TCP performance optimization for 10 Gb/s LHCOPN connections

Tiziana.Ferrari@cnaf.infn.it

on behalf of
M. Bencivenni, T.Ferrari, D. De Girolamo, Stefano Zani (INFN CNAF)
Andreas Hirstius (CERN)

HEPiX Spring Meeting, Roma, Apr 3 2006

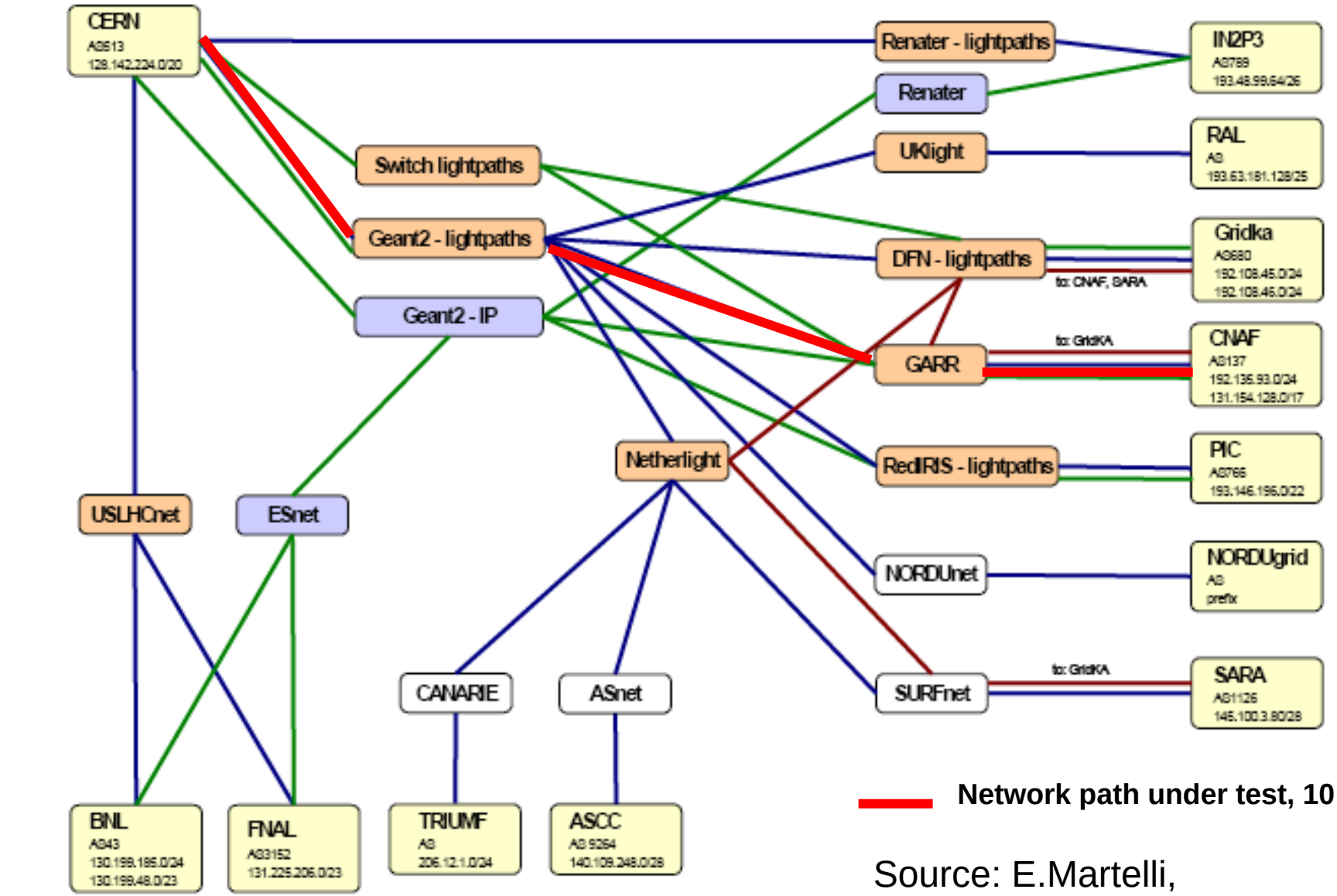
Objectives

- Test the 10 Gb/s path CERN – CNAF in preparation to the LHC Service Challenges
- Identify the list of sw and hw parameters to be tuned in order to optimize single-flow achievable throughput
- Compare different TCP stacks (Reno and BIC) and Linux kernel versions
- Compare different 10 GigaEthernet NICs

