Network performance

- historically much network performance research was based on the assumption that network traffic was random
  - a Poisson distribution of traffic
- Paxson and Floyd 1994, Willinger 1995 found this assumption to be wrong!
  - found network traffic to be full of bursts and then idle time
- a safer method of performance prediction is to use
  - infinite sink/infinite source
  - set up a machine or program to continually transmit data
  - set up a machine or program to continually receive data

Network performance

- network performance will depend on
  - processor speed, device speed
  - timeouts, interrupt handling
  - operating system efficiency, buffers
  - application demands
  - channel capacity and channel latency

Performance

- performance management and congestion management
  - difficult issues to solve
  - instinctive answers maybe wrong or make performance worse!

- eg. adding more memory to routers may help up to a point but add too much memory and this will degrade the network performance

- Nagle (1987) discovered that if routers have an infinite amount of memory, congestion gets worse, not better, because by the time a packet reaches the front of a queue, they will have timed out, causing retransmission of the same packet. Both copies of the packet will eventually be passed from this router to the next and the new incoming packet will suffer the same fate as the outgoing packet!

Measuring network performance

- the basic loop used to improve network performance contains the following steps:
  - (i) measure the relevant network parameters and performance
  - (ii) try and understand what is going on
  - (iii) change one parameter

- repeat loop until the performance is good enough or
  - performance cannot be improved

Pitfalls to be avoided when measuring network performance

- make sure that the sample size is large enough
  - do not measure the time for one packet to be transmitted
  - rather transmit a thousand..million packets
  - divide the total time by the number of packets sent
- make sure that samples are representative
  - ideally, the whole sequence of packets should be repeated at different times of the day
  - (If a representative measurement of a multiuser network is required)
  - all this can be ignored if you are measuring an otherwise idle network with infinite source and infinite sink

Pitfalls to be avoided when measuring network performance

- be careful when using coarse grained clocks
  - computer clocks operate by adding one to a counter at regular intervals
  - incremented every millisecond
  - or updated every 1/10th second

- you need to know and this should be reflected by the total length of time of the performance run
  - otherwise if you measure the time for 10 packets to be transmitted when the clock is updated every 1/100 second
  - you might find your network can transmit ∞ packets/second!

Pitfalls to be avoided when measuring network performance

- caching can wreak havoc with measurements
  - protocol implementations might buffer and cache packets
    - hidden from the user and systems programmer
  - if you send only 1000 packets across a network
    - a sizable percentage are buffered
    - your test program might think all packets have been received
  - when in reality many are still in transit

System design for better performance

Table 1
Ethernet device driver executing on
a 80486 @ 66 MHz with a ne2000 (16 bit) NIC

Blocksize (bytes)	Throughput (k bytes/sec)
512	408
1024	528
1514	586

System design for better performance

Table 2
Ethernet device driver executing on
a Pentium @ 133 MHz with a ne2000 (16 bit) NIC

Blocksize (bytes)	Throughput (k bytes/sec)
512	597
1024	682
1514	714

System design for better performance

Table 3
Ethernet device driver executing on
a 80486 @ 66 MHz with a 3c503 (8 bit) NIC

Blocksize (bytes)	Throughput (k bytes/sec)
512	218
1024	244
1514	254

System design for better performance

- note Table 1, 2, 3: all measurements taken on an idle network with an infinite source and infinite sink

System design for better performance

- Tanenbaum’s seven rules follow
  - rule number 8 is an additional, non Tanenbaum, rule

Rule 1: CPU speed is more important than network speed

- one of the big performance problems is getting data on and off the network

- very difficult to run a network at full speed due to
  - operating system software overhead
  - device driver overhead

Rule 1: CPU speed is more important than network speed

- very often doubling the CPU speed will see a near doubling of throughput
  - eg Tables 1 and Table 2

- once the higher layers are added, expect the performance to drop further, essentially the system becomes largely defendant on CPU performance (for ne2000 cards!)

Rule 2: Reduce packet count to reduce software overhead

- from rule 1 we see that
  - reduce software overhead it is likely that throughput will rise
- one way of reducing the overhead per byte is to transmit more bytes in each packet
  - the overhead per packet is nearly constant
- eg compare throughput difference between using 512 byte frame to 1514 byte frame
  - = 178 KBytes/sec

Rule 3: Minimise context switches

- context switches and system calls are deadly to performance
  - cause a long series of cache misses
- can be minimised via buffering and not using a system that employs system calls
  - eg. microkernel
- buffering data also has the benefit of helping the system reduce packet count (rule 2)
  - but buffering may imply copying which is bad (rule 4)

