SOVIA:
A User-level Sockets Layer Over Virtual Interface Architecture

Jin-Soo Kim

jinsoo@computer.org

Computer System Research Department
Electronics and Telecommunications Research Institute (ETRI)
Korea

Outline

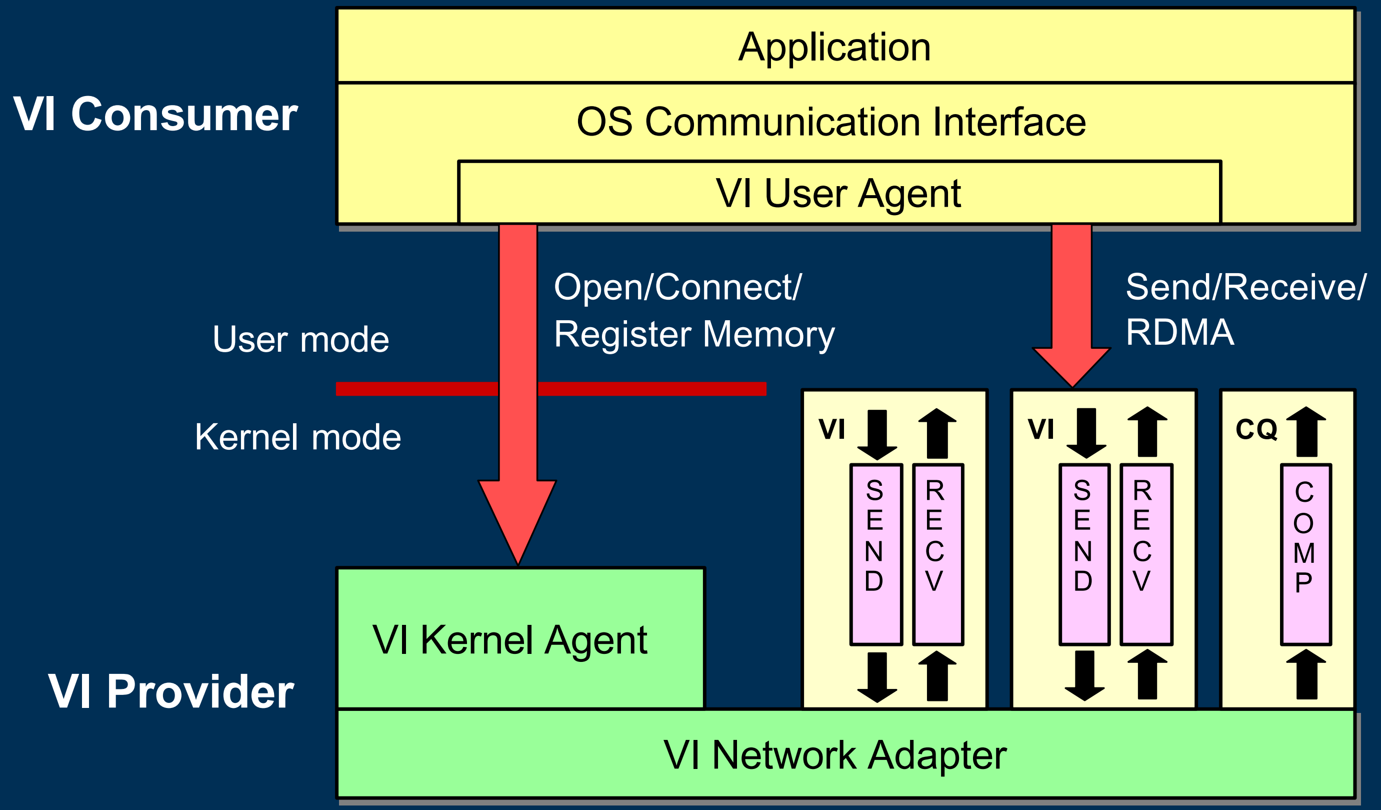
- Introduction
- Performance Issues
- Compatibility Issues
- Results on Real Applications
- Concluding Remarks

Introduction

Virtual Interface Architecture

- **Overheads in traditional communication architecture**
  - Protocol overhead in TCP/IP
  - Data copying
  - Context switching

- **Virtual Interface Architecture**
  - Led by Compaq, Intel, and Microsoft
  - A standard software interface for user-level access to a network hardware


Application
OS Communication Interface
VI User Agent
VI Consumer
Open/Connect/
Register Memory
Send/Receive/
RDMA
User mode
Kernel mode
VI
SEND
RECV
VI
SEND
RECV
CQ
COMP
VI Kernel Agent
VI Provider
VI Network Adapter

VI Provider Library (VIPL)

Hardware Connection

VipOpenNic ()
VipCloseNic ()

Endpoint Creation and Destruction

VipCreateVi ()
VipDestoryVi ()

Connection Management

VipConnectWait ()
VipConnectAccept ()
VipConnectReject ()
VipConnectRequest ()
VipDisconnect ()

Memory Protection and Registration

VipCreatePtag ()
VipDestroyPtag ()
VipRegisterMem ()
VipDeregisterMem ()

Querying

VipQueryNic ()
VipQueryVi()
VipSetViAttributes ()
VipSetMemAttributes ()
VipQueryMem ()
VipQuerySystemManagementInfo ()

Error

VipErrorCallback ()

Data Transfer

VipPostSend ()
VipSendDone()
VipSendWait ()
VipSendNotify ()
VipPostRecv ()
VipRecvDone ()
VipRecvWait ()
VipRecvNotify ()

Completion Queues

VipCreateCQ ()
VipDestroyCQ ()
VipResizeCQ ()
VipCQDone ()
VipCQWait ()
VipCQNotify ()

Motivation

Problem in VIPL

- "VIA is considered by many systems designers to be too low a level for application programming."

