Network tuning

For high bandwidth connections (1-10 Gbits/s) the default settings might not be sufficient.

MTU

The most important parameter to tune is the ethernet MTU (Maximum Transfer Unit), by default usually 1500, but should be adjusted to 9000 for higher throughput. This should be done with a command like:

```
ifconfig eth1 mtu 9000
```

In my case with:

```
ifconfig sara10g mtu 9000
```

sysctl

The next thing that should be adjusted are kernel parameters. This is done though sysctl. `sysctl -a` will give you all available settings, the ones that interest us, are mostly those in `net.core.*`

First we need to calculate what buffer size we need. The optimal buffer size is twice the bandwidth*delay product of the link:

```
  buffer size = 2 * bandwidth * delay
```

The ping program can be used to get the delay. Since ping gives the round trip time (RTT), this formula can be used instead of the previous one:

```
  buffer size = bandwidth * RTT.
```

For our connection to SARA this then becomes:

```
=-- bee22.grid.sara.nl ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 27034ms
rtt min/avg/max/mdev = 3.379/3.770/7.313/1.071 ms
```

```
buffer size = 10 Gbit/s * 0.00377 / 8 = 4712500 bytes
```

Current values are:

```
net.core.wmem_max = 131071
net.core.rmem_max = 131071
net.core.wmem_default = 124928
```

```
net.core.rmem_default = 124928
```

Let's adjust those to a value that allows for a bit more then the average measured by *ping*.

```
sysctl -w net.core.wmem_max=10000000
sysctl -w net.core.rmem_max=10000000
sysctl -w net.core.wmem_default=4712500
sysctl -w net.core.rmem_default=4712500
sysctl -w net.ipv4.tcp_mem='1542240      4712500   10000000'
sysctl -w net.ipv4.tcp_wmem='4096        4712500   10000000'
sysctl -w net.ipv4.tcp_rmem='4096        4712500   10000000'
```

We should also check and if needed adjust these values, otherwise it might remember slow settings for up to 10 minutes:

```
 # don't cache ssthresh from previous connection
 net.ipv4.tcp_no_metrics_save = 1
 net.ipv4.tcp_moderate_rcvbuf = 1
```

```
 sysctl -w net.ipv4.tcp_no_metrics_save=1
 sysctl -w net.ipv4.tcp_moderate_rcvbuf=1
```

And then these is this:

```
 # for 10 GigE, use this
 # net.core.netdev_max_backlog = 30000
```

```
 sysctl -w net.core.netdev_max_backlog=30000
```

And of course we need to do

```
sysctl -w net.ipv4.route.flush=1
```

To save the changes.

The same thing of course has to be done on the other side.

Results

Initial result:

```
lexar002 12:33-137> iperf -c bee22.grid.sara.nl -p 24000 -i 1
------------------------------------------------------------
Client connecting to bee22.grid.sara.nl, TCP port 24000
TCP window size: 4.49 MByte (default)
------------------------------------------------------------
[  3] local 145.100.118.2 port 37094 connected with 145.100.32.53 port 24000
```

```
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  32.2 MBytes    270 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  1.0- 2.0 sec  29.2 MBytes    245 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  2.0- 3.0 sec  29.2 MBytes    245 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  3.0- 4.0 sec  27.9 MBytes    234 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  4.0- 5.0 sec  29.3 MBytes    246 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  5.0- 6.0 sec  29.2 MBytes    245 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  6.0- 7.0 sec  27.7 MBytes    233 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  7.0- 8.0 sec  29.3 MBytes    246 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  8.0- 9.0 sec  26.2 MBytes    220 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  9.0-10.0 sec  26.4 MBytes    221 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec    287 MBytes    240 Mbits/sec
```

After tweaking on the LOFAR side:

```
lexar002 12:33-138> iperf -c bee23.grid.sara.nl -p 24000 -i 1
------------------------------------------------------------
Client connecting to bee23.grid.sara.nl, TCP port 24000
TCP window size: 4.49 MByte (default)
------------------------------------------------------------
[  3] local 145.100.118.2 port 50444 connected with 145.100.32.54 port 24000
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  47.0 MBytes    394 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  1.0- 2.0 sec  45.8 MBytes    384 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  2.0- 3.0 sec  45.4 MBytes    380 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  3.0- 4.0 sec  45.7 MBytes    383 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  4.0- 5.0 sec  46.5 MBytes    390 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  5.0- 6.0 sec  48.3 MBytes    405 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  6.0- 7.0 sec  48.2 MBytes    404 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  7.0- 8.0 sec  48.4 MBytes    406 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  8.0- 9.0 sec  48.0 MBytes    402 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  9.0-10.0 sec  49.7 MBytes    417 Mbits/sec
```

```
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec    473 MBytes    396 Mbits/sec
```

And after having also tweaked the SARA side:

```
lexar002 12:36-140> iperf -c bee23.grid.sara.nl -p 24000 -i 1
------------------------------------------------------------
Client connecting to bee23.grid.sara.nl, TCP port 24000
TCP window size: 4.49 MByte (default)
------------------------------------------------------------
[  3] local 145.100.118.2 port 55028 connected with 145.100.32.54 port 24000
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec    624 MBytes  5.23 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  1.0- 2.0 sec    647 MBytes  5.43 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  2.0- 3.0 sec    646 MBytes  5.42 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  3.0- 4.0 sec    645 MBytes  5.41 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  4.0- 5.0 sec    646 MBytes  5.42 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  5.0- 6.0 sec    656 MBytes  5.51 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  6.0- 7.0 sec    656 MBytes  5.50 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  7.0- 8.0 sec    657 MBytes  5.51 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  8.0- 9.0 sec    654 MBytes  5.49 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  9.0-10.0 sec    658 MBytes  5.52 Gbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  6.34 GBytes  5.44 Gbits/sec
Elapsed: 0:10.23 - CPU: 1.896u+11.748s = 133.2%
```

Please note that this number of ~5.5 Gbit/s has not been reliably attained between LOFAR and SARA machines, probably due to other load on the SARA systems. But even somewhat lower values around 4 Gbit/s still should be faster than the disks in the machines.

