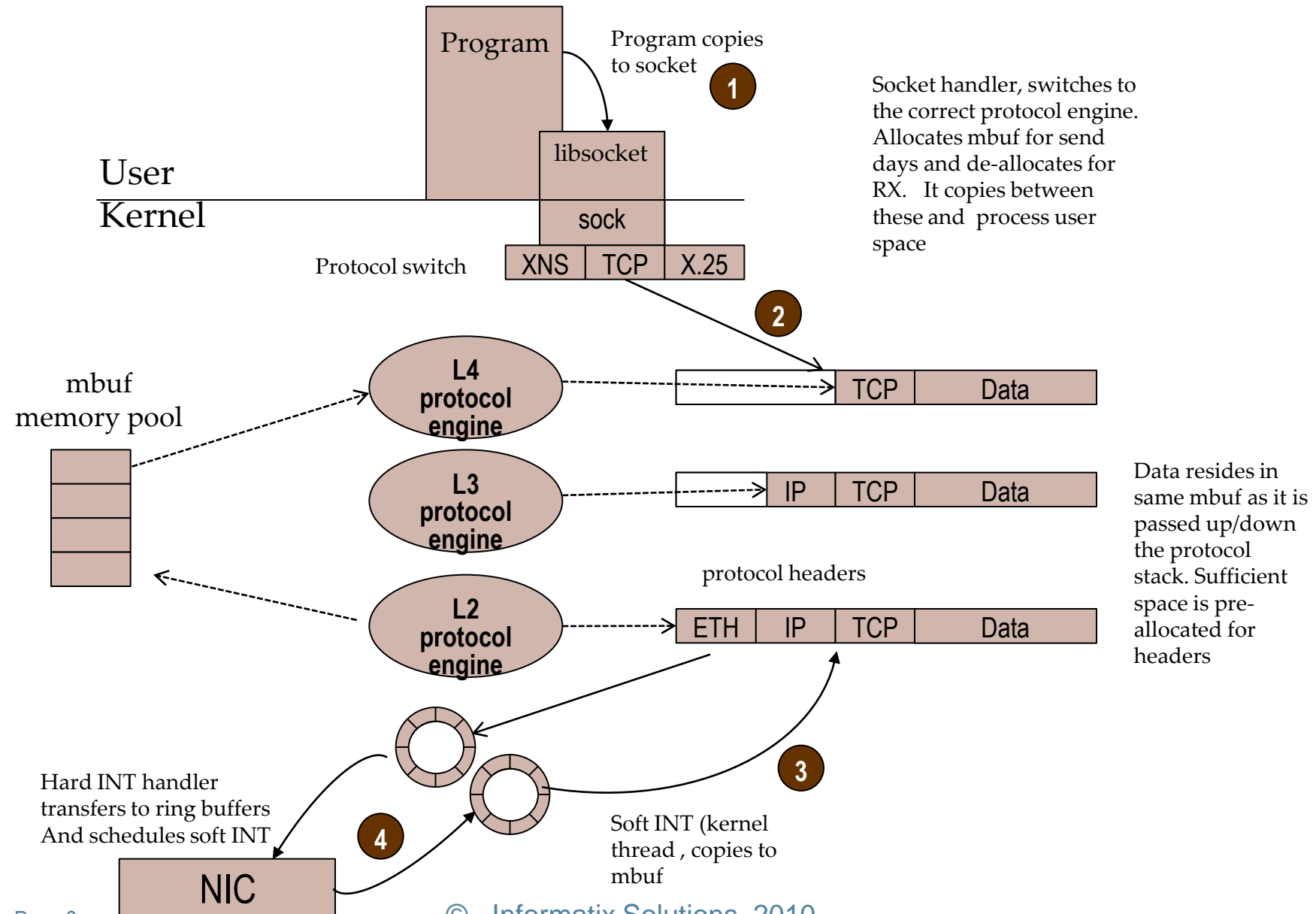
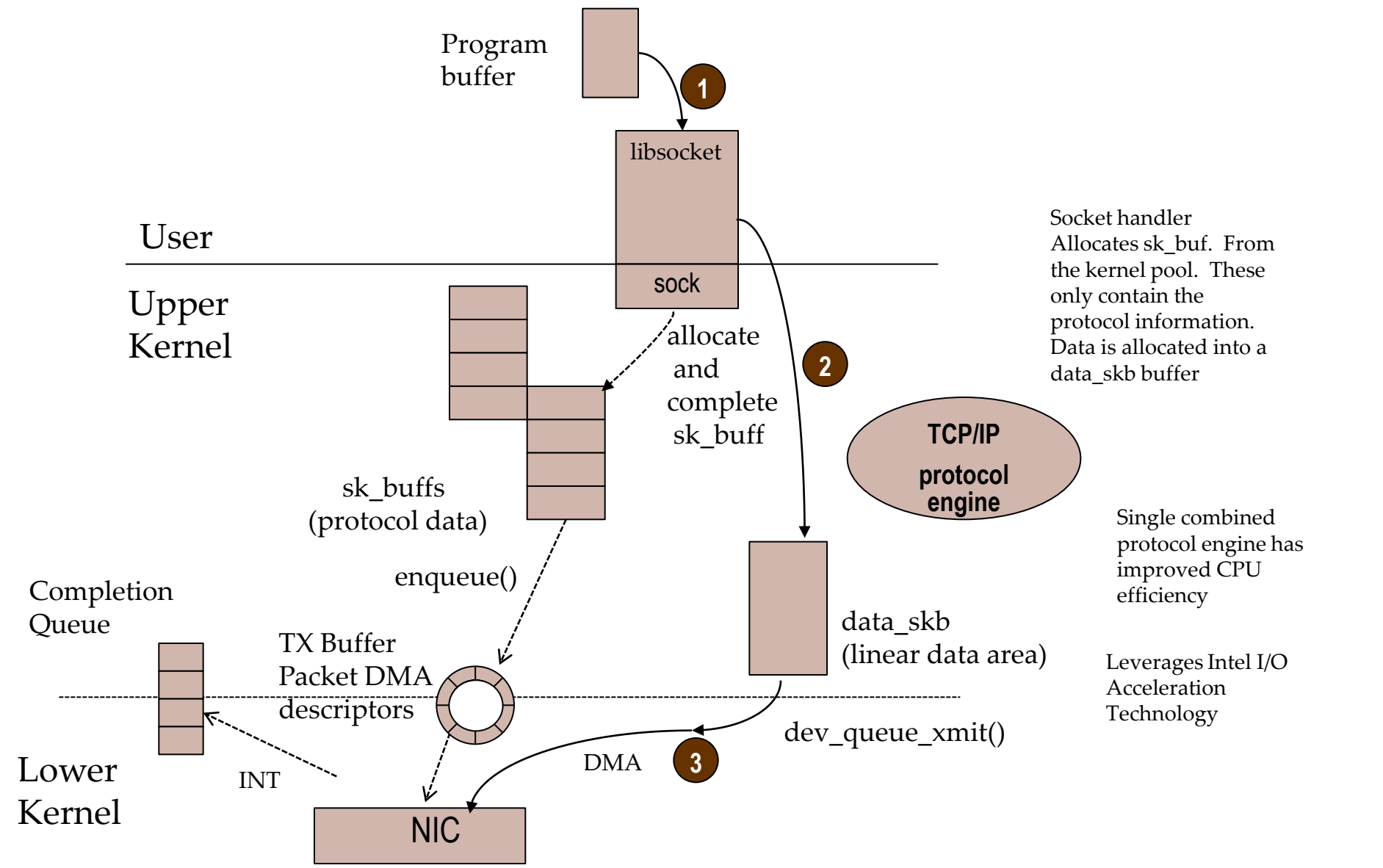
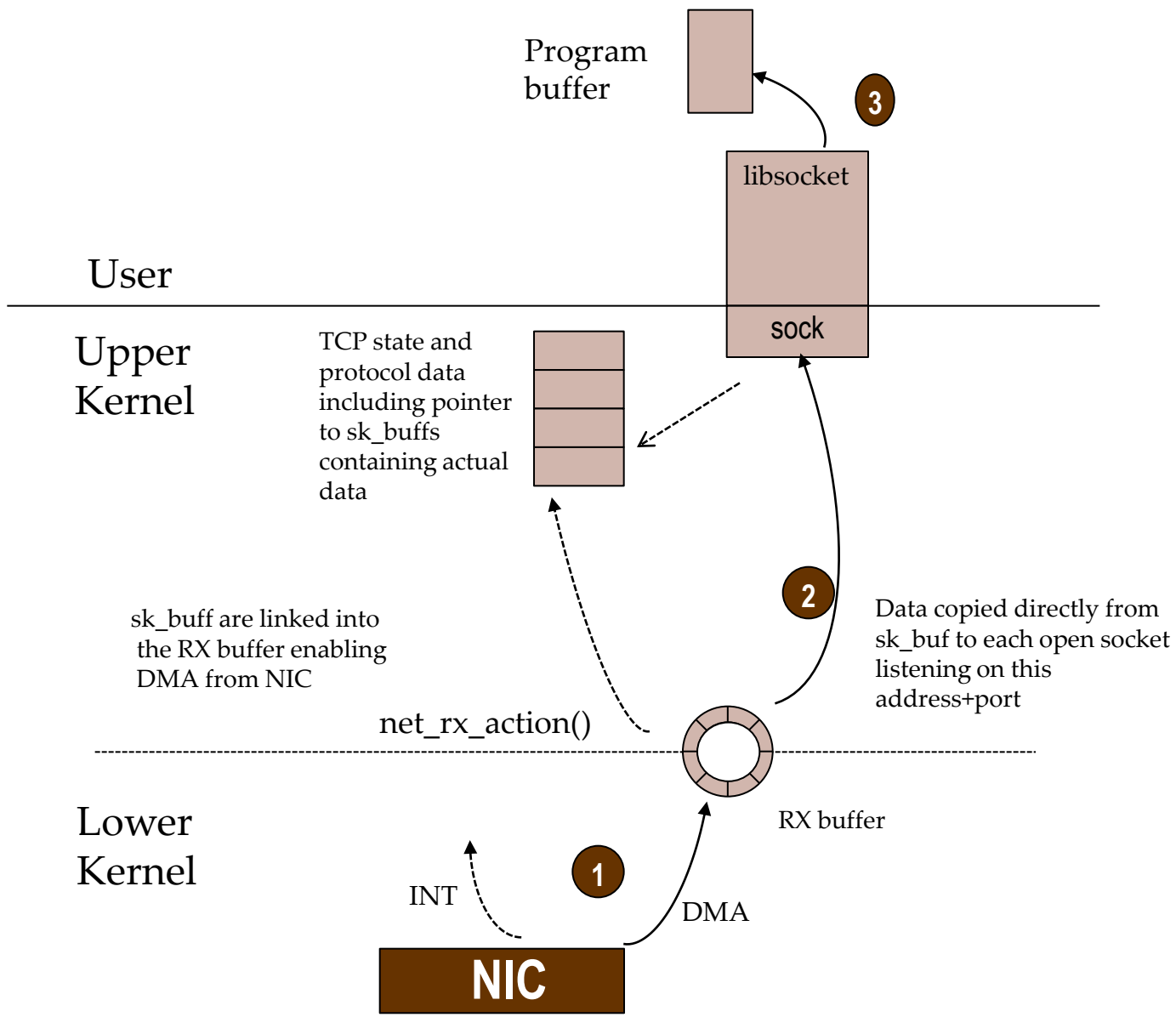


