CSCI-GA.2250-001

Operating Systems

Networking

Hubertus Franke
frankeh@cs.nyu.edu

Why we need networking ?

- Everything is now connected !!!
- Everything is an online service now
  - (whatsapp, facebook, Netflix, tempsensor, Ring)
- IoT (internet of things) exploding
- Folks are constantly online

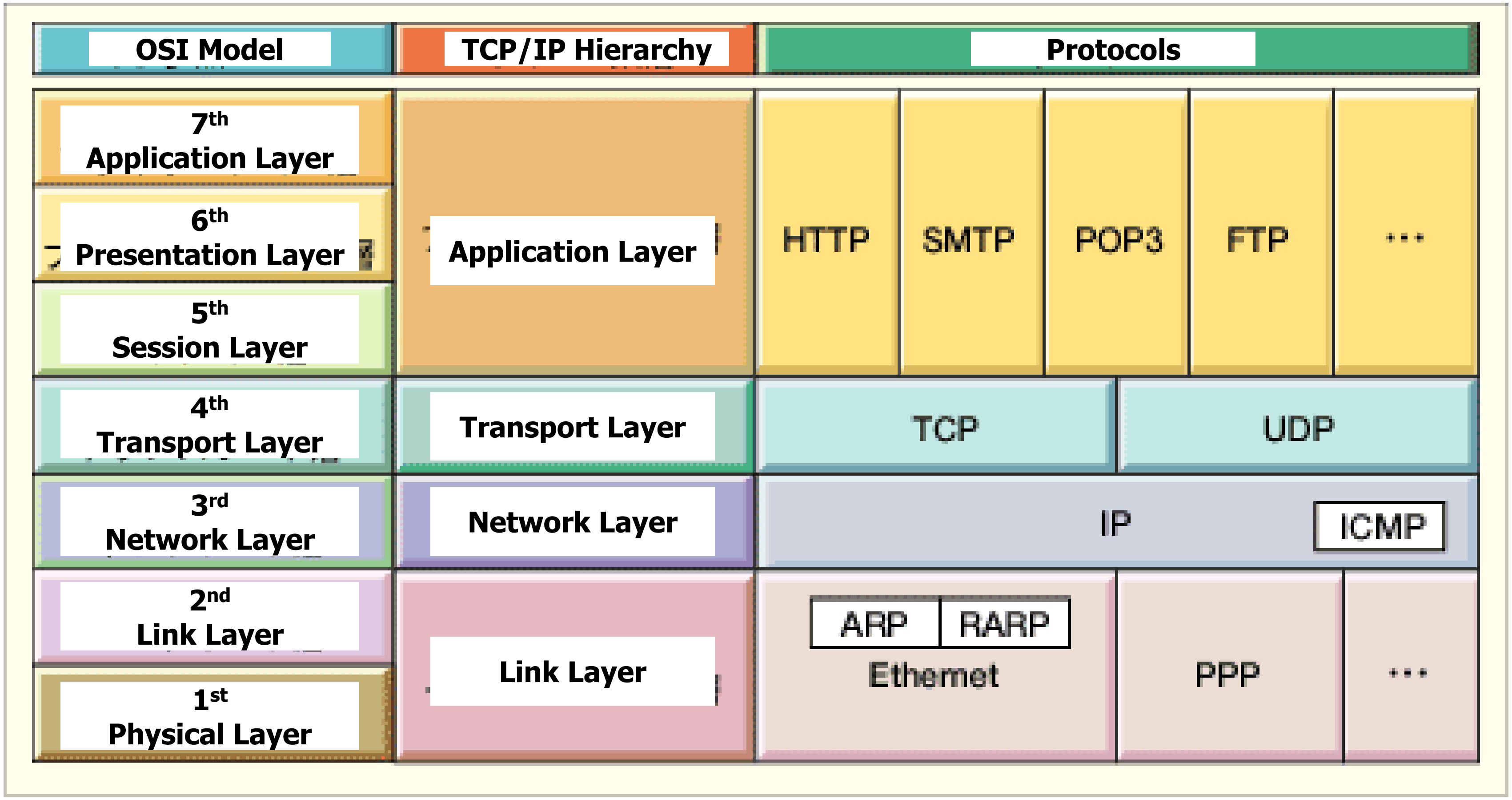
What is a “internet”?

- A set of *inter*connected *net*works
- **The** Internet is the most famous example
- Physical networks can be completely different technologies:
  - Ethernet, ATM, modem, …
- *Routers* (nodes) are devices on multiple networks that pass traffic between them
- Individual networks pass traffic from one router or endpoint to another
- Each endpoint has a unique MAC address (Media Access Control / 48-bits)
- But, its more about the services that are consumed over the network than the technology building the network
- **TCP/IP is what links them together.**

TCP/IP protocol family

- TCP/IP hides the details as much as possible
- IP : Internet Protocol
  – UDP : User Datagram Protocol
    - RTP, traceroute
  – TCP : Transmission Control Protocol
    - HTTP, FTP, ssh