-- TFCC Cluster Computing White Paper

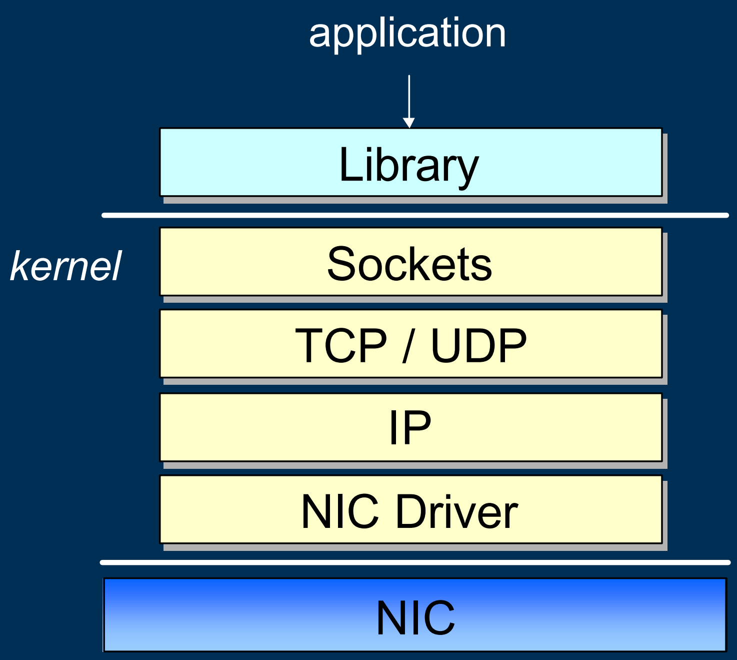
Requirements for a new layer

- High Performance
- Portability

➡ **SOVIA (Sockets Over VIA)**

Berkeley Sockets API

✍ Traditional architecture



application

Library

kernel

Sockets

TCP / UDP

IP

NIC Driver

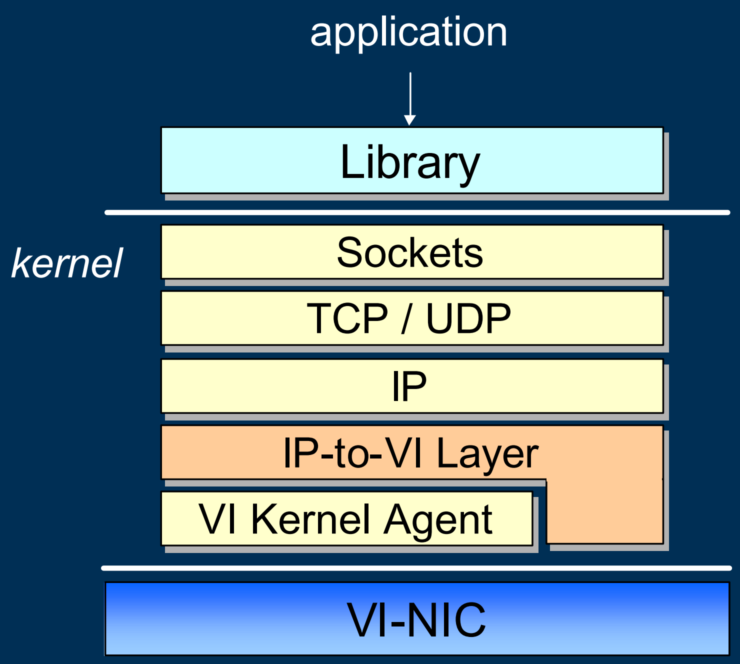
NIC

- Pros
  - ✍ Full compatibility
- Cons
  - ✍ TCP/IP overhead
  - ✍ Context switching
  - ✍ Data copy

Sockets over VIA (1)

✍ Using an IP-to-VI Layer

- Giganet's LANE (LAN Emulation) Driver

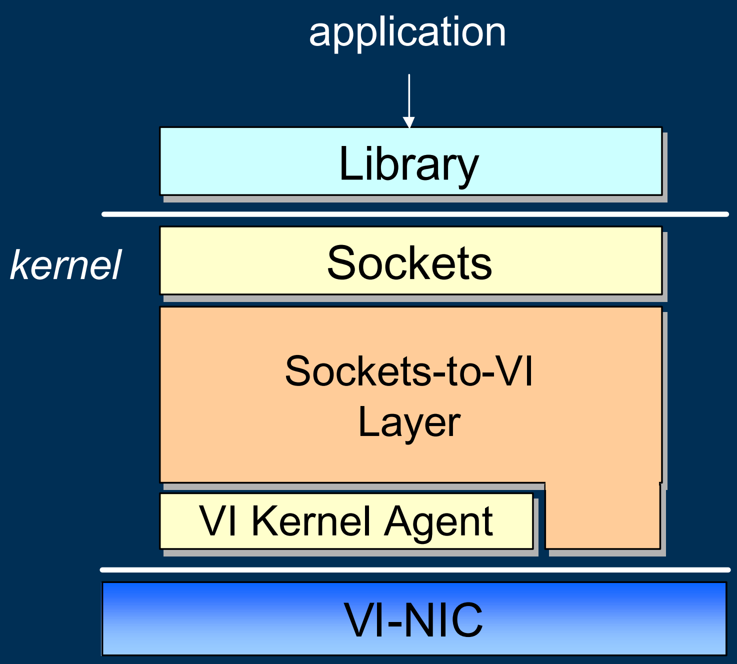


- Pros
  - ✍ Full compatibility
- Cons
  - ✍ TCP/IP overhead
  - ✍ Context switching
  - ✍ Data copy
  - ✍ Emulating Connectionless IP over connection-oriented VIA

Sockets over VIA (2)

Using a Sockets-to-VI Layer

- VIsocket on Solaris

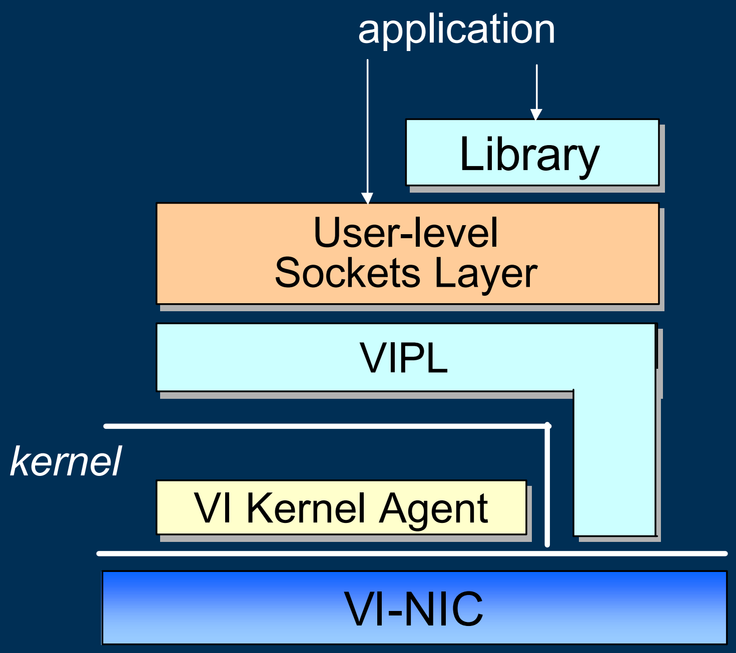


- Pros
  - ~~TCP/IP overhead~~
  - Good compatibility
- Cons
  - Context switching
  - Data copy

