



Network Performance Measurement

Network Performance Metrics

- Planning performance management
- Metrics
 - Network
 - *Systems*
 - *Services*
- Definitions

Planning

- What's the intention?
 - *Baselining, Troubleshooting, Planning growth*
 - Defend yourself from accusations -"it's the network!"
- Who is the information for?
 - Administration, NOC, customers
 - How to structure and present the information
- Reach: Can I measure everything?
 - Impact on devices (measurements and measuring)
 - Balance between amount of information and time to get it

Metrics

Network performance metrics

- Channel capacity, nominal & effective
- Channel utilization
- Delay and *jitter*
- Packet loss and errors

Metrics

What we are not discussing:

System performance metrics

- Availability
- Memory, CPU Utilization, *load*, *I/O wait*, etc.

Service performance metrics

Available here and in class outline:

<http://nsrc.org/workshops/2009/summer/ref/performance-metrics.pdf>

Common network performance measurements

- Relative to traffic:
 - Bits per second
 - Packets per second
 - *Unicast vs. non-unicast* packets
 - Errors
 - Dropped packets
 - Flows per second
 - Round trip time (RTT)
 - Jitter (variation between packet RTT)

Nominal channel capacity

- The maximum number of bits that can be transmitted for a unit of time (eg: bits per second)
- Depends on:
 - Bandwidth of the physical medium
 - Cable
 - Electromagnetic waves
 - Processing capacity for each transmission element
 - Efficiency of algorithms in use to access medium
 - Channel encoding and compression

Effective channel capacity

- Always a fraction of the nominal channel capacity
- Dependent on:
 - Additional overhead of protocols in each layer
 - Device limitations on both ends
 - Flow control algorithm efficiency, etc.
 - For example: TCP

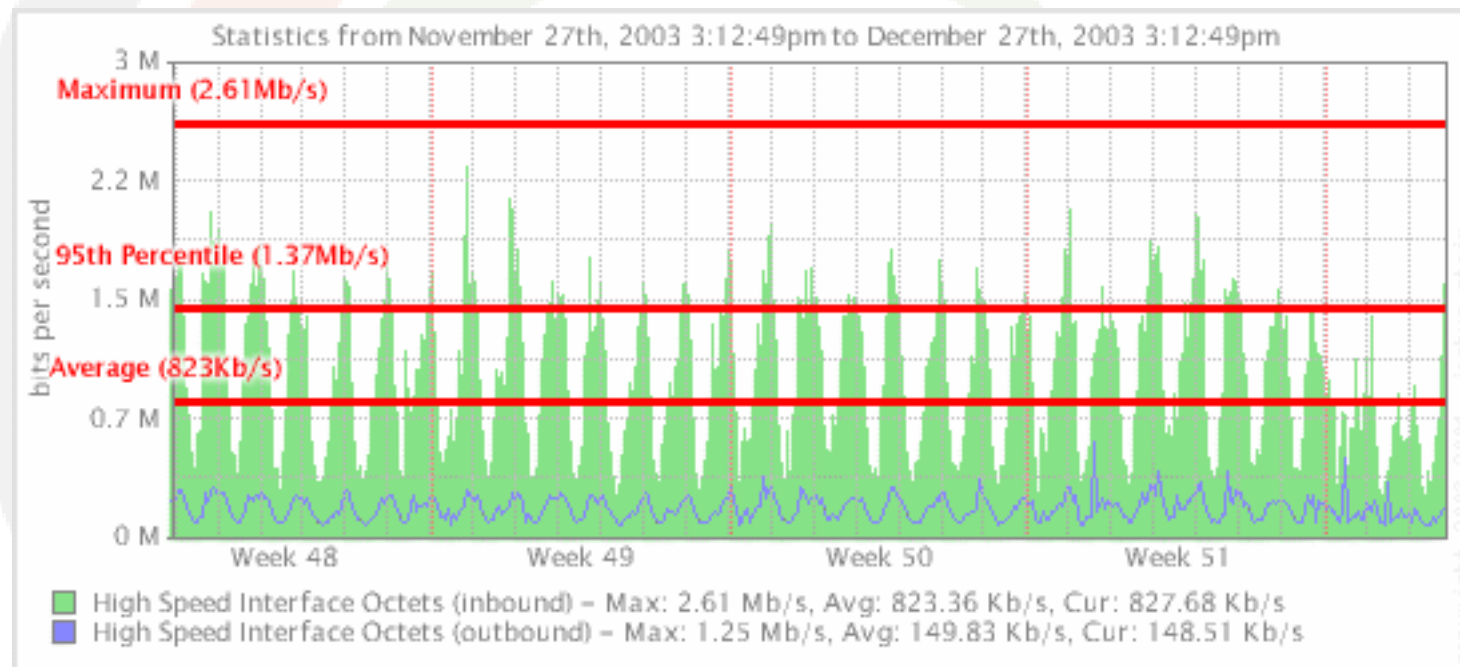
Channel utilization

- What fraction of the nominal channel capacity is actually in use
- Important!
 - Future planning
 - What utilization growth rate am I seeing?
 - For when should I plan on buying additional capacity?
 - Where should I invest for my updates?
 - Problem resolution
 - Where are my bottlenecks, etc.

