

Virtual Interface Architecture (VIA)



Hassan Shojanian

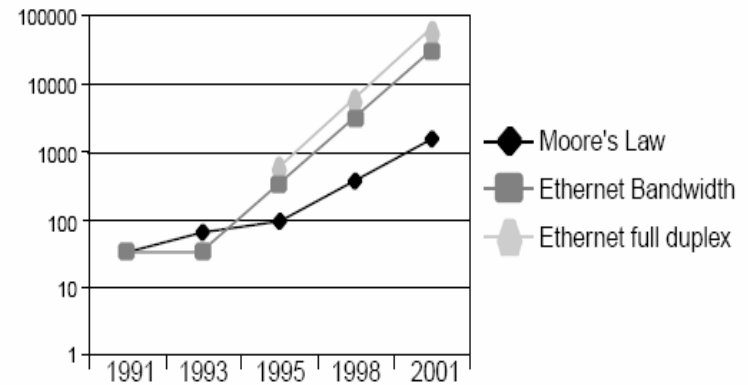


Agenda

- Introduction
- Software overhead
- VIA Concepts
- A VIA sample
- Design alternatives
- M-VIA
- Comparing with InfiniBand Architecture
- Conclusions & further of VIA
- References

Introduction

- Why user-level messaging layer?
 - Fast increasing network bandwidths. Surpassing Moore's law and increase in CPU speed.
 - Applications not able to take advantage of this high bandwidth.
 - Poor Inter-Process Communication (IPC) results in low performance within a cluster. → not scalable



Network Speed vs. Processor Speed Over Time

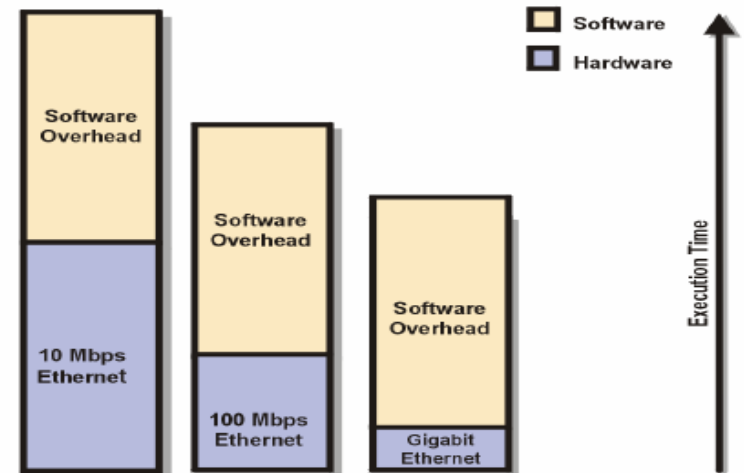
Figure taken from [Cameron '02]

Introduction -2

- Why user-level messaging layer?
 - High data rate of physical layer is not proportional to communication performance:
 - Software overhead
 - Traffic pattern

Transmission time vs. protocol overhead

Figure taken from [Bialek '01]



Stack Time vs. Hardware Execution Time

- Solution?
 - Software overhead comes from code execution by CPU, so...
 - Just get a better CPU?!!!
 - Faster CPU: Better Pipelining and higher clocks. Also means higher penalty for shuffling CPU state...
 - Context switch and interrupt handling are still very expensive!
 - Get specialized networks? System Area Networks (SANs) are around for a while, but:
 - Unique and proprietary → high cost, difficult to get HW/SW interoperability.
 - **★ VIA fills the gap between performance and standardization.**

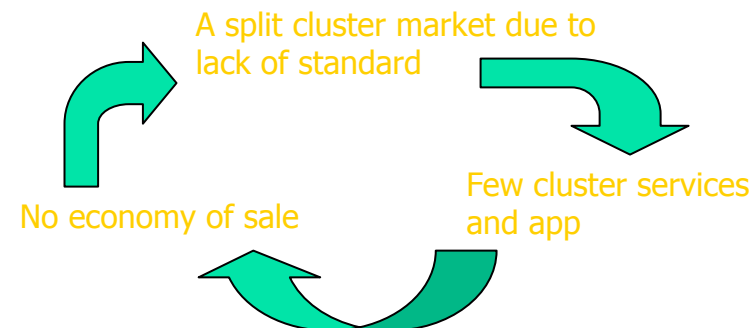


Figure taken from [Bialek '01]



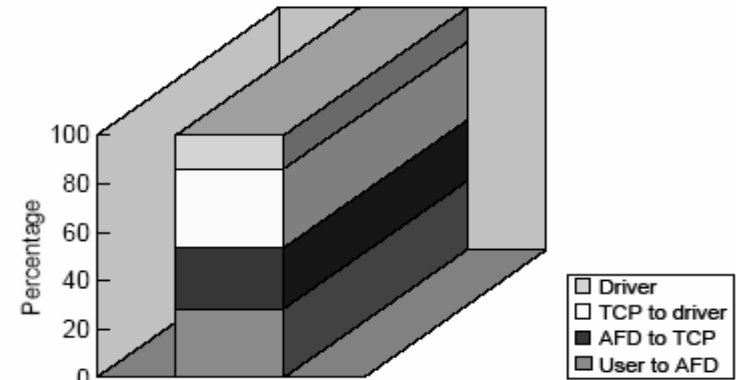
Introduction -3

- VIA:

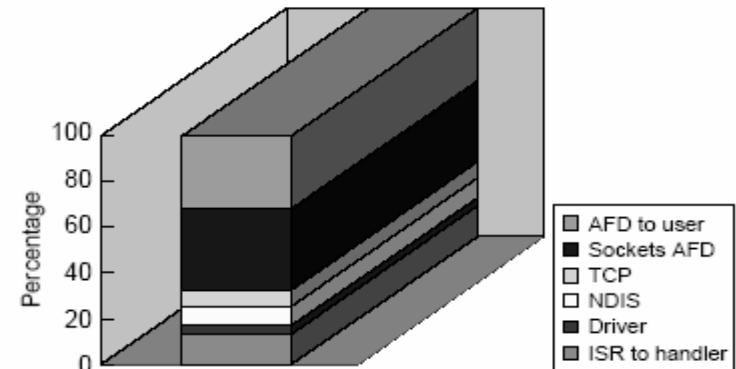
- Initiated by Intel, Compaq and Microsoft in 1996.
- Open, independent of OS, CPU and network media.
- Provides portability, compatibility.
- Employs user-level messaging to:
 - provide direct access to network interface (minimizing number of instructions to execute)
 - avoid intermediate copies
 - bypass OS virtualization/protection
 - avoid interrupts
- Provides protected, zero-copy user-level access.

Software overhead

- Software overhead comes from:
 - Many layers of software
 - Multiple data copy
 - Multiple context switches
 - Interrupt handling



(a)



(b)

Figure taken from [Dunning '98]

Sending (a) and receiving (b) overhead in two Pentium Pro servers.

