Endnodes

Network Bottlenecks

Networks are complex.

To provide ease-of-use it is necessary to introduce abstractions (socket interfaces, prefix-based forwarding, …)

Abstractions can result in performance penalty

We want to address the performance gap while keeping the abstractions (for ease-of-use)

Endnode bottlenecks

Endnodes are designed for general purpose computing

Structure

- Layered software, to ease development
- Implement protection mechanisms
- Implement general mechanisms

Scale

- Design choices that are ok for low traffic do not scale to high numbers

Examples

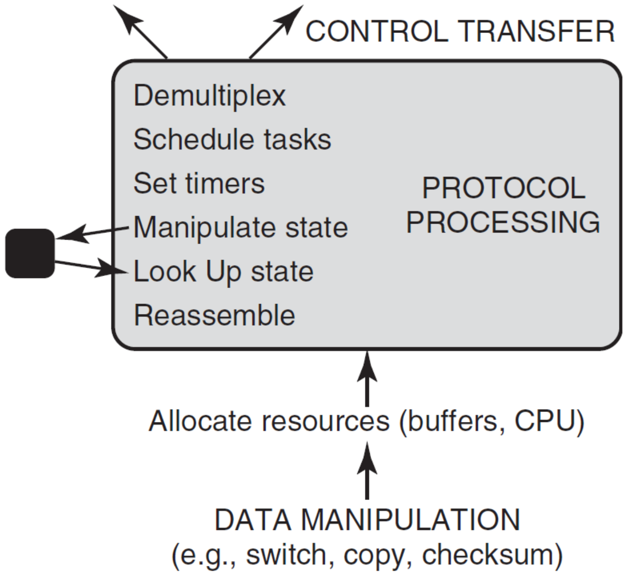
- Multiple copies across protection domains
- Control overhead
- Timers

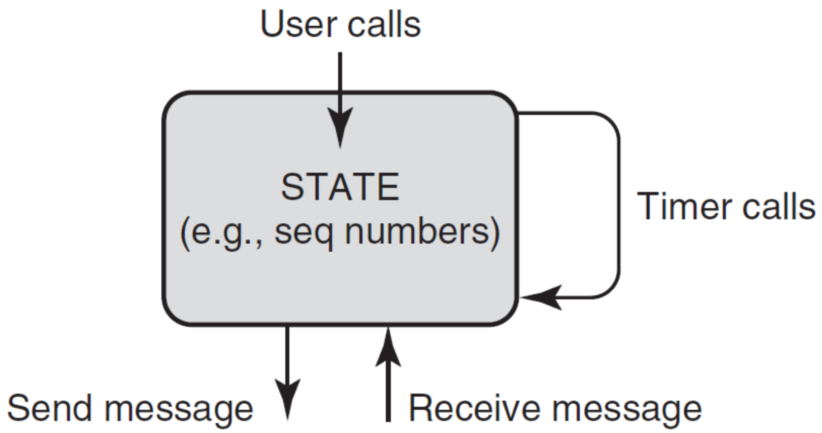
Protocols
Abstract Model

A protocol is:

- a state machine
- interfaces
- message formats

Most functions are common to all protocols





Protocols
Performance Environment and Measures

Measures

- throughput (completed jobs per second)
- latency (time to complete a job)

We focus on performance measures local to a node

Higher throughput generally means high latency, so a tradeoff must be found. The best solution depends on the environment

Some considerations

- Backbone links much higher speed than Access links
- HTTP >70% traffic, TCP >90%
- 50% transfers are small files (<50KB)
- RTTs >> propagation delays
- Poor locality in the backbone
- Small packets (about 50% packets is 40 bytes)

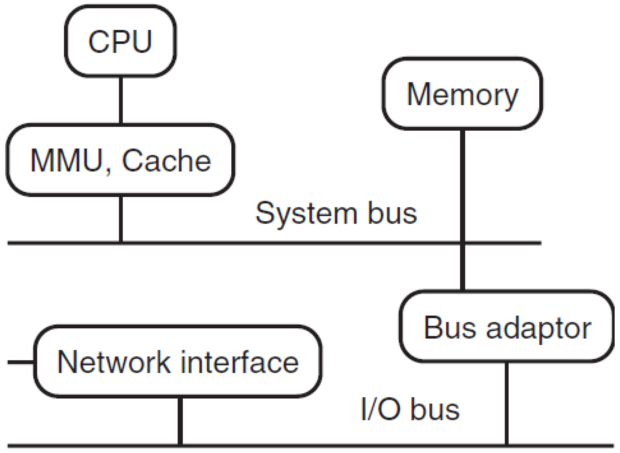
Node Architectures
Endnodes

Endnodes rely on caches that exploit

- temporal locality (same location is reused frequently)
- spatial locality (using a location is followed by using a nearby location)

