Tuning Systems for TCP/IP Performance on High Latency Networks

Worldwide LHC Computing Grid Tier-2 Workshop in Asia

Mark Bowden, Wenji Wu, Matt Crawford
(bowden@fnal.gov, wenji@fnal.gov, crawdad@fnal.gov)

TCP_buffer_size = RTT * BW

(round-trip time) (Bandwidth)

example: Chicago-CERN RTT = 125 msec

160 KBytes = 125 msec * 10 Mbps
1.6 MBytes = 125 msec * 100 Mbps
16 MBytes = 125 msec * 1000 Mbps
160 MBytes = 125 msec * 10000 Mbps

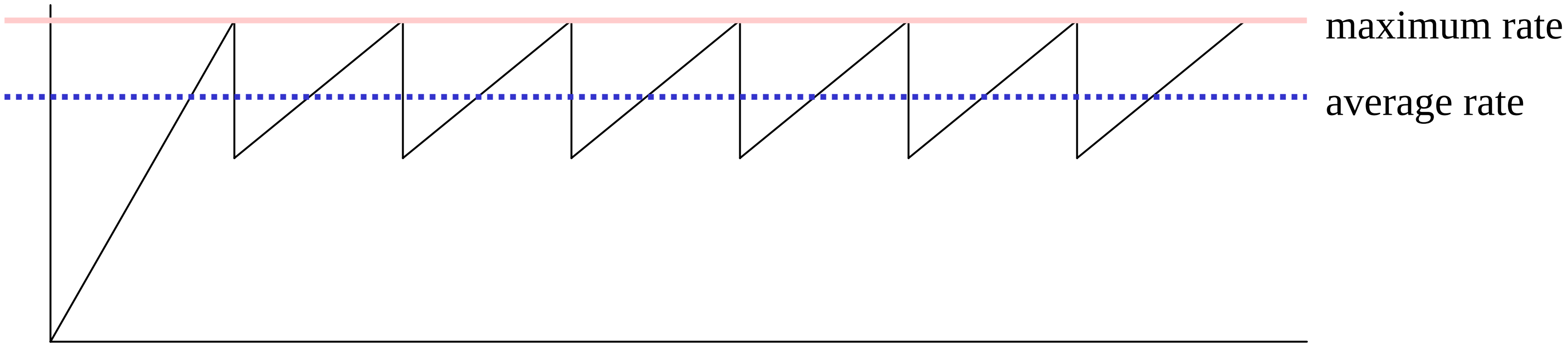
174 KBytes = 125 msec * 11 Mbps

(default tcp_rmem*)

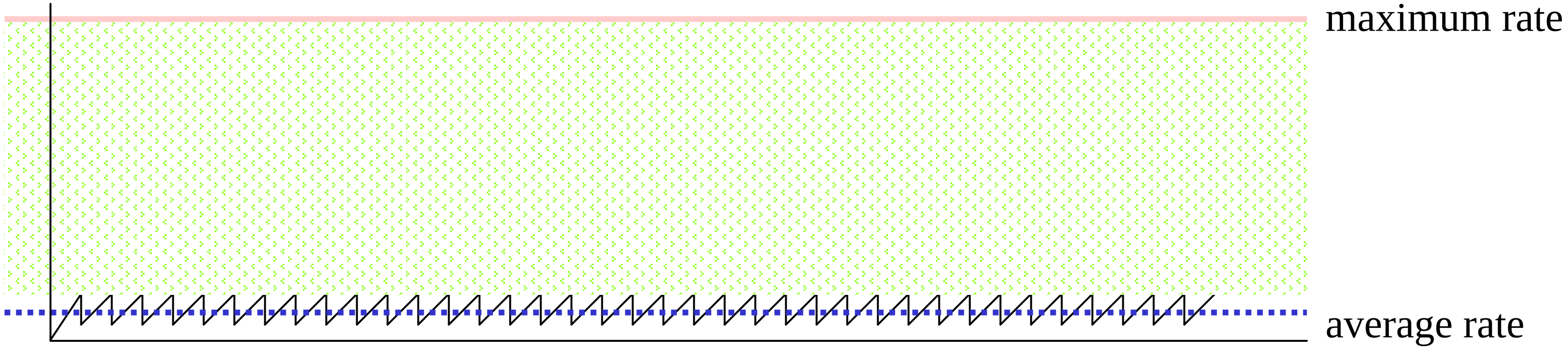
* increased to 2MBytes in 2.6.18?

TCP Bandwidth

(simplified)

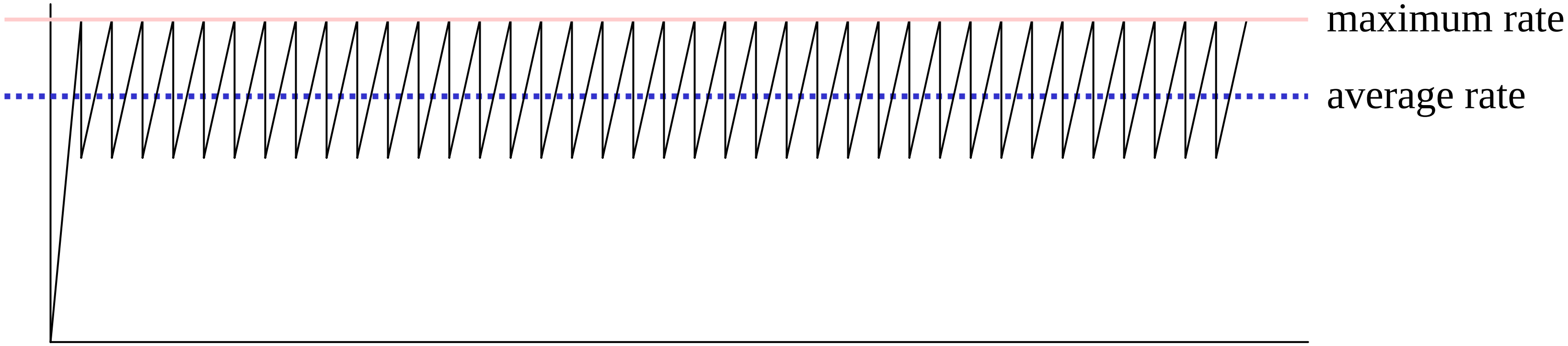


TCP bandwidth (single high-speed stream)

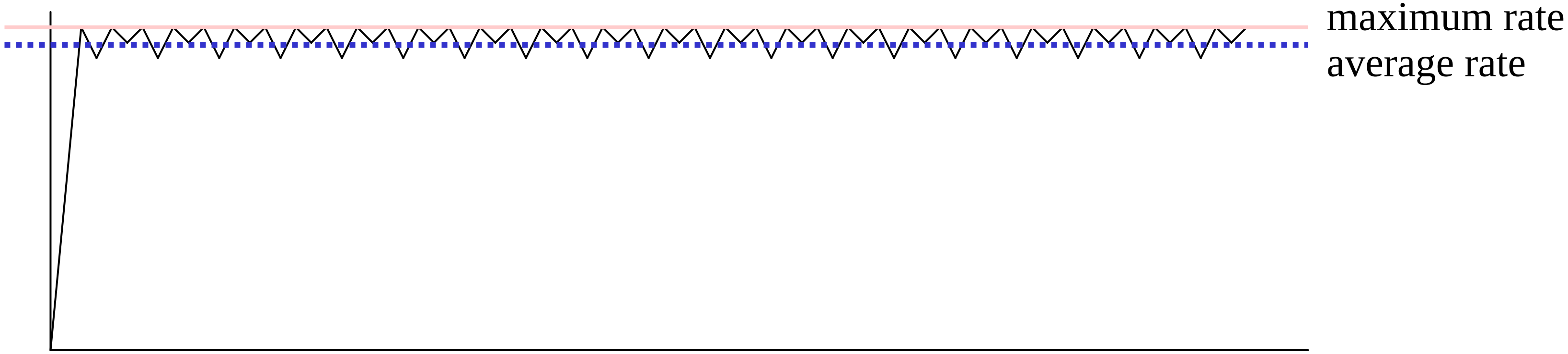


TCP bandwidth (single low-speed stream)

TCP Bandwidth



TCP bandwidth (multiple low-speed streams added in phase)



TCP bandwidth (multiple low-speed streams added out of phase)

=> Total TCP bandwidth improves with multiple streams.

Commonly used approach (GridFTP, etc).

What is the ideal buffer size for multiple streams?

Could just set buffers to 16MBytes

- good for any rate from 0-1Gbps per stream
- typical value recommended by tuning guides for high RTT-BW links

Tried this...machines were unstable.

Why?

Log shows memory allocation failures.

Assumption is out-of-memory condition in kernel (low) memory.