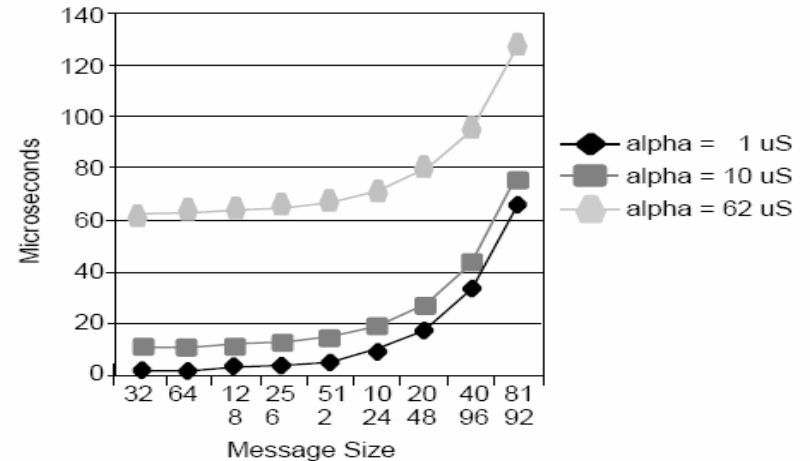
Software overhead -2

- And causes:
 - High latency
 - Low bandwidth

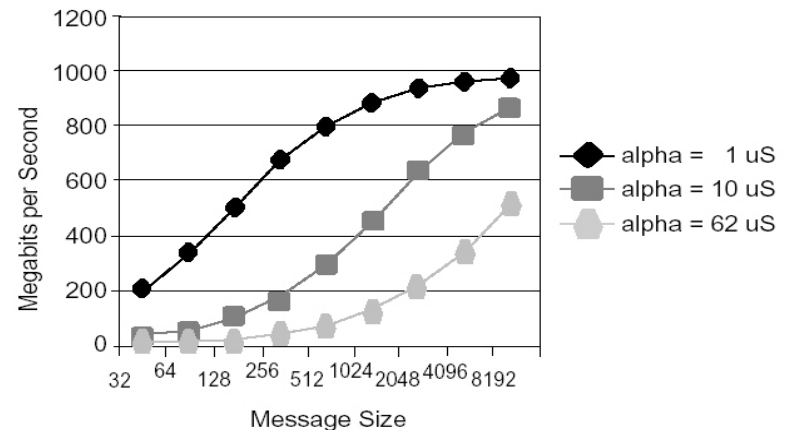
α is the fixed, non-overlapped transfer time per message (not including wire time) dominated by software overhead. It is like t_0 in $t(m) = t_0 + m/r_\infty$.

Note that even each graph assumes a fixed α for messages of different sizes and ignores the effect of data-copy of messages with different payloads. Including that will result in higher latency and lower bandwidth and strengthening the argument.

Figures taken from [Cameron '02]



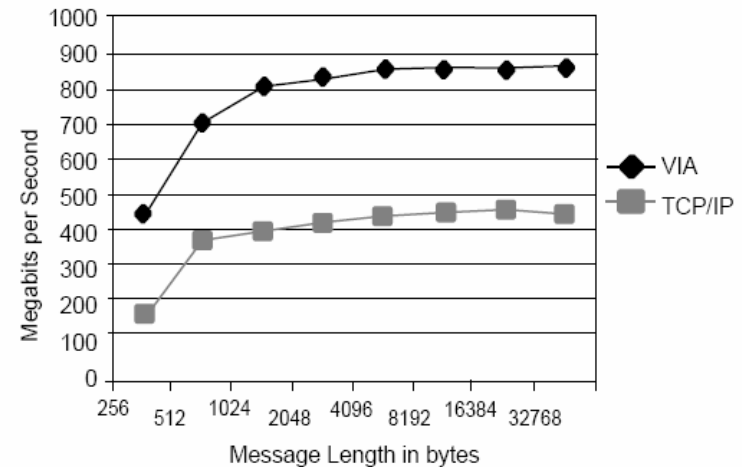
Effect of Software Overhead on Latency



Effect of Software Overhead on Bandwidth

Software overhead -3

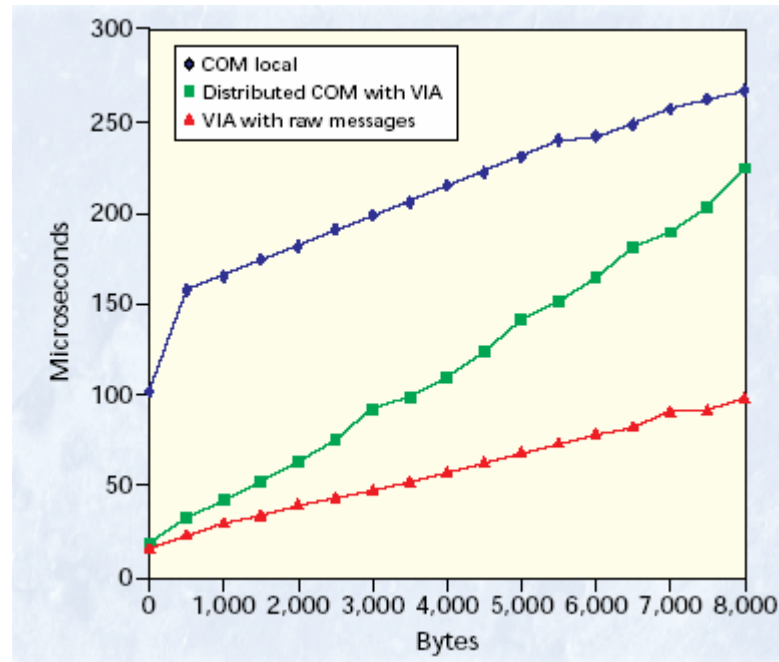
- Effect of traffic pattern
 - 80/80 rule of network distribution:
 - 80% of messages have less than 256 bytes
 - 80% of total data delivered by messages of greater than 8192 bytes
 - For example a study of a network running NFS traffic at UCSD shows this bi-model behavior: 86% less than 200, 9% more than 8192
 - → Software overhead is more predominant in these small messages.
 - Can't achieve half of the bandwidth even for large messages.
 - Let's say we want to match overhead time and wire time on a 1Gbps network: $t(m) = t_0 + m/r_\infty$
 - A 200 bytes message 1.6 us
 - If running on a 200MHz CPU and assuming 1 instr./cycle, can only execute 320 instructions.
 - Even if running on a 2GHz CPU and assuming 0.5 instr./cycle, can only execute 1600 instructions.
 - Can't have such a low overhead with all the software layers.
 - How about an interrupt or context switch?



Measured One-way Bandwidth

Two directly connected PIII 800 MHz,
with GigaNet cLAN vs. Gigabit Ethernet
Figure taken from [Cameron '02]

Software overhead -4



Round-trip times for a local remote procedure call (LRPC) under Windows NT (COM local) versus a remote procedure call over the Virtual Interface Architecture using GigaNet's GNN1000 interface (Distributed COM with VIA). By bypassing the kernel the remote call is actually faster than the local one. The round-trip latency of raw messages over the VIA (VIA with raw messages) shows that there is room for improvement in the DCOM protocol.

Figure taken from [von Eaiken '98]

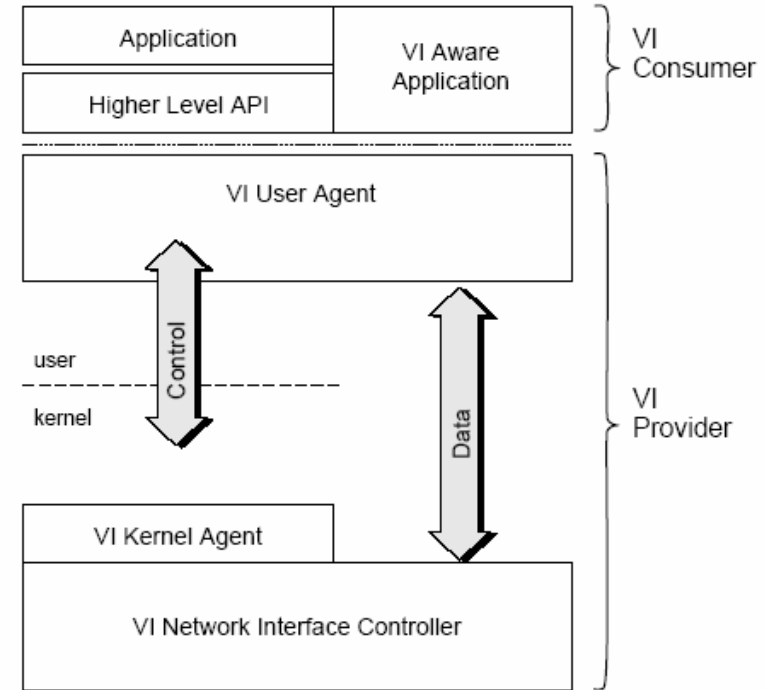


VIA concepts

- VIA itself virtualizes the NIC and provides protection rather than OS.
- Employs a *message-passing* model rather than *shared-memory* model.
- Connection oriented. Endpoints called VI (Virtual Interface)
 - Asynchronous queuing structure
- Point-to-point communication. No multicasting.