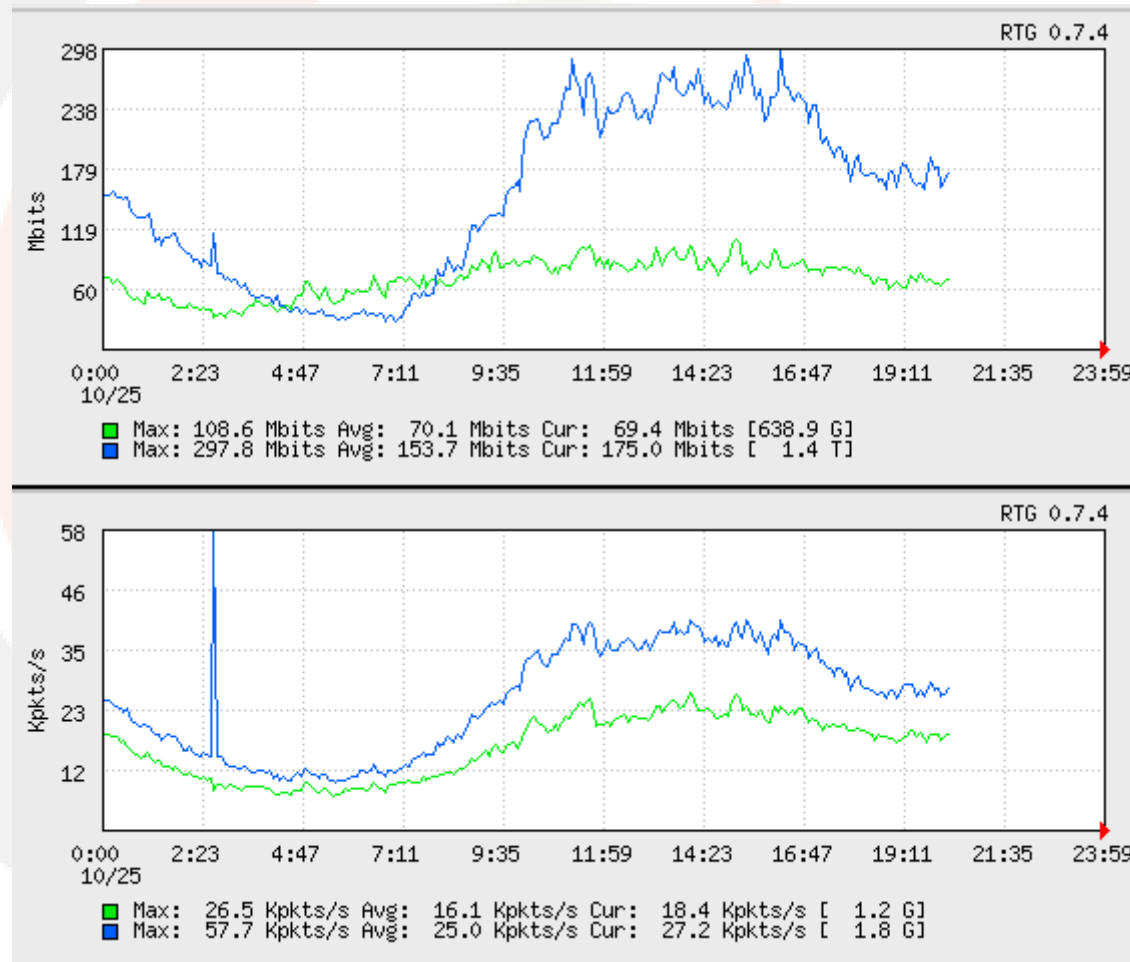
95th Percentile

- The smallest value that is larger than 95% of the values in a given sample
- This means that 95% of the time the channel utilization is equal to or *less* than this value
 - Or rather, the peaks are discarded from consideration
- Why is this important in networks?
 - Gives you an idea of the standard, sustained channel utilization.
 - ISPs use this measure to bill customers with “larger” connections.

95th Percentile



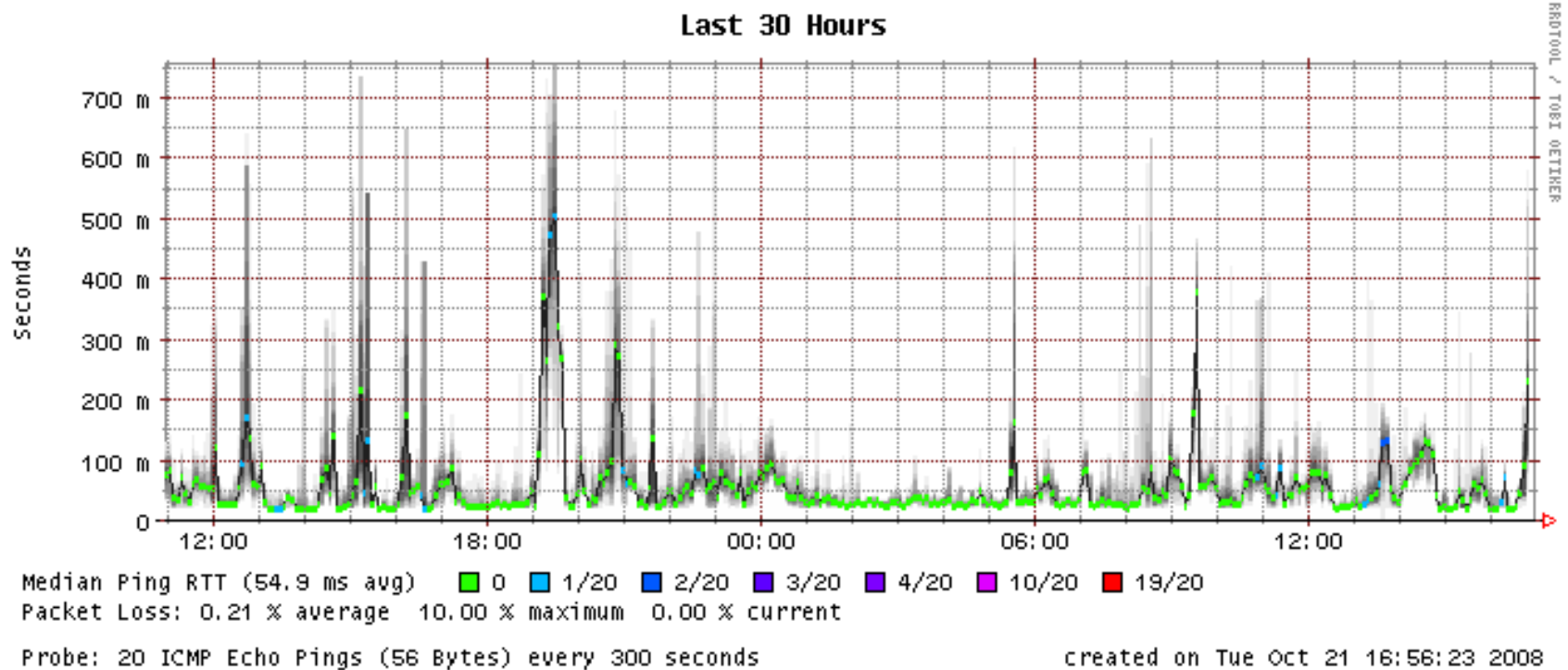
Bits per second vs Packets p.s.



End-to-end delay

- The time required to transmit a packet along its entire path
 - *Created by an application, handed over to the OS, passed to a network card (NIC), encoded, transmitted over a physical medium (copper, fibre, air), received by an intermediate device (switch, router), analyzed, retransmitted over another medium, etc.*
 - The most common measurement uses *ping* for total round-trip-time (RTT).