Invoking OOM-Killer? (probably not)

“Bad things happen when you're out of lowmem.” - Linux documentation

log (1)

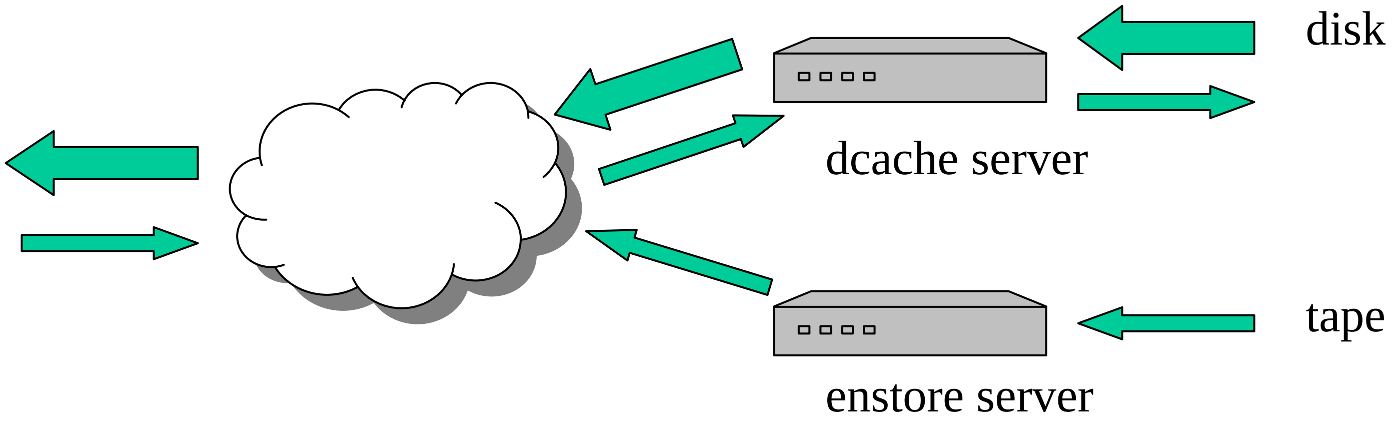
```
Aug  4 17:42:34 cmsstor26 kernel: java: page allocation failure. order:0, mode:0x20
...
Aug  4 17:42:38 cmsstor26 kernel: java: page allocation failure. order:0, mode:0x20
...
Aug  4 17:42:41 cmsstor26 kernel: java: page allocation failure. order:0, mode:0x20
Aug  4 17:42:41 cmsstor26 kernel:  [<c013e7ab>] __alloc_pages+0x35e/0x406
Aug  4 17:42:41 cmsstor26 kernel:  [<c0140f89>] kmem_getpages+0x30/0x85
Aug  4 17:42:41 cmsstor26 kernel:  [<c0141bcd>] cache_grow+0xb2/0x154
Aug  4 17:42:41 cmsstor26 kernel:  [<c0141e4e>] cache_alloc_refill+0x1df/0x21c
Aug  4 17:42:41 cmsstor26 kernel:  [<c014214a>] __kmalloc+0x7f/0x81
Aug  4 17:42:41 cmsstor26 kernel:  [<c0293f23>] alloc_skb+0x35/0xc9
Aug  4 17:42:41 cmsstor26 kernel:  [<f88f5dcd>] e1000_alloc_rx_buffers+0x64/0x38c [e1000]
Aug  4 17:42:41 cmsstor26 kernel:  [<c0108f38>] convert_fxsr_to_user+0x122/0x180
Aug  4 17:42:41 cmsstor26 kernel:  [<f88f55ca>] e1000_clean_rx_irq+0x210/0x508 [e1000]
Aug  4 17:42:41 cmsstor26 kernel:  [<f88f4cf6>] e1000_clean+0x42/0xe6 [e1000]
Aug  4 17:42:41 cmsstor26 kernel:  [<c02999a0>] net_rx_action+0x88/0x16e
Aug  4 17:42:41 cmsstor26 kernel:  [<c011f589>] __do_softirq+0x69/0xd5
Aug  4 17:42:41 cmsstor26 kernel:  [<c011f627>] do_softirq+0x32/0x34
Aug  4 17:42:41 cmsstor26 kernel:  [<c0104c95>] do_IRQ+0x35/0x5a
Aug  4 17:42:41 cmsstor26 kernel:  [<c01034c6>] common_interrupt+0x1a/0x20
Aug  4 17:42:41 cmsstor26 kernel:  [<c0101ec2>] restore_sigcontext+0x3a/0x159
Aug  4 17:42:41 cmsstor26 kernel:  [<c0102164>] sys_rt_sigreturn+0xad/0xed
Aug  4 17:42:41 cmsstor26 kernel:  [<c0102a91>] syscall_call+0x7/0xb
Aug  4 17:42:41 cmsstor26 kernel: Mem-info:
Aug  4 17:42:41 cmsstor26 kernel: DMA per-cpu:
Aug  4 17:42:41 cmsstor26 kernel: cpu 0 hot: low 2, high 6, batch 1 used:5
Aug  4 17:42:41 cmsstor26 kernel: cpu 0 cold: low 0, high 2, batch 1 used:1
Aug  4 17:42:42 cmsstor26 kernel: cpu 1 hot: low 2, high 6, batch 1 used:2
Aug  4 17:42:42 cmsstor26 kernel: cpu 1 cold: low 0, high 2, batch 1 used:1
Aug  4 17:42:42 cmsstor26 kernel: Normal per-cpu:
Aug  4 17:42:42 cmsstor26 kernel: cpu 0 hot: low 62, high 186, batch 31 used:157
Aug  4 17:42:42 cmsstor26 kernel: cpu 0 cold: low 0, high 62, batch 31 used:43
Aug  4 17:42:42 cmsstor26 kernel: cpu 1 hot: low 62, high 186, batch 31 used:100
Aug  4 17:42:42 cmsstor26 kernel: cpu 1 cold: low 0, high 62, batch 31 used:56
```

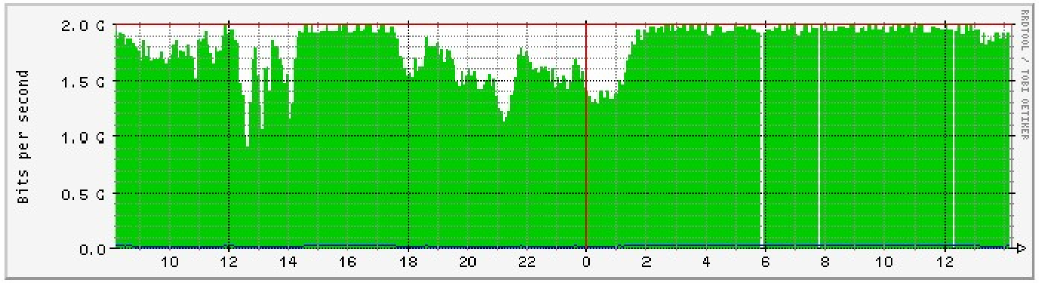
log (2)

```
Aug  4 17:42:42 cmsstor26 kernel: HighMem per-cpu:
Aug  4 17:42:42 cmsstor26 kernel: cpu 0 hot: low 62, high 186, batch 31 used:68
Aug  4 17:42:42 cmsstor26 kernel: cpu 0 cold: low 0, high 62, batch 31 used:57
Aug  4 17:42:42 cmsstor26 kernel: cpu 1 hot: low 62, high 186, batch 31 used:77
Aug  4 17:42:42 cmsstor26 kernel: cpu 1 cold: low 0, high 62, batch 31 used:13
Aug  4 17:42:42 cmsstor26 kernel: Free pages:       18268kB (744kB HighMem)
Aug  4 17:42:42 cmsstor26 kernel: Active:513325 inactive:1421680 dirty:134572 writeback:18101
unstable:0 free:4567 slab:130154 mapped:512806 pagetables:1832
Aug  4 17:42:42 cmsstor26 kernel: DMA free:5276kB min:4680kB low:5848kB high:7020kB active:0kB
inactive:0kB present:16384kB pages_scanned:21 all_unreclaimable? yes
Aug  4 17:42:42 cmsstor26 kernel: lowmem_reserve[]: 0 880 8944
Aug  4 17:42:42 cmsstor26 kernel: Normal free:12248kB min:32768kB low:40960kB high:49152kB
active:144176kB inactive:144684kB present:901120kB pages_scanned:0 all_unreclaimable? no
Aug  4 17:42:42 cmsstor26 kernel: lowmem_reserve[]: 0 0 64512
Aug  4 17:42:42 cmsstor26 kernel: HighMem free:744kB min:512kB low:640kB high:768kB
active:1909124kB inactive:5542036kB present:8257536kB pages_scanned:0 all_unreclaimable? no
Aug  4 17:42:42 cmsstor26 kernel: lowmem_reserve[]: 0 0 0
Aug  4 17:42:42 cmsstor26 kernel: DMA: 1*4kB 1*8kB 1*16kB 0*32kB 0*64kB 1*128kB 0*256kB 0*512kB
1*1024kB 0*2048kB 1*4096kB = 5276kB
Aug  4 17:42:42 cmsstor26 kernel: Normal: 0*4kB 1*8kB 1*16kB 0*32kB 1*64kB 1*128kB 1*256kB
3*512kB 2*1024kB 2*2048kB 1*4096kB = 12248kB
Aug  4 17:42:42 cmsstor26 kernel: HighMem: 0*4kB 1*8kB 16*16kB 1*32kB 3*64kB 0*128kB 1*256kB
0*512kB 0*1024kB 0*2048kB 0*4096kB = 744kB
...
Aug  4 17:42:43 cmsstor26 kernel: java: page allocation failure. order:0, mode:0x20
...
Aug  4 17:42:46 cmsstor26 kernel: swapper: page allocation failure. order:0, mode:0x20
...
Aug  4 17:42:50 cmsstor26 kernel: swapper: page allocation failure. order:0, mode:0x20
...
```

dual Xeon 3GHz, 8 GByte memory, Linux 2.6.13-2smp
2 x 1Gbps (bonded) network interface, NAPI
Fibre-channel disk controller

application - dcache server (most network traffic is outgoing)



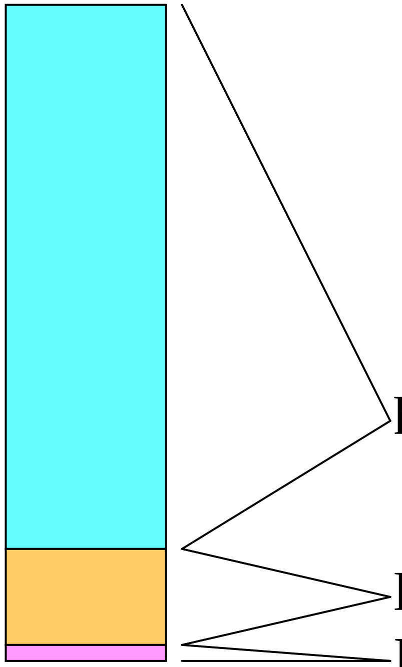


32 bit, 8GByte Memory Map

dcache application

Linux disk cache

networking buffers,
page tables, etc



high mem - 7 GBytes

low mem - 896 MBytes

DMA mem - 16 MBytes

- network buffers are in low mem ... < 600 MBytes available

Failure Scenario

- TCP queues fill up, using most of kernel memory

- memory allocation failures when trying to get additional RX buffers

- goes into swap to try to free memory, but this just adds to CPU load and increases RX buffer backlog (nothing swappable in low memory)