VIA concepts -2

- Bypassing OS issues:
 - Virtualization?
 - Protection?!
- Functional partitioning:
 - VI Consumer
 - Manages its own buffer requirement → optimization
 - Capable of directly notifying NIC
 - VI Provider (includes NIC too)
 - Manages protection, virtualization
 - Virtual to physical address translation
 - Synchronization of completed work
 - Reliable transport service
- Separation of control and data paths
 - Control path:
 - Connection setup
 - Memory registration
 - Address translation
 - Data path
 - Send/Receive
 - RDMA Read/Write



VIA Model for OS Bypass

Figure taken from [Cameron '02]



VIA concepts -3

- Some terms:

- VIA-NIC

- Network Interface Controller complying with VIA Spec.
- Replaces HW multiplexing, and transfer scheduling traditionally done by OS.
- Might provide a reliability protocol.

- VI Provider Layer (VIPL)

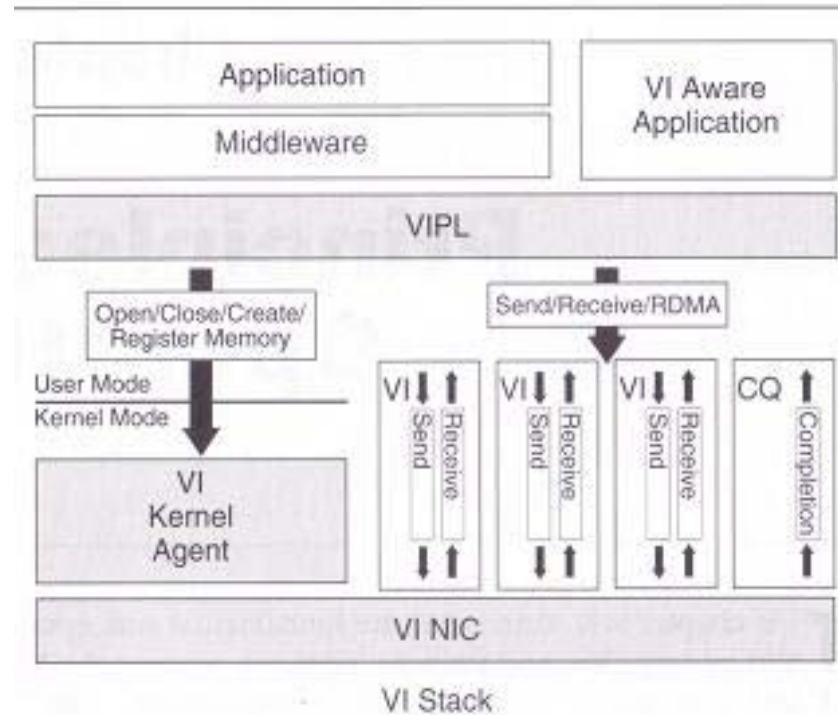
- Set of functions and data structures
- Provides the software interface for user-level processes.
- VIA-aware applications can use it directly; others rely on higher level abstraction.
- “VIPL API” provided in VIA Spec. as an *example VIPL*
 - It’s become the *de-facto* API standard rather than *de-jour*.

- VI Kernel Agent:

- A software residing in OS (device-driver).
- Minimal. Not involved in most portion of data transfer path.

VIA concepts -4

- VI instance:
 - It's a process interface to network.
 - Owned by a single process.
 - A process can own multiple of them.
 - Two Work Queues; one Send and one Receive. A queue is a linked list of descriptors.
 - Completion Queue (CQ) is for completion notification. A Send/Receive queue can be associated with a CQ of other VI's in the same process.



The VI Stack

Figure taken from [Cameron '02]

VIA concepts -5

- Data movement operations:
 - Send/Receive
 - Source/destination buffers, number of bytes to transfer.
 - Buffers don't need to be contiguous → scatter/gather list.
 - Helps a lot with layered-protocol header/trailer manipulation.
 - Data can be gathered from dis-contiguous buffers at source.
 - Data can be scattered to dis-contiguous buffers at destination.
 - RDMA-Write
 - Sender *pushes* data from memory of sending app to a *specific* location in the receiving app (destination add/handle communicated before).
 - Data can be gathered at the source but not scattered.
 - RDMA-Read
 - Destination *pulls* data from a *specific* location of sending app to location in the receiving app (so source address is communicated before).
 - Data can be scattered at the destination but not gathered.

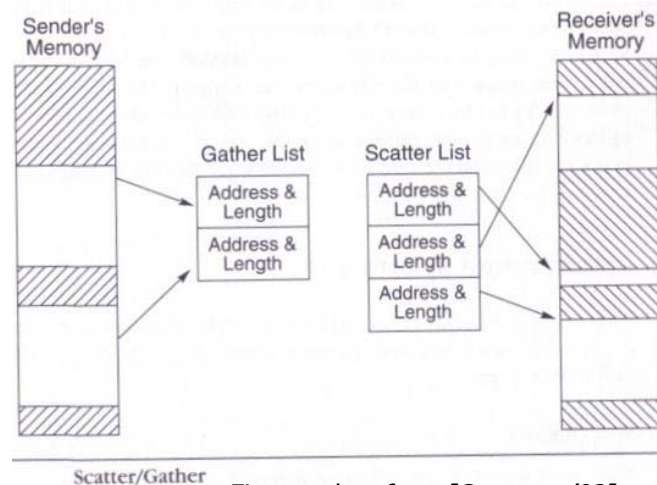


Figure taken from [Cameron '02]

- Receive Assumption: A buffer must be already posted before receiving data. This to maintain zero-copy at receive; skipping intermediate buffers.

VIA concepts -6

■ Descriptors

- Specifies a data movement operation.
- Resides in host memory; accessed by both application and VI-NIC.
- Each descriptor has one control segment, variable no. data segments. RDMA descriptors have an address segment too.
- "Next descriptor address" field points to the next descriptor in queue.