Performance Metrics and Parameters

- Performance metrics:
  - Achievable throughput (at the application level, memory-to-memory transfers, TCP and UDP)
  - Achievable throughput convergence time
  - CPU utilization
- Parameters:
  - TCP application parameters:
    - send/receive socket buffer
    - Application read/write block size
  - PCI-X bus: Max Memory Byte Read Count
  - Linux kernel version:
    - **2.4.21-32.0.1**.EL.cernsmp 2.6 vs **2.6.15.1** #2 SMP
  - Queuing: txqueue e backlog queue sizes

Testbed: WAN Network Layout



Network path under test, 10 Gb/s

Source: E.Martelli, LHCOPN meeting, Jan 2006

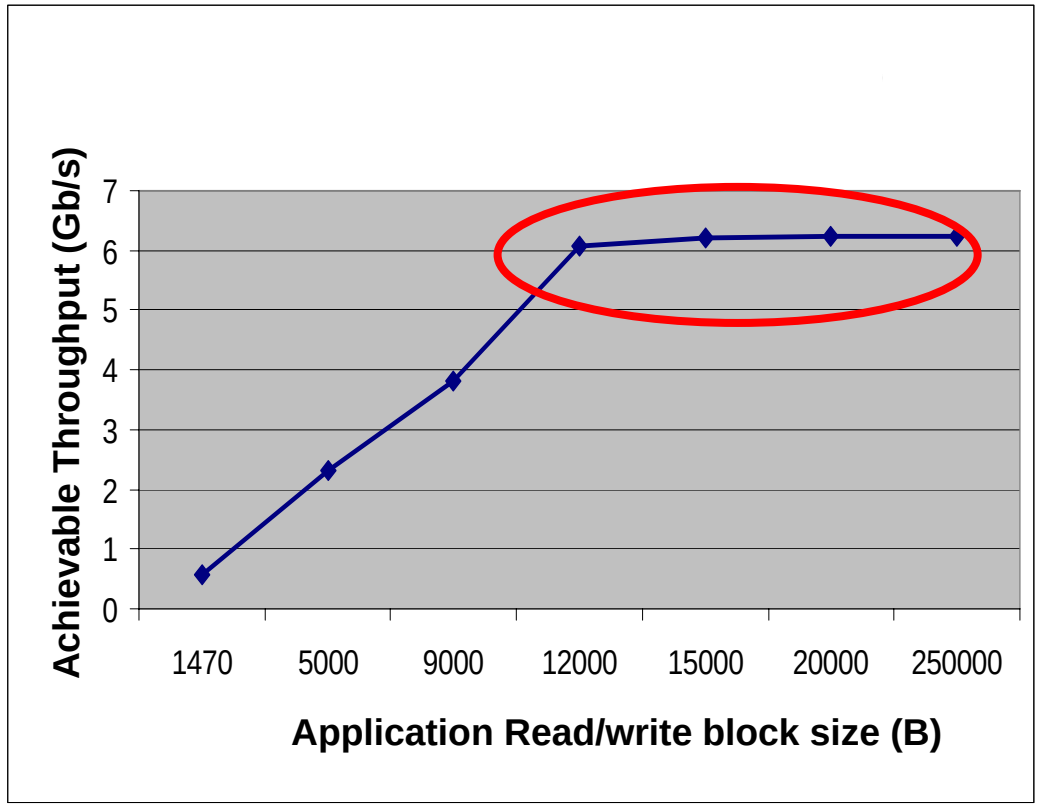
Testbed: Servers

- CERN (1 server):
  - 4 processors, GenuineIntel, IA-64, Itanium 2, cpu MHz: 1499.782942
  - NIC: 10 GigaEthernet s2io
  - LAN: connection to Force10 CE switch
- CNAF (2 servers):
  - 2 processors, AMD Opteron Processor 252 (Athlon), cpu MHz: 2589.457
  - NIC: Intel PRO/10GbE
  - LAN: connection BlackDiamond (10 GbE), directly connected to the GARR PoP in Bologna

- Overall max point-to-point performance measured:
  - 6.8 Gb/s CNAF → CERN, 5 TCP streams, TCP BIC, MTU = 9000 B

Tuning at the application-level: application read/write block size and send/receive socket buffer size

- Large application read/write block sizes give:
  - Higher CPU utilization and reduction of idle time
  - Increase of achievable throughput up to the total utilization of the overall number of CPU cycles available on the tx server
- Send/receive socket buffer:
  - Min size needed (according to our tests): 20 MB



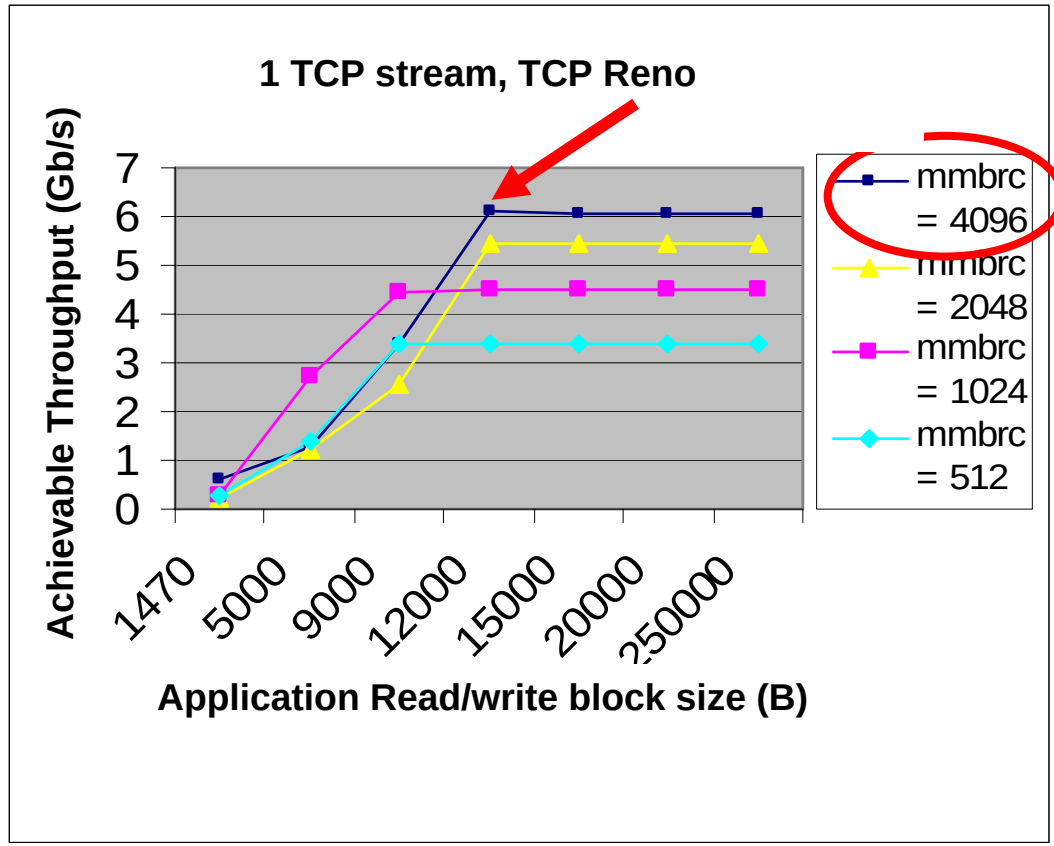
PCI-X Bus Configuration: Max Memory Byte Read Count (mmbrc)

- mmbrc – defines the max number of bytes that can be exchanged on the PCI-X bus during a single transaction from/to and I/O device to/from the server RAM
- Range:
  - 512,1024, 2048, 4096 B

→ A large mmbrc grants:

→ A better data/transmission overhead ratio on the PCI-X bus

→ More efficient CPU utilization

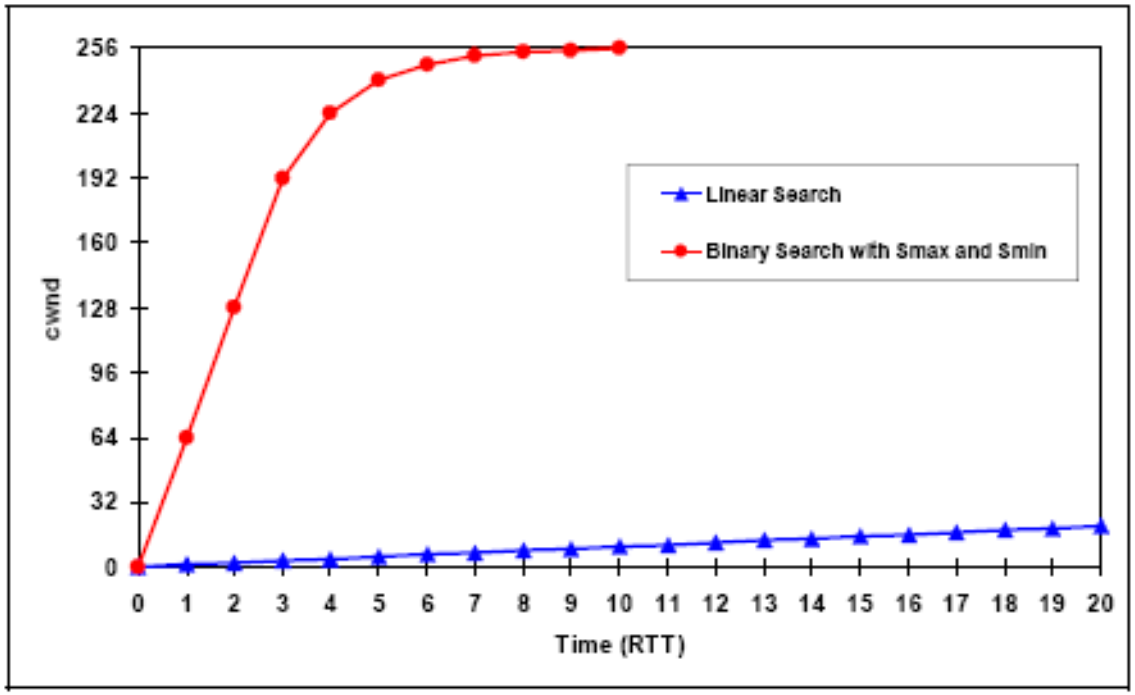


TCP BIC (Binary Increase Congestion control)

- Default protocol stack in the latest Linux kernel 2.6.x versions
- Slow start and Logarithmic increase of the congestion window parameter (cwnd) + additional increase
- Binary search algorithm when increasing/decreasing cwnd: min/target/max

– **In case of loss**:
  - max' = cwnd
  - min' = cwnd / 2
  - target = (max' + min') / 2