Historical measurement of delay



Types of Delay

- Causes of end-to-end delay
 - Processor delays
 - Buffer delays
 - Transmission delays
 - Propagation delays

Processing delay

- Required time to analyze a packet header and decide where to send the packet (eg. a routing decision)
 - Inside a router this depends on the number of entries in the routing table, the implementation of data structures, hardware in use, etc.
- This can include error verification / checksumming (i.e. IPv4, IPv6 header checksum)

Queuing Delay

- The time a packet is enqueued until it is transmitted
- The number of packets waiting in the queue will depend on traffic intensity and of the type of traffic
- Router queue algorithms try to adapt delays to specific preferences, or impose equal delay on all traffic.

Transmission Delay

- The time required to push all the bits in a packet on the transmission medium in use
- For N =Number of bits, S =Size of packet, d =delay

$$d = S/N$$

- For example, to transmit 1024 bits using Fast Ethernet (100Mbps)

$$d = 1024/1 \times 10^8 = 10.24 \text{ micro seconds}$$

Propagation Delay

- Once a bit is 'pushed' on to the transmission medium, the time required for the bit to propagate to the end of its physical trajectory
- The velocity of propagation of the circuit depends mainly on the actual distance of the physical circuit
 - In the majority of cases this is close to the speed of light.
- For d = distance, s = propagation velocity

$$PD = d/s$$

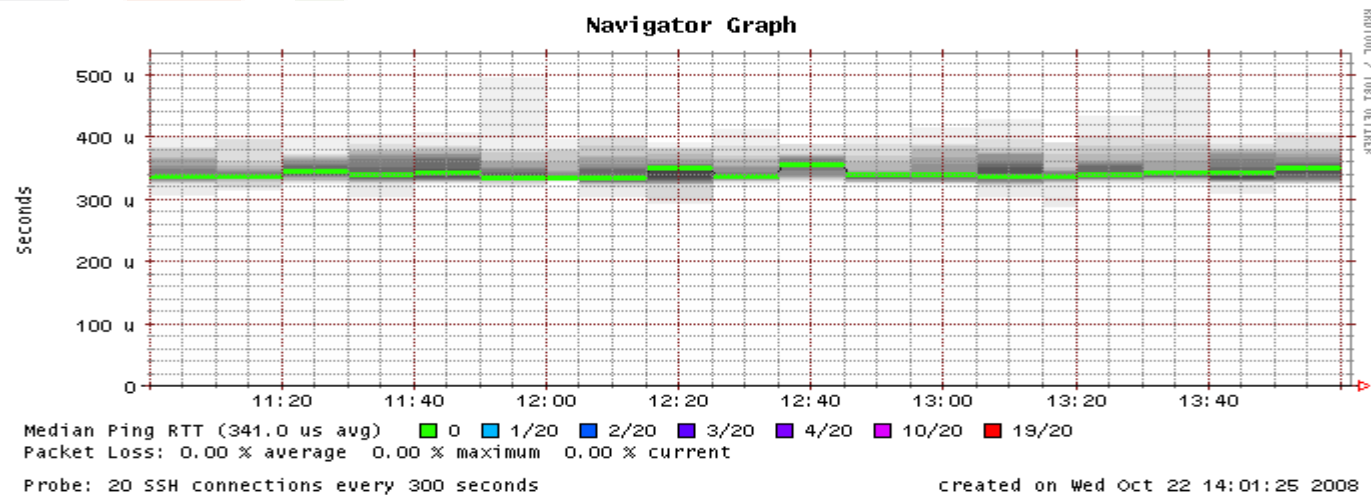
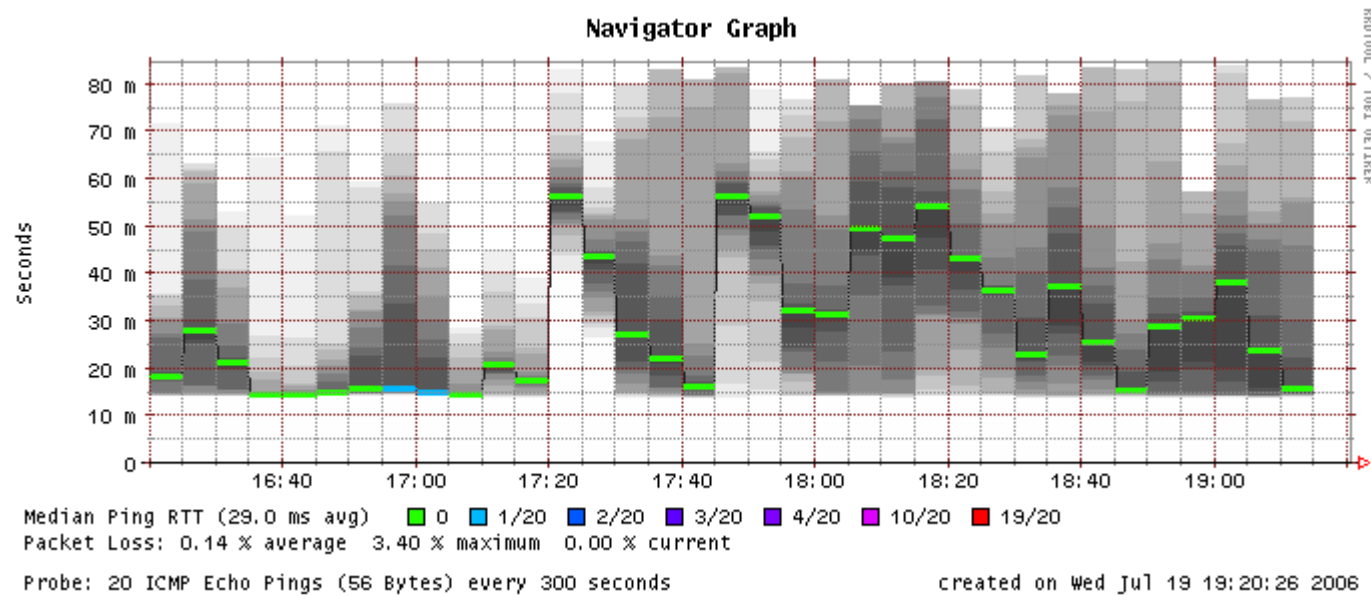
Transmission vs. Propagation

- Can be confusing at first
- Consider this example:
 - Two 100 Mbps circuits
 - 1 km of optic fiber
 - Via satellite with a distance of 30 km between the base and the satellite
 - For two packets of the same size which will have the larger transmission delay? Propagation delay?

