Direct Sockets

25.1.2005

Christian Leber
christian@leber.de

Lehrstuhl Rechnerarchitektur
Universität Mannheim

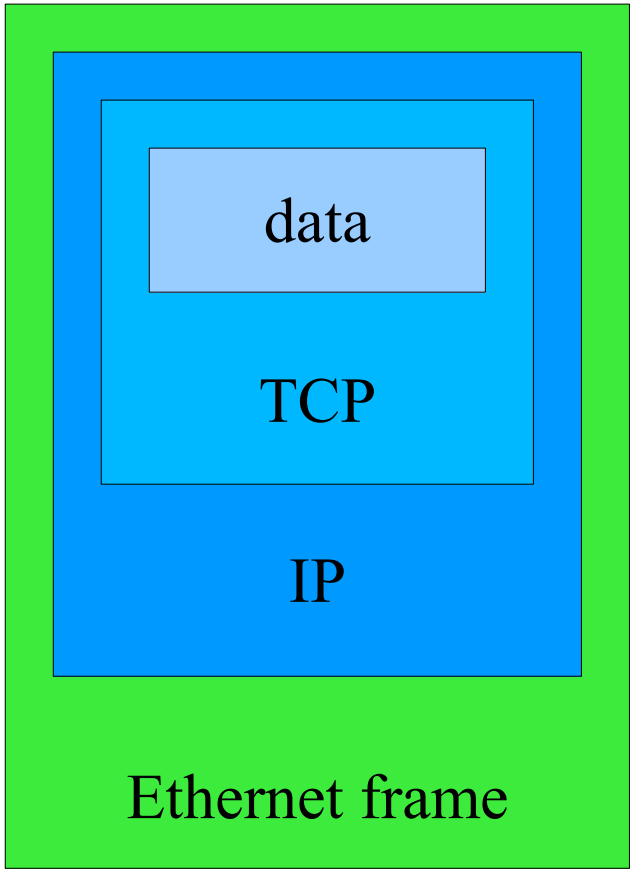
Outline

- Motivation
- Ethernet, IP, TCP
- Socket Interface
- Problems with TCP/IP over Ethernet
- Solutions for some of this problems
- SOVIA: Sockets Layer Over Virtual Interface Architecture
- SDP: Sockets Direct Protocol
- Myrinet Sockets-GM
- 10 Gigabit Ethernet
- Conclusion

Why direct sockets?

- for higher bandwidth and lower latency
- fast networks (Myrinet, Infiniband, Quadrics and of course Atoll) are deployed in clusters, why should they not be used to speed-up socket applications?
- there are many “legacy” applications that are using TCP/IP sockets, so it would be interesting to speed them up without changing the applications.

Layers



Ethernet

- relative high latency
- low bandwidth (unidirectional 1 Gbit)
- unreliable data transfer
- small MTU (Maximum Transfer Unit)
- simple cabling
- very cheap (virtually free for small network)
- works out of the box
- very good availability

IP – Internet Protocol

- unreliable protocol to transfer datagrams
- datagrams may be:
  - delayed
  - lost
  - duplicated
  - delivered out of sequence

TCP – Transmission Control Protocol

- TCP provides a reliable datastream on top of IP, it handles:
  - lost datagrams
  - corrupted datagrams
  - out of sequence delivery
  - reliable only in terms of not hostile environments, TCP doesn't provide authentication or encryption
- 4.2BSD was released in 1983 and included the first widely available release of TCP/IP and the sockets API [Stev]

Socket interface

- int **socket**(int domain, int type, int protocol);
  creates and endpoint for socket communication and returns a descriptor
- int **connect**(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
  initiates a connection
- **read** and **write** may be used like with a file
- int **bind**(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
  bind gives the socket sockfd the local address my_add
- int **listen**(int s, int backlog);
  int **accept**(int s, struct sockaddr *addr, socklen_t *addrlen);
  "*To accept connections, a socket is first created with socket(2), a willingness to accept incoming connections and a queue limit for incoming connections are specified with listen, and then the connections are accepted with accept(2).*" (listen manpage)

Socket Interface – Client Example

```
char buf[4096];
int len;
int fd=socket(PF_INET,SOCK_STREAM,IPPROTO_TCP);
struct sockaddr_in si;

si.sin_family=PF_INET;
inet_aton("127.0.0.1",&si.sin_addr);
si.sin_port=htons(80);
connect(fd,(struct sockaddr*)si,sizeof si);
write(fd,"GET / HTTP/1.0\r\n\r\n");
len=read(fd,buf,sizeof buf);
close(fd);
```

(example from "Scalable Network Programming Or: The Quest For A Good Web Server (That Survives Slashdot", Felix von Leitner)

Socket Interface – Server Example

```
int cfd,fd=socket(PF_INET,SOCK_STREAM,IPPROTO_TCP);
struct sockaddr_in si;

si.sin_family=PF_INET;
inet_aton("127.0.0.1",&si.sin_addr);
si.sin_port=htons(80);
bind(fd,(struct sockaddr*)si,sizeof si);
listen(fd);
while ((cfd=accept(fd,(struct sockaddr*)si,sizeof si)) != -1) {
  read_request(cfd);   /* read(cfd,...) until "\r\n\r\n" */
  write(cfd,"200 OK HTTP/1.0\r\n\r\n"
            "That's it. You're welcome.",19+27);
  close(cfd);
}
```

(example from "Scalable Network Programming Or: The Quest For A Good Web Server (That Survives Slashdot", Felix von Leitner)

Socket Interface - Select

- int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)

```
int some_fd;
fd_set read_set;
struct timeval tv;

FD_ZERO(&read_set); FD_SET(some_fd,&read_set);
tv.tv_sec=23; tv.tv_usec=0;
if (select(1,&read_set,0,0,&tv)==0) /* read, write, error, timeout */
  timeout();
if (FD_ISSET(some_fd, &read_set))
  can_read_on_some_fd();
```

