CS 455: Introduction To Distributed Systems [Networking]

Shrideep Pallickara
Computer Science
Colorado State University

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 work with IP/TCP/UDP when you can write directly to network?
- 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 **Ident** field
  - To enable fragment reassembly
  - Chosen by the source host
- If all fragments do not arrive at receiving host?
  1. Receiver ***gives up*** reassembly [reassembly timeout: 15 seconds RFC0791]
  2. ***Discards*** fragments that did arrive
- IP **does not attempt** to recover from missing fragments

IPv4 Packet header

0	4	8	16	19 – 31
Version	HLen	TOS	Length	
Ident			Flags	Offset
TTL		Protocol	Checksum	
SourceAddr				
DestinationAddr				
Options (variable)				Pad (variable)
Data				

Header fields used in IP fragmentation:
Fragmentation occurs at 8-byte boundaries

Start of header				
Ident = x	0	DF	0	Offset = 0
Rest of header				
1400 data bytes				

Unfragmented packet

Start of header				
Ident = x			1	Offset = 0
Rest of header				
512 data bytes				

Fragmented packet

Header fields used in IP fragmentation:
Fragmentation occurs at 8-byte boundaries

Start of header				
Ident = x			1	Offset = 64
Rest of header				
512 data bytes				

Fragmented packet

Start of header				
Ident = x			0	Offset = 128
Rest of header				
376 data bytes				

Fragmented packet

IPv6 (and comparing with IPv4)

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

0 4 8 12 16 19 31

Version	Traffic Class	Flow Label	
Payload Length		Next Header	Hop Limit
SourceAddr [16 bytes]			
DestinationAddr [16 bytes]			

IPv6 Packet Header is fixed at 40 bytes … So there is no Header Length

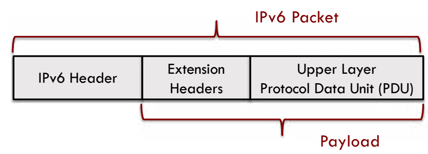
IPv6 Packet Header: Some more details [1/2]

- **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 (**16 bits**): 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

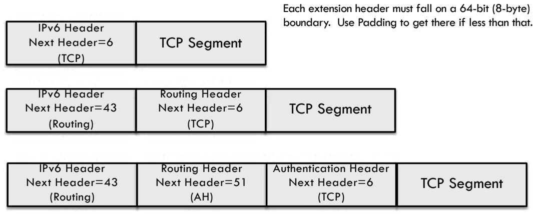
Structure of the IPv6 Packet



PDU typically contains an upper layer protocol header and its payload.
For e.g.: a TCP segment, UDP Datagram, or an ICMPv6 message

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.



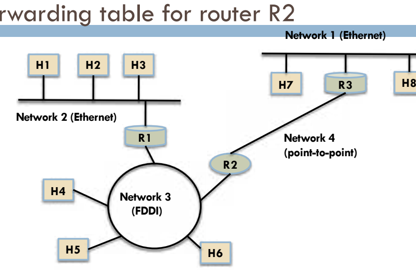
Fragmentation Header: 44

Datagram forwarding

Datagram forwarding in IP: Datagrams contains IP address of destination

- Network part uniquely identifies a single physical network
- Hosts/routers that share the network part
  - Connected to *same* physical network
- Every physical network has a router
  - Connected to at least *one other* physical network

A simple internetwork: Forwarding table for router R2



Example forwarding table: For Router R2

Network Num	Next Hop
1	R3
2	R1

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

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* indentify 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

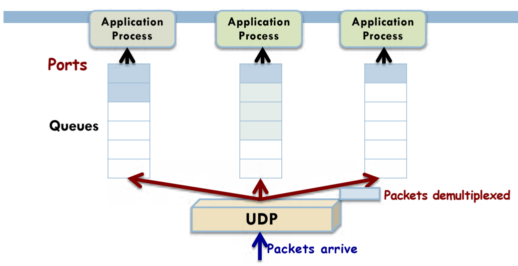
0 16 31

SrcPort	DstPort
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
- ***Psuedoheader***: From the IP header
  - Protocol number
  - Source IP address
  - Destination IP address

  } Verify if message is delivered between the correct endpoints
- UDP length
  - Used *twice*

RELIABLE BYTE STREAM
TCP

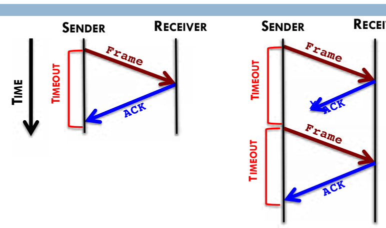
Components of Reliable delivery

- **Acknowledgements**
  - Confirm receipt of data
- **Timeouts**
  - *Retransmit* if ACK not received within a specified time
- Use of ACKs and timeouts to implement reliable delivery
  - Sometime called ARQ (**a**utomatic **r**epeat re**q**uest)