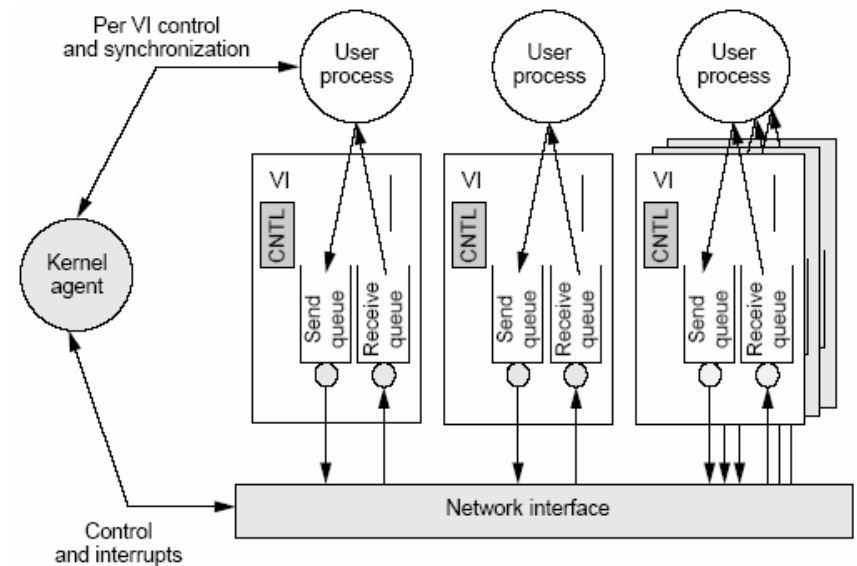
Send and Receive descriptor format					
15:14	13:12	8:11	7:0		Byte offset
control	segment count	Memory handle	Next descriptor address		Control Segment
Status		Total length	Immediate data	Reserved	
Segment length		Memory handle	Buffer virtual address		Data Segment

RDMA descriptor format					
15:14	13:12	8:11	7:0		Byte offset
control	segment count	Memory handle	Next descriptor address		Control Segment
Status		Total length	Immediate data	Reserved	
Segment length		Remote memory handle	Remote buffer virtual address		Address Segment
Segment length		Memory handle	Buffer virtual address		Data Segment

VIA concepts -7

Work Queues

- Each VI consists of two work queues: a Send Queue and a Receive Queue; maintained as FIFO.
- Send, RDMA-Write and RDMA-Read operations placed in the Send queue; Receive command in the Receive Queue.
- VipPostSend() and VipPostRecv()
- When a new operation submitted, it'll become the tail of queue.
- Link field of the last descriptor is NULL.



VI queues.

Figure taken from [Dunning '98]

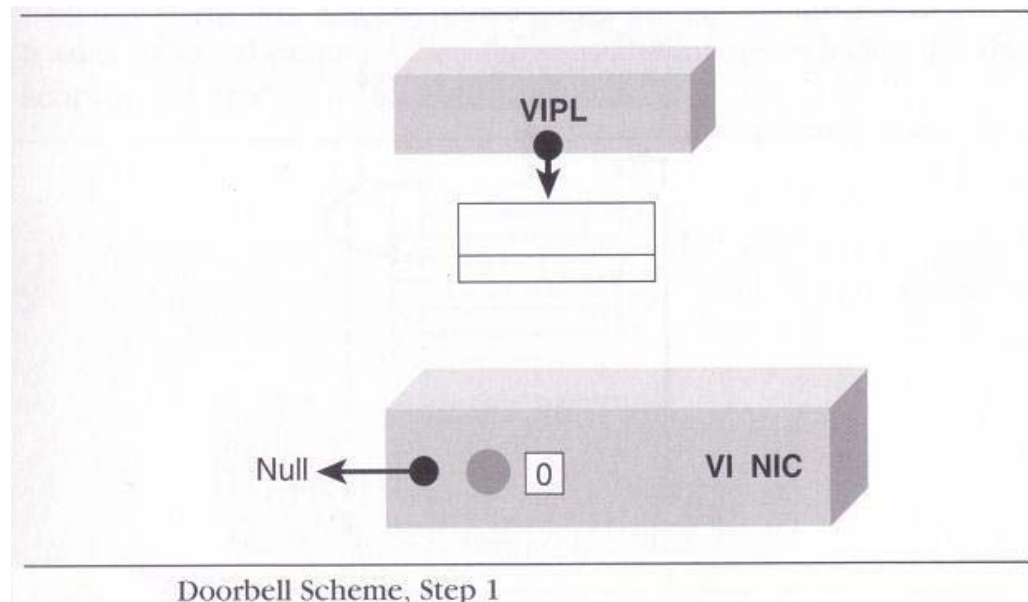
VIA concepts -8

■ Doorbell

- App notifies VI-NIC about a new transfer (after adding it to the queue) by ringing the “doorbell”.
- Doorbell is per queue; potentially a VI-NIC register, mapped to user-level address space.
- A “token” is passed at ringing time; identifying the descriptor at the head of queue. Its format is implementation dependent; not described by Spec.
- VI-NIC only keep tracks of the head descriptor and number of descriptors in each queue, not all content of the queue.

Step 1: VIPL has added a descriptor to a queue but not yet notified the VI-NIC.

Figure taken from [Cameron '02]

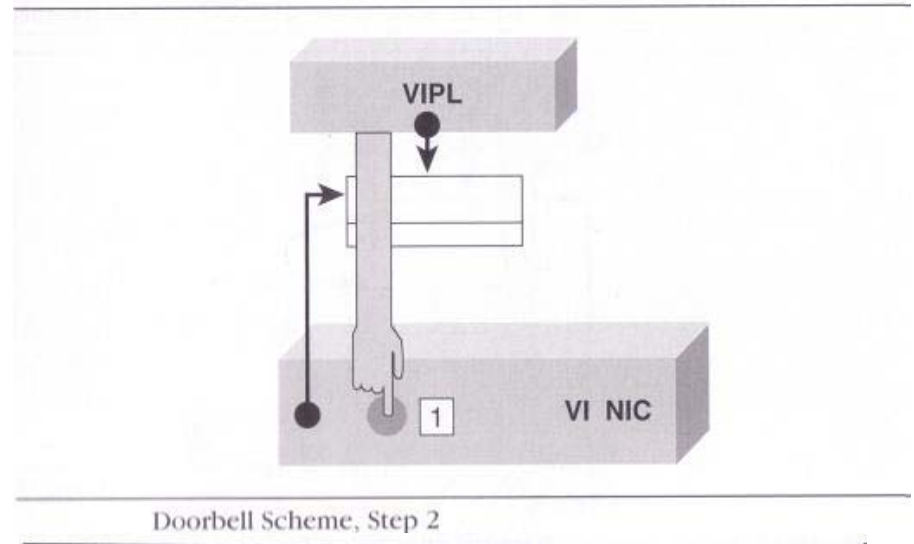


VIA concepts -9

Doorbell...

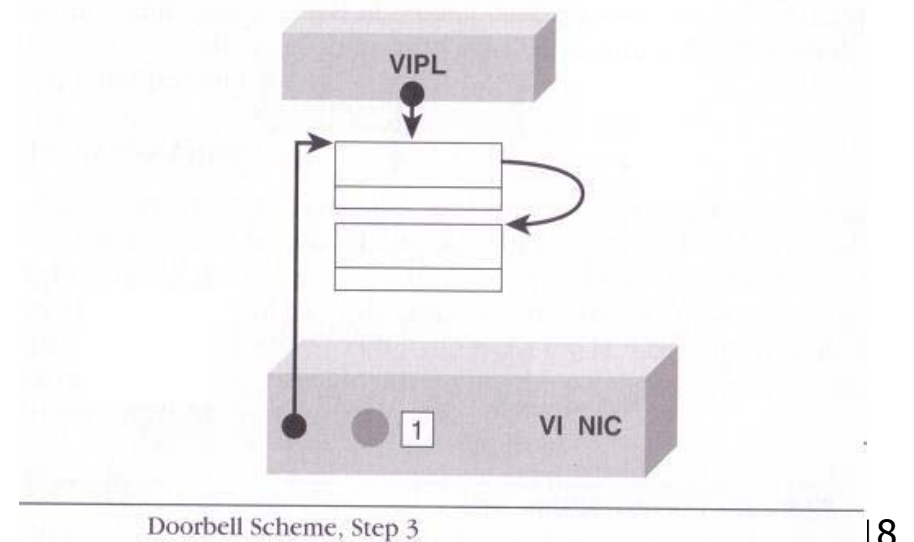
Step 2: VIPL rings the doorbell associated to the work queue. NIC increments the count of outstanding descriptors and keeps pointer to head of queue.

