Transport Layer

Transport service is the essence of computer networking.

- The primary function is to enhance Quality of Service (QoS) provided by network layer.
- Duties:
  - ☐ provide reliable, efficient data transport from host to host
  - ☐ independent of underlying physical network(s)
- Service may be either connectionless or connection-oriented (Tanenbaum emphasizes connection-oriented)

Transport Layer

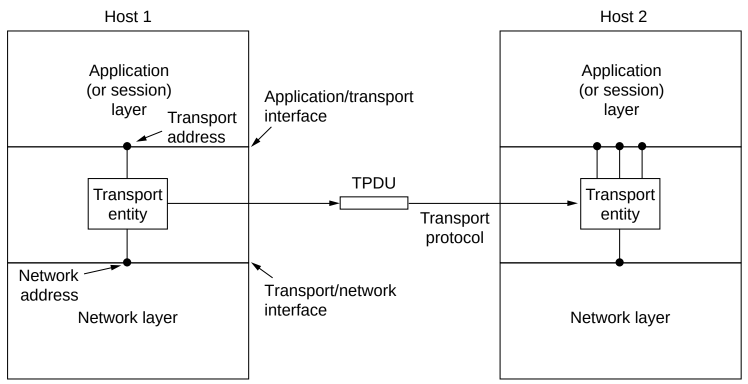
- Level of functionality required in the transport layer depends on the service provided by the network layer
  - ☐ reliable network layer → minimal transport layer
  - ☐ unreliable network layer → sophisticated transport layer
  - ☐ in reality, very little trust is put in network layers, so transport protocols are built to handle worst-case network layer
  - ☐ in TCP/IP, this is recognized; IP is designed to be simple and unreliable, TCP handles reliability (once!)

Defining QoS

Quality of service defines the worst-case performance for several measures; the carrier must meet or exceed this.

- Issue is more important for real-time traffic.
- A Possible List of QoS Parameters:
  - ☐ Minimum data rate
  - ☐ Maximum data rate
  - ☐ Sustained data rate
  - ☐ Propagation delay
  - ☐ Jitter
  - ☐ Error rate
  - ☐ Protection
  - ☐ Priority
  - ☐ Resilience

Network, Transport, and Application Layers



Transport Protocols

- In many ways, resemble data link protocols
- Similarities
  - must deal with error control
  - must handle sequencing
  - must implement flow control
- Differences stem from the fact that transport service is provided over a subnet, while data link service is provided over a single link
- Differences between transport and data-link layers:
  -
  -
  -
  -
  -

Characteristics of Transport Protocols

- Unit of data exchange
  - TPDU (Transport Protocol Data Unit)
  - may also be called a *segment* (TCP)
- Must be a mechanism to allow *processes*, not just to communicate between machines
  - multiplexing across network connections
- Establishment and releasing of connections
- Must be able to identify a connection, so transport protocol will know to which process incoming data should be delivered
- Ordering (and reordering) of TPDUs (sequence numbers)

Characteristics of Transport Protocols

- Expedited data
  - ☐ example?
  - ☐ why does this requirement present problems?
- Flow control is often implemented with a sliding window protocol, as in the data link layer. (Differences will be described later.)
- Maintenance of connections in absence of data being sent, as well as inactivity timers to detect lost TPDUs
- Checksums on data and headers
  - ☐ often simply add up bytes, modulo 256. WHY?
  - ☐
  - ☐

Addressing

To whom should each message be sent?

- Generic term is TSAP (Transport Service Access Point)
- Example: IP address with port number
- Normally, a transport entity supports multiple TSAPs.
- Example: ftp, telnet, http, etc. all have TSAPs
- Problem: how to know the TSAP of a given service?
  - ☐ Some services have stable, well-known TSAPs (as above)
  - ☐ Use *Initial Connection Protocol*:
    - Process server listens on set of TSAPs
    - On connection, it spawns-off the requested server

Addressing

An alternative scheme to handle servers that must exist independently of a process server is using a name server.