Packet processing only shows spatial locality

Bus architecture limits the network-memory throughput



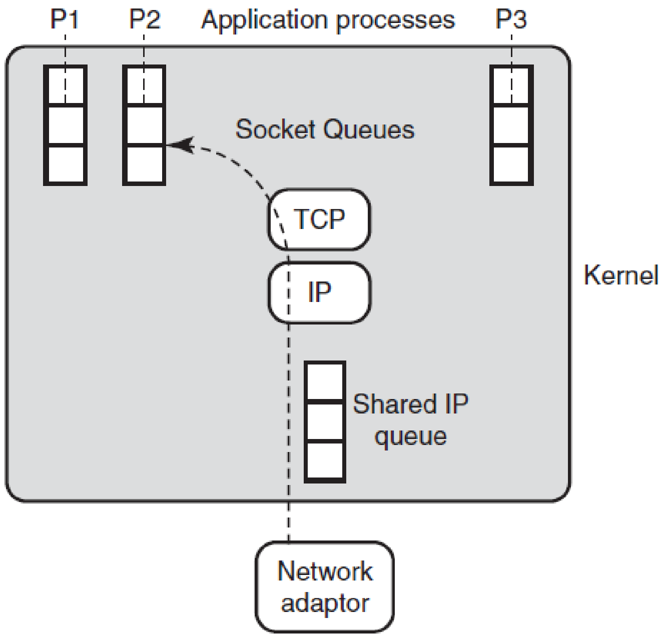
Endnodes
Processing of a Received Packets

Example from BSD UNIX

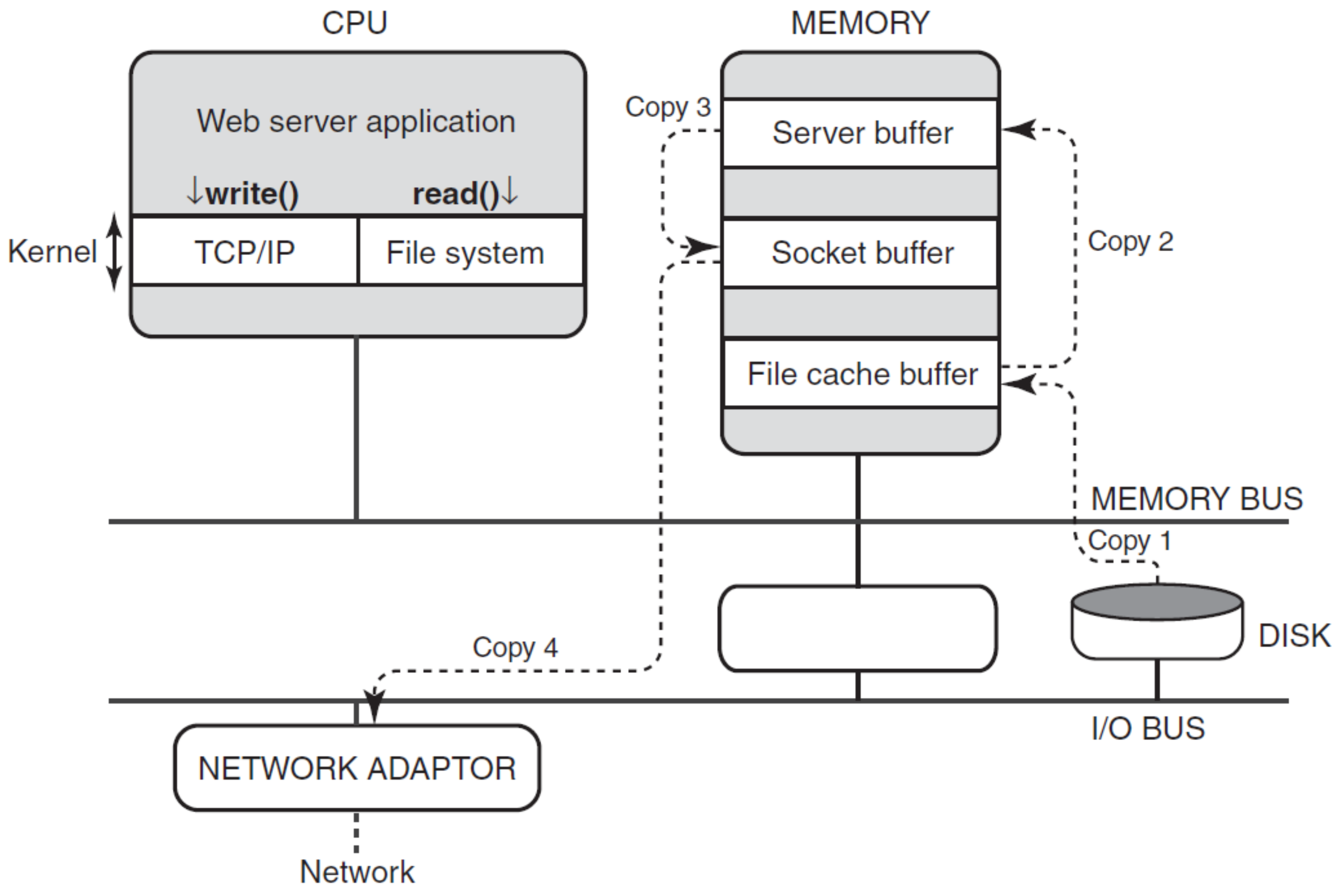
- The arrival of a packet generates an interrupt
- The interrupt handler copies the packet to a kernel queue and requests an OS thread (soft interrupt)
- The soft interrupt handler queues the packet in a socket queue and wakes a process

Receiver livelock problem

- When packets arrive too quickly handlers consume all the CPU, queues overflow and packets are lost



Drawbacks of Modularity: Multiple Copies



Avoiding Multiple Copies
Why multiple copies?