Rule 4: Minimise copying

- copying is generally thought to be very bad for performance.
- whenever possible systems should pass the address of a block to be transmitted, rather than copy the block into an area for transmission
- note that protocol implementations may be forced to copy data at the data link layer, in case retransmission is necessary
- one key to performance is to keep all copying to a minimum
- this may well conflict with rule 3 and rule 2
- a careful balance needs to be struck between rules 2, 3 and 4

Rule 5: You can buy more bandwidth but not a lower delay

- the first rule states that if you want more throughput you can just buy it
- adding a second fibre optic cable next to the first one doubles the throughput
  - does nothing about the delay (latency)
- always make sure the system will perform well enough given the inherent transmission delay

Rule 6: Avoiding congestion is better than recovering from it

- when a network is congested
  - packets are lost
  - throughput is wasted
  - useless delays are introduced
- recovering from congestion is complicated and takes time
- avoid it in the first place!

Rule 7: Minimise timeouts

- timeouts are necessary in networks
  - to implement retransmission of lost packets

- when a timeout goes off and a packet is retransmitted it causes a great cost to the network

- calibrate timeouts carefully note that a timeout which takes too long to expire may cause a little extra delay for one connection, but a timeout which expires too early will use scarce CPU time on this machine and all intermediate routers!

Rule 7 (continued)

- timeout implementation is expensive in CPU time
- a Pentium 133 MHz transmitting a 64 byte frame can spend 26% of its time handling the timeout facility
  - the cost of timeout mechanism should be added to the Exec column as the timeout routines called upon the services of the executive almost exclusively in these tests
- the following table shows where processor time is being spent in a microkernel when transmitting blocks over a ne2000 running at a theoretical 100 MHz

Simulation results of a Pentium 133 MHz

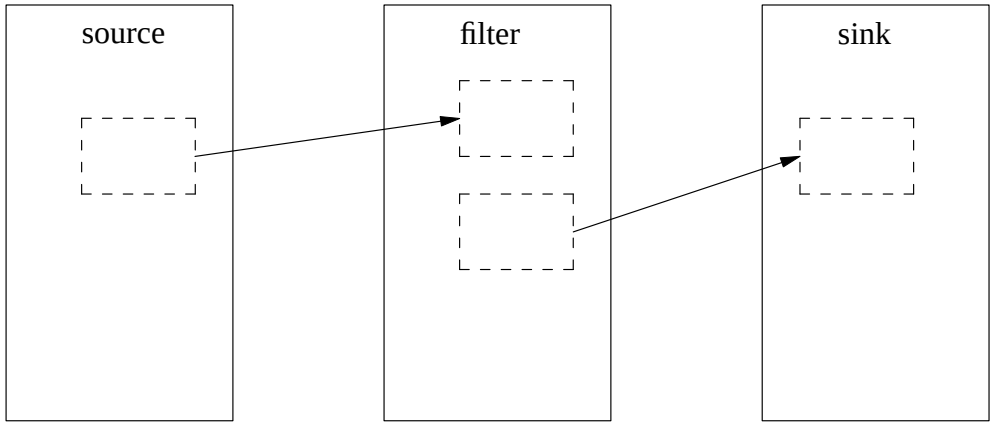
Simulated results							
Frame Bytes	Tpt KBytes	Percentage of processor time					
		ne(poll)	Exec	8390	Timers	MBuf	Sys
64	1969	24(19)	18	13	8	12	16
96	2560	33(29)	15	11	7	11	14
128	3011	41(37)	14	11	6	10	13
256	4096	59(57)	9	7	4	7	9
512	4995	75(73)	6	4	3	4	5
1024	5769	86(85)	3	2	2	2	3
1514	5937	90(89)	2	2	2	2	2

Rule 8: Avoid network hardware if possible

- (not a Tannenbaum rule)
- use network protocols to construct parallel applications, but if possible keep the client and servers on the same processor die but in a different processor core
- here are the results from writing a 3 stage pipeline on a Dual Core AMD Opteron™ processor 175 stepping 02 running at 2200 MHz with 2 GByte of RAM
  - dual core processor has 1 MByte of L2 cache, 64 KBytes L1 instruction cache and 64 KBytes data cache

Experiment conducted on a dual core Opteron using the csn message passing paradigm

- three stage pipeline (with the filter employing double buffering)



Throughput of data in relation to buffer size

Block size (bytes)	Throughput (MByte/Sec)		
	Single	Double	Quadruple
64	1	1	1
128	2	2	2
256	4	5	4
512	8	12	12
1024	14	11	23
2048	19	40	33
4096	47	37	70
8192	89	69	67
16384	137	137	178
32768	205	203	248
65536	283	274	306
131072	365	297	303
262144	329	285	257
524288	272	280	279
1048576	268	268	297

Results

Conclusions

- experiment showed CSN capable of transmitting 365 MB/Sec
  - compared to an ideal single socket, sink, source scenario which was measured at 822 MB/Sec