- poll() is a variant of select

Problems with TCP/IP over Ethernet

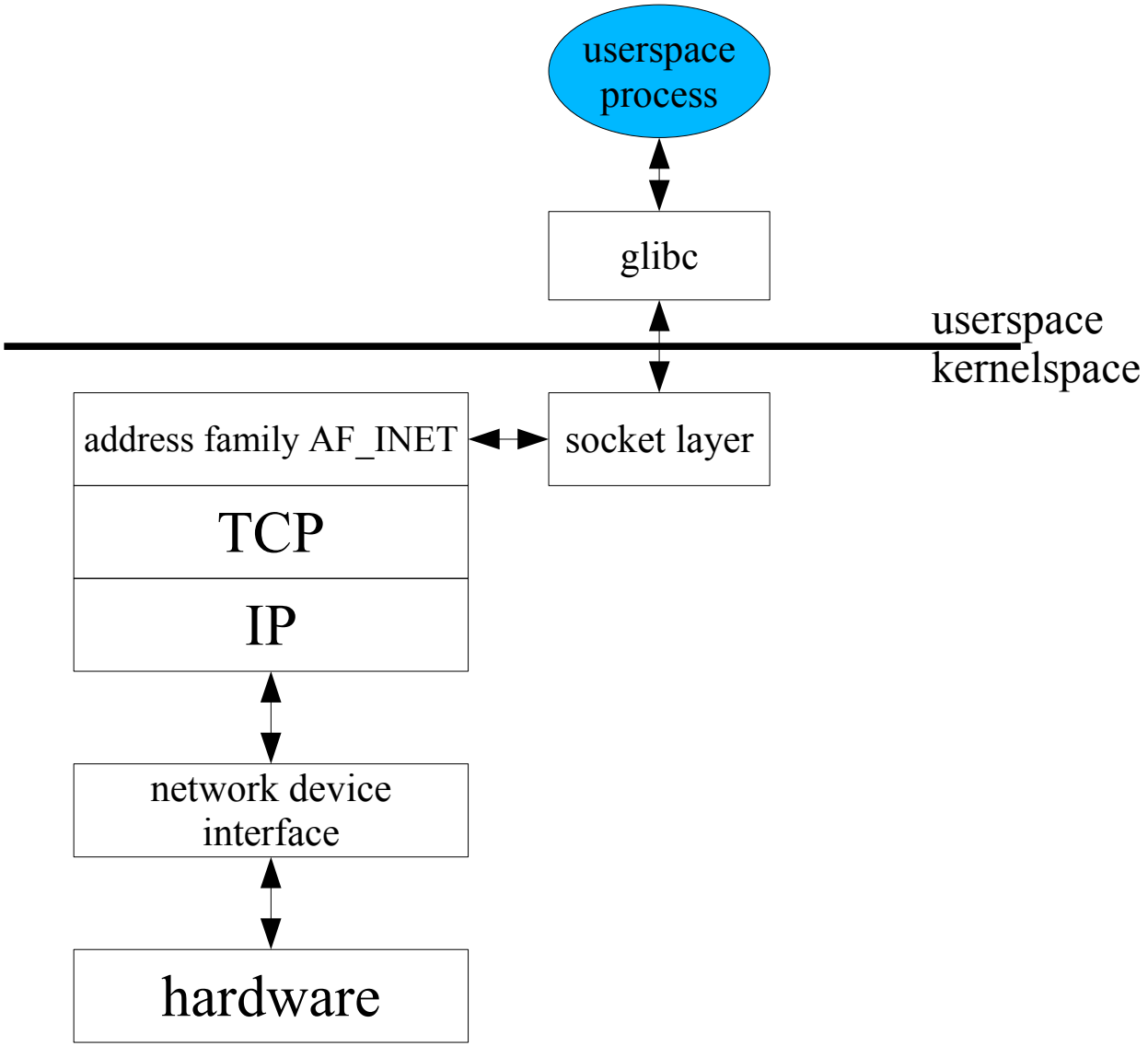
Problems:

- copying
  - increases latency
  - memory bandwidth
- CPU utilization
- userspace/kernelspace context switching
  this is not so bad any more, because it's down to about 376 cycles with Intel P4 or to 167 cycles with AMD64
- interrupts

Problems with TCP/IP over Ethernet

- MTU is too small
  - the MTU (Maximum Transfer Unit) is for Fast Ethernet (100MBit) 1500 Byte
  - for Gigabit Ethernet the default is still 1500 Byte, but up to 9000 Byte are also supported

Components


userspace process
glibc
userspace
kernelspace
address family AF_INET
socket layer
TCP
IP
network device interface
hardware

Solutions for some of this problems

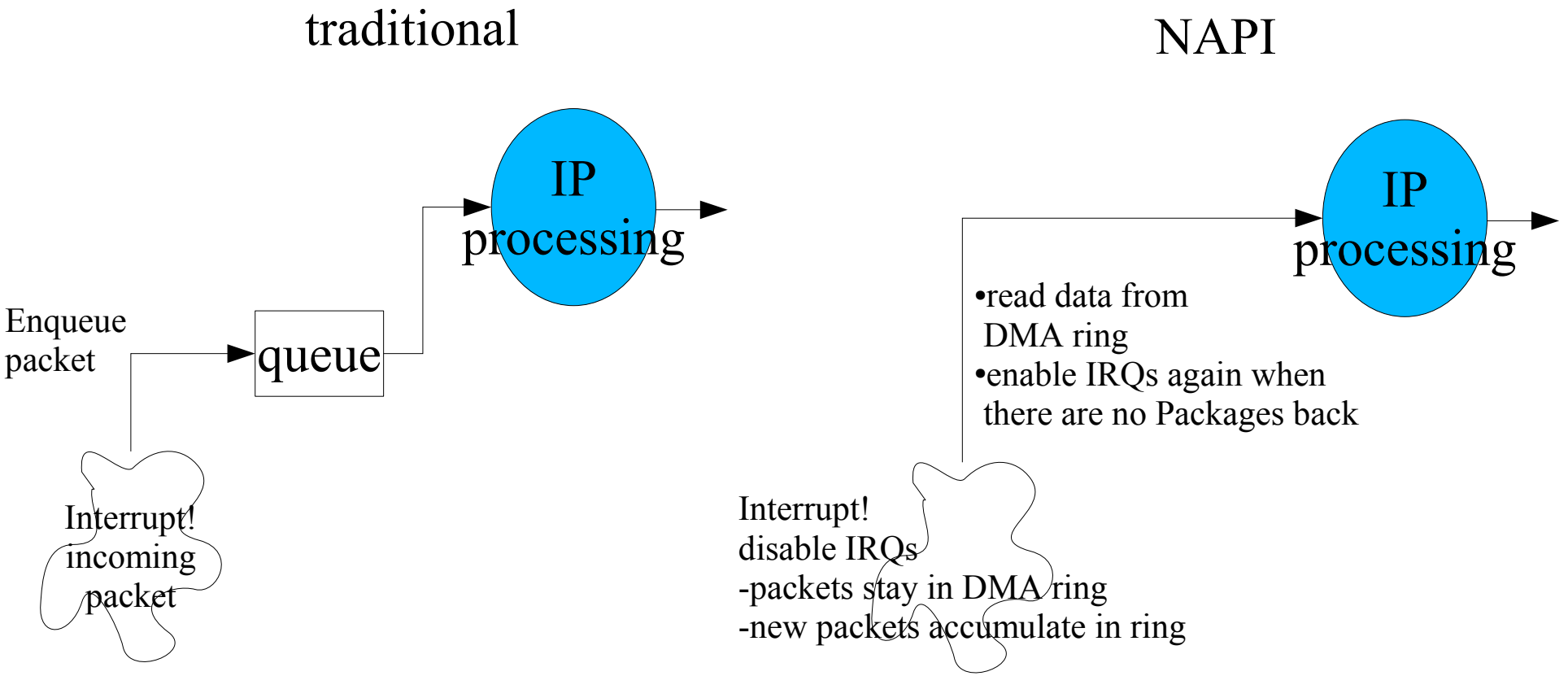
- It's necessary to avoid:
  - copying
  - interrupts

  to speed up the data transfer and/or reduce the resources utilisation.
- On the following slides some possible ways of achieving this will be shown that are already widely deployed in hard- and software.
- This techniques are also useful for things like the department firewall, a file server or a web server hosted in some facility.