Multiple copies:

- Disk – cache
- Cache – application
- Application – kernel
- Kernel – adaptor (+TCP checksum calculation)

$\left.\begin{matrix} \\ \\ \\ \end{matrix}\right\}$ Disk - memory

$\left.\begin{matrix} \\ \end{matrix}\right\}$ memory – adaptor

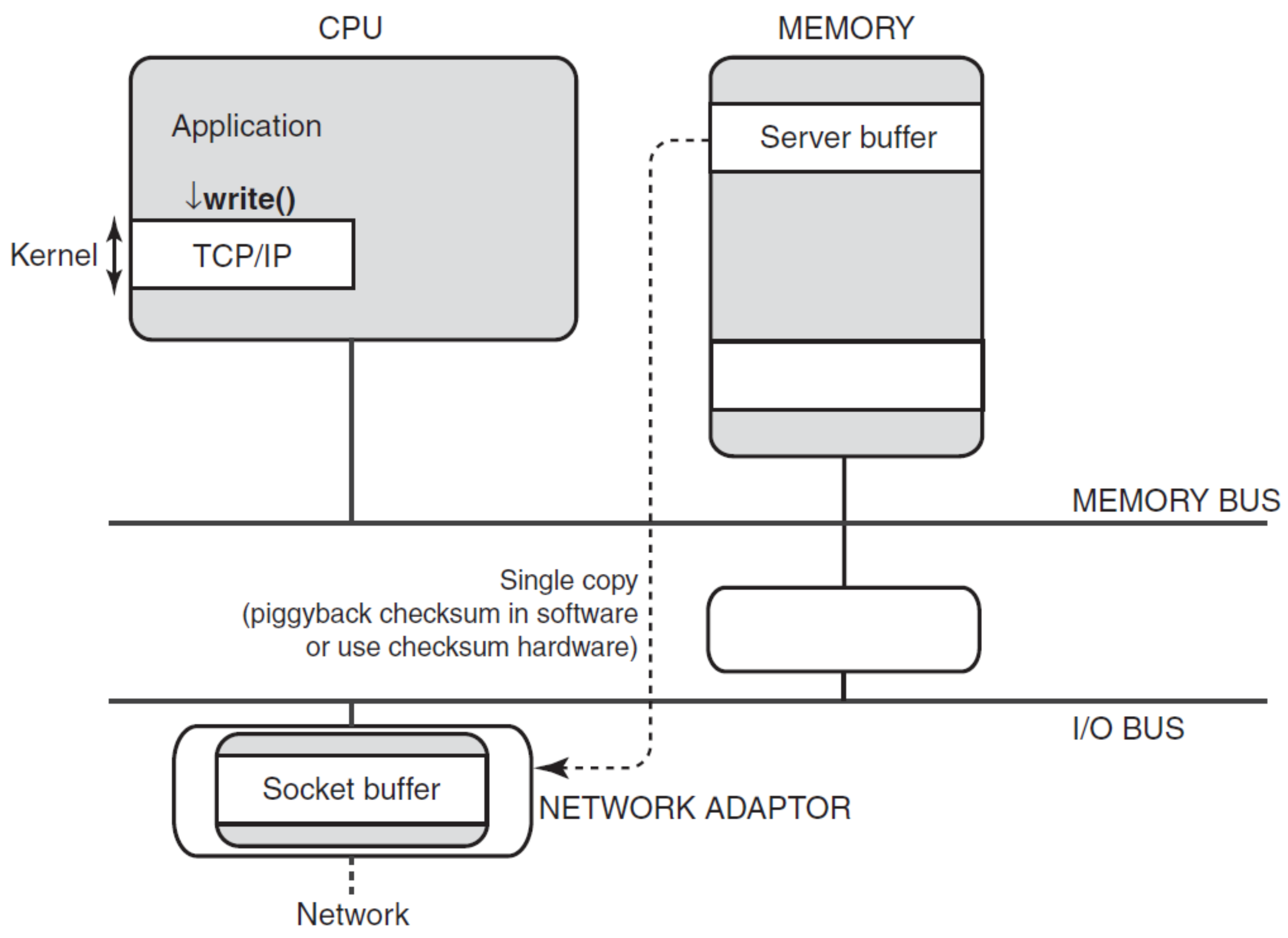
Consume

- Bus bandwidth
- Memory

Possible solutions:

- Use network adaptor memory
- Copy-on-write
- Remote DMA

Solution 1: exploit adaptor memory



Solution 1: exploit adaptor memory

Suppose OS uses memory mapped architecture.

Part of kernel memory can be in the adaptor.