- Name server: special process that listens on well-known TSAP.
  - ☐ User sends request containing name of service
  - ☐ Name server responds with TSAP address.
- New Services must register selves with name server.

Connection Management

- Problem of delayed duplicates
- (Problematic) Solution Strategies:
  - ☐ Throwaway transport addresses
  - ☐ Unique ID for each connection
- Solution:
  - ☐ use a clock not affected by crashes
  - ☐ base sequence numbers on the clock
  - ☐ limit packet lifetimes
  - ☐ wait following crash before using certain sequence numbers
- Result addresses only data, not connection establishment
  - ☐ both sides must agree on initial sequence number
  - ☐ To actually establish the connection: use three-way handshake

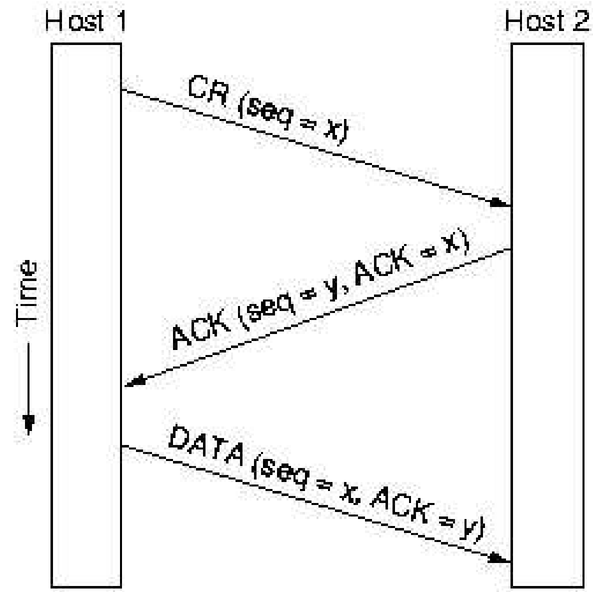
Three-way Handshake

- side A chooses sequence number x and sends to B in a connection request
- side B replies with a connection confirm acknowledging x and containing its own initial sequence number, y (which may be x)
- side A acks B's choice, y
- may also be used in connection disconnect solution: three-way handshake

Once the connection is established, any sliding window protocol can be used for data flow control.

Three-Way Handshake

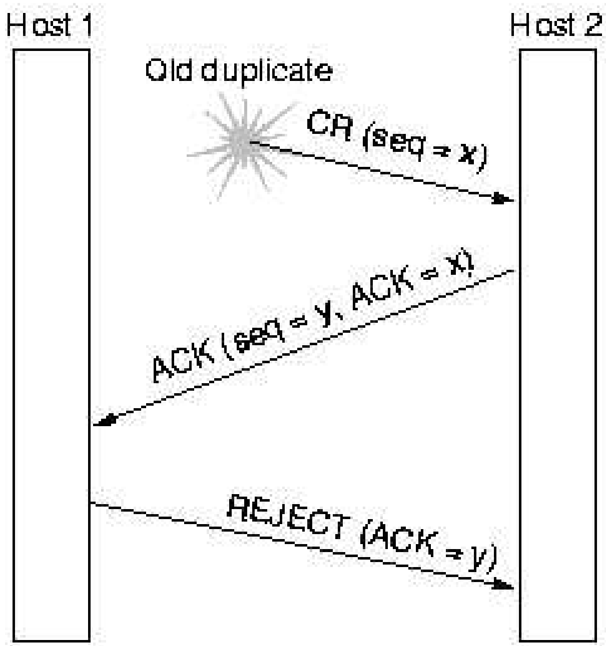
Normal operation



(a)