TCP BYPASS

- 1969 - BBN implement first packet switching network control programs on custom hardware to support ARPANET which were to evolve into the first IP routers
- 1973 - TCP/IP defined V. Cerf, E. Cain and R. Kahn as part of ARPANET program funded by the US DoD to enable interconnection of different networks
- 1983 - University of California, Berkeley add TCP/IP networking, sockets and rcommands to UNIX 4.2 BSD
 - Basis for most host implementations and defines standard for network programming
 - TCP/IP one of several protocols alongside UNIX domain sockets, XNS, X.25 and HYPERchannel
- 1983 - Sun Microsystems ships Sun-2, first commercial system with TCP/IP and Ethernet built in as standard. They were to later add NFS, Yellowpages (NIS) and RPC's.
- 1985 - Dennis Ritchie, Bell Labs, creates kernel streams to allow easy implementation of protocol stacks as part of UNIX Eighth Edition
- 1989 - Sun and AT&T jointly implement streams based TCP/IP stack as part of SVR4 development and for use in Solaris 2. (They also implemented a full ISO stack).
- 1992 – Linus Torvalds posts Linux version 0.98 which included TCP/IP networking
- 1994 - Microsoft release it's own Winsocket TCP/IP add-on implementation for Windows 3.11 joining several 3rd party packages already available. It would not become standard until Windows 95 was released the following year
- 2001 Linux community re-implement TCP/IP stack to improve performance with version 2.4
- 2004 - Sun replaces Steams TCP/IP stack with FireEngine to improve performance as part of Solaris 10
- 2007 - Microsoft re-implement TCP/IP stack as part of win6 (Svr 2008) to improve performance





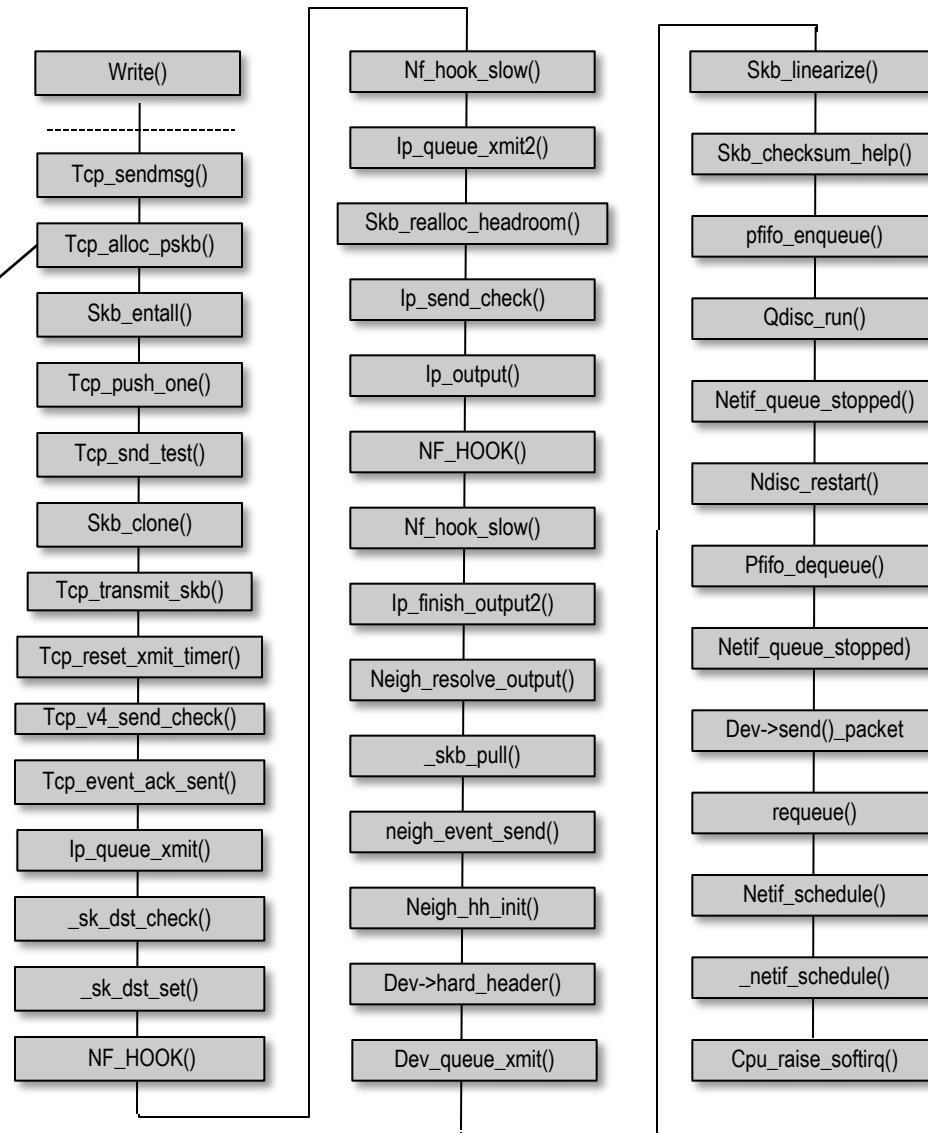


Linux TCP/IP stack – Tx Call Flow

Assumes route
found and
connection
already
established

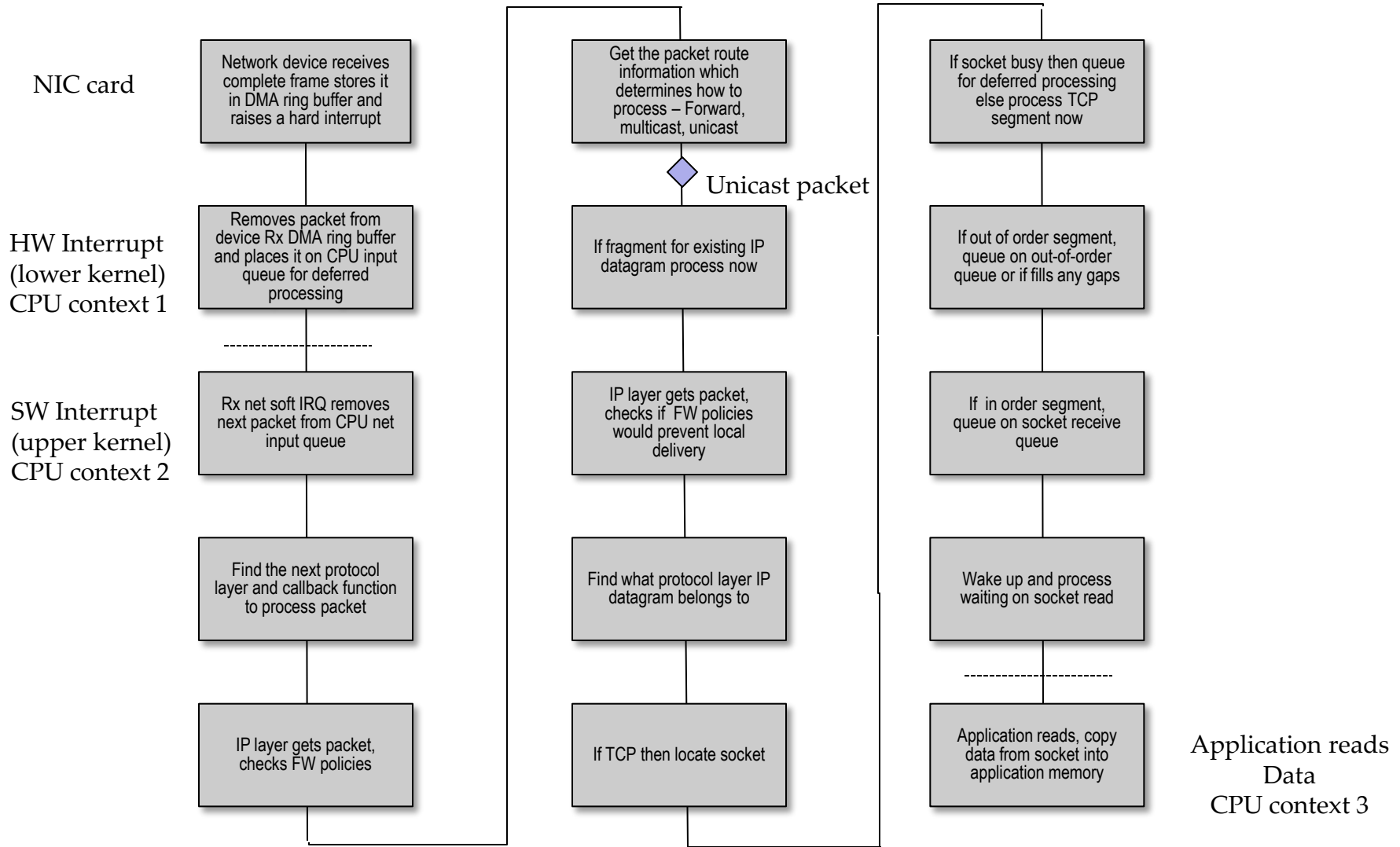
Queue TCP
segment in
sockets write
queue