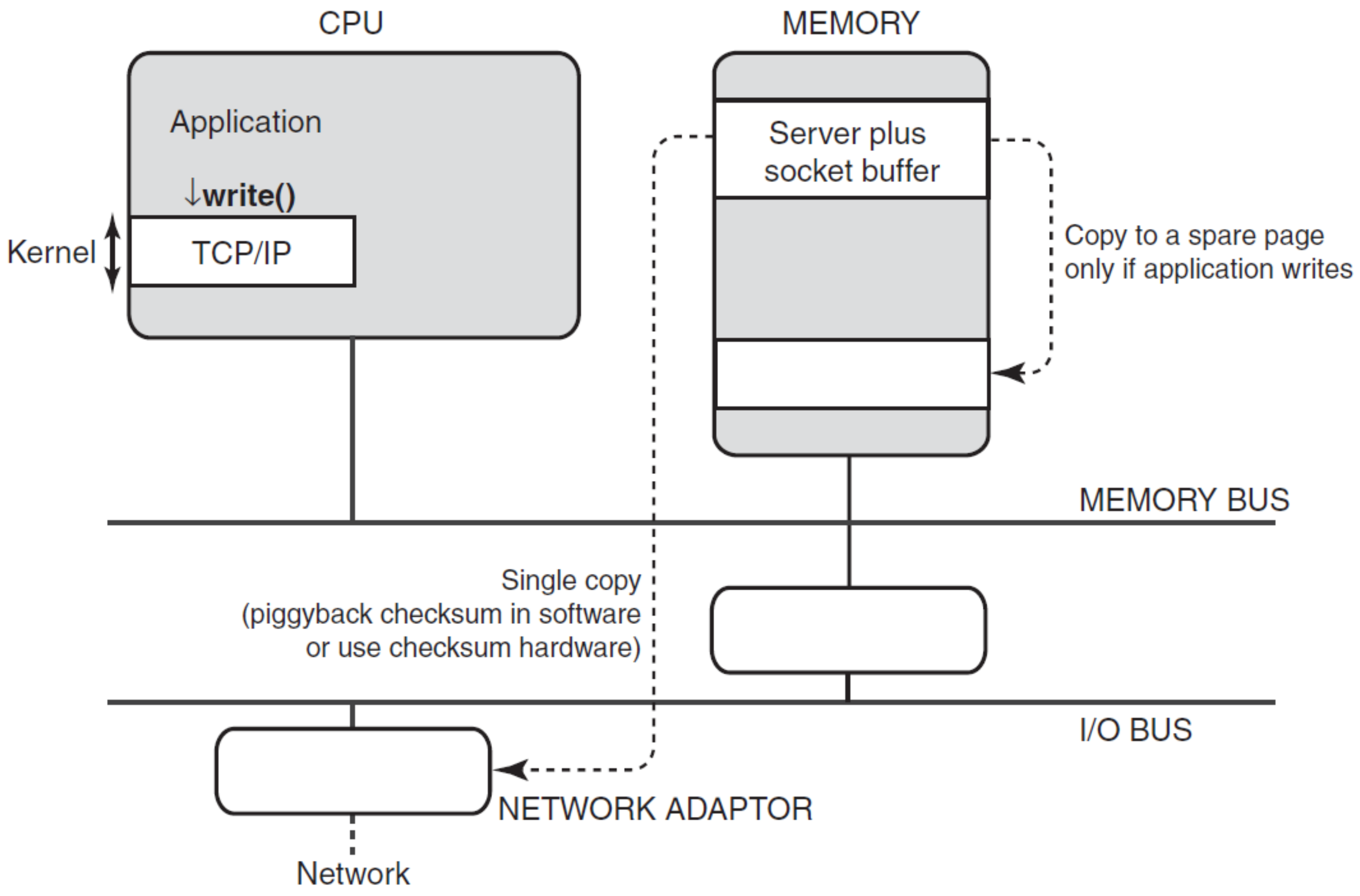
The checksum can be calculated on-the-fly:

- In case of PIO (Processor I/O) during the copy from memory to the (mamory mapped) adaptor memory
- In case of DMA (Directed Memory Access) by the adaptor receiving the data

Flaws:

- Lots of memory in the adaptor
- In reception, errored data could be delivered to the application
- In reception, data could be delivered to the application before it is acknowledged

Solution 2: Copy-on-write


CPU
Application
↓write()
Kernel
TCP/IP
MEMORY
Server plus socket buffer
Copy to a spare page only if application writes
MEMORY BUS
Single copy (piggyback checksum in software or use checksum hardware)
I/O BUS
NETWORK ADAPTOR
Network

Solution 2: Copy-on-write

Some OS offer copy-on-write (COW), allowing a process to replicate a virtual page at low cost.

