End-to-End Communication

Goal: Interconnect multiple LANs.

Why?

- Physical limitations on the number of hosts and distance of link.
- Intrinsic performance limitations.

Problems:

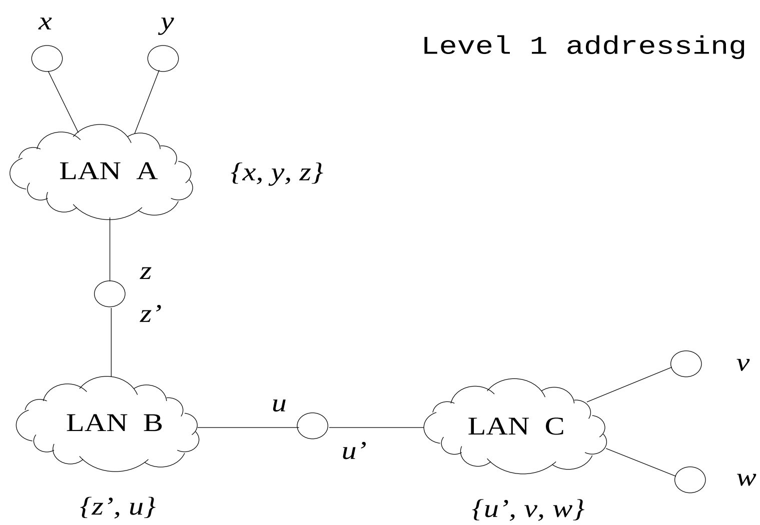
- Diverse LANs; how to make them talk to each other (internetworking)?
- How to choose paths (routing)?
- How to dynamically regulate flow (congestion control)?

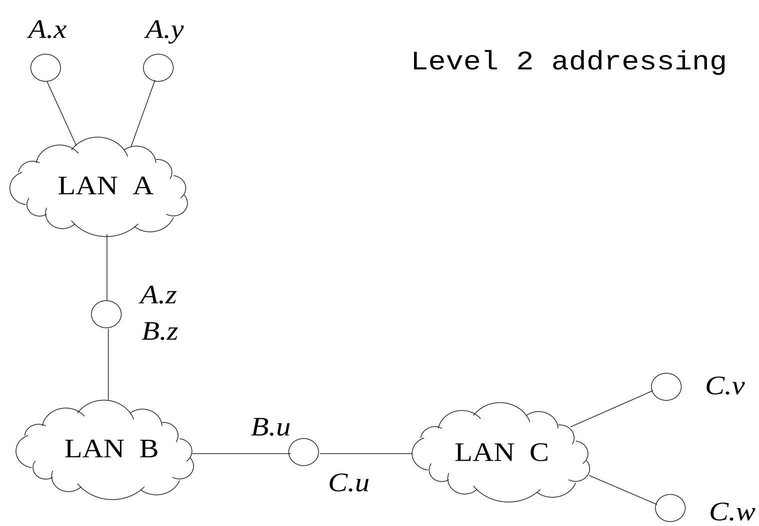
- How to provide QoS (network support/end system support)?
- How to render transparent and efficient network services (e.g., network computing)?
- How to achieve robustness and fault-tolerance?

Translation problem of internetworking:

- address translation (LAN addr. $\leftrightarrow$ WAN addr.)
- protocol translation
  - frame format
  - MAC

$\longrightarrow$ minimum necessary mechanism

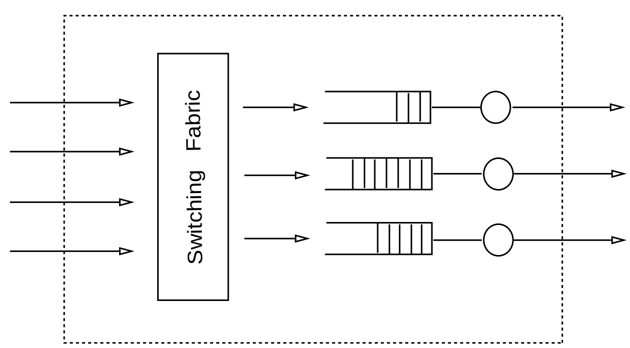

x
y
Level 1 addressing
LAN A
{x, y, z}
z
z'
v
LAN B
u
u'
LAN C
w
{z', u}
{u', v, w}


A.x
A.y
Level 2 addressing
LAN A
A.z
B.z
C.v
LAN B
B.u
C.u
LAN C
C.w

Packet switching vs. circuit switching

Switch design:

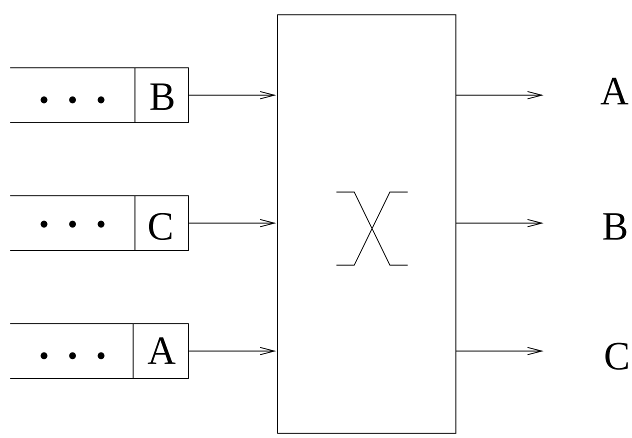
- Hardware (e.g., shuffle-exchange network).
- Software (workstation as router or gateway).
- Hybrid (e.g., DSP).



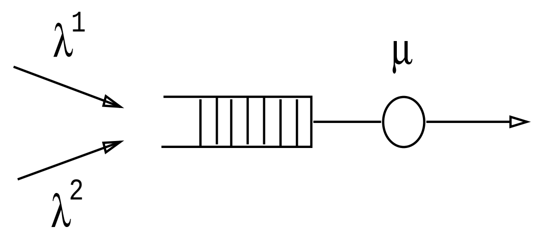
Problem with input-buffered switch design:

$\longrightarrow$ head-of-line blocking

In general, less efficient than output-buffered switches.