TX buffers (normal load, 128KByte buffers)

```
Proto  Recv-Q        Send-Q        Local Address              Foreign Address          State
...
tcp         0         82389        cmsstor26.fnal.gov:44803  lxfsra0802.cern.ch:20003 ESTABLISHED
tcp         0         92646        cmsstor26.fnal.gov:44805  lxfsra0802.cern.ch:20003 ESTABLISHED
tcp         0         86769        cmsstor26.fnal.gov:44804  lxfsra0802.cern.ch:20003 ESTABLISHED
tcp         0         82389        cmsstor26.fnal.gov:44807  lxfsra0802.cern.ch:20003 ESTABLISHED
tcp         0         89393        cmsstor26.fnal.gov:44806  lxfsra0802.cern.ch:20003 ESTABLISHED
tcp         0         92646        cmsstor26.fnal.gov:44790  lxfsrc4401.cern.ch:20003 ESTABLISHED
tcp         0         92646        cmsstor26.fnal.gov:44791  lxfsrc4401.cern.ch:20003 ESTABLISHED
tcp         0         92646        cmsstor26.fnal.gov:44789  lxfsrc4401.cern.ch:20003 ESTABLISHED
tcp         0         95270        cmsstor26.fnal.gov:44792  lxfsrc4401.cern.ch:20003 ESTABLISHED
tcp         0         92646        cmsstor26.fnal.gov:44793  lxfsrc4401.cern.ch:20003 ESTABLISHED
tcp         0         73481        cmsstor26.fnal.gov:44799  lxfsrc4205.cern.ch:20000 ESTABLISHED
tcp         0         71984        cmsstor26.fnal.gov:44798  lxfsrc4205.cern.ch:20000 ESTABLISHED
tcp         0         74793        cmsstor26.fnal.gov:44802  lxfsrc4205.cern.ch:20000 ESTABLISHED
tcp         0         82241        cmsstor26.fnal.gov:44800  lxfsrc4205.cern.ch:20000 ESTABLISHED
...
```

- TX buffers contain data which has not been acknowledged

- even under normal load, the TX buffers are using memory

- we DON'T see memory allocation failures for TX buffers

  (system has control of TX rate and can limit buffer requests)

RX buffers (high load, 1MByte buffers)

```
Proto  Recv-Q        Send-Q        Local Address              Foreign Address           State
...
tcp   1897545             0        cmsstor26.fnal.gov:33776  t2gw02.t2.ucsd.edu:41428 ESTABLISHED
tcp   1881263             0        cmsstor26.fnal.gov:33777  t2gw02.t2.ucsd.edu:41427 ESTABLISHED
tcp   1897545             0        cmsstor26.fnal.gov:33776  t2gw02.t2.ucsd.edu:41426 ESTABLISHED
tcp   1863367             0        cmsstor26.fnal.gov:33777  t2gw02.t2.ucsd.edu:41425 ESTABLISHED
tcp   1897545             0        cmsstor26.fnal.gov:33776  t2gw02.t2.ucsd.edu:41424 ESTABLISHED
tcp   1740696             0        cmsstor26.fnal.gov:33775  t2gw02.t2.ucsd.edu:40927 ESTABLISHED
tcp   1880671             0        cmsstor26.fnal.gov:33777  t2gw02.t2.ucsd.edu:41415 ESTABLISHED
tcp   1877031             0        cmsstor26.fnal.gov:33776  t2gw02.t2.ucsd.edu:41414 ESTABLISHED
tcp   1895937             0        cmsstor26.fnal.gov:33777  t2gw02.t2.ucsd.edu:41413 ESTABLISHED
tcp   1877031             0        cmsstor26.fnal.gov:33776  t2gw02.t2.ucsd.edu:41412 ESTABLISHED
tcp   1858697             0        cmsstor26.fnal.gov:33777  t2gw02.t2.ucsd.edu:41411 ESTABLISHED
tcp   1877031             0        cmsstor26.fnal.gov:33776  t2gw02.t2.ucsd.edu:41410 ESTABLISHED
tcp   1805465             0        cmsstor26.fnal.gov:33777  t2gw02.t2.ucsd.edu:41409 ESTABLISHED
tcp   1866774             0        cmsstor26.fnal.gov:33776  t2gw02.t2.ucsd.edu:41408 ESTABLISHED
...
```

- RX buffers contain data which has been received, but not offloaded to application

- we DO see memory allocation failures for RX buffers

(system has less control of RX rate)

Avoiding Out-of-Memory Situation

- with large buffers AND many connections, it's easy to exhaust the TCP buffer space in lowmem ... example: 20 connections x 16MBytes x 2 = 640MBytes

- for a typical Gridftp session, the bandwidth is divided into ~20 parallel streams ... 1MByte buffers are sufficient

notes: 1) receive buffers fill up when processor is busy

2) transmit buffers can fill up in normal operation

3) 600 MBytes = 3 seconds (@2Gbps)

Avoiding Out-of-Memory Situation

- test...maximum TCP buffer sizes set to lower values (suitable for multiple connections, with remote data sources and local destinations)

tcp_wmem = 4096 32768 131072

tcp_rmem = 4096 87380 1048576

=> system up for 20+ days

- then tried increasing TCP receive buffer maximum by 4X

tcp_wmem = 4096 32768 131072

tcp_rmem = 4096 87380 4194304

=> system crashed after a few hours

Out-of-Memory

Why doesn’t Linux just drop packets instead of running out of memory?

Appears that default setting for *tcp_mem* (total TCP memory) is too high

- set by system at startup
- 1.6 GBytes on 32 bit highmem machine (only 600 MBytes of kernel memory available)
- should be reduced by a factor of 4

Out-of-Memory

tcp_mem test

- 256MByte machine
- *tcp_mem* at default value of 98304 pages (400MBytes)
- *tcp_rmem* set to 80MBytes
- 5 TCP connections

=> out-of-memory

- reduced *tcp_mem* to 40960 (165 MBytes)

=> stable for any number of connections

Hidden Problems?

```
Proto Recv-Q Send-Q Local Address               Foreign Address             State
...
tcp        0 370841 cmsstor26.fnal.gov:33302    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 286161 cmsstor26.fnal.gov:33303    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 319741 cmsstor26.fnal.gov:33300    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 337261 cmsstor26.fnal.gov:33301    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 386901 cmsstor26.fnal.gov:33298    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 338721 cmsstor26.fnal.gov:33299    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 300761 cmsstor26.fnal.gov:33296    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 287621 cmsstor26.fnal.gov:33297    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 262801 cmsstor26.fnal.gov:33310    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 388361 cmsstor26.fnal.gov:33311    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 389821 cmsstor26.fnal.gov:33308    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 389821 cmsstor26.fnal.gov:33309    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 357701 cmsstor26.fnal.gov:33306    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 287621 cmsstor26.fnal.gov:33307    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 379601 cmsstor26.fnal.gov:33304    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 388361 cmsstor26.fnal.gov:33305    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 386901 cmsstor26.fnal.gov:33294    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 261341 cmsstor26.fnal.gov:33295    thpc-1.unl.edu:20001        FIN_WAIT1
tcp        0 268641 cmsstor26.fnal.gov:33293    thpc-1.unl.edu:20001        FIN_WAIT1
...
```

- netstat shows a group of connections in FIN_WAIT1 state with full send queues

- appears to last indefinitely (days)

- with large *tcp_wmem*, this condition could saturate low memory

RX Overruns

With lower values of *tcp_mem*, *tcp_rmem* and *tcp_wmem*, machine is stable (no out-of-memory conditions)...

but there is still a problem with RX overruns.

- overruns occur at rates > 200 Mbps (relatively low)

- worse when machine is under load

- not apparent unless you examine ifconfig (silent packet drops)