- The copied page is a pointer to the same physical copy of the original page.
- If either virtual page is modified, then the OS generates a new physical copy. (This should be uncommon)

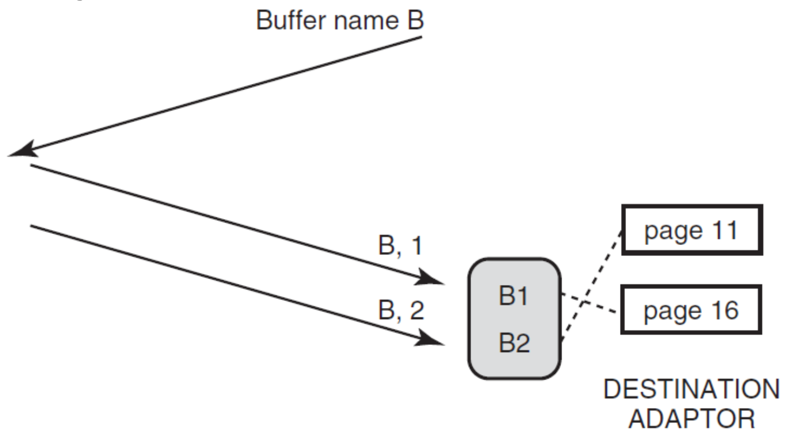
Problem: not available in UNIX and Windows

A similar solution implemented in Solaris using shared memories, but changes the classic UNIX copy semantics (i.e. the application can overwrite the buffer after *writing* it to the OS)

Solution 3: Remote DMA (RDMA)

Move data between two memories across the network without involvement of the CPUs. Two problems:

- How the receiving adaptor knows where to place the data
- How security is maintained.



Widely used in Storage Area Networks (SAN):

- Fiber Channel and Infiniband use proprietary protocols
- ISCSI uses TCP/IP. Requires TCP implemented in the adaptor.

Problem 1 - Buffer Validation of Application Device Channels

Application programs send network data through the OS kernel

- the process makes a system call
- the kernel copies data into kernel space
- the kernel processes the packet and copies it into the adaptor's memory

Receive is similar

How to avoid one copy

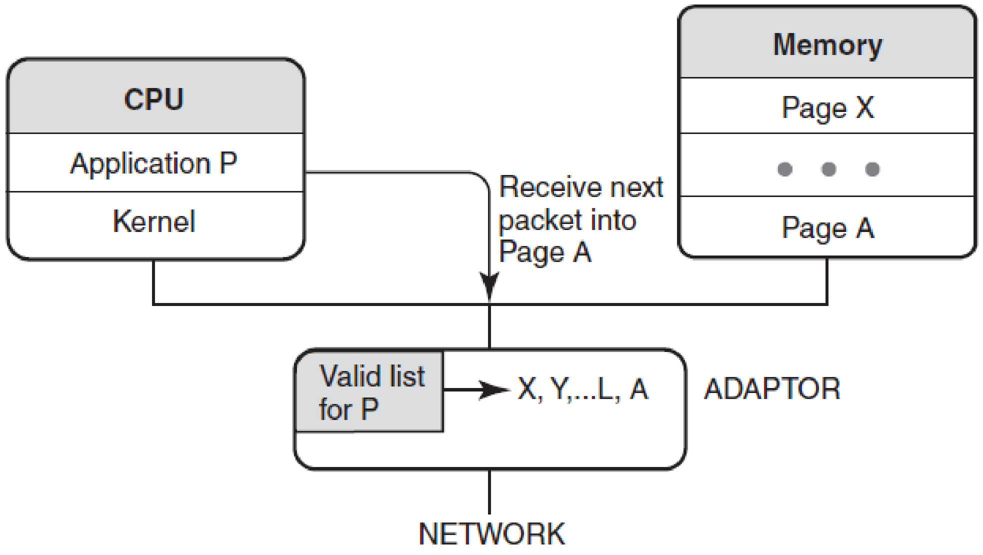
- ADCs allow applications to send/receive data directly to/from the adaptor
- The OS gives the adaptor a list of valid pages for each application P
- When P makes a request to send from/receive to page A, the adaptor must check if A is in the valid list

Problem 1 - Statement

When P makes a Receive into A, the adaptor must validate the page.

If the set of pages is a linear list, validation is O(n), where n is the number of pages

Can we make it faster?



Problem 1 - Hints

Apply the following principles:

- Use better data structures
- Pass hints in interfaces

Note that a hash table has two disadvantages:

- Calculating hash is expensive
- Worst case bound is not good

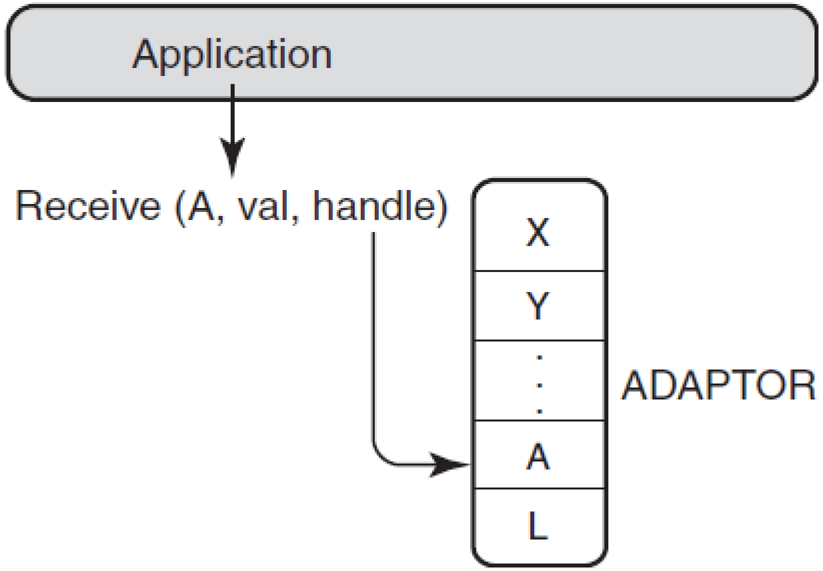
Problem 1 - Solution

The set of valid pages is stored as an array

The application passes an index to the array

Validation is made consists of

- Bounds checking
- Lookup
- Compare



Demultiplexing
Layered Demultiplexing

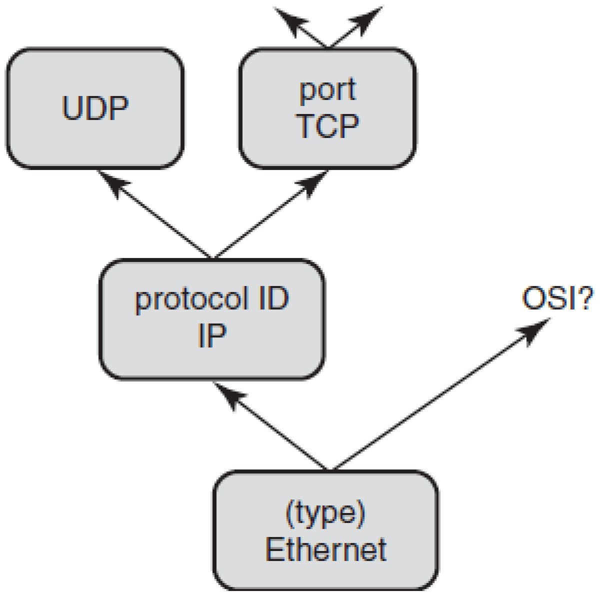
Traditional approach is layered

- lookup costs at each layer

Early demultiplexing finds the entire path in a pass.

Advantages

- prioritize packets
- use specialized code
- fast dispatch when layers are implemented in user space



Goals

Safety

- if each user program can specify the packets it wishes to receive, must protect against errors or malicious users

Speed

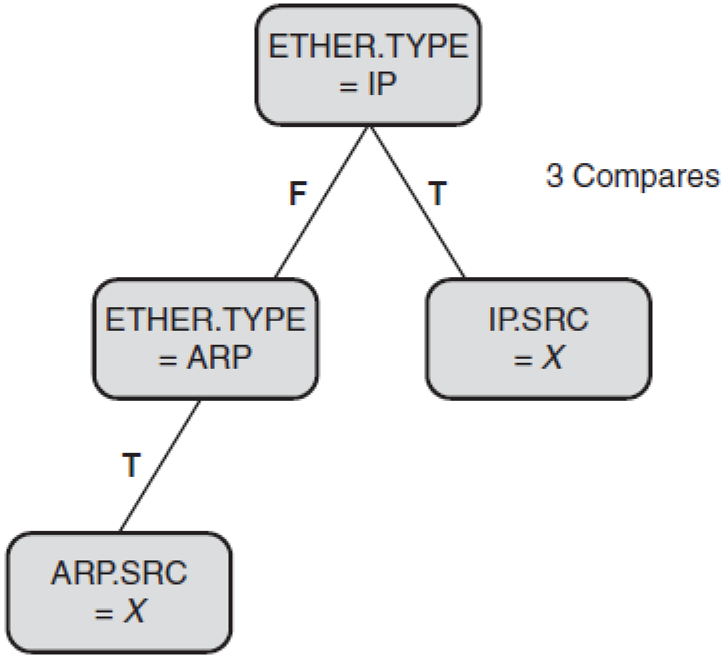
- wire speed processing

Composability

- if there are n filters, composition should be quicker than checking n filters

Berkeley Packet Filter (BPF)

Control Flow Graph model. Uses a register-based language.



