Frequently asked questions from the previous class surveys

- False positives in checksums?
- Frames and bit-patterns
- What if the IP address is changed mid-transmission?
- Payload length: bytes or words (4-bytes)
- Process-per-message: Is there a separate one per-message?
- Why not send maximum length datagram messages every time?
- Purpose of trailer? End of message or Here’s what’s next
- Why does OSI need 7 layers?
- Why not use TOS in IPv4 for other things?

Topics covered in today’s lecture

- IP routing
- IPv6
- UDP
- TCP

Every network type has a Maximum Transmission Unit (MTU)

- Largest IP datagram that it can carry in its frame
- Smaller than the largest packet-size of network
  - IP datagram needs to fit in the payload of link-layer frame

Fragmentation necessary when datagram path includes network with smaller MTU

- All fragments carry same identifier in Id field
- To enable fragment reassembly
- Chosen by the source host
- If all fragments do not arrive at receiving host?
  - Receiver gives up reassembly [reassembly timeout: 15 seconds RFC0791]
  - Discards fragments that did arrive
- IP does not attempt to recover from missing fragments

IPv4 Packet header

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>ILen</td>
<td>4</td>
</tr>
<tr>
<td>TOS</td>
<td>5</td>
</tr>
<tr>
<td>TTL</td>
<td>7</td>
</tr>
<tr>
<td>Protocol</td>
<td>8</td>
</tr>
<tr>
<td>Checksum</td>
<td>10</td>
</tr>
<tr>
<td>SourceId</td>
<td>12</td>
</tr>
<tr>
<td>DestinationId</td>
<td>16</td>
</tr>
<tr>
<td>Options (variable)</td>
<td>20</td>
</tr>
<tr>
<td>Pad (variable)</td>
<td>32</td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>

(0 4 8 12 16 32)
IPv6 Packet Header

- **Version**: 4 bits (0110)
- **Traffic Class**: 6+2 bits
  - Differentiated Services for QoS
  - Anything that ends in 2 “1” bits is intended for experimental or local use
- **Payload Length**: 20 bits
- **SourceAddr ([16 bytes])**
- **DestinationAddr ([32 bytes])**

IPv6 Packet Header: Some more details [1/2]

- **Version**: 4 bits (0110)
- **Traffic Class**: 6+2 bits
  - Differentiated Services for QoS
  - Anything that ends in 2 “1” bits is intended for experimental or local use
- **Flow Label**: (20 bits)
  - If it is non-zero, serves as a hint to routers and switches with multiple outbound paths that these packets should stay on the same path, so that they will not be reordered
- **Payload length (16 bits)**: Size of payload including extension headers

IPv6 versus IPv4: Key Differences

- Source and destination addresses are 128-bits (16 bytes) in IPv6
- IPv6 treats Options as extension headers
- To simplify processing of packets in routers, IPv6 did away with fragmentation
  - Responsibility for packet fragmentation is at the end points
  - IPv6 hosts must perform:
    1. Path MTU discovery
    2. Perform end-to-end fragmentation
    3. Send packets no larger than the default MTU = 1280
- As of 2014, IPv4 still carried >99% of worldwide Internet traffic
IPv6 Packet Header: Some more details [2/2]

- Next Header (8 bits)
  - Specifies the type of the next header
- Hop Limit (8 bits)
  - Replaces the time-to-live field of IPv4
- Destination and Source Addresses (128-bits or 16 bytes each)
  - Note: The IPv6 packet header has no checksum
  - Transport or application layer protocols are assumed to provide sufficient error detection

Structure of the IPv6 Packet

- IPv6 Packet
  - IPv6 Header
  - Extension Headers
  - Upper Layer Protocol Data Unit (PDU)

IPv6 Extension Headers: The chain of pointers using the Next Header field

- Each extension header must fall on a 64-bit (8-byte) boundary. Use Padding to get there if less than that.

IPv6 Extension Headers:

- IPv6 Header: Next Header=6 (TCP)
  - TCP Segment
- Routing Header: Next Header=43
  - Routing Header: Next Header=6 (TCP)
  - TCP Segment
- Authentication Header: Next Header=51
  - Authentication Header: Next Header=6 (TCP)
  - TCP Segment

Fragmentation Header: 44

Datagram forwarding in IP:

- Datagrams contain IP address of destination
  - Network part uniquely identifies a single physical network
  - Hosts/routers that share the network part
    - Connected to some physical network
  - Every physical network has a router
    - Connected to at least one other physical network

Datagram forwarding:

- For e.g.: a TCP segment, UDP Datagram, or an ICMPv6 message

A simple internetwork:

- Forwarding table for router R2
- Network 1 (Frame Relay)
- Network 2 (Frame Relay)
- Network 3 (FDDI)
- Network 4 (point-to-point)
### Example forwarding table:

For Router R2

<table>
<thead>
<tr>
<th>Network Num</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R3</td>
</tr>
<tr>
<td>2</td>
<td>R1</td>
</tr>
</tbody>
</table>

### Error Reporting in IP communications

- IP drops datagrams when the going gets tough
  - But does not fail silently
- IP always configured with a companion protocol
  - Internet Control Message Protocol (ICMP)

### ICMP defines a collection of error messages

- When router/host is unable to process datagrams successfully
  - ICMP error message sent back to source
- Examples
  - Destination host is unreachable
  - Reassembly process failed
  - TTL reached 0
  - IP header checksum failed