RX Overruns

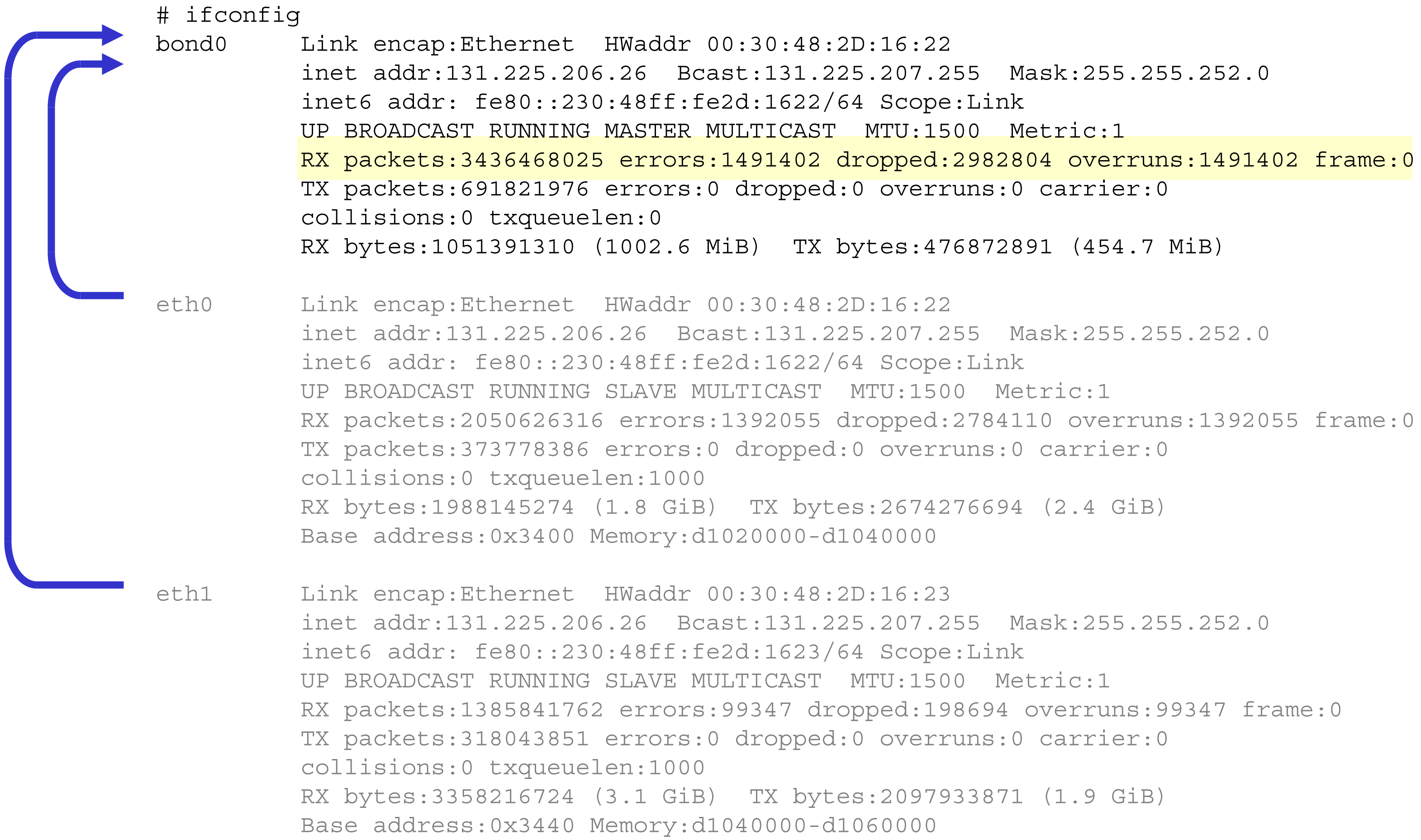
Reasons for lower RX rate

- RX takes significantly more processing time

- kernel and driver both give TX priority over RX
  ...not what you would normally expect!

- Linux memory management not optimized for networking

RX Overruns



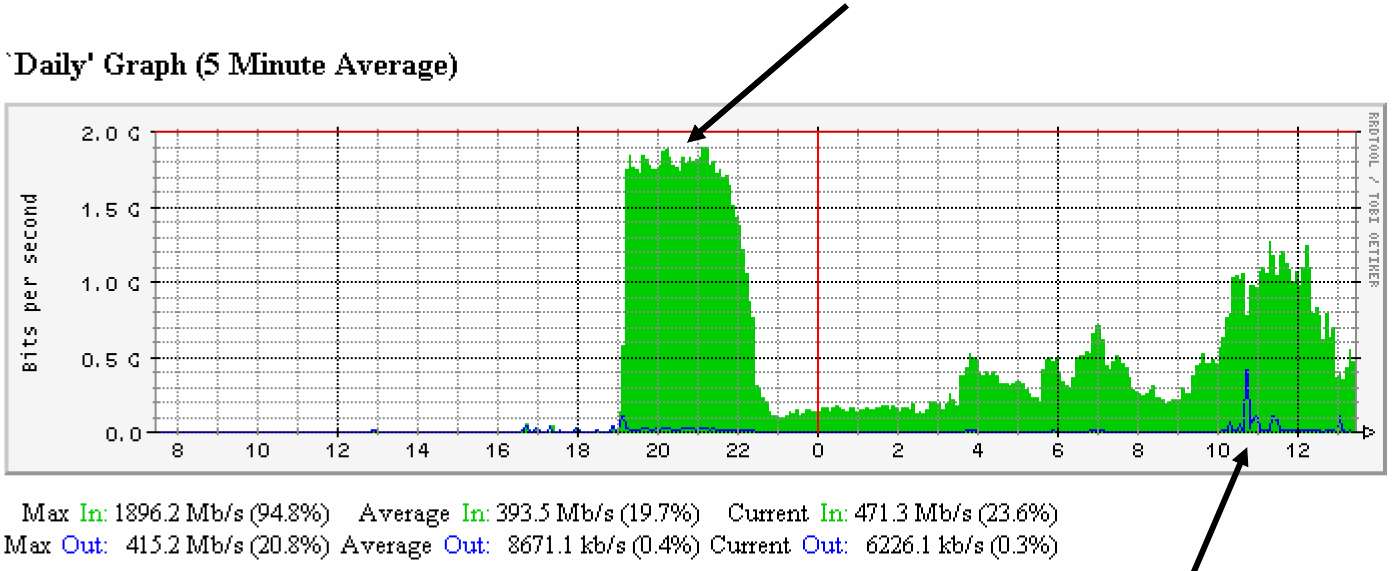
```
# ifconfig
bond0     Link encap:Ethernet  HWaddr 00:30:48:2D:16:22
          inet addr:131.225.206.26  Bcast:131.225.207.255  Mask:255.255.252.0
          inet6 addr: fe80::230:48ff:fe2d:1622/64 Scope:Link
          UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
          RX packets:3436468025 errors:1491402 dropped:2982804 overruns:1491402 frame:0
          TX packets:691821976 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1051391310 (1002.6 MiB)  TX bytes:476872891 (454.7 MiB)

eth0      Link encap:Ethernet  HWaddr 00:30:48:2D:16:22
          inet addr:131.225.206.26  Bcast:131.225.207.255  Mask:255.255.252.0
          inet6 addr: fe80::230:48ff:fe2d:1622/64 Scope:Link
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:2050626316 errors:1392055 dropped:2784110 overruns:1392055 frame:0
          TX packets:373778386 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1988145274 (1.8 GiB)  TX bytes:2674276694 (2.4 GiB)
          Base address:0x3400 Memory:d1020000-d1040000

eth1      Link encap:Ethernet  HWaddr 00:30:48:2D:16:23
          inet addr:131.225.206.26  Bcast:131.225.207.255  Mask:255.255.252.0
          inet6 addr: fe80::230:48ff:fe2d:1623/64 Scope:Link
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:1385841762 errors:99347 dropped:198694 overruns:99347 frame:0
          TX packets:318043851 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3358216724 (3.1 GiB)  TX bytes:2097933871 (1.9 GiB)
          Base address:0x3440 Memory:d1040000-d1060000
```

RX Overruns

Full rate TX (no errors)


`Daily' Graph (5 Minute Average)

Max In: 1896.2 Mb/s (94.8%) Average In: 393.5 Mb/s (19.7%) Current In: 471.3 Mb/s (23.6%)
Max Out: 415.2 Mb/s (20.8%) Average Out: 8671.1 kb/s (0.4%) Current Out: 6226.1 kb/s (0.3%)

RX activity > ~200 Mbps => RX overruns, dropped packets

RX Overruns

```
# vmstat 1
procs  -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 0  9 453420 539524   1832 5622468    0    0 78336     0 4952  3724 16 11 34 39
 1 10 453420 467852   1832 5693968    0    0 71680     0 4539  3322 11 11 45 32
 0 11 453420 392792   2036 5767604    0    0 73992   133 4947  3867 19 13  2 67
 1  9 453420 328312   2200 5832180    0    0 64576    32 3791  3319 11 11 20 58
13 10 453420 695972   2216 5462964    0    0 72800   427 4906  3989 25 18 23 34
 9 10 453420 648108   2248 5509472    0    0 46740     0 4681  3199 90 10  0  0
10  8 453420 573956   2296 5583524    0    0 74088     0 5177  3756 77 14  0  9
17 10 453420 522124   2328 5633932    0    0 50396   352 5031  3363 85 15  0  0
16  8 453420 475268   2344 5680716    0    0 46832     0 5164  3283 87 13  0  0
13  9 453420 419592   2344 5732976    0    0 52284   192 4327  3069 88 11  0  0
 2 10 452968 324940   2360 5826892    0    0 93928     0 5005  3840 65 14  4 16
 0  9 452968 312664   2356 5840936    0    0 81368    24 5093  4534 37 13  7 44
 0  9 452968 312476   2360 5841712    0    0 77600   176 4802  4866 18 15  6 60
```

I/O, CPU

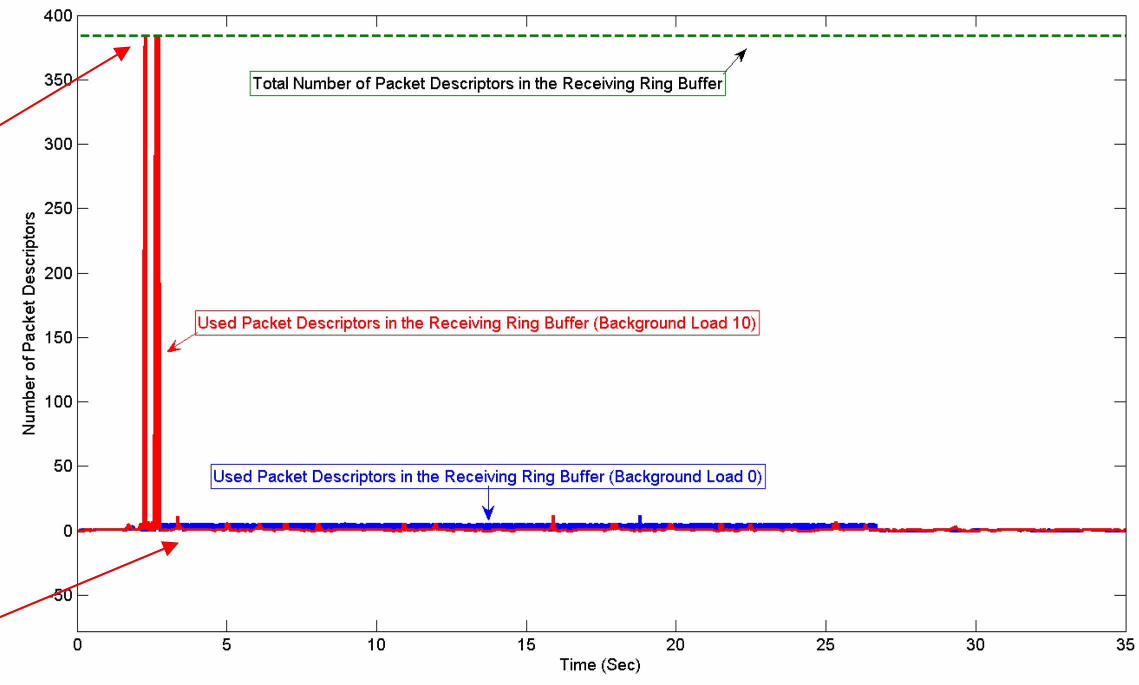
process backlog

high disk read rate (TX)

```
top - 08:55:45 up 11:07,  1 user,  load average: 19.21, 16.71, 9.73
Tasks: 111 total,   1 running, 110 sleeping,   0 stopped,   0 zombie
Cpu(s):  4.6% us, 10.1% sy,  0.0% ni,  0.0% id, 75.9% wa,  0.5% hi,  8.9% si
Mem:   8310800k total,  8168972k used,   141828k free,     2788k buffers
Swap: 17133272k total,   479180k used, 16654092k free,  5320120k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
12787 root      19   0  718m 481m  12m S 17.9  5.9  53:01.88 java
26233 root      25   0  710m 460m  12m S 10.9  5.7  48:58.19 java
 2969 root      25   0  676m 428m  12m S  1.3  5.3  51:57.64 java
```

