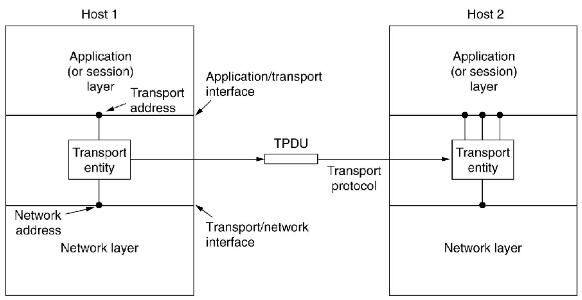
Network, Transport, and Application Layers

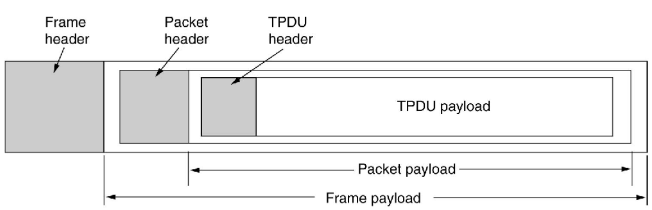
- TPDU (Transport Protocol Data Unit)
- Transport service provider
- Transport service user



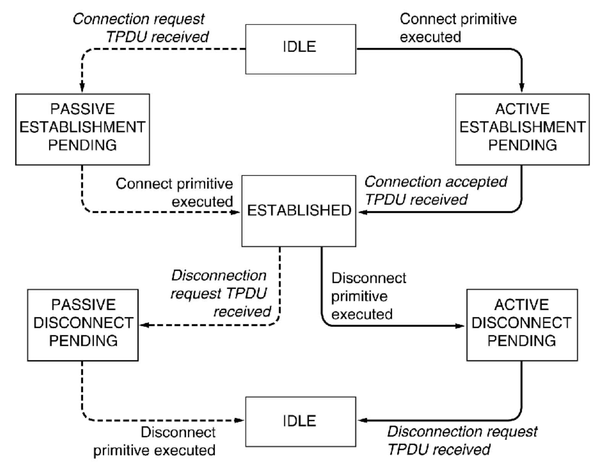
Connection Service Primitives

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Nesting



Simple Connection Management

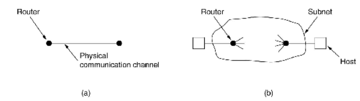


Berkeley Sockets

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345                /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                    /* block transfer size */

int main(int argc, char **argv)
{
 int c, s, bytes;
 char buf[BUF_SIZE];                     /* buffer for incoming file */
 struct hostent *h;                      /* info about server */
 struct sockaddr_in channel;             /* holds IP address */

 if (argc != 3) fatal("Usage: client server-name file-name");
 h = gethostbyname(argv[1]);             /* look up host's IP address */
 if (!h) fatal("gethostbyname failed");

 s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
 if (s <0) fatal("socket");
 memset(&channel, 0, sizeof(channel));
 channel.sin_family= AF_INET;
 memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
 channel.sin_port= htons(SERVER_PORT);

 c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
 if (c < 0) fatal("connect failed");

 /* Connection is now established. Send file name including 0 byte at end. */
 write(s, argv[2], strlen(argv[2])+1);

 /* Go get the file and write it to standard output. */
 while (1) {
      bytes = read(s, buf, BUF_SIZE);    /* read from socket */
      if (bytes <= 0) exit(0);           /* check for end of file */
      write(1, buf, bytes);              /* write to standard output */
 }
}

fatal(char *string)
{
 printf("%s\n", string);
 exit(1);
}
```

Environments

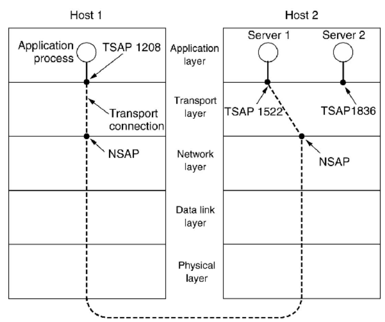


Data Link

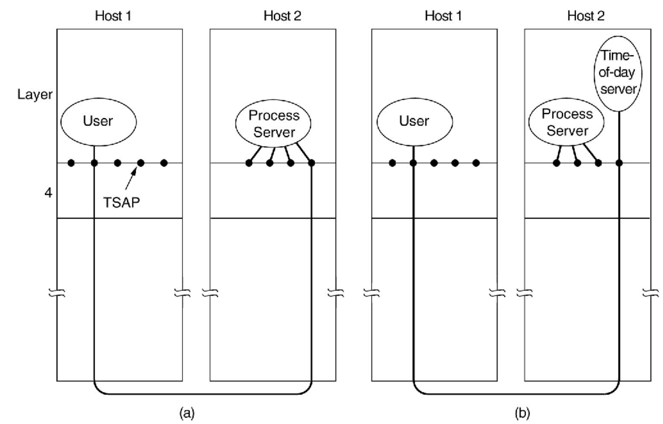
Transport

Addressing

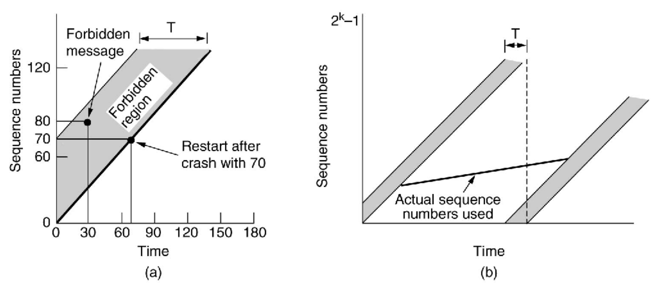
- TSAP (Transport Service Access Point)
- NSAP (Network Service Access Point)



Initial Connection Protocol



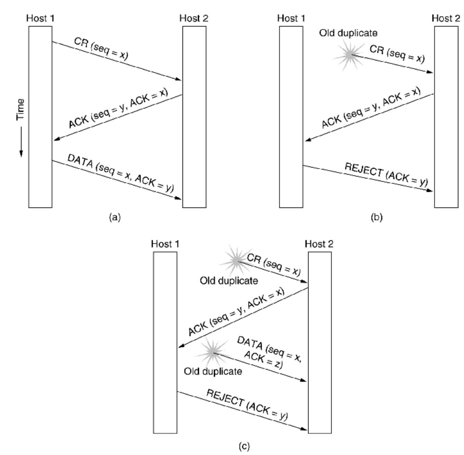
Forbidden Region



Restart

Resynchronize

Three-way Handshake



Abrupt Disconnect with loss of Data

Two-army Problem

Connection Teardown Scenarios

Buffer Management

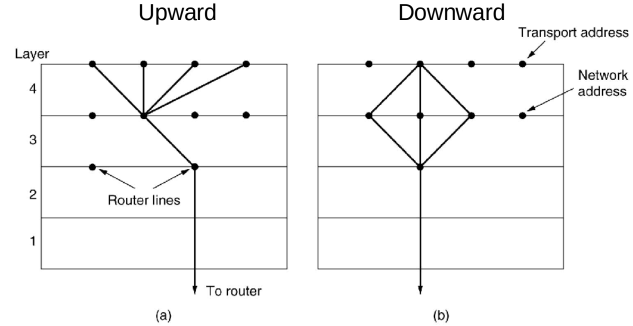
Fixed sized buffers

Variable sized buffers

Circular buffer

Deadlock – Dropped Window Advertisement

Multiplexing



e.g, channel bonding

Crash Recovery

A – Acknowledge
W – Write
C – Crash

	Strategy used by receiving host					
	First ACK, then write			First write, then ACK		
Strategy used by sending host	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message

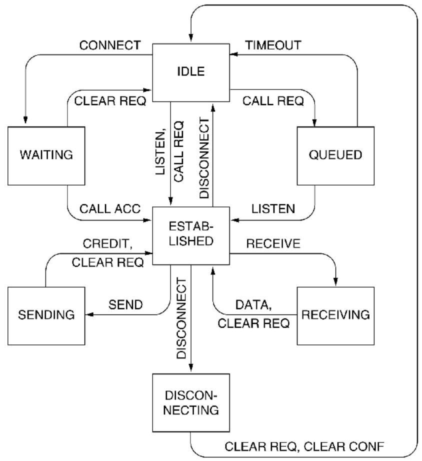
State Machine for Simple Transport Protocol

		State						
		Idle	Waiting	Queued	Established	Sending	Receiving	Dis-connecting
Primitives	LISTEN	P1: ~/Idle P2: A1/Estab $\overline{P2}$: A2/Idle		~/Estab				
	CONNECT	$\overline{P1}$: ~/Idle P1: A3/Wait						
	DISCONNECT				P4: A5/Idle $\overline{P4}$: A6/Disc			
	SEND				P5: A7/Estab $\overline{P5}$: A8/Send			
	RECEIVE				A9/Receiving			
Incoming packets	Call_req	P3: A1/Estab $\overline{P3}$: A4/Queu'd						
