15-441 Computer Networking

Lecture 7 – Transport Protocols

Outline

- Akamai
- Transport introduction
- Error recovery
- TCP flow control

Simple Hashing

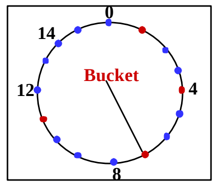
- Given document XYZ, we need to choose a server to use
- Suppose we use modulo
- Number servers from 1…n
  - Place document XYZ on server (XYZ mod n)
  - What happens when a servers fails? n → n-1
    - Same if different people have different measures of n
  - Why might this be bad?

Consistent Hash

- “view” = subset of all hash buckets that are visible
- Desired features
  - Balanced – in any one view, load is equal across buckets
  - Smoothness – little impact on hash bucket contents when buckets are added/removed
  - Spread – small set of hash buckets that may hold an object regardless of views
  - Load – across all views # of objects assigned to hash bucket is small

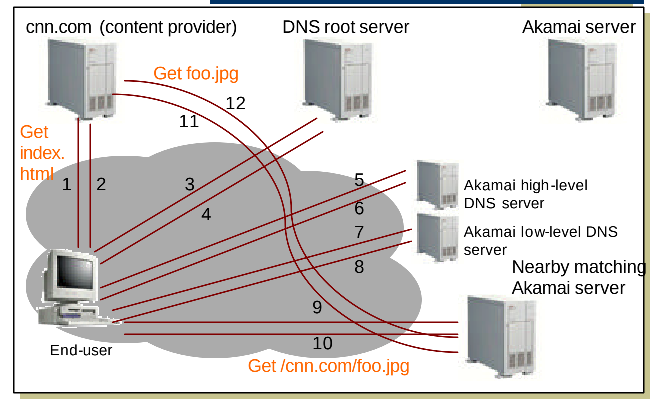
Consistent Hash – Example

- Construction
  - Assign each of C hash buckets to random points on mod 2^n circle, where, hash key size = *n.*
  - Map object to random position on circle
  - Hash of object = closest clockwise bucket

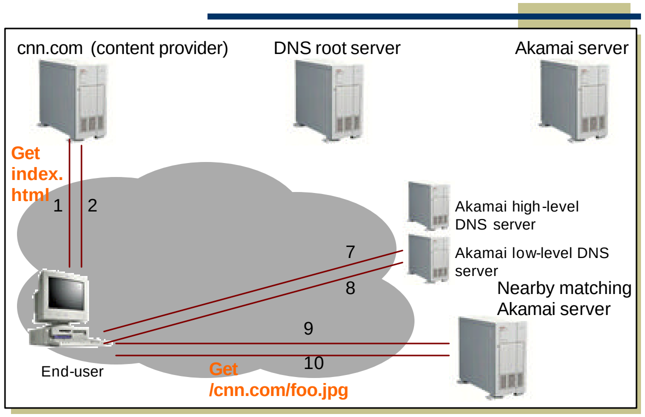


- Smoothness → addition of bucket does not cause movement between existing buckets
- Spread & Load → small set of buckets that lie near object
- Balance → no bucket is responsible for large number of objects

How Akamai Works



Akamai – Subsequent Requests



HTTP (Summary)

- Simple text-based file exchange protocol
  - Support for status/error responses, authentication, client-side state maintenance, cache maintenance
- Workloads
  - Typical documents structure, popularity
  - Server workload
- Interactions with TCP
  - Connection setup, reliability, state maintenance
  - Persistent connections
- How to improve performance
  - Persistent connections
  - Caching
  - Replication

Outline

- Akamai
- Transport introduction
- Error recovery
- TCP flow control

Functionality Split

- Network provides best-effort delivery
- End-systems implement many functions
  - Reliability
  - In-order delivery
  - Demultiplexing
  - Message boundaries
  - Connection abstraction
  - Congestion control
  - …

Transport Protocols

- UDP provides just integrity and demux
- TCP adds…
  - Connection-oriented
  - Reliable
  - Ordered
  - Point-to-point
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled

UDP: User Datagram Protocol [RFC 768]

- "No frills," "bare bones" Internet transport protocol
- "Best effort" service, UDP segments may be:
  - Lost
  - Delivered out of order to app
- *Connectionless:*
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others

Why is there a UDP?

- No connection establishment (which can add delay)
- Simple: no connection state at sender, receiver
- Small header
- No congestion control: UDP can blast away as fast as desired

UDP, cont.