RX Overruns



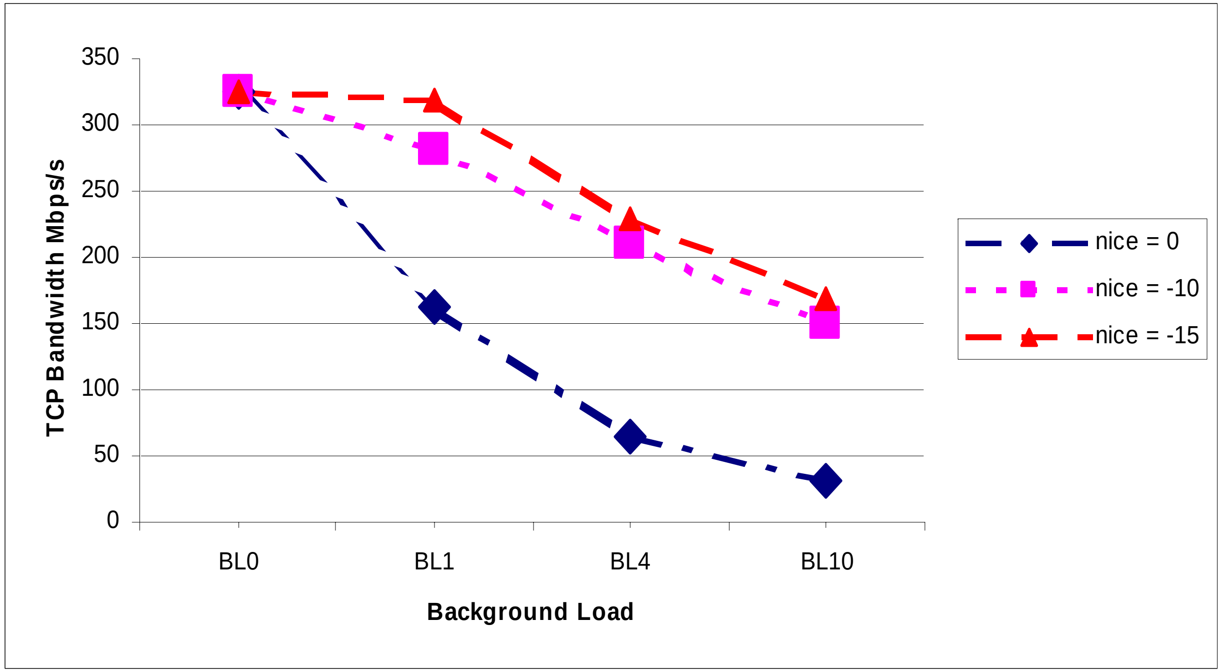
Running out packet descriptors

TCP throttles rate to avoid loss

- TCP doesn’t guarantee no packet loss...under load, the receive ring buffer can fill up, dropping packets until TCP adjusts the window size

Network Performance vs Process Priority

Nice value	Time slice
+19	5 *ms*
0	100 *ms*
-10	600 *ms*
-15	700 *ms*
-20	800 *ms*

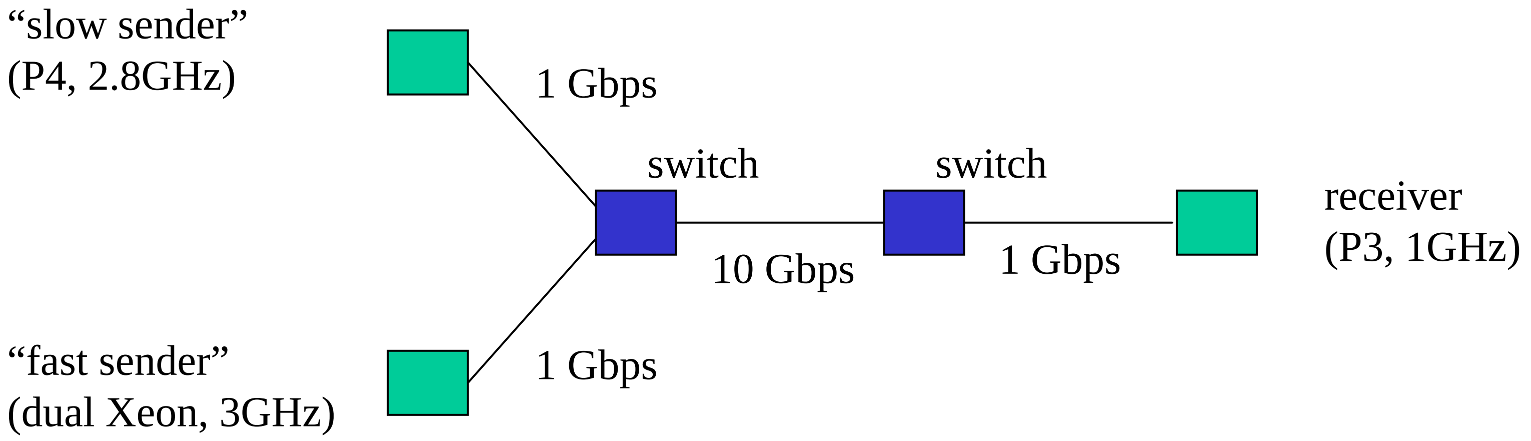


- increasing the priority (reducing the “nice” value) of the receiving process can improve TCP performance

“The Performance Analysis of Linux Networking –Packet Receiving”
Wu, Crawford, Bowden, International Journal of Computer Communications (in press)

Network Performance due to Linux Scheduler

(slower $\overset{?}{=}$ faster)



“slow sender”
(P4, 2.8GHz)

1 Gbps

switch

switch

receiver
(P3, 1GHz)

10 Gbps

1 Gbps

“fast sender”
(dual Xeon, 3GHz)

1 Gbps

“Interactivity vs. Fairness in Networked Linux Systems”, Wu, Crawford
[submitted to "The International Journal of Computer Networks (Elsevier)", September 2006]

Network Performance due to Linux Scheduler

- Linux classifys processes as “interactive” or “non-interactive” based on their run/sleep characteristics.

- a process that wakes up for a short period of time to handle an I/O event (e.g., pressing a key on the keyboard) and then goes back to sleep is an “interactive” process.

- a process that runs continuously during its timeslice is a “non-interactive” process.

- depending on the packet arrival rate, a network receiving process can end up being classified as either “interactive” or “non-interactive”.

- the longer the gap between packets, the more likely that the receive process will be classified as “interactive”...this can happen even if the process is running for 90% of it’s timeslice.

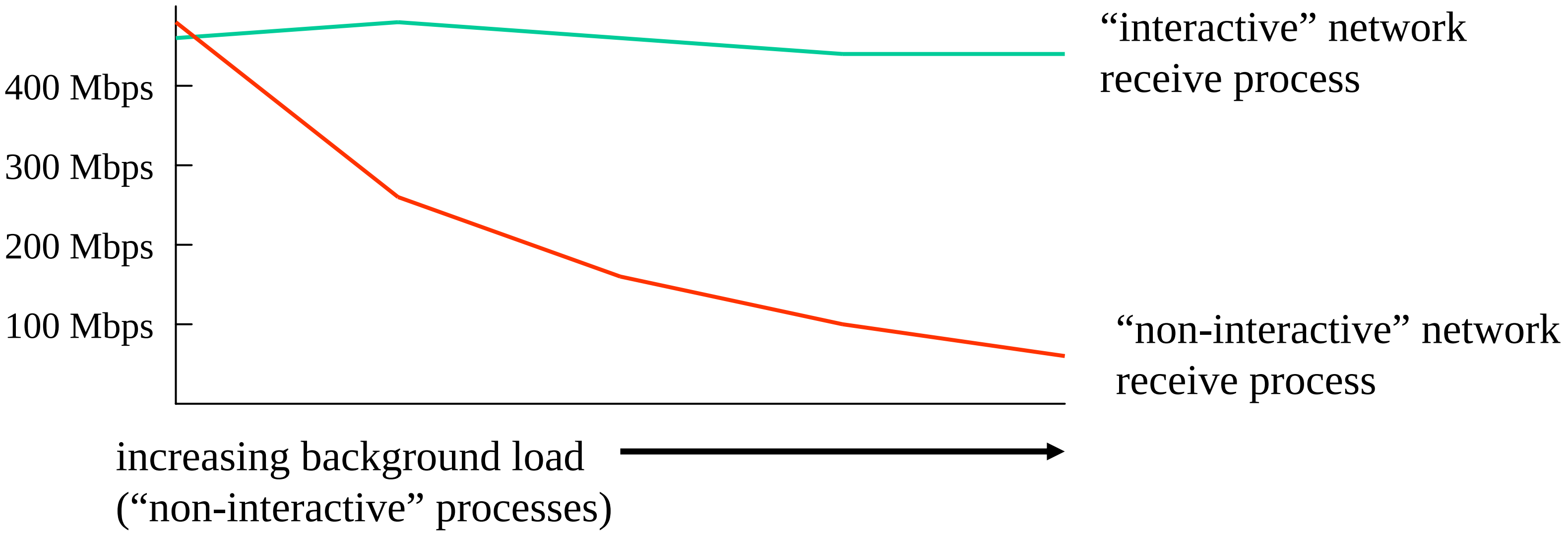
Network Performance due to Linux Scheduler

- to minimize response latency, “interactive” processes are allowed to run more often and are given priority over “non-interactive” processes.