Figure taken from [Cameron '02]



Step 3: VIPL links another descriptor on the work queue. NIC not notified yet.

Figure taken from [Cameron '02]

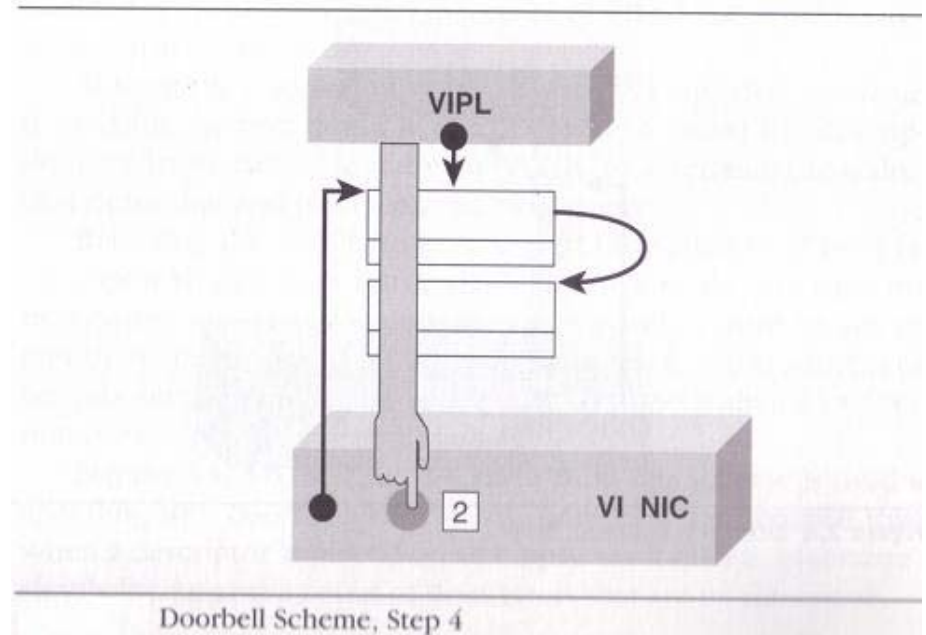


VIA concepts -10

■ Doorbell...

Step 4: VIPL rings the doorbell again. NIC increments the count of outstanding descriptors. But head is not changed.

Figure taken from [Cameron '02]



VIA concepts -11

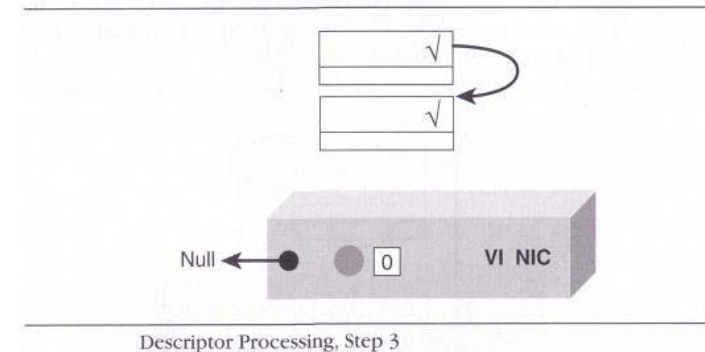
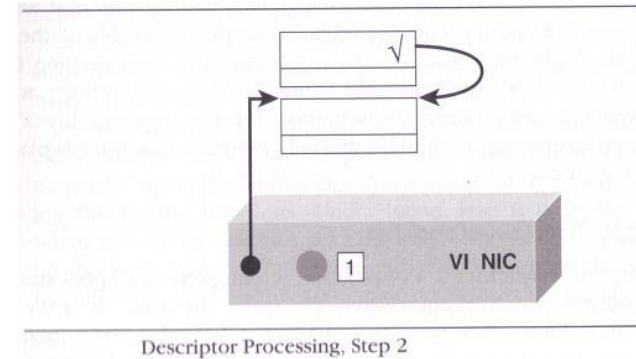
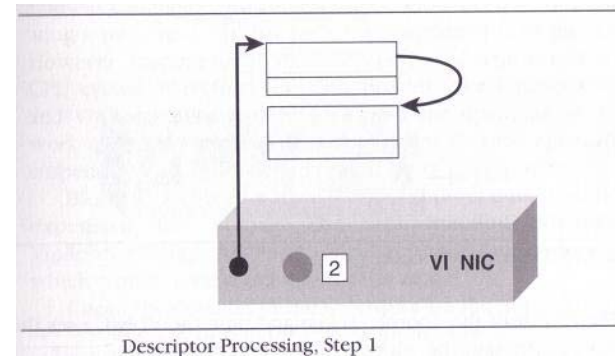
Consuming descriptors from work queue by VI-NIC:

Step 1: NIC has two outstanding descriptors for this work queue. Using its pointer to queue head, locates and processes the operation.

Step 2: NIC has processed the first descriptor and has set the completion status. The count of outstanding descriptors decremented. NIC follows the link of completed descriptor to find the next descriptor.

Step 3: NIC has processed the last descriptor. The count is decremented to 0 and the saved pointer set to NULL as there is no more outstanding descriptor.

Figures taken from [Cameron '02]





VIA concepts -12

■ Synchronization/Completion

- After completing an operation, NIC turns on the “done bit” of status field of the corresponding descriptor.
- Application figures out about completion of an operation by:
 - Polling:
 - VipSendDone(), VipReceiveDone() queries the done bit of the descriptor.
 - As descriptor is in user space, it doesn't require a system call. Very fast.
 - Blocking wait:
 - Polling is not good for operations taking a long time to complete.
 - VipSendWait(), VipReceiveWait() block till head of the work queue is completed.
 - It's much more expensive than polling; requires one or two system calls; and might require an interrupt for resumption.

VIA concepts -13

■ Completion Queues

- Let's say an app has several work queues/VIs and wants to wait for next completion on any of them ...
- Instead of having a thread for each queue to wait or poll, completion on all the queues can be aggregated through a single completion queue.
- A completion queue belongs to a single process though multiple VIs may use one completion queue.
- Completion queue can be polled or waited on by `VipCqDone()` and `ViCqWait()`.
- They return the VI handle and the work queue (Send or Receive) that the completion happened on.



Figure taken from Intel website



VIA concepts -14

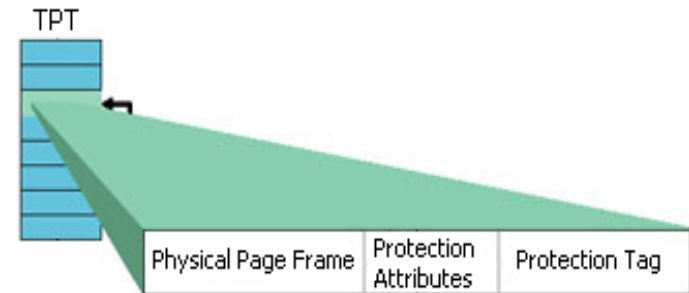
■ Protection

- VI-NIC must ensure that a process is not writing to or reading from other's memory locations. A *protection-tag* is used.
- Protection-tag:
 - An opaque data type containing a unique value
 - Assigned by the kernel agent
 - Created by `VipCreatePtag()`
 - Each VI is associated with a single protection tag.
 - An app might use different or one protection tag(s) for its multiple VIs.
 - Only used with VIPL calls crossing the kernel. VI Kernel Agent reject calls using incorrect protection tags.
 - Some of other APIs using protection tags:
 - `VipCreateVI()`, `VipRegisterMemory()`

VIA concepts -15

■ Memory registration

- VIA allows application to use virtual addresses when referencing memory locations.
- But VI-NIC needs to access the memory with physical addresses → translation required.
- Also, VI-NIC is the body to enforce the memory protection among applications/OS.
- Mechanism is called Translation and Protection Table (TPT). Its implementation is beyond the spec.; only a reference implementation provided.
- Protection attributes:
 - By default all memory has Read access.
 - Write Access: The memory can be destination of a Send operation.
 - RDMA-Read access: The memory may be the source of a RDMA-Read.
 - RDMA-Write access: The memory may be the destination of a RDMA-Write.
- The sequence (`VipRegisterMemory()`):
 - A memory region being registered may span multiple pages but must be contiguous.
 - Kernel Agent locks (pins down/wires) the memory to prevent it being paged out.
 - Kernel Agent allocates one TPT entry for each page in memory region and sets its fields.
 - Kernel Agent creates and returns a memory handle associated with the region.
 - *Later, TPT entries can be retrieved using the handle.*



Translation and Protection Table Format

Figure reproduced based on [Cameron '02]

VIA concepts -16

Memory registration...

