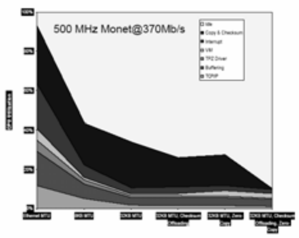
Internet Programming & Protocols
Lecture 26

TCP instrumentation

TCP overhead



www.cs.utk.edu/~dunigan/ipp/

TCP instrumentation

- Externally monitoring a TCP flow with tcpdump/tcptrace
- BSD systems all provide TCP_DEBUG socket option for trace
- TCP code keeps a lot of aggregate TCP statistics
  - netstat –s
  - Linux /proc/net/netstat
- Per flow statistics
  - Linux socket option TCP_INFO
  - Linux Web100

Counting events

- Linux TCP code is sprinkled with event counters
- Reported by netstat or in /proc

```
if (!(TCP_SKB_CB(skb)->sacked&(TCPCB_LOST|TCPCB_SACKED_ACKED))) {
                tcp_inc_pcount(&tp->lost_out, skb);
                TCP_SKB_CB(skb)->sacked |= TCPCB_LOST;
                flag |= FLAG_DATA_SACKED;

        NET_INC_STATS_BH(LINUX_MIB_TCPLOSTRETRANSMIT);
  }
…
if (tp->undo_marker && !tp->undo_retrans) {
              DBGUNDO(sk, tp, "D-SACK");
              tcp_undo_cwr(tp, 1);
              tp->undo_marker = 0;
              NET_INC_STATS_BH(LINUX_MIB_TCPDSACKUNDO);
       }
```


netstat -s

```
Tcp:
    9261 active connections openings
    174 passive connection openings
    1 failed connection attempts
    211 connection resets received
    3 connections established
    436246 segments received
    468946 segments send out
    2002 segments retransmited
    181 bad segments received.
    1137 resets sent
TcpExt:
    TCPPureAcks: 22705
    TCPHPAcks: 149324
    TCPRenoRecovery: 0
    TCPSackRecovery: 141
    TCPSACKReneging: 0
    TCPFACKReorder: 4
    TCPSACKReorder: 49
    TCPRenoReorder: 0
    TCPTSReorder: 3
    TCPFullUndo: 3
    TCPPartialUndo: 44
    TCPDSACKUndo: 0
    TCPLossUndo: 129
    TCPLoss: 54
    TCPLostRetransmit: 0
    TCPRenoFailures: 2
    TCPSackFailures: 69
    TCPLossFailures: 15
    TCPFastRetrans: 255
    TCPForwardRetrans: 72
    TCPSlowStartRetrans: 67
    TCPTimeouts: 752
    TCPRenoRecoveryFail: 0
    TCPSackRecoveryFail: 0
    TCPSchedulerFailed: 6
    TCPRcvCollapsed: 98
    TCPDSACKOldSent: 685
    TCPDSACKOfoSent: 3
    TCPDSACKRecv: 65
    TCPDSACKOfoRecv: 0
    TCPAbortOnSyn: 0
    TCPAbortOnData: 215
    TCPAbortOnClose: 54
    TCPAbortOnMemory: 0
    TCPAbortOnTimeout: 20
    TCPAbortOnLinger: 0
    TCPAbortFailed: 0
    TCPMemoryPressures: 0
```


TCP_INFO

- Linux setsockopt() (not portable)
- Add to your app. to report interesting TCP variables linux/tcp.h

```
struct tcp_info
{
        __u8    tcpi_state;
        __u8    tcpi_ca_state;
        __u8    tcpi_retransmits;
        __u8    tcpi_probes;
        __u8    tcpi_backoff;
        __u8    tcpi_options;
        __u32   tcpi_rcv_mss;
        __u32   tcpi_unacked;
        __u32   tcpi_sacked;
        __u32   tcpi_lost;
        __u32   tcpi_retrans;
        __u32   tcpi_fackets;
        __u32   tcpi_rtt;
        __u32   tcpi_rttvar;
        __u32   tcpi_snd_ssthresh;
        __u32   tcpi_snd_cwnd;
        __u32   tcpi_advmss;
        __u32   tcpi_reordering;
};
```