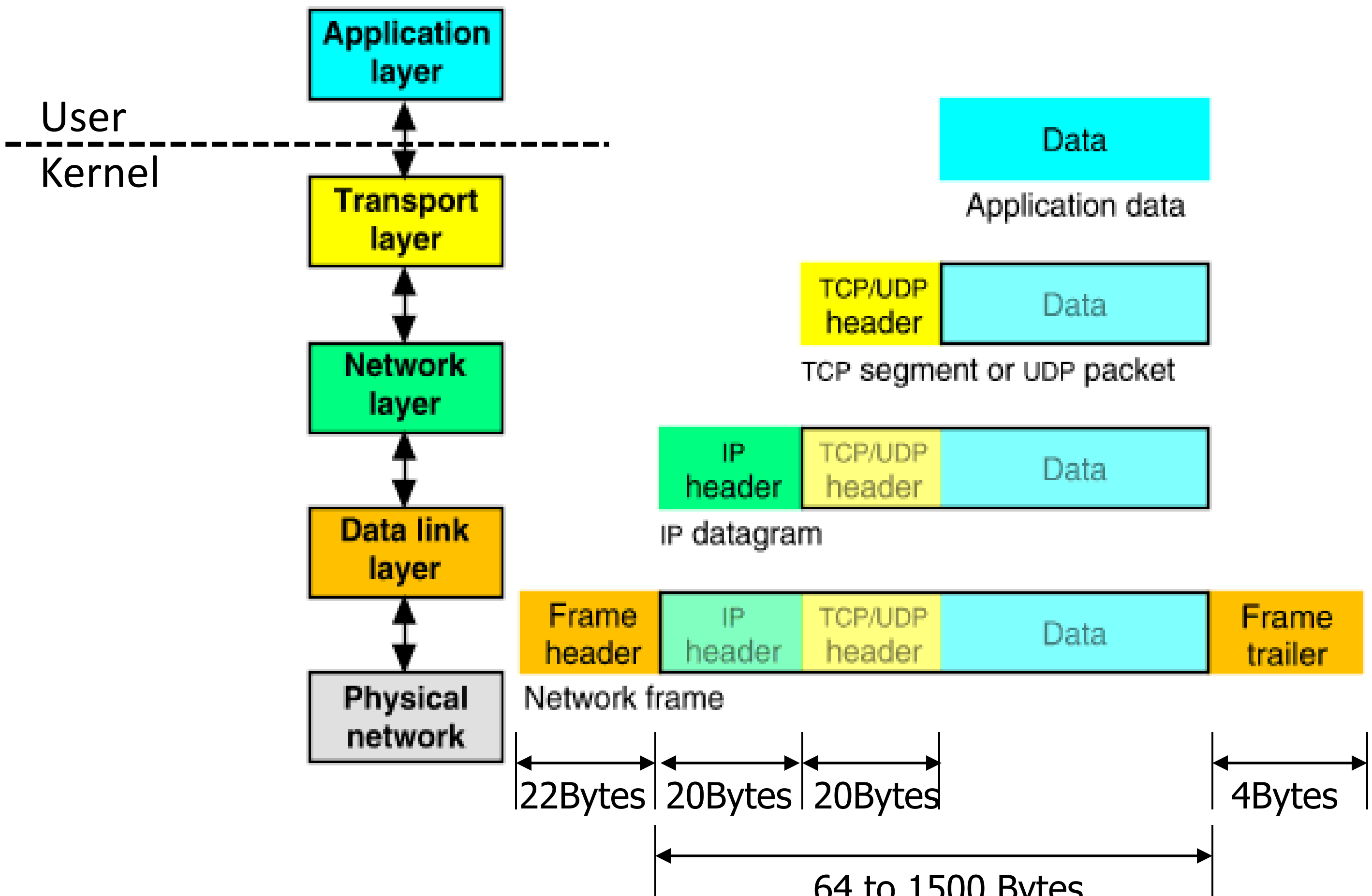
OSI and Protocol Stack

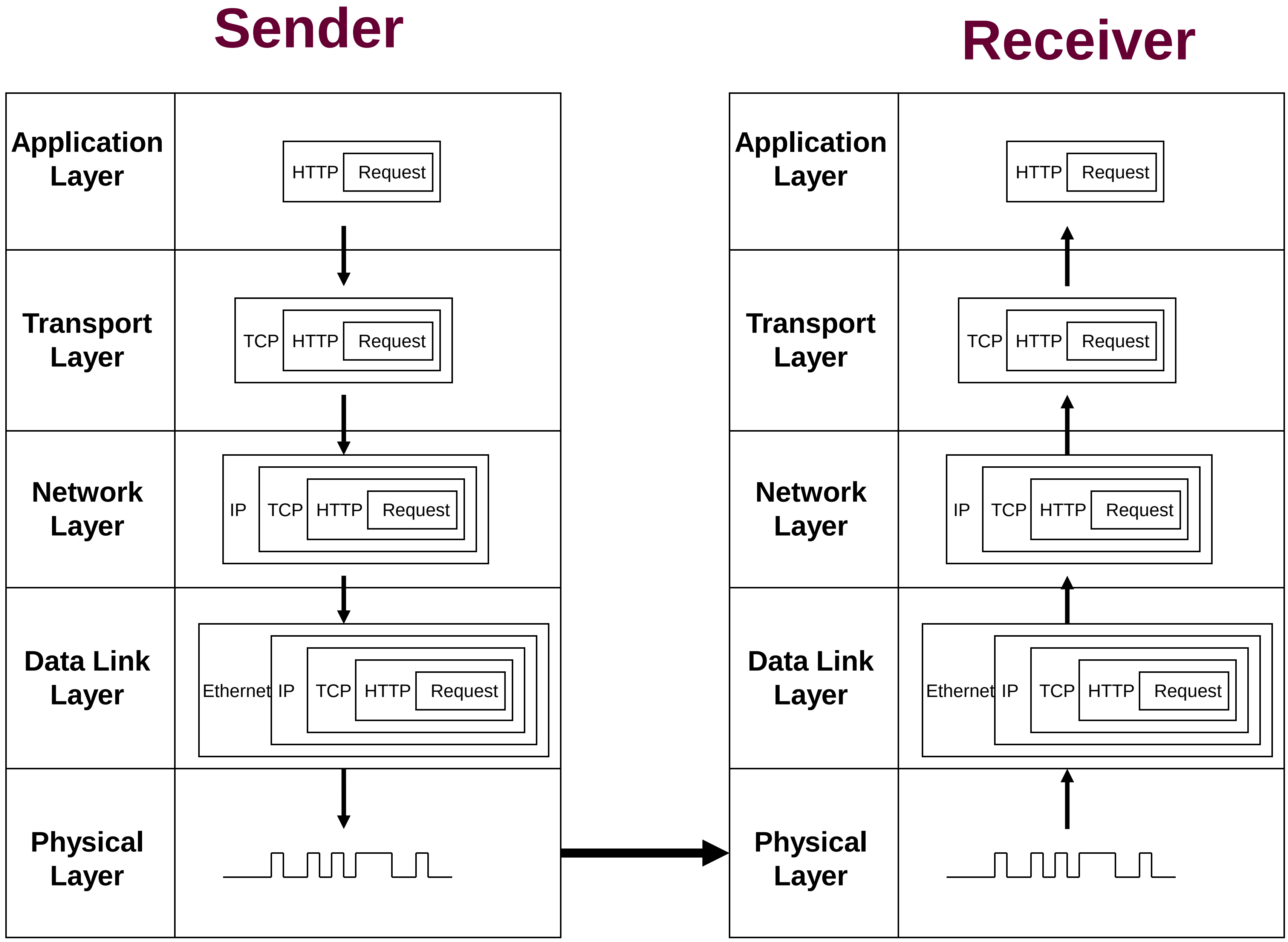


Link Layer : includes device driver and network interface card
Network Layer : handles the movement of packets, i.e. Routing
Transport Layer : provides a reliable flow of data between two hosts
Application Layer : handles the details of the particular application

Packet Encapsulation

- The data is sent down the protocol stack
- Each layer adds to the data by prepending headers




Sender
Receiver
Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer
HTTP
Request
TCP
IP
Ethernet

IP

- Responsible for end to end transmission between nodes/machines
- Sends data in individual packets
- Maximum size of packet is determined by the networks and devices
  - Packet is **fragmented** if it is too large (can be hardware or software)
  - MTU: maximum transmission unit (aka packet size)
- Unreliable
  - Packets might be lost, corrupted, duplicated, delivered out of order

IP addresses

- 4 bytes
  - e.g. 163.1.125.98
  - Each device normally gets one (or more)
  - In theory there are about ***4 billion*** available

- You can have private networks so there can be replication.

  - 10.0.0.0 … 10.255.255.255 (256^3 = 16M devices)
  - 172.16.0.0 … 172.31.255.255 (16*256^2 = 1M devices)
  - 192.168.0.0 … 192.168.255.255 (256^2 = 65K devices)

- To get out of a private network you need access to an IP outside those ranges or bridge via NAT (Network address translation) or a gateway

Allocation of IP addresses

- Controlled centrally by ICANN
  - Fairly strict rules on further delegation to avoid wastage
    - Have to demonstrate actual need for them
- Organizations that got in early have bigger allocations than they really need

IPv6

- Created due to the IPv4 limitations (2^32) ➔ now 128 bit addresses
  - Make it feasible to be very wasteful with address allocations and assign each device its unique IPv6 id.
- Many other new features
  - Built-in autoconfiguration, security options, …
- Slowly entering into production use yet:

  Google's statistics show IPv6 usage at ~22% and US at 32% times

- Time comparison for a simple “curl command” (2016)

NEW YORK

IPv4 x IPv6 Connection/Total Time Comparison

DOMAIN	CONNECT TIME		TOTAL TIME	
	IPv4	IPv6	IPv4	IPv6
GOOGLE	.031 sec	.030 sec	.061 sec	.055 sec
FACEBOOK	.062 sec	.055 sec	.100 sec	.085 sec
YOUTUBE	.030 sec	.003 sec	.063 sec	.056 sec
WIKIPEDIA	.036 sec	.035 sec	.043 sec	.042 sec
NETFLIX	.036 sec	.068 sec	.050 sec	.085 sec
LINKEDIN	.035 sec	.037 sec	.042 sec	.044 sec
PANDORA	.102 sec	.092 sec	.176 sec	.158 sec
CLOUDFLARE	.029 sec	.030 sec	.035 sec	.033 sec
SUCURI	.035 sec	.035 sec	.041 sec	.042 sec

IP packets

- Source and destination addresses
- Protocol number
  - 1 = ICMP, 6 = TCP, 17 = UDP
- Various options
  - e.g. to control fragmentation
- Time to live (TTL)
  - Prevent routing loops

IP Datagram

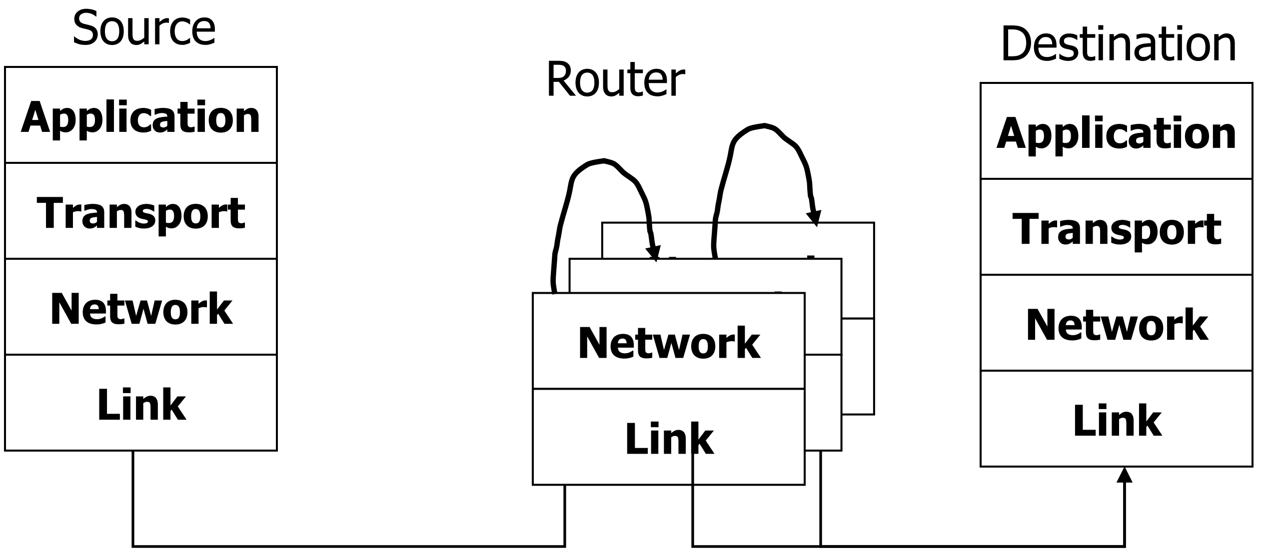
<table>
<tr><td>0</td><td>4</td><td>8</td><td>16</td><td>19</td><td>24 31</td></tr>
<tr><td>Vers</td><td>Len</td><td>TOS</td><td colspan="3">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Flags</td><td colspan="2">Fragment Offset</td></tr>
<tr><td colspan="2">TTL</td><td>Protocol</td><td colspan="3">Header Checksum</td></tr>
<tr><td colspan="6">Source Internet Address</td></tr>
<tr><td colspan="6">Destination Internet Address</td></tr>
<tr><td colspan="5">Options...</td><td>Padding</td></tr>
<tr><td colspan="6">Data...</td></tr>
</table>

Field	Purpose
Vers	IP version number
Len	Length of IP header (4 octet units)
TOS	Type of Service
T. Length	Length of entire datagram (octets)
Ident.	IP datagram ID (for frag/reassembly)
Flags	Don’t/More fragments
Frag Off	Fragment Offset

Field	Purpose
TTL	Time To Live - Max # of hops
Protocol	Higher level protocol (1=ICMP, 6=TCP, 17=UDP)
Checksum	Checksum for the IP header
Source IA	Originator’s Internet Address
Dest. IA	Final Destination Internet Address
Options	Source route, time stamp, etc.
Data...	Higher level protocol data

We only looked at the IP addresses, TTL and protocol #

IP Routing

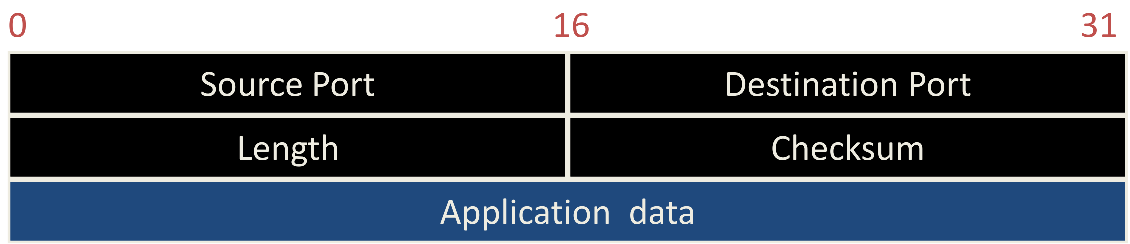


- Routing Table
  - Destination IP address
  - IP address of a next-hop router
  - Flags
  - Network interface specification

UDP
(User Datagram Protocol)

- Thin layer on top of IP
- Adds packet length + checksum
  - Guard against corrupted packets
- Also source and destination *ports*
  - Ports are used to associate a packet with a specific application at each end
- Still unreliable:
  - Duplication, loss, out-of-orderness possible
- Allows multiplexing over IP

UDP datagram



Field	Purpose
Source Port	16-bit port number identifying originating application
Destination Port	16-bit port number identifying destination application
Length	Length of UDP datagram (UDP header + data)
Checksum	Checksum of IP pseudo header, UDP header, and data

Typical applications of UDP

- Where packet loss etc is better handled by the application than the network stack
- Where the overhead of setting up a connection isn’t wanted