- Memory Handle = (Virtual Address >> 12) – TPT Index
- TPT Index = (Virtual Address >> 12) – Memory Handle
- Drawbacks. This sample method can not detect:
 - A virtual address can be paired with a wrong memory handle.
 - A reference made to a location beyond the registered memory.

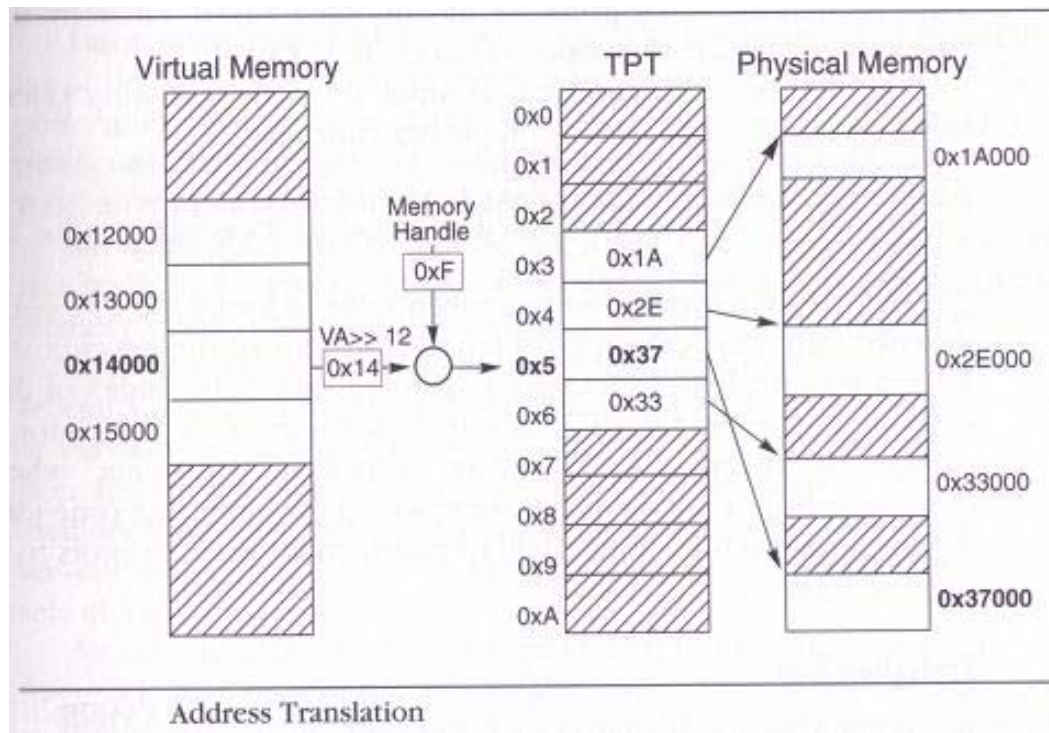


Figure taken from [Cameron '02]

VIA concepts -17

■ Reliability levels

- VIA designed with 3 types of network in mind:
 - Inherently unreliable networks; like Ethernet where packets can be dropped at congestion and re-ordered.
 - Reliable networks, like Myrinet; no packet dropped at congestion; order preserved.
 - Reliable networks with end-to-end acknowledgement protocol; like ServerNet and InfiniBand; may optionally support retry on error.

	Unreliable	Reliable Delivery	Reliable Reception
Detecting corrupt data	Yes	Yes	Yes
Data delivered at most once	Yes	Yes	Yes
Data delivered exactly once	No	Yes	Yes
Data order guaranteed	No	Yes	Yes
Data loss detected	No	Yes	Yes
Connection broken on error	No	Yes	Yes
RDMA-Read supported	No	Optional	Optional
RDMA-Write supported	Yes	Yes	Yes
Send/RDMA-Write descriptor marked done	When data leaves initiator	When data leaves initiator	When data arrives at target



VIA concepts -18

■ Other APIs:

- **Opening the VI NIC:** App must call `VipOpenNic()` and use the handle for memory registration, VI and CQ creation.
- **Creating VIs:** App calls `VipCreateVi()/VipDestroyVi()`. VIPL asks Kernel Agent for allocating a new VI. Send and Receive queues created implicitly.
- **Creating Completion queues:** `VipCreateCq()`. It implicitly allocates memory and registers it. Then any number of work queues of VIs created later can point to this CQ.
- **Making connections:**
 - Network address has two parts: A unique host address, and an endpoint discriminator. The format is implementation specific; e.g. ["Node 1", "foo server"]
 - Client/Server: Asymmetric. Server waits first for a connection request to arrive, then accept or rejects it. Client simply request connection. `VipConnectWait()`, `VipConnectAccept()`, `VipConnectReject()`, `VipConnectRequest()`.
 - Peer-to-peer: Symmetric. Each peer first issues a connect request, then blocks or polls on the status. `VipConnectPeerRequest()`, `VipConnectPeerDone()` or `VipConnecPeerWait()`.

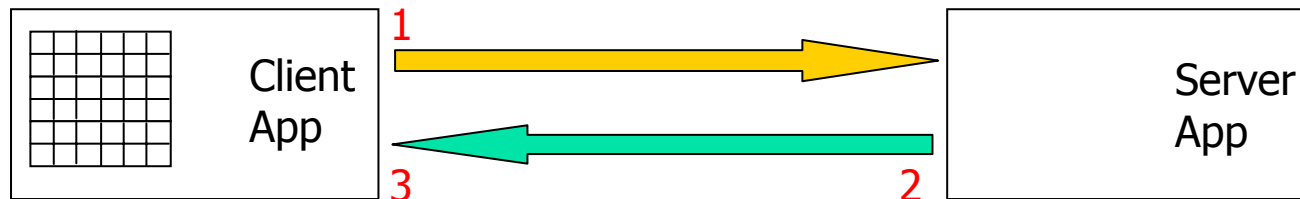
■ Major VIPL APIs:

Hardware Connection	Memory Protection and Registration	Data Transfer
VipOpenNic () VipCloseNic ()	VipCreatePtag () VipDestroyPtag () VipRegisterMem () VipDeregisterMem ()	VipPostSend () VipSendDone () VipSendWait () VipSendNotify () VipPostRecv () VipRecvDone () VipRecvWait () VipRecvNotify ()
Endpoint Creation and Destruction	Querying	Completion Queues
VipCreateVi () VipDestroyVi ()	VipQueryNic () VipQueryVi () VipSetViAttributes () VipSetMemAttributes () VipQueryMem () VipQuerySystemManagementInfo ()	VipCreateCQ () VipDestroyCQ () VipResizeCQ () VipCQDone () VipCQWait () VipCQNotify ()
Connection Management	Error	
VipConnectWait () VipConnectAccept () VipConnectReject () VipConnectRequest () VipDisconnect ()	VipErrorCallback ()	

Figure taken from [Kim '01]

A VIA sample

- The sample application is a simple crossword puzzle solver (from [Cameron '02]).
- Client program issues a request (with a clue) and server replies with the answer.
- The puzzle is an NxM array of characters in the client's address space. Replies written directly into this array using RDMA-Write.
- *Across* answers are allowed only to have a whole answer contiguous in memory.
- Data exchanges:
 - 1. Client transmits a clue (just an integer) to the server using Send operation. Starting address for the answer in array is also sent along.
 - 2. The server issues a RDMA-Write of the answer.
 - 3. The receive completion on the client indicates the answer has been written to the client memory.