Runs in same
CPU thread
throughout.
May switch
context at
completion



Queue sk_buff in
Tx DMA ring
buffer

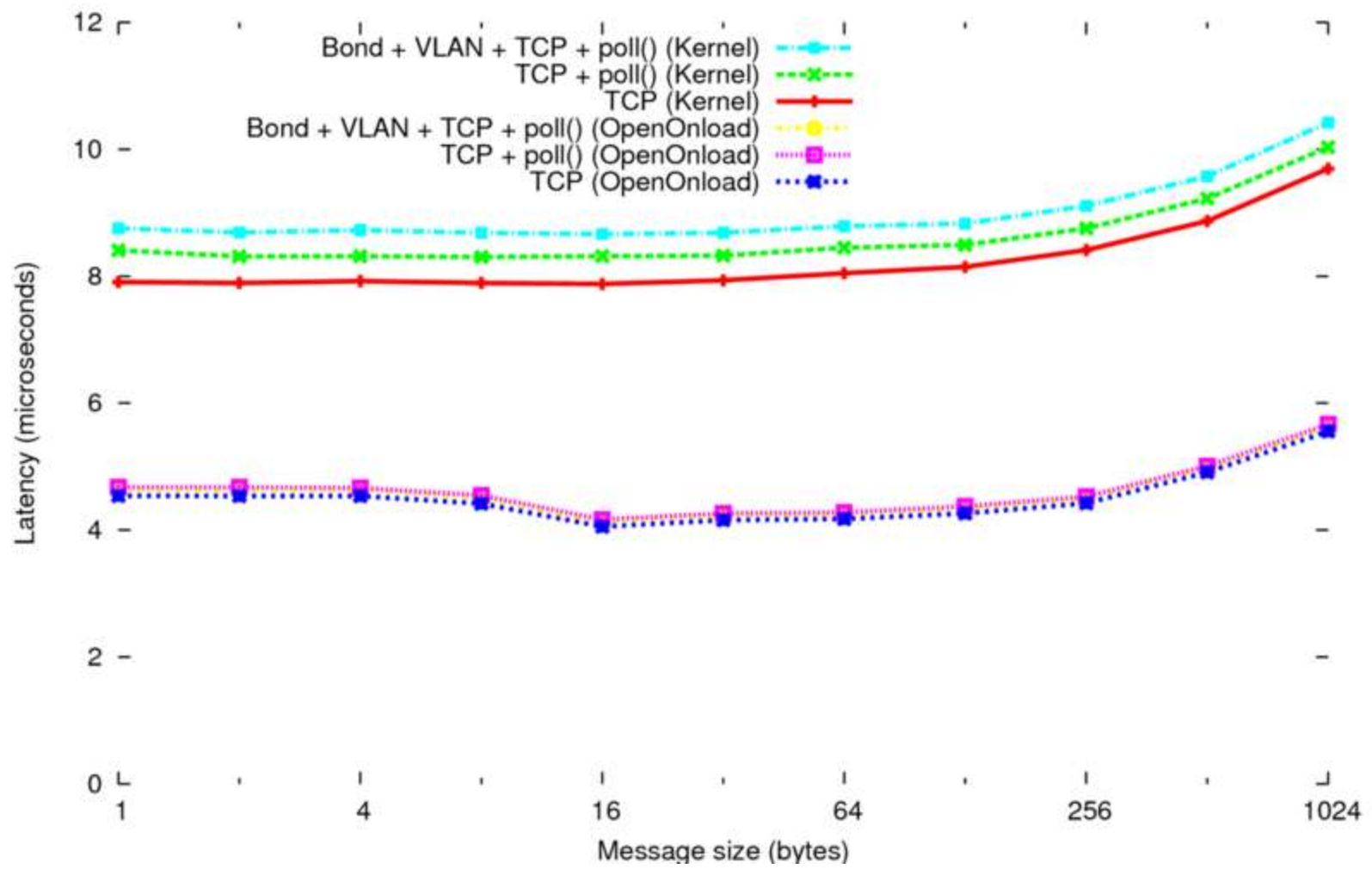
Linux TCP/IP stack – Rx Call Flow



- Zero copy
 - Avoid copy from User space to kernel space, DMA from/to User space. Issues with holding onto buffers to support TCP retransmission requests, no current API/signalling to tell User space code when it can discard buffer. Implementations available but not in mainstream distributions.
 - Only the webserver Use case of Filecopy write to socket available in mainstream since this can DMA from the file buffer cache and can be called directly from the TCP code to release the buffer. Note syntax differences between UNIX implementations cause portability issues.
 - NIC optimizations
 - Interrupt coalescing. Reduces CPU load and can therefore increase throughput where this is a constraint but adversely impacts latency
 - NAPI support lowers CPU load by disabling RX interrupts and only polling under high load. Adaptive; resorts back to RX interrupts when load is lowered
 - Scatter gather I/O transfers from multiple blocks in a single DMA operation avoiding a kernel memory copy up to the 64K allowed for an IP packet
 - Offloads for TCP Segmentation, checksums, Large Receive
 - Receive Side Scaling (RSS) spreads RX load across multiple CPUs
 - TCP Offload engines
 - User Space TCP/IP implementations
 - Bypass TCP altogether
- 
- Covered next

- Leveraging NIC processing power to reduce host CPU demand
- Original Patents filed by Larry Boucher (Auspex later Alacritech) in 1990
- Typically no throughput or latency improvement since NIC CPU is not as powerful as host
- Issues with how to integrate with existing host stack and host based firewalls – referred to as Parallel Offload
- Microsoft licensed Offload Patents and supports it with the “Chimney” implementation in Windows Svr 2008
 - Fully integrated stack
 - Dependencies on NIC driver quality
- Linux community decided against integrated stack and only support presenting the offload engine as a iSCSI initiator device
 - Avoids host stack integration issues
 - Limits offload to block storage devices
 - OFED stack for RDMA support recommended rather than offload engines
- No Solaris support for either modes

- Developed initially to allow easier experimentation with TCP/IP protocols – e.g. Daytona
- Later adapted to leverage zero copy and maintain CPU context
 - DMA's directly to/from User memory
- Challenge is integration with kernel stack and state management
- www.Openonload.org developed by David Riddoch, now of Solarflare
 - Only TCP/IP (UNIX) stack to achieve single threaded wire speed throughput on 10G Ethernet, (compared with 5.5Gbps host based)
 - Linux only, and limited to Solarflare HW
 - Solarflare reported Openonload latency of ~10 μ S RTT on 10ge
 - Constraints:
 - Solarflare standard support only covers the standard kernel TCP/IP stack
 - Enterprise OpenOnload support available at an extra \$400 (per card)
 - Trades CPU for lowest latency, e.g. RX spin wait rather than interrupts
 - No Broadcast. No Multicast within same host, single (offload receiver) for each MC group (unless sharing same offload User stack) –transparently move to kernel stack
 - Bonding only between (offloaded) Solarflare ports
 - No ipfilters, netstat, tcpdump, wireshark



Courtesy Steve Pope Solarflare, Nov 11 2010

- TCP provides error detection and correction, in-order delivery of data, flow control and congestion management
- IP provides routing, packet chopping and aggregation, error detection and a address namespace
- IP allows us to scale by routing across multiple L2 subnets
- We can only replace TCP/IP by finding alternatives to these
- By limiting to Layer2 networks we can drop IP but need to provide a namespace and still need to overcome any scalability issues
- But Ethernet has no guaranteed delivery and is permitted to arbitrarily drop packets
- Ethernet subnets are rarely extended beyond 1024 addresses due to broadcast issues
- FibreChannel provides error detection, reliable in-order delivery, credit based flow control and a namespace but error recovery is very expensive and it is lacking multicast support

- iWARP - RDMA over Ethernet
- Converged Ethernet – Data Centre Enhanced Ethernet
- GAMMA – Genoa Active Message Machine
- Open-MX – Myricom API
- InfiniBand – Converged interconnect

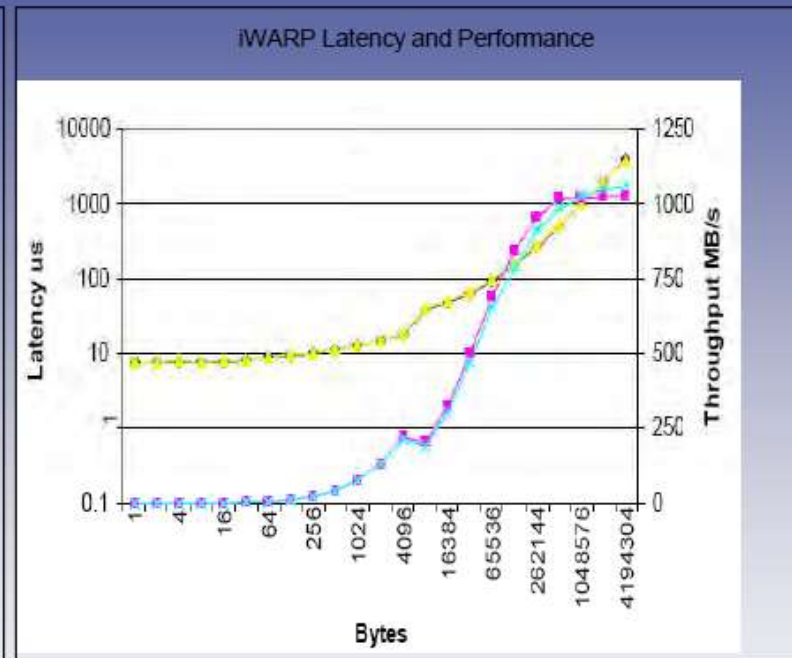
- Designed to provide RDMA over Ethernet
- Implemented on both 1G and 10G Ethernet NICs with RDMA support
- Replaced proprietary RDMA implementations
- A suite of protocols composed of RDMAP (RDMA protocol specifications) and DDP (Direct Data Placement over reliable transports) for RDMA over Ethernet
- Defined by www.rdmaconsortium.org and submitted as draft to IETF
- Uses TCP to provide reliability and in order delivery of data
- Implemented in hardware by some NIC vendors e.g NetEffect (Intel), NexXen, Silicom
- API complexity and TCP overhead resulted in lack of take up from developers

