**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. ↔ WAN addr.)

• protocol translation
  − frame format
  − MAC

→ minimum necessary mechanism
Level 1 addressing

Level 2 addressing
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:

\[ \rightarrow \text{head-of-line blocking} \]

In general, less efficient than output-buffered switches.
Logical switch:

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

→  scheduling with real-time constraints (O.S.)

→  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 $\Delta t$ 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 $\Delta t \gg 0$. 
Circuit switching

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

→ 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.

→ 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.

    ➔ packet as fully autonomous entity

During single conversation, packets may take different routes.

    ➔ 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.

→ 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.

\[\rightarrow\text{ routing problem}\]

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.

\[\rightarrow\text{ two-level addressing}\]
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 — 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 . . .

![Diagram of unreliable medium between A and B connected to LAN/WAN.]("Black Box")
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).

<table>
<thead>
<tr>
<th></th>
<th>version</th>
<th>header length</th>
<th>TOS</th>
<th>total length</th>
<th>fragmentation identifier</th>
<th>flags</th>
<th>fragment offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **options (if any)**
Fragmentation and reassembly:

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

→ potential size mismatch problem (64 kB)

→ ...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.

\[ \rightarrow \text{fragmentation problem} \]

\[ \rightarrow \text{exists at several boundaries in protocol stack} \]

\[ \rightarrow \text{set 2nd flag bit to disable fragmentation} \]

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:

<table>
<thead>
<tr>
<th>Class</th>
<th>Network ID</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>1 1 0</td>
<td></td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>1 1 1 0</td>
<td>Multicast Address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network ID</th>
<th>Host ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Subnet ID</th>
<th>Subnet Mask</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.10.2.0</td>
<td>255.255.255.0</td>
<td>Interface 0</td>
</tr>
<tr>
<td>128.10.3.0</td>
<td>255.255.255.0</td>
<td>Interface 1</td>
</tr>
<tr>
<td>128.10.4.0</td>
<td>255.255.255.0</td>
<td>128.10.4.250</td>
</tr>
</tbody>
</table>

Either destination host is directly connected on the same LAN or not.
Table look-up I:

- For each entry, compute $DestSubnetID = DestAddr \text{ AND } SubnetMask$.

- Compare $DestSubnetID$ with $SubnetID$ and take action.

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

$\rightarrow$ 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

- end-to-end mechanism
- runs on top of link-based mechanism
- 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").

→ port number as process demux key

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

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

Payload

Checksum calculation (pseudo header):

<table>
<thead>
<tr>
<th>Source Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination Address</td>
</tr>
<tr>
<td>00 ··· 0</td>
</tr>
</tbody>
</table>
Goals of TCP:

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

→ connection-oriented (i.e., stateful)
→ complex mixture of functionalities
Segmentation task: Provide “stream” interface to higher level protocols

\[\rightarrow\text{ view: contiguous stream of bytes}\]

- 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>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acknowledgement Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Header Length</th>
<th>Options (if any)</th>
<th>__________</th>
<th>__________</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>U A P R S F</td>
<td>S Y I N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checksum</th>
<th>Urgent Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Window Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA (if any)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</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):

<table>
<thead>
<tr>
<th>Source Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination Address</td>
</tr>
<tr>
<td>00 · · · 0</td>
</tr>
</tbody>
</table>

4
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.

→ 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?

$\rightarrow$ impossibility, in general

$\rightarrow$ lunch date problem
Call Collision:

\[\text{SYN} = 1, \ \text{Seq. No.} = X\]

\[\text{SYN} = 1, \ \text{Seq. No.} = Y\]

\[\text{Ack. No.} = X + 1\]

\[\text{SYN} = 1, \ \text{Seq. No.} = X\]

\[\text{Ack. No.} = Y + 1\]

\[\rightarrow \) only single TCB gets allocated

\[\rightarrow \) 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 \textbf{MaxSendBuffer}, \textbf{MaxRcvBuffer}
Note asynchrony between TCP module and application.