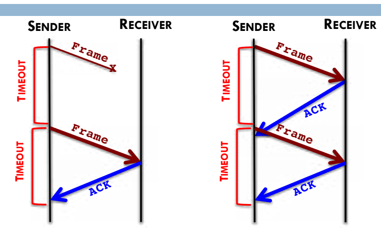
Simplest ARQ is the stop-and-wait algorithm

- 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)



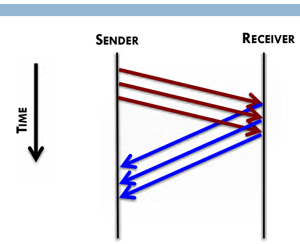
Stop-and-wait (2/2)



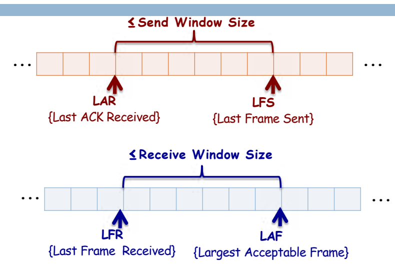
Sliding window: Try to fill the network pipe

- Delay x Bandwidth product is 8 KB
- Data frames = 1KB
- Sender could transmit 9th frame
  - When ACK for the 1st frame arrives

Timeline for the sliding window



Sliding Window on Sender/Receiver



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

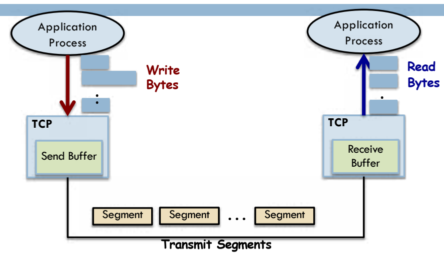
TCP: Setup and Teardown

- 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)

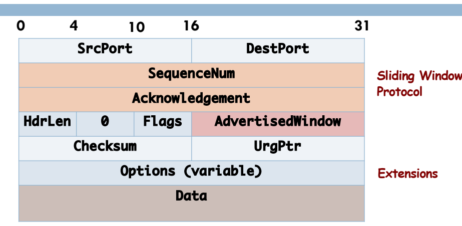
TCP Segments & how they come about

- 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 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 Header Format



SourceAddr and DestinationAddr from IP

Relationship between `SequenceNum`, `Acknowledgement` and `AdvertisedWindow`

Each byte of data has a sequence number
SequenceNum contains sequence number for first byte of data in segment

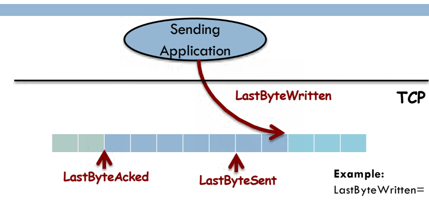
TCP Sliding Window [1/2]

- Guarantees **reliable** delivery of data
- Data is delivered in **order**
- Enforces **flow control** between the sender and receiver

TCP Sliding Window [2/2]

- 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

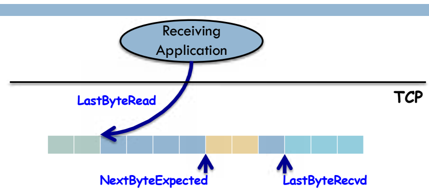
TCP Send Buffer



LastByteAcked ≤ LastByteSent
LastByteSent ≤ LastByteWritten

Example:
LastByteWritten= 3000
LastByteSent=2800
LastByteAcked:= 2400
How many unacknowledged bytes? (3000-2400)= 600

TCP Receive Buffer



LastByteRead ≤ NextByteExpected

NextByteExpected ≤ LastByteRecvd + 1

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

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

LastByteRcvd – LastByteRead ≤ MaxRcvBuffer

AdvertisedWindow =

$$\text{MaxRcvBuffer} - \underbrace{(\,(\text{NextByteExpected} - 1) - \text{LastByteRead}\,)}_{\text{Space Utilized in the receiver's buffer}}$$

The advertised window may potentially shrink

- If the process is reading data as fast as it arrives?
  - The advertised window *stays open*
    - i.e. `AdvertisedWindow = 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

LastByteSent – LastByteAcked ≤ AdvertisedWindow

EffectiveWindow =
AdvertisedWindow – (LastByteSent – LastByteAcked)

EffectiveWindow should be > 0 before source can send more data

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

- *Computer Networks: A Systems Approach. Larry Peterson and Bruce Davie. 4th edition. Morgan Kaufmann. ISBN: 978-0-12-370548-8.* Chapters [4, 5]
- https://en.wikipedia.org/wiki/IPv6
- Understanding the IPv6 Header: https://www.microsoftpressstore.com/articles/article.aspx?p=2225063&seqNum=4