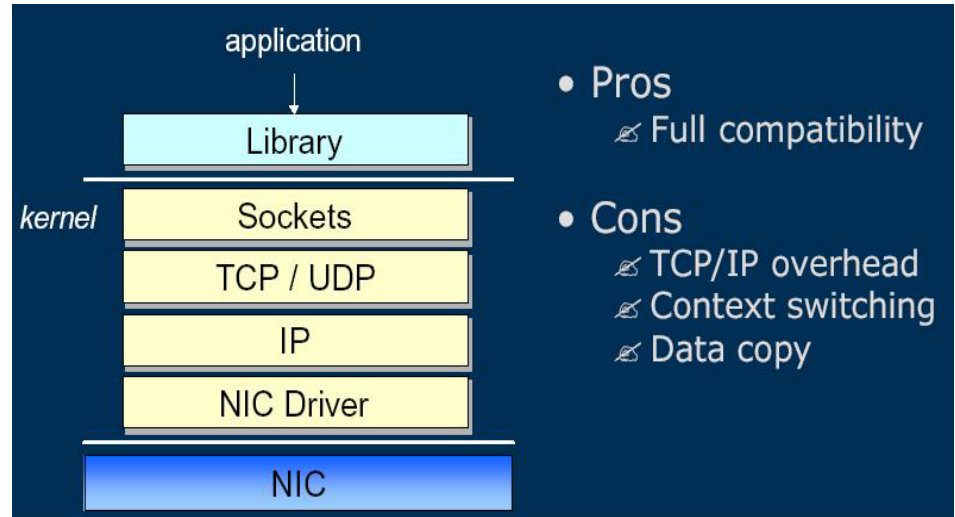
Design alternatives

- VIA spec. is flexible. Different choices exist for implementing various components such as doorbells, address translation, CQ, ... (see [Banikazemi '00]).
 - When a descriptor posted, NIC needs to access its content (which is in host memory). Transfer of descriptor can be:
 - Host-initiated through PIO.
 - NIC-initiated through DMA.
 - By caching the descriptor in NIC memory, its transfer time will not increase latency of receive messages.
 - Software doorbells:
 - Keeping doorbell in NIC memory. NIC polls all doorbells all the time. Expensive for NIC CPU.
 - Keeping a centralized queue of operation in NIC. As all VIs share this queue, need synchronized access managed by kernel. Transition to kernel required for each posting. Didn't find it very expensive (2.27 us).
- Peak bandwidth of 101.4 MBytes/s and latency of 18.2 us over IBM Netfinity NT Cluster.

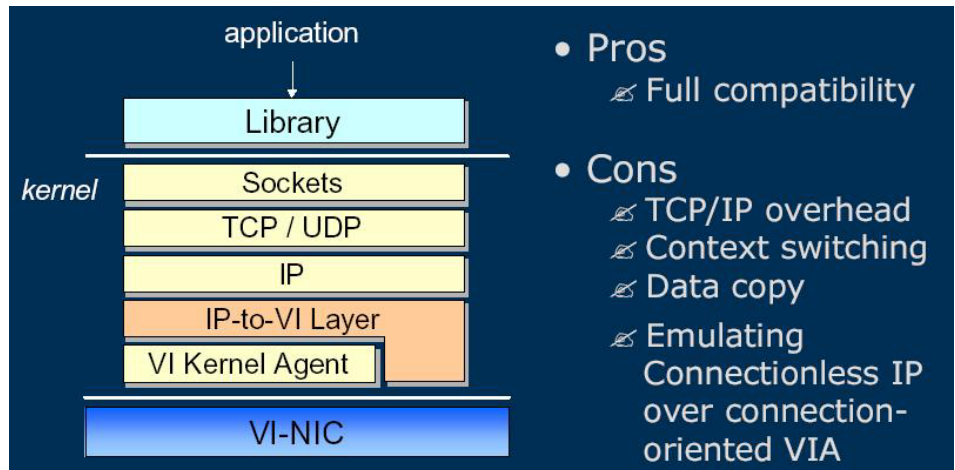
Design alternatives -2

- SOVIA (see [Kim '01])

- Traditional Berkeley Sockets



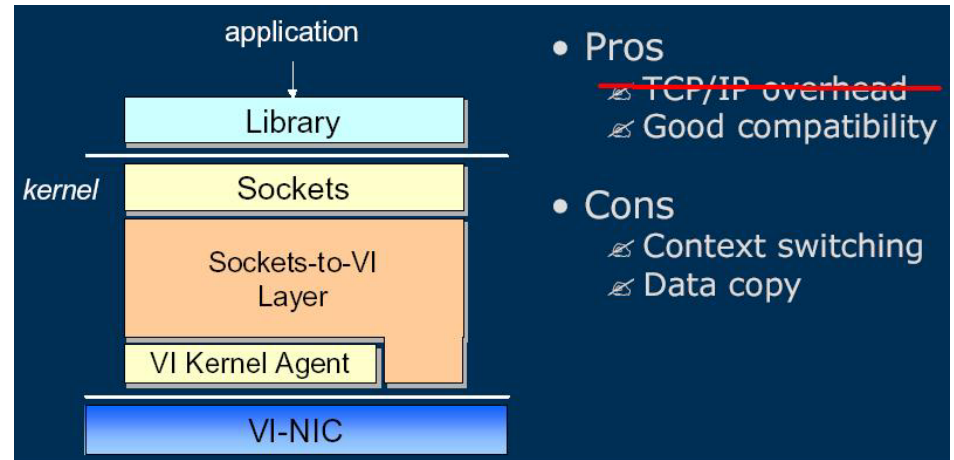
- Using IP to VI layer



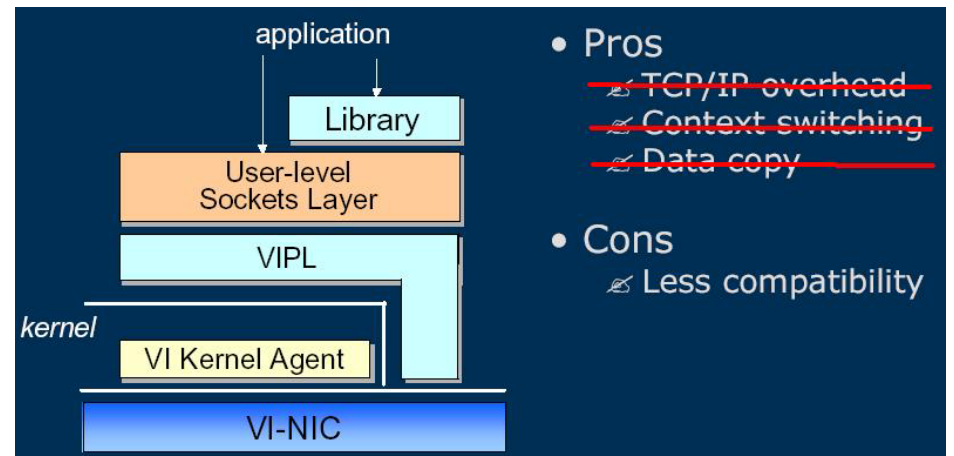
Design alternatives -3

■ SOVIA...