### ICMP also defines some control messages

- Router sends control messages back to host
- Example: ICMP-Redirect tells that there is a better route to destination
- Network has two routers R1 and R2 and host uses R1 as default
- When R1 receives a datagram and it knows R2 is a better choice?
  - Send ICMP-Redirect to host
  - Host then uses R2 for future datagrams to that host

### User Datagram Protocol

- **Simplest** possible transport protocol
- Extends host-to-host into process-to-process communications
- No additional functionality to best-effort service provided by underlying network
- Adds demultiplexing
  - Allows applications on a host to share the service

---

**UDP SIMPLE DEMULTIPLEXER**
UDP identification of processes

- Processes indirectly identify each other
  - Abstract locator called port
- Source sends a message to a port
  - Destination receives messages from a port
- Process is identified by a port on a particular host

Format of a UDP header

```
0  16  31
SrcPort  DetPort
Length   Checksum
Data
```

A port is just an abstraction

- Typically implemented as a message queue
- When message arrives?
  - Protocol appends message to end of the queue
- UDP
  - If the queue is full, message is discarded
  - No flow-control mechanism

UDP message queue: The port abstraction

Some work that UDP does do besides demultiplexing: Checksumming

- UDP header
- Message body
- Pseudoheder: From the IP header
  - Protocol number
  - Source IP address
  - Destination IP address
- UDP length
  - Used twice

RELIABLE BYTE STREAM
TCP
Components of Reliable delivery

- **Acknowledgements**
  - Confirm receipt of data

- **Timeouts**
  - Retransmit if ACK not received within a specified time
  - Use of ACKs and timeouts to implement reliable delivery
  - Sometimes called ARQ (automatic repeat request)

Simplest ARQ is the stop-and-wait algorithm

- After transmitting one frame
  - Sender waits for ACK before transmitting the next frame
- If the ACK does not arrive after a period of time
  - Sender retransmits the original frame

Stop-and-wait (1/2)

<table>
<thead>
<tr>
<th>Time</th>
<th>SENDER</th>
<th>RECEIVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td>Frame</td>
<td></td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>Frame</td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td>Frame</td>
<td></td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>Frame</td>
<td></td>
</tr>
</tbody>
</table>

Stop-and-wait (2/2)

<table>
<thead>
<tr>
<th>Time</th>
<th>SENDER</th>
<th>RECEIVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td>Frame</td>
<td></td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>Frame</td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td>Frame</td>
<td></td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>Frame</td>
<td></td>
</tr>
</tbody>
</table>

Sliding window: Try to fill the network pipe

- DELAY \times BANDWIDTH product is 8 KB
- Data frames = 1KB
- Sender could transmit 9th frame
  - When ACK for the 1st frame arrives

Timeline for the sliding window

<table>
<thead>
<tr>
<th>Time</th>
<th>SENDER</th>
<th>RECEIVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transmission Control Protocol (TCP) [1/2]
- Reliable, in-order delivery of byte streams
- Full duplex protocol
  - Each connection supports a pair of byte streams
  - Flowing in different directions
- Includes flow control mechanism
  - Allows receiver to limit the data sender
  - Control how much data can be transmitted at a time

Transmission Control Protocol (TCP) [2/2]
- Includes multiplexing mechanism
  - Multiple apps on a given host
- Implements a congestion-control mechanism
  1. Throttle how fast TCP sends data
  2. Keep sender from overloading the network

Flow control and congestion control
- Flow control is an end-to-end issue
  - Don’t overrun capacity of receiver
- Congestion control is about hosts & networks interact
  - Don’t cause switches and links to be overloaded

TCP: Setup and Teardown
- Two sides of the connection agree to exchange data
  - Establish shared state
  - 3 packets exchanged (SYN, SYN-ACK, ACK)
- Connection teardown
  - Let each host know it is OK to free the shared state
  - 4 packets exchanged (FIN, ACK, FIN, ACK)

TCP Segments & how they come about
- TCP
  - Accepts data from a data stream
  - Breaks it up into chunks
  - Adds a TCP header ... creating a TCP segment
- Segment is then encapsulated in an IP datagram
- TCP packet is a term that you will often hear
  - Segment is more precise, packets are generally datagrams, frames are at the link layer
How TCP manages a byte stream

TCP Header Format

TCP Sliding Window

TCP Send Buffer
**Flow Control: Buffers are of finite size**

**MaxSendBuffer and MaxRcvBuffer**

- Receiver throttles sender
  - Advertises a window
  - No bigger than what it can buffer

\[
\text{LastByteRecvd} - \text{LastByteRead} \leq \text{MaxRcvBuffer}
\]

\[
\text{AdvertisedWindow} = \text{MaxRcvBuffer} - (\text{NextByteExpected} - 1) - \text{LastByteRead}
\]

**Flow Control: Buffers are of finite size**

**MaxSendBuffer and MaxRcvBuffer**

- On the sender side, TCP adheres to the advertised window from the receiver

\[
\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}
\]

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

**EffectiveWindow should be > 0** before source can send more data

---

**The advertised window may potentially shrink**

- If the process is reading data as fast as it arrives?
  - The advertised window stays open
    - i.e., \( \text{AdvertisedWindow} = \text{MaxRcvBuffer} \)

- If the receiving process falls behind?
  - Advertised window becomes smaller with every segment that arrives
  - Until it becomes 0

---

**The contents of this slide-set are based on the following references**

- Understanding the IPv6 Header: https://www.microsoftpressstore.com/articles/article.aspx?p=2225063&seqNum=4

---

CS455: Introduction to Distributed Systems [Spring 2018]
Dept. Of Computer Science, Colorado State University

**SLIDES CREATED BY: SHRIDEEP PALICKARA**