



EEC-484/584 Computer Networks

Lecture 18

Wenbing Zhao

wenbing@ieee.org

(Lecture notes are based on materials supplied by
Dr. Louise Moser at UCSB and Prentice-Hall)



Outline

- The Transport layer
 - Performance issues
- Review for midterm #2
- Reminder: Midterm #2, Nov 9 Wednesday
 - Chapters 5-6
 - Closed book, closed notes

8 November 2005

EEC484/584

Wenbing Zhao



Performance Issues

- Performance Problems in Computer Networks
- Network Performance Measurement
- System Design for Better Performance
- Fast TPDU Processing
- Protocols for Gigabit Networks

8 November 2005

EEC484/584

Wenbing Zhao



Performance Problems in Computer Networks

- Congestion - caused by temporary resource overloads
 - If more traffic suddenly arrives at a router than the router can handle, congestion will build up and performance will suffer
- Structural resource imbalance
 - For example, if a gigabit communication line is attached to a low-end PC, the poor CPU will not be able to process the incoming packets fast enough and some will be lost

8 November 2005

EEC484/584

Wenbing Zhao

Performance Problems in Computer Networks

5

- Broadcast storm - overloads can also be synchronously triggered
 - For example, if a TPDU contains a bad parameter (e.g., the port for which it is destined), in many cases the receiver will thoughtfully send back an error notification (recall the ICMP packets)
- Lack of system tuning
 - For example, if a machine has plenty of CPU power and memory but not enough of the memory has been allocated for buffer space, overruns will occur and TPDU's will be lost
- Networks with high bandwidth-delay product (bandwidth X roundtrip time)
 - E.g., Gigabit Ethernet
 - Receiver's window should be at least as large as bandwidth-delay

8 November 2005

EEC484/584

Wenbing Zhao

Performance Problems in Computer Networks



At t = 0

After 500 μsec



After 20 msec

after 40 msec

The state of transmitting one megabit from San Diego to Boston

Network Performance Measurement

7

- The basic loop for improving network performance
 - Measure relevant network parameters, performance
 - Try to understand what is going on
 - Change one parameter

8 November 2005

EEC484/584

Wenbing Zhao

Network Performance Measurement

8

- **Where** measurements take place - both physically and in the protocol stack
- What to measure ?
 - **How long** activity takes - start a timer when beginning some activity and stop the timer at the end of the activity
 - E.g., knowing how long it takes for a TPDU to be acknowledged
 - **How often** some event has happened - use counters
 - E.g., number of lost TPDU's
 - **How many / how much**
 - E.g., the number of bytes processed in a certain time interval

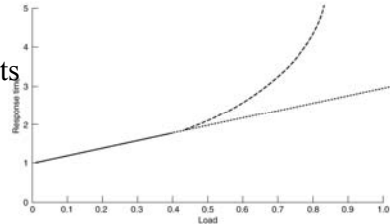
8 November 2005

EEC484/584

Wenbing Zhao

Network Performance Measurement

- Avoid pitfalls during performance measurements
 - Make sure that the sample size is large enough
 - Make sure that the samples are representative
 - Be careful when using a coarse-grained clock
 - Be sure that nothing unexpected is going on during your tests
 - Caching can wreak havoc with measurements
 - Understand what you are measuring
 - Be careful about extrapolating the results



System Design for Better Performance

10

Rules of thumb

- CPU speed is more important than network speed
- Reduce packet count to reduce software overhead
- Minimize context switches
- Minimize copying
- You can buy more bandwidth but not lower delay
- Avoiding congestion is better than recovering from it
- Avoid timeouts