BPF and Tcpdump

BPF is executed in the kernel

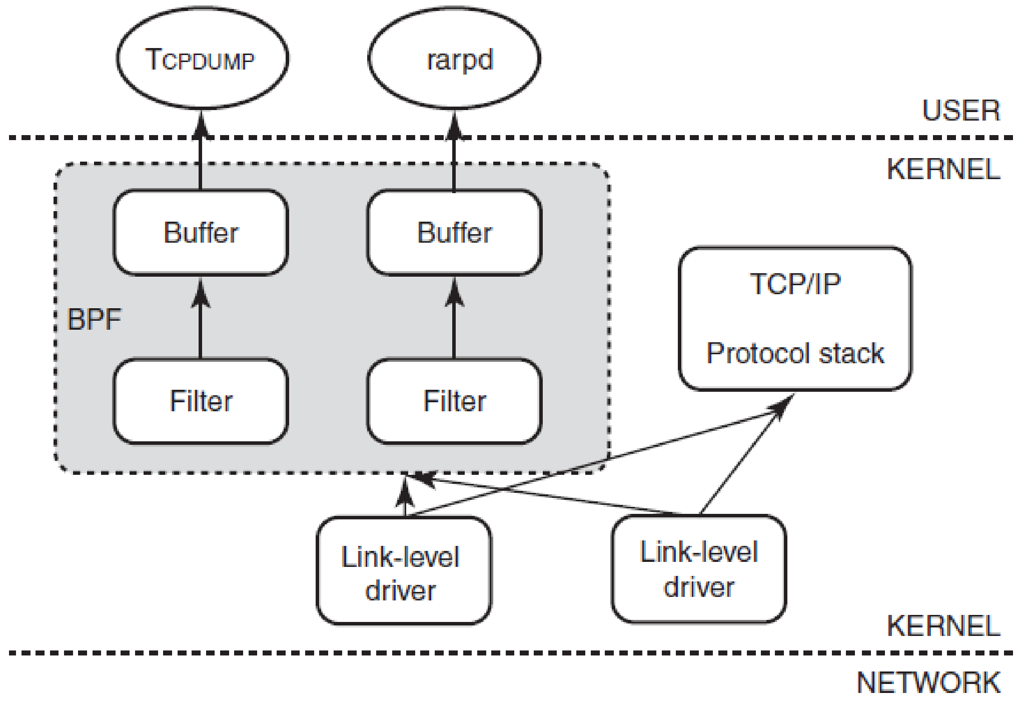
Receives packets from the network adaptor (usually configured to deliver all the packets regardless of MAC)

Packets are also delivered to the TCP/IP stack

The user-kernel interface is the pcap (packet capture API)

Several programs use the pcap API, tcpdump is the most known

BPF and Tcpdump



BPF and tcpdump

Performance considerations

- Filters before buffering, saves time when most packets are unwanted
- Packets are buffered waiting that the application fetches them
- Returns packets in batches

BPF filters and BPF machine

Filters are specified using the BPF syntax and compiled into machine code for the BPF virtual machine (pseudo-machine in the original 1993 paper)

The BPF machine consists of

•Accumulator

•Index register (x)

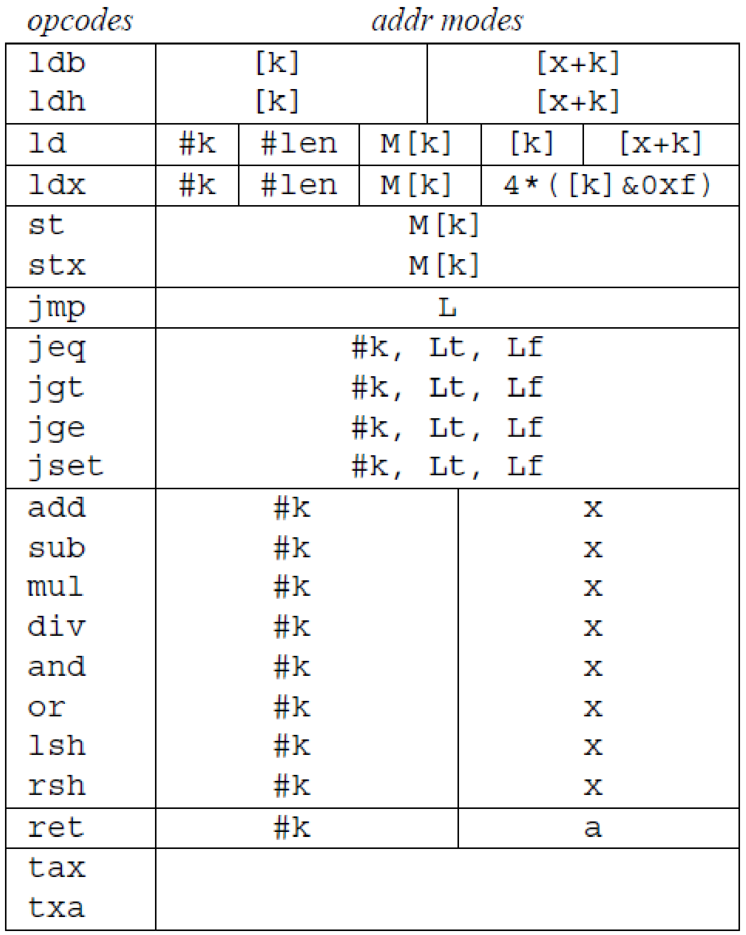
•Memory store

•Program counter

BPF machine instructions

- Load to accumulator or index register
  - immediate value
  - packet data at fixed or calculated offset
  - packet size
  - memory
- Store accumulator or index register to memory
- ALU instructions
- Branch based on comparison between constant or register and accumulator
- Return the size of the packet prefix to save (0 for discard)
- Miscellaneous (e.g. register transfer)