- by rate limiting the sender, it is theoretically possible to cause reclassification of the receive process (from “non-interactive” to “interactive”), and yield a higher total transfer bandwidth...especially in the presence of a high background load in the receiver.

- this is not a recommended method for increasing throughput, since it relies on misclassification of the receive process...interrupt coalescing, jumbo frames, TCP/IP offloading, lowering the data receiving process’ nice value or simply reducing the receiver background load are all preferred.

Network Performance due to Linux scheduler



- the "interactive" network receive process has fairly constant share of CPU time

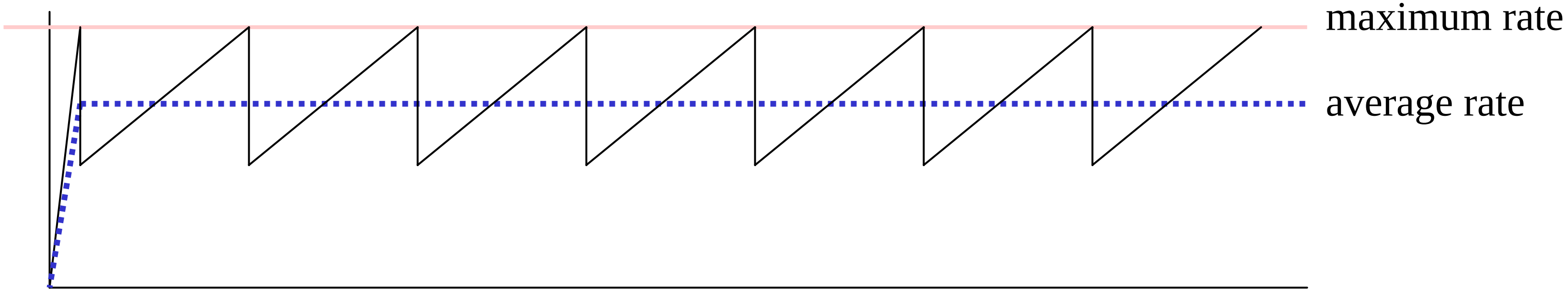
- if there are no other processes running, the "non-interactive" receive process gets a similar CPU share

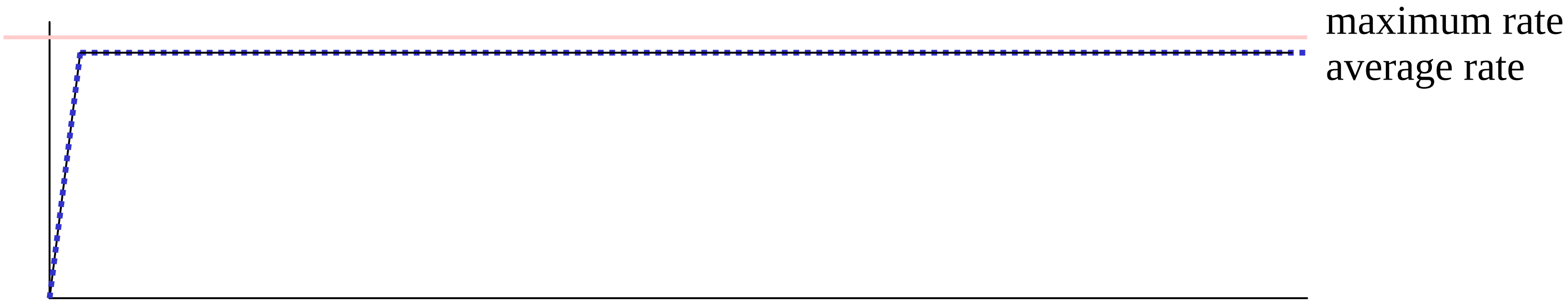
- if there are additional processes running, the "non-interactive" receive process competes for time with other non-interactive processes

Rate Limiting

- rate limiting a “fast” sender can improve throughput

- mainly useful on links with dedicated bandwidth, if the transfer is receiver limited





Rate Limiting

- to add a (400 Mbps) rate limit for traffic to destination xx.xx.xx.xx

```
#tc qdisc add dev eth0 root handle 1:0 htb
#tc class add dev eth0 parent 1:0 classid 1:1 \
       htb rate 400mbit ceil 400mbit
#tc filter add dev eth0 parent 1:0 protocol ip \
       prio 0 u32 match ip dst xx.xx.xx.xx flowid 1:1
```

- to remove the rate limit

```
#tc qdisc del dev eth0 root
```

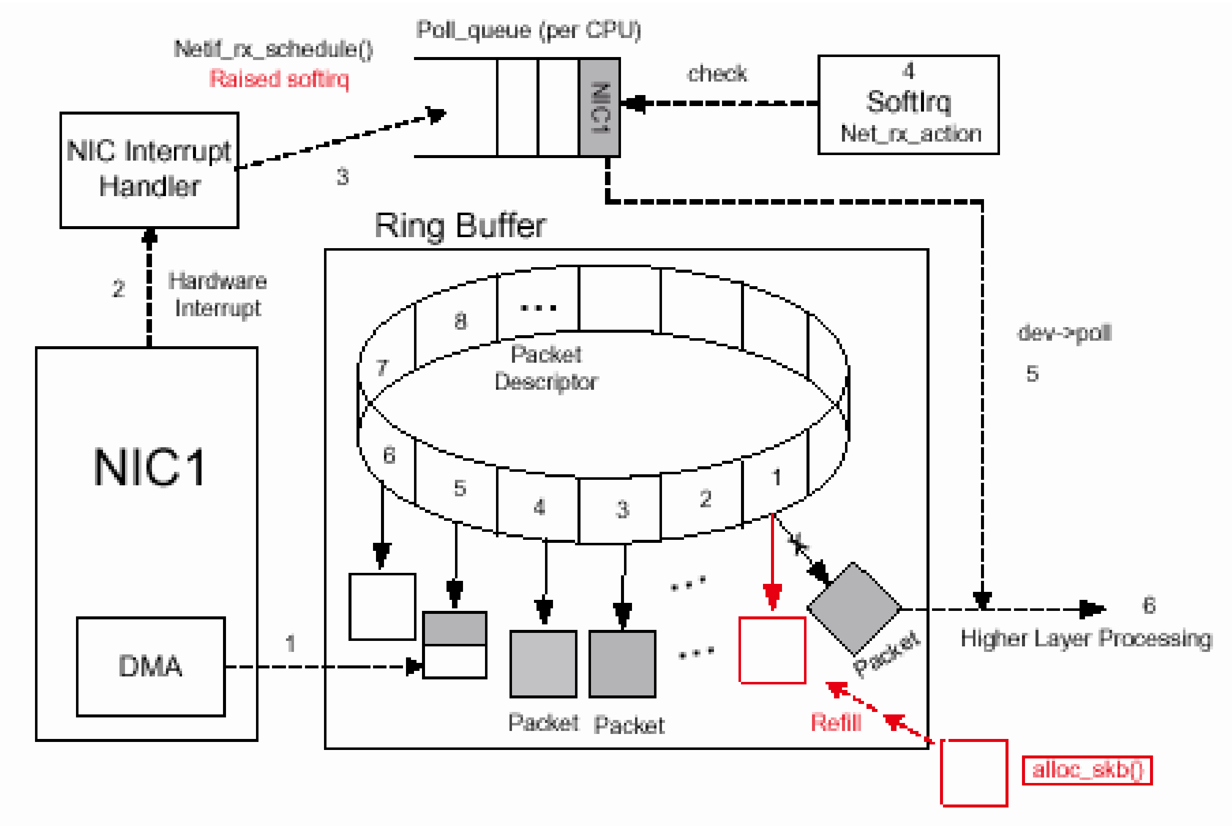
e1000 (Intel NIC) Driver

- using NAPI driver (hardware interrupt on first packet, polling of subsequent packets)

- TX has higher priority than RX (TX processing first in softirq handler)

- driver slow to release empty slots in RX ring buffer. Dropping the NAPI “weight” to 16 should improve this...requires driver recompile.

NAPI driver (RX)



- converges to interrupt driven system at low packet rates

- converges to polled system at high packet rates

NAPI Driver

- we see interrupt rates as high as 20KHz on 32 bit machines (Xeon, Intel NIC), 35KHz on 64 bit machines (Opteron, 3Com NIC)