Three-Way Handshake

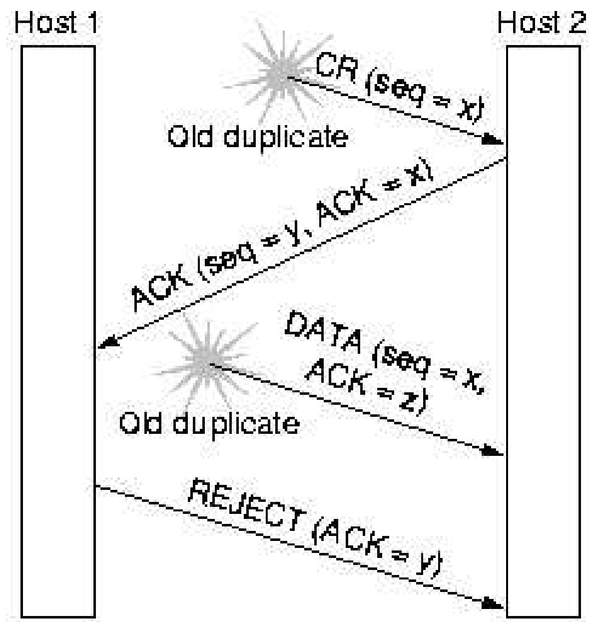
Old duplicate connection REQ appears



(b)

Three-Way Handshake

Duplicate connection REQ and duplicate ACK



(c)

Releasing a Connection

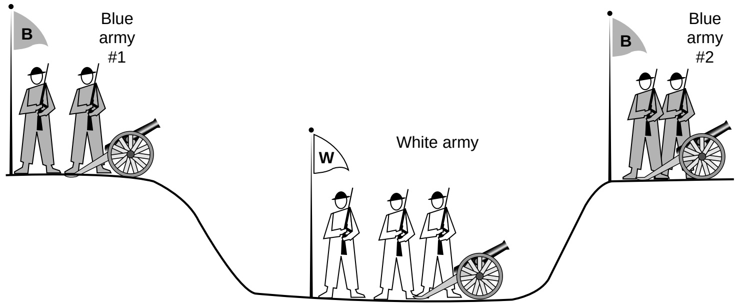
It's easier than connection establishment, but it still has some problems.

- Two styles of connection release:
  - ☐ Asymmetric
    - One party terminates connection.
    - Abrupt termination
    - Data loss is possible
  - ☐ Symmetric
    - Each direction is released independently
    - Works well with fixed data size
    - Works well with no errors.

Two-Army Problem

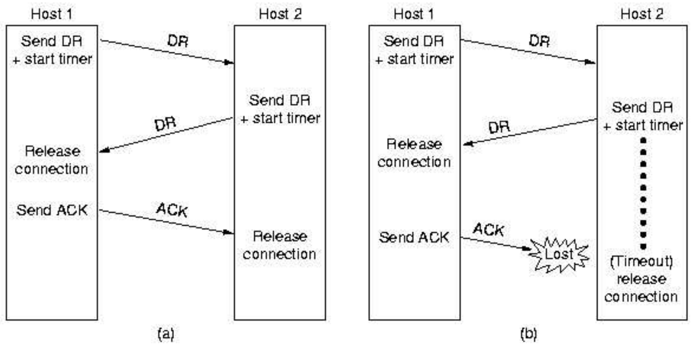
In symmetric release, the last party sending an ACK is never certain that the ACK got through.

- Largest army always wins
- White army is larger than either blue army but blue armies together are larger than white
- What protocol allows the blue armies to communicate so that they synchronize the attack?



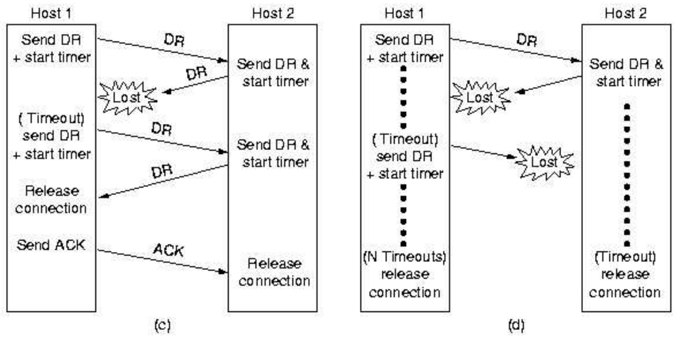
Releasing a Connection

(a) Normal operation and (b) losing the final ACK



Releasing a Connection

(c) Losing a response and (d) losing response as well as subsequent disconnect REQs.