Packet Loss

- Occur due to the fact that buffers are not infinite in size
 - When a packet arrives to a buffer that is full the packet is discarded.
 - Packet loss, if it must be corrected, is resolved at higher levels in the network stack (transport or application layers)
 - Loss correction using retransmission of packets can cause yet more congestion if some type of (flow) control is not used (to inform the source that it's pointless to keep sending more packets at the present time)

Jitter



Flow Control and Congestion

- Limits the transmission amount (rate) because the receiver cannot process packets at the same rate that packets are arriving.
- Limit the amount sent (transmission rate) because of loss or delays in the circuit.

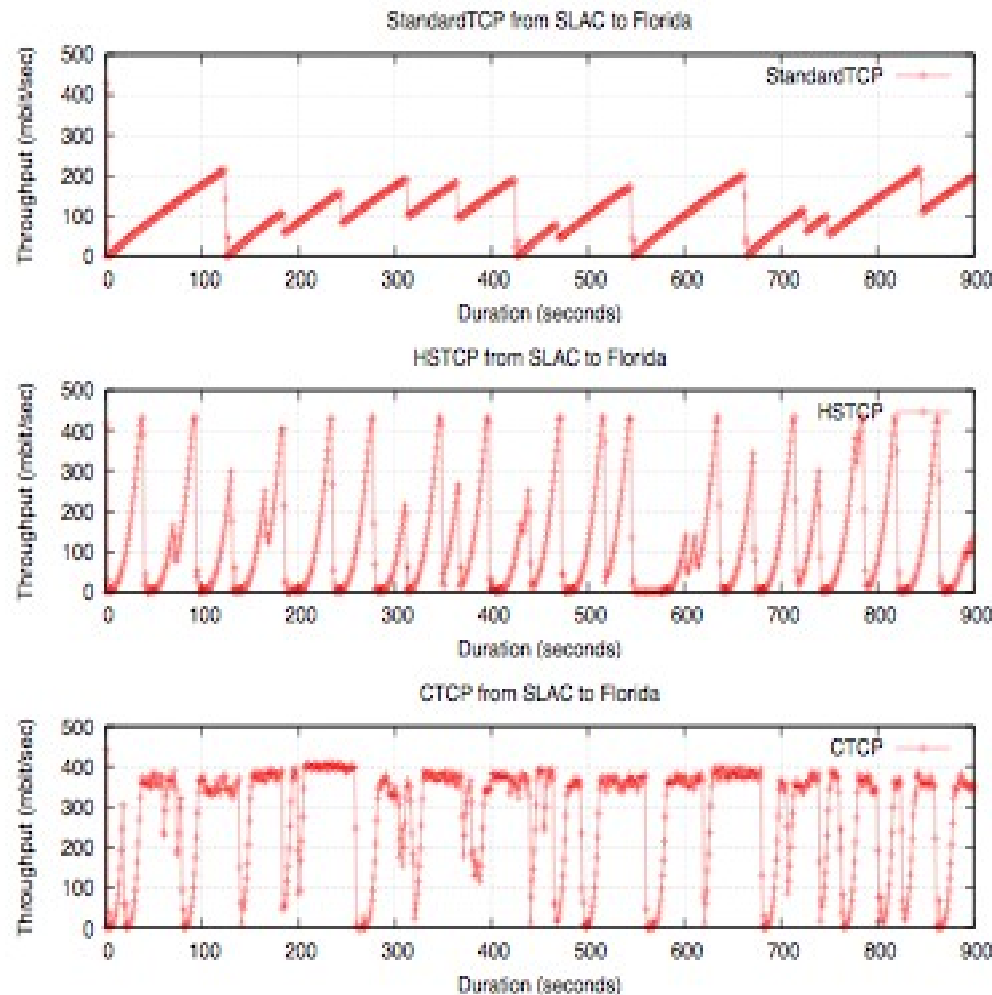
Controls in TCP

- IP (Internet Protocol) implements service that not connection oriented.
 - There is no mechanism in IP to deal with packet loss.
- TCP (Transmission *Control* Protocol) implements flow and congestion control.
 - Only on the ends as the intermediate nodes at the network level do not talk TCP

Congestion vs. Flow in TCP

- **Flow**: controlled by window size (RcvWindow), which is sent by the receiving end.
- **Congestion**: controlled by the value of the congestion window (Congwin)
 - Maintained independently by the sender
 - This varies based on the detection of packets lost
 - Timeout or receiving three ACKs repeated
 - **Behaviors:**
 - Additive Increments / Multiplicative Decrements (AIMD)
 - Slow Start
 - React to *timeout* events

Different TCP Congestion Control Algorithms





Local analysis

- **As we know... Before we blame the network, let's verify whether the problem is ours.**
- **What can go wrong locally?**
 - Hardware problems
 - Excessive load (CPU, memory, I/O)
- **What's considered 'normal'?**
 - Use analysis tools frequently
 - Become familiar with the normal state and values for your machine.
 - **It is essential to maintain history**
 - SNMP agents and databases

Performance analysis in Unix

- Three main categories:
 - Processes
 - Processes that are executing (running)
 - Processes that are waiting (sleeping)
 - waiting their turn
 - blocked
 - Memory
 - Real
 - Virtual
 - I/O (Input/Output)
 - Storage
 - Network

Key indicators

- **Insufficient CPU**
 - Number of processes waiting to execute is always high
 - High CPU utilization (load avg.)
- **Insufficient memory**
 - Very little free memory
 - Lots of swap activity (swap in, swap out)
- **Slow I/O**
 - Lots of blocked processes
 - High number of block transfers