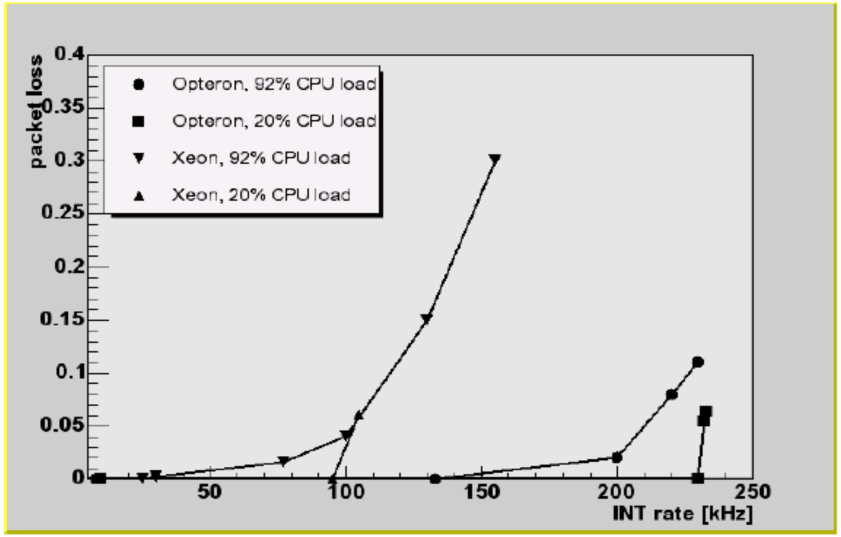
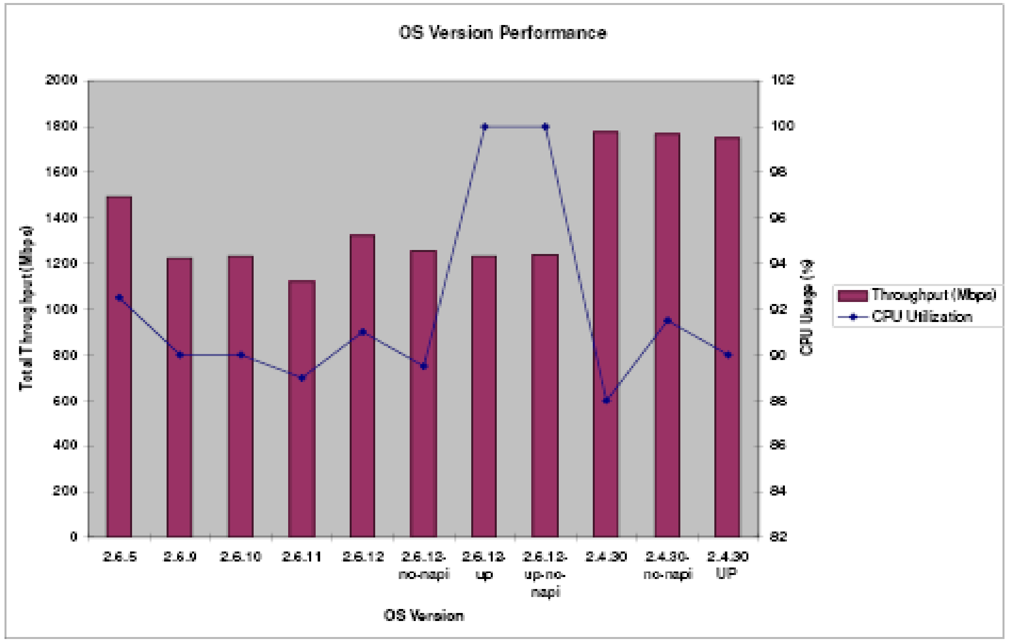


Figure 16: Packet loss as a function of Interrupt rate on the Xeon, and Opteron based systems.

http://www-lphe.epfl.ch/publications/2005/lphe-2005-006.pdf

- Xeon processors should use NAPI mode driver

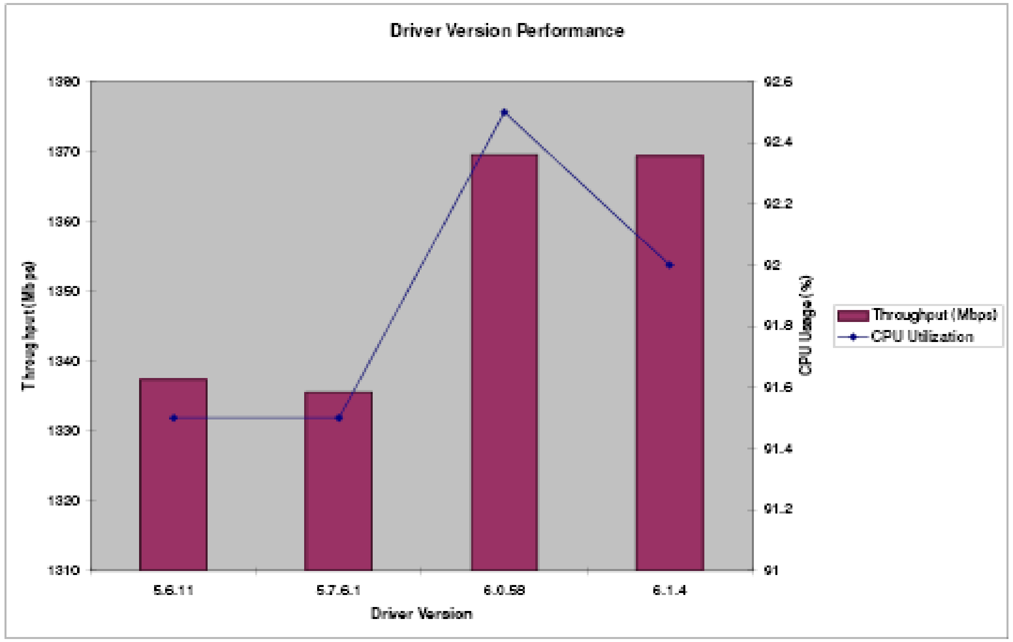
e1000 Driver



- network performance vs operating system version
- we are currently using 2.6.13
- note 2.4 better than 2.6

http://www.linuxsymposium.org/2005/linuxsymposium_procv2.pdf

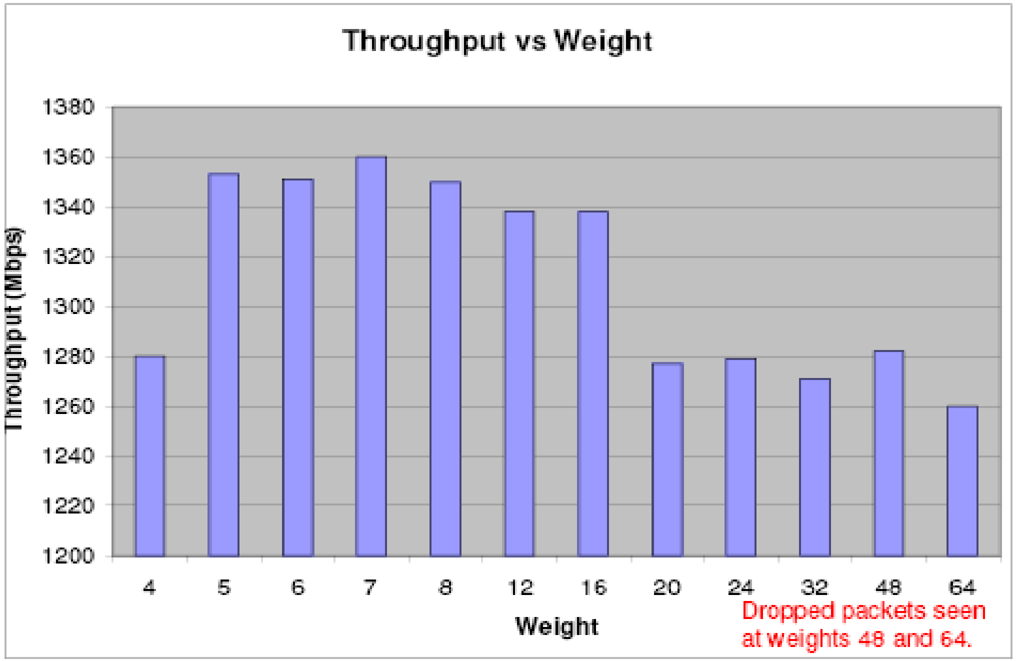
e1000 Driver



- network performance vs driver version
- we are currently using 6.0.53

http://www.linuxsymposium.org/2005/linuxsymposium_procv2.pdf

e1000 Driver



- network performance vs NAPI weight
- we are currently using default weight of 64
- lower weight requires driver recompile

http://www.linuxsymposium.org/2005/linuxsymposium_procv2.pdf

Linux Memory Manager

- tends to work against fast networking applications

- too aggressive in trying to find free memory

- most networking applications would probably run better with reduced memory management

- likely reason for lower performance in 2.6

Linux Memory Manager

1) swap

- Linux has been described as “swap happy”

- for a networking application, running swap in low kernel memory situations just adds to the cpu/disk load, which increases the packet queue backlog and causes the system to run out of memory sooner

- if swap is successful in freeing any low memory (not likely), it will be too late to avoid dropping packets...so might as well drop packets immediately and give the processing time to the application that is emptying RX buffers

- with sufficient memory and dedicated applications, high mem swap is probably unnecessary

- *vm.swappiness* doesn’t seem to have much (any?) effect on lowmem

- increasing *vm.min_free_kbytes* is a frequently suggested solution for memory allocation failures (this may or may not be helpful, since it causes swap to run earlier)

Swap

```
# vmstat 1
procs -----------memory----------  ---swap--   -----io---- --system-- ----cpu----
r  b   swpd   free   buff  cache   si     so    bi     bo    in    cs us sy id wa
3  1    128 387248    836 1694104   0      0  9452      0 13122 22524 14 21 61  4
1  3    128 374104    832 1237808   0      0 11096      1 14122 21923 16 22 55  7
4  4    128 375468    832 789568    0      0 10348 163022 12410 20327 16 26 36 22
1  8    128 273168    832 602368    0      0  9372 173050 11207 14964 13 33 14 41
7  6    128 273292    172 541148    0      0  7504 180769  9501  9868  3 46  5 47
2  2    252 294232    156 450944    0    124  6184  80557  9853 18823  9 44 14 33
1  7   1296 273168    152 140768    0   1044 38804 118225  4146 13763 21 26 24 29
0  5 113076 274036    152 156372    0 111780 22552 116428  2501  5014  3 10 44  3
0  8 171552 285444    152 173992    0  58476 25204  58900  3763  5972  3  6 28 62
3  5 225256 273788    152 199236  128  53704 29596  54248  5156  7033  4  7  7 82
2  6 294288 275648    152 199504   32  69144  8312  69312  1878  4208  3  6  6 85
2  7 348992 273284    152 218624   16  54592 38124  54912  3311  5630  4  6  6 84
1  5 401308 274276    152 246188    0  52316 32944  52788  4497  6849  4  5 14 76
1  5 464512 292008    152 225416    0  63204 20700  63516  2989  5876  3  7  2 88
1  5 535648 281468    152 240044    0  71136 32728  71360  2891  5495  4  6 24 66
...
```

- any swap activity is bad for time critical operation

- check /proc/meminfo
  if *committed_AS* is greater than physical memory, there is potential for swap

Linux Memory Manager

2) disk caching

- high memory on our 8 GByte dcache machines is 7 GBytes.