NAPI – New API

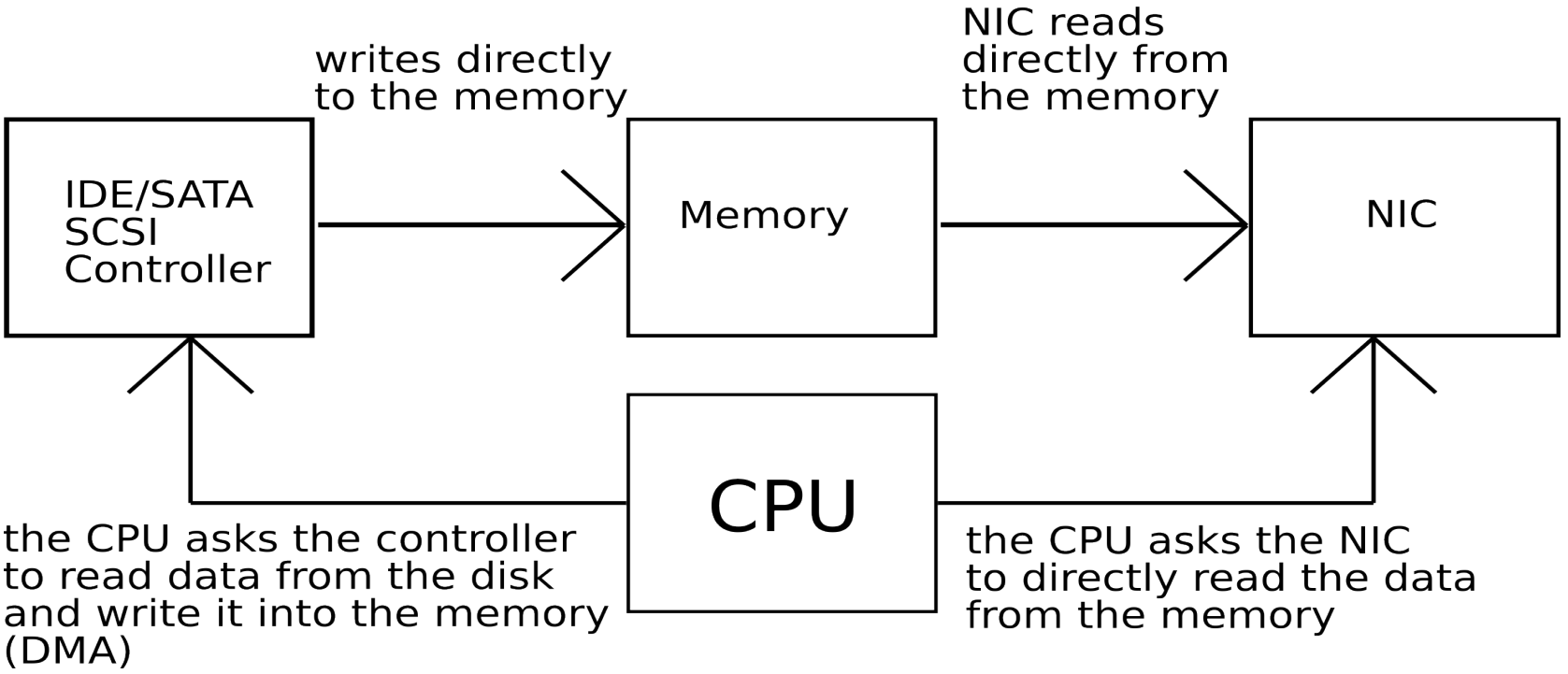
- NAPI is an API for network (Ethernet) drivers in Linux [NAPI]
- modern Ethernet cards are using a “ring” of DMA buffers to store received packages
- therefore it's possible to:
  - disable IRQs as long as there are new packages on the interface to not get into the state of “congestion collapse”, the IRQ is enabled again when there are no packages back
  - remove the backlog queue

NAPI – New API


traditional
IP processing
Enqueue packet
queue
Interrupt! incoming packet
NAPI
IP processing
•read data from DMA ring
•enable IRQs again when there are no Packages back
Interrupt!
disable IRQs
-packets stay in DMA ring
-new packets accumulate in ring

sendfile

- ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
- copies the data directly from in_fd (a file) to out_fd (a socket) in kernelspace
- supports zerocopy



sendfile

- zerocopy from “userspace” memory is also supported with a little trick:
  - create some file
  - mmap it to memory
  - write to the memory location
  - use sendfile() to send it away

TSO – TCP Segmentation Offload [TSO]

- DMA transfers in bigger chunks, checksumming and segmenting to smaller packets (for example 1,5 kb) is done by the NIC
- linux-kernel mailing list 2 Sep 2002 <scott.feldman@intel.com>

So, fire up you favorite networking performance tool and compare the performance gains between 2.5.32 and 2.5.33 using e1000. I ran a quick test on a dual P4 workstation system using the commercial tool Chariot:

Tx/Rx TCP file send long (bi-directional Rx/Tx)
w/o TSO: 1500Mbps, 82% CPU
w/ TSO: 1633Mbps, 75% CPU

Tx TCP file send long (Tx only)
w/o TSO: 940Mbps, 40% CPU
w/ TSO: 940Mbps, 19% CPU

- this applies to Intel e1000 Ethernet cards, but of course there are other chips with similar possibilities