Flow Control and Buffering

An important difference from Data Link Layer is that the number of connections can make buffering impractical.

- Sliding window protocol is used
- Network Layer:
  - ☐ Unreliable (datagram) service: sending transport layer must buffer
  - ☐ Reliable service: other tradeoffs are possible.
  - ☐ If a reliable network layer ACKs a packet, why can't the sender simply remove it from the buffer?
- Buffer size:
  - ☐ Chain of fixed-sized buffers
  - ☐ Chain of variable-sized buffers
  - ☐ Single large buffer per connection

Buffering

Whether to buffer at the source or destination depends on the traffic being carried.

- Low-bandwidth, bursty traffic: buffer at the sender
- High-bandwidth traffic: buffer at the receiver
- Dynamic Window management: variable-sized windows

	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	Potential deadlock

- Solution?

Multiplexing

Multiplexing may occur at several layers in the network architecture.

- Transport-layer multiplexing may be required for
  - ☐ Carrier pricing decisions
  - ☐ Carrier technical decisions
- Upward Multiplexing: multiplex different transport connections onto one network connection.
- Downward Multiplexing: multiplex one transport connection onto several network connections.

Crash Recovery

This is only an issue if hosts and routers are subject to crashes (which is always).

- Recovery from network and router crashes is straightforward
- Recovery from Host crashes:
  - ☐ Client desires to continue after server crashes/reboots
  - ☐ Example?
- One solution strategy after a crash:
  - ☐ Server requests status from all clients
  - ☐ Client exists in one of two states and replies accordingly
  - ☐ Problems with this approach?
- Ultimately, an *end-to-end acknowledgement* is probably impossible to achieve.

TCP and UDP

Two main protocols exist in the internet. In practice, TCP is used far more heavily than UDP.

- TCP: Transmission Control Protocol
  - ☐ Connection-oriented
  - ☐ Reliable end-to-end byte stream
  - ☐ Connection over an unreliable internetwork
- UDP: User Data Protocol
  - ☐ Connectionless
  - ☐ Unreliable, datagrams
  - ☐ Connection over an unreliable internetwork

TCP Service Model

TCP service is obtained through the use of sockets

- Each socket has an address consisting of IP address and port
- Connections must be explicitly established on both machines
- Socket primitives:

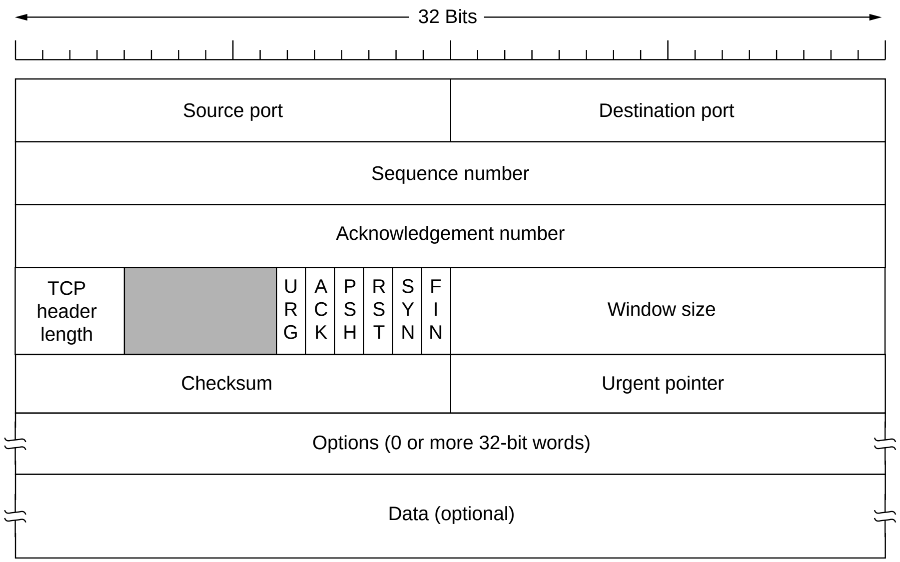
Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over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

The TCP Protocol

Data is exchanged in the form of a segment (TCP's version of the TPDU).