- VOIP
- NFS – Network File System
- Most games

TCP

- Reliable, *full-duplex*, *connection-oriented*, *stream* delivery
  - Interface presented to the application doesn't require data in individual packets
  - Data is guaranteed to arrive, and in the correct order without duplications
    - Or the connection will be dropped
  - Imposes significant overheads
- Allows multiplexing of IP

Applications of TCP

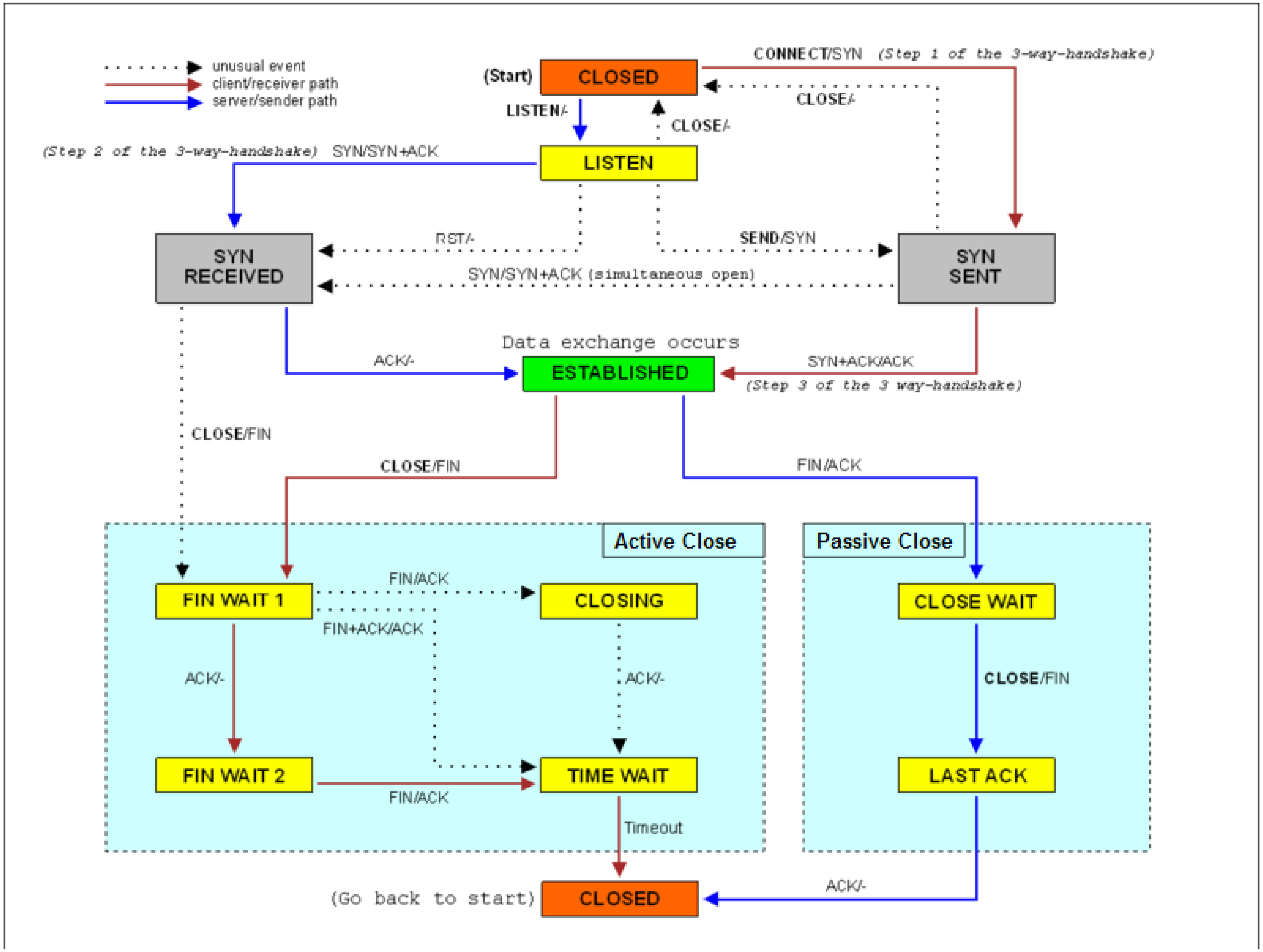
- Most things!
  - HTTP, FTP, …

- Saves the application a lot of work, so used unless there’s a good reason not to

TCP implementation

- Connections are established using a *three-way handshake*
- Data is divided up into packets by the operating system
- Packets are numbered, and received packets are acknowledged
- Connections are explicitly closed
  - (or may abnormally terminate)

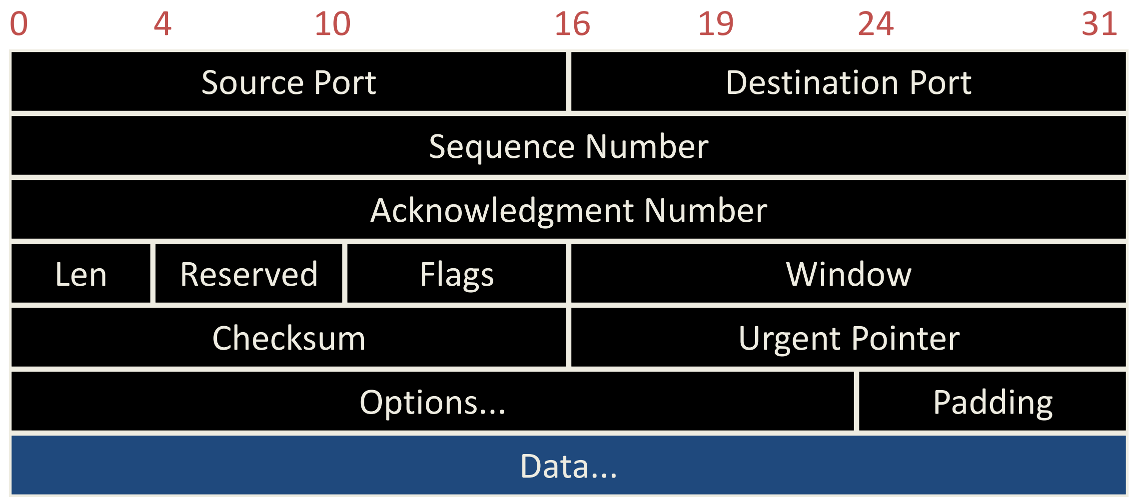
TCP connection state diagram



TCP Packets

- Source + destination ports
- Sequence number (used to order packets)
- Acknowledgement number (used to verify packets are received)

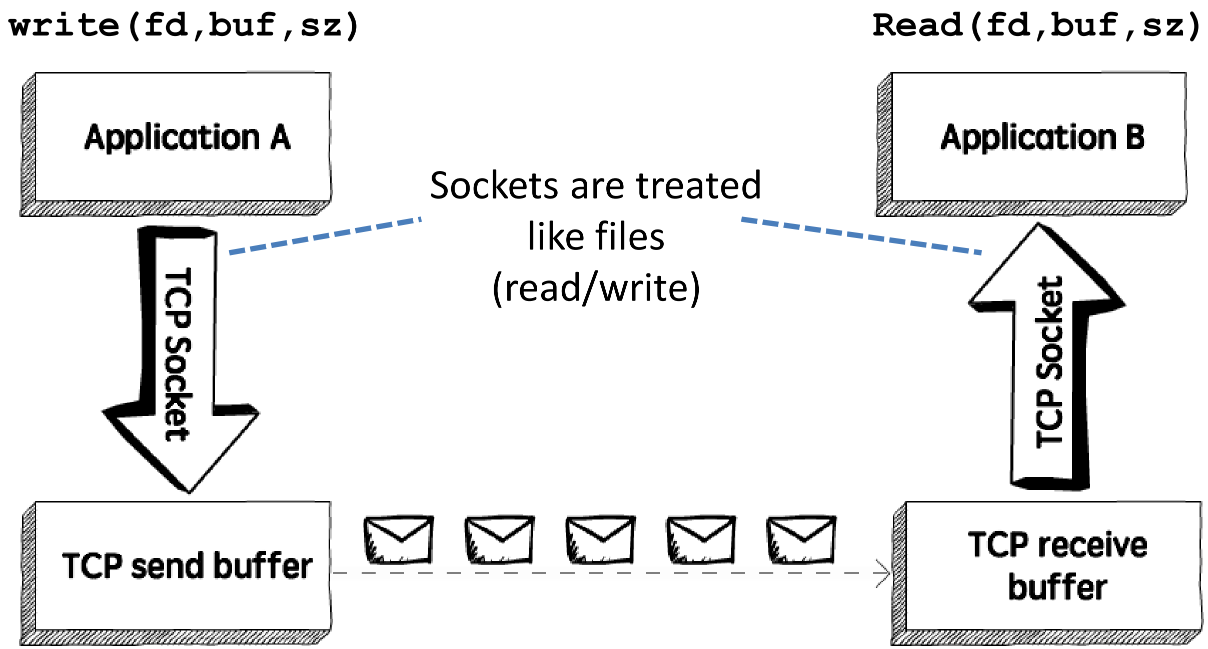
TCP Segment



Field	**Purpose**
Source Port	Identifies originating application
Destination Port	Identifies destination application
Sequence Number	Sequence number of first octet in the segment
Acknowledgment #	Sequence number of the next expected octet (if ACK flag set)
Len	Length of TCP header in 4 octet units
Flags	TCP flags: SYN, FIN, RST, PSH, ACK, URG
Window	Number of octets from ACK that sender will accept
Checksum	Checksum of IP pseudo-header + TCP header + data
Urgent Pointer	Pointer to end of “urgent data”
Options	Special TCP options such as MSS and Window Scale

You just need to know port numbers, seq and ack are added

Basics of TCP/IP



- How do we make sure data is not lost and delivered in order ?
- We want to utilize available bandwidth (changing over time due to other users)
- How much can we send? If application doesn't consume we must buffer (limited)
- How to throttle traffic on ingestion, often your CPU (app) can produce more network data (e.g. 200Gbps) than your network card can handle (10Gbps)

Example

- `int socket(int domain, int type, int protocol);`

  domain: AF_UNIX , AF_INET, AF_INET6 , ..
  type: SOCK_STREAM, SOCK_DGRAM, ...

  creates a socket object

- `int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);`

  Binds a socket object to an IP(..) address

```
struct sockaddr {
    sa_family_t sa_family;
    char        sa_data[14];
}
```

```
myname.sin_family = AF_INET;
myname.sin_addr.s_addr = inet_addr("129.6.25.3");
/* specific interface */
myname.sin_port = htons(1024);
```