	Call_acc		~/Estab					
	Clear_req		~/Idle		A10/Estab	A10/Estab	A10/Estab	~/Idle
	Clear_conf							~/Idle
	DataPkt						A12/Estab	
	Credit				A11/Estab	A7/Estab		
Clock	Timeout			~/Idle				

Predicates
P1: Connection table full
P2: Call_req pending
P3: LISTEN pending
P4: Clear_req pending
P5: Credit available

Actions
A1: Send Call_acc
A2: Wait for Call_req
A3: Send Call_req
A4: Start timer
A5: Send Clear_conf
A6: Send Clear_req
A7: Send message
A8: Wait for credit
A9: Send credit
A10: Set Clr_req_received flag
A11: Record credit
A12: Accept message

Graphical Version of FSM



UDP

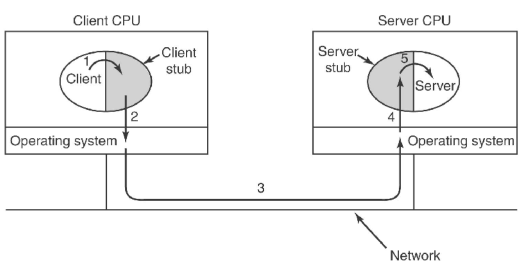
- User datagram protocol
  - multiplexes and demultiplexes
- Segments
  - 8 byte header
  - data

32 Bits

Source port	Destination port
UDP length	UDP checksum

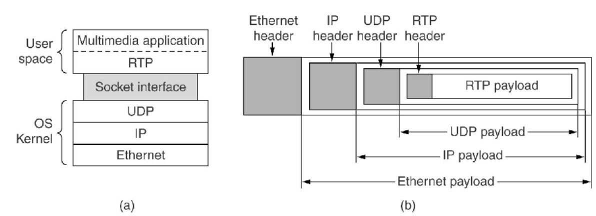
RCP over UDP

- Stubs
- Argument marshaling
  - pointers (length may not be known)
  - argument types may not be known (printf)
  - globals



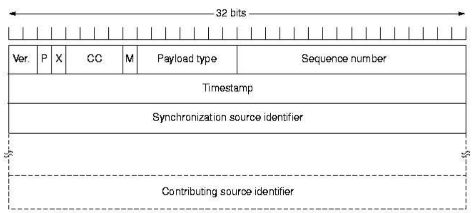
Real-Time Transport Protocol

- Transport protocol that runs in the application layer
- Multimedia



RTP Header

- Timestamp
  - jitter
- RTCP – Real-time Transport Control Protocol
  - no data, just control

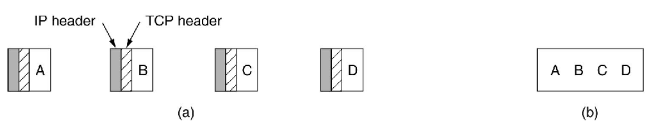


Sample Assigned Ports

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial file transfer protocol
79	Finger	Lookup information about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