- Designed to support what is called FCoE but is best described as SCSI commands over reliable Ethernet in a L2 network
- Dedicated Ethertype 0x8906 which is lossless, and 8 separate Pause channels
- The majority of new 10G Ethernet NICs provide the hardware capabilities require to support Converged Ethernet
 - FCoE drivers are still only available¹ from a subset of these and add a premium to those NICS (e.g. Brocade, Emulex, Qlogic)
- Defined by a collection of standards emerging from the IEEE Data Centre Bridging (DCB) working group
 - IEEE 802.1p Class Based Flow Control
 - IEEE 802.1Q Virtual LAN
 - IEEE 802.1au Congestion Notification
 - DCB Exchange protocol
 - Working group defining a spanning tree replacement with TRILL
- Reality is currently best defined by Cisco Nexus today
- API's are targeted for Storage use today but OFED support for CE with RDMA is now improving (RoCEE). Promising in medium term

Note1: OpenFCoE moved into Linux mainstream 2.6.30 with initial support for Intel and Broadcom

RoCEE v. iWarp Performance

- Mellanox results, comparing their CEE (CE enhanced with RDMA) and iWARP
- Single trip latency measured
- Comparable InfiniBand under same test and measurement conditions is ~1uS



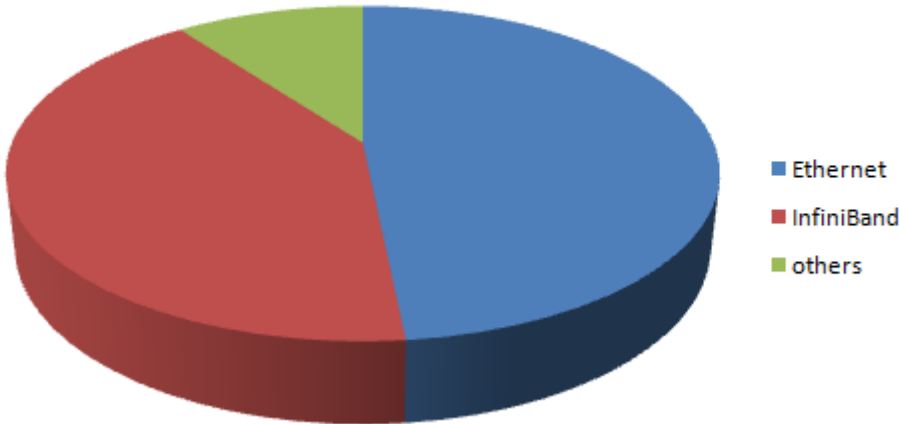
Features	Low Latency Ethernet	iWARP
Latency	~3us	~8us
Efficiency	Sustains; ~9us @ 4KB	Increases to ~50us @ 4KB
Bandwidth	Line-rate @ 512B	Line-rate @ 512,000B

Ethernet Technology Summit, San José, Feb 2010, Gilad Shainer, Mellanox

- **GAMMA – Genoa Active Message Machine**
 - Open source project led by the University. Used in other projects such as FOAM and includes an implementation of MPICH to support HPC applications
 - Reported to have achieved latencies of around 10.2µS
 - NIC dependencies, currently only supported by specific Intel and Broadcom hardware
 - The port being used to run GAMMA must be dedicated and cannot also support a TCP/IP stack
 - Requires some Linux kernel changes
- **Open-MX**
 - Myricom were a market leader in cluster interconnects and developed a popular API used mainly by the HPC community
 - Now Open sourced this API and support converged Ethernet adaptors with RDMA
 - Supported by a number of HPC applications including MPI and PVFS2
 - Leverages Intel's I/O Acceleration Technology
 - Best run on Myricom's own 10G NICs for support reasons

www.top500.org

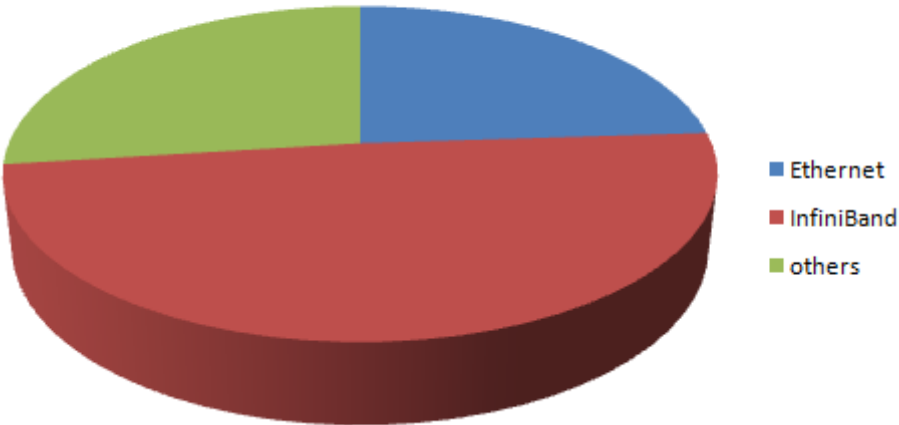
count



interconnect	count	Rmax sum
InfiniBand	207	15,895,656
Ethernet	242	7,776,035
others	51	8,762,992
totals	500	32,434,683

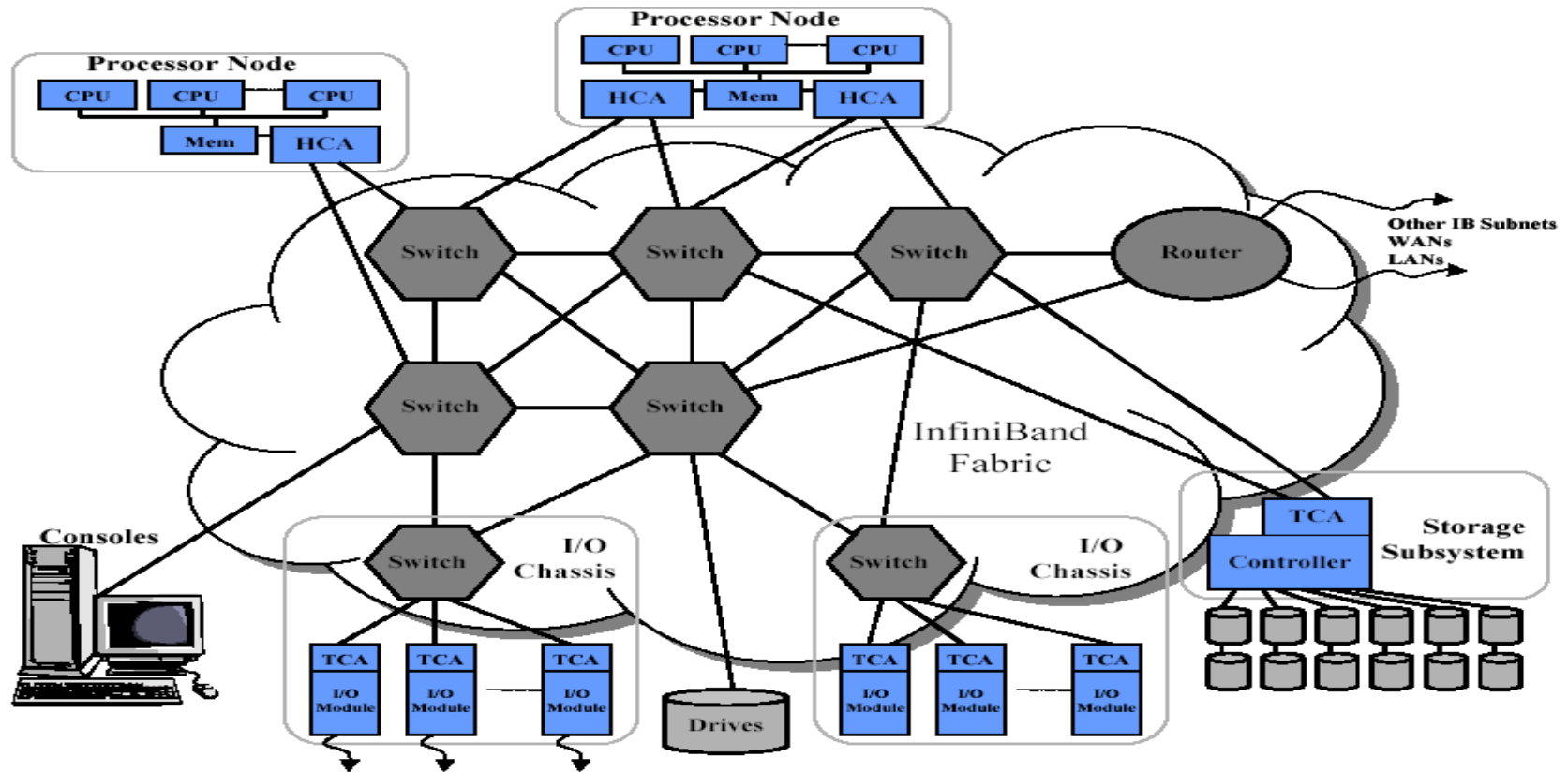
Rmax is measured using the Linpack benchmark and is a measure of compute cluster throughput