- Often used for streaming multimedia apps
  - Loss tolerant
  - Rate sensitive
- Other UDP uses (why?):
  - DNS, SNMP
- Reliable transfer over UDP
  - Must be at application layer
  - Application-specific error recovery

32 bits

Source port #	Dest port #
Length	Checksum
Application data (message)	

Length, in bytes of UDP segment, including header

UDP segment format

UDP Checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment – optional use!

Sender:

- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1's complement sum) of segment contents
- Sender puts checksum value into UDP checksum field

Receiver:

- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonethless?*

High-Level TCP Characteristics

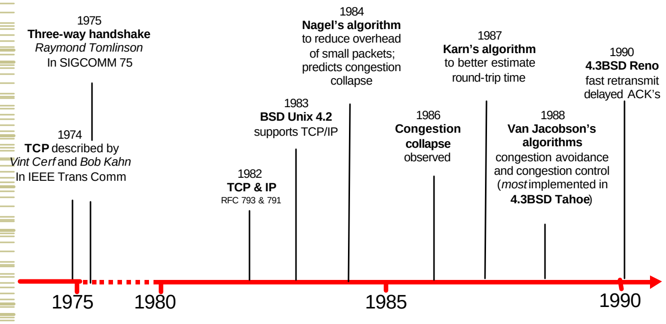
- Protocol implemented entirely at the ends
  - Fate sharing
- Protocol has evolved over time and will continue to do so
  - Nearly impossible to change the header
  - Uses options to add information to the header
  - Change processing at endpoints
  - Backward compatibility is what makes it TCP

TCP Header

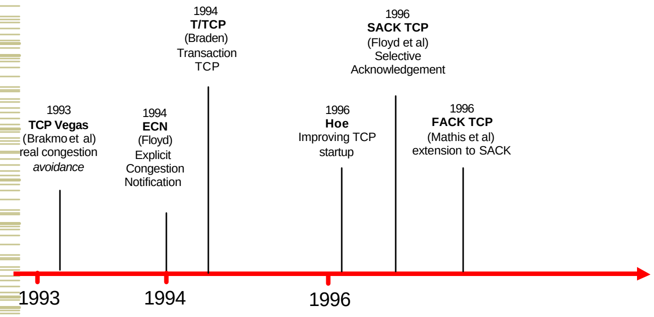
Flags: SYN
FIN
RESET
PUSH
URG
ACK

Source port		Destination port	
Sequence number			
Acknowledgement			
HdrLen	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			
Data			

Evolution of TCP



TCP Through the 1990s

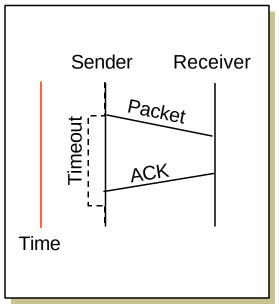


Outline

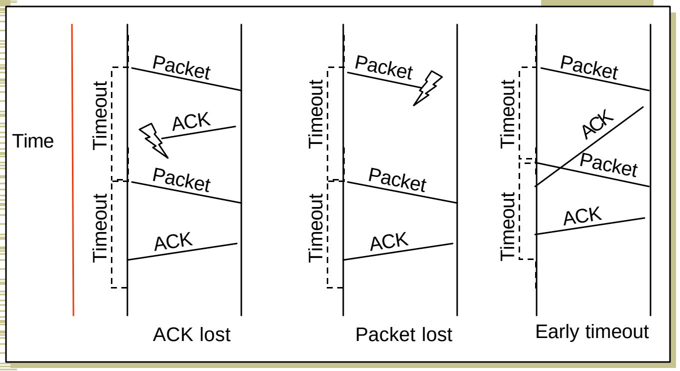
- Akamai
- Transport introduction
- Error recovery
- TCP flow control

Stop and Wait

- ARQ
  - Receiver sends acknowledgement (ACK) when it receives packet
  - Sender waits for ACK and timeouts if it does not arrive within some time period
- Simplest ARQ protocol
- Send a packet, stop and wait until ACK arrives



Recovering from Error

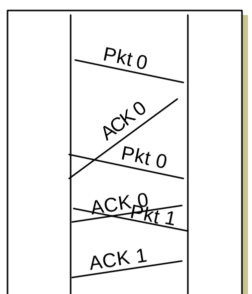


Problems with Stop and Wait

- How to recognize a duplicate
- Performance
  - Can only send one packet per round trip

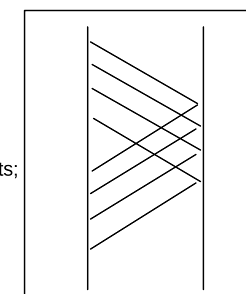
How to Recognize Resends?