Local analysis

- Luckily, in Unix there are dozens of useful tools that give us lots of useful information about our machine
- Some of the more well-known include:
 - vmstat
 - top
 - lsof
 - netstat
 - tcpdump
 - Wireshark (Ethereal)
 - iptraf
 - iperf

vmstat

- Show periodic summary information about processes, memory, paging, I/O, CPU state, etc

vmstat <-options> <delay> <count>

```
# vmstat 2
procs -----memory----- --swap-- -----io----- --system-- -----cpu-----
r  b   swpd   free   buff   cache    si   so    bi    bo    in     cs us sy id wa
2  0  209648  25552  571332  2804876    0    0     3     4     3      3 15 11 73  0
2  0  209648  24680  571332  2804900    0    0     0    444   273  79356 16 16 68  0
1  0  209648  25216  571336  2804904    0    0     6   1234   439  46735 16 10 74  0
1  0  209648  25212  571336  2804904    0    0     0     22   159 100282 17 21 62  0
2  0  209648  25196  571348  2804912    0    0     0    500   270  82455 14 18 68  0
1  0  209648  25192  571348  2804912    0    0     0    272   243  77480 16 15 69  0
2  0  209648  25880  571360  2804916    0    0     0    444   255  83619 16 14 69  0
2  0  209648  25872  571360  2804920    0    0     0    178   220  90521 16 18 66  0
```

top

- Basic performance tool for Unix/Linux environments
- Periodically show a list of system performance statistics:
 - CPU use
 - RAM and SWAP memory usage
 - Load average (cpu utilization)
 - Information by process

Load Average

- Average number of active processes in the last 1, 5 and 15 minutes
 - A simple yet useful measurement
 - Depending on the machine the acceptable range considered to be normal can vary:
 - Multi-processor machines can handle more active processes per unit of time (than single processor machines)

top

- **Information by process (most relevant columns shown):**
 - PID: Process ID
 - USER: user running (owner) of the process
 - %CPU: Percentage of CPU utilization by the process since the last sample
 - %MEM: Percentage of physical memory (RAM) used by the process
 - TIME: Total CPU time used by the process since it was started

top

- **Some useful *interactive* commands**
 - **f** : Add or remove columns
 - **F** : Specify which column to order by
 - **<** , **>** : Move the column on which we order
 - **u** : Specify a specific user
 - **k** : Specify a process to kill (stop)
 - **d** , **s** : Change the display update interval

netstat

- **Show us information about:**
 - Network connections
 - Routing tables
 - Interface (NIC) statistics
 - Multicast group members

netstat

Some useful options

- n**: Show addresses, ports and userids in numeric form
- r**: Routing table
- s**: Statistics by protocol
- i**: Status of interfaces
- l**: Listening sockets
- tcp, --udp**: Specify the protocol
- A**: Address family [inet | inet6 | unix | etc.]
- p**: Show the name of each process for each port
- c**: Show output/results continuously

netstat

Examples:

```
# netstat -n --tcp -c
```

Active Internet connections (w/o servers)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	272	::ffff:192.188.51.40:22	::ffff:128.223.60.27:60968	ESTABLISHED
tcp	0	0	::ffff:192.188.51.40:22	::ffff:128.223.60.27:53219	ESTABLISHED

```
# netstat -lnp --tcp
```

Active Internet connections (only servers)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	0.0.0.0:199	0.0.0.0:*	LISTEN	11645/snmpd
tcp	0	0	0.0.0.0:3306	0.0.0.0:*	LISTEN	1997/mysqld

```
# netstat -ic
```

Kernel Interface table

Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
eth0	1500	0	2155901	0	0	0	339116	0	0	0	BMRU
lo	16436	0	18200	0	0	0	18200	0	0	0	LRU
eth0	1500	0	2155905	0	0	0	339117	0	0	0	BMRU
lo	16436	0	18200	0	0	0	18200	0	0	0	LRU
eth0	1500	0	2155907	0	0	0	339120	0	0	0	BMRU
lo	16436	0	18200	0	0	0	18200	0	0	0	LRU
eth0	1500	0	2155910	0	0	0	339122	0	0	0	BMRU
lo	16436	0	18200	0	0	0	18200	0	0	0	LRU
eth0	1500	0	2155913	0	0	0	339124	0	0	0	BMRU

netstat

Examples:

```
# netstat -tcp -listening --program
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 *:5001                  *:.*                     LISTEN      13598/iperf
tcp        0      0 localhost:mysql         *:.*                     LISTEN      5586/mysqld
tcp        0      0 *:www                   *:.*                     LISTEN      7246/apache2
tcp        0      0 t60-2.local:domain    *:.*                     LISTEN      5378/named
tcp        0      0 t60-2.local:domain    *:.*                     LISTEN      5378/named
tcp        0      0 t60-2.local:domain    *:.*                     LISTEN      5378/named
tcp        0      0 localhost:domain       *:.*                     LISTEN      5378/named
tcp        0      0 localhost:ipp           *:.*                     LISTEN      5522/cupsd
tcp        0      0 localhost:smtp          *:.*                     LISTEN      6772/exim4
tcp        0      0 localhost:953           *:.*                     LISTEN      5378/named
tcp        0      0 *:https                 *:.*                     LISTEN      7246/apache2
tcp6       0      0 [::]:ftp                [::]:*                   LISTEN      7185/proftpd
tcp6       0      0 [::]:domain             [::]:*                   LISTEN      5378/named
tcp6       0      0 [::]:ssh                [::]:*                   LISTEN      5427/sshd
tcp6       0      0 [::]:3000               [::]:*                   LISTEN      17644/ntop
tcp6       0      0 ip6-localhost:953      [::]:*                   LISTEN      5378/named
tcp6       0      0 [::]:3005               [::]:*                   LISTEN      17644/ntop
```

netstat

Examples:

```
$ sudo netstat -atup
```

Active Internet connections (servers and established) *(if run as root PID/Program name is included)*

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	*:35586	*:*	LISTEN	2540/ekpd
tcp	0	0	localhost:mysql	*:*	LISTEN	2776/mysqld
tcp	0	0	*:www	*:*	LISTEN	14743/apache2
tcp	0	0	d229-231.uoregon:domain	*:*	LISTEN	2616/named
tcp	0	0	*:ftp	*:*	LISTEN	3408/vsftpd
tcp	0	0	localhost:domain	*:*	LISTEN	2616/named
tcp	0	0	*:ssh	*:*	LISTEN	2675/sshd
tcp	0	0	localhost:ipp	*:*	LISTEN	3853/cupsd
tcp	0	0	localhost:smtp	*:*	LISTEN	3225/exim4
tcp	0	0	localhost:953	*:*	LISTEN	2616/named
tcp	0	0	*:https	*:*	LISTEN	14743/apache2
tcp6	0	0	[::]:domain	:::*	LISTEN	2616/named
tcp6	0	0	[::]:ssh	:::*	LISTEN	2675/sshd
tcp6	0	0	ip6-localhost:953	:::*	LISTEN	2616/named
udp	0	0	*:50842	*:*		3828/avahi-daemon:
udp	0	0	localhost:snmp	*:*		3368/snmpd
udp	0	0	d229-231.uoregon:domain	*:*		2616/named
udp	0	0	localhost:domain	*:*		2616/named
udp	0	0	*:bootpc	*:*		13237/dhclient
udp	0	0	*:mdns	*:*		3828/avahi-daemon:
udp	0	0	d229-231.uoregon.ed:ntp	*:*		3555/ntpd
udp	0	0	localhost:ntp	*:*		3555/ntpd
udp	0	0	*:ntp	*:*		3555/ntpd
udp6	0	0	[::]:domain	:::*		2616/named
udp6	0	0	fe80::213:2ff:fe1f::ntp	:::*		3555/ntpd
udp6	0	0	ip6-localhost:ntp	:::*		3555/ntpd
udp6	0	0	[::]:ntp	:::*		3555/ntpd