Sockets over VIA (3)

Using a User-level Sockets Layer

- MS WinSock Direct Path, SOVIA



application

Library

User-level Sockets Layer

VIPL

kernel

VI Kernel Agent

VI-NIC

- Pros
  - ~~TCP/IP overhead~~
  - ~~Context switching~~
  - ~~Data copy~~

- Cons
  - Less compatibility

Goals of SOVIA

- **Provide a simple, yet versatile communication service**
- **Accelerate the existing Sockets-based applications with a reasonable effort**
  - Parallel and cluster file systems
  - User-level software DSMs, …
- **Target for another upper layers**
  - RPC (Remote Procedure Calls)
  - MPI (Message Passing Interface), …

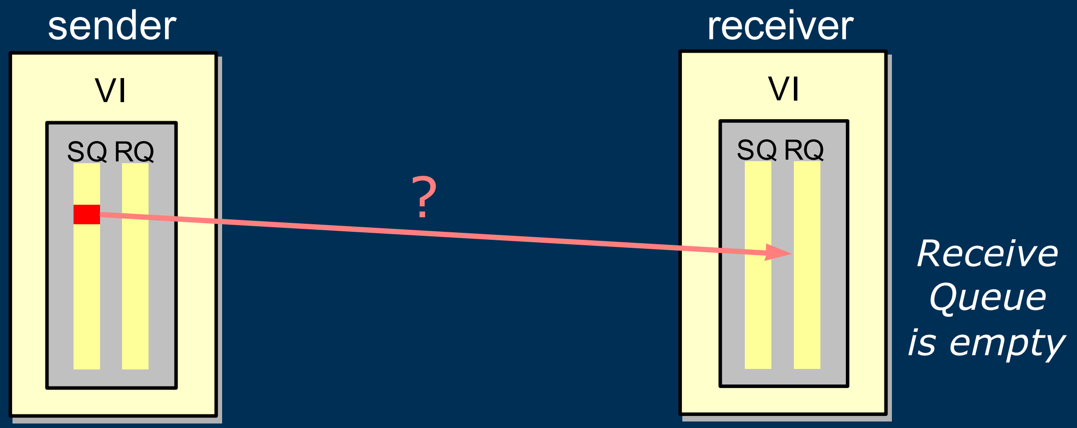
Performance Issues

- **Minimizing latency**
- Maximizing bandwidth
- Evaluation

Synchronization Protocol (1)

✍ VIA’s pre-posting constraint

- The receiver should pre-post a descriptor before the sender initiates a data transfer.
- A high-level synchronization protocol needs to be implemented between the sender and receiver.



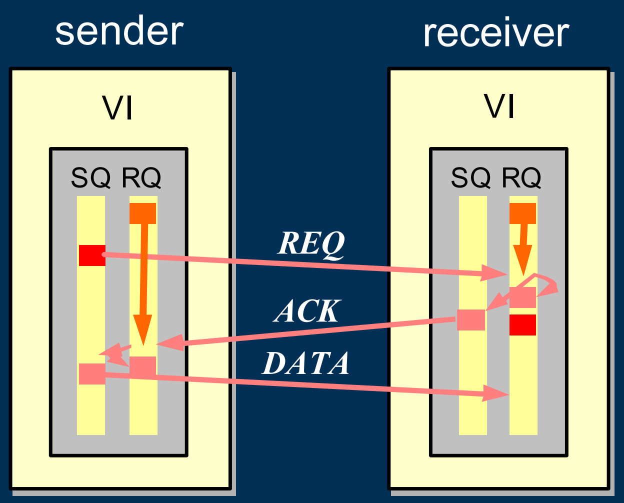
Receive Queue is empty

Synchronization Protocol (2)

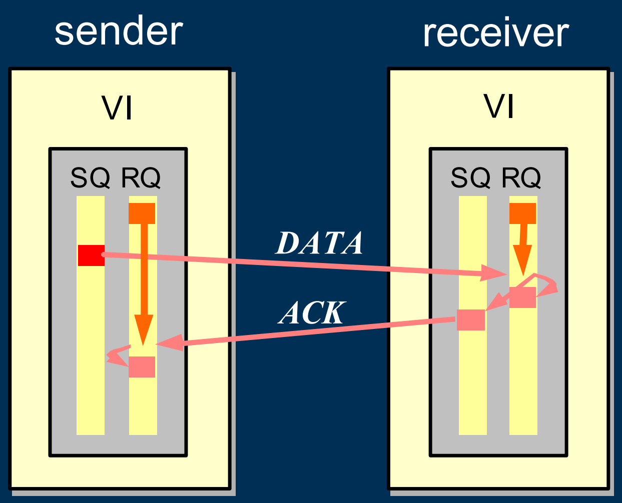
✍ Satisfying pre-posting constraint

- Guarantee that at least one descriptor is available on the RQ for each send.