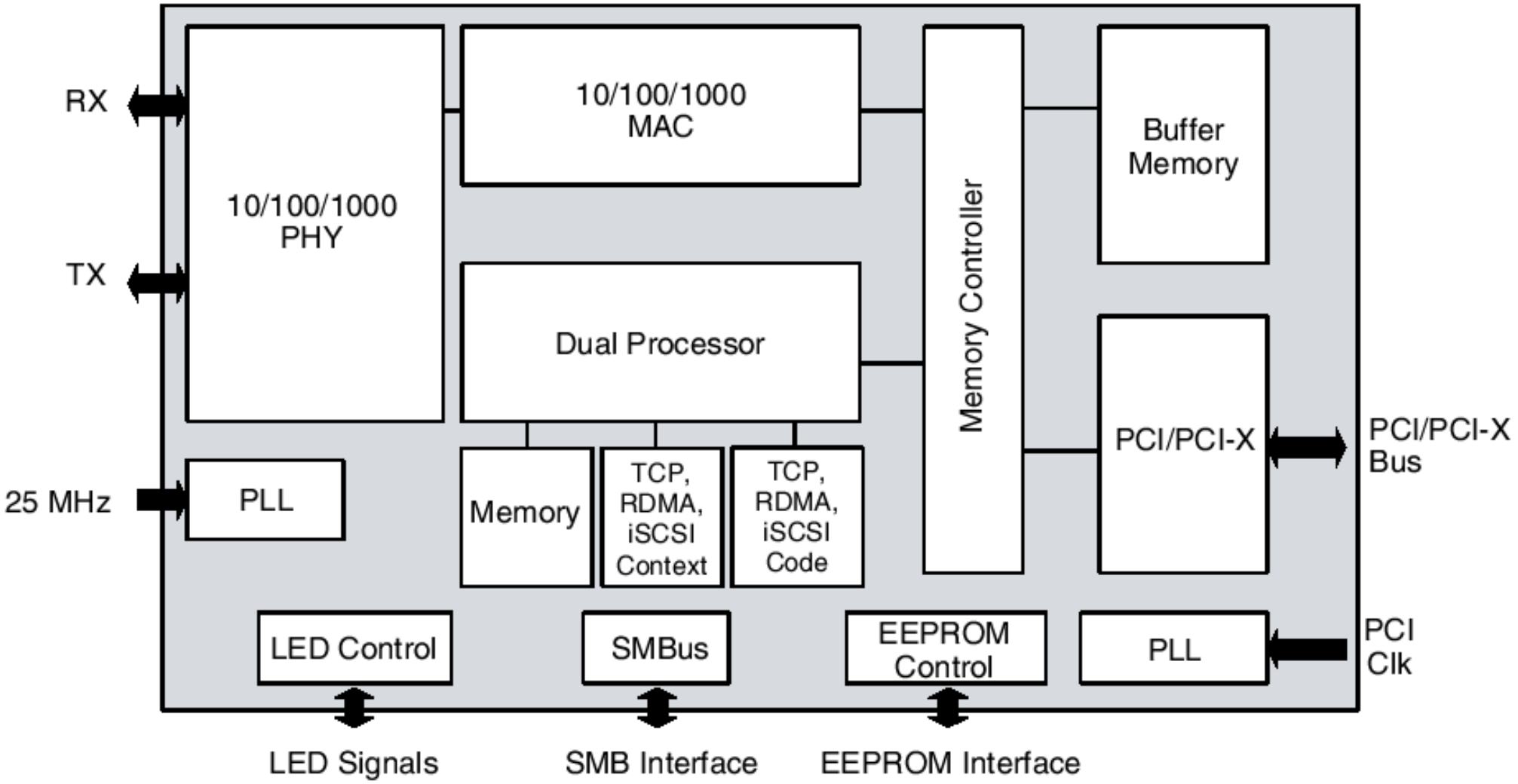
TOE – TCP Offload Engine

- device that handles TCP/IP completely
- saves resources on the host system
- but:
  - TCP/IP has many corner cases, therefore the firmware has to be very complex and is therefore error prone
    (linux TCP/IP is at least 70k lines of code)
  - it is questionable if or how fast vendors will supply security fixed in the case of a problem

TOE – TCP Offload Engine

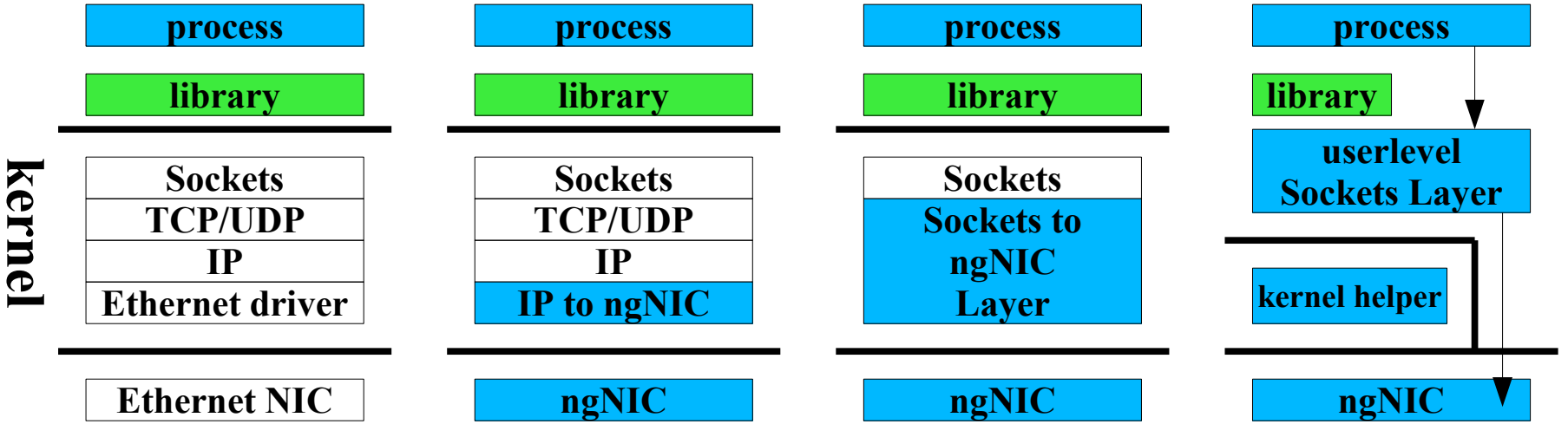
- an example for such an TOE is the Broadcom BCM5706 [BCM]
  - two MIPS CPUs
  - Gigabit Ethernet
  - RMDA over TCP (iWarp)
  - iSCSI
  - layout compatible to a normal Gigabit Ethernet chip from Broadcom
  - $35 in big quantities