8 November 2005

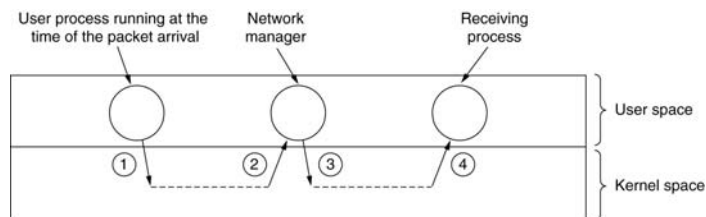
EEC484/584

Wenbing Zhao

System Design for Better Performance

11

- Four context switches to handle one packet with a user-space network manager



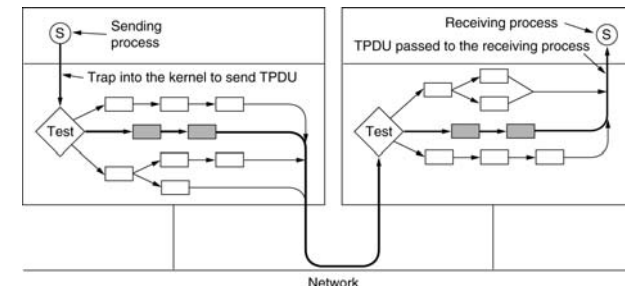
8 November 2005

EEC484/584

Wenbing Zhao

Fast TPDU Processing

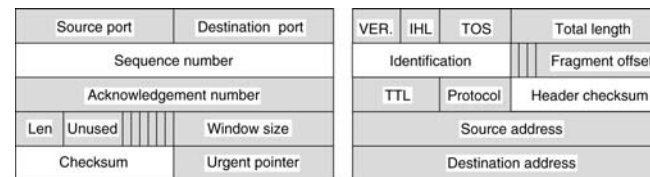
- TPDU processing overhead has two components: overhead per TPDU and overhead per byte
- The key to fast TPDU processing is to separate out the normal case (one-way data transfer) and handle it specially



Fast TPDU Processing - Sending Side

- Headers of consecutive data TPDU are almost the same. To take advantage of this fact, a prototype header is stored within the transport entity
- At the start of the fast path, it is copied to a scratch buffer, word by word
- Those fields that change from TPDU to TPDU are then overwritten in the buffer
- A pointer to the full TPDU header plus a pointer to the user data are then passed to the network layer
- The network layer could use similar approach for fast processing and minimize copying

Fast TPDU Processing - Sending Side

(a)
TCP header(b)
IP header

In both cases, the shaded fields are taken from the prototype without change

Fast TPDU Processing - Receiving Side

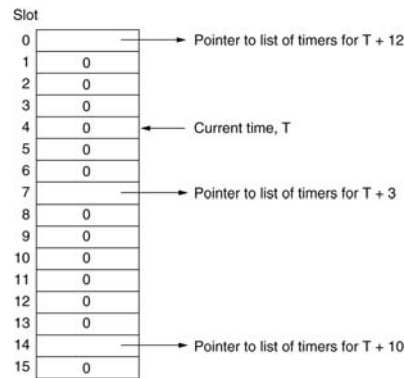
- Once a TPDU is received, the connection record is located from the hash table
- The TPDU is then checked to see if it is a normal one
 - State is ESTABLISHED and neither side is closing, TPDU is expected
- The fast path then updates the connection record and copies the data to the user
- The general scheme of first making a quick check to see if the header is what is expected and then having a special procedure handle that case is called **header prediction**
- Many TCP implementations use it

Fast TPDU Processing

- Two other areas where major performance gains are possible
 - Buffer management - avoiding unnecessary copying
 - Timer management - nearly all timers set do not expire
 - They are set to guard against TPDU loss, but most TPDU arrive correctly and their acknowledgements also arrive correctly.
 - It is important to optimize timer management for the case of timers rarely expiring.
- Timer management schemes:
 - A linked list of timer events sorted by expiration time
 - Timing wheel

Fast TPDU Processing

- **Timing wheel** - An array for timer management. A more efficient approach if maximum timer interval is bounded and known in advance
 - Each slot corresponds to one clock tick
 - To add a timer, an entry is made into an appropriate slot
 - To cancel a timer, the corresponding slot is located and the entry is removed
 - When the clock ticks, the current time pointