Debugging end-to-end performance in commodity operating system

Pavel Cimbál, CTU, xcimbal@quick.cz
Sven Ubik, CESNET, ubik@cesnet.cz
End-to-end performance

- Protective QoS -> proactive QoS
- Interaction of all computer system components (applications, operating system, network adapter, network)
- Decided to concentrate on E2E performance on Linux PCs
- Need to increase TCP windows, how much?
- What „autoconfiguration“ do we need?
How big are my TCP windows?

Linux 2.2

getsockopt();  
resulting windows  
rescaling and clamping  
SO_RCVBUF  
advertised win  
rcv_wnd is halved here!  
SO_SNDBUF  
sender’s win  
snd_cwnd, snd_cwnd_ssthresh

setsockopt();  
sysctl max clamping  
internal min values  
SO_RCVBUF  
rmem_max  
*2  
256 B  
SO_SNDBUF  
wmem_max  
*2  
2048 B

doubling!

sysctl defaults  
used without checks  
rmem_default  
wmem_default
How big are my TCP windows?
Linux 2.4

getsockopt();
- SO_RCVBUF
- SO_SNDBUF

resulting windows
- advertised win
- sender’s win

rescaling and clamping
- advWinscale, app_window
- window_clamp, rcv_ssthresh
- snd_cwnd, snd_cwnd_clamp
- snd_cwnd_ssthresh

setsockopt();
- SO_RCVBUF
- SO_SNDBUF

sysct1 max clamping
- rmem_max
- wmem_max

doubling!
- *2

internal min values
- MIN_PCVBUF
- MIN_SNDBUF

sysctl defaults
- tcp_rmem[1]
- tcp_wmem[1]

used without checks
Is the bigger the better?

\[ \text{rwnd} \leq \text{pipe capacity} \quad - \quad \text{bw} = \frac{\text{rwnd}}{\text{rtt}} \]

\[ \text{rwnd} > \text{pipe capacity} \quad - \quad \text{AIMD controlled, bw} \sim \left(\frac{\text{mss}}{\text{rtt}}\right)^*\frac{1}{\sqrt{p}} \]

- Linux is not RFC standard slow start + congestion avoidance, includes a lot of modifications
- Interaction with network adapter must be considered
- TCP cache must be considered
- Router queues must be considered
Testing environment

Cesnet (CZ) -> Uninett (NO)
12 hops, 1GE + OC-48, rtt=40 ms, wire pipe capacity<=5 MB
tcpdump on a mirrored port

pathload: 149 measurements
- 40 too low (50-70 Mb/s)
- 10 too high (1000 Mb/s)
- 99 realistic (750-850 Mb/s),
  but range sometimes
too wide (150 Mb/s)

iperf: 50 measurements
  1 MB  135 Mb/s
  2 MB  227 Mb/s
  4 MB  107 Mb/s
  8 MB  131 Mb/s
  16 MB 127 Mb/s
Gigabit interfaces and txqueuelen

ifconfig eth0 txqueuelen 1000
TCP cache

- initial ssthresh locked at 1.45 MB
- echo 1 > /proc/sys/net/ipv4/route/flush
What is the right “pipe capacity”

Gigabit routers have very big buffers

150 Mb/s overload buffered for 2 s => buffer=37.5 MB
Using “buffered pipe” is not good

- No increase in throughput over using „wire pipe“ in long term
- Self-clocking adjusts sender to bottleneck speed, but does not stop sender from accumulating data in queues
- Filled-up queues are sensitive to losses caused by cross-traffic
How to use not too much more than “wire pipe”

• Can sender control filling pipe by checking RTT?

• Can receiver better moderate its advertised window?
scp

Cesnet -> Uninett, 1.5 MB window, 10.4 Mb/s, 9% load CPU
scp, cont.

Patched scp, with increased CHAN_SES_WINDOW_DEFAULT:

set to 20, rwnd=1.5 MB  48 Mb/s, 45% CPU load
set to 40, rwnd=1.5 MB  88 Mb/s, 85% CPU load
PERT

- Performance Enhancement and Response Team
- TF-NGN initiative (TERENA, DANTE, so far 6 NRENs)
- Comparable to CERT
- A support structure for users to help solve performance issues when using applications over a computer network
  - hierarchical structure
  - accepting and resolving performance cases
  - knowledge dissemination
  - measurement and monitoring infrastructure
- Relations with other organizations
- Pilot project in 2003
Conclusion

• Configuration and interaction of existing components (application, TCP buffers, OS, network adapter, router buffers) may have greater influence on E2E performance than new congestion control algorithms

• Use „wire pipe“ rather than „buffered pipe“

• Autoconfiguration should not just save memory and set buffers „large enough“, but also „small enough“

http://www.cesnet.cz/english/project/qosip
http://staff.cesnet.cz/~ubik
Backup slides
scp, cont.

PROTOCOL 2.0

CHAN_SSES_WINDOW_DEFAULT (~128kB)

PKT 1  PKT 2  ...  PKT N2  PKT N1  PKT N

Performance impact:
BW < WIN_DEFAULT / RTT

CHAN_SSES_PACKET_DEFAULT (~32kB)

pk 1  pk 2  pk 3  ...  pk M3  pk M2  pk M1  pk M

Latency / interactivity impact:
TCP MSS  Non MSS Nagle delay packet pk M
Small WIN_DEFAULT != M*MSS & Nagle’s algorithm > additional RTT for whole PKT N

(+ Additional possible performance impact: Interferences TCP_WIN & WIN_DEFAULT )

TCP
SCAMPI

- Scaleable Monitoring Platform for the Internet
- 2 1/2 year project since April 2002
- To provide high-level monitoring API (MAPI) for monitoring applications
- To overcome gap between network speed and processor speed
- Will run on top of a commodity network adapter and a specialized network adapter
- Proposed initial monitoring applications:
  - packet capture
  - QoS monitoring
  - netflow statistics
  - intrusion and DoS attempts detection