- Use sequence numbers
  - both packets and acks
- Sequence # in packet is finite -- how big should it be?
  - For stop and wait?
- One bit – won't send seq #1 until received ACK for seq #0



How to Keep the Pipe Full?

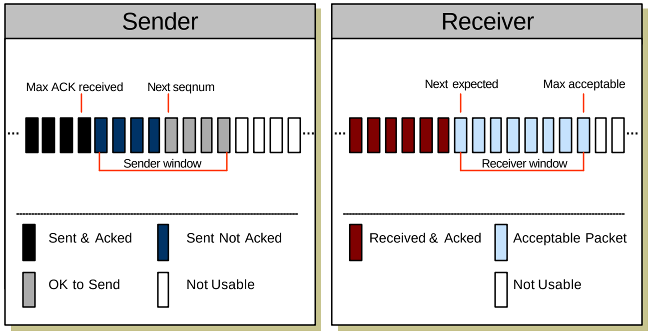
- Send multiple packets without waiting for first to be acked
  - Number of pkts in flight = window
- Reliable, unordered delivery
  - Several parallel stop & waits
  - Send new packet after each ack
  - Sender keeps list of unack'ed packets; resends after timeout
  - Receiver same as stop & wait
- How large a window is needed?
  - Suppose 10Mbps link, 4ms delay, 500byte pkts
    - 1? 10? 20?
  - Round trip delay * bandwidth = capacity of pipe



Sliding Window

- Reliable, ordered delivery
- Receiver has to hold onto a packet until all prior packets have arrived
  - Why might this be difficult for just parallel stop & wait?
  - Sender must prevent buffer overflow at receiver
- Circular buffer at sender and receiver
  - Packets in transit ≤ buffer size
  - Advance when sender and receiver agree packets at beginning have been received

Sender/Receiver State



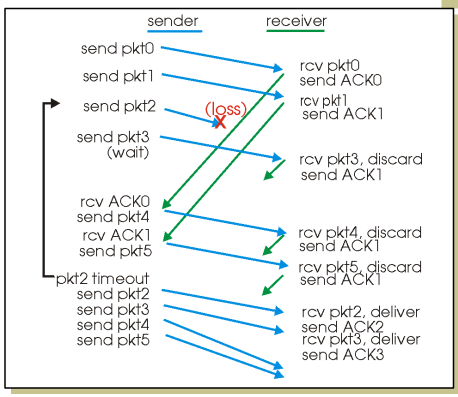
Window Sliding – Common Case

- On reception of new ACK (i.e. ACK for something that was not acked earlier)
  - Increase sequence of max ACK received
  - Send next packet
- On reception of new in-order data packet (next expected)
  - Hand packet to application
  - Send cumulative ACK – acknowledges reception of all packets up to sequence number
  - Increase sequence of max acceptable packet

Loss Recovery

- On reception of out-of-order packet
  - Send nothing (wait for source to timeout)
  - Cumulative ACK (helps source identify loss)
- Timeout (Go-Back-N recovery)
  - Set timer upon transmission of packet
  - Retransmit all unacknowledged packets
- Performance during loss recovery
  - No longer have an entire window in transit
  - Can have much more clever loss recovery

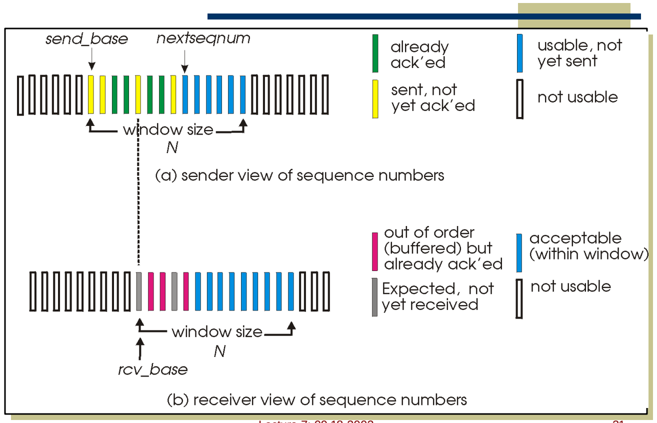
Go-Back-N in Action



Selective Repeat

- Receiver *individually* acknowledges all correctly received pkts
  - Buffers packets, as needed, for eventual in-order delivery to upper layer
- Sender only resends packets for which ACK not received
  - Sender timer for each unACKed packet
- Sender window
  - N consecutive seq #'s
  - Again limits seq #s of sent, unACKed packets

