

## Protocol implementation

See Chapter 16 of Keshav & draft chapters by Varghese

Copyright © 2003, Tim Moors

## Outline

- Control flow: Partitioning: hardware/software, kernel/user-space, “interface among protocol layers” [Keshav]
- Data flow: Buffer management
- Header templates & prediction
- Protocol header and trailer design

Copyright © 2003, Tim Moors

## Hardware vs software partitioning

- Hardware: optics & electronics
  - Software: Programs to control hardware
- Grey areas in between, e.g. is a NIC that includes a processor “hardware” or “hardware+software”?

Occasionally, specialised processors are used for intensive higher-layer processing, e.g. compression, encryption TCP/IP, or parts thereof, are *sometimes* implemented in NIC hardware to reduce CPU processing burden for high-speed links. e.g. Gigabit Ethernet NICs may calculate checksum so that CPU doesn't need to, TCP/IP Offload Engines (e.g. Adaptec ANA-7711) implement all (or almost all) of TCP/IP Network layer and above *usually* implemented as software on Central Processing Unit. Network processors in routers. Link layer *often* implemented in hardware (Network Interface Card) + software (driver to interface to NIC to computer) Physical layer *must* have some hardware components. May also have software (e.g. channel quality monitoring & Software Defined Radios)

Application
Presentation
Session
Transport
Network
Data link
Physical

Copyright © 2003, Tim Moors

## Kernel vs user-space implementation

Operating systems split system memory into:

- **the kernel** (mediates access to resources, trusted, reliable; primitive & hairy to develop)
- **user-space** (applications, sophisticated, frequently changed)

Protocols can be implemented in either kernel or user space.

- Kernel has high performance (little state to preserve when incoming packet causes context switch) & is secure (end-user can't change protocols, e.g. disable TCP's congestion control)
- User-space implementations: Easy to customise & develop

Kernel usually implements link, network & transport layers; sockets interface to applications that implement higher layers (e.g. HTTP)

Copyright © 2003, Tim Moors

## Buffer management

- In theory, layers are independent: When a layer processes a packet, it stores that packet in its own buffer.
- In practice, multiple layers are implemented on one processor & copying information between separate buffers is wasteful.
- Most implementations use “buffer cut-through”: packet remains static in memory, and layers exchange the packet by passing pointers towards it.

Copyright © 2003, Tim Moors

## Linux Socket buffers: `sk_buff` (`skb`)

Implementations of protocols under Linux convey information between layers by using `sk_buff`s

`sk_buff` structures (details on next slide) contain:

- pointers to storage: `head`, “data” (packet, including encapsulation), `tail`.
- pointers to protocol headers (`h`, `nh`, `mac`)
- `len`: convey length between adjacent layers, e.g. IP to TCP
- `checksum`: Carries checksum computed in NIC to transport layer

[Cox]

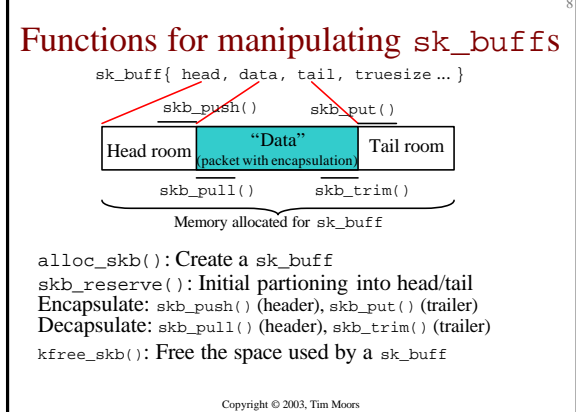
Copyright © 2003, Tim Moors

```

struct sk_buff {
    struct sk_buff *next, *prev; // Link buffers to form lists
    struct sk_buff_head *list; /* List we are on */
    ...
    union { struct tcp_hdr *th; ... } h; /* Transport layer header */
    union { ... } nh; /* Network layer header */
    union { ... } mac; /* Link layer header */
    ...
    unsigned int len; /* Length of actual data */
    unsigned int data_len;
    unsigned int csum; /* Checksum */
    ...
    unsigned char ip_summed; /* Driver fed us an IP checksum */
    ...
    unsigned int truesize; /* Buffer size */
    unsigned char *head; /* Head of buffer */
    unsigned char *data; /* Data head pointer */
    unsigned char *tail; /* Tail pointer */
    unsigned char *end; /* End pointer */
    void (*destructor)(struct sk_buff *); /* Destruct function */
};