Three-way Handshaking



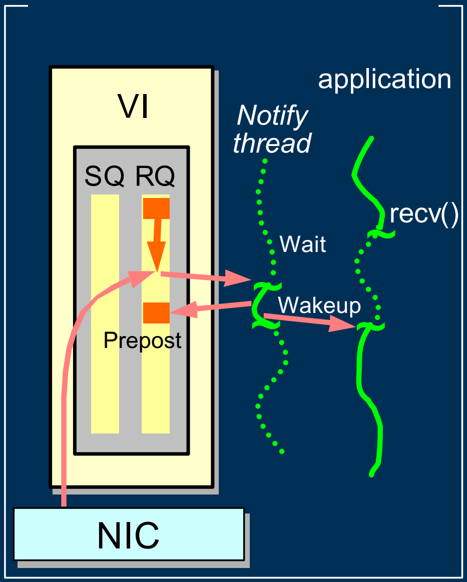
Two-way Handshaking

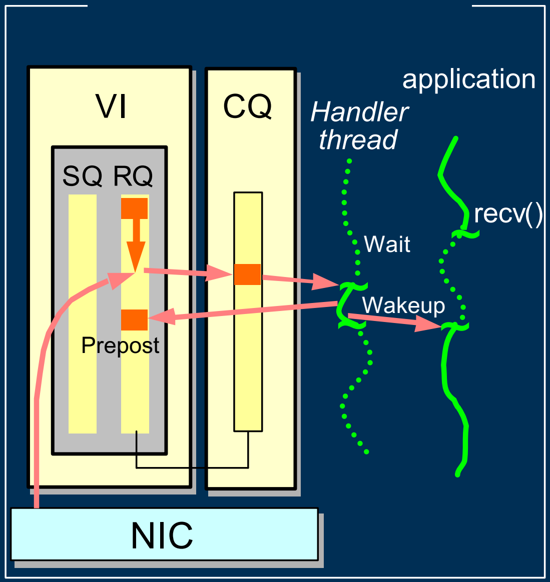


Message Handling Strategies

- **Incoming message handling**
  - The completed descriptors should be extracted from a queue manually.
  - Incoming messages are delivered asynchronously.
  - When and by whom?

Multi-threaded Handling

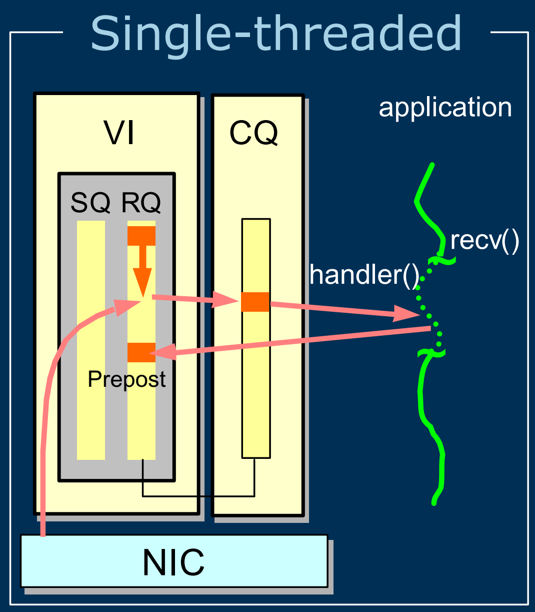




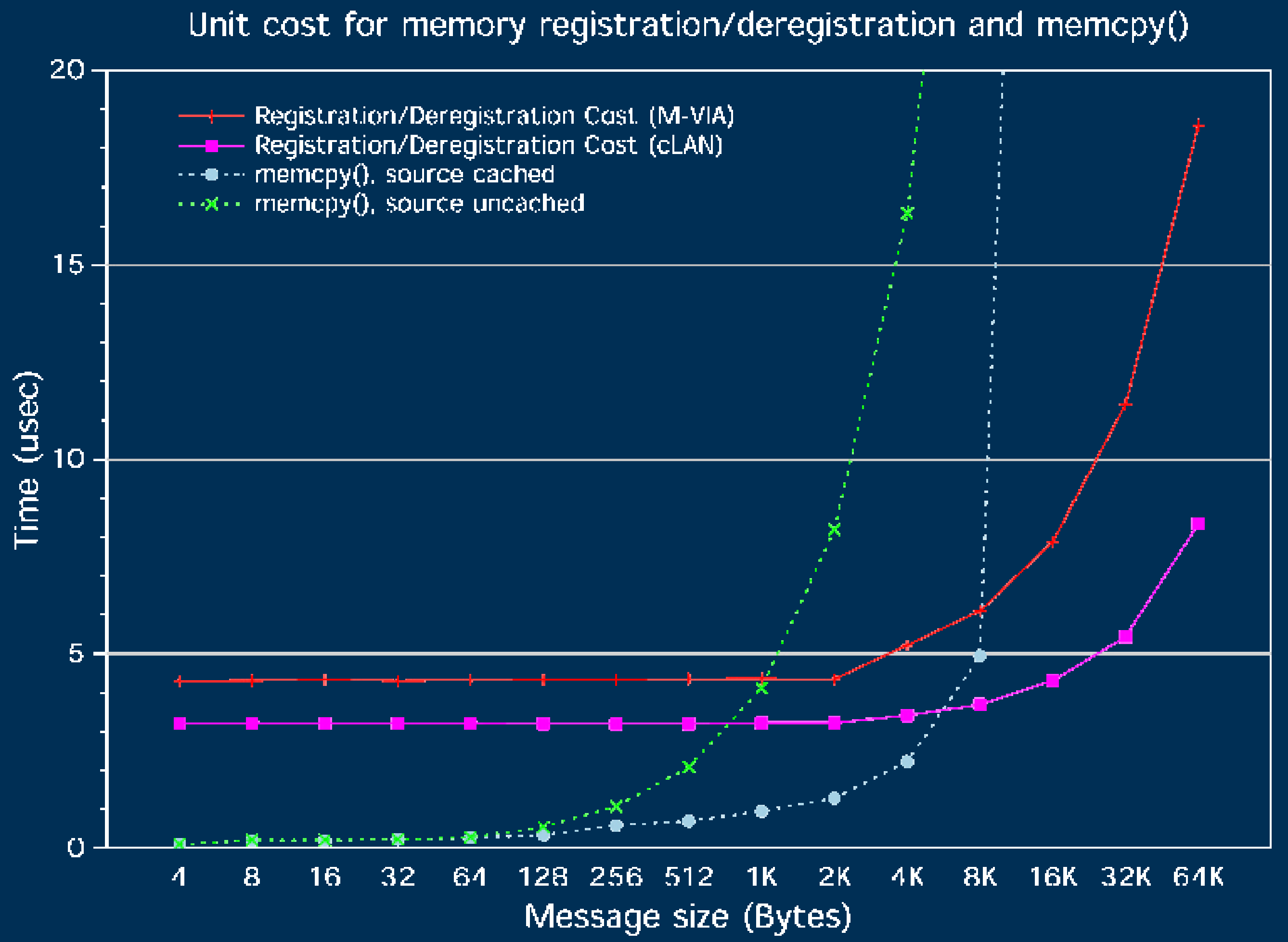
?

