TCP offload is a dumb idea whose time has come

Jeffrey Mogul
JeffMogul@acm.org
Large Scale Systems Group
HP Labs, Palo Alto
One-slide summary

What is TCP Offload?
• Moving IP and TCP processing to the Network Interface (NIC)

Why is it a dumb idea?
• Fundamental performance issues
• Practical deployment issues
• Poor justification (wrong applications)

Why has its time come?
• So we can offload higher-level protocols
TCP Offload Engines (TOEs)

User mode
Application

Kernel
Sockets
TCP
IP
NIC driver

Ethernet
Network IC

becomes

User mode
Application

Kernel
Sockets
NIC driver

TCP
IP
Ethernet
Network IC

TOE
Typical justifications for TCP offload

- Reduction of host CPU cycles for protocol header processing, checksumming
- Fewer CPU interrupts
- Fewer bytes copied over the memory bus
- Potential to offload expensive features such as encryption
Why TCP offload is dumb: Performance (part 1: technology issues)

- TCP/IP headers don’t take many CPU cycles
  - Cf. Jacobson’s “Header prediction” code
- Moore’s Law works against “smart” NICs
  - Complexity increases time-to-market
  - CPUs keep getting faster & benefit from large volumes
- TOEs impose complex interfaces
  - Protocol between TOE & CPU can be worse than TCP
  - Could require passing more context info
Why TCP offload is dumb: Performance (part 2: management)

- Suboptimal buffer management
  - Very hard to avoid buffer copy (esp. on receive)
  - But buffer copies are the real performance issue

- Connection management overhead
  - For short connections, overwhelms any savings

- Ditto for event management overhead

- Resource management
  - Virtual resources (e.g., ports) must be managed
  - Coordination with host OS adds overhead
Why TCP offload is dumb: Performance (part 3: alternatives)

• Much simpler NIC extensions can be effective
• For example:
  – TCP checksum offload (can avoid CPU data-touching)
  – Afterburner (Dalton et al. 1995) for single-copy TCP
• Sometimes the OS implementation just sucks
Why TCP offload is dumb: Deployment issues (part 1: using TOEs)

- Scaling is harder for TOEs than for host CPUs
  - Large systems have large buffer pools, routing tables
  - TOEs reduce allocation flexibility

- Programmable NICs: more vulnerable to hackers?
  - Programmability is always a potential hole
  - But: many modern NICs are already programmable

- More system management interfaces to deal with
  - Or, seams showing between “integrated” interfaces
  - TOEs may lack state visibility available in host OS
Why TCP offload is dumb: Deployment issues (part 2: maintenance)

- TOEs likely to have more bugs than simple NICs
  - IP/TCP implementations often need fixes/upgrades
  - Doubles the number of code bases to manage
- More code bases means QA is harder, slower
- Problem isolation becomes harder
  - Finger-pointing between OS and TOE vendors
- Exposes customers to risk of TOE vendor failure
  - Lack of support worse for TOEs than for simple NICs
Why TCP offload is dumb: Mismatched applications

- Traditional applications for TCP:
  - WAN applications (email, FTP, Web, IM, USENET)
  - Short connections, and many of them at once
  - IP/TCP packet processing costs do not dominate
- Problem areas for TCP offload:
  - High network delay (obviates low-delay NIC tricks)
  - Lots of connections, lots of connection management
  - Low ratio of packet processing costs to other costs
- So: traditional TCP apps don’t need offload
Insights

• Sweet spot for TCP offload might be apps with:
  – Very high bandwidth
  – Relatively low end-to-end latency network paths
  – Long connection durations
  – Relatively few connections

• Typical examples of these might be:
  – Storage-server access
  – Graphics
  – Cluster interconnect
Network-I/O convergence?

• Promising aspects:
  – Replace special-purpose hw w/ cheap commodity parts
    • 1Gbit or 10Gbit Ethernet
  – Only one fabric to provision, connect, and manage
    • More scalable and interoperable

• Challenges:
  – Data copy costs dominate (busses are too slow)
  – Zero-copy and single-copy seem too hard to adopt
What’s so hard about zero-copy TCP?

• On receive: headers interspersed with data
  – Page-remapping tricks often fail to help
• On transmit: buffer ownership issues
  – Application can’t touch buffer before it’s ACKed
• Some techniques force new APIs on applications
• Changing commercial OS stacks is a nightmare
• Lots of people have tried to make this work
  – Has anyone really succeeded?
Side-stepping the problems: RDMA

• Remote Direct Memory Access
• New protocol layer *between* transport and apps
  – App @ host X registers buffer regions w/ local RDMA
  – Region IDs are sent (somehow) to App @ host Y
  – App @ Y reads/writes data buffers in X’s memory
  – RDMA layer knows what is data, what is header
• Intended for hardware implementation (RNIC)
  – Allowing zero-copy for many (not all) applications
Aha!: RDMA requires transport offload

- Must offload transport in order to offload RDMA
  - Transport could be (e.g.) TCP+MPA shim, or SCTP
- RDMA well matched to storage access
  - Fits easily below NFSv4, DAFS, iSCSI
- The right characteristics for transport offload
  - Data-center networks, long connections
- Simplifies many problems w/generic TCP offload
  - Explicit protocol-visible separation of data & headers
RDMA NICs (RNICs)
Why should we believe that this will fly?

• NIC vendors want to ship RNICs in volume
  – They need to raise the price point over current NICs
  – RDMA allows generic solution (vs. iSCSI NICs)
  – InfiniBand isn’t a high-volume market (yet?)

• System, OS, and storage vendors want it
  – Cheaper hardware, simpler data centers
  – Willing to deal with a new protocol layer

• Upper-Level Protocols (ULPs) ready & waiting(?)
  – NFSv4, DAFS, iSCSI extensions for RDMA (iSER)
What could go wrong?

• Many problems of TOEs still apply
  – E.g., multiple code bases, resource allocation
• So far, the benefits have been “elusive”
  – May need well-integrated NIC + 10 Gbit LANs
• Extension to user-level networking is tricky
  – New API; transmit buffer-pinning still a problem
• Standardization not quite done
  – SCTP vs. TCP; MPA concerns; security questions
Summary

• Generic TCP offload seems like a bad idea
  – “solution in search of a problem”
  – Cure is usually worse than the disease
• RDMA offload justifies transport offload
  – OK, jury is still out on that
• New networking model might change OS APIs
  – Are read() and write() really the only way to go?
• RDMA requires “OS thinking” in new places
Odds and ends

- **SCTP**: an alternative to TCP
  - Doesn’t require MPA shim to get message boundaries
  - Not ready to ship in silicon, yet

- **RDMA or DDP (Direct Data Placement)**?
  - DDP: remote-write only; should be simpler
  - Are remote reads & other RDMA verbs necessary?

- **Security**: not a simple issue
  - Implementations of a secure protocol may have bugs
  - Consequences of exploited bug: free access to memory