TCP Byte Stream

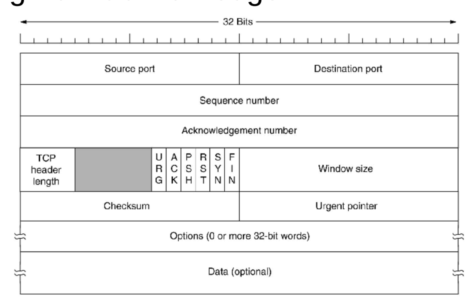
- Receiver doesn't know how sender sent the data



- Urgent data – generate a signal on the receiver

TCP Header

- Segment – 20 byte header + options + data
- MTU – Maximum Transfer Unit
- Window specifies # of bytes that may be sent starting from acknowledge

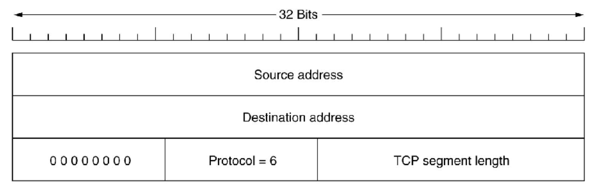


Sample Options

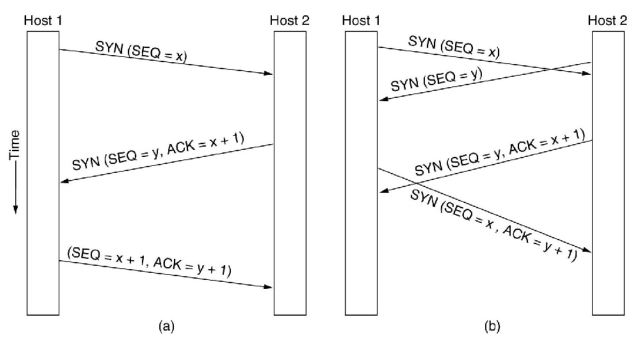
- Window scale
  - fast links with long delay may be idle most of the time
- Selective retransmit (rather than go back n)

TCP Checksum

- Checksums header + data + pseudo-header
- Pseudo-header:



TCP Connection

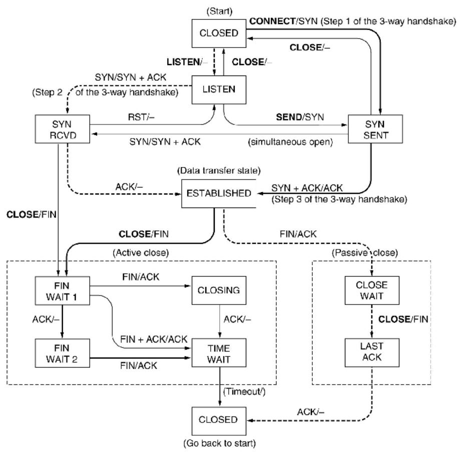


TCP Connection State

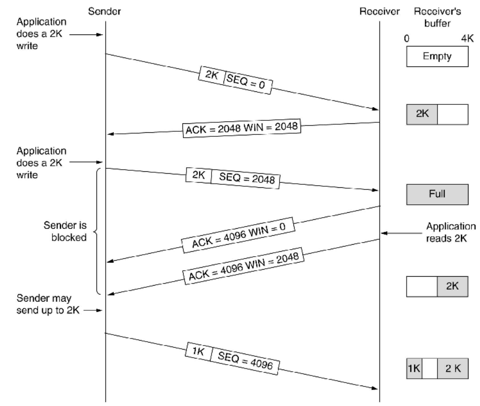
State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

TCP Connection Establishment

Heavy solid –
normal client
Heavy dashed –
normal server
Light –
unusual events



Window Management in TCP

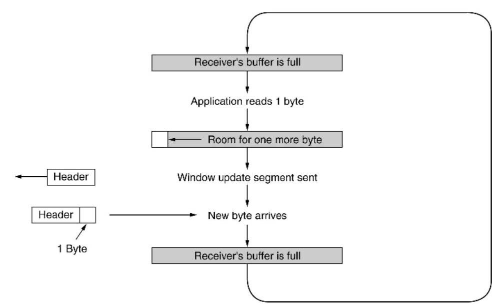


Window

- Receive window is 0
  - urgent packets
  - 1-byte packet to recover lost window announcement
- Senders may buffer data
  - 1 character / segment, 40 bytes of headers
  - Nagle: send 1 character, buffer until ACK
    - good balance in many cases
    - bad for things like mice (erratic jumps over the network)

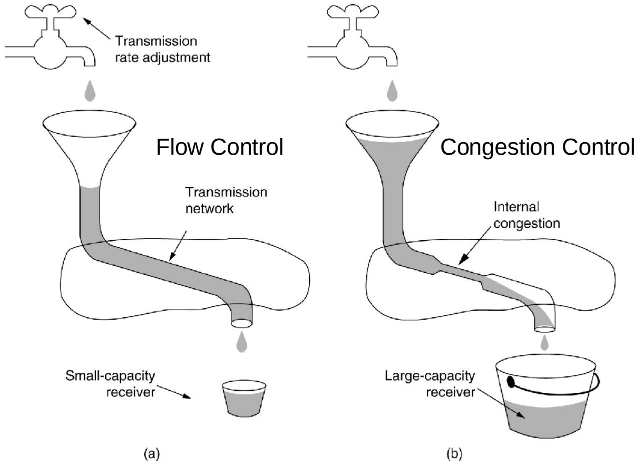
Silly Window Syndrome

receiver consumes 1 byte at a time



Clark's solution: no window update until