Web100

- **NSF funded** (PSC/NCAR/NCSA) *web100.org*
- **Modified Linux kernel**
  - instrumented kernel to read/set TCP variables for a specific flow
  - readable: RTT, counts (bytes, pkts, retransmits,dups), state (SACKs, windowscale, cwnd, ssthresh)
  - settable: buffer sizes
  - 100+ TCP variables (IETF MIB) (/proc/web100/)
- **GUI to display/modify a flow's TCP variables, real-time**
- **API for network-aware applications or tuning daemon**
- **Net100 extensions:**
  - additional tuning variables and algorithms
  - event notification
  - Java bandwidth tester http://*whisper.cs.utk.edu:7123*

Web100 patches to Linux kernel

From tcp_input.c

```
if (dst->reordering && tp->reordering != dst->reordering) {
                tp->sack_ok &= ~2;
                tp->reordering = dst->reordering;
                WEB100_VAR_SET(tp, RetranThresh, tp->reordering);
        }
```


Web100 GUI

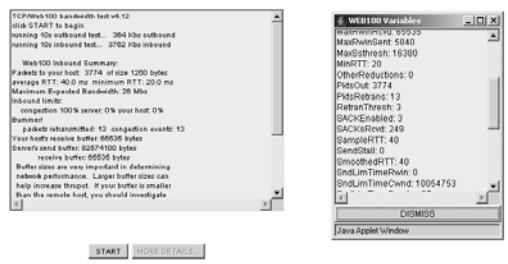
"Creating a window into the network"

Instrumented iperf

```
[whisper web100]% iperf100 -w 2m -c wisp.csm.ornl.gov
------------------------------------------------------------
Client connecting to wisp.csm.ornl.gov, TCP port 5001
TCP window size:  4.0 MByte (WARNING: requested  2.0 MByte)
------------------------------------------------------------
[  3] local 160.36.58.221 port 34229 connected with 160.91.212.75 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.2 sec   114 MBytes  93.3 Mbits/sec
CongestionSig          2 CurCwnd           311320
CurMSS              1448 CurSsthresh       179552
DupAcksIn              0 LimCwnd         94894680
LimRwin                0 MaxCwnd           359104
MaxRTT                40 MaxRwinRcvd      3144448
MaxRwinSent         5840 MaxSsthresh       179552
MinRTT                10 OtherReductio          0
PktsOut            82356 PktsRetrans            1
RetranThresh           3 SACKEnabled            3
SACKsRcvd              0 SampleRTT             30
SendStall              3 SmoothedRTT           30
SndLimTimeRwi          0 SndLimTimeCwn   10202454
SndLimTimeSen       2164 Timeouts               0
WinScaleRcvd           8 WinScaleSent           6
DupAcksOut             0 DataPktsIn             0
```

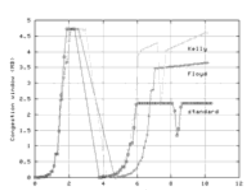

Java bandwidth tester

- Server running on Web100 host that can report back TCP flow variables to the java client



Web100 trace daemon

- Separate daemon to capture TCP variables for designated flows
  - Use Web100 API
  - Config file to specify path or port for tracing
  - C and python version
- Example, capture bandwidth, cwnd, and ssthresh every 0.1 seconds
- plot

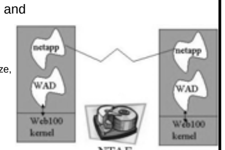


Net100 TCP Tuning Daemon

- **Work-around Daemon (WAD)** … a hack
  - tune unknowing sender/receiver at startup and/or during flow (path/target based tuning)
  - Web100 kernel extensions
    - pre-set *windowscale* to allow dynamic tuning
    - uses *netlink* to alert daemon of socket open/close (or poll)
    - besides existing Web100 buffer tuning, new tuning parameters and algorithms
    - knobs to disable Linux 2.4 caching, burst mgt., and sendstall
  - config file with static tuning data
    - mode specifies dynamic tuning (AIMD options, NTAF buffer size, concurrent streams)
  - daemon periodically polls NTAF for tuning data
  - can do out-of-kernel tuning (e.g., Floyd)
  - written in C (also Python version)

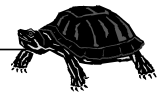
WAD config file

```
[bob]
 src_addr: 0.0.0.0
 src_port: 0
 dst_addr: 10.5.128.74
 dst_port: 0
 mode:  1
 sndbuf: 2000000
 rcvbuf: 100000
 wadai:  6
 wadmd:  0.3
 maxssth: 100
 divide:  1
 reorder:  9
 sendstall: 0
 delack: 0
 floyd: 1
 kellyai: 0
```