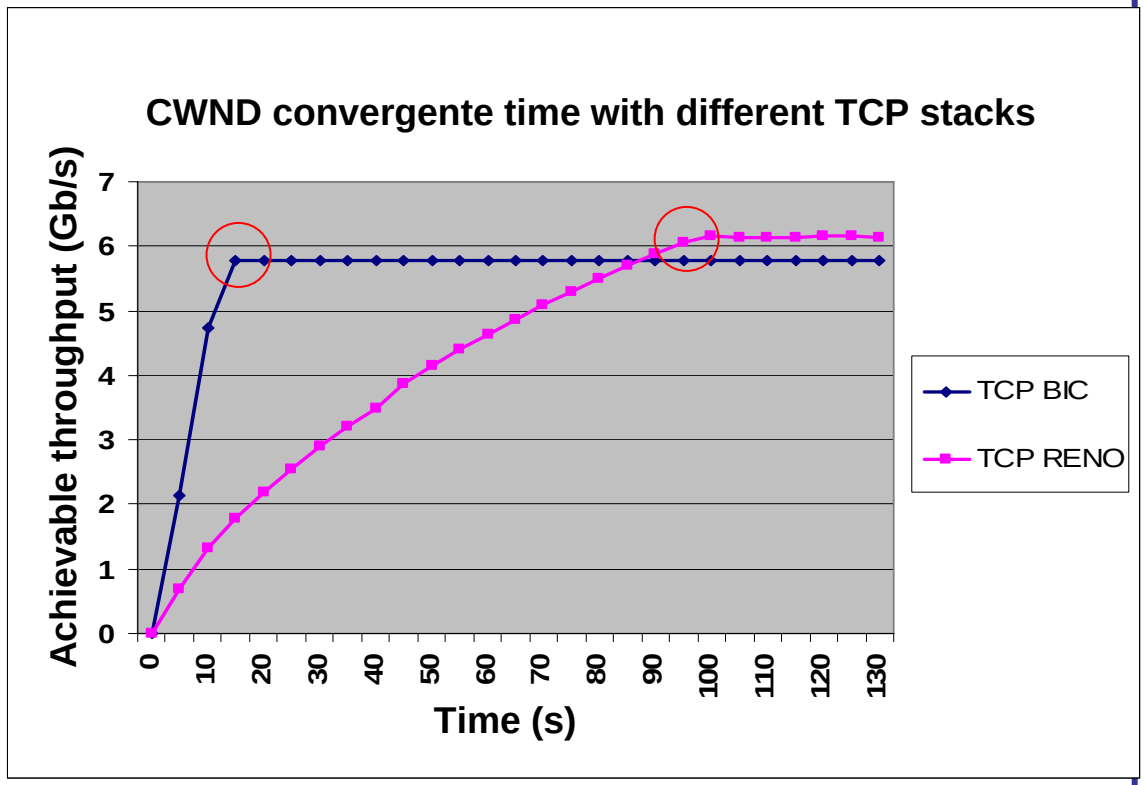
– **In case of growth (until max – min < S_min):**
  - max' = max
  - min' = cwnd
  - target = (max' + min') / 2
  - If target – cwnd > S_max
    – Cwnd' = cwnd + S_max (additive increase)



TCP Reno vs TCP BIC over WAN

- TCP Reno → default in kernel 2.4
- TCP BIC → default in kernel 2.6.x (different process scheduling algorithms in latest kernels as well)

- TCP BIC logarithmic increase gives:
  - Faster convergence to quilibrium
  - TCP Fairness: bounded for all window sizes
  - RTT Fairness: for large windows, throughput distribution between streams with different RTT similar to AIMD algorithms



Best performance measured over WAN

After tuning of the parameters mentioned before:

5 streams, TCP Reno → 6.2 Gb/s

5 streams, TCP BIC → 6.8 Gb/s

→ TCP BIC provides bettern performance in case of high-bandwidth long-distance network paths

→ Kernel 2.6 offers process scheduling algorithms that are more efficient → better performance in case of multiple streams, while on with a single stream TCP Reno can perfom better