Example

- `int listen(int sockfd, int backlog);`

  Tells OS to prepare for incoming requests at that socket address (SOCK_STREAM) and buffer up to <backlog> outstanding requests.

- ```
  int accept(int sockfd,
             struct sockaddr *restrict addr,
             socklen_t *restrict addrlen);
  ```

  accepts connection (after socket/bind/listen) and returns new socket with addr/addrlen filled with connection peer information. Sockfd can continue to accept other connection requests.
  ```

Application Interface

CLIENT

```
int sd = socket();

connect(sd);

read(sd,..);
write(sd,..);
close(sd);
```

SERVER

```
int sd = socket();

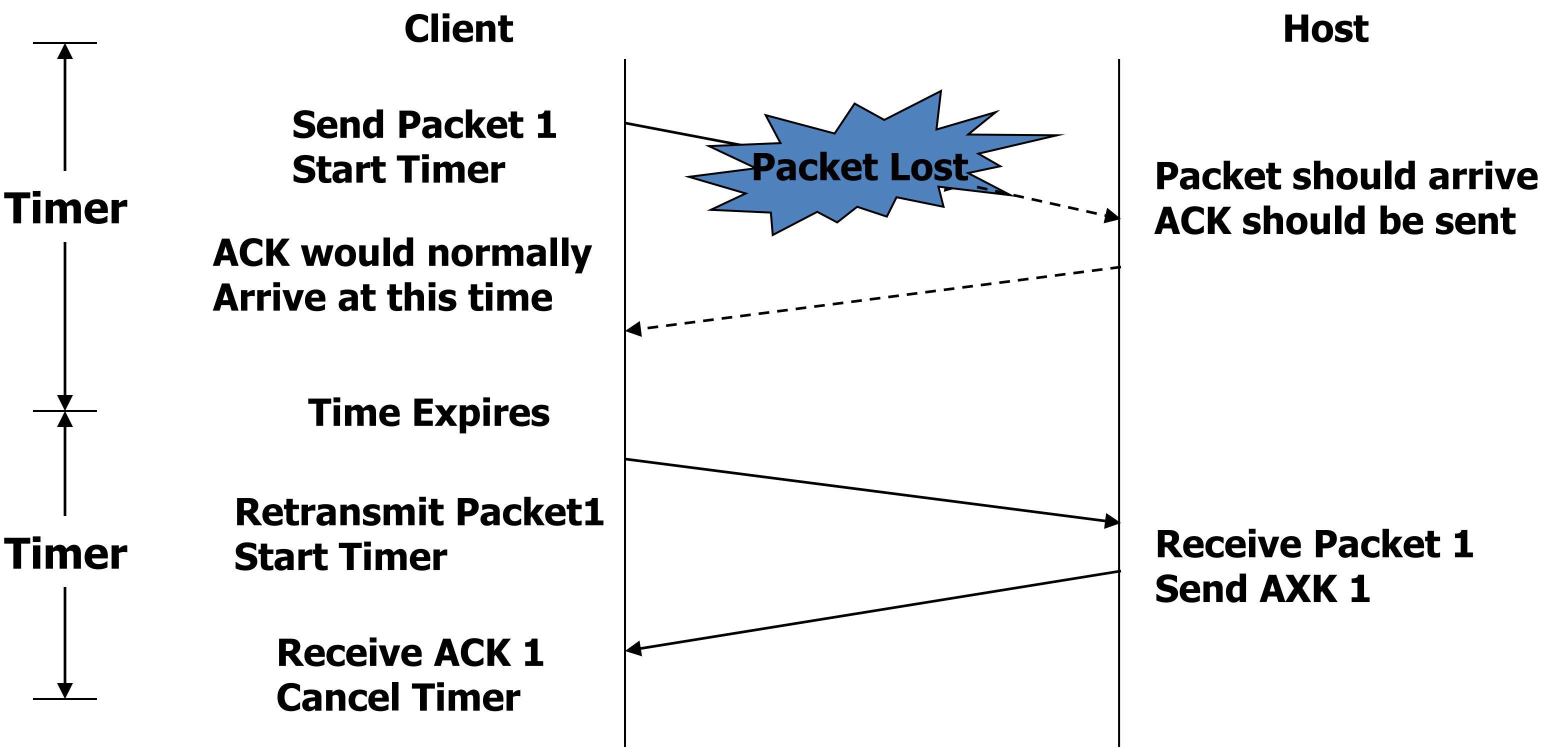
bind(sd);

listen(sd,..);

while (cond) {
    sd2 = accept(sd,..);

    write(sd2, ..);
    read(sd2, ..);
    close(sd2);
}
close(sd);
```

TCP : Data transfer

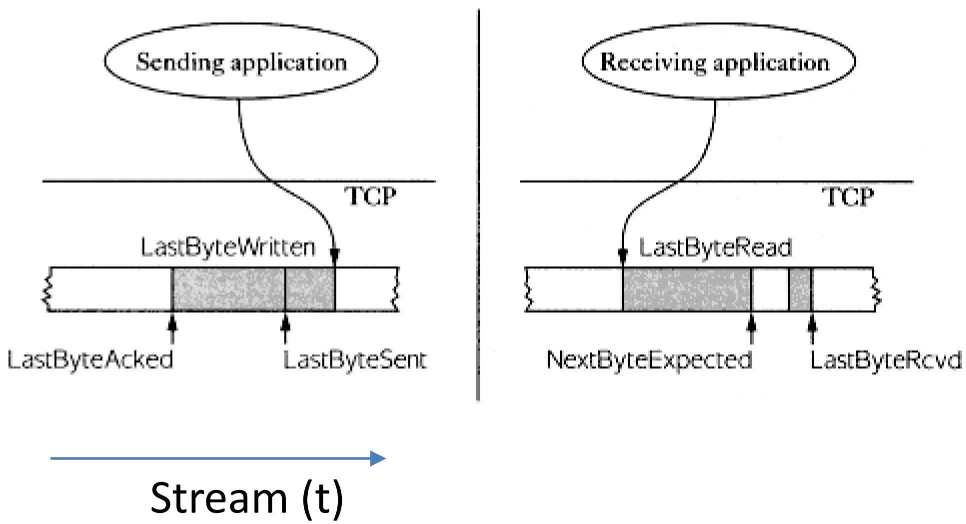


- Several packets can be transmitted, while waiting for acks; this is not serialized
- Up the TCP/IP subsystem to keep track of outstanding packets

TCP/IP Details

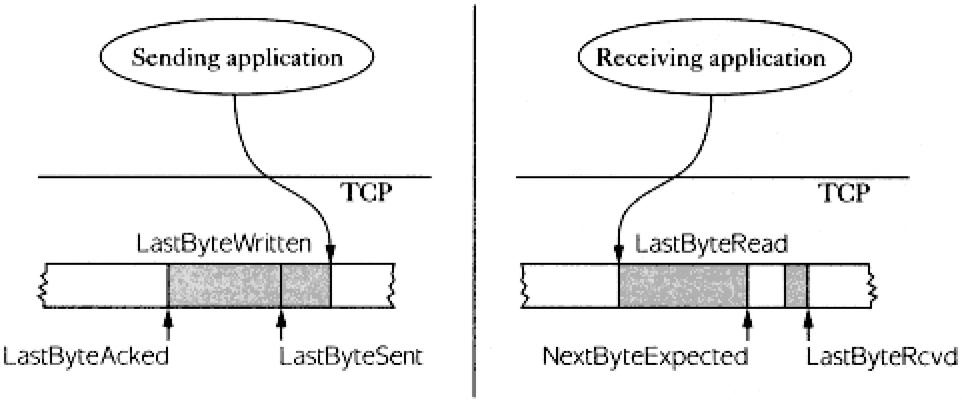
- TCP/IP uses a sliding window to buffer and control data flow

- The sliding window serves several purposes:
- (1) it guarantees the reliable delivery of data
- (2) it ensures that the data is delivered in order,
- (3) it enforces flow control between the sender and the receiver.



Flow Control

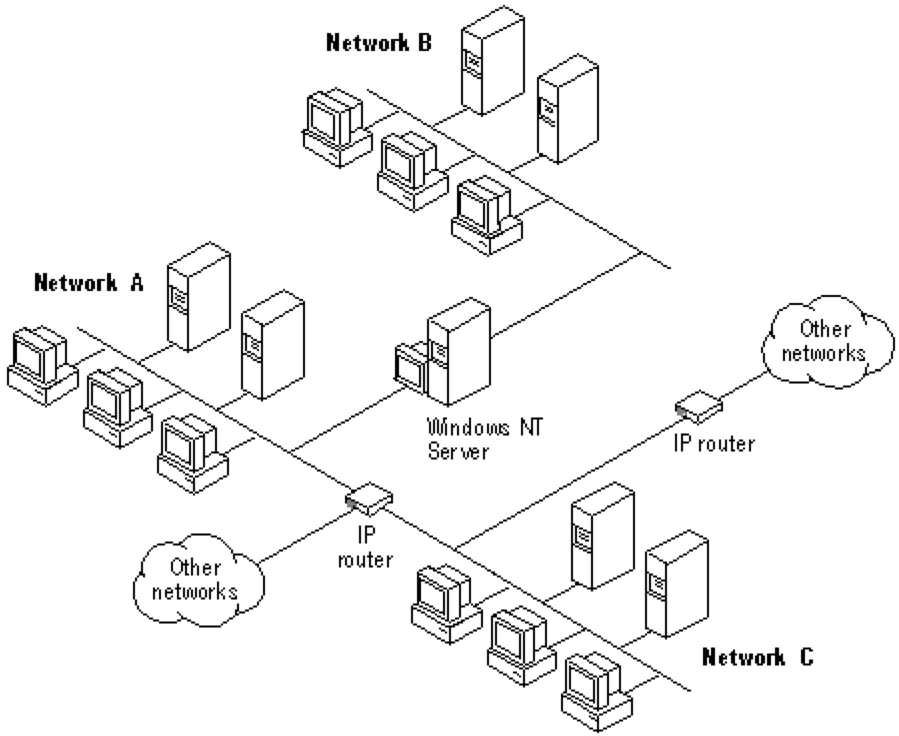
- Max Send and Receive Buffer sizes
- In order delivery to the consumer
- Acknowledgement of reception
- Retransmit when ack is not received in RTT (RoundTripTime) setting
- Only when last byte is ack'd can the window move



Congestion Control

- Window size synonymous with current available bandwidth:
  - No error → we could push more data → increase the window
  - Error → we are pushing to much → decrease the window
- Slow Start
  - Start with 1 congestion window and then doubling it
- When ack timeout occurs reduce the window size
- Fast Retransmit
  - When out of order packet is received immediately ACK
- Fast Recovery
- Many different Flow Control protocols

Routing


Network B
Network A
Windows NT
Server
Other
networks
IP router
IP
router
Other
networks
Network C

Routing

- How does a device know where to send a packet?
  - All devices need to know what IP addresses are on directly attached networks
  - If the destination is on a local network, send it directly there
- How do we discover this IP addresses
  - DNS & ARP

Routing (cont)

- If the destination address isn't local
  - Most non-router devices just send everything to a single local router (aka gateway)
  - Routers need to know which network corresponds to each possible IP address