NTAF is out-of-band path prober

Things that slow us down … TCP



- SNDBUF limits
- RCVBUF limits
- NIC speed or bottleneck link speed
- Slow-start, delayed ACK, Nagle
- Packet loss and congestion
  - TCP recovery variants (Tahoe to Westwood)
- Packet reordering
- Slow ACK path (asymmetric net)
- TCP implementation
- Application "protocol"

- Recovery rate sensitive to RTT (speed of light) and MSS

Performance of a TCP implementation

- What makes one TCP implementation better than another?
  - In this case, we're not concerned about the flavors of TCP, but how the kernel TCP implementation interacts with the OS and the hardware
- Metrics
  - Network throughput and latency
  - CPU load
  - Number of active flows
- What are the bottlenecks in a TCP implementation?
- How can we improve the implementation
  - Software tricks
  - Hardware assist
- An efficient TCP implementation is
  - faster (throughput and latency)
  - Uses less memory
  - Provides more CPU cycles to the application
  - could extend life of battery-operated network devices

Overhead for TCP

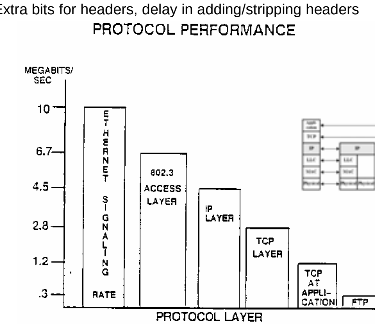
- Per transfer overhead
  - Application overhead of SEND or RECEIVE
  - OS overhead of handling system call(s)
    - Context switch
    - Allocating buffers

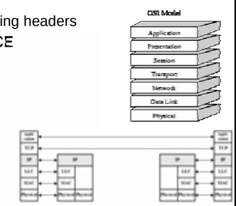
  \+ fewer calls helps (big write's)
- Per packet overhead
  - Transport protocol overhead for creating segments
  - Queuing to NIC
  - NIC transfer and NIC interrupts

  \+ bigger MTU helps (jumbo frames)
- Per byte overhead
  - Copying data
  - Checksums

Does layering affect performance?

•Extra bits for headers, delay in adding/stripping headers





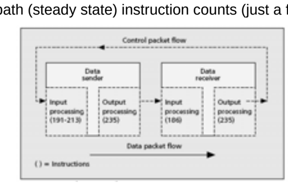
Not quite this bad, but you get the idea.

TCP software optimizations

- Faster CPUs help keep up with faster NICs
- What is the minimum number of cycles to process a TCP packet?
- *BSD and Linux have fast-path processing for "expected" packets
  - Van Jacobson claims TCP receive can be done in 30 instructions
- Linux combines memory copy with checksum
- Cache-aligned data structures and page-aligned buffers
- Linux has many caches to re-use common information
  - Memory buffers leave info pre-set for headers etc.
- Faster timer management
- More efficient SACK handling
  - Though only invoked when losses (or out of order packets), SACK info can be extensive for flows with big windows
- The limiting factor is usually memory copies!

'89 TCP overhead results

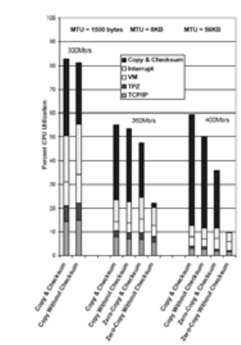
- Common path (steady state) instruction counts (just a few 100)



- What slows us down is memory accesses (checksum, copy)

Costs*	
Per byte:	
User-system copy	200 µs
TCP checksum	185 µs
Network-memory copy	386 µs
Per packet:	
Ethernet driver	100 µs
TCP + IP + ARP protocols	100 µs
Operating system overhead	240 µs

Duke TCP overhead analysis



How do we reduce the copy/checksum overhead?