Sender side: maintain invariants

- \( \text{LastByteAcked} \leq \text{LastByteSent} \leq \text{LastByteWritten} \)
- \( \text{LastByteWritten} - \text{LastByteAcked} < \text{MaxSendBuffer} \)
  
  \[ \rightarrow \text{ buffer flushing (advance window)} \]
  
  \[ \rightarrow \text{ application blocking} \]

- \( \text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow} \)

Thus,

\[
\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})
\]

\[ \rightarrow \text{ upper bound on new send volume} \]
Receiver side: maintain invariants

- \( \text{LastByteRead} < \text{NextByteExpected} \leq \text{LastByteRcvd} + 1 \)
- \( \text{LastByteRcvd} - \text{NextByteRead} < \text{MaxRcvBuffer} \)

\[ \rightarrow \] buffer flushing (advance window)

\[ \rightarrow \] application blocking

Thus,

\[ \text{AdvertisedWindow} = \text{MaxRcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead}) \]
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

\[ \rightarrow \text{ design choice: smart sender/dumb receiver} \]

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

\[ \text{SenderWindowSize} < \frac{(\text{MaxSeqNum} + 1)}{2} \]

\[ \rightarrow \quad \text{32-bit sequence space}/16\text{-bit window space} \]

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

<table>
<thead>
<tr>
<th>bandwidth</th>
<th>time until wrap-around $^\dagger$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>6.4 hrs</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>57 min</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>13 min</td>
</tr>
<tr>
<td>FDDI (100 Mbps)</td>
<td>6 min</td>
</tr>
<tr>
<td>OC-3 (155 Mbps)</td>
<td>4 min</td>
</tr>
<tr>
<td>OC-12 (622 Mbps)</td>
<td>55 sec</td>
</tr>
<tr>
<td>OC-24 (1.2 Gbps)</td>
<td>28 sec</td>
</tr>
</tbody>
</table>

$^\dagger$ From P & D for 32-bit sequence space
Even more importantly, “keeping-the-pipe-full” consideration.

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Delay-bandwidth product †</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>18 kB</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>122 kB</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>549 kB</td>
</tr>
<tr>
<td>FDDI (100 Mbps)</td>
<td>1.2 MB</td>
</tr>
<tr>
<td>OC-3 (155 Mbps)</td>
<td>1.8 MB</td>
</tr>
<tr>
<td>OC-12 (622 Mbps)</td>
<td>7.4 MB</td>
</tr>
<tr>
<td>OC-24 (1.2 Gbps)</td>
<td>14.8 MB</td>
</tr>
</tbody>
</table>

† From P & D for 100 ms latency
**RTT estimation**

... important to not underestimate nor overestimate.

Karn/Partridge: Maintain running average with precautions

\[ \text{EstimateRTT} \leftarrow \alpha \cdot \text{EstimateRTT} + \beta \cdot \text{SampleRTT} \]

- **SampleRTT** computed by sender using timer
- \( \alpha + \beta = 1; \ 0.8 \leq \alpha \leq 0.9, \ 0.1 \leq \beta \leq 0.2 \)
- **TimeOut** \( \leftarrow 2 \cdot \text{EstimateRTT} \) or
  
  \[ \text{TimeOut} \leftarrow 2 \cdot \text{TimeOut} \] (if retransmit)

\[ \rightarrow \text{need to be careful when taking SampleRTT} \]

\[ \rightarrow \text{infusion of complexity} \]

\[ \rightarrow \text{still remaining problems} \]
Hypothetical RTT distribution:

\[ \text{# Samples} \quad \text{RTT} \]

\[ \text{# Samples} \quad \text{RTT} \]

\[ \rightarrow \quad \text{need to account for variance} \]
Jacobson/Karels:

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

Here $0 < \delta < 1$.

Finally,

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

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

$\rightarrow$ persistence timer

$\rightarrow$ how to keep multiple timers in UNIX