- Fixed 20-byte header
- Zero or more data bytes
- Segment size limitations:
  - ☐ IP payload maximum is 65,535 bytes
  - ☐ Maximum Transfer Unit (MTU) of each network
- Uses sliding window protocol
- Issues:
  - ☐ Fragmentation
  - ☐ Sequencing
  - ☐ Delay
  - ☐ Congested/broken intermediate networks

TCP Segment Header



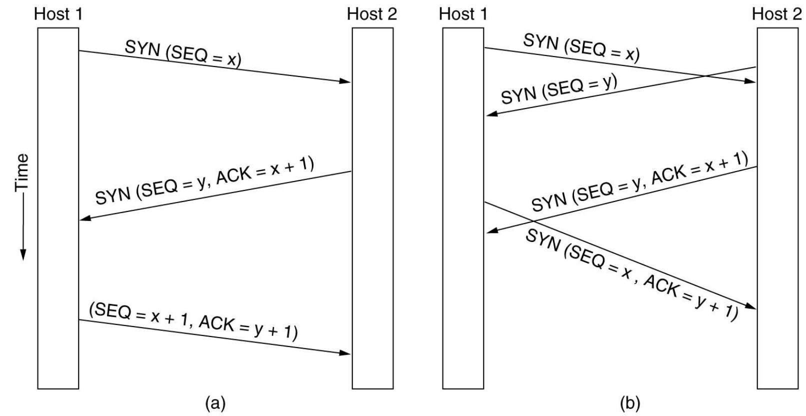
TCP Pseudoheader

Source address		
Destination address		
0 0 0 0 0 0 0 0	Protocol = 6	TCP segment length

32 Bits

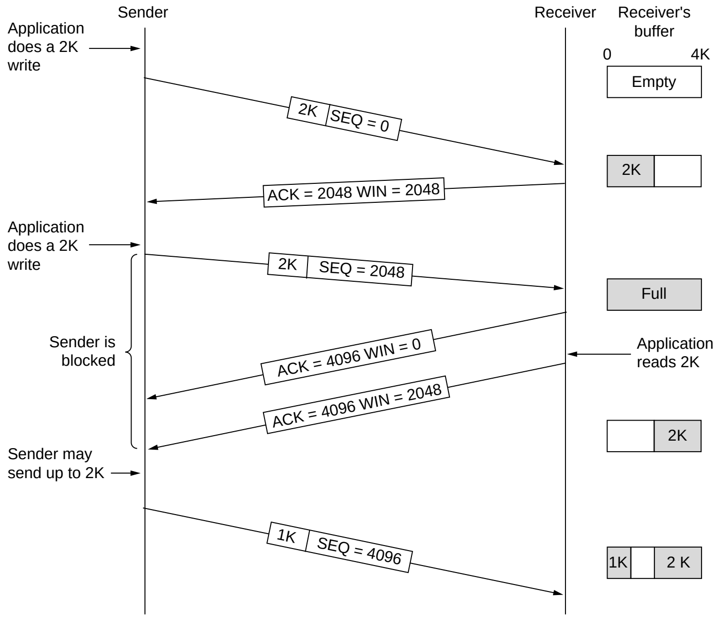
TCP Connection Management

- Three-way handshake is used
- Server passively waits using LISTEN and ACCEPT
- Client issues CONNECT
- Transport entity checks for server process
- Server process issues ACCEPT or REJECT