TOE – TCP Offload Engine



source: Broadcom BCM5706 data sheet

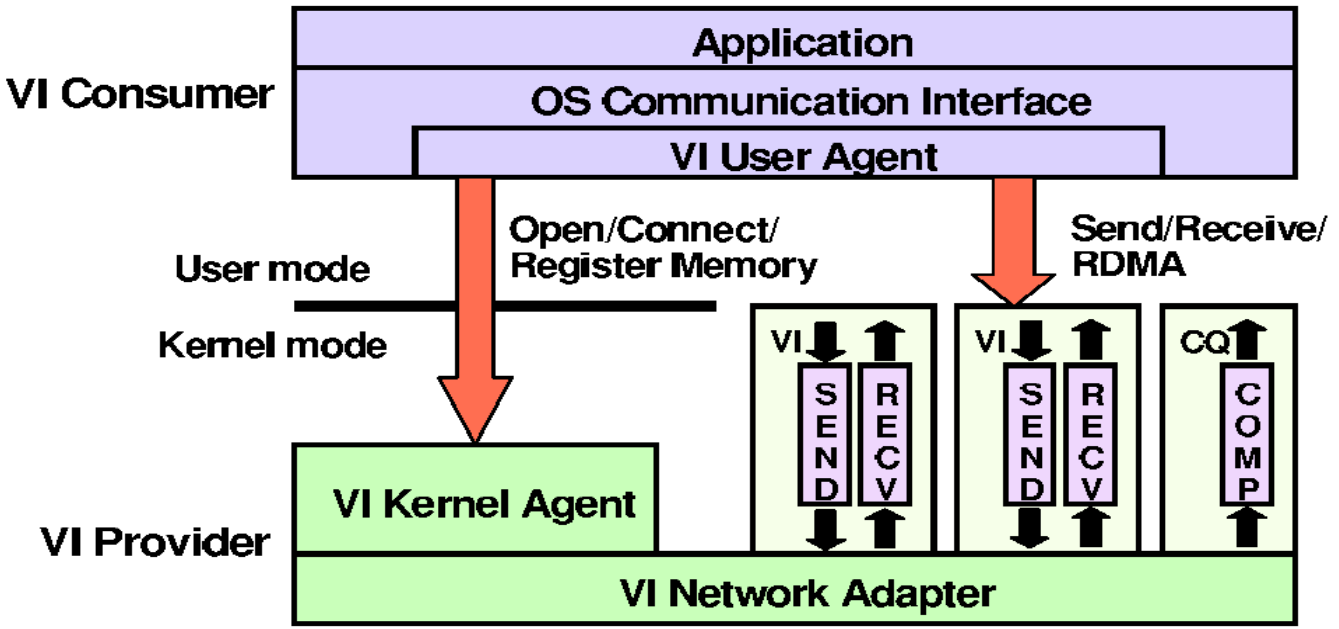
Ways to implement direct sockets



ngNIC means next generation NIC, like: Myrinet, Infiniband...

SOVIA

- “A User-level Sockets Layer Over Virtual Interface Architecture” [SOVIA]
- implemented on top of VIA



Source:http://www.cse.ohio-state.edu/~panda/788/papers/4i_sovia.pdf

SOVIA

- userspace implementation

```
// find symbols in libc during initialization
dlhandle = dlopen("libc.so.6",RTLD_LAZY);
sockops->socket = dlsym(dlhandle, "socket");
sockops->bind = dlsym(dlhandle, "bind");
...
dlclose(dlhandle);
...
int socket(int domain,int type,int proto) {
  if(type==SOCK_VIA)
    return sov_socket(domain,type,proto);
  else
    return sockops->socket(domain,type,proto);
}
```

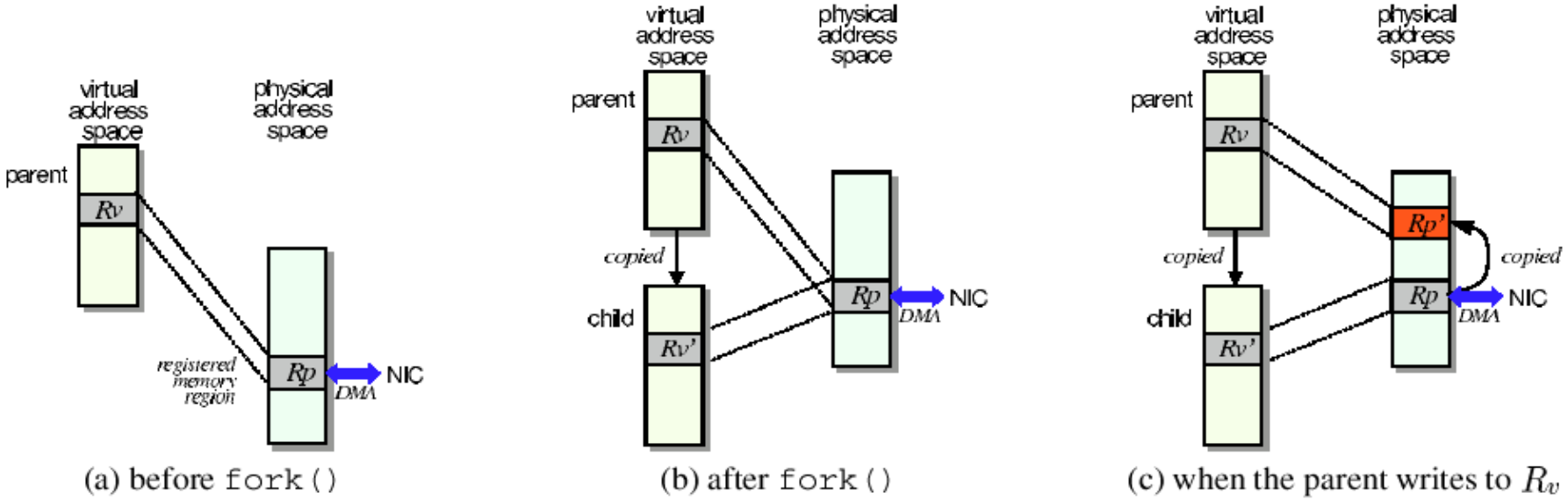
```
int sov_socket(int domain,int type,int proto) {
  int s=open("/dev/null", O_RDWR);
  sockdes[s] = sov_newsock(domain,typo,proto);
  return s;
}

int write(int s,void *buf,size_t size) {
  if(sockdes[s])
    return sov_write(s,buf,size);
  else
    return ___libc_write(s,buf,size);
}
```

- not usable for static linked applications
- but obviously it's expected that all applications that should make use of it, have to be changed.

SOVIA

- every piece of memory that is used to send data from it has to be registered
- this leads to a problem when using fork(); Linux is using COW (copy on write) when a process is forked, as soon as one of the processes writes to a shared page, the page will be copied, therefore the physical address isn't correct any more, the NIC hardware won't know were it is.
  SOVIA is therefore using shared memory for this areas