- Sockets to VI layer



- User-level Sockets layer



Figures taken from [Kim '01]



M-VIA (from [M-VIA '02])

- M-VIA (Modular-VIA) is a complete high-performance implementation of the Virtual Interface Architecture for Linux.
- It was written at the [NERSC](#) center.
- It consists of a loadable kernel module and a user level library, and also requires a modified device driver (also a loadable kernel module).
- In the case of VIA over loopback, M-VIA uses a separate loopback device driver since there is no need for it to interact with the normal loopback device.
- M-VIA is designed to be easily portable to new network devices. If an appropriate device class already exists for a network card, only a few changes to the device driver are needed.
- M-VIA coexists with traditional networking protocols, so you can run TCP/IP on the same physical network.
- M-VIA passes the strict Intel conformance tests, with only minor exceptions

M-VIA -2

NICs supported:

NIC	2.2.X kernel	2.4.X kernel
DEC DC21x4x (Tulip) based Fast Ethernet cards	Yes	Yes
Intel Pro/100 Fast Ethernet cards	Yes	Yes
3Com "Boomerang"-class Fast Ethernet cards	Buggy	Yes
SysKonnnect SK-98xx Gigabit Ethernet cards	Yes	Yes
Intel Pro/1000 Gigabit Ethernet cards	Only older NICs	No
Packet Engines GNIC-II (Hamachi) Gigabit Ethernet cards	Yes	Yes
Packet Engines GNIC-I (Yellowfin) Gigabit Ethernet cards	No longer supp'ed	No

If you do not have one of the NICs listed above, you can use the the M-VIA loopback device within a single node

Performance:

Network	Protocol	Latency (us)	Bandwidth (MB/s)
Packet Engines GNIC II	TCP	59	31
Packet Engines GNIC II	M-VIA	19	60
Tulip Fast Ethetnet	TCP	65	11.4
Tulip Fast Ethetnet	M-VIA	23	11.9

■ Design:

- **Kernel Agent:** Performs privileged operations on behalf of VIPL and assists M-VIA Device Drivers with operations requiring operating system support.
 - **Connection Manager:** Establishes logical point-to-point connections between VIs.
 - **Protection Tag Manager:** Allocates, deallocates, and validates memory protection tags (Ptags).
 - **Registered Memory Manager:** Handles the registration of user communication buffers and descriptor buffers.
 - **Error Queue Manager:** Provides a mechanism for posting asynchronous errors by VIA devices and blocking.
 - **Kernel Extensions:** Provides functionality required for efficient implementation.
- **M-VIA Device Drivers:** Provides abstraction of VI NIC and the ability to override all of the default functionality of the Kernel Agent layer should allow any natively supported or software emulated VIA device to be supported by M-VIA.
- **Device Classes:** For logically grouping commodity network interfaces into common categories such as Ethernet, ATM, FDDI, etc. and meant for rapid development through code reuse.
- **VI Provider Library:** M-VIA contains a single VI Provider Library, which is interoperable with native hardware and software VIA devices developed within the Modular VIA framework.

■ True zero-copy?

- Lacking hardware support, is only zero-copy (no memory-to-memory copy in host) on sends; requires one copy on receives. The copy is unavoidable for without special hardware support.
- On x86, uses fast trap on the sender to avoid the overhead of a system call. This fast trap achieves transition to the kernel's privilege level without most of the overhead of a system call (does not yield the CPU). On the receiver, data is received directly with a low-level hardware interrupt that does all the processing necessary.



Comparing with InfiniBand Architecture

- A comprehensive spec. including electrical and mechanical configuration of IBA physical media, format of packets, sets of operation, semantics and management interface.
- Includes many VIA concepts; like RDMA, user-level access to HW, work queue model.
- From the HW perspective IBA is much more complete; its SW doesn't go as far as VIA.
- Doesn't include an API. Defines a set of *verbs* instead providing abstract description of an IBA NIC. Doesn't specify calling sequence, data types.
- IBA provides more operations: *Atomic Compare & Swap* and *Fetch & Add*.
- Extends memory protection model. Granularity of memory registration is at byte level rather than page level. Also *Memory Windows...*
- A work request is IBA's equivalent of VIA's work queue.
- More transport services: Reliable/unreliable/raw Datagram.



Conclusions & Future of VIA

- Largely achieved its goals
 - HW vendors developed VI-compliant products.
 - Commercial applications developed using VI directly, like DB2 EEE, Oracle Database, MS SQL (native VIA support for application server to database-server communication or among database servers; over 33% improvement comparing to TCP/IP. See [Bialek '01]).
 - Legacy message passing interfaces implemented on top of VI, like MPI (MVICH) and Sockets.
- Improvements can be done:
 - Increasing limited number of VIs (limited to 64000).
 - Connection orientation: Requires managing more VIs.
 - Memory registration:
 - Hard to implement a legacy message-passing interface on top of VIA. How to specify up front what memory will be used for send or receive?
 - Many pages can be locked into physical memory. Performance penalty...
 - Receive assumption:
 - Unfamiliar for SW developers. Socket allows send before a posted receive.
 - Credit-based flow control to hold messages at sending node till receive is posted on the other side



Conclusions & Future of VIA -2

- Improvements can be done...
 - Insufficient standardization: VIPL should have been defined as part of the standard rather than only “an example”.
 - More detailed spec. for HW interfaces (like doorbell) would have made writing portable VIPL library easier across HW implementations.
- Future:
 - VI Developer’s Forum (VIDF) took on the task of evolving VI architecture and standardizing VIPL API.
 - Version 1.1 of VIPL is defined, including extensions requested by database vendors. It had to be published as “VIDF Extensions to VIPL-1.0” because of disagreement between 3 original promoters of VIA.
 - Direct Access Transport Collaborative (DATC) is defining a new API that is not derived from VIA spec., adding kernel API that allows device drivers and OS access VI capabilities (like zero-copy and RDMA). Will also develop OS/HW independent user-level APIs.



References

- [Cameron '02] Don Cameron, Greg Regnier, "*The Virtual Interface Architecture*," Intel Press, 2002.
- [Dunning '98] Dave Dunning, Greg Regnier, Gary McAlpine, D. Cameron, B. Shubert, F. Berry, A. Merritt, E. Gronke, C. Dodd, D. Cameron, G. Regnier, "*The Virtual Interface Architecture*," IEEE Micro, Volume 18 Issue: 2, March-April 1998, pp: 66–76.
- [von Eicken '98] Thorsten von Eicken, Werner Vogels, "Evolution of the Virtual Interface Architecture," IEEE Computer, Volume 31 Issue: 11, November 1998, pp: 61–68.
- [Bialek '01] Boris C. Bialek, "*Virtual Interface Architecture and Microsoft SQL Server 2000*," Dell Corp. White Paper, January 2001.
- [Banikazemi '00] Banikazemi, M.; Moorthy, V.; Herger, L.; Panda, D.K.; Abali, B., "Efficient Virtual Interface architecture (VIA) support for the IBM SP switch-connected NT clusters", in *Proceedings of 14th International Parallel and Distributed Processing Symposium, 2000. IPDPS 2000*, 1-5 May 2000, pp.: 33 -42.
- [Kim '01] Jin-Soo Kim, et al., "SOVIA: A User-level Sockets Layer Over Virtual Interface Architecture," in *Proceedings of the 2001 IEEE International Conference on Cluster Computing (CLUSTER.01)*.
- [M-VIA '02] *M-VIA version 1.2 documentations*, NERSC, Sep. 2002.