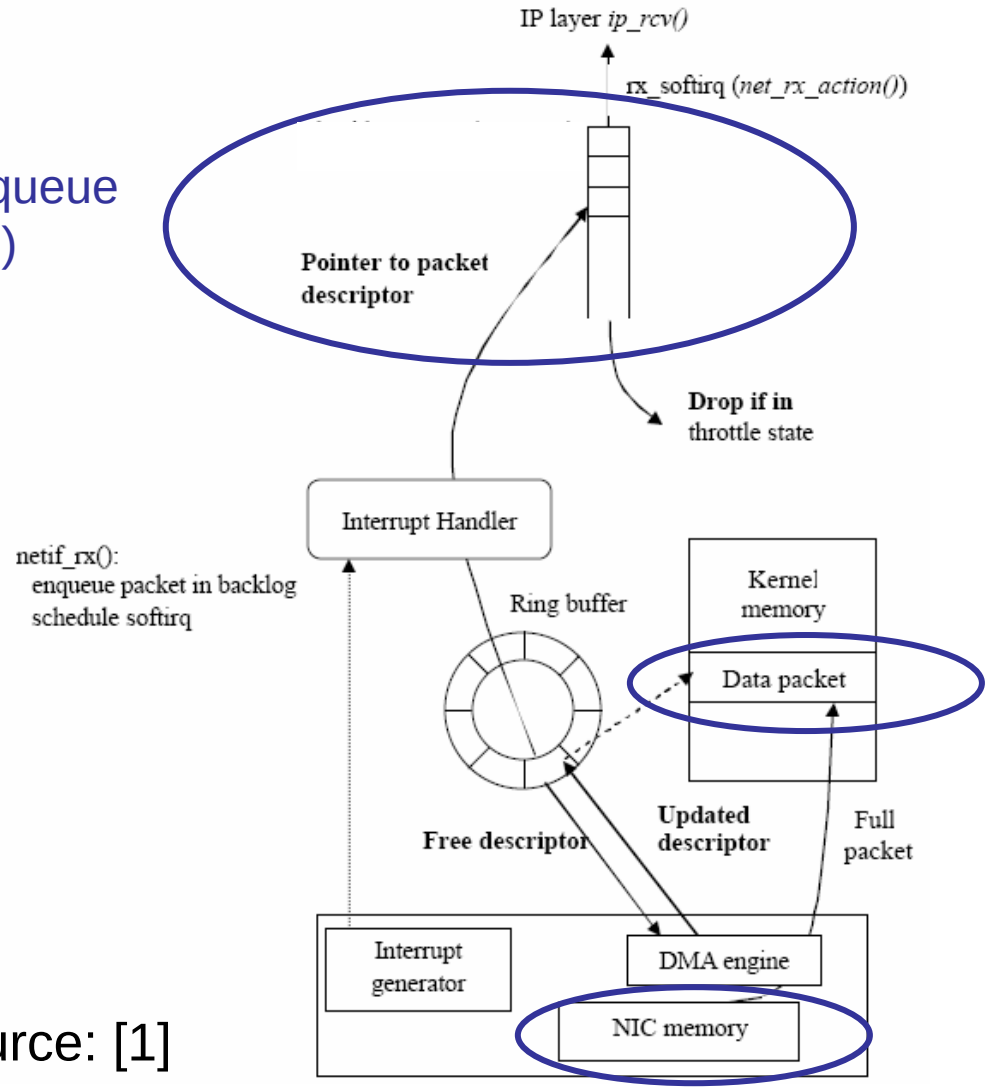
→ Kernel 2.6: more CPU demanding

Txqueue and backlog queue sizes

- Tuning of the size of the backlog (rx) and transmission queue (tx) in the kernel is necessary in order to:
  - avoid that packet descriptors are lost because of no available space in the txqueue (before reaching the tx NIC) and the backlog queue (i.e. at the very last stage of the transmission process), due to a high tx and/or rx rate
- The minimum queue size depends on the NIC speed and on the MTU in use:
  - jumbo frames (9000 B), 10 GigaEthernet →**1000 packets**
  - 1500 B, 1 GigaEthernet → **10000 packets**

Queuing in the Linux kernel: backlog queue during reception (*)

Backlog queue (per CPU)