Rmax sum



InfiniBand supports most of the world's top500 compute power

InfiniBand fabric switching – logical topology



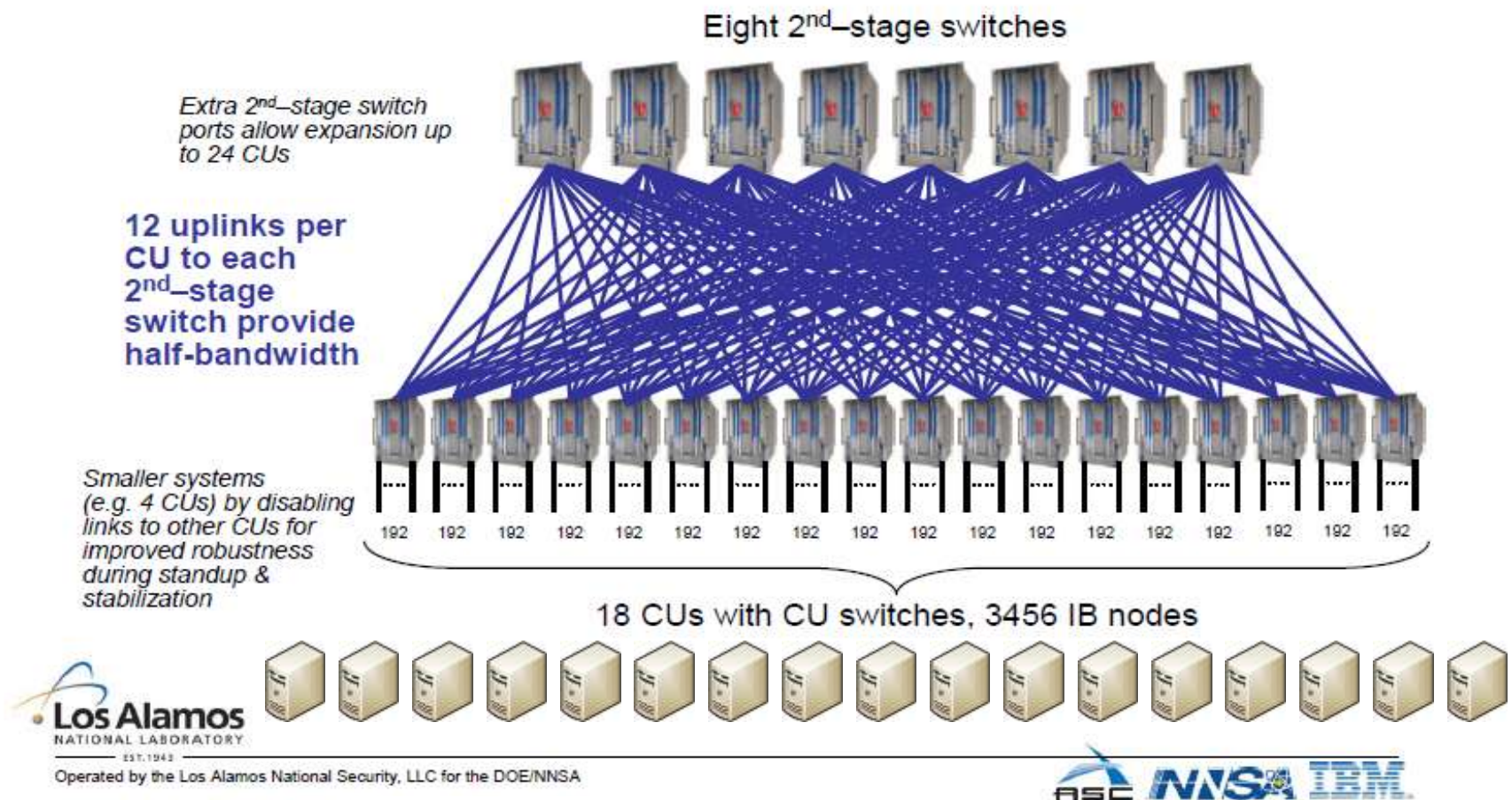
- Layer 2 network with 1,4,12 channel link aggregation (individual packets stripped across all channels)
- Reliable delivery of packets
- RDMA is part of the specification, mandatory component
- Wide range of available path selection algorithms – no spanning tree
- Cut through implementation - end to end, 100nS port to port

Drawing courtesy of InfiniBand Trade Association

Version 1.6

InfiniBand scalability has been proven

- DOE "Roadrunner" cluster, #3 in top500 June2010 with 122,400 cores across 3456 nodes
- 26x 288 port InfiniBand switches configured into a single L2 subnet (total of 37,440 internal and external ports to manage)
- Started building August 2008, operational since April 2009



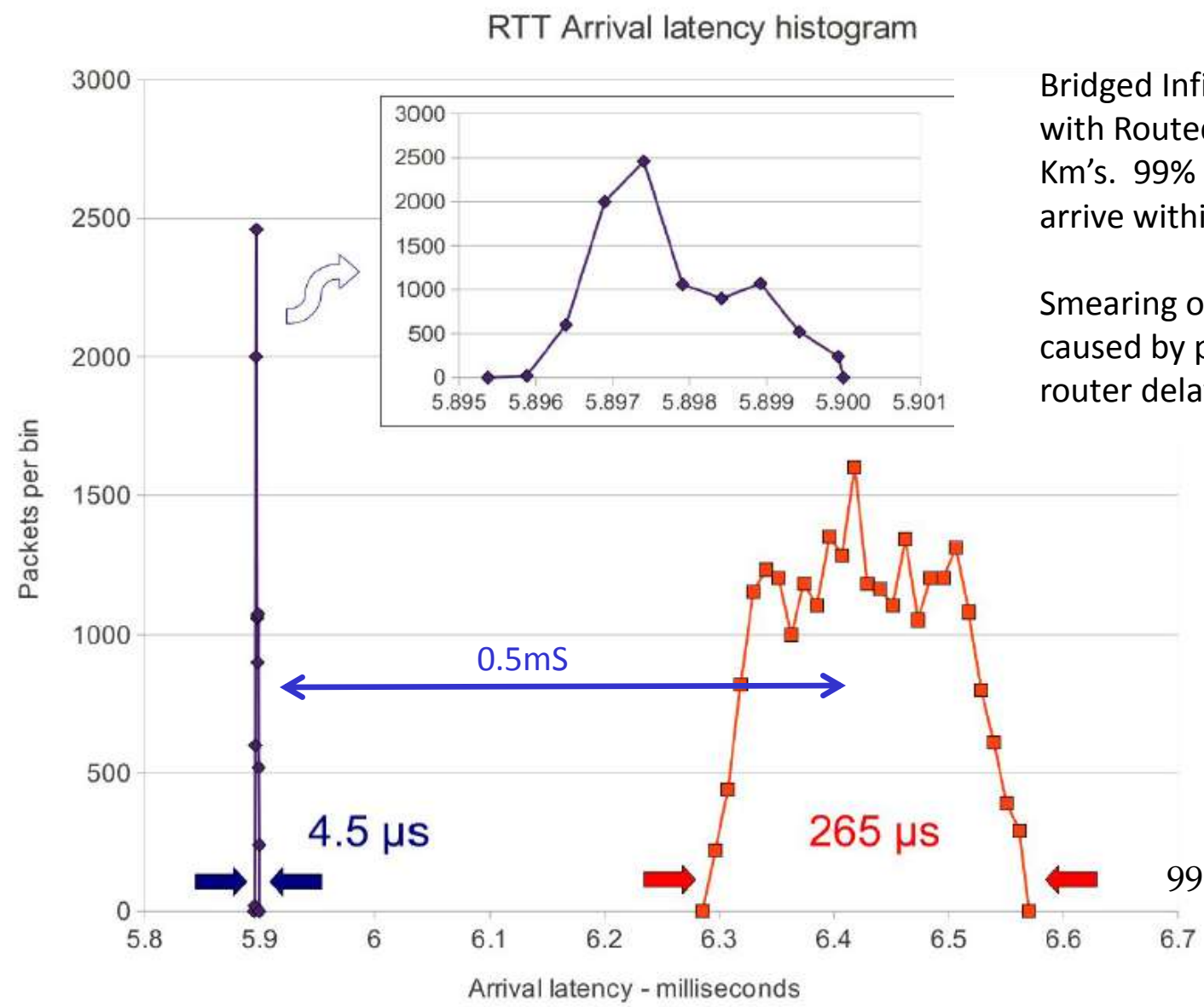
Advantages of InfiniBand over Ethernet

- Cut through design with hardware generated link and end2end CRC's and late packet invalidation
 - InfiniBand has better error detection and correction at link level
 - Ethernet reverts to store and forward under numerous conditions
 - Some popular Ethernet “cut through” switches have to wait for 1024 bytes to be received
- Implicit link aggregation, bundles of 1,4,12 physical links into a single “logical” channel. Handled transparently by the hardware
 - Ethernet trunking is inconsistently implemented e.g. Cisco EtherChannel v. 802.3ad
 - Ethernet keeps individual packets to single link; InfiniBand stripes with 64 byte transmission units which lowers serialization latency.
- Standardized RDMA to lower CPU overhead and increase throughput
 - Only way to achieve wire speed, single session throughput as B/W > 10Gbp/s
- Legacy Ethernet protocol and buffering constrains large switch implementation – max possible today:
 - InfiniBand 648 port zero contention 40Gbps, 3052 ports at 20Gbps, zero contention.
 - Cisco Nexus 7018 with total of 512x 10Ge ports but only 128 at zero contention
- Latency
 - Ethernet today is 710nS (port to port FIFO for cut through switches) BladeTech G1824
 - InfiniBand QDR is 100 nS (port to port FIFO) Voltaire 4036

- Mesh networks provide the greatest bandwidth but are difficult to manage
 - Ethernet Spanning tree simply prunes them! DCB working on Trill as a replacement
 - The HPC community have solved this with a number of pluggable path selection algorithms for InfiniBand with support for mesh topologies and that avoid packet loops
- Congestion Management and QoS
 - Reliable networks create backpressure on senders until packets can be delivered.
 - Unless managed, this will prevent any packets from being transmitted from a specific interface
 - InfiniBand includes a sophisticated mechanism of Service Levels, Virtual Lanes, packet arbitration and protocol registration which enables different applications, ports or servers to share the same fabric and be allocated different bandwidths and priorities to match their individual latency requirements
 - Converged Ethernet offers:
 - 8 pause channels but no method to map against applications
 - Pause is still a reactive, blunt approach to congestion management

	<i>Service Level</i>	<i>QoS Level</i>	<i>Virtual Lane</i>	<i>Arb Low Weight</i>	<i>Arb High Weight</i>	<i>Worst case delay</i>
	0	Default	0	32		10μS
	1	Default	0			
	2	Default	0			
	3	Default	0			
	4	GPFS	2	32		10μS
	5	Default	0			
	6	IP	3	32		10μS
	7	SDP	4	32		10μS
MC	8	NM	5		32	1.92μS
	9	Default	0			
Trading	10	EX	6		64	1.06μS
	11	Default	0			
Tsync	12	Clock	7		32	1.92μS
	13	Default	0			
	14	Default	0			
	15	IB Mngt	0			

- All applications sharing same physical interconnect but each is assigned to a different Service Level
- Guarantees low latency for those applications that need it
- Standard based QoS configuration – works across any InfiniBand switch , including mixed environments



Bridged InfiniBand compared with Routed Ethernet across 615 Km's. 99% of InfiniBand packets arrive within 4.5 μ S window.

