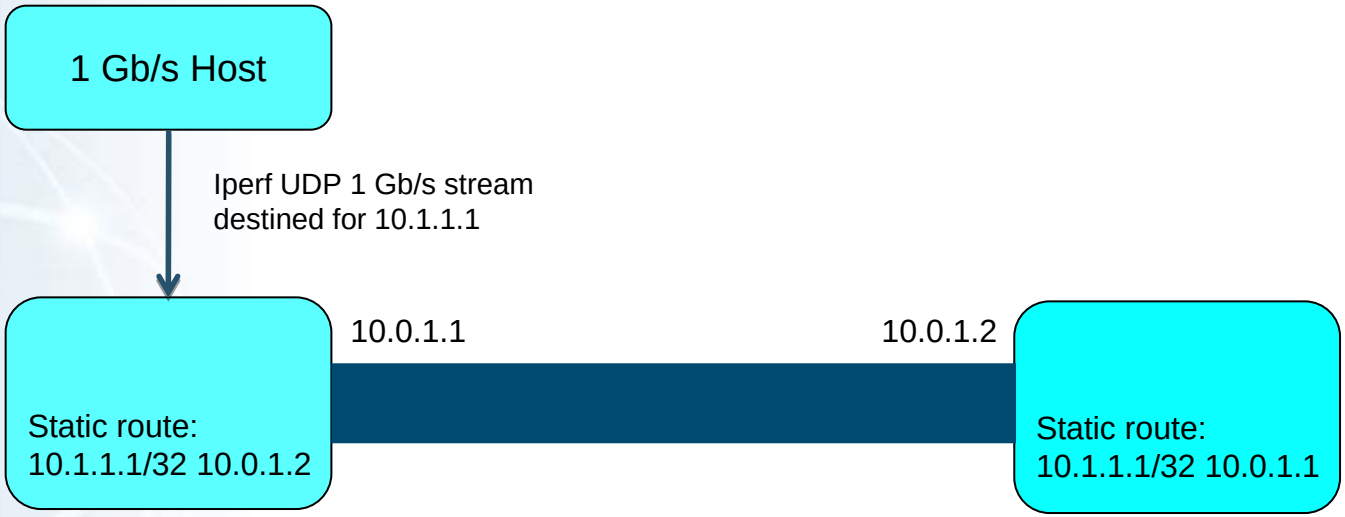
UDP doesn’t require a receiver

If you have good counters on your switches & routers those can be used to measure

Turns out UDP reception can be very resource intensive resulting in drops at the NIC at high rates (8-9 Gb/s)

Routing loops for fun and profit

Generate 10 Gb/s of traffic using a 1 Gb/s source host



Use the –T option to Iperf to control the number of times the traffic loops
Can also use firewall filters to discard a certain TTL range.
Other filters may be prudent as well.
Use firewall filters to count traffic on your router.

(Do not try this at home, the author is a highly insane network engineer.)

Iperf Development

Iperf Development
(Iperf is dead, long live Iperf)

Iperf 2

- Iperf 2 is widely used
- Current version is 2.0.5 (released July 8, 2010)
- No further development, maintenance only
  - critical patches
  - sporadic releases, only when necessary

Iperf 3

- Currently in development
- Current version is 3.0b1 (released July 8, 2010)
- Weekly beta releases (starting this past Thursday)
- Eventually replace Iperf 2

Iperf 3: current status

Working

- TCP
- Control channel
  - Stream setup
  - Test parameter negotiation
  - Results Exchange
- Clean code!

Coming Soon

- UDP tests
- API with sane error reporting, library
- Timeline at: http://code.google.com/p/iperf/wiki/Iperf3Roadmap

More Information

Iperf 2:
http://sourceforge.net/projects/iperf/
Iperf 3:
http://code.google.com/p/iperf/

User Discussion:
iperf-users@lists.sourceforge.net
Developer Discussion:
iperf-dev@googlegroups.com

Network performance:
http://fasterdata.es.net/

Jon Dugan <jdugan@es.net>