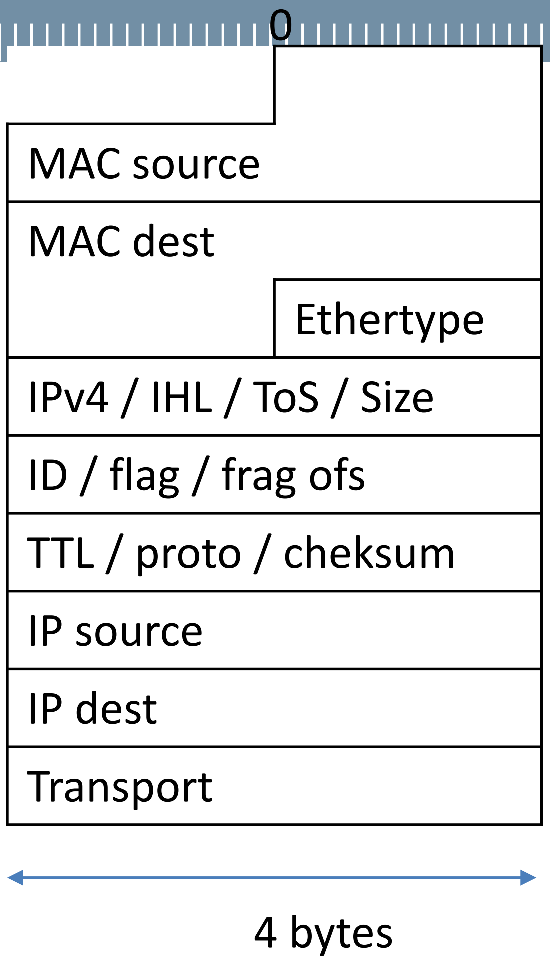
Set di instruzioni BPF



opcodes	*addr modes*				
ldb	[k]	[x+k]			
ldh	[k]	[x+k]			
ld	#k	#len	M[k]	[k]	[x+k]
ldx	#k	#len	M[k]	4*([k]&0xf)	
st	M[k]				
stx	M[k]				
jmp	L				
jeq	#k, Lt, Lf				
jgt	#k, Lt, Lf				
jge	#k, Lt, Lf				
jset	#k, Lt, Lf				
add	#k	x			
sub	#k	x			
mul	#k	x			
div	#k	x			
and	#k	x			
or	#k	x			
lsh	#k	x			
rsh	#k	x			
ret	#k	a			
tax					
txa					

Example: all packets to/from host 10.0.01

```
(000) ldh    [12]      Ethertype
(001) jeq    #0x800      IPv4    jt 2     jf 6
(002) ld     [26]       IP src
(003) jeq    #0xa000001          jt 12    jf 4
(004) ld     [30]
                       IP dst
(005) jeq    #0xa000001          jt 12    jf 13
(006) jeq    #0x806     ARP      jt 8     jf 7
(007) jeq    #0x8035    RARP     jt 8     jf 13
(008) ld     [28]
(009) jeq    #0xa000001          jt 12    jf 10
(010) ld     [38]
(011) jeq    #0xa000001          jt 12    jf 13
(012) ret    #65535
(013) ret    #0
```



Example: all packets to tcp port 80

```
(000) ldh      [12]   Ethretype
(001) jeq      #0x86dd  IPv6    jt 2     jf 6
(002) ldb      [20]
(003) jeq      #0x6 Next hdr    jt 4     jf 15
(004) ldh      [56]
(005) jeq      #0x50  TCP       jt 14    jf 15
(006) jeq      #0x800           jt 7     jf 15
(007) ldb      [23]
(008) jeq      #0x6             jt 9     jf 15
(009) ldh      [20]
(010) jset     #0x1fff          jt 15    jf 11
(011) ldxb     4*([14]&0xf)
(012) ldh      [x + 16]
(013) jeq      #0x50            jt 14    jf 15
(014) ret      #65535
(015) ret      #0
```

BPF syntax

An expression consists of 1+ primitives combined with logical operators

A primitive consists of 1+ qualifier and a name or number

Three kinds of qualifier

•type: host, net, port, portrange, …

•direction: src, dst, src or dst, src and dst

•protocol: ether, ip, ip6, arp, tcp, udp, …

To read a field in a protocol syntax is: proto[expr:size]

In addition some special keywords (e.g. broadcast) and arithemetic ops

Examples:

‘not ip’ : all non ip packets

‘ip[2:2] == 40’ : ip packets with size = 40

tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net *localnet*

tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'

Tcpdump and Wireshark

Tcpdump is command line packet analyzer that uses the pcap API

Prints on stdout:

•packet timestamp

•packet content (can parse most common protocols)

Can also save the packets to a file in the pcap file format

Wireshark is a GUI tool similar to TCPdump that can also parse most application layer protocols

Can also postprocess and filter the captured packets

•pros: filtering rules for application layer protcols

•cons: filtering in user space after capture

Pcap file format

A global header

- magic number and version number
- timezone
- timer accuracy
- max length of captured packets
- link type

For each packet an header and the packet data

- time stamp (seconds and microseconds)
- captured size
- original size