Smearing of Ethernet arrivals caused by packet buffering and router delays

InfiniBand
Ethernet

Tested with multicast over IPoIB, requiring no application changes

99% arrivals distribution

Connection Type	Description	Message Size (Max)
Reliable Connection	Acknowledged-Connection Oriented	2 GB
Reliable Datagram	Acknowledged-Multiplexed	2 GB
Unreliable Connection	Unacknowledged-Connection Oriented	2 GB
Unreliable Datagram	Unacknowledged-Multiplexed	256 B-4 KB
Raw Datagram	Unacknowledged-Connectionless	256 B-4 KB

- Implemented in hardware and supported with RDMA
- Defined by and accessible as the InfiniBand VERBS
- Basis for implementing all upper layer protocols
- Reliable types are fully lossless
- Unreliable Datagram used for Multicast

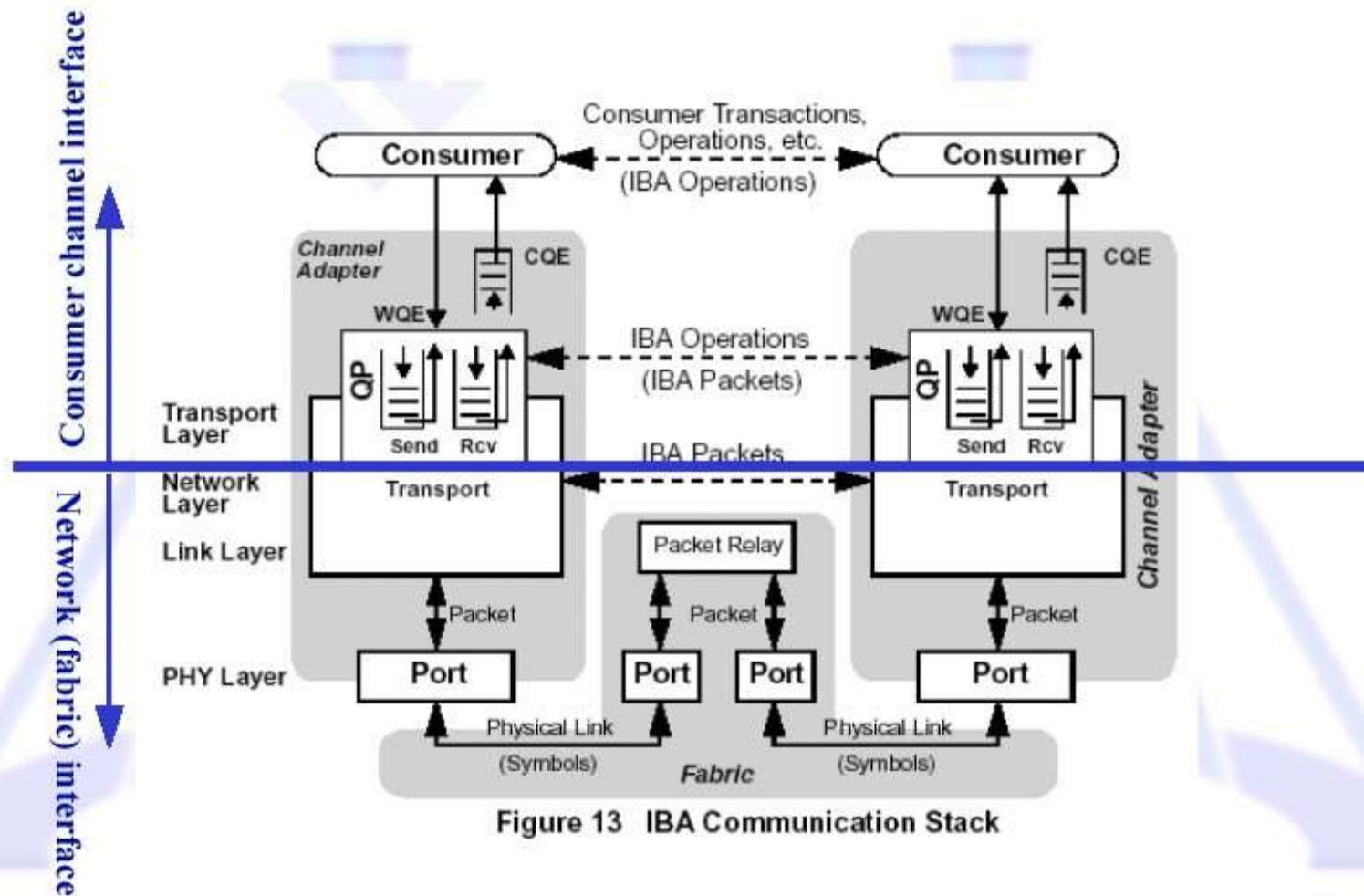


Figure 13 IBA Communication Stack

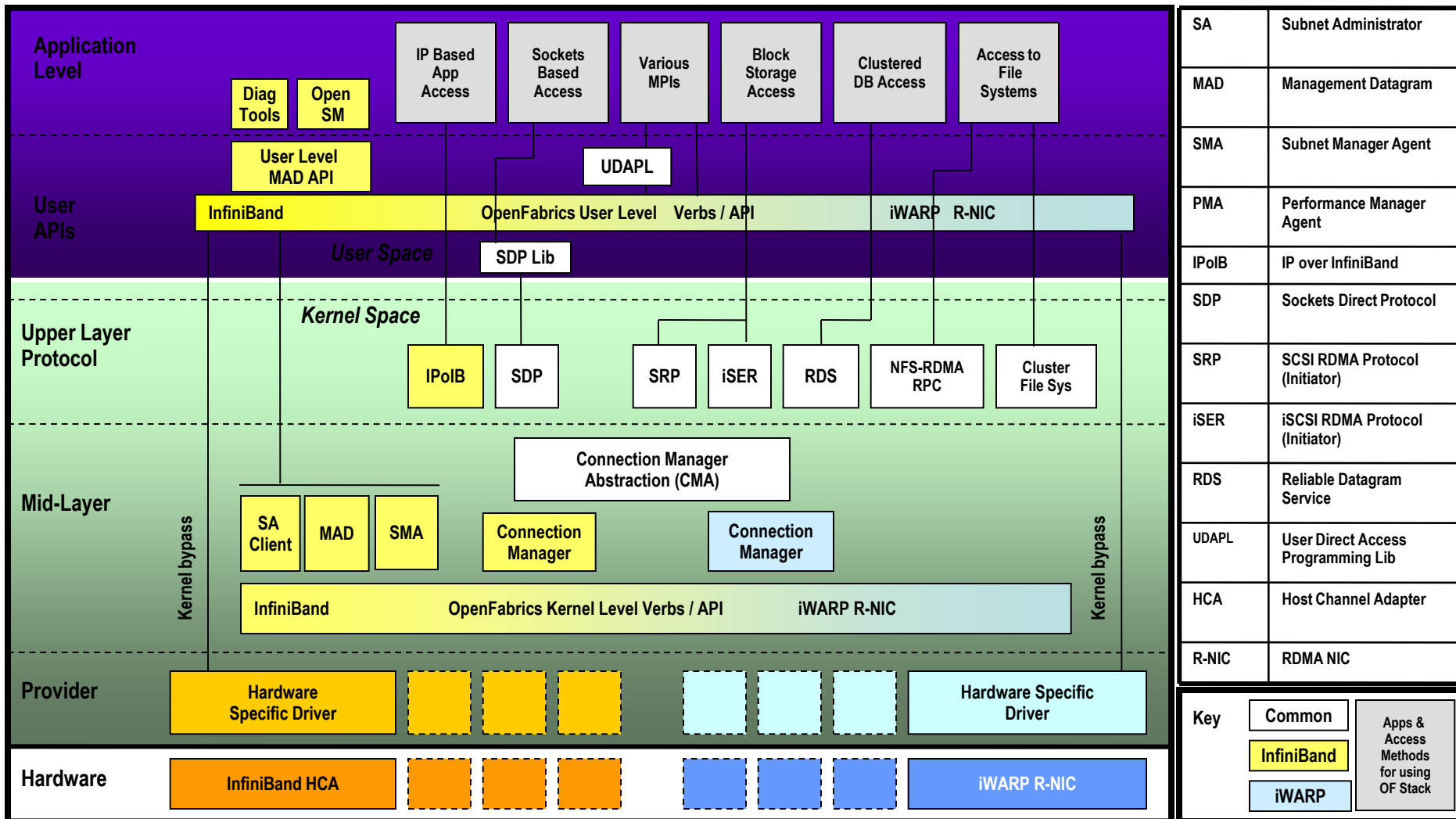
Credit based flow control at physical layer provides reliable delivery

Diagram courtesy of Mellanox

Where does InfiBand come from?

- Originally defined by the InfiniBand Trade Association, formed by Intel, Sun Microsystems and IBM with participation from HP, Compaq and Microsoft
- Current standard is IBA 1.2 - 2400 pages of specification
- Open Fabric Alliance
 - Formerly OpenIB, produce the OFED (Open Fabric Enterprise Distribution) for Linux and Windows
 - Supports both InfiniBand and CNE's with RDMA
 - Accepted by Torvolds and incorporated into Linux kernel since 2.6.14
 - Maintainers include – Chelsio, Cisco, Ohio State Univ., Oracle, IBM, Intel, Mellanox, Neteffect, Qlogic and Voltaire
- Available as a package option for Red Hat and SuSE Linux
- Included within Solaris, AIX, HP-UX
- Microsoft include with Computer Cluster (Grid) release but also freely available from OFED and Mellanox
- Native protocol support included with:
 - MilleniumIT, IBM WebSphere LLM, NYSE Technologies, 29West LBM, IBM GPFS, RNA networks, TIBCO FTL
- InfiniBand switches from Mellanox/Voltaire, Sun/Oracle and Qlogic