Logical switch:



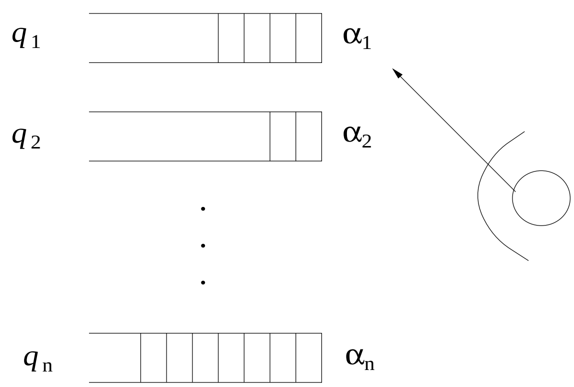
- Enqueueing (who-to-drop, overwrite).
- Dequeueing (FIFO, weighted fair queueing).

$\longrightarrow$ scheduling with real-time constraints (O.S.)

$\longrightarrow$ real-time systems community

Weighted fair queueing

Given n sources and priority weights $\alpha_1, \alpha_2, \ldots, \alpha_n$, perform *weighted* round-robin on n queues $q_1, q_2, \ldots, q_n$.



Given Δt service time, dequeue $\alpha_i \Delta t$ packets (bits) from queue q_i.

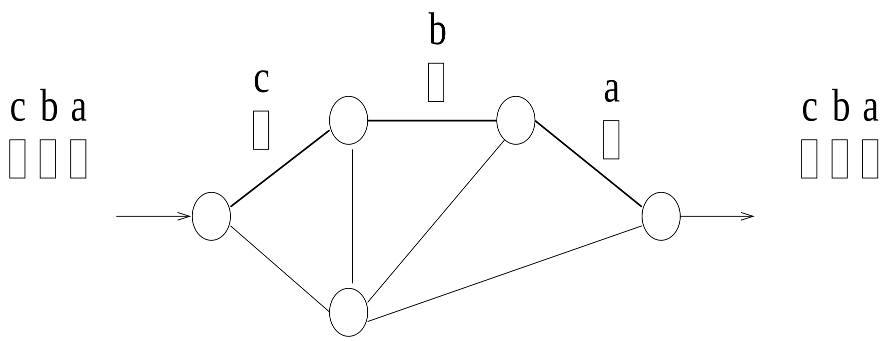
As $\Delta t \to 0$, more finegrained and fair. In practice, need approximation for $\Delta t \gg 0$.

<u>Circuit switching</u>

Establish fixed path (route) from source to destination—channel.

$\longrightarrow$ connection-oriented

All packets belonging to the same connection traverse same path.



- Permanent virtual circuits (PVC). E.g., line leasing.
- Switched virtual circuits (SVC). E.g., regular telephone calls.

Benefit: Simplicity.

- One-time call set-up cost (admission control).
- Smaller routing table.
- Allows simplified switch design.
- Under low packet loss rate, in-order delivery.
- Easier accounting for reservation-based resource allocation.

Drawback: Performance.

- For lengthy connection, “goodness” of initial path may change.
- High initial call set-up cost (real-time applications).
- Less fault-tolerant.

$\longrightarrow$ solution adopted for ATM networks

Following a fixed path:

- source routing
- call set-up

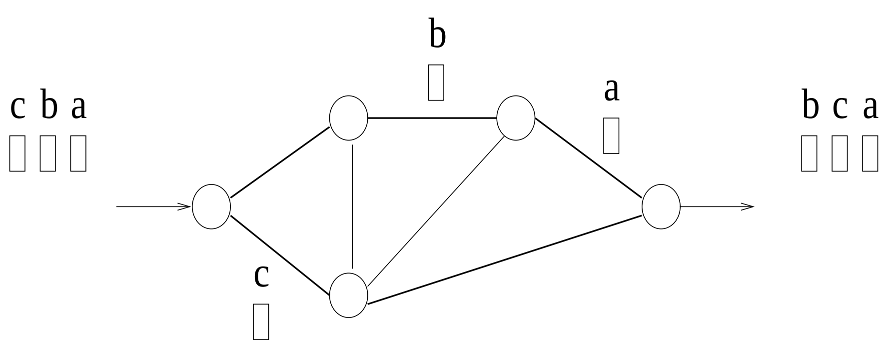
Packet switching

Treat each packet as *independent* unit, with full source/ destination addressing.

$\longrightarrow$ packet as fully autonomous entity

During single conversation, packets may take different routes.

$\longrightarrow$ store-and-forward networks



Benefits: Performance.

- Can adaptively find "good" path for each packet of a conversation.
- More fault-tolerant.
- More responsive to interactive real-time applications.

Drawback: Increased complexity.

- Switch design is more complex.
- bigger routing table.
- Increased processing overhead incurred at switches.
- Out-of-order delivery—re-sequencing cost.

"Message switching."

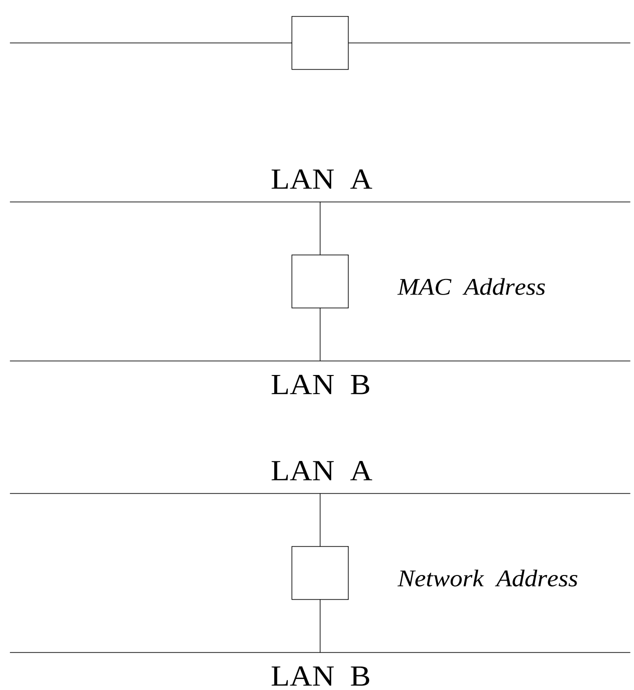
Active network proposal.