```
[frankeh@access2 ~]$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         nyu-vl424-gw.ne 0.0.0.0         UG    100    0        0 eth0
128.122.49.0    0.0.0.0         255.255.255.0   U     100    0        0 eth0
128.122.49.0    0.0.0.0         255.255.255.0   U     100    0        0 eth0
```

```
[frankeh@access2 ~]$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  wwhgwa-vl424.net.nyu.edu (128.122.49.2)  0.356 ms  0.368 ms  0.364 ms
 2  10.254.4.20 (10.254.4.20)  0.330 ms  0.342 ms  0.307 ms
 3  128.122.1.36 (128.122.1.36)  0.382 ms  0.371 ms  0.371 ms
 4  ngfw-palo-vl1500.net.nyu.edu (192.168.184.228)  0.806 ms  0.759 ms  0.733 ms
 5  nyugwa-outside-ngfw-vl3080.net.nyu.edu (128.122.254.114)  1.023 ms  1.055 ms  1.021 ms
 6  dmzgwb-ptp-nyugwa-vl3082.net.nyu.edu (128.122.254.111)  1.513 ms  1.300 ms  1.440 ms
 7  128.122.254.70 (128.122.254.70)  0.952 ms  1.043 ms  0.995 ms
 8  ix-xe-7-3-2-0.tcore2.nw8-new-york.as6453.net (64.86.62.13)  1.351 ms  1.318 ms  1.265 ms
 9  if-ae-0-2.tcore1.nw8-new-york.as6453.net (209.58.75.217)  1.898 ms  1.976 ms  1.863 ms
10  if-ae-3-2.tcore1.n0v-new-york.as6453.net (216.6.90.72)  3.668 ms  3.677 ms  2.304 ms
11  72.14.223.124 (72.14.223.124)  2.133 ms  2.068 ms  2.042 ms
12  108.170.248.33 (108.170.248.33)  3.020 ms  3.206 ms  3.131 ms
13  209.85.245.99 (209.85.245.99)  2.064 ms 108.170.235.183 (108.170.235.183)  2.303 ms 209.85.243.193 (209.85.243.193)  2.060 ms
14  google-public-dns-a.google.com (8.8.8.8)  2.076 ms  2.051 ms  2.283 ms
```

DNS: (domain name service)

Name to IP lookup

```
frankeh@NYU2: nslookup access.cims.nyu.edu
Server:                 127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:           access.cims.nyu.edu
Address: 128.122.49.15
Name:           access.cims.nyu.edu
Address: 128.122.49.16
```

```
frankeh@NYU2: nslookup www.cnn.com
Server:                 127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
www.cnn.com canonical name = turner-tls.map.fastly.net.
Name:           turner-tls.map.fastly.net
Address: 151.101.209.67
```

Name lookups can change
Reflecting that multiple systems
serve the same service/content

```
frankeh@linax1[~]$ nslookup www.cnn.com
Server:                 128.122.253.79
Address:        128.122.253.79#53

Non-authoritative answer:
www.cnn.com canonical name = turner-tls.map.fastly.net.
Name:           turner-tls.map.fastly.net
Address: 151.101.117.67
```

ARP
(Address Resolution Protocol)

- Protocol to discover Link layer address (e.g. MAC address)
- who has IP <128.122.49.137> ?
  - broadcasts address on local network
  - owning node responds with MAC
  - receiving node caches the MAC in ARP cache

```
frankeh@linax1[~]$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
128.122.49.137           ether   52:54:00:c2:72:79   C                     eth1
128.122.49.112           ether   52:54:00:67:66:e2   C                     eth1
128.122.49.99            ether   00:14:4f:2b:0f:ae   C                     eth1
128.122.49.10            ether   24:6e:96:19:24:28   C                     eth1
128.122.49.3             ether   88:75:56:3c:a6:c0   C                     eth1
128.122.49.75            ether   52:54:00:d5:b0:2e   C                     eth1
128.122.49.95            ether   00:21:28:a3:10:43   C                     eth1
```

```
frankeh@linax1[~]$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
VM-PUBPC11.CIMS.NYU.EDU  ether   52:54:00:c2:72:79   C                     eth1
PROXY1.CIMS.NYU.EDU      ether   52:54:00:67:66:e2   C                     eth1
MX.CIMS.NYU.EDU          ether   00:14:4f:2b:0f:ae   C                     eth1
FS-U2.CIMS.NYU.EDU       ether   24:6e:96:19:24:28   C                     eth1
WSSGW-VL424.NET.NYU.EDU  ether   88:75:56:3c:a6:c0   C                     eth1
WEBMAIL.CIMS.NYU.EDU     ether   52:54:00:d5:b0:2e   C                     eth1
MAILFS.CIMS.NYU.EDU      ether   00:21:28:a3:10:43   C                     eth1
```

Network setup

- Static

  information is stored in filesystem

/etc/network/interfaces