TCP Transmission Policy

- Window management is disjoint from ACKs

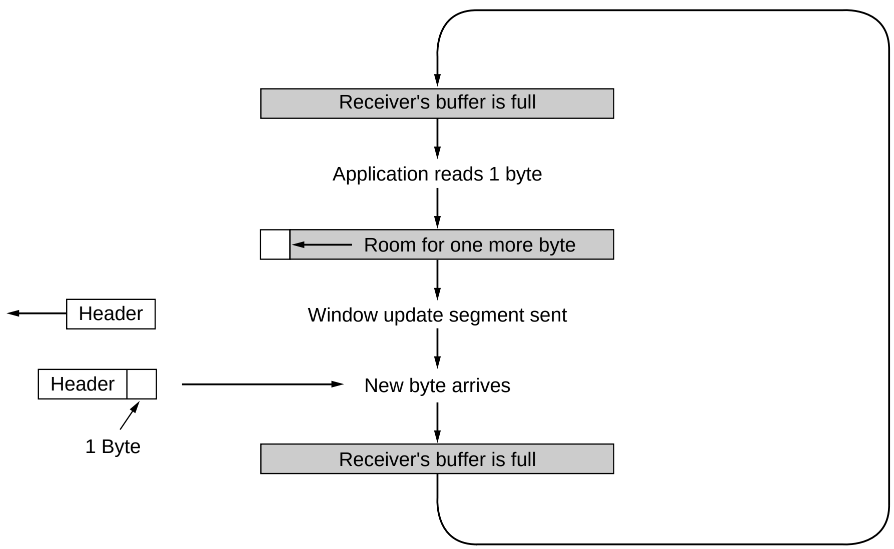


Low Bandwidth Bursty Traffic

Senders are not required to immediately transmit data when it becomes available.

- Example: useful for editor commands within telnet session.
  - ☐ Editor reacts to each keystroke
  - ☐ Worst case: 21 bytes per segment, 41 byte IP datagrams
  - ☐ Four packets per keystroke (why?)
- Solutions
  - ☐ Delayed acknowledgements
  - ☐ Nagle's algorithm

Silly Window Syndrome



TCP Congestion Control

Congestion is the condition in which offered load is greater than the network can handle.

- Only real solution: slow down the data rate
- Follow the Law of Conservation of Packets
- Packet loss formerly caused by:
  - ☐ Noise on transmission line
  - ☐ Packet discard due to congestion
- Congestion Prevention addresses two potential problems:
  - ☐ Network capacity (Congestion) window
  - ☐ Receiver window
  - ☐ Send minimum of two windows

Slow Start Algorithm

- Sender initializes congestion window to max segment size
- Each successful ACK before timeout doubles max segment size
- Window grows exponentially until timeout or receiver window size

TCP Timer Management

- Retransmission timer is most important.
- How long should timeout interval be?
- Unlike DLL, transport round trip time (RTT) is highly variable
- Solution: continually adjust timeout interval
- Problem: handling retransmitted segments
  - ACK comes in: which segment does it belong to?
  - Wrong guess contaminates RTT estimate
  - Fix: ignore retransmitted segments for estimate of RTT

UDP

UDP address applications such as client-server having only one request and reply (formal connection simply adds overhead).

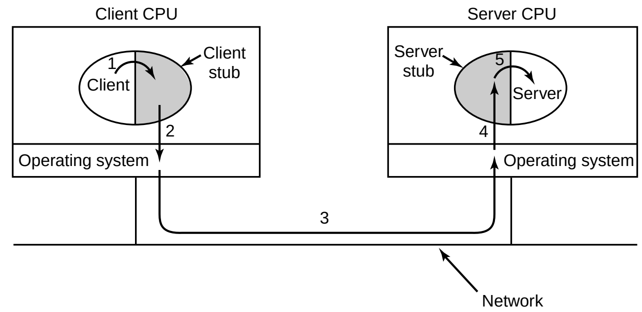
32 Bits

Source port	Destination port
UDP length	UDP checksum

Remote Procedure Call

Arose from the desire to have a programming-language interface similar to a function call.

- Makes network applications easier to program
- Key ideas:
  - ☐ Process on machine 1 calls procedure that executes on machine 2
  - ☐ Message passing is invisible to the user
- Implementation:
  - ☐ Client is bound to **client stub**
  - ☐ Server is bound to **server stub**
  - ☐ Steps:



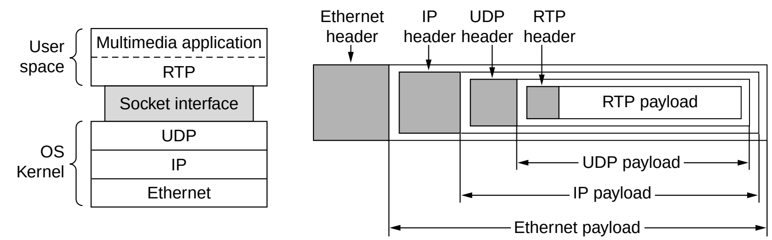
Remote Procedure Call

- Problems or Issues
  - ☐ Pointer parameters
    - ◦ Separate address spaces (and machines) makes it impossible in general
    - ◦ Some tricks can be used to work for specific cases
  - ☐ Precise size of parameter not being specified
  - ☐ Deducing type of the parameter
  - ☐ Global variables cannot be shared across machines

The Real-Time Transport Protocol (RTP)

UDP is used widely in real-time multimedia applications but lacks some features.

- Why not use TCP?
- RTP is a generic protocol used for transmission of streaming multimedia
  - ☐ Position in protocol stack is in user space and (normally) on top of UDP



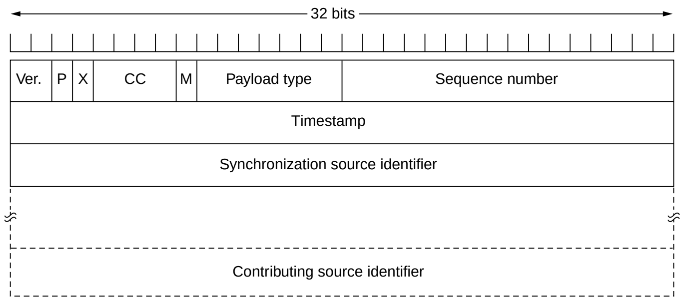
(a) (b)

RTP Functionality

- Multiplex several real-time data streams onto a single UDP stream
  - ☐ Can be unicast or multicast
  - ☐ Not treated specially by routers
- Operation
  - ☐ Each packet is sequenced
  - ☐ No flow/error control, no acks, no retransmissions
  - ☐ Multiple samples per payload are allowed
  - ☐ Timestamps are used to assist in buffering

RTP Header

- Three words with potential for extensions



RTP Header

- Fields
  - ☐ Version: current version, is 2
  - ☐ P: padding
  - ☐ X: extension header presence indicator
  - ☐ CC: number of contributing sources
  - ☐ M: marker bit for specific applications
  - ☐ Type: encoding algorithm
  - ☐ Sequence number: counts packets
  - ☐ Timestamp: time relative to stream start
  - ☐ Synchronization Source Identifier: identifies owning stream
  - ☐ Contributing Source Identifier: identifies what streams might be mixed
- **Real Time Transport Control Protocol** (RTCP): 'sibling' protocol used for feedback, synchronization, and user interface

Wireless TCP and UDP

Theoretically, transport protocols are independent of underlying network technology.

- Existing TCP is optimized for *wireline* networks
- Congestion control algorithms cause poor wireless performance
  - ☐ congestion: slow down
  - ☐ High error rate: speed up
- Solutions:
  - ☐ Indirect TCP
  - ☐ Modify network layer code

Transport Layer Performance Issues

In computer networks, numerous complex interactions with unforseen consequences exist

- Little underlying theory is of any use
- Issues to be examined:
  1. Performance problems
  2. Measuring network performance
  3. System design for better performance
  4. Fast TPDU processing
  5. Protocols for high(er)-performance networks

Performance Problems

Other than congestion (already examined), performance can also degrade due to various hardware and software problems.

- Structural resource imbalances
- Synchronously triggered overloads:
  - ☐ Broadcast storms
  - ☐ Simultaneous reboots
- Poor system tuning
  - ☐ Low buffer space
  - ☐ Improper timeout interval
- Outdated protocols over high-performance networks
- Jitter

Measuring Network Performance

Improvement of network performance is inexact.