TCP/Generic Segmentation offloading

Furthermore enabling TCP/Generic Segmentation offloading should make a difference when trying to reach even higher values. On the LOFAR machines enabling this, allows the speed to go up to 8.44 Gbit/s, but it seems that the SARA machines can't handle this:

This is without TSO/GSO:

```
ethtool -K sara10g  tso off
ethtool -K sara10g  gso off
```

This is with TSO/GSO on the LOFAR side:

```
ethtool -K sara10g  gso on
ethtool -K sara10g  tso on
```

If we look at the data in detail the difference is in the ramp-up to higher speeds at the start. With TSO/GSO the LOFAR machines try to start using it around 4 Gbit/s, which gives the wavy pattern at the start of the second image, while without TSO/GSO it just overshoots what the SARA side can handle, until the send buffer fills up, and then it throttles back down to a steady state immediately.

The SARA side has a kernel that is too old to enable this feature. (as far as I know).

Useful diagnostics

tcpdump

You can run tcpdump like this to create a file with information (or without the -w to write to stdout, usually with a very low -c value).

```
[root@lexar002:data]#  tcpdump -w Adriaan.dump -i sara10g  -c 100000 port
24000
tcpdump: listening on sara10g, link-type EN10MB (Ethernet), capture size 96
bytes
100000 packets captured
100000 packets received by filter
0 packets dropped by kernel
```

Then later you can browse the dump you made: Important things to watch for:

1. wscale this needs to be hight to allow the window on either side to scale with 64k * 2^wscale. It should be high enough to scale the window to a size that can contain the wmem/rmem buffers st with sysctl above. (so 4.7MB buffers need at least a wscale of 9 for a size of 8.096 MB).
2. ***sackOK***: means the effective SACK algorithm can be used to selectively acknowledge received packages, saving a lot of re-transmitting in case of dropped packages.
3. the **win** parameter shows the window either side uses, the sending side will stay at a low value (70 in the example), the receiving side will grow up until the maximum it supports. The larger this is the bigger the number of packages that can be “in flight”.

```
[root@lexar002:data]# tcpdump -n -r Adriaan.dump -v |more

reading from file Adriaan.dump, link-type EN10MB (Ethernet)

13:52:27.236099 IP (tos 0x0, ttl 64, id 29566, offset 0, flags [DF], proto
TCP (6), length 52) 145.100.118.2.44180 > 145.100.32.53.24000:
```

```
S, cksum 0x5bc1 (correct), 3376685929:3376685929(0) win 17920 <mss
8960,nop,nop,sackOK,nop,wscale 8>

13:52:27.239538 IP (tos 0x0, ttl 61, id 0, offset 0, flags [DF], proto TCP
(6), length 52) 145.100.32.53.24000 > 145.100.118.2.44180: S,

cksum 0x35a2 (correct), 2973398225:2973398225(0) ack 3376685930 win 17920
<mss 8960,nop,nop,sackOK,nop,wscale 10>

13:52:27.239555 IP (tos 0x0, ttl 64, id 29567, offset 0, flags [DF], proto
TCP (6), length 40) 145.100.118.2.44180 > 145.100.32.53.24000:., cksum 0xd97d
(correct), ack 1 win 70

13:52:27.239581 IP (tos 0x0, ttl 64, id 29568, offset 0, flags [DF], proto
TCP (6), length 64) 145.100.118.2.44180 > 145.100.32.53.24000: P, cksum
0xb932 (incorrect (→ 0x7f83), 1:25(24) ack 1 win 70

13:52:27.239642 IP (tos 0x0, ttl 64, id 29569, offset 0, flags [DF], proto
TCP (6), length 9000) 145.100.118.2.44180 > 145.100.32.53.2400 0:
.25:8985(8960) ack 1 win 70

13:52:27.243045 IP (tos 0x0, ttl 61, id 36441, offset 0, flags [DF], proto
TCP (6), length 40) 145.100.32.53.24000 > 145.100.118.2.44180:., cksum 0xd999
(correct), ack 25 win 18

13:52:27.243065 IP (tos 0x0, ttl 64, id 29570, offset 0, flags [DF], proto
TCP (6), length 9000) 145.100.118.2.44180 > 145.100.32.53.2400 0: .
8985:17945(8960) ack 1 win 70

13:52:27.243205 IP (tos 0x0, ttl 61, id 36443, offset 0, flags [DF], proto
TCP (6), length 40) 145.100.32.53.24000 > 145.100.118.2.44180: ., cksum
0xb688 (correct), ack 8985 win 35
```

tcpgrok

Paul Boven also has a nice tool called tcpgrok, that is able to parse tcpdumps and create for example the graphs above.

Manufacturer recommendations

myri_readme

Links

I didn't make all of this up myself. I've gathered most information from the links below, and advice by Paul Boven.

Links on how to improve network performance:

- http://wwwx.cs.unc.edu/~sparkst/howto/network_tuning.php
- http://www.psc.edu/networking/perf_tune.html
- http://www-didc.lbl.gov/TCP-tuning/
- http://fasterdata.es.net/TCP-tuning/linux.html
- http://www.myri.com/scs/READMES/README.myri10ge-linux