Isof (List Open Files)

- Isof is particularly useful because in Unix everything is a file: unix sockets, ip sockets, directories, etc.
- Allows you to associate open files by:
 - p**: PID (Process ID)
 - i** : A network address (protocol:port)
 - u**: A user

Isof

- **Example:**

- First, using *netstat -ln -tcp* determine that port 6010 is open and waiting for a connection (LISTEN)

```
# netstat -ln --tcp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 127.0.0.1:6010          0.0.0.0:*               LISTEN
tcp      0      0 127.0.0.1:6011          0.0.0.0:*               LISTEN
```

Isof

Determine what process has the port (6010) open and what other resources are being used:

```
# lsof -i tcp:6010
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
sshd	10301	root	6u	IPv4	53603		TCP	localhost.localdomain:x11-ssh-offset (LISTEN)
sshd	10301	root	7u	IPv6	53604		TCP	:::1:x11-ssh-offset (LISTEN)

```
# lsof -p 10301
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
sshd	10301	root	cwd	DIR	8,2	4096	2	/
sshd	10301	root	rtd	DIR	8,2	4096	2	/
sshd	10301	root	txt	REG	8,2	379720	1422643	/usr/sbin/sshd
sshd	10301	root	mem	REG	8,2	32724	1437533	/usr/lib/libwrap.so.0.7.6
sshd	10301	root	mem	REG	8,2	15088	3080329	/lib/libutil-2.4.so
sshd	10301	root	mem	REG	8,2	75632	1414093	/usr/lib/libz.so.1.2.3
sshd	10301	root	mem	REG	8,2	96040	3080209	/lib/libnsl-2.4.so
sshd	10301	root	mem	REG	8,2	100208	1414578	/usr/lib/libgssapi_krb5.so.2.2
sshd	10301	root	mem	REG	8,2	11684	1414405	/usr/lib/libkrb5support.so.0.0
sshd	10301	root	mem	REG	8,2	10368	3080358	/lib/libsetrans.so.0
sshd	10301	root	mem	REG	8,2	7972	3080231	/lib/libcom_err.so.2.1
sshd	10301	root	mem	REG	8,2	30140	1420233	/usr/lib/libcrack.so.2.8.0
sshd	10301	root	mem	REG	8,2	11168	3080399	/lib/security/pam_succeed_if.so

...

Isof

What network services am I running?

```
# lsof -i
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
firefox	4429	hervey	50u	IPv4	1875852		TCP	192.168.179.139:56890->128.223.60.21:www (ESTABLISHED)
named	5378	bind	20u	IPv6	13264		TCP	*:domain (LISTEN)
named	5378	bind	21u	IPv4	13267		TCP	localhost:domain (LISTEN)
sshd	5427	root	3u	IPv6	13302		TCP	*:ssh (LISTEN)
cupsd	5522	root	3u	IPv4	1983466		TCP	localhost:ipp (LISTEN)
mysqld	5586	mysql	10u	IPv4	13548		TCP	localhost:mysql (LISTEN)
snmpd	6477	snmp	8u	IPv4	14633		UDP	localhost:snmp
exim4	6772	Debian-exim	3u	IPv4	14675		TCP	localhost:smtp (LISTEN)
ntpd	6859	ntp	16u	IPv4	14743		UDP	*:ntp
ntpd	6859	ntp	17u	IPv6	14744		UDP	*:ntp
ntpd	6859	ntp	18u	IPv6	14746		UDP	[fe80::250:56ff:fec0:8]:ntp
ntpd	6859	ntp	19u	IPv6	14747		UDP	ip6-localhost:ntp
proftpd	7185	proftpd	1u	IPv6	15718		TCP	*:ftp (LISTEN)
apache2	7246	www-data	3u	IPv4	15915		TCP	*:www (LISTEN)
apache2	7246	www-data	4u	IPv4	15917		TCP	*:https (LISTEN)
...								
iperf	13598	root	3u	IPv4	1996053		TCP	*:5001 (LISTEN)
apache2	27088	www-data	3u	IPv4	15915		TCP	*:www (LISTEN)
apache2	27088	www-data	4u	IPv4	15917		TCP	*:https (LISTEN)

tcpdump

- Show received packet headers by a given interface. Optionally filter using boolean expressions.
- Allows you to write information to a file for later analysis.
- Requires administrator (root) privileges to use since you must configure network interfaces (NICs) to be in “promiscuous” mode.
 - Note: promiscuous mode is not very useful when you are connected by a switch.

tcpdump

Some useful options:

- **-i** : Specify the interface (ex: -i eth0)
- **-l** : Make stdout line buffered (view as you capture)
- **-v**, **-vv**, **-vvv**: Display more information
- **-n** : Don't convert addresses to names (avoid DNS)
- **-nn** : Don't translate port numbers
- **-w** : Write raw packets to a file
- **-r** : Read packets from a file created by '-w'

tcpdump

Boolean expressions

- Using the 'AND', 'OR', 'NOT' operators
- Expressions consist of one, or more, primitives, which consist of a qualifier and an ID (name or number)
 - Expression ::= [NOT] <primitive> [AND | OR | NOT <primitive> ...]
 - <primitive> ::= <qualifier> <name|number>
 - <qualifier> ::= <type> | <address> | <protocol>
 - <type> ::= host | net | port | port range
 - <address> ::= src | dst
 - <protocol> ::= ether | fddi | tr | wlan | ip | ip6 | arp | rarp | decnet | tcp | udp

tcpdump

Examples:

- Show all HTTP traffic that originates from 192.168.1.1

```
# tcpdump -lnXvvv port 80 and src host 192.168.1.1
```

- Show all traffic originating from 192.168.1.1 *except* SSH