- Repeat until (performance.isBetter())
  - ☐ Measure relevant network parameters/performance
  - ☐ Understand it
  - ☐ Change one parameter
- Pitfalls:
  - ☐ Sample size
  - ☐ Representative traffic samples
  - ☐ Coarse-grained clocks
  - ☐ Unexpected projects
  - ☐ Caching
  - ☐ Understanding of measurements
  - ☐ Extrapolation of results

System Design for Performance

Improvement of a bad design results in an optimized bad design.

- Rules of thumb:
  - ☐ CPU speed is more important
  - ☐ Reduce software overhead
  - ☐ Minimize context switches
  - ☐ Minimize copying
  - ☐ Improving bandwidth is easy, delay is not
  - ☐ Avoid congestion instead of recovering from it
  - ☐ Avoid timeouts

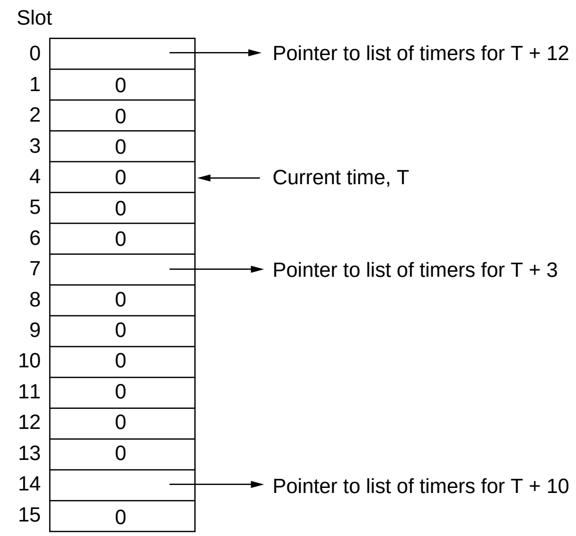
Fast TPDU Processing

Essentially, the main obstacle to fast networking is protocol software.

- Two components: reduce overhead per TPDU and per byte
- Key: separate out normal case
- Fast-path processing, sender:
  - ☐ Test for normal TPDU
  - ☐ maintain prototype TPDU header in transport entity
  - ☐ Minimize time to update changeable header variables
- Fast-path processing, receiver:
  - ☐ Locate connection record for incoming TPDU
  - ☐ Implement header prediction
  - ☐ Copy data to user, compute checksum

Buffer and Timer Management

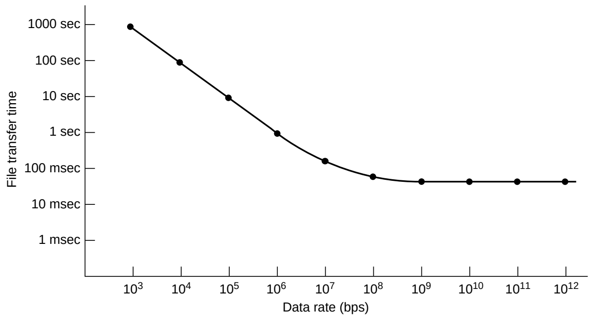
- Buffer management: avoid unnecessary copying
- Timer management: optimize for non-expiration
  - ☐ Common scheme: linked list to store timers
  - ☐ Optimization: timing wheel:



Gigabit Network Protocol Problems

Old protocols are inadequate for newer high-bandwidth networks.

- 16 or 32 bits isn't enough
- Communication speeds are catching up to CPU speeds
- Go back n performs poorly on lines with a large *bandwidth delay product*
- Variance in packet arrival times can be as important as mean delay
- Gigabit lines limited by delay, not bandwidth:



Gigabit Network Protocol Design

Design for speed, not for bandwidth optimization.

- Minimize processing time instead of number of bits on the wire
- Avoid unnecessary special-purpose hardware
- Avoid feedback
- Simplify packet layout
- Separately checksum header and data
- Allow large maximum data size
- Concentrate on successful case in the protocol