Selective Repeat: Sender, Receiver Windows



Sequence Numbers

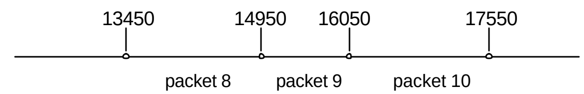
- How large do sequence numbers need to be?
  - Must be able to detect wrap-around
  - Depends on sender/receiver window size
- E.g.
  - Max seq = 7, send win=recv win=7
  - If pkts 0..6 are sent succesfully and all acks lost
    - Receiver expects 7,0..5, sender retransmits old 0..6!!!
- Max sequence must be ≥ send window + recv window

Outline

- Akamai
- Transport introduction
- Error recovery
- TCP flow control

Sequence Number Space

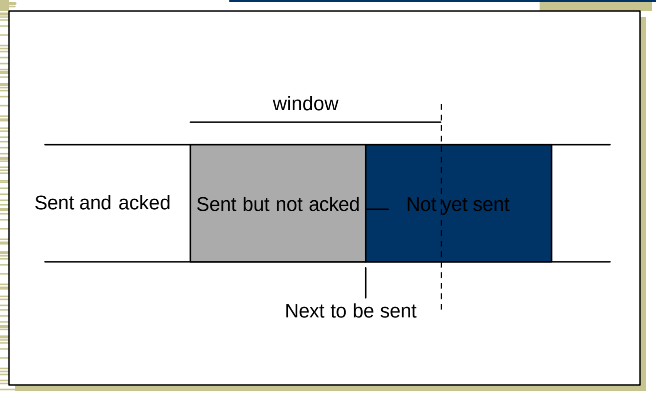
- Each byte in byte stream is numbered.
  - 32 bit value
  - Wraps around
  - Initial values selected at start up time
- TCP breaks up the byte stream in packets.
  - Packet size is limited to the Maximum Segment Size
- Each packet has a sequence number.
  - Indicates where it fits in the byte stream



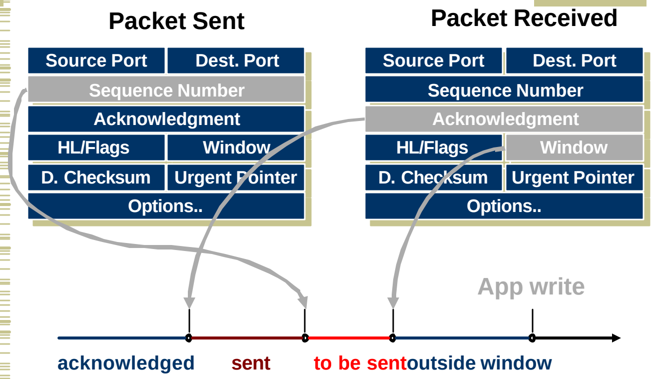
TCP Flow Control

- TCP is a sliding window protocol
  - For window size *n*, can send up to *n* bytes without receiving an acknowledgement
  - When the data is acknowledged then the window slides forward
- Each packet advertises a window size
  - Indicates number of bytes the receiver has space for
- Original TCP always sent entire window
  - Congestion control now limits this

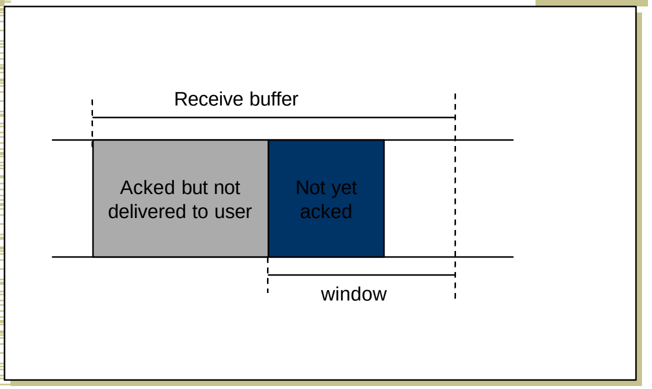
Window Flow Control: Send Side



Window Flow Control: Send Side



Window Flow Control: Receive Side



TCP Persist

- What happens if window is 0?
  - Receiver updates window when application reads data
  - What if this update is lost?
- TCP Persist state
  - Sender periodically sends 1 byte packets
  - Receiver responds with ACK even if it can't store the packet

Performance Considerations

- The window size can be controlled by receiving application
  - Can change the socket buffer size from a default (e.g. 8Kbytes) to a maximum value (e.g. 64 Kbytes)
- The window size field in the TCP header limits the window that the receiver can advertise
  - 16 bits → 64 KBytes
  - 10 msec RTT → 51 Mbit/second
  - 100 msec RTT → 5 Mbit/second

Next Lecture

- TCP connection setup
- TCP reliability
- Congestion control