Single-threaded Handling

- The application thread handles incoming messages when it calls communication-involved functions.
- Can overlap the communication with the computation if multiple descriptors are pre-posted.



Conditional Registration



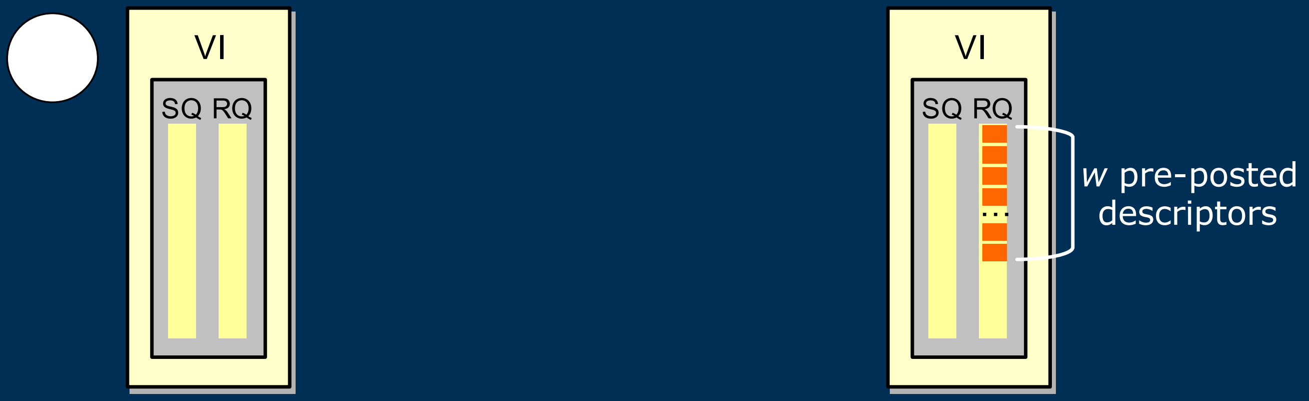
Performance Issues

- Minimizing latency
- **Maximizing bandwidth**
- Evaluation

Flow Control

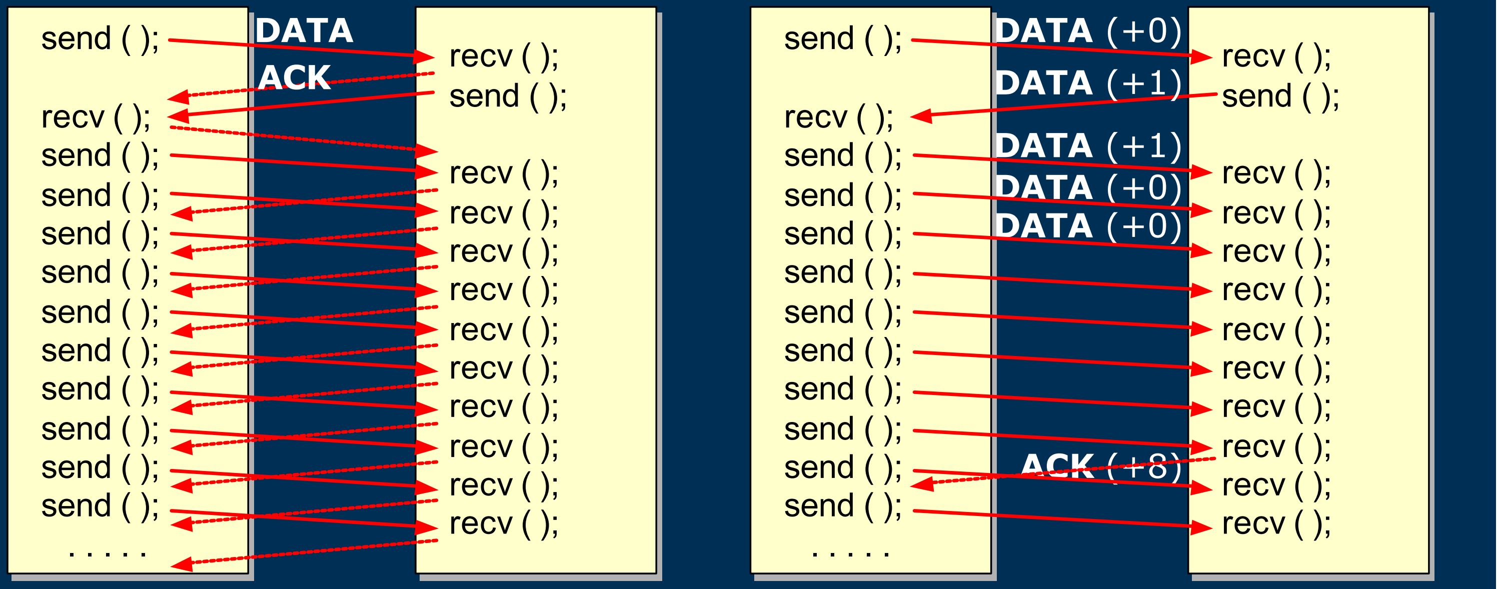
A sliding window protocol

- Receiver pre-posts *w* descriptors
- send() decreases *w* by one
- Sender is blocked if *w* = 0
- Window size *w* is increased by ACK



Delayed ACKs and Piggybacking

- Delay up to *t* ACKs
- Piggyback ACKs to DATA



send ( );
recv ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
. . . . .

DATA
ACK

recv ( );
send ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );

send ( );
recv ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
send ( );
. . . . .

DATA (+0)
DATA (+1)
DATA (+1)
DATA (+0)
DATA (+0)
ACK (+8)

recv ( );
send ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );
recv ( );

Before **After (*t* = 8)**

Combining Small Messages

- TCP uses Nagle algorithm
- Append outgoing data into a buffer (< 2KB), and start a timer
- The buffer is flushed either
  - when the timer expires
  - when there is no enough room
  - when the size is > 2KB
  - when the application calls recv() or close()
- Dynamically turned off for latency-sensitive applications.

Performance Issues

- Minimizing latency
- Maximizing bandwidth
- **Evaluation**

Evaluation Platform

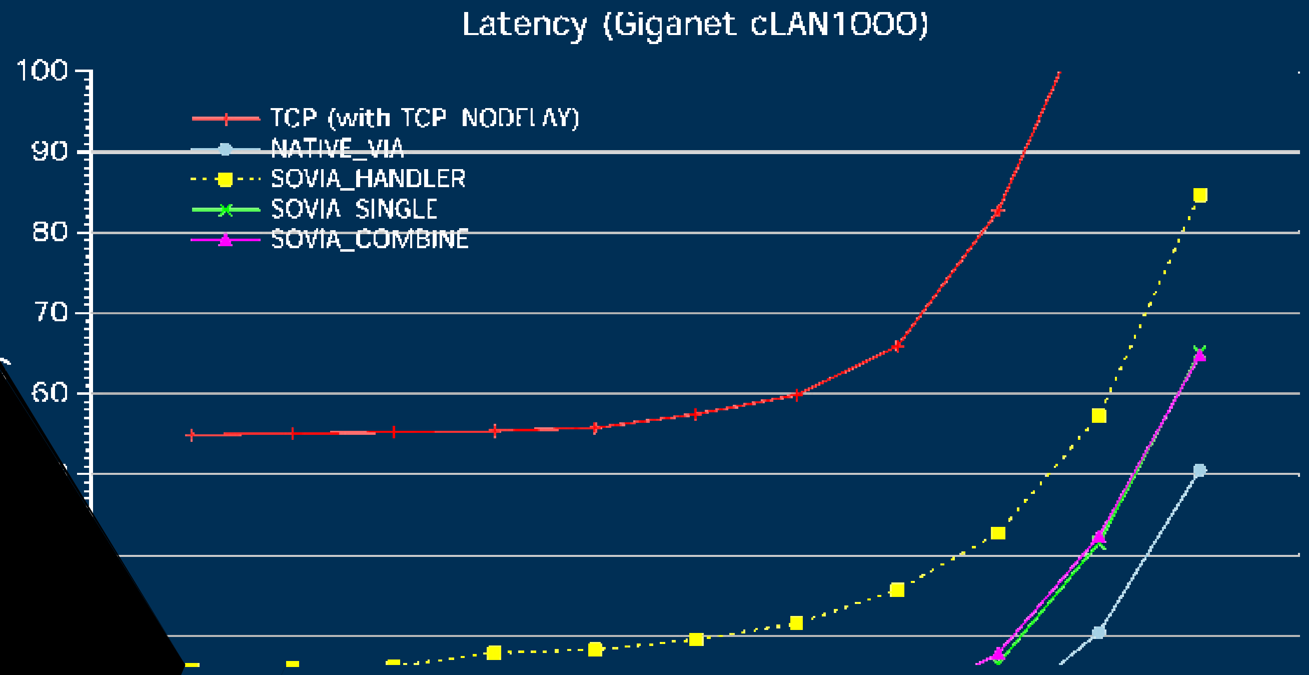
Linux servers

- Intel L440GX+ motherboard
- Intel Pentium III-500MHz
- 256MB main memory
- Linux kernel 2.2.16

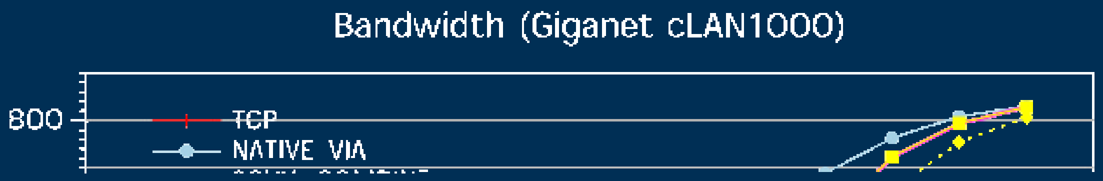
VIA implementations

- cLAN v1.1.1 on Giganet cLAN1000

Latency (cLAN)



Bandwidth (cLAN)



Design Choices

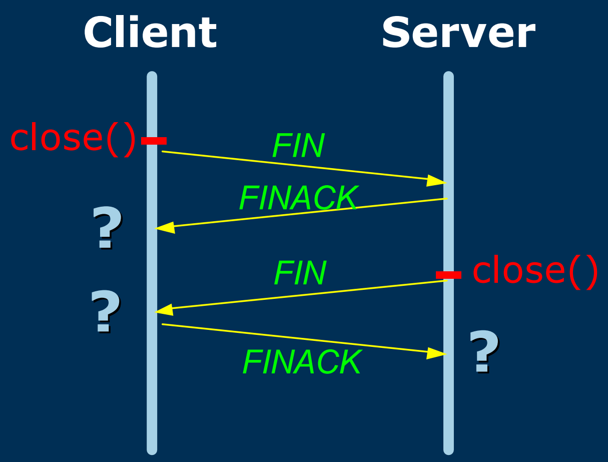
- ✍ **Minimizing latency**
  - Two-way handshaking
  - Single-threaded implementation
  - Conditional memory registration
- ✍ **Maximizing bandwidth**
  - A sliding window protocol
  - Delayed acknowledgments and piggybacking
  - Combining small messages

Compatibility Issues

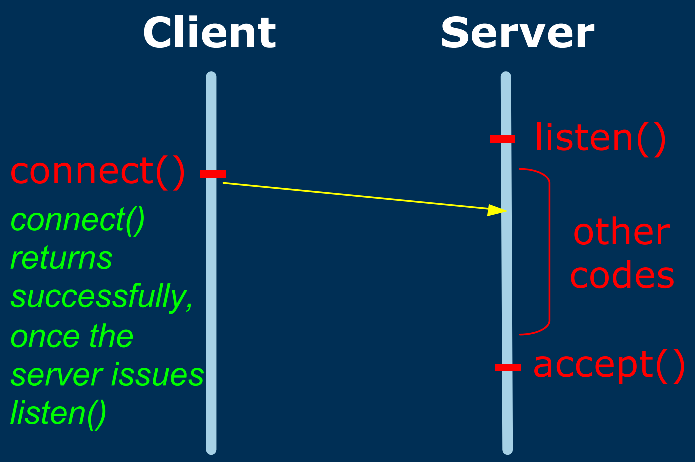
- **Connection management**
- Enhancing portability