buffer has room for an MTU worth of data

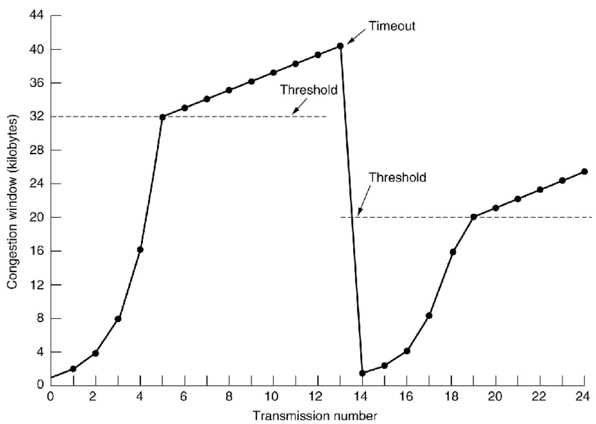
Congestion Control



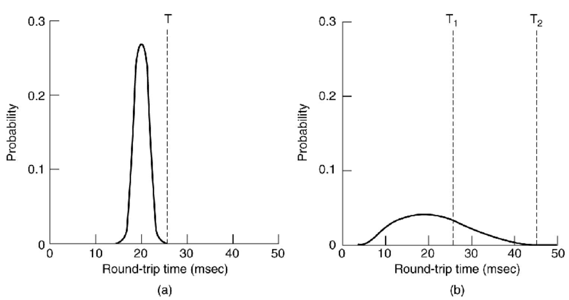
TCP Congestion Control

- Three parameters
  - Flow control window
  - Threshold
  - Congestion window
- Flow control bounds congestion window
- Threshold starts at 64K
- Congestion window starts at 1 and grows
  - " slow start" until threshold: +1 per segment
  - linear once it passes threshold: +1 per batch
- On timeout
  - threshold = congestion / 2
  - congestion = 1

Example Illustrating Threshold and Congestion Window



TCP Timer Management Probability Density for ACK arrival



Timers

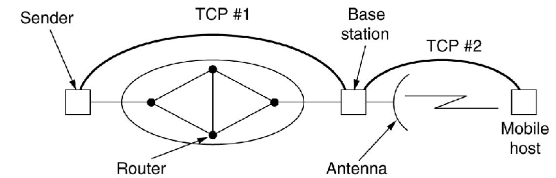
- Estimating RTT $RTT = \alpha RTT + (1-\alpha) M$

  $\alpha = 7/8$
- Timeout $\beta\, RTT\ (\beta = 2)$
- Dynamic estimate of varience

  $$D = \alpha D + (1-\alpha)\left|(RTT - M)\right|$$
- Karn's algorithm
  - don't update RTT for retransmitted segment
- Persistence time
  - dropped window advertisements
- Keepalive timer
- TIMED WAIT

Wireless TCP and UDP

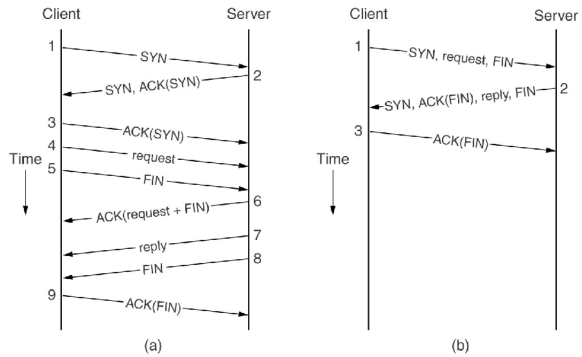
- Jacobson's rule no longer holds
- Indirect TCP: split TCP into two connections



- Alternate: base station becomes active in making the last kilometer reliable

Transactional TCP

- SYN includes data

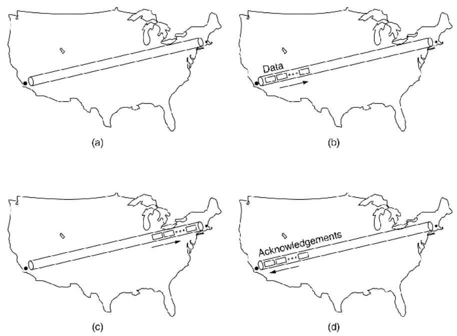


Performance Issues

- Problems
- Measurement
- System design
- Fast TPDU processing
- Newer protocols

Bandwidth-delay product