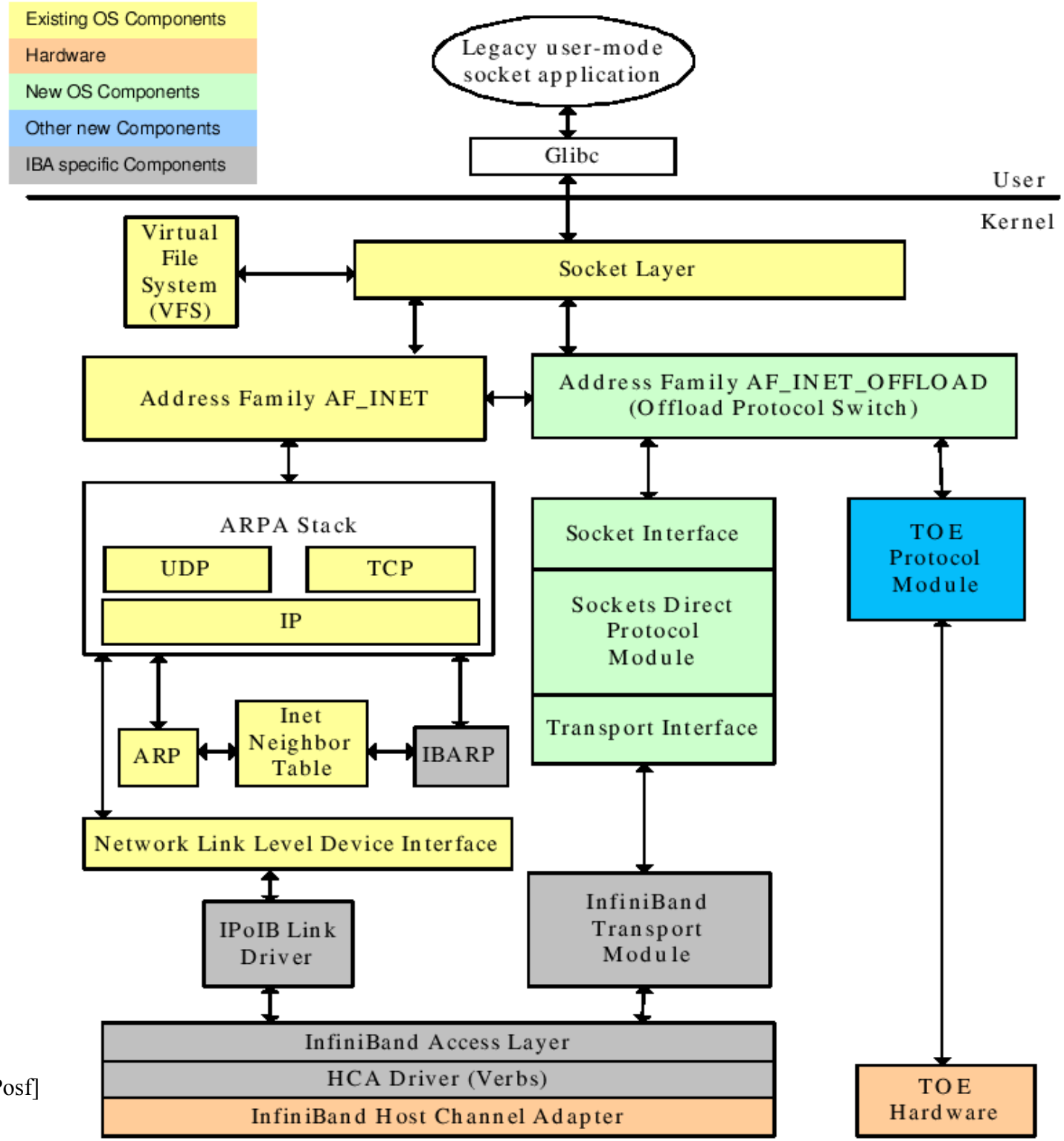
Source: http://www.cse.ohio-state.edu/~panda/788/papers/4i_sovia.pdf

SOVIA

- this is a proof of concept, but not an implementation that would have much use in real world applications
- requires source changes to the applications that should make use of it
- NO solution for select() or poll()
  - this would be essential for most socket application
- no multithreading
- paperware

SDP – Sockets Direct Protocol

- implements socket semantics over RDMA
- uses either Infiniband or iWARP
- more information and code may be found on http://infiniband.sourceforge.net

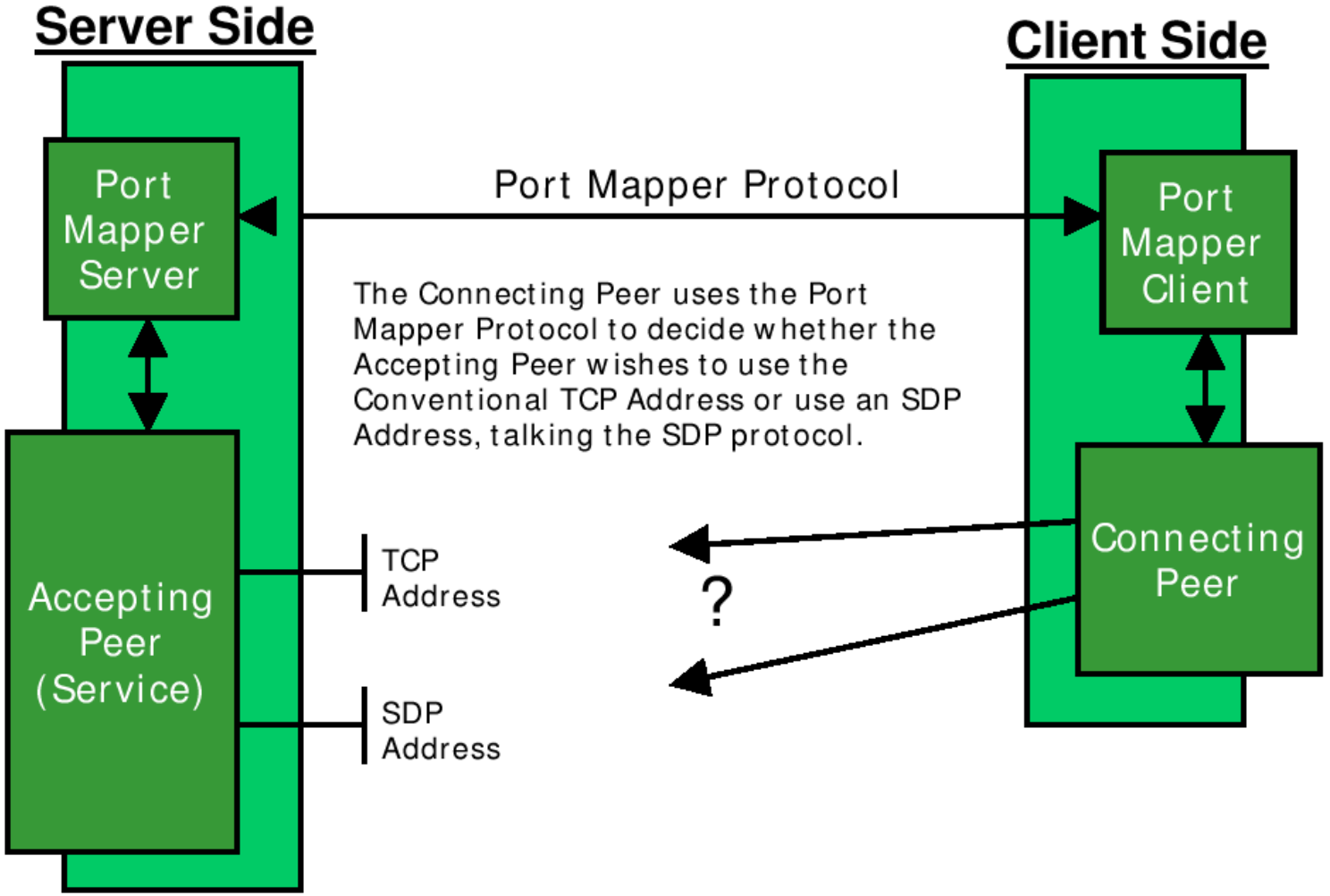


Source: [SDPosf]

