The Receiver’s Buffer
Small it may be
But throttle the mightiest sender
It can
Not just the how much
But also the when
Or if at all

Frequently asked questions from the previous class surveys
IP, TCP, etc: Who handles it? The OS? Hardware?
How do you deal with out of order packets?
Who reconstructs the fragmented IP packets?
Demux Fields
How do routers know where to send the packets to?

Topics covered in today’s lecture
- IP routing
- IPv6
- UDP
- TCP

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, OR (3) send packets no larger than the default MTU=1280
- As of 2014, IPv4 still carried >99% of worldwide Internet traffic

IPv6 Packet Header

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
<td>4</td>
<td>Version of IPv6</td>
</tr>
<tr>
<td>Traffic Class</td>
<td>4</td>
<td>8</td>
<td>Traffic class</td>
</tr>
<tr>
<td>Flow Label</td>
<td>12</td>
<td>16</td>
<td>Flow label</td>
</tr>
<tr>
<td>Payload Length</td>
<td>28</td>
<td>32</td>
<td>Payload length</td>
</tr>
<tr>
<td>Next Header</td>
<td>32</td>
<td>36</td>
<td>Next header</td>
</tr>
<tr>
<td>Hop Limit</td>
<td>36</td>
<td>40</td>
<td>Hop limit</td>
</tr>
<tr>
<td>Source Addr</td>
<td>40</td>
<td>56</td>
<td>Source address (16 bytes)</td>
</tr>
<tr>
<td>Destination Addr</td>
<td>56</td>
<td>72</td>
<td>Destination address (16 bytes)</td>
</tr>
</tbody>
</table>

IPv6 Packet Header is fixed at 40 bytes... So there is no Header Length
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**: Size of payload including extension headers

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)

Payload
```

PDU typically contains an upper layer protocol header and its payload.

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

---

**Extension Header**

- If the Next Header field is non-zero
  - It defines an extension header
- Current extension header types
  - Information for routers, route definition, fragment handling, authentication, encryption, etc.
- Each extension header has a specific size and defined format
- If an extension header is present?
  - Follows the basic header and precedes the payload AND
  - Includes a Next Header

---

**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.

---

**UDP Simple Demultiplexer**
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 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**

<table>
<thead>
<tr>
<th>0</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>DstPort</td>
<td>Length</td>
</tr>
<tr>
<td>Checksum</td>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>

**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
  - **Pseudoheader:** From the IP header
    - Protocol number
    - Source IP address
    - Destination IP address
  - **UDP** length
  - **Used twice**

Verify if message is delivered between the correct endpoints
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
    - Sometime 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

Sliding window: Try to fill the network pipe

- DELAY x BANDWIDTH product is 8 KB
- Data frames = 1KB
- Sender could transmit 9th frame
  - When ACK for the 1st frame arrives
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 applications 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 how 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 a 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 Sliding Window
- Guarantees reliable delivery of data
- Data is delivered in order
- Enforces flow control between the sender and receiver

TCP Header Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>16</td>
</tr>
<tr>
<td>DestPort</td>
<td>16</td>
</tr>
<tr>
<td>SequenceNum</td>
<td>32</td>
</tr>
<tr>
<td>Acknowledgement</td>
<td>32</td>
</tr>
<tr>
<td>Window</td>
<td>32</td>
</tr>
<tr>
<td>Checksum</td>
<td>32</td>
</tr>
<tr>
<td>UrgPtr</td>
<td>16</td>
</tr>
<tr>
<td>Options</td>
<td>variable</td>
</tr>
<tr>
<td>Data</td>
<td>0</td>
</tr>
</tbody>
</table>

Relationship between SequenceNum, Acknowledgement and AdvertisedWindow

TCP Sliding Window
- Sender has a limit on unacknowledged data
  - Limited to no more than AdvertisedWindow bytes of unacknowledged data
- Receiver selects AdvertisedWindow
  - Based on memory set aside for connection’s buffer space
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
  
  \[
  \text{LastByteRcvd} - \text{LastByteRead} \leq \text{MaxRcvBuffer}
  \]

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

  - Space Utilized in the receiver's buffer

TCP Receive Buffer

**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

**Flow Control: Buffers are of finite size MaxSendBuffer and MaxRcvBuffer**

- On the sender size, 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
Protecting against wraparound:
32-bit sequence space
- TCP assumes each segment has a max lifetime
  - Maximum segment lifetime (MSL)
  - Currently this is 120 seconds
- Sequence number used on a connection might wrap-around
  - Within the MSL

Time until 32-bit sequence number wraps around
<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Time until wraparound</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>6.4 hours</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>57 minutes</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>1.5 minutes</td>
</tr>
<tr>
<td>FDDI (100 mbps)</td>
<td>6 minutes</td>
</tr>
<tr>
<td>STS-3 (155 Mbps)</td>
<td>4 minutes</td>
</tr>
<tr>
<td>STS-12 (622 Mbps)</td>
<td>55 seconds</td>
</tr>
<tr>
<td>STS-24 (1.2 Gbps)</td>
<td>28 seconds</td>
</tr>
</tbody>
</table>

Keeping the pipe full
- **AdvertisedWindow** field (16-bits) must be big enough
  - To allow sender to keep the pipe full
  - 16 bit allows a max window size of 64 KB \(2^{16}\)
- If receiver has unlimited buffer space?
  - **AdvertisedWindow** dictated by DELAY X BANDWIDTH product

Required Window Size for 100 ms delay
<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Delay x 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>STS-3 (155 Mbps)</td>
<td>1.8 MB</td>
</tr>
<tr>
<td>STS-12 (622 Mbps)</td>
<td>7.4 MB</td>
</tr>
<tr>
<td>STS-24 (1.2 Gbps)</td>
<td>14.8 MB</td>
</tr>
</tbody>
</table>

TCP extensions: Use 32-bit timestamp to extend sequence number space
- Distinguish between different incarnations of the same sequence number
- Timestamp not treated as part of sequence number
  - For ordering etc.
  - Just protects against wraparound

TCP Extension: Allow TCP to advertise larger window
- Fill larger DELAY X BANDWIDTH pipes
- Include option defining scaling factor
- Option allows TCP endpoints to agree that **AdvertisedWindow** counts larger chunks

---

SLIDES CREATED BY: SHRIDEEP PALICKARA
A caveat regarding Options

- You cannot solve all problems with Options
- TCP Header has room for only **44 bytes of options**
  - HdrLen is 4 bits long, so header length cannot exceed $16 \times 32$-bit $= 64$ bytes
  - Adding a TCP option that extends the space available for options?

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