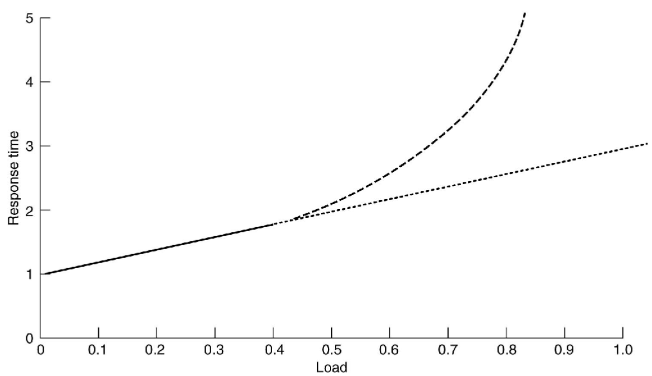
- bandwidth times RTT



Measurement

- Large enough sample size
  - multiple TPDUs
  - statistical significance
- Sampling times
- Coarse grained clocks
- Check for external events
- Watch for caching
- Understand what you are measuring
- Watch out for extrapolation

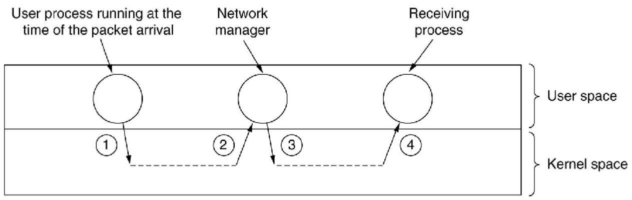
Extrapolation



System Design

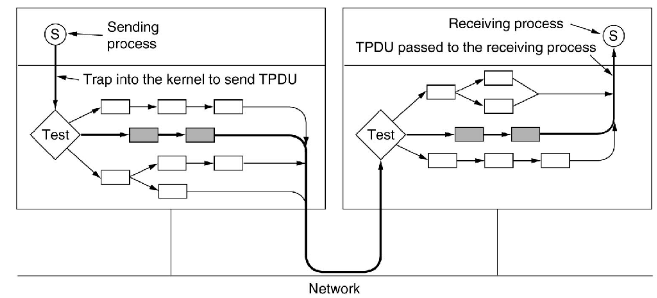
- Rule #1: CPU speed is more important than network speed
  - protocol processing is the bottleneck
- Rule #2: Reduce packet count
  - per packet overhead
  - pipelined processors
  - Nagle and Clark's silly window syndrome
- Rule #3: Minimize context switches (*)
- Rule #4: Minimize copies
- Rule #5: Bandwidth is easy, latency is hard
- Rule #6: Congestion avoidance is better than detection and recover
- Rule #7: Avoid timeouts

Context Switches

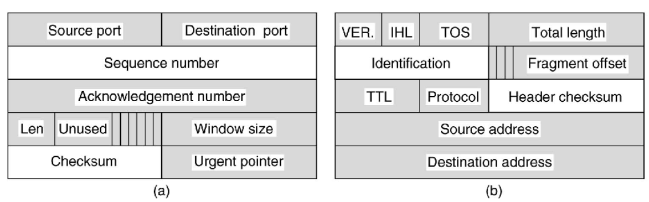


Fast TPDU processing

- Know what the fast path is

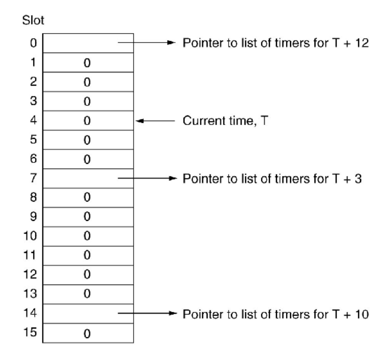


TCP and IP fields that remain constant in a stream



Timer Management

- Timer wheel



Gigabit Networks

- Sequence number wraparound
- Communication speeds are improving faster than processor speeds
- Go back n is bad when bandwidth-delay product is large
- Gigabit lines are delay limited rather than bandwidth limited (see graph)
- Variance may be more important than bandwidth

Design for speed, not bandwidth optimization

Transfer 1 MB file over 4000 km line