```
# tcpdump -lnXvvv src host 192.168.1.1 and not port 22
```

wireshark

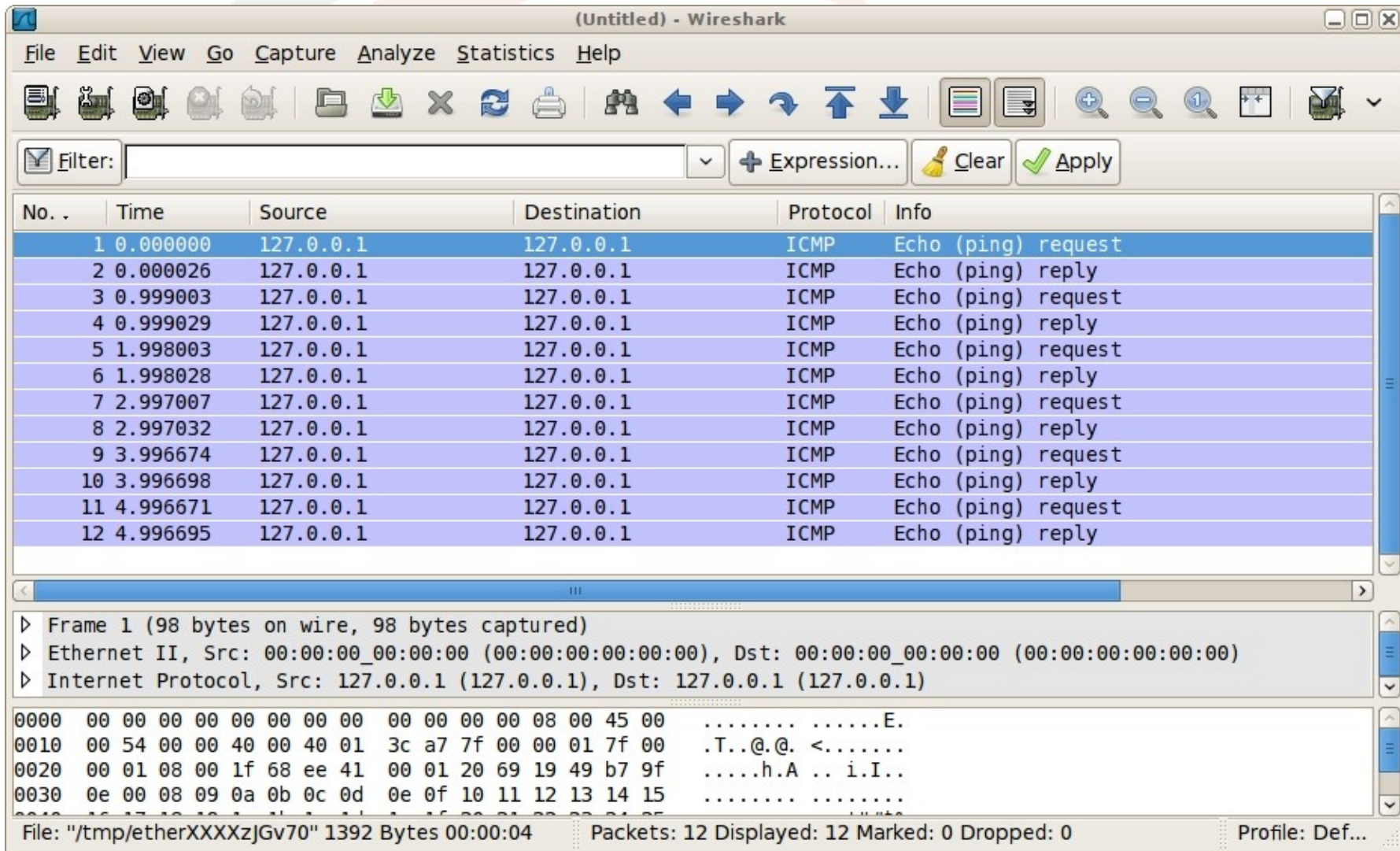
- Wireshark is a graphical packet analyser based on *libpcap*, the same library that *tcpdump* utilizes for capturing and storing packets
- The graphical interface has some advantages, including:
 - Hierarchical visualization by protocol (drill-down)
 - Follow a TCP “conversation” (Follow TCP Stream)
 - Colors to distinguish traffic types
 - Lots of statistics, graphs, etc.

wireshark

- Wireshark is what came after *Ethereal*.
- The combination of *tcpdump* and *wireshark* can be quite powerful. For example:
 - # tcpdump -i eth1 -A -s1500 -2 dump.log port 21
 - \$ sudo wireshark -r dump.log



wireshark



(Untitled) - Wireshark

File Edit View Go Capture Analyze Statistics Help

Filter: + Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	127.0.0.1	127.0.0.1	ICMP	Echo (ping) request
2	0.000026	127.0.0.1	127.0.0.1	ICMP	Echo (ping) reply
3	0.999003	127.0.0.1	127.0.0.1	ICMP	Echo (ping) request
4	0.999029	127.0.0.1	127.0.0.1	ICMP	Echo (ping) reply
5	1.998003	127.0.0.1	127.0.0.1	ICMP	Echo (ping) request
6	1.998028	127.0.0.1	127.0.0.1	ICMP	Echo (ping) reply
7	2.997007	127.0.0.1	127.0.0.1	ICMP	Echo (ping) request
8	2.997032	127.0.0.1	127.0.0.1	ICMP	Echo (ping) reply
9	3.996674	127.0.0.1	127.0.0.1	ICMP	Echo (ping) request
10	3.996698	127.0.0.1	127.0.0.1	ICMP	Echo (ping) reply
11	4.996671	127.0.0.1	127.0.0.1	ICMP	Echo (ping) request
12	4.996695	127.0.0.1	127.0.0.1	ICMP	Echo (ping) reply

Frame 1 (98 bytes on wire, 98 bytes captured)

Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)

Internet Protocol, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)