$\longrightarrow$ packet contains *program* and *data*

Interconnecting LANs

Methods:

- Repeaters (physical layer).
- Bridges (data link layer).
- Routers or gateways (network layer).



Bridges

- Promiscuous mode.
- Backward learning (track source addresses).
- Transparency (e.g., IEEE LAN standards).
- Spanning Tree (loop problem).
  - Goal: Build spanning tree rooted at lowest ID (serial number) bridge.
  - Send out/forward configuration messages containing smallest *locally observed* ID with distance information.
  - Stop generating and only forward if own priority is overridden.
  - Update shortest distance by 1.
  - Eventually stabilizes to shortest path solution.
  - Perlman's method is a form of *self-stabilization*.

Routers

Maintain shortest-path table to relevant nodes. Forward network layer packets (e.g., IP datagrams) based upon this table.

$\longrightarrow$ routing problem

Actually:

- If network address matches local address, then use ARP to look up MAC address of the destination and pass to data link layer.
- If it does not, then use ARP to look up MAC address of next hop and pass to data link layer.

$\longrightarrow$ two-level addressing

Benefit of bridge:

- Simple form of modularization.
- Achieves load splitting (performance), fault-tolerance, security.

Drawback of bridge-based design vis-à-vis routers?

Internet Protocol (IP)

Goals:

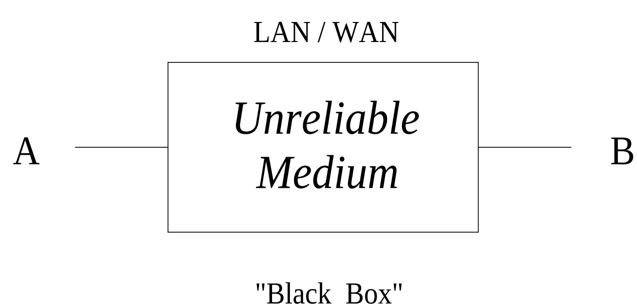
- Interconnect diverse LANs into one logical entity.
- Implement *best effort* (unreliable, connectionless) service model.

Specifies

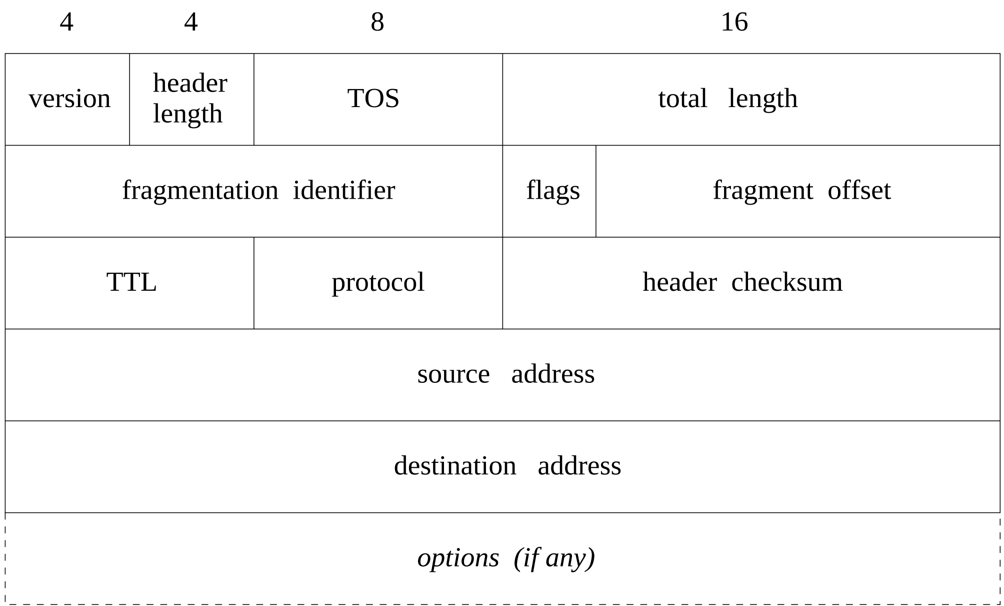
- Common language for carrying out non-LAN-specific conversations (protocol standards).
- Functionality and design philosophy.

Best effort vs. guaranteed service:

- Much easier to implement best effort service; no resource reservation.
- Simplifies router design but increases complexity of end stations $\longrightarrow$ trade-off
- Necessitates higher-up functional layer (transport layer) to achieve reliable transmission over unreliable medium.
- Duplication of work.
- Routers/switches already becoming more complex due to QoS; why not dispense with transport layer . . .



IP packet (datagram) format:



- Header length: in 4 byte (word) units.
- TOS (type-of-service): Most routers do not support.
- 4 bytes used for fragmentation.
- TTL (time-to-live): Prevent cycling (default 64).
- Protocol: demultiplexing key (TCP 6, UDP 17).

Fragmentation and reassembly:

LAN has *maximum transmission unit* (MTU)—maximum frame size; e.g., Ethernet 1500 B, FDDI 4500 B.