Connection Management

- Problems in the single-threaded implementation



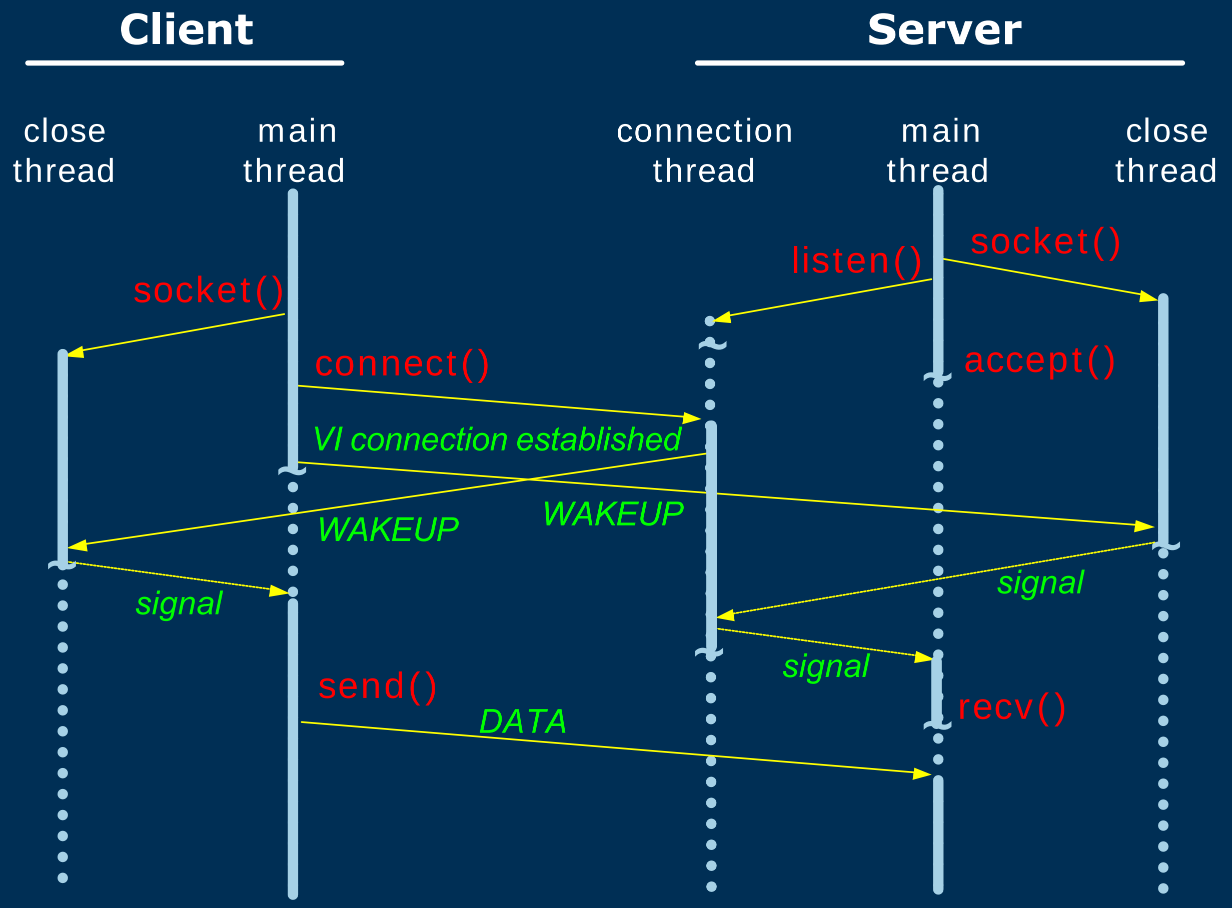
No chance to handle incoming messages after last close()



connect() returns successfully, once the server issues listen()

VipConnectWait() can’t be called in listen(), as it would block the thread.

Establishing a Connection



Client

close thread

main thread

Server

connection thread

main thread

close thread

socket()

listen()

socket()

connect()

accept()

VI connection established

WAKEUP

WAKEUP

signal

signal

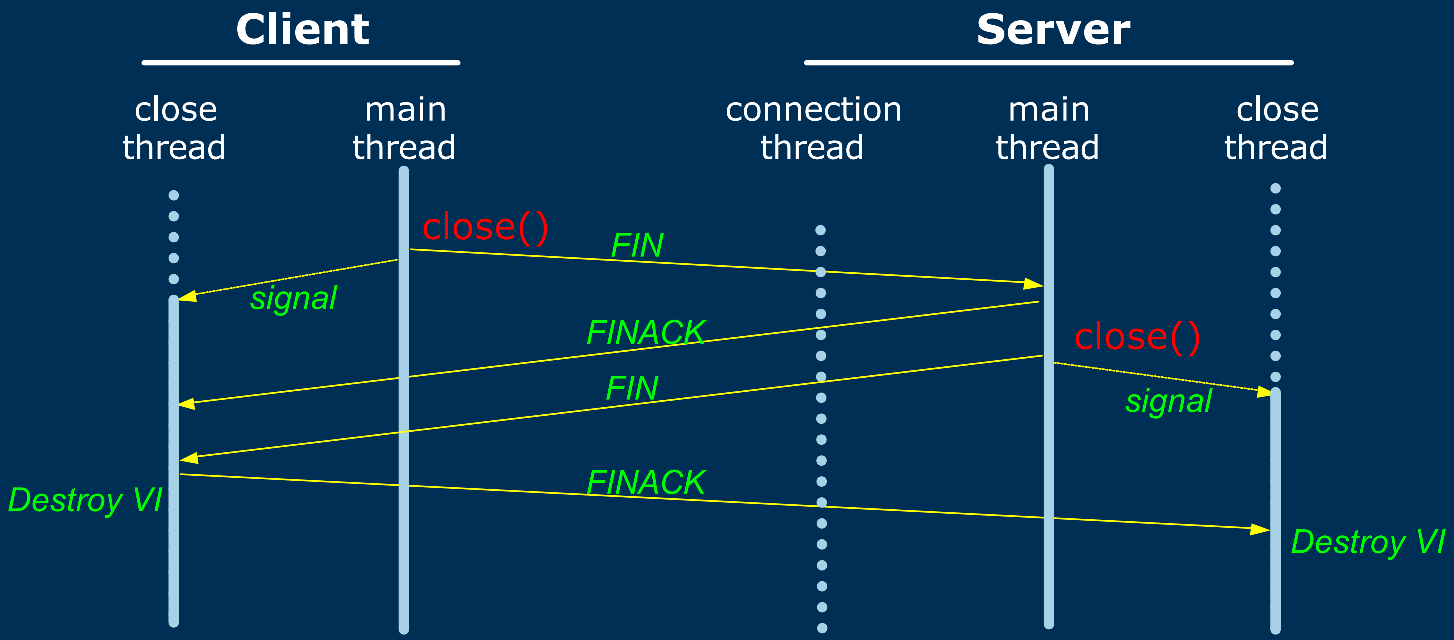
signal

send()

DATA

recv()

Closing a Connection



- The close thread is not activated if there is an open connection.
- Hence, the presence of the close/connection thread dose not affect the application’s performance.