```
auto eth0
iface eth0 inet static
address 192.168.0.42
network 192.168.0.0
netmask 255.255.255.0
broadcast 192.168.0.255
gateway 192.168.0.1
```

- Dynamic

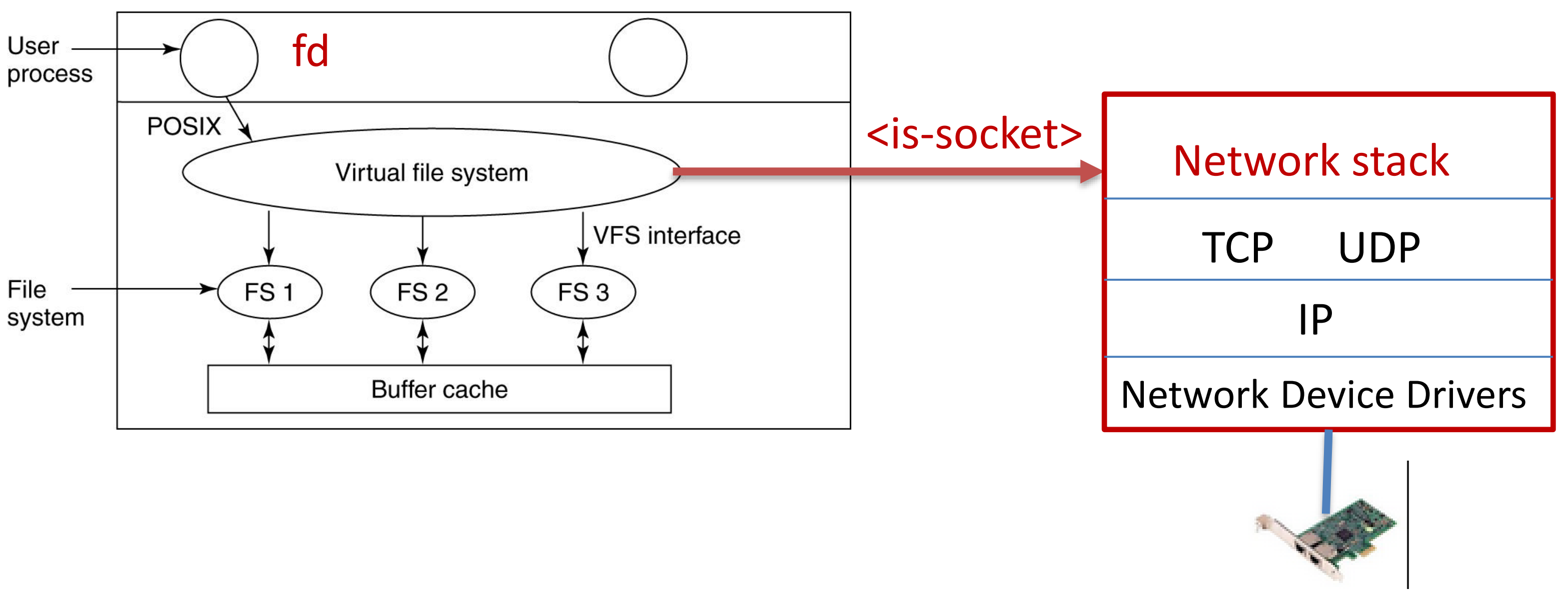
  information is retrieved from DHCP (dynamic host configuration protocol)
- DHCP server provides conf

/etc/network/interfaces

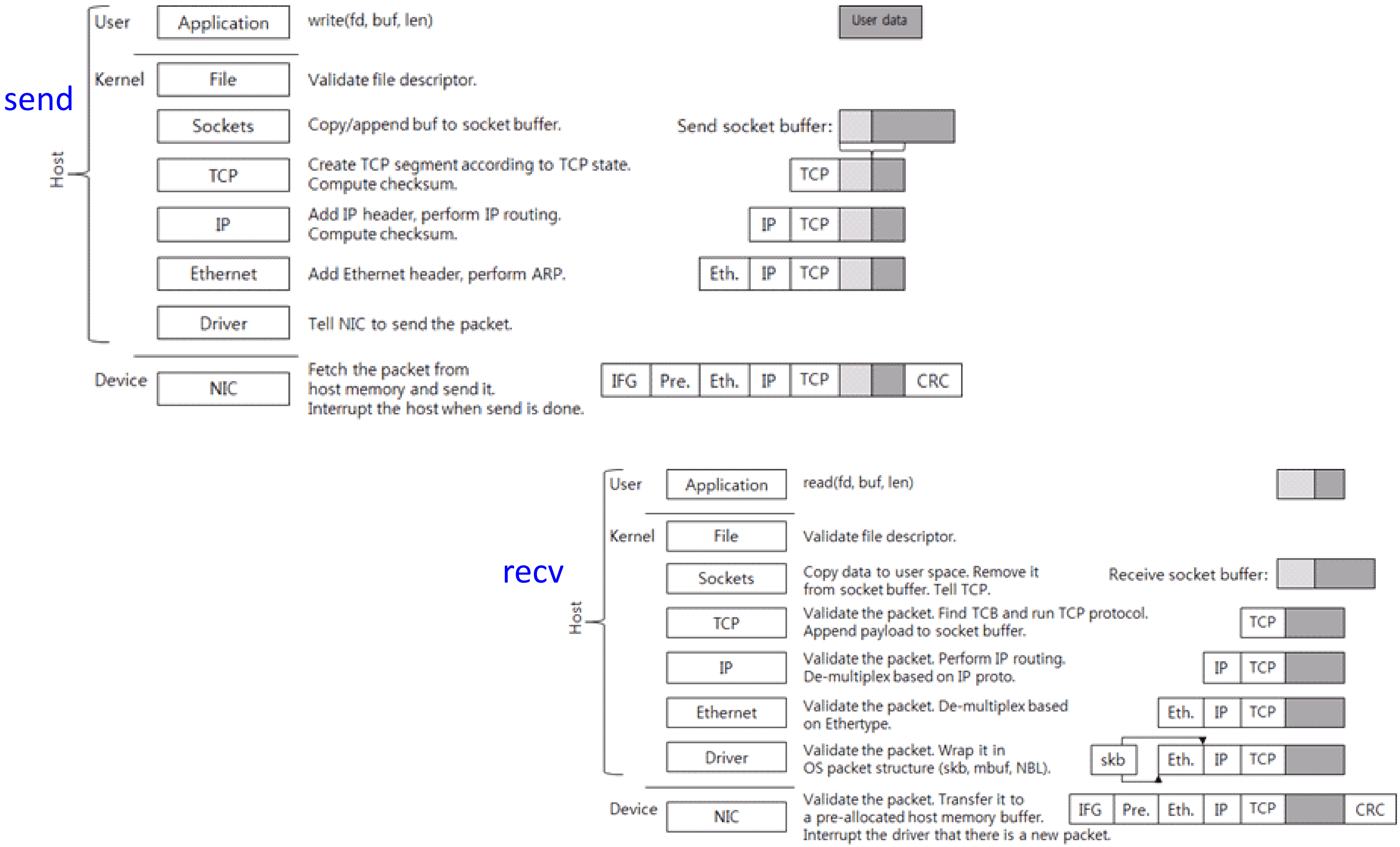
```
auto eth0
iface eth0 inet dhcp
```

Kernel Interface

- Unix → everything is a file descriptor
- Sockets are special file handles
- Diverted at high level into network stack



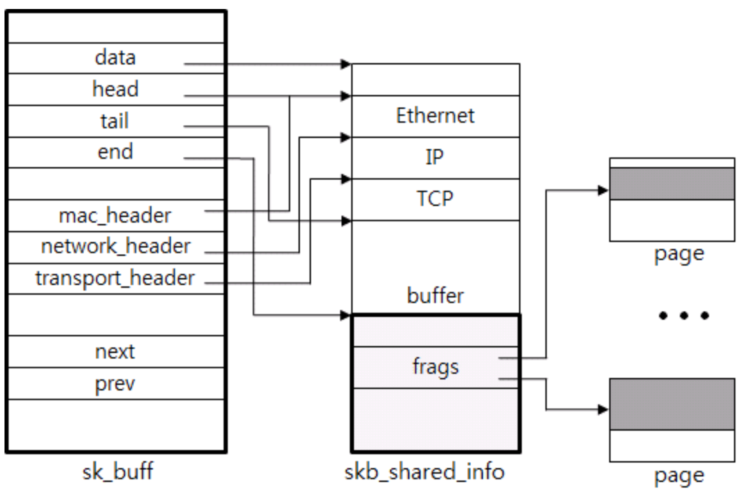
Putting it all together

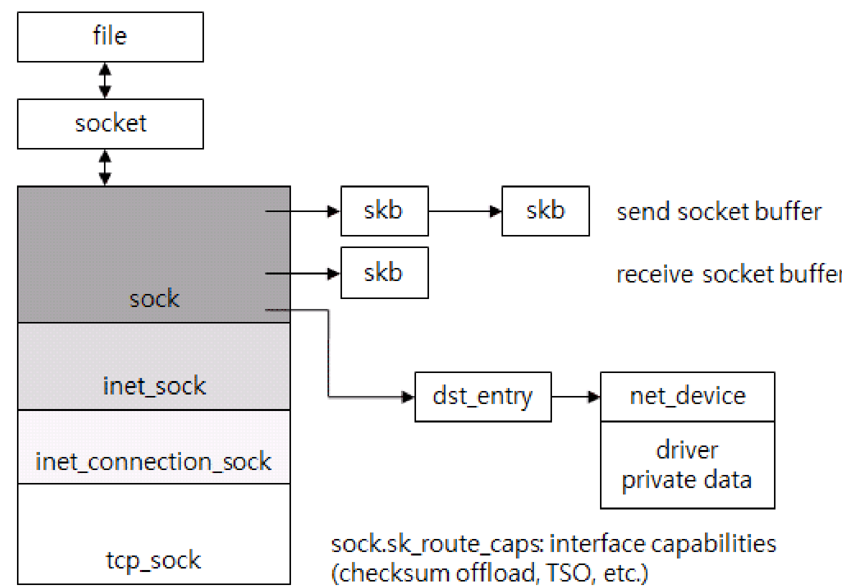


(source: https://www.cubrid.org/blog/understanding-tcp-ip-network-stack)

SKB (Socket Kernel Buffer)

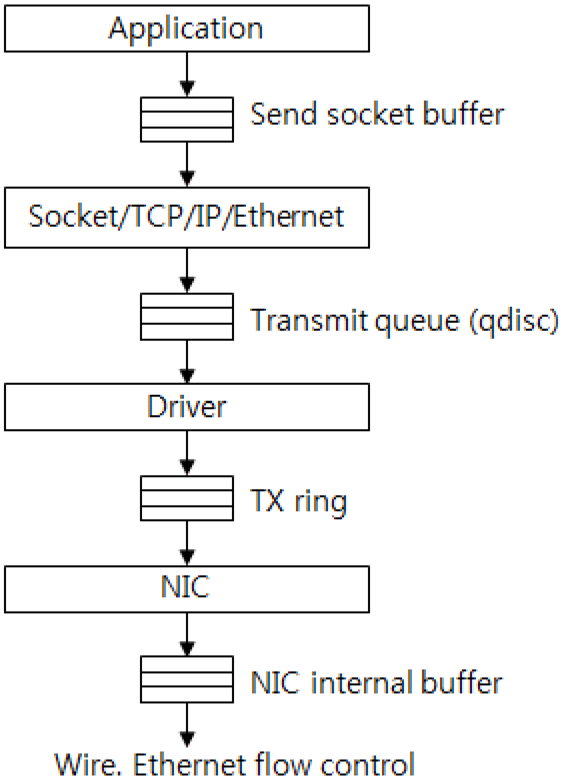
Generic object for storing socket data.

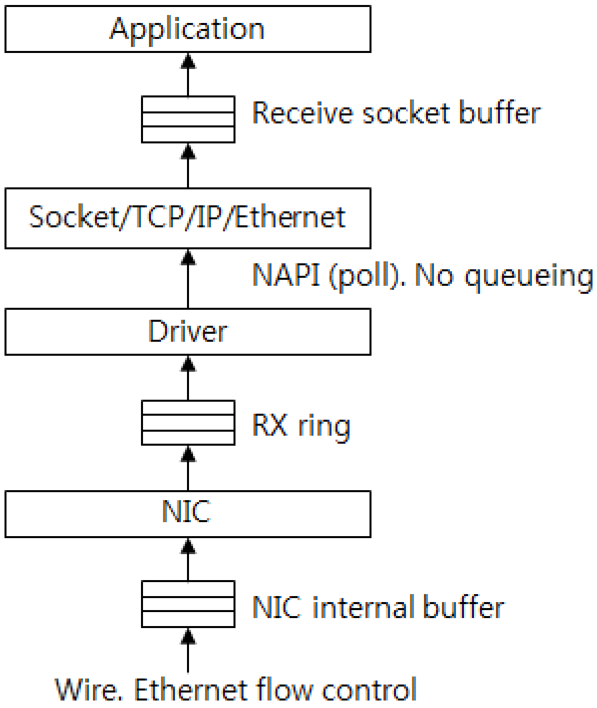




(source: https://www.cubrid.org/blog/understanding-tcp-ip-network-stack)

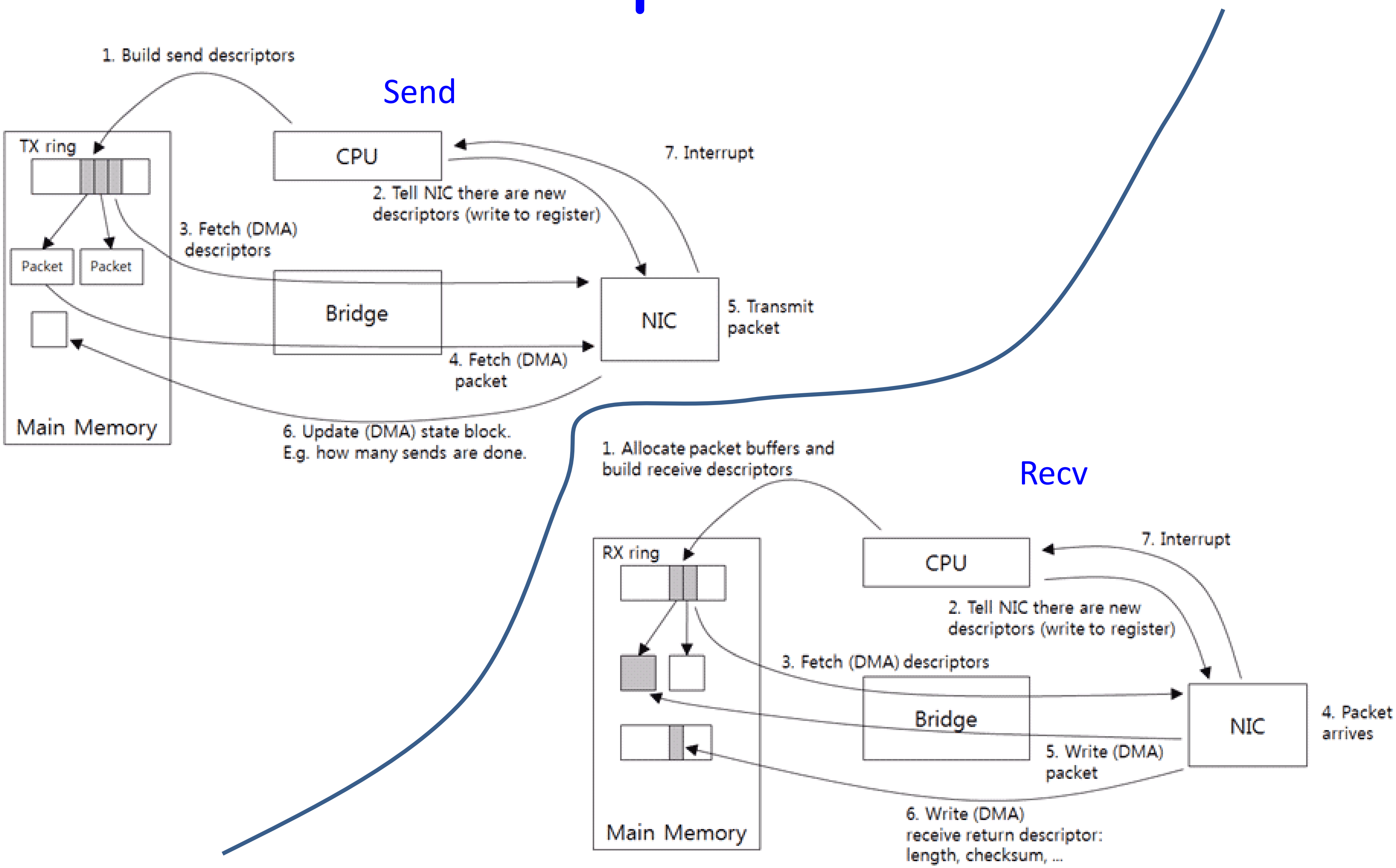
Down and Up flow of Data





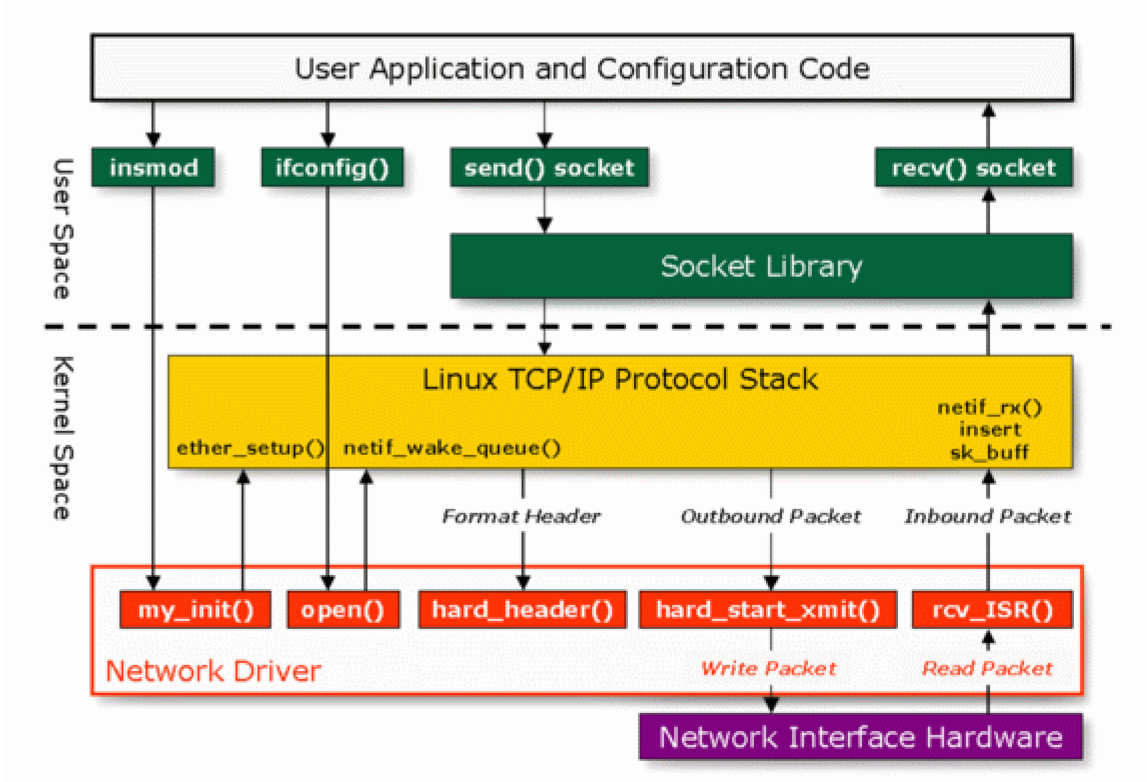
(source: https://www.cubrid.org/blog/understanding-tcp-ip-network-stack)

Down and Up flow of Data

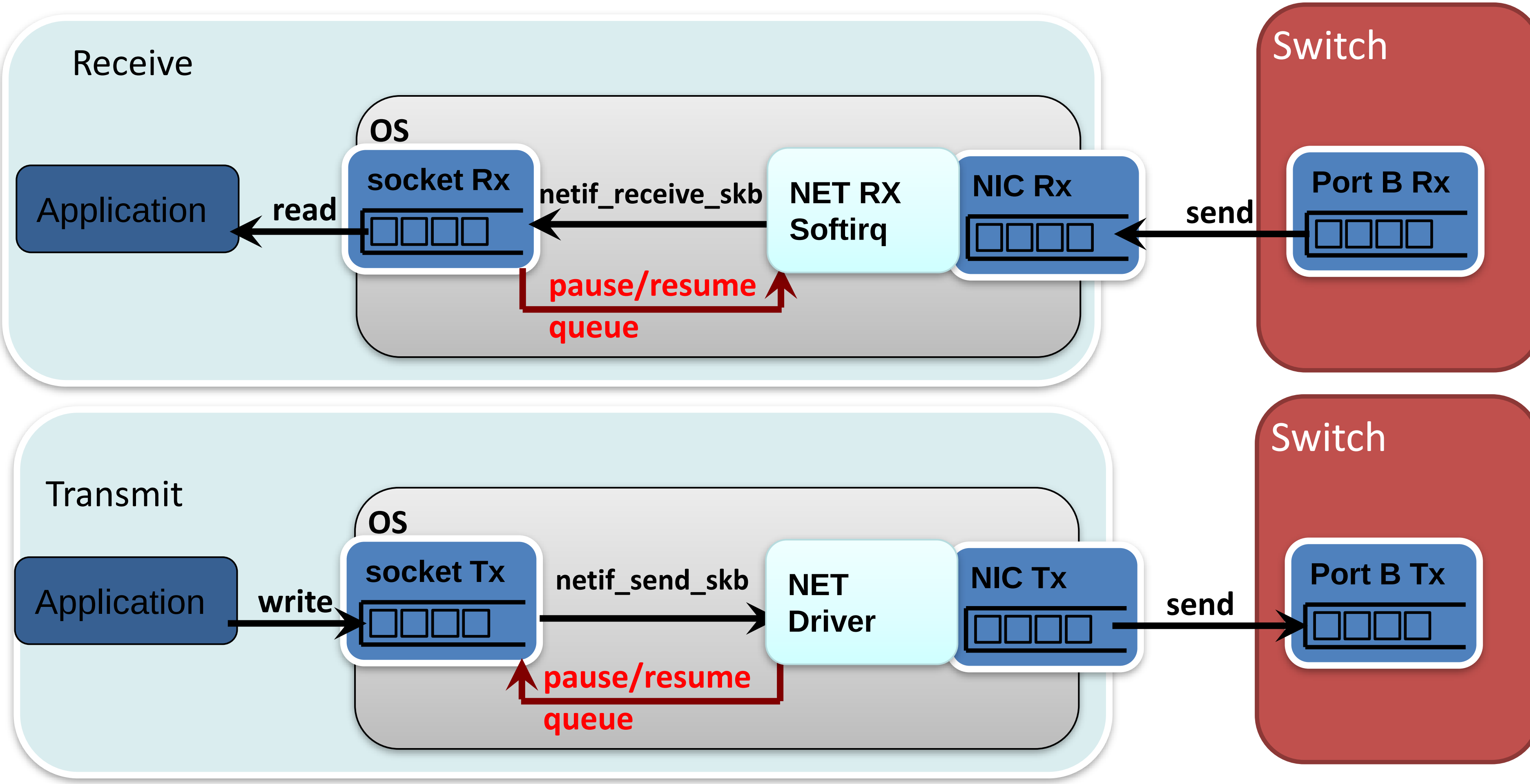


(source: https://www.cubrid.org/blog/understanding-tcp-ip-network-stack)

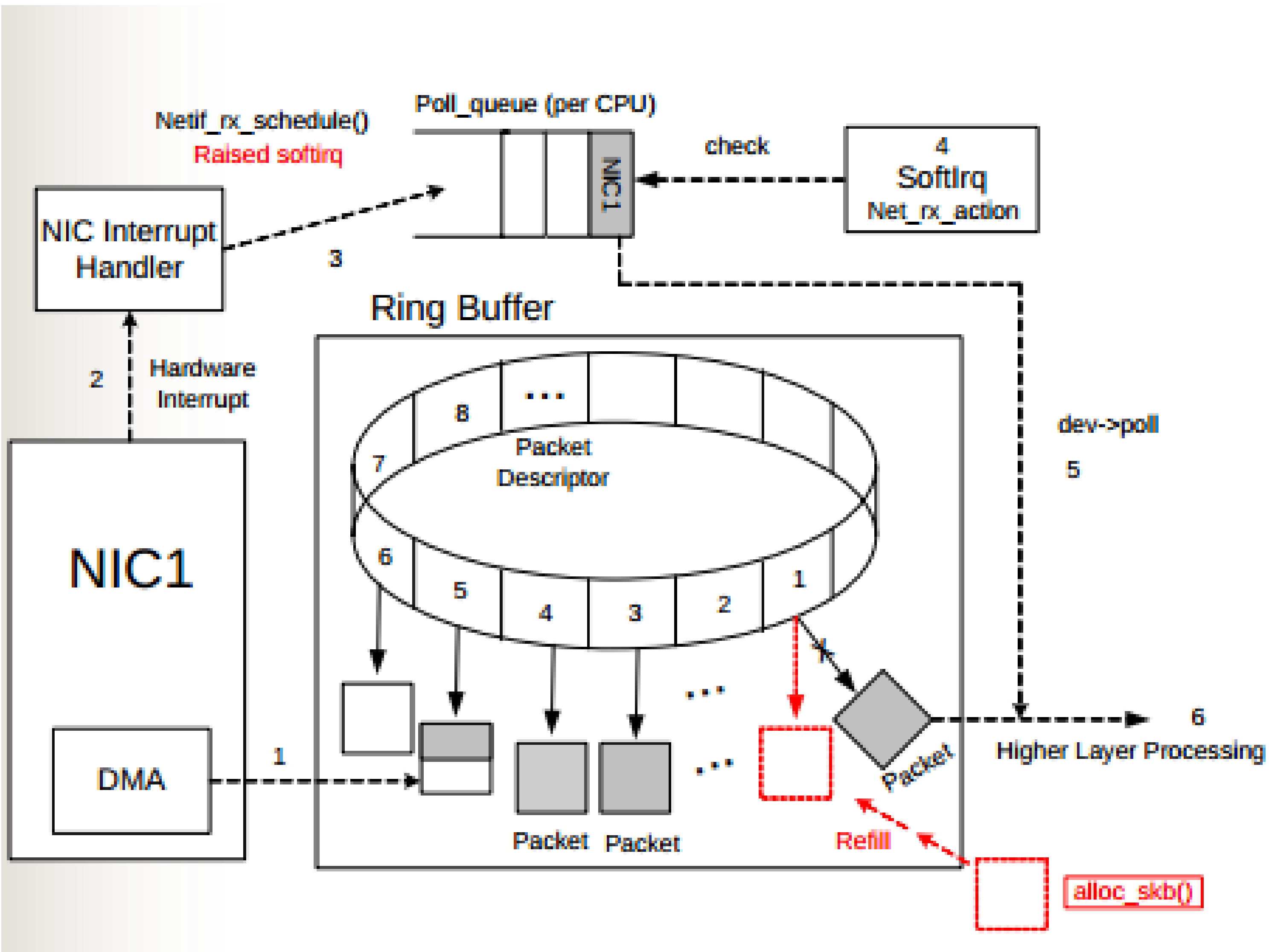
HighLevel View: Network Stack



Device Driver Details



Linux Tx/Rx Ring handling



Network Security

- Utilizes keys, encryption, and encapsulation
- Keys are exchanged (e.g. RSA)

  The encryption key is public and it is different from the decryption key which is kept secret.

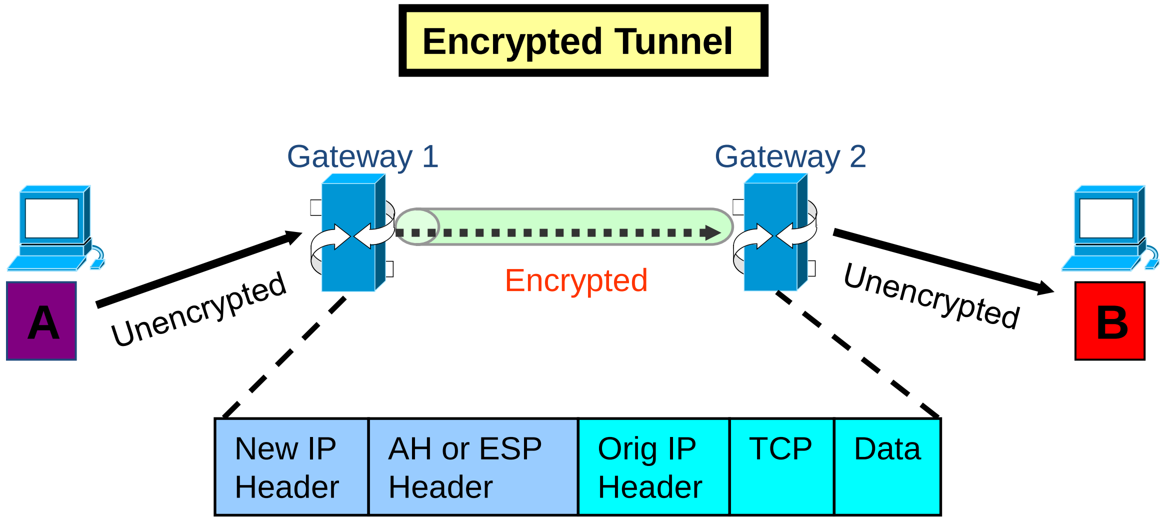
- Can be done at
  - application level (https , ssh, sftp, ..)
    using
    a) SSL (secure socket layer) now deprecated <u>or</u>