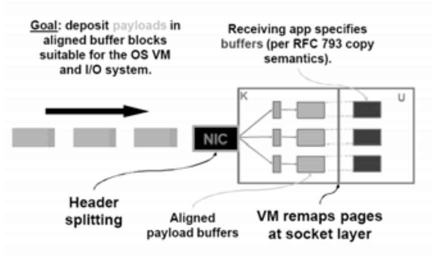
Zero-copy alternatives

- Option 1: page flipping
  - NIC places data in page-aligned memory
  - OS uses virtual memory hardware to "move" the data
- Option 2: scatter/gather API
  - NIC puts the data where ever
  - Application can get it from where ever
- Option 3: direct data placement (DDP)
  - Headers tell the NIC where to place the data

- Each solution involves the application, the OS, and the NIC

Ref: Chase@Duke

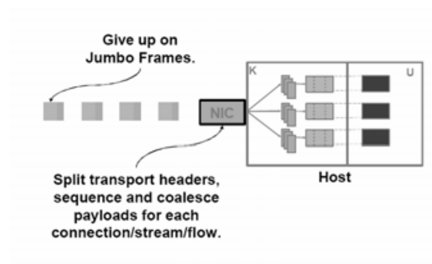
Page flipping



Need page-sized MTU's (4K ? 8K?)

Ref: Chase@Duke

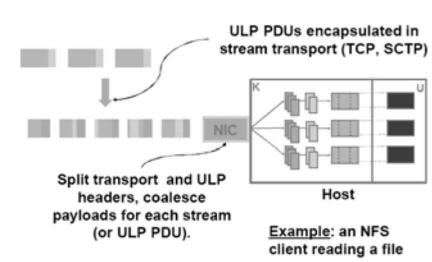
Page flipping with small MTUs



Ref: Chase@Duke

Page flipping with user level protocol (ULP)

- Application "messages" contain headers with placement info
- If NIC is doing the splitting, need really smart NIC (parse NFS header)



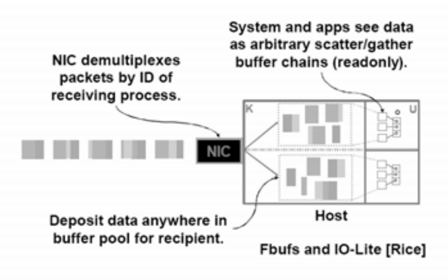
Ref: Chase@Duke

Page flipping Pros and Cons

- Pro: sometimes works.
  - Application buffers must match transport alignment.
- NIC must split headers and coalesce payloads to fill aligned buffer pages.
- NIC must recognize and separate ULP headers as well as transport headers.
- Page remap requires TLB shootdown for SMPs.
  - Cost/overhead scales with number of processors.

Ref: Chase@Duke

Option 2: scatter/gatter



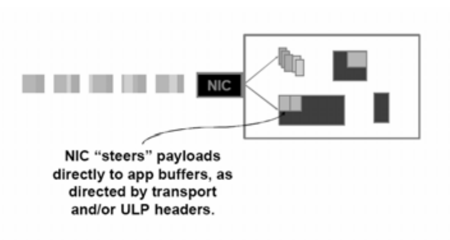
Ref: Chase@Duke

Scatter/gather Pros and Cons

- Pro: just might work.
- New APIs
- New applications
- New NICs
- New OS
- May not meet app alignment constraints.

Ref: Chase@Duke

Option 3: direct data placement (DDP)



Ref: Chase@Duke

DDP examples

- TCP Offload Engines (TOE) can steer payloads directly to preposted buffers.
  - Similar to page flipping ("pack" each flow into buffers)
  - Relies on preposting, doesn't work for ULPs
- ULP-specific NICs (e.g., iSCSI)
  - Proliferation of special-purpose NICs
  - Expensive for future ULPs
- RDMA on non-IP networks
  - VIA, Infiniband, ServerNet, etc.

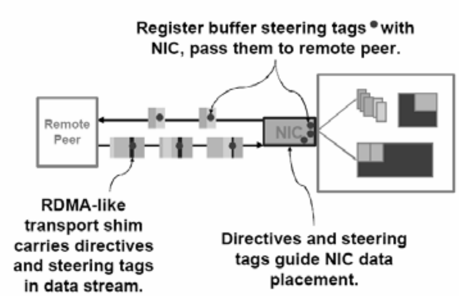
Ref: Chase@Duke

DDP Pros and Cons

- Effective: deposits payloads directly in designated receive buffers, without copying or flipping.
- General: works independent of MTU, page size, buffer alignment, presence of ULP headers, etc.
- Low-impact: if the NIC is "magic", DDP is compatible with existing apps, APIs, ULPs, and OS.
- Of course, there are no magic NICs…

Ref: Chase@Duke

Remote Direct Memory Access (RDMA)



Ref: Chase@Duke

The case for RDMA over IP

- RDMA-like functions offer a general solution for fast-path data movement.
  - New transport functions to enhance performance.
- Lots of experience with RDMA on non-IP interconnects.
  - RDMA is an accepted direct-access model.
  - Protocols and APIs already exist to use RDMA.
- Leverages IP infrastructure, generality, cost.
  - IP everywhere
- RDMA NICs are general across a range of apps and ULPs.