SDP Port Mapper

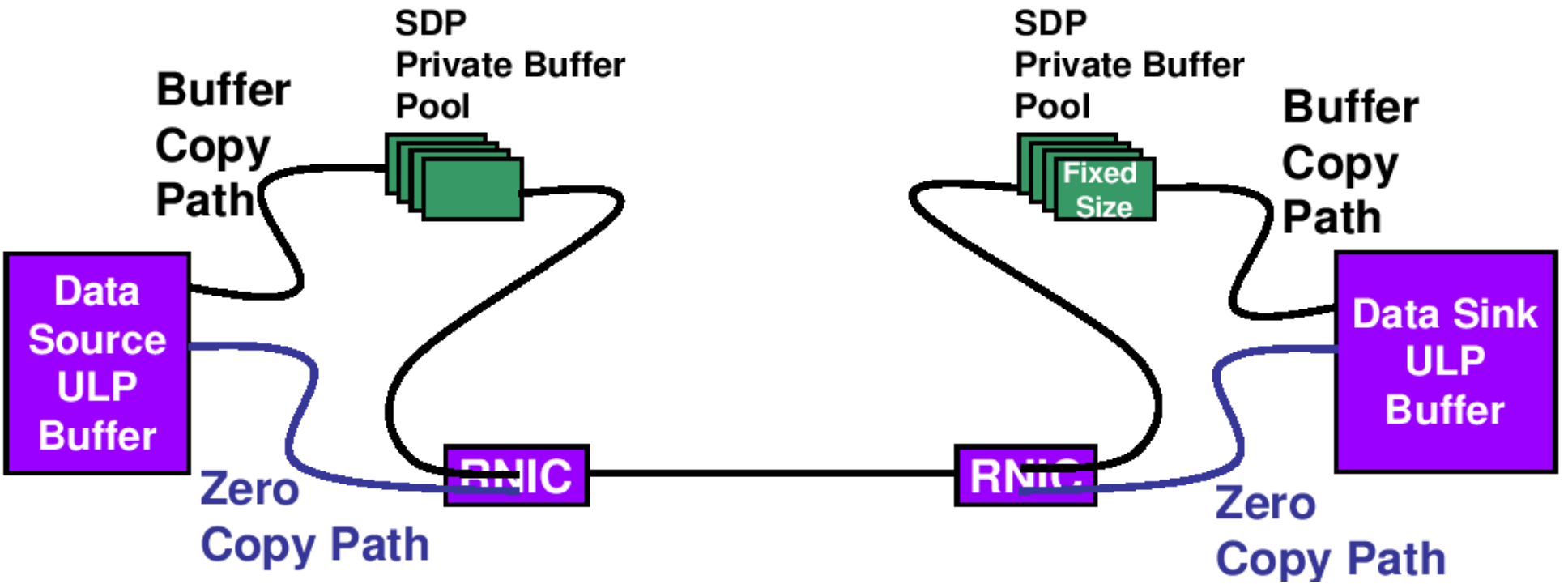
- has to run on all nodes of the SDP enabled network
- Port Mapper manages the address translation
- processes may use IP address and TCP port like if they would connect via Ethernet
  - maps IP/port to SDP address
  - fully transparent for the applications

SDP Port Mapper



source: [SDPtut]

SDP Buffering



Source: [SDPosf]

- small amount of data: copying
- big amount of data: pinning
  - write has to be blocking

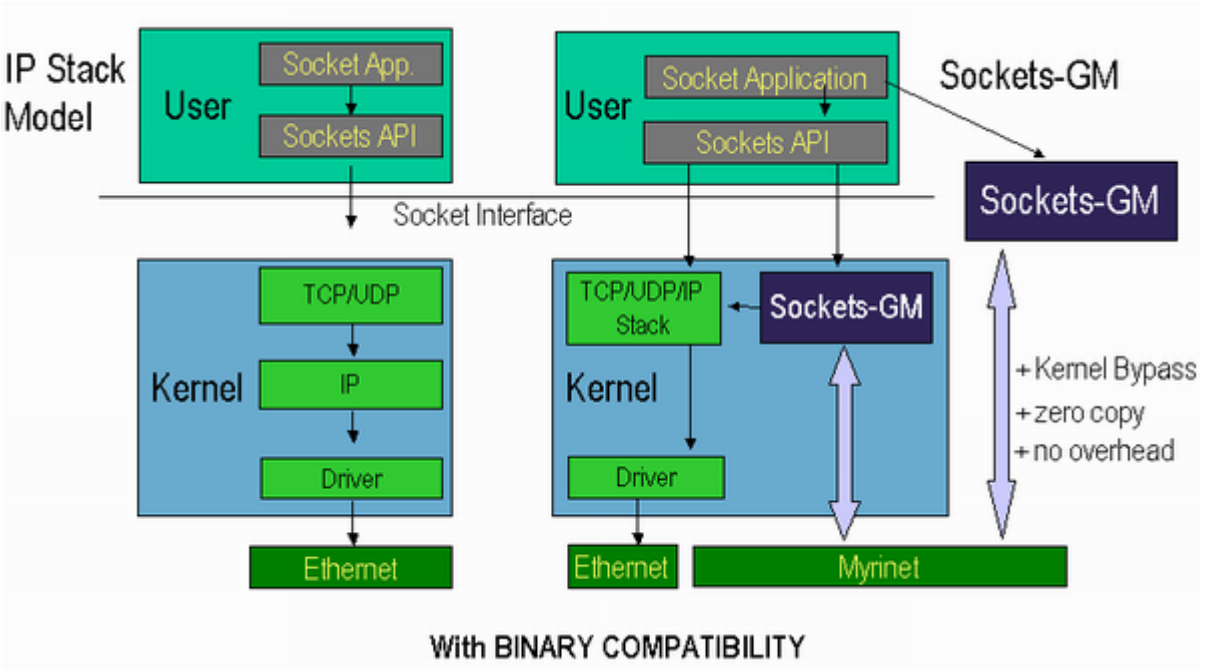
Problems with SDP

- The real problem with SDP is a licensing/patent issue.
  Infiniband was nearly completely merged into Linux as of 2.6.11-pre1 this month, the missing part is SDP.

  (Posted Dec 2, 2004 16:14 UTC (Thu) by guest roland_dreier)
  The patches being proposed for inclusion do not include anything related to SDP. Until the SDP/Microsoft patent situation is resolved, it will not be possible to merge SDP into the kernel. (http://lwn.net/Articles/112531/)

  Microsoft doesn't even say in it's “License Agreement” what patents it claims to own. (http://www.microsoft.com/mscorp/ip/standards/)
  It's impossible to find out if this patents are also claimed in Europe and/or Germany If this would be the case it's unclear if they are legal.