    b) TLS (transport layer security)
  - kernel level (Ipsec)

TLS setup/handshake

- Client:
  - connects to a TLS-enabled server requesting a secure connection and the client presents a list of supported ciphers and hash
    - Cipher: Diffie Hellman , Eliptic Curve, …
    - Hash Function: MD5, SHA-1, SHA-2, …
- Server:
  - selects a cipher and hash function it supports and notifies the client of the decision.
  - provides identification in the form of a digital certificate. The certificate contains the server name, the trusted certificate authority (CA) that vouches for the authenticity of the certificate, and the server's public encryption key.
- Client:
  - confirms the validity of the certificate.
  - Initiate Session Key (symmetric) using Diffie Hellman key exchange
  - Session Key used for all communication on secure connection

Modes

- Transport mode: host -> host
- Tunnel mode: host->gateway or gateway->gateway



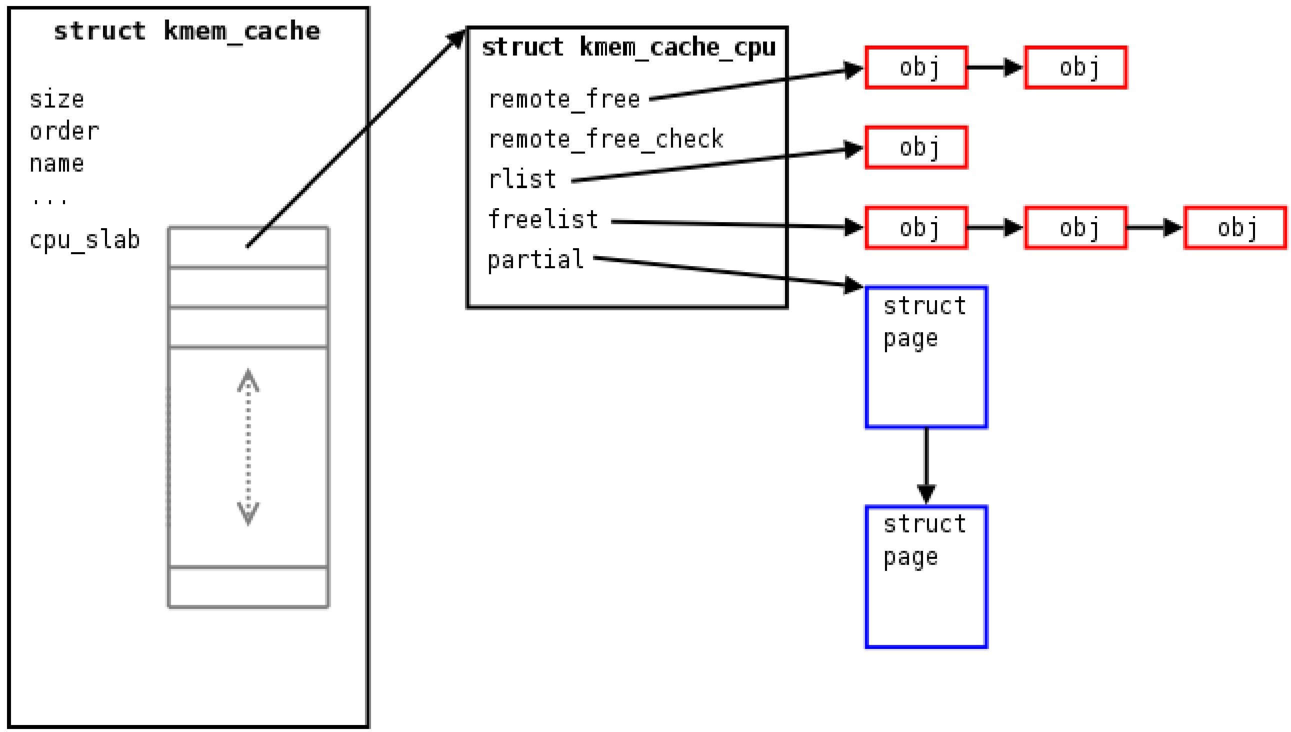
Authentication header (AH)
Encapsulating security payload (ESP)

Some other useful general "stuff" (not just network)

- Slab-Cache
  - The primary motivation for slab allocation:
    - initialization and destruction of kernel data objects can actually outweigh the cost of allocating memory for them.
    - As object creation and deletion are widely employed by the kernel, mitigating overhead costs of initialization can result in significant performance gains.
    - "object caching" was therefore introduced in order to avoid the invocation of functions used to initialize object state.
  - Group same dynamically allocated objects under one "allocator" object
  - E.g. skb_buff_alloc()

Some useful general "stuff"

- General implementation



- Other features:
  - Coloring → Cache utilization

Linux Example of slab caches

- #> slabtop