```

0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  00 54 00 00 40 00 40 01 3c a7 7f 00 00 01 7f 00  .T..@.@. <.....
0020  00 01 08 00 1f 68 ee 41 00 01 20 69 19 49 b7 9f  ....h.A .. i.I..
0030  0e 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15  .....
  
```

File: "/tmp/etherXXXXzJGv70" 1392 Bytes 00:00:04 Packets: 12 Displayed: 12 Marked: 0 Dropped: 0 Profile: Def...

iptraf

- **Many measurable statistics and functions**
 - By protocol/port
 - By packet size
 - Generates logs
 - Utilizes DNS to translate addresses
- **Advantages**
 - Simplicity
 - Menu-based (uses “curses”)
 - Flexible configuration

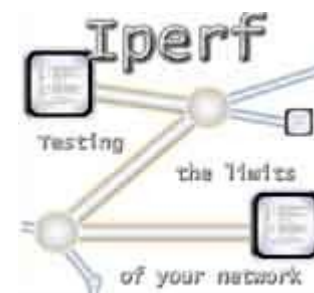
iptraf

- You can run it periodically in the background (-B)
 - It allows you, for example, to run as a cron job to periodically analyze logs.
 - Generate alarms
 - Save in a data base
 - Has a great name... “Interactive Colorful IP LAN Monitor”
 - etc...

Example: `iptraf -i eth1`

iperf

- To measure network throughput between two points
- *iperf* has two modes, *server* and *client*
- Easy to use
- Great to help determine optimal TCP parameters
 - TCP window size for optimal throughput



iperf

- Using UDP you can generate packet loss and *jitter* reports
- You can run multiple parallel sessions using *threads*
- Supports IPv6

Iperf parameters

```
Usage: iperf [-s|-c host] [options]
       iperf [-h|--help] [-v|--version]
```

Client/Server:

-f, --format	[kmKM]	format to report: Kbits, Mbits, KBytes, MBytes
-i, --interval	#	seconds between periodic bandwidth reports
-l, --len	#[KM]	length of buffer to read or write (default 8 KB)
-m, --print_mss		print TCP maximum segment size (MTU - TCP/IP header)
-p, --port	#	server port to listen on/connect to
-u, --udp		use UDP rather than TCP
-w, --window	#[KM]	TCP window size (socket buffer size)
-B, --bind	<host>	bind to <host>, an interface or multicast address
-C, --compatibility		for use with older versions does not sent extra msgs
-M, --mss	#	set TCP maximum segment size (MTU - 40 bytes)
-N, --nodelay		set TCP no delay, disabling Nagle's Algorithm
-V, --IPv6Version		Set the domain to IPv6

Server specific:

-s, --server	run in server mode
-U, --single_udp	run in single threaded UDP mode
-D, --daemon	run the server as a daemon

Client specific:

-b, --bandwidth	#[KM]	for UDP, bandwidth to send at in bits/sec (default 1 Mbit/sec, implies -u)
-c, --client	<host>	run in client mode, connecting to <host>
-d, --dualtest		Do a bidirectional test simultaneously
-n, --num	#[KM]	number of bytes to transmit (instead of -t)
-r, --tradeoff		Do a bidirectional test individually
-t, --time	#	time in seconds to transmit for (default 10 secs)
-F, --fileinput	<name>	input the data to be transmitted from a file
-I, --stdin		input the data to be transmitted from stdin
-L, --listenport	#	port to recieve bidirectional tests back on
-P, --parallel	#	number of parallel client threads to run
-T, --ttl	#	time-to-live, for multicast (default 1)

iperf - TCP

```
$ iperf -s
```

```
-----  
Server listening on TCP port 5001  
TCP window size: 85.3 KByte (default)  
-----
```

```
[ 4] local 128.223.157.19 port 5001 connected with 201.249.107.39  
port 39601  
[ 4] 0.0-11.9 sec      608 KBytes      419 Kbits/sec  
-----
```

```
# iperf -c nsrc.org
```

```
-----  
Client connecting to nsrc.org, TCP port 5001  
TCP window size: 16.0 KByte (default)  
-----
```

```
[ 3] local 192.168.1.170 port 39601 connected with 128.223.157.19  
port 5001  
[ 3] 0.0-10.3 sec      608 KBytes      485 Kbits/sec  
-----
```

Iperf - UDP

```
# iperf -c host1 -u -b100M
```

```
-----  
Client connecting to nsdb, UDP port 5001
```

```
Sending 1470 byte datagrams
```

```
UDP buffer size: 106 KByte (default)  
-----
```

```
[ 3] local 128.223.60.27 port 39606 connected with 128.223.250.135 port 5001
```

```
[ 3] 0.0-10.0 sec 114 MBytes 95.7 Mbits/sec
```

```
[ 3] Sent 81377 datagrams
```

```
[ 3] Server Report:
```

```
[ 3] 0.0-10.0 sec 114 MBytes 95.7 Mbits/sec 0.184 ms 1/81378 (0.0012%)  
-----
```

```
$ iperf -s -u -i 1
```

```
-----  
Server listening on UDP port 5001
```

```
Receiving 1470 byte datagrams
```

```
UDP buffer size: 108 KByte (default)  
-----
```

```
[ 3] local 128.223.250.135 port 5001 connected with 128.223.60.27 port 39606
```

```
[ 3] 0.0- 1.0 sec 11.4 MBytes 95.4 Mbits/sec 0.184 ms 0/ 8112 (0%)
```

```
[ 3] 1.0- 2.0 sec 11.4 MBytes 95.7 Mbits/sec 0.177 ms 0/ 8141 (0%)
```

```
[ 3] 2.0- 3.0 sec 11.4 MBytes 95.6 Mbits/sec 0.182 ms 0/ 8133 (0%)
```

```
...
```

```
[ 3] 8.0- 9.0 sec 11.4 MBytes 95.7 Mbits/sec 0.177 ms 0/ 8139 (0%)
```

```
[ 3] 9.0-10.0 sec 11.4 MBytes 95.7 Mbits/sec 0.180 ms 0/ 8137 (0%)
```

```
[ 3] 0.0-10.0 sec 114 MBytes 95.7 Mbits/sec 0.184 ms 1/81378 (0.0012%)  
-----
```

Bibliography

Monitoring Virtual Memory with vmstat

<http://www.linuxjournal.com/article/8178>

How to use TCPDump

<http://www.erg.abdn.ac.uk/users/alastair/tcpdump.html>

linux command tcpdump example

<http://smartproteam.com/linux-tutorials/linux-command-tcpdump/>

simple usage of tcpdump

<http://linux.byexamples.com/archives/283/simple-usage-of-tcpdump/>

TCPDUMP Command man page with examples

<http://www.cyberciti.biz/howto/question/man/tcpdump-man-page-with-examples.php>

TCPDump Tutorial

<http://inst.eecs.berkeley.edu/~ee122/fa06/projects/tcpdump-6up.pdf>