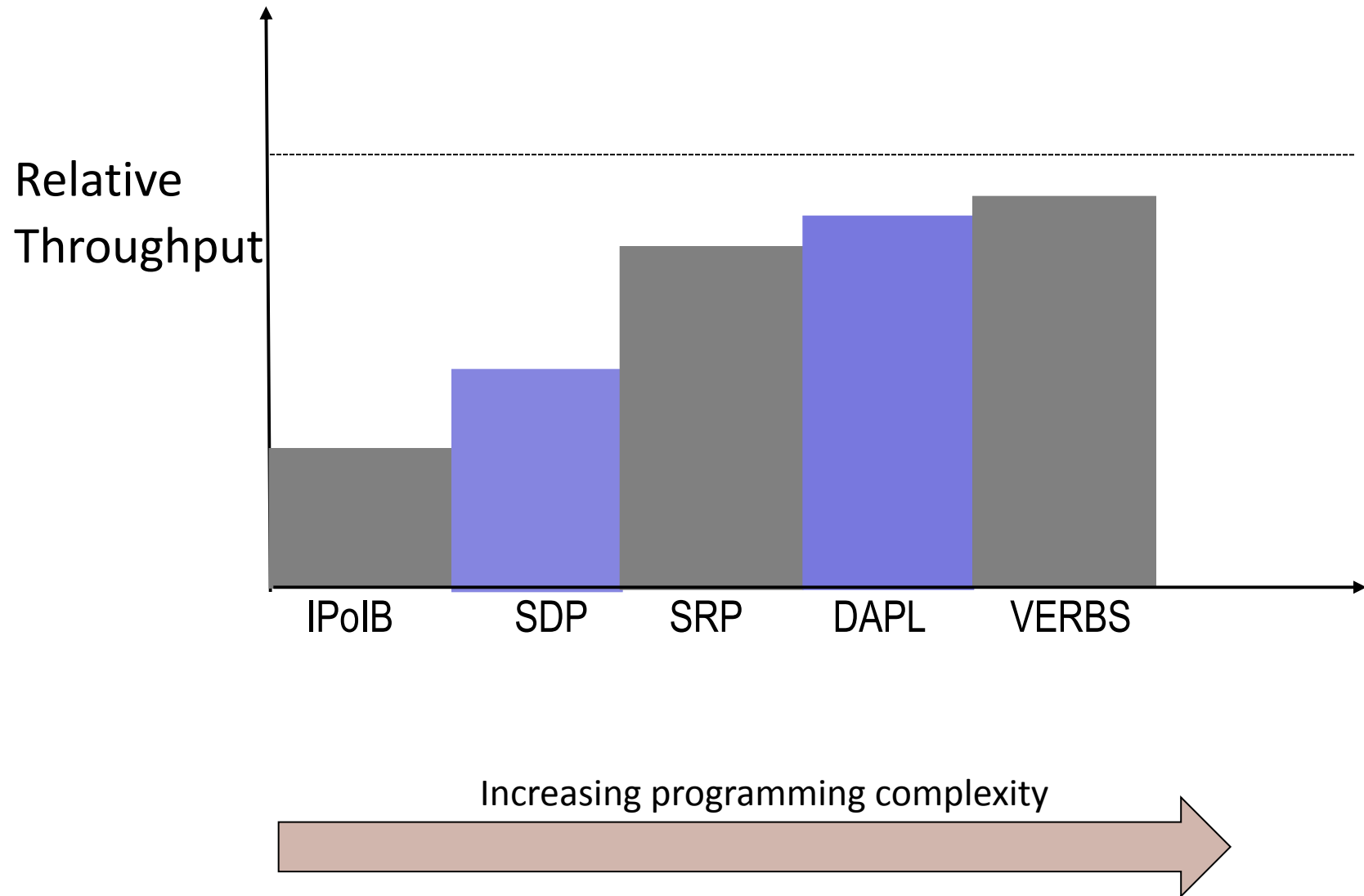




SA	Subnet Administrator
MAD	Management Datagram
SMA	Subnet Manager Agent
PMA	Performance Manager Agent
IPoIB	IP over InfiniBand
SDP	Sockets Direct Protocol
SRP	SCSI RDMA Protocol (Initiator)
iSER	iSCSI RDMA Protocol (Initiator)
RDS	Reliable Datagram Service
UDAPL	User Direct Access Programming Lib
HCA	Host Channel Adapter
R-NIC	RDMA NIC

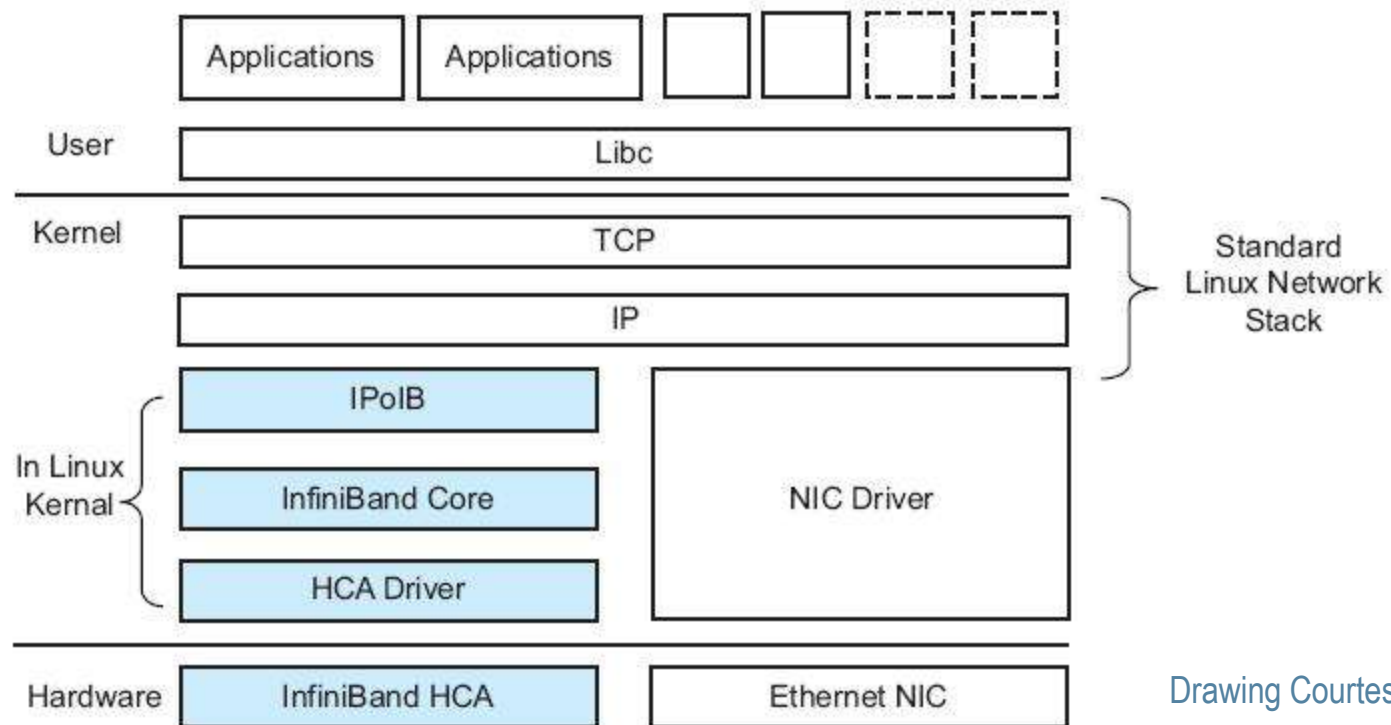
Key	Common	Apps & Access Methods for using OF Stack
	InfiniBand	
	iWARP	

- Just treat it like a TCP/IP socket
 - Running IP over InfiniBand (IPoIB)
 - A logical IP interface (/dev/ib0) for each InfiniBand partition
 - Full support of all IP protocols and multicast
- Just treat it like a storage device
 - SRP attached disks appear identical to FibreChannel attached drive
- For optimal performance:
 - Bypass TCP/IP stack
 - zcopy and RDMA transfer direct between Application memory spaces is required. This does require the support by application code. Several API's of varying degrees of complexity and performance provide this
- Highest performance, sometimes referred to as RAW is to use the Verbs provided by the hardware.



Defined by IETF in RFC 4391.

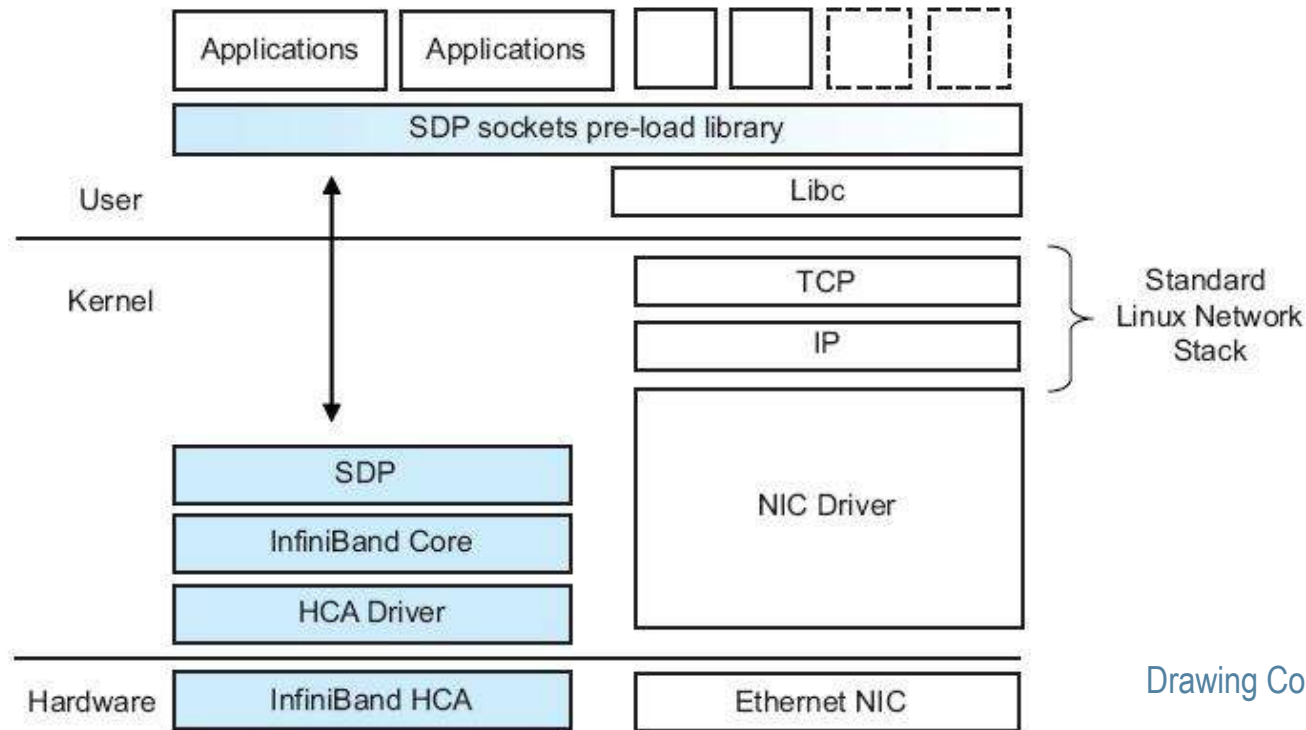
Implemented in OFED and other vendor stacks