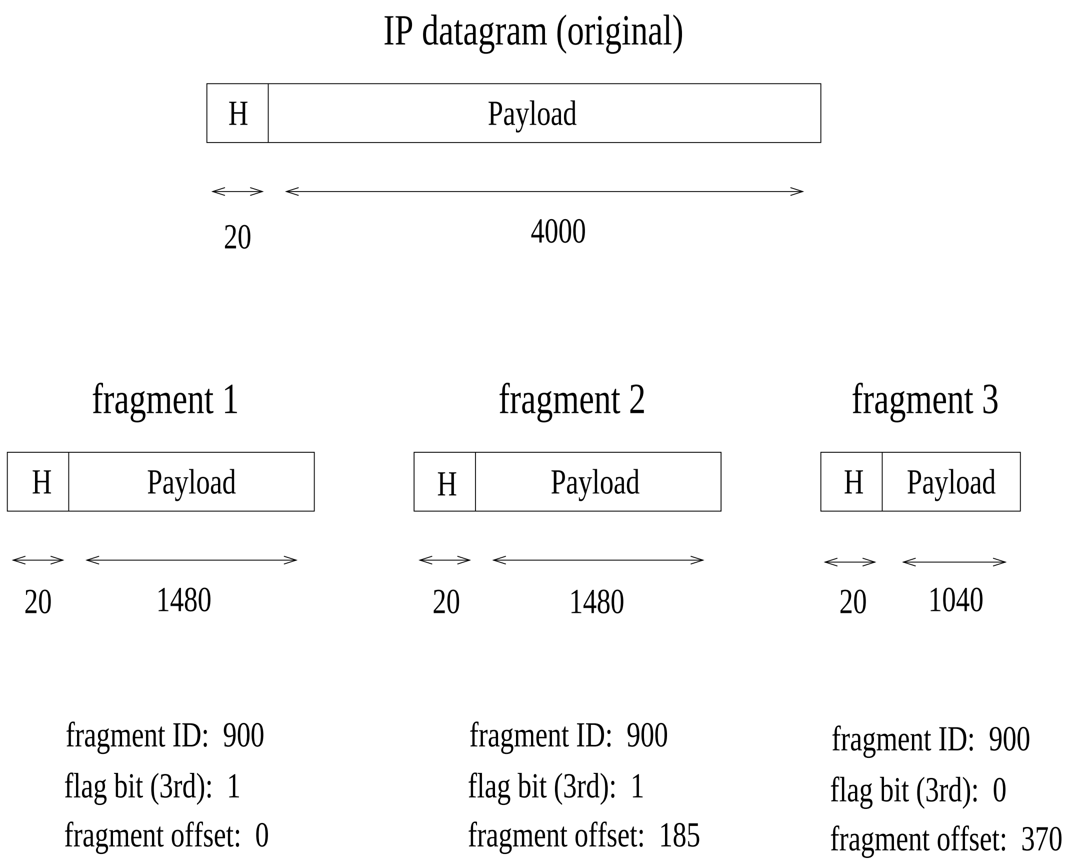
$\longrightarrow$ potential size mismatch problem (64 kB)

$\longrightarrow$ …when hopping from LAN to LAN

Solution: Fragment IP packet when needed, maintain sequencing information, then reassemble at destination.

- Assign unique fragmentation ID.
- Set 3rd flag bit if fragmentation in progress.
- Sequence fragments using offset in units of 8 bytes.

Example: IP fragmentation (Ethernet MTU)



Note: Each fragment is an *independent* IP packet.

Destination discards all fragments of an IP packet if one is lost.

$\longrightarrow$ fragmentation problem

$\longrightarrow$ exists at several boundaries in protocol stack

$\longrightarrow$ set 2nd flag bit to disable fragmentation

TCP: Negotiate at start-up TCP segment (packet) size based on MTU; 1 kB or 512 B are common. Seek compatibility with IP.

IP address format:

		7	24
Class A	0	Network ID	Host ID
		14	16
Class B	1 0	Network ID	Host ID
		21	8
Class C	1 1 0	Network ID	Host ID
		28	
Class D	1 1 1 0	Multicast Address	

Dotted decimal notation: 10000000 00001011 00000011 00011111 $\leftrightarrow$ 128.11.3.31

Symbolic name to IP address translation—domain name server (DNS).

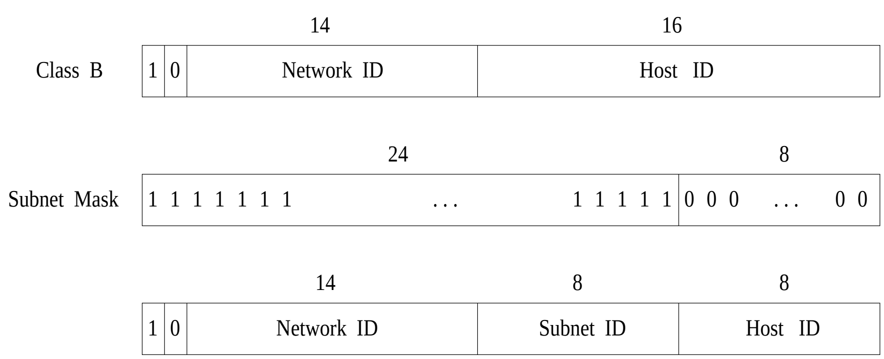
Notice hierarchical organization ("2-level").

Each interface (NIU) has an IP address; single host can have multiple IP addresses.

Running out of unused addresses (IPv6).

Potential problem: Waste of address space. Giving each network own network ID is inefficient.

Solution: *Subnetting*; group several physical networks into one.



To determine subnet ID:

- Perform ANDing of IP address and subnet mask.
- Needed for routing.
- 3-level hierarchy (IP).

Forwarding and address resolution:

... mechanics of routing when *routing table* is given.

Subnet ID	Subnet Mask	Next Hop
128.10.2.0	255.255.255.0	Interface 0
128.10.3.0	255.255.255.0	Interface 1
128.10.4.0	255.255.255.0	128.10.4.250

Either destination host is directly connected on the same LAN or not.

Table look-up I:

- For each entry, compute *DestSubnetID* = *DestAddr* AND *SubnetMask*.
- Compare *DestSubnetID* with *SubnetID* and take action.

One more task left: Translate destination host (or next hop node) IP address into LAN address.

$\longrightarrow$ address resolution protocol (ARP)

Table look-up II:

- If ARP table contains entry, using LAN address send to destination.
- If ARP table does not contain entry, broadcast ARP Request packet with destination IP address.
- Encapsulate ARP packet into LAN frame.
- Update ARP table upon receipt of feedback.

Dynamically maintain ARP table:

- Use timer for each entry (15 min) to invalidate entries.
- Upon receipt of ARP Request (if applicable), update own ARP if entry is absent; ARP Request frame contains source IP address and LAN address.

Standards documents: RFC (Request for Comments)

- RFC 791 (IP)
- RFC 826 (ARP)
- RFC 903 (RARP)
- RFC 894 (Ethernet)
- RFC 793 (TCP)
- RFC 768 (UDP)
- etc.