Stage 3

Backlog queue

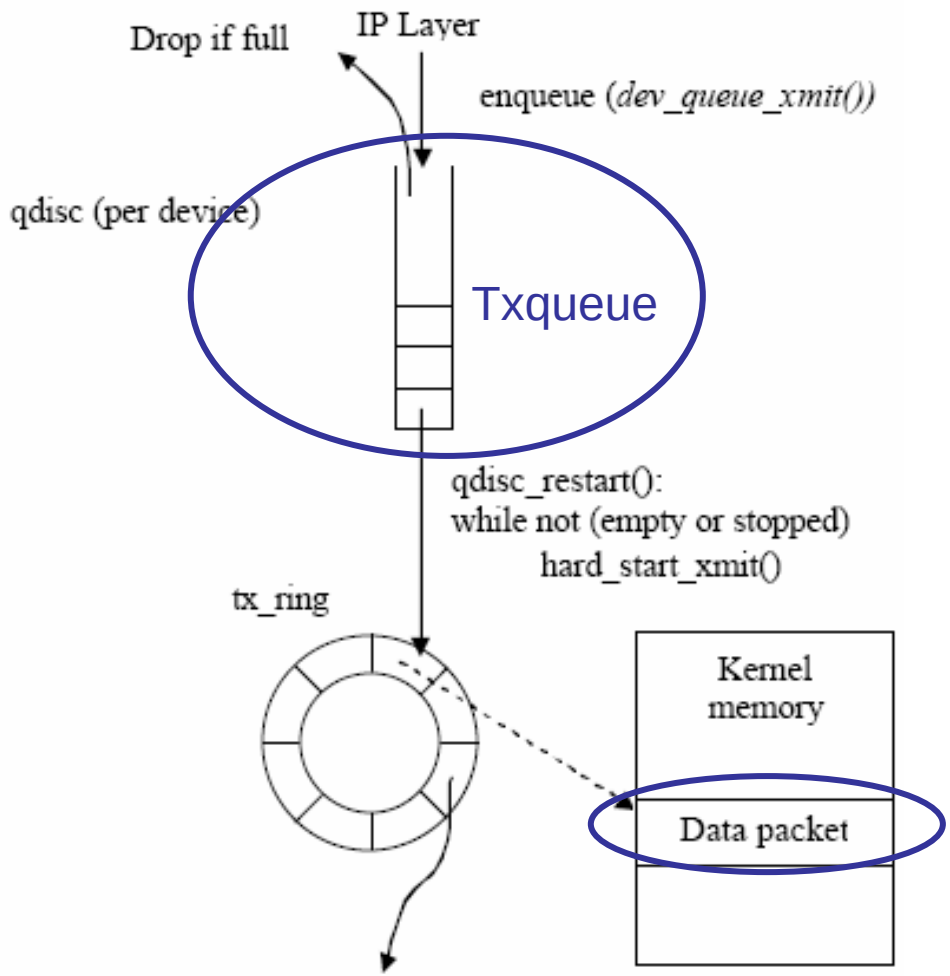
Stage 2

Receive socket Buffer in RAM

Stage 1

(*) source: [1]

Queuing in the Linux kernel: txqueue during transmission (*)



Stage 1

Txqueue queue

Stage 2

Send socket buffer

Stage 3

To DMA engine, NIC

(*) source: [1]