Drawing Courtesy of the IBTA

Fully supports multicast, using standard socket options and Linux Bonding module

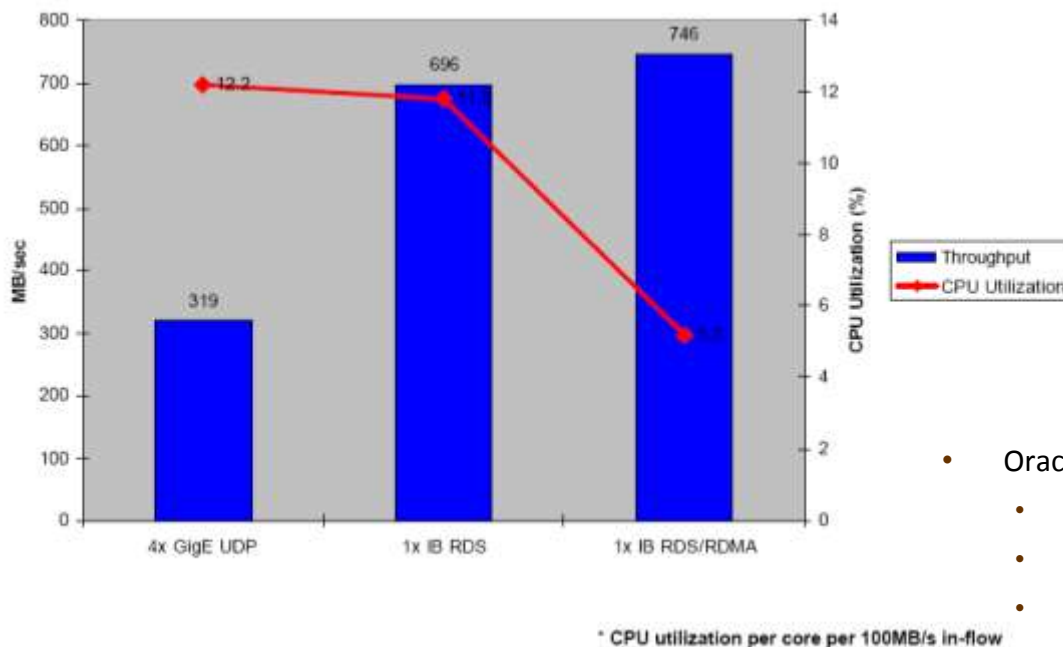
Host configurable in Connected (default) or datagram mode



Drawing Courtesy of the IBTA

- Preload libsdp to switch existing binaries to use SDP
- Only for TCP – no UDP or Multicast
- Defined addresses or networks in /etc/libsdp.conf to automatically switch to SDP
- Reverts to TCP if not able to establish an SDP connection
- Configure when to switch to RDMA, default 32K
- Independently implemented on both Solaris and Windows

- Reliable datagram Socket – Guaranteed delivery
 - block until received or poll for completion
- `socket(2)` API supported
 - `socket`, `bind`, `ioctl`, `sendmsg`, `recvmsg`, `poll`, `getsockopt`/`setsockopt`
- Uses IPoIB for address resolution
- Peer-to-peer connection model
 - node-to-node connection
 - on-demand connection setup
 - connect on first `sendmsg()`
 - disconnect on error or inactivity
- Uses Reliable Connection Queue Pair
- `zcopy` send/recv
- Used by Oracle for RAC Interconnect



- Oracle Exadata Database Machine X2-8
 - 128 CPU cores and 2TB for data base processing
 - 168 CPUs cores for storage processing
 - 5.3 TB Flash Cache
 - 100TB storage per rack
 - Embedded 40G InfiniBand
 - I/O Bandwidth of 50GB/seconds

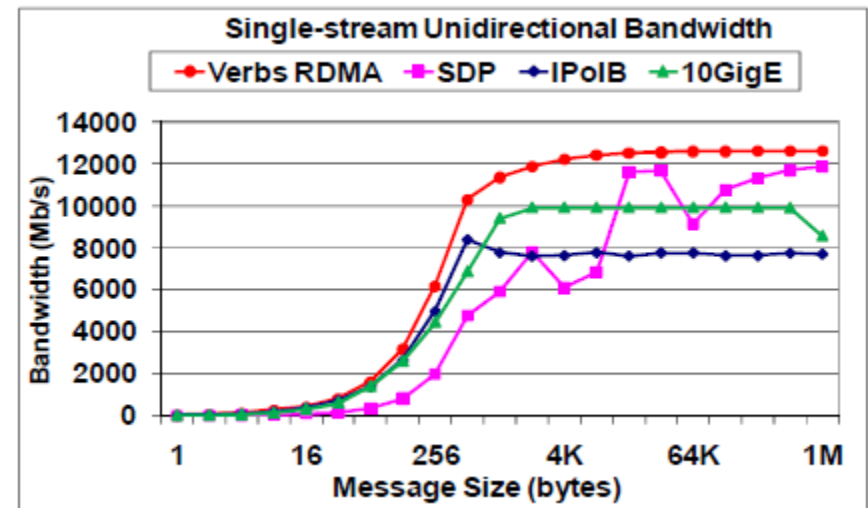
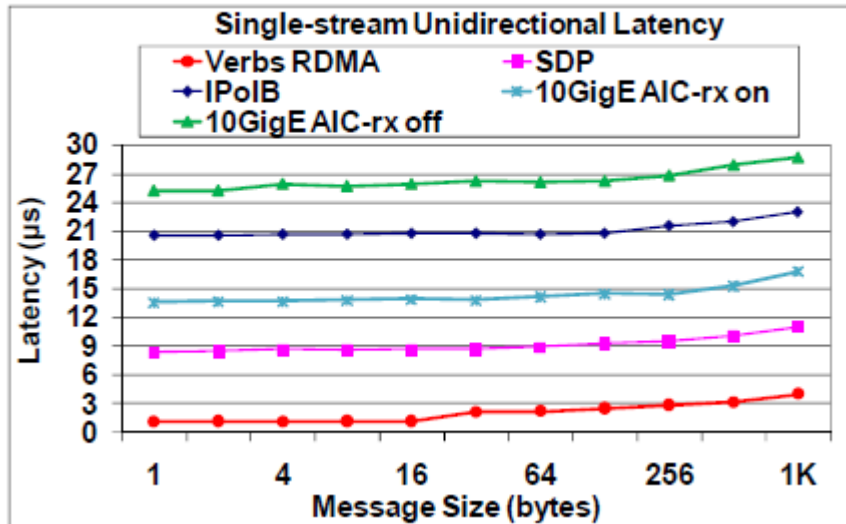
http://www.infinibandta.org/events/IBTATechForum08_/IBTA_TechForum_08_Agenda/8_Oracle_Database_and_InfiniBand_Technology_presentation.pdf

- Direct Access Programming Library
- Defined by the www.datcollaborative.org
- Generic 'C' API for RDMA access
- User space transfers using RDMA
- Register memory to pin into memory
- Shared key to manage access between peers
- High cost to establish but then very high B/W at low CPU overhead e.g. 2GB transfer takes a single 'C' call
- Highest B/W using Writes
- DAPL supports both InfiniBand and Ethernet iWARP RDMA
- Implemented in Linux and Solaris believed to be there for Windows, HP-UX and AIX
- Very complex API with around 76 functions, limited adoption from application community

- VERBS initially defined in the InfiniBand standard (IBA) but also supported in RoCEE as a result of its InfiniBand encapsulation across Ethernet
- Programming model requires dealing with Queue Pairs, registering memory for RDMA access and asynchronous completion events
- Native Verbs API (VAPI)
 - HW specific flavour of the IBA defined verbs. HCA dependent
 - Highest level of performance
 - Need to use GUIDs and LIDs for addressing
 - Can expose HCA chip dependencies
 - Constantly change as new features are added to the hardware, e.g. Tavor -> Arbel
 - Used mostly by MPI programs to add InfiniBand support
- Libibverbs API
 - Linux library supporting both InfiniBand and RoCEE enabled Ethernet cards
 - Provides some level of isolation from HW specifics
 - Wrapping of native VERBS into Linux library
 - Refer to Mellanox “RDMA Aware Programming manual” and to OFED source code of test tools for examples

- Linux only RDMA API
 - **rdma_cm(7) - Linux man page**
- Developed for both InfiniBand and RoCEE
- Based on sockets but adapted to support Queue Pairs (QP's)
- Limited to connection management, i.e. creates and connects QP's
 - Leverages IPoIB namespace for which simplifies addressing
- Includes support for Multicast connections
- Alternate Path Migration (APM) support now included to enable reliability through server port failover (Linux bonding driver only supports IPoIB)
- Use in combination with libibverbs to send/receive data

Latency and throughput by protocol



“Evaluation of ConnectX Virtual Protocol Interconnect for Data Centers”;
 R. Grant, A. Afsahi, P.Balaji; Ontario University and Argonne National
 Laboratory
 Testing same ConnectX card in both 40g InfiniBand and 10g CNEE modes
 with Jumbo frames for throughput

- The TCP/IP stack is now a major performance bottleneck and source of latency and jitter
 - Typical latencies (using same measurement approach) :
 - TCP/IP 20μS, OpenOnload 10μS, native InfiniBand 5μS
- To bypass TCP fully, requires a reliable layer 2 network with congestion management and mesh topology support - still extremely difficult to achieve with current Ethernet switches
- Today, the best option for achieving this is InfiniBand, currently wins on Price, Performance and Latency
- Standardize on the OFED API's to bypass TCP/IP and so that as and when Ethernet becomes the better option, it can be swapped in without application changes
- Chose the programming paradigm best suited to your requirements trading off programming complexity with performance/latency
 - RDMA programming is complex
- Can mix models with:
 - Use existing code with SDP for TCP and IPoIB for all legacy and any non-latency critical apps
 - Modify latency critical apps to use libibverbs and rdma_cm

See also:

- www.openfabrics.org
- www.infinibandta.org
- “RDMA Aware Programming Manual”, www.mellanox.com
- “TCP/IP Architecture, Design and Implementation in Linux”, S. Seth
- “Understanding Linux network internals”, Christian Benvenuti
- Ask about Informatix Solutions workshop deeper dive modules including:

Programming with OFED
Ethernet v. InfiniBand
Java and .NET latency tuning
Low latency tuning for Solaris and Linux

InfiniBand Overview
Long Distance InfiniBand
InfiniBand Management and Observability
InfiniBand diagnostics and troubleshooting
InfiniBand Product Selection

www.informatix-sol.com

richard@informatix-sol.com