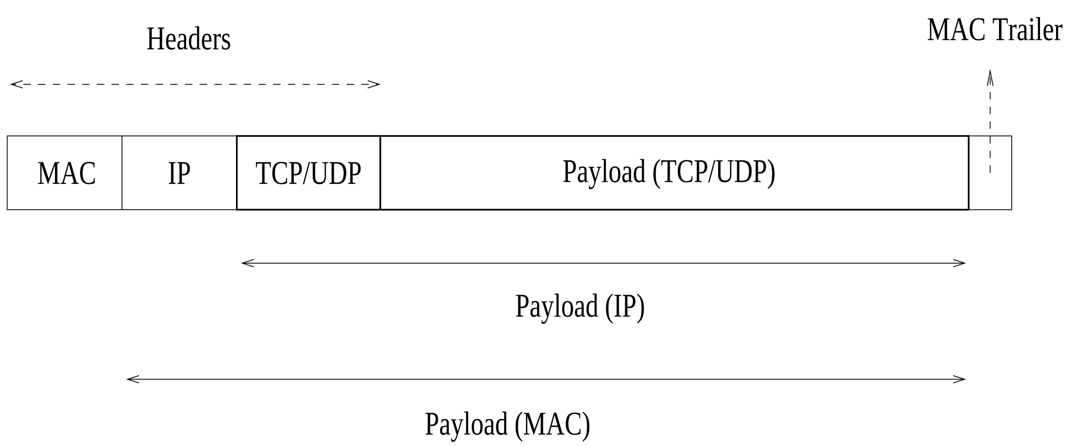
Transport Protocols: TCP/UDP Structure

$\longrightarrow$ end-to-end mechanism

$\longrightarrow$ runs on top of link-based mechanism

$\longrightarrow$ treat network layer as black box

Three-level encapsulation:



Network layer assumptions:

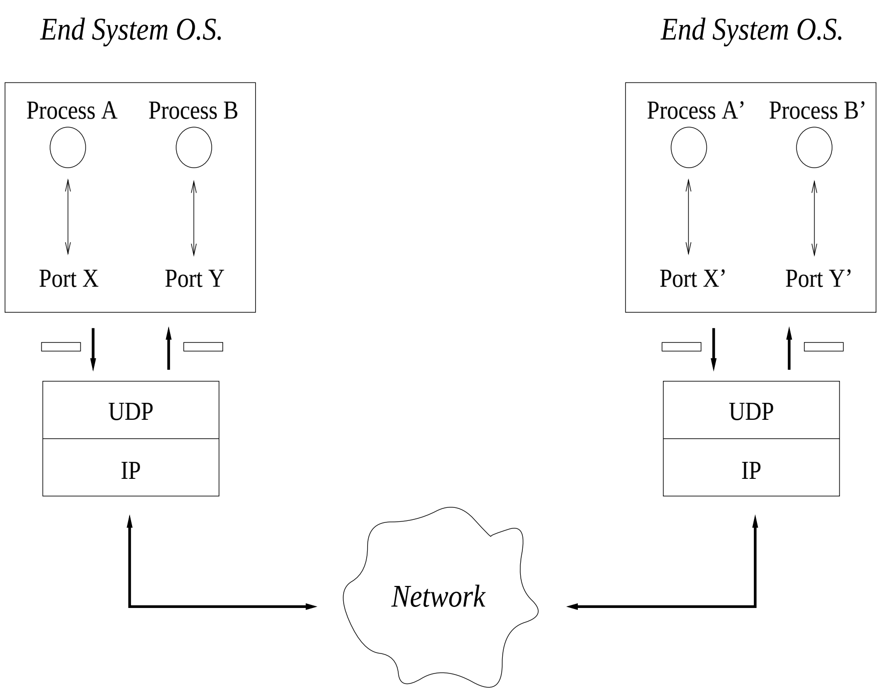
- unreliable
- out-of-order delivery (in general)
- absence of QoS guarantees (delay, throughput etc.)
- insecure (IPv4)

Additional (informal) performance properties:

- works "fine" under low load conditions
- can break down under high load conditions
- behavior range predictable (to certain extent)

Goal of UDP: Process identification ("multiplexing").

$\longrightarrow$ port number as process demux key



- form of end host processing (O.S.)
- generally: end system support (e.g., scheduling)

UDP packet format:

2	2
Source Port	Destination Port
Length	Checksum
Payload	

Checksum calculation (pseudo header):

4		
Source Address		
Destination Address		
$00 \cdots 0$	Protocol	UDP Length

Goals of TCP:

- process identification
- reliable communication (ARQ)
- speedy communication (congestion/flow control)
- segmentation

$\longrightarrow$ connection-oriented (i.e., stateful)

$\longrightarrow$ complex mixture of functionalities

Segmentation task: Provide "stream" interface to higher level protocols

$\longrightarrow$ view: contiguous stream of bytes

- segment stream of bytes into blocks or *segments* of fixed size
- segment size determined by TCP MTU (Maximum Transmission Unit)
- use also for reliability mechanism

TCP packet format:

<table>
<tr><td colspan="8">2</td><td>2</td></tr>
<tr><td colspan="8">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="9">Sequence Number</td></tr>
<tr><td colspan="9">Acknowledgement Number</td></tr>
<tr><td>Header Length</td><td></td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window Size</td></tr>
<tr><td colspan="8">Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="9">Options (if any)</td></tr>
<tr><td colspan="9">DATA (if any)</td></tr>
</table>

- Sequence Number: position of first byte of payload
- Acknowledgement: next byte of data expected (receiver)
- Header Length (4 bits): 4 B units
- URG: urgent pointer flag
- ACK: ACK packet flag
- PSH: override TCP buffering
- RST: reset connection
- SYN: establish connection
- FIN: close connection
- Window Size: receiver's advertised window size
- Checksum: prepend pseudo-header
- Urgent Pointer: byte offset in current payload where urgent data begins
- Options: MTU; take min of sender & receiver (default 556 B)

Checksum calculation (pseudo header):