```

*The structure above is for kernel version 2.4.7, defined in <linux/skbuff.h>. Ellipses in red (...) and // comments added by Tim Moors*



### Why information may still need to be copied...

Application may:

- Prepare data before it determines that it should be sent on the network (e.g. editor used for word processing & email) & so be unable to use special network buffers.
- Expect to be able to change memory containing data that it has asked to be sent across the network.

Kernel may need to:

- Keep information for longer than the application, e.g. TCP retransmissions after application closes.
- Add/remove encapsulation, or perform segmentation/reassembly
- Process received packets to determine which app they are for

Options:

- Move in CPU's memory space
- Move to network interface card (limited storage) [Afterburner]

Copyright © 2003, Tim Moors

### Data "touching" operations

Memory technologies have increased size (about 60% p.a.) but not speed (about 4% p.a., c.f. fibre 200% p.a.)  
 ⇒ Want to minimise number of times payload is "touched" (read/written)

Operations that need to access each bit being transmitted, rather than just controlling groups of bits (e.g. control operations & reassembly), e.g.:

- copying
- integrity checks
- encryption
- compression

**Integrated Layer Processing (ILP):** Perform data touching functions from multiple layers whenever accessing memory, e.g.

- read-compress-encrypt-integrity-write, rather than
- read-compress-write ... read-encrypt-write ... read-integrity-write

Copyright © 2003, Tim Moors

### Data unit structuring

Copyright © 2003, Tim Moors

### Common protocol fields

- Version number
- Addressing / demultiplexing
- Length
- Integrity check
- Payload(!)

- Protocols above the application layer tend to have more complicated structures, e.g. HTTP, whose variability doesn't lend themselves to simple analysis!!!!

Copyright © 2003, Tim Moors

## Ethernet

Preamble	Destination Address	Source Address	Type	(Data)	(Pad)	CRC
8B	6B	6B	2B	0-1500B	0-46B	4B

No explicit version number;  
 Redundancy in "Type" distinguishes DIX from 802.3  
 Length is determined by valid CRC since preamble & loss of signal  
 Receiver checking: source isn't multicast, 64 = length = 1518, pad==0

Copyright © 2003, Tim Moors

## IEEE 802.11 Wireless LANs

Ver	Typ	Subtype	Dir	F	R	P	D	W	O	Duration
Destination Address										
Source Address										
Basic Service Set Identifier										
Frag#					Sequence number					
Payload										
Frame Check Sequence										

Dir: Direction: 2 flags indicating to/from Distribution System  
 F = More fragments, R = Retry, P = Power management, D = More data, W = WEP, O = Order

- 1<sup>st</sup> word: version, type of frame (influences format; some include 4<sup>th</sup> address; some types reserved), flags & duration (for virtual carrier)
- Addresses start in 2<sup>nd</sup> word, DA 1<sup>st</sup>

Copyright © 2003, Tim Moors

## Internet Protocol, v4 (1974)

TTL → IPv6 Hop Limit      TOS → IPv6 "Traffic Class"

Version	IHL	Type of Service PRECE.D F R 0 0	Total Length	Identification	Flags 0 DF MF	Fragment Offset	Header Checksum
Source Address							
Destination Address							
Options							
Padding							

IPv6 uses e2e fragmentation

Copyright © 2003, Tim Moors

## Internet Protocol, v6 (1995)

Version	Traffic Class	Flow Label	Next Header	Hop Limit
Payload Length				
Source Address ... (128b)				
Destination Address ... (128b)				

- Next header may be an IP option or upper layer

Assumes upper layer has a checksum with pseudoheader including SA/DA

Copyright © 2003, Tim Moors

## Pseudoheaders

Not all information covered by an integrity check value (ICV, e.g. CRC) need be transmitted

e.g. 1: authentication system:

- Source: preface packet with a secret key, calculate ICV over key+packet, transmit packet+ICV
- Receiver: calculates ICV over key+received packet, compare to received ICV, difference indicates source doesn't know key or payload contains errors ⇒ discard

e.g. 2: TCP/UDP: End-systems calculate checksum over TCP/UDP segment prefaced by a pseudoheader containing important IP fields: SA&DA, protocol & segment length. If these fields in the IP header contain errors, then they will be detected at the transport layer

Zero	Source Address	Destination Address	Protocol	Length
...				

pseudoheader  
header

Copyright © 2003, Tim Moors

## Transmission Control Protocol

Source Port	Destination Port		
Sequence Number			
Acknowledgment Number			
Data Offset	Reserved	U A P R S F R C S S W T G K H T N N	Window
Checksum		Urgent Pointer	
Options		Padding	
data			

No version number. Variants are distinguished by the use of options  
 Length information is carried in parallel with segment, e.g. in sk\_buff  
 Data offset ~ IPv4 IHL ⇒ length of options  
 Options are padded for word alignment of data  
 Urgent/Ack not needed ∀ segments, but included (could be options)

Copyright © 2003, Tim Moors

## Principles of formatting data units

- Formatting reflects processing order
- Formatting to conserve processing
- Optimise for the common case
- Simplify receiver processing

Copyright © 2003, Tim Moors

## Formatting reflects processing order

**Packet headers:** Locating addresses, protocol & other identifiers at the beginning of the packet allows a receiver to process these identifiers first, determine where to buffer the payload, and then store the payload in the appropriate place.

Good examples:

- Ethernet DA before SA, addresses before type
- TCP: port numbers before flags

Counterexample: IP header: Version number needed first, DA/SA

**Packet trailers:**

- Integrity checks in trailer (e.g. Ethernet) allow check to be computed as payload is transmitted & check is then appended at end. Check in header (e.g. TCP/UDP) can use otherwise wasted bits & avoid need for trailer
- Length in trailer helps source (may not know length until run out of payload to send) but hinders destination (needs to know length to determine size of buffer to allocate to payload)

Copyright © 2003, Tim Moors

## Formatting to conserve processing

- Align fields on word boundaries
- e.g. TCP has 32b alignment: 32b fields on separate

c.f. formatting to conserve bandwidth: Eliminate reserved bits.

Copyright © 2003, Tim Moors

## Optimise for the common case

Try to make the most common packet headers simple and small to maximise speed/efficiency.

Relegate less common features to options

Good example: IP options: record-route, timestamps, source routing rarely used  $\Rightarrow$  include this information in options, only when needed

- IPv6 orders options so that the most commonly processed ones (relating to routers) are at the beginning, making their location easier to predict. Less commonly used ones are located at the end of the list.

Bad example: TCP: Not all segments carry acknowledgements or urgent payload. TCP includes these fields (32b ack, 16b urgent ptr) in all segments & has flags to indicate if they are used. Could re-design using options. Particularly for rare urgent feature (acks are relatively common)

Copyright © 2003, Tim Moors

## Simplify receiver processing

- Sources exert control by initiating events; receivers have to respond to those events  $\Rightarrow$  reception is complicated by less predictability
- Receivers also have to check for abnormalities in protocol fields, e.g. IP header: Is header length valid? Is reserved flag=0? Is SA a unicast address?
- Receiver processing can be the bottleneck
  
- Good example: ATM VCIs, VPIs: downstream switch selects them, allowing simple table lookup, rather than being selected by upstream switch
  
- Counterargument: some protocols require extensive maintain processing at the sender, e.g. TCP – RTT estimation etc [Clark]

Copyright © 2003, Tim Moors

## References

[Afterburner] C. Dalton et al: "Afterburner. A network-independent card provides architectural support for high performance protocols", *IEEE Network*, July 1993

[Varghese]

[Cox] A. Cox: "Network buffers and memory management", *Linux Journal*(29), Sep. 1996

[Clark] D. Clark et al: "An analysis of TCP processing overhead", *IEEE Communications Magazine*, 27(6):23-9

[ILP] D. Clark and D. Tennenhouse: "Architectural considerations for a new generation of protocols", *Proc. SIGCOMM 90*, pp. 200-8

Copyright © 2003, Tim Moors