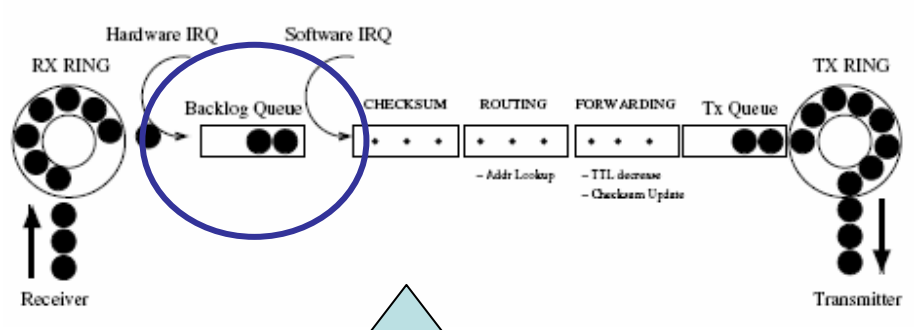
Interrupt coalescence

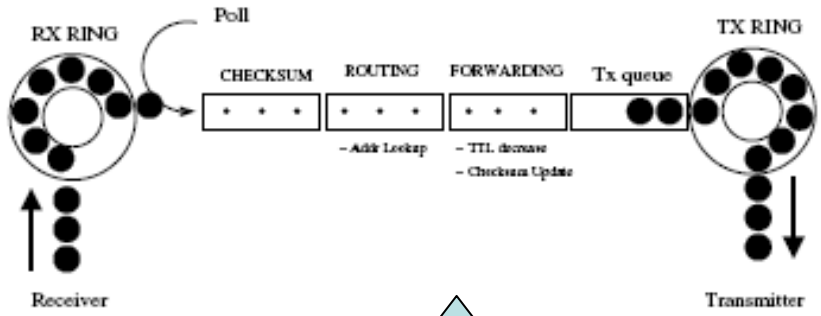
- Most modern NICs provide *interrupt moderation* or *interrupt coalescing* mechanisms to reduce the number of IRQs generated when receiving packets.
- In this case, interrupts are generated only after a number of packets has been transmitted/received, or after a timeout from the last IRQ generated has expired, whichever comes first.
- This allows to relieve the CPU from IRQ storms generated during high traffic load, improving the forwarding rate.

New API

- Receive livelock can be easily avoided by disabling IRQ generation on all NICs and letting the operating system decide when to poll the NIC hardware status register to determine whether new packets have been received.
- The NIC polling frequency is determined by the operating system and, as a consequence, a polling-driven stack may increase the packet forwarding latency under light traffic load.
- The network softIRQ is modified so as to run poll on all interfaces on the polling list in a round-robin fashion to enforce fairness. No more than a *budget B* of packets can be extracted from NIC reception rings in a single invocation of the network softIRQ, in order to limit the time the CPU spends for processing packets.
- Whenever poll extracts less than *Q* packets from a NIC reception ring, it reverts such NIC to interrupt mode by removing it from the polling list and re-enabling IRQ notification.

Comparison of the two approaches (^)





Interrupt-driven network stack

NAPI Network stack (poll)

(^) source: [2]

Additional 10Gb results (1/1)

- Memory to Memory:

  S2IO/Neterion Xframe NIC results (in collab. with DataTag)
  - WAN measurments in collaboration with DataTag
  - CERN → CalTech
  - Server:
    - CERN: Quad Itanium 2
    - CalTech: Dual Opteron
    - S2IO 10 Gb NICs
  - Transfer rate: 7.2 Gb/s
- Disk configuration:
  - one CPU servicing the interrupts for 1 RAID controller with 8 disks (3 CPUs, 3 controllers, 24 disks)
  - one CPU for the 10Gb NIC interrupts
  - HDs in JBOD mode, sw RAID0
  - (with new ARECA controllers: hw RAID 5, sw RAID 0 → same read performance and ~900MB/s write)

Additional 10Gb results (1/2)

- Disk-to-memory transfers (CERN → CalTech, 2004)
  - Server:
    - CERN: Quad Itanium2 with 24x SATA disks and 3x 3Ware 9500 controllers
      - local disk I/O: 1.1GB/s read; ~500MB/s write
    - Caltech: Quad Opteron 24x SATA disks and 3x SuperMicro controller
      - local disk I/O: ~450MB/s read; ~450MB/s write
  - single stream transfer rate: ~700MB/s
  - multi stream transfer rate: ~690 MB/s
- Disk-to-disk transfers
  - CERN → CalTech: ~375 MB/s (limited by receiving side)
  - CalTech → CERN: ~475 MB/s (limited by receiving side)

Conclusions

- Careful tuning at the application, kernel and PCI level needed
- Good understanding of kernel queuing mechanisms is important
- TCP BIC and TCP Reno in different kernel versions:
  - Comparison is difficult at 10 Gb/s as transmission is CPU-bound and kernels can differ significantly
  - TCP BIC: better convergence and overall achievable throughput in presence of multiple streams
  - NIC hardware architecture: can considerably affect the max achievable throughput

References

- [1] *Technical Report DataTAG-2004-1 FP5/IST DataTAG Project: A Map of the Networking Code in Linux Kernel 2.4.20*
- [2] *Open-Source PC-Based Software Routers: A Viable Approach to High-Performance Packet Switching, A.Bianco et alt.*