4

Source Address		
Destination Address		
$00 \cdots 0$	Protocol	UDP Length

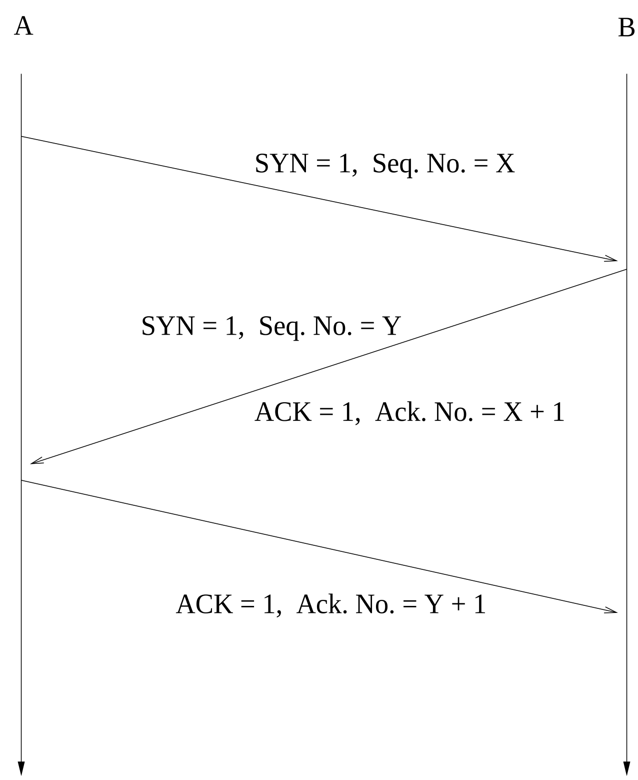
Nagle's algorithm:

- do not want to send too many 1 B payload packets
- rule: connection can have only one such unacknowledged packet outstanding
- while waiting for ACK, incoming bytes are accumulated (i.e., buffered)

... compromise between real-time constraints and efficiency.

$\longrightarrow$ useful for `telnet`-type applications

TCP connection establishment (3-way handshake):



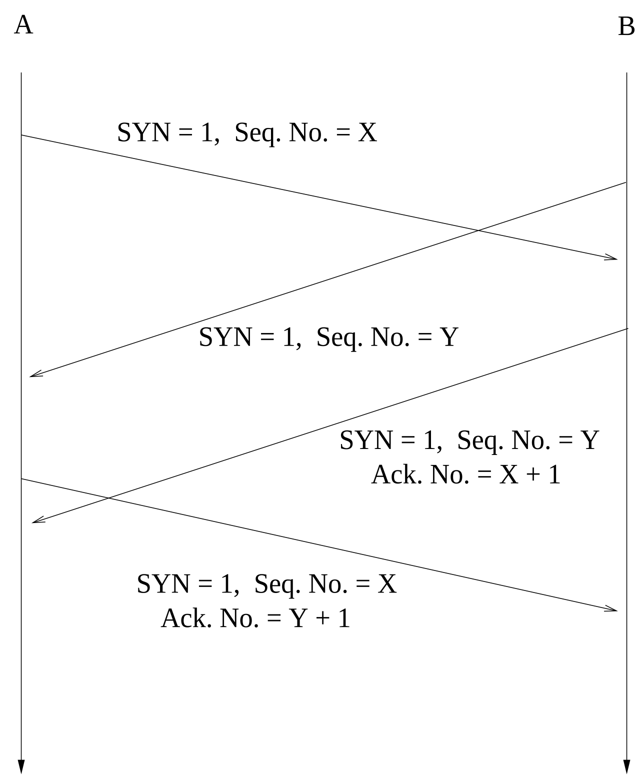
- X, Y are chosen randomly
- piggybacking
- sequence number prediction
- lingering packet problem

2-person consensus problem: Are A and B in agreement about the state of affairs after 3-way handshake?

$\longrightarrow$ impossibility, in general

$\longrightarrow$ lunch date problem

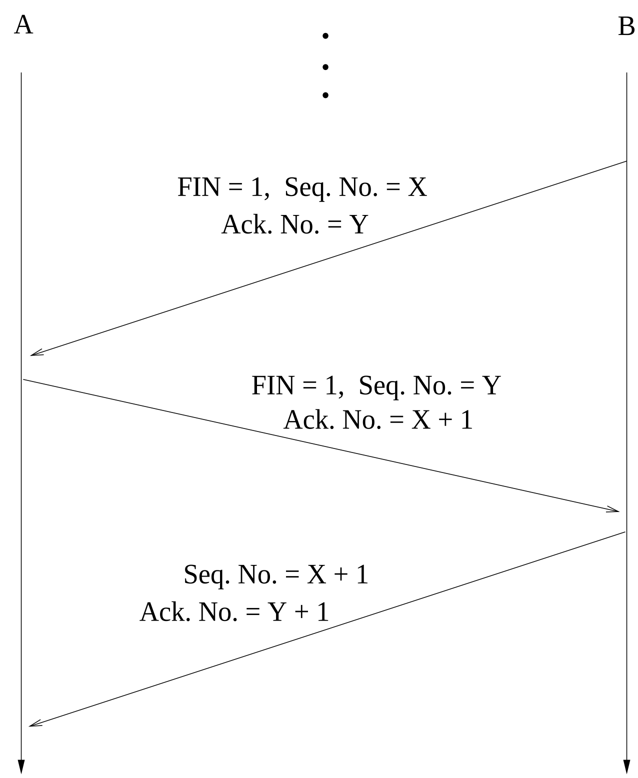
Call Collision:



$\longrightarrow$ only single TCB gets allocated

$\longrightarrow$ unique full association

TCP connection termination:



- full duplex
- half duplex

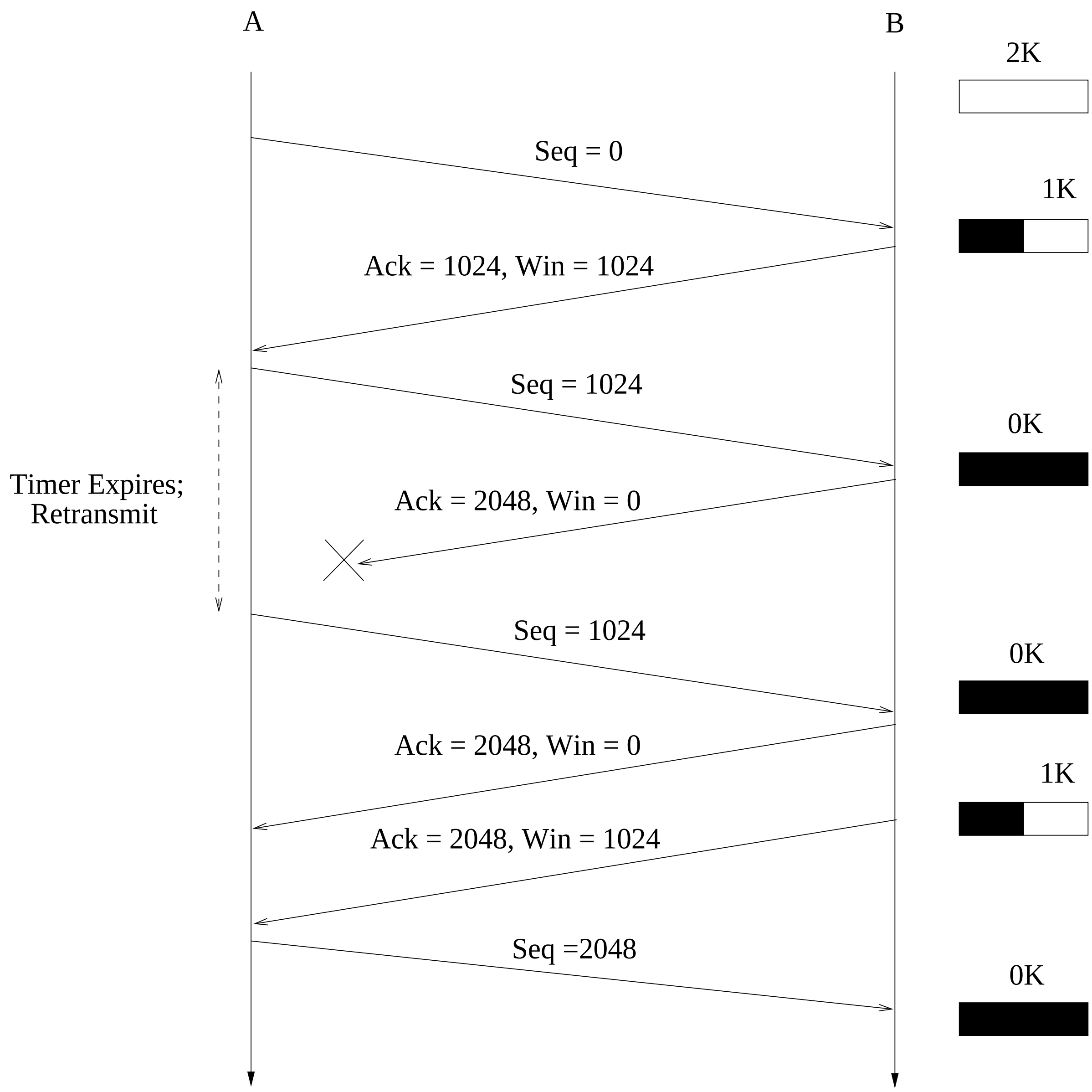
More generally, finite state machine representation of TCP's control mechanism:

TCP's State-transition Diagram comes here

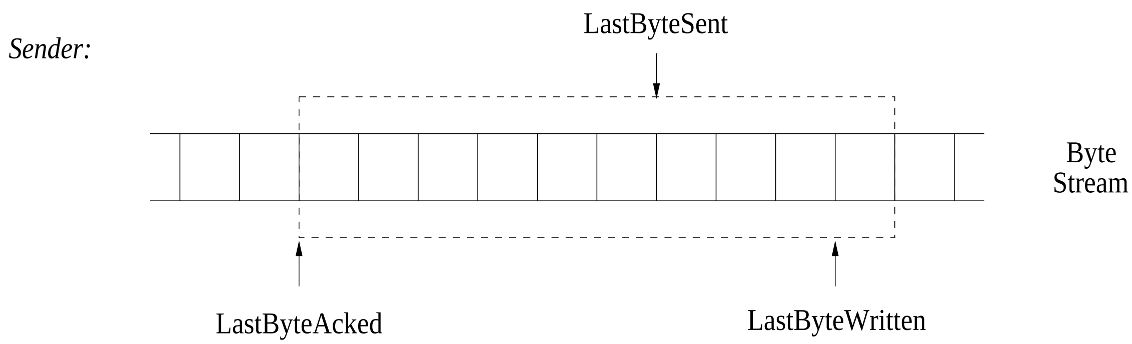
Features to notice:

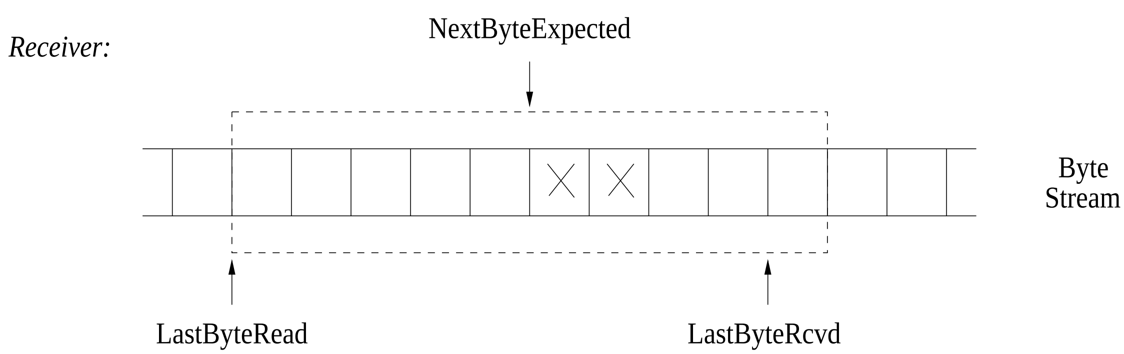
- Connection set-up:
  - client's transition to `ESTABLISHED` state without ACK
  - how is server to reach `ESTABLISHED` if client ACK is lost?
  - TCP: default ACKing executed by all data packets; no extra overhead incurred
  - note: `ESTABLISHED` is macrostate
  - not a complete transition diagram
- Connection tear-down:
  - three normal cases
  - special issue with `TIME WAIT` state

Basic TCP data transfer:



TCP's sliding window protocol





- sender, receiver maintain buffers `MaxSendBuffer`, `MaxRcvBuffer`

Note asynchrony between TCP module and application.