- Linux uses all **free** high mem (~5 GBytes) for disk buffer cache (philosophy “unused memory is wasted memory”)

- this is not a memory issue, but the additional buffer cache increases the CPU load and has limited value

Linux Memory Manager

3) buddy system

- continuously combines zero-order blocks (single pages) into higher order blocks, so when a zero-order block is requested there may be none left

- likewise not a memory issue, but superfluous combining and splitting of blocks also adds to the CPU load and delays allocations

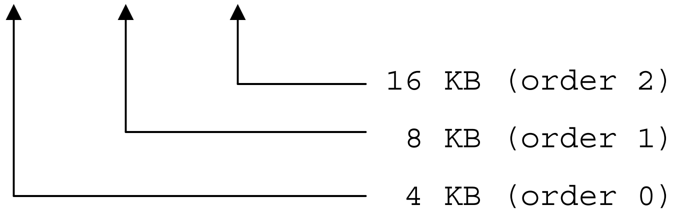
```
/proc/buddyinfo @ 1 second intervals

Node 0, zone   Normal    240    453   5072   4247    871     73      2     11      2      0      1

Node 0, zone   Normal     63    551   5075   4247    871     73      2     11      2      0      1

Node 0, zone   Normal     30    546   5078   4247    871     73      2     11      2      0      1

Node 0, zone   Normal      0    501   5078   4247    871     73      2     11      2      0      1
```



16 KB (order 2)

8 KB (order 1)

4 KB (order 0)

Linux Memory Manager

4) memory overcommit

- Linux allows overbooking of memory...the assumption is that not all allocated memory will actually be used

- the decision to allow an allocation request is based mainly on the size of the request...small requests are more likely to go unchecked

- all network buffer requests appear small

- if the system runs out of memory, it can invoke the “OOM-killer”

- not sure if overcommit distinguishes between shared memory and highmem machines (?) ... may not be an issue for highmem machine

> *“By default, Linux follows an optimistic memory allocation strategy. This means that when malloc() returns non-NULL there is no guarantee that the memory really is available. This is a really bad bug.”*
>
> http://www.novell.com/coolsolutions/qna/11511.html

Recommended for 32 bit machines

1) reduce *tcp_mem* by factor of 4

2) set *tcp_rmem* and *tcp_wmem* buffers to values appropriate for individual streams (not total bandwidth)

3) turn off memory overcommit (*vm.overcommit_memory* = 2)

(“paranoid” mode)

4) increase *vm.min_free_kbytes* to 65536

May be useful...but no obvious effect

1) increase *txqueuelen* to 1000

2) reduce *vm.swappiness*

To be Determined

1) update kernel and NIC driver

2) reduce “weight” of NAPI interface to 16

3) increase RX and TX ring buffer size to 4096

4) increase RX descriptors

64 bit Machines

Switching to a 64 bit machine (and 64 bit OS) resolves many of the network and memory related problems

- no out-of-memory with larger tcp buffers
- few, if any, RX overruns

but, there may be an increase in non-network issues

- possibly related to 64 bit kernel and 64 bit JVM

odd behavior (64 bit machines)

```
top - 12:23:40 up 31 days,  3:30,  2 users,  load average: 2.00, 2.03, 2.00
Tasks: 116 total,   1 running, 115 sleeping,   0 stopped,   0 zombie
Cpu(s): 50.2% us,  0.2% sy,  0.0% ni, 48.3% id,  1.4% wa,  0.0% hi,  0.0% si
Mem:   8111928k total,  7555632k used,   556296k free,     2644k buffers
Swap: 16779884k total,  2672556k used, 14107328k free,  1616196k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
13486 root      20   0 1138m 708m  11m S 99.9  8.9   8421:05 java
13003 root      20   0 1148m 716m  11m S 99.9  9.0   9862:50 java
16737 root      16   0  3948 1776 1508 S  0.3  0.0   0:01.34 sshd
    1 root      16   0  4748  552  456 S  0.0  0.0   0:01.95 init
    2 root      RT   0     0    0    0 S  0.0  0.0   0:04.90 migration/0
...
```

- processes in infinite loop

odd behavior (64 bit machines)

```
procs -----------memory----------    ---swap-- -----io---- --system--  ----cpu----
 r  b   swpd   free     buff  cache     si    so     bi     bo   in     cs   us sy id wa
53  0 1734720 675968   1652 4832860    0     0      0      0  360 1083196 10 90  0  0
52  0 1734720 675968   1652 4832860    0     0      0      0  342 1075831 11 89  0  0
52  0 1734720 675968   1652 4832860    0     0      0      0  332 1082861 11 89  0  0
52  0 1734720 675968   1652 4832860    0     0      0     32  360 1082769 11 89  0  0
52  0 1734720 676016   1652 4832860    0     0      0      0  341 1082945 11 89  0  0
53  0 1734720 676016   1652 4832860    0     0      0      0  345 1083795 11 89  0  0
52  0 1734720 676140   1652 4832860    0     0      0      0  337 1081053 11 89  0  0
52  0 1734720 676140   1652 4832860    0     0      0      0  341 1083057 11 89  0  0
52  0 1734720 676172   1652 4832860    0     0      0     22  351 1083682 11 89  0  0
52  0 1734720 676172   1652 4832860    0     0      0      0  441 1074681 12 88  0  0
52  0 1734720 676172   1652 4832860    0     0      0     16  355 1083077 12 88  0  0
52  0 1734720 676172   1652 4832860    0     0      0      0  363 1082475  9 91  0  0
52  0 1734720 676204   1652 4832860    0     0      0      0  370 1082768 11 89  0  0
52  0 1734720 676204   1652 4832860    0     0      0     12  338 1082964 11 89  0  0
52  0 1734720 676204   1652 4832860    0     0      0      0  372 1063590 10 90  0  0
52  0 1734720 676204   1652 4832860    0     0      0      0  343 1083154 11 89  0  0
52  0 1734720 676344   1652 4832860    0     0      0      0  379 1074343 12 88  0  0
52  0 1734720 676344   1652 4832860    0     0      0      0  326 1083535 13 87  0  0
52  0 1734720 676344   1652 4832860    0     0      0      0  351 1083205 11 89  0  0
52  0 1734720 676344   1652 4832860    0     0      0      0  435 1075193 10 90  0  0
52  0 1734720 676352   1652 4832860    0     0      0      0  332 1082909  9 91  0  0
```

- context switching storm (1 MHz)

Conclusions

with anecdotal comments from Google

- 32 bit Linux has insufficient kernel space for large networking applications
  - switch to a 64 bit machine if possible
  - limit maximum TCP buffer sizes
  - limit number of connections per machine
  - dedicated machines for local and remote traffic
  - try “4G/4G” kernel (not mainline, possible incompatibilities, slower)

> “Oh, the answer is very simple: it's not going to happen. EVER. You need more than 32 bits of address space to handle that kind of memory. This is not something I'm going to discuss further...This is not negotiable.”
>
> Linus Torvalds - 1999 (on running 32 bit Linux with more than 2GB of memory)

Conclusions

- 2.6 Linux kernels are less network-friendly

> “Using the 2.6 kernel [vs 2.4] on embedded systems implicates the following disadvantages:
> ...context switches up to **96%** slower, local communication latencies up to **80%** slower, file system latencies up to **76%** slower, local communication bandwidth **less than 50%** in some cases...”
>
> http://www.denx.de/wiki/Know/Linux24vs26

> “...found through his studies that kernel 2.4...outperformed kernel 2.6 in every test and under every configuration in terms of throughput and thus concluded that kernel 2.6 still has room for improvement.”
>
> http://os.newsforge.com/article.pl?sid=05/07/22/1054216&tid=2&tid=18

> “Is there any chance Linus will freeze 2.6 and make the current development tree 2.7? It seems like ever since around 2.6.8 things have been getting progressively worse (page allocation failures, ...”
>
> http://www.gatago.com/linux/kernel/15485908.html

> “FreeBSD 4.9: Drops no packets at 900K pps
> Linux 2.4.24: Starts dropping packets at 350K pps
> Linux 2.6.12: Starts dropping packets at 100K pps”
>
> http://www.gatago.com/linux/kernel/14693564.html

Conclusions

- Linux network RX has low priority...packet drops are “expected”
  - limit TX and other activity during RX
  - modify driver to improve RX priority
  - don’t worry (dropped packets will be resent)

> “... if the RX path gets very busy packets will end up being dropped ... by virtue of DMA rings being filled up ... and that is an acceptable compromise.”
>
> http://www.spinics.net/lists/netdev/msg10601.html

> “The issue is that RX is absolute, as you cannot "decide" to delay or selectively drop since you don't know what’s coming. Better to have some latency than dropped packets. But if you don't dedicate to RX, then you have an unknown amount of cpu resources doing "other stuff". The capacity issues always first manifest themselves as RX overruns, and they always happen a lot sooner on MP machines than UP machines. The LINUX camp made the mistake of not making RX important enough, and now their 2.6 kernels drop packets all over the ranch. But audio is nice and smooth...”
>
> http://leaf.dragonflybsd.org/mailarchive/users/2005-12/msg00007.html

Conclusions

- Linux memory management is not optimized for network applications

  - limit swap
  - reduce(!) RAM (increases available low mem, reduces unnecessary disk caching)

> “...there's some latency involved... if the CPU is stuck in an interrupt handler refilling a huge network RX ring, then waking kswapd won't do anything and you will run out of memory.”
>
> http://oss.sgi.com/archives/xfs/2004-12/msg00008.html

> “The bug density in Linux and its programs remains an embarrassment ...
> As bug nests go, Linux memory/swap management probably remains its greatest core problem...”
>
> http://lwn.net/Articles/140257/

The Fermilab wide area networking group is committed to direct support of CMS, but will also work with and assist any LHC Tier2 site as time permits.

Please contact us with any interesting network problems…at worst we can tell you that your problem is novel, at best we may be able to show that it's been solved (if you are a network researcher, reverse "worst" and "best").

Matt Crawford (crawdad@fnal.gov)
Wenji Wu (wenji@fnal.gov)
Mark Bowden (bowden@fnal.gov)
et al