```
frankeh@frankeh-vb1: ~
File  Edit  View  Terminal  Help
 Active / Total Objects (% used)    : 194584 / 196400 (99.1%)
 Active / Total Slabs (% used)      : 6944 / 6944 (100.0%)
 Active / Total Caches (% used)     : 66 / 83 (79.5%)
 Active / Total Size (% used)       : 40252.62K / 40617.09K (99.1%)
 Minimum / Average / Maximum Object : 0.01K / 0.21K / 8.00K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
 48720  48701  99%    0.13K   1624       30      6496K dentry
 41408  41407  99%    0.06K    647       64      2588K buffer_head
 33969  33968  99%    0.61K   2613       13     20904K ext4_inode_cache
 13754  13608  98%    0.09K    299       46      1196K vm_area_struct
 10240  10092  98%    0.01K     20      512        80K kmalloc-8
  9180   9176  99%    0.05K    108       85       432K sysfs_dir_cache
  6656   6440  96%    0.12K    208       32       832K kmalloc-128
  4608   4547  98%    0.02K     18      256        72K anon_vma
  3712   3458  93%    0.03K     29      128       116K kmalloc-32
  3584   3103  86%    0.02K     14      256        56K kmalloc-16
  3471   3415  98%    0.29K    267       13      1068K radix_tree_node
  2893   2889  99%    0.35K    263       11      1052K inode_cache
  2240   2158  96%    0.06K     35       64       140K kmalloc-64
  1580   1570  99%    0.38K    158       10       632K proc_inode_cache
  1530   1517  99%    0.04K     15      102        60K Acpi-Operand
   968    937  96%    0.50K    121        8       484K kmalloc-512
   966    958  99%    0.09K     23       42        92K kmalloc-96
   861    791  91%    0.19K     41       21       164K kmalloc-192
   782    778  99%    0.45K     46       17       368K shmem_inode_cache
   666    653  98%    0.44K     74        9       296K mm_struct
   624    621  99%    2.00K     78        8      1248K kmalloc-2048
```