Myrinet – Sockets GM [GMSOCKS]



Source: http://www.myri.com/myrinet/performance/Sockets-GM/socketsgm-concept.png

- userspace implementation
- **or** kernelspace implementation, you can choose one

Myrinet – Sockets GM

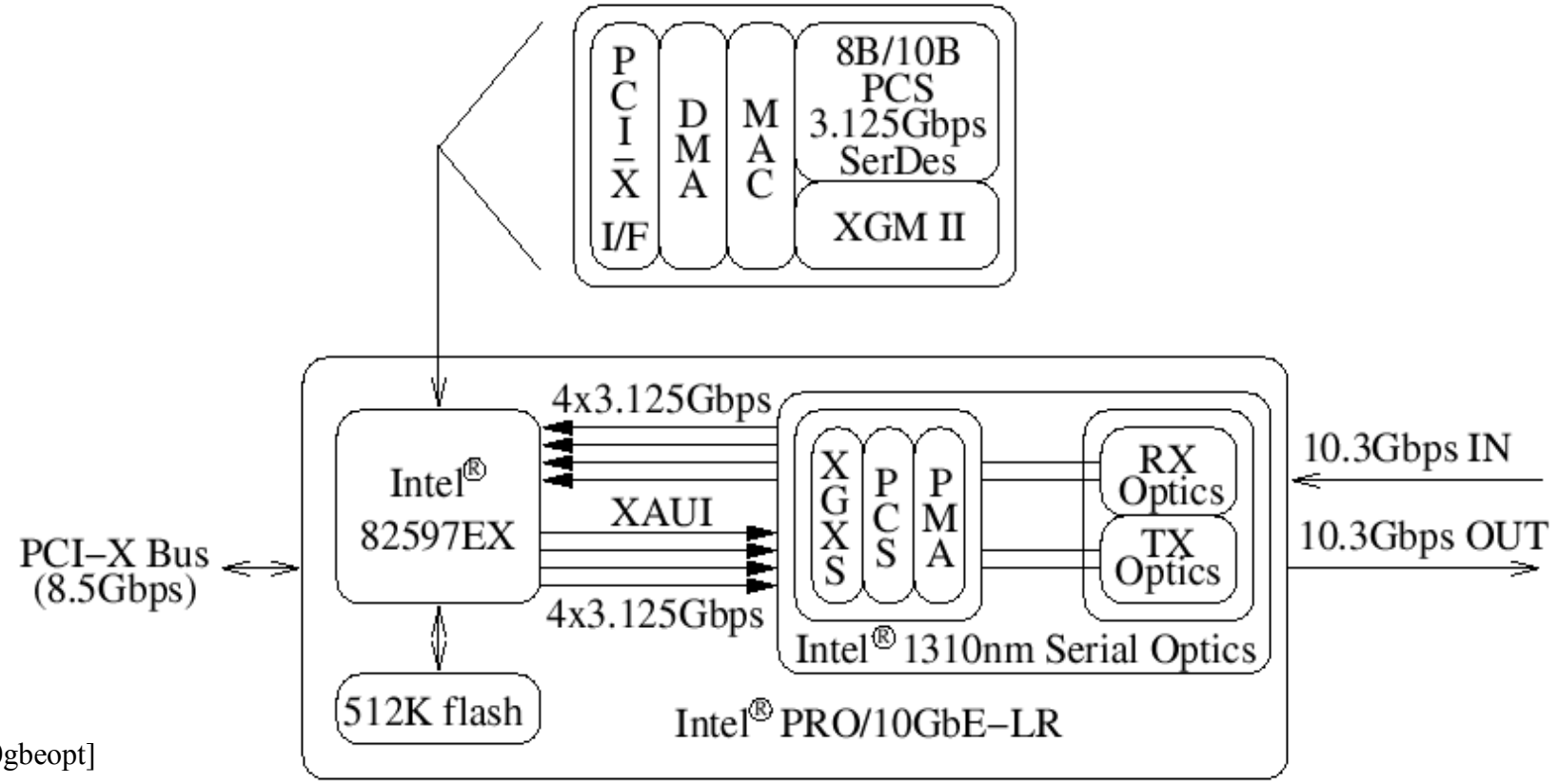
- Userspace implementation
  - claims to be transparent
  - select is implemented, but possibly limited in scalability
  - limitations:
    - processes that are using multiple forks
    - does obviously not work for static binaries
    - doesn't support epoll or sendfile
- Kernelspace implementation
  - transparent

10 Gbit Ethernet

- very good TCP/IP performance:
  Myrinet with TCP/IP emulation layer: 1,853 Gb/s 30µs
  Quadrics Elan3 with TCP/IP: 2,240 Gb/s <30µs
  10 Gbit Ethernet: 4,11 Gb/s 19µs
  (numbers from the "Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters and Grids: A Case Study" SC'03, November 15-21, 2003, Phoenix, Arizona, USA)
- with RDMA over TCP (iWarp) it could also be used as a transportation layer for SDP

10 Gbit Ethernet

- Hardware available, but **very** expensive (about 4700 USD)
- as of now only available as fiber version, but 10Gigabit Ethernet over copper seems to be possible [10gbecop]



Source:[10gbeopt]

Conclusion

- SDP is clearly the most advanced and the most convenient implementation of direct sockets, but said problem will stop wide adoption for now.
- the question if a userspace or a kernelspace implementation is better, depends completely on the application
- when 10 Gigabit Ethernet is available for a decent price and the application require only TCP/IP (and not the other and faster interfaces of Myrinet, Infiniband or Atoll) - it is the way to go.

questions?

Resources

- [Stev] W. Richard Stevens – Unix Network Programming Vol. 1
- manpages of: socket, connect, bind, listen, accept, select, poll
- "Scalable Network Programming Or: The Quest For A Good Web Server (That Survives Slashdot", Felix von Leitner
  http://bulk.fefe.de/scalable-networking.pdf
- [NAPI] "Beyond Softnet" Jamal Hadi Salim, Znyx Networks and Robert Olsson, Usenix 2001
  http://www.usenix.org/publications/library/proceedings/als01/full_papers/jamal/jamal_html/napi2.html
- [TSO] TSO support in Linux: http://kerneltrap.org/node/397
- [BCM] BCM5706 10/100/1000BASE-T TCP Offload Engine, RDMA, ISCSI/ISER and Ethernet Controller
  http://www.broadcom.com/products/product.php?product_id=BCM5706
- [SOVIA] "SOVIA: A User-level Sockets Layer Over Virtual Interface Architecture" Jin-Soo Kim, Kangho Kim, and Sung-In Jung, Electronics and Telecommunications Research Institute (ETRI)
  http://www.cse.ohio-state.edu/~panda/788/papers/4i_sovia.pdf

Resources

- SDP
  - [SDPtut] http://www.rdmaconsortium.org/home/SDP_tutorial_v1.0d.pdf
  - [SDPosf] http://infiniband.sourceforge.net/archive/OSF_SDP_HLD.pdf
- [GMSOCKS] Myrinet/Sockets-GM
  - Sockets-GM Overview and Performance
    http://www.myri.com/myrinet/performance/Sockets-GM/
  - http://www.myri.com/news/02512/slides/Fischer_sockets-gm.pdf
- [10gbeopt] “Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters and Grids: A Case Study” SC'03, November 15-21, 2003, Phoenix, Arizona, USA
  http://www.sc-conference.org/sc2003/paperpdfs/pap293.pdf
- [10gbecop] 10GE: Ethernet-Beschleunigung mit Kupferkabel
  http://www.heise.de/newsticker/meldung/52663