Ref: Chase@Duke

The case against

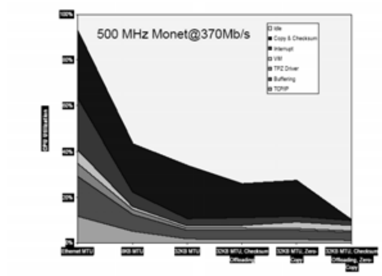
- Requires a standard protocol for direct data placement over IP transports.
  - Interoperability
  - Must leverage and conform to existing/future framework for security and management.
- Requires some extension of ULPs or apps to use it.
  - Low to moderate
- “No RDMA protocol exists that can solve X.”

Ref: Chase@Duke

Duke experiments (Chase)

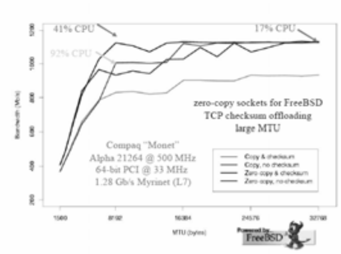
- Extend a public-domain Unix kernel with high-speed networking optimizations known in the literature.
  - FreeBSD 4.0...
  - ...extended for zero-copy sockets and checksum offloading.
  - Zero-copy TCP and NFS/UDP
  - Tigon GE and Trapeze/Myrinet
- Use *netperf* to benchmark TCP/IP on different configurations, using *iprobe* to profile host CPU activity.
  - 2 types of PCs and 2 types of Alphas
  - 32- and 64-bit PCI
  - vary MTUs from 1500 bytes up to 32KB
- Experiment with various combined optimizations, and study the numbers.

CPU overhead for TCP



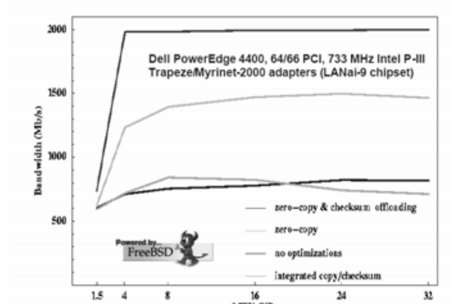
Ref: Chase@Duke

Duke optimizations



Ref: Chase@Duke

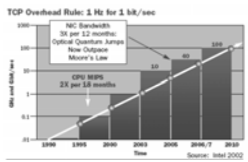
Netperf/TCP at 2 Gbs (Duke)



Ref: Chase@Duke

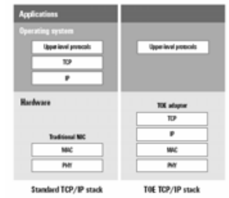
TCP in hardware

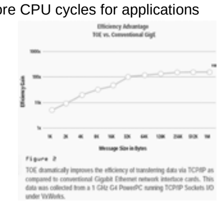
- Checksums on NICs
  - Saves that extra memory pass
  - Violates the layered architecture
  - Transport layer needs to know link layer will do (or has done) checksum
  - Available in many NIC's now – supported by Linux, Windows
- Interrupt coalescing on NICs
  - One interrupt for a several packets
  - Common for GigE and higher
  - "buffering" could increase latency
- TCP offload Engines (TOE)
  - Need 1Hz CPU for 1 bit/sec net
- Custom hardware transports (XTP)
  - SGI proposal in late 80s
  - Transport layer in silicon (NIC)



TCP offload engines

- Special NICs that do much of TCP in the NIC
- Does checksums
- For write's, can do segmentation
- Reduce memory copies with DMA
- Reduce interrupt load
- Suited to long flows with few errors
- More CPU cycles for applications





TCP/IP and low latency nets

- Can (should?)TCP/IP compete with custom protocols
  - Cluster computing
    - Myrinet vs 10GigE
    - VIA
  - Storage area networks (SANs)
    - iSCSI and fiber channel
  - Parallel computer interconnects
    - Infiniband, cross-bars, HiPPi, fat-trees
    - Big MTU's
    - High speed (20+ gbs)
    - Custom message passing (MPI) or distributed shared memory (SHMEM)
      - put/get semantics (zero-copy, RDMA-like)
      - Latency below 5 us

Next time …

- Review