Sender side: maintain invariants

- $\texttt{LastByteAcked} \leq \texttt{LastByteSent} \leq \texttt{LastByteWritten}$
- $\texttt{LastByteWritten} - \texttt{LastByteAcked} < \texttt{MaxSendBuffer}$

$\longrightarrow$ buffer flushing (advance window)

$\longrightarrow$ application blocking

- $\texttt{LastByteSent} - \texttt{LastByteAcked} \leq \texttt{AdvertisedWindow}$

Thus,

$$\begin{aligned}\texttt{EffectiveWindow} = \texttt{AdvertisedWindow} &- \\ &(\texttt{LastByteSent} - \texttt{LastByteAcked})\end{aligned}$$

$\longrightarrow$ upper bound on new send volume

Receiver side: maintain invariants

- $\texttt{LastByteRead} < \texttt{NextByteExpected} \leq \texttt{LastByteRcvd} + 1$
- $\texttt{LastByteRcvd} - \texttt{NextByteRead} < \texttt{MaxRcvBuffer}$

$\longrightarrow$ buffer flushing (advance window)

$\longrightarrow$ application blocking

Thus,

$$\begin{aligned}\texttt{AdvertisedWindow} = \texttt{MaxRcvBuffer} &- \\ &(\texttt{LastByteRcvd} - \texttt{LastByteRead})\end{aligned}$$

Three problems:

How to let sender know of changed in receiver window size after `AdvertisedWindow` becomes 0?

- trigger ACK event on receiver side when `AdvertisedWindow` becomes positive
- sender periodically sends 1-byte probing packet

$\longrightarrow$ design choice: smart sender/dumb receiver

Silly window syndrome: Assuming receiver buffer is full, what if application reads one byte at a time with long pauses?

- can cause excessive 1-byte traffic
- if `AdvertisedWindow` $<$ MSS then set `AdvertisedWindow` $\leftarrow 0$

Sequence number wrap-around problem: recall sufficient condition

$$\texttt{SenderWindowSize} < (\texttt{MaxSeqNum} + 1)/2$$

$\longrightarrow$ 32-bit sequence space/16-bit window space

However, more importantly, time until wrap-around important due to possibility of roaming packets.

bandwidth	time until wrap-around †
T1 (1.5 Mbps)	6.4 hrs
Ethernet (10 Mbps)	57 min
T3 (45 Mbps)	13 min
FDDI (100 Mbps)	6 min
OC-3 (155 Mbps)	4 min
OC-12 (622 Mbps)	55 sec
OC-24 (1.2 Gbps)	28 sec

† From P & D for 32-bit sequence space

Even more importantly, "keeping-the-pipe-full" consideration.

bandwidth	delay-bandwidth product †
T1 (1.5 Mbps)	18 kB
Ethernet (10 Mbps)	122 kB
T3 (45 Mbps)	549 kB
FDDI (100 Mbps)	1.2 MB
OC-3 (155 Mbps)	1.8 MB
OC-12 (622 Mbps)	7.4 MB
OC-24 (1.2 Gbps)	14.8 MB

† From P & D for 100 ms latency

RTT estimation

. . . important to not underestimate nor overestimate.

Karn/Partridge: Maintain running average with precautions

$$\texttt{EstimateRTT} \leftarrow \alpha \cdot \texttt{EstimateRTT} + \beta \cdot \texttt{SampleRTT}$$

- `SampleRTT` computed by sender using timer
- $\alpha + \beta = 1$; $0.8 \leq \alpha \leq 0.9$, $0.1 \leq \beta \leq 0.2$
- $\texttt{TimeOut} \leftarrow 2 \cdot \texttt{EstimateRTT}$ or

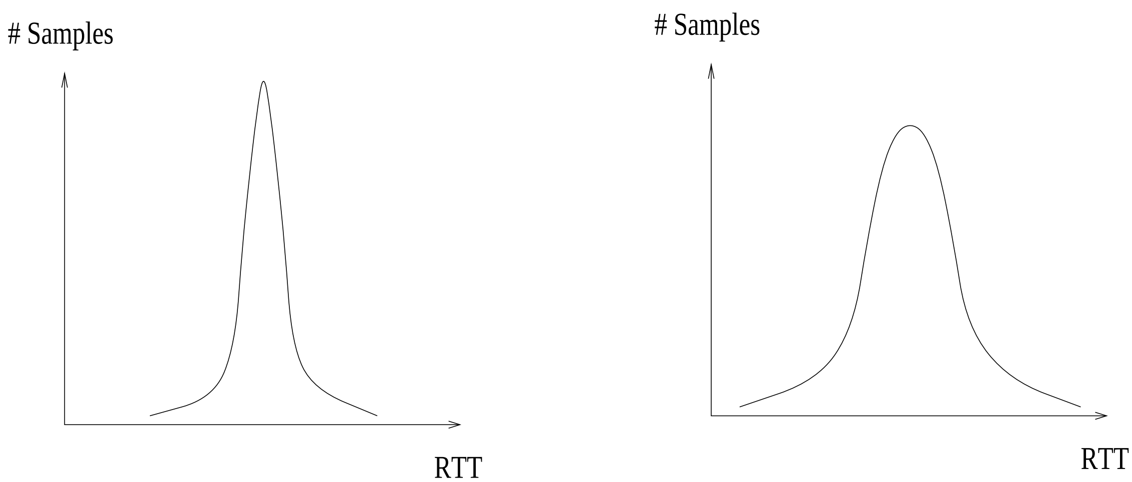
  $\texttt{TimeOut} \leftarrow 2 \cdot \texttt{TimeOut}$ (if retransmit)

$\longrightarrow$ need to be careful when taking `SampleRTT`

$\longrightarrow$ infusion of complexity

$\longrightarrow$ still remaining problems

Hypothetical RTT distribution:



$\longrightarrow$ need to account for variance

Jacobson/Karels:

- $\texttt{Difference} = \texttt{SampleRTT} - \texttt{EstimatedRTT}$
- $\texttt{EstimatedRTT} = \texttt{EstimatedRTT} + \delta \cdot \texttt{Difference}$
- $\texttt{Deviation} = \texttt{Deviation} + \delta(|\texttt{Difference}| - \texttt{Deviation})$

Here $0 < \delta < 1$.

Finally,

- $\texttt{TimeOut} = \mu \cdot \texttt{EstimatedRTT} + \phi \cdot \texttt{Deviation}$

where $\mu = 1$, $\phi = 4$.

$\longrightarrow$ persistence timer

$\longrightarrow$ how to keep multiple timers in UNIX