Compatibility Issues

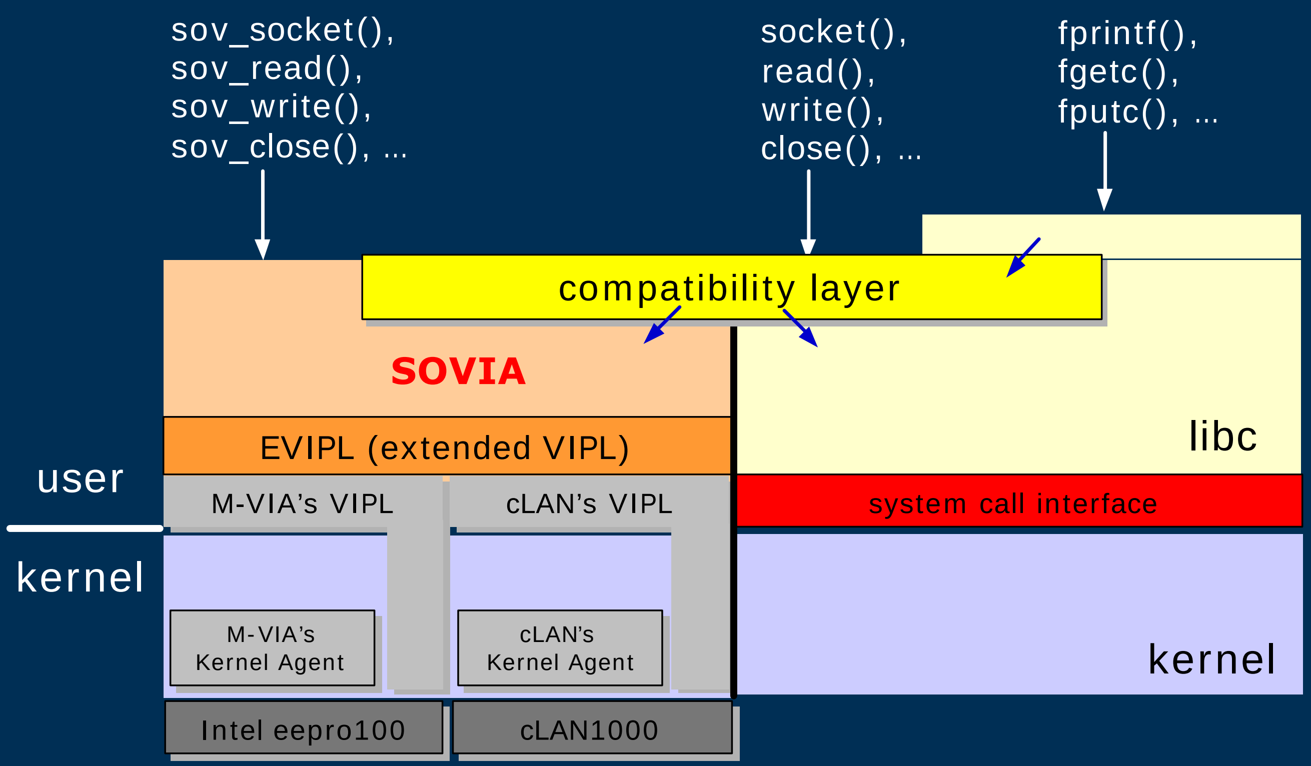
- Connection management
- **Enhancing portability**

Issues in Porting Applications

- **Sockets-specific interface**
  - socket(), connect(), accept(), …
- **File system interface**
  - read(), write(), close(), …
- **Standard I/O library**

```
int s;
FILE *fp;
  . . .
s = socket (AF_INET, SOCK_STREAM, 0);
connect (s, (struct sockaddr *) &server, sizeof(server));
fp = fdopen (s, “w”);
fprintf (fp, “Hello, world…\n”);
```

Enhancing Portability



Other Compatibility Issues

- **Fork()**
  - OK if the child process is not involved in any communication – Had to fix the VIPL due to the copy-on-write problem.
  - Sockets can not be shared with child processes.
- **Exec()**
  - Exec() will destroy any user-level data.
- **Normal termination**
  - Register a cleanup function using atfinalize().
- **Abnormal termination**
  - Catch the abnormal termination in signal handlers and call a cleanup function.

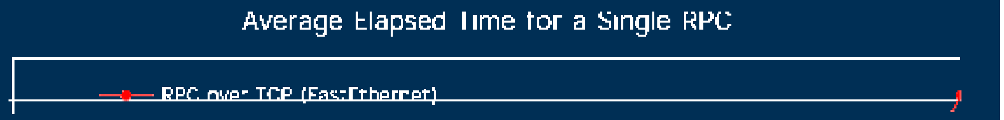
Results on
Real Applications

FTP Performance

✍ Linux NetKit 0.16

- linux-ftpd-0.16 & netkit-ftp-0.16

	File 1	File 2
File size (bytes)	19,090,223	145,864,380
TCP/IP on Fast Ethernet	90Mbps (1.63 sec)	90Mbps (12.7 sec)
TCP/IP on cLAN	262Mbps (0.59 sec)	254Mbps (4.61 sec)
SOVIA on cLAN	573Mbps (0.27 sec)	532Mbps (2.20 sec)
Local copy (in ramdisks)	611Mbps (0.25 sec)	538Mbps (2.17 sec)


Average Elapsed Time for a Single RPC
RPC over TCP (FastEthernet)

Concluding Remarks

- **The SOVIA layer provides**
  - High performance from VIA
  - Portability based on Sockets API

- **On-going & Future Work**
  - Accelerate other applications
    - Parallel file systems (e.g. PVFS)
    - Network Block Device (e.g. GNBD)
    - User-level software DSMs
    - Message-passing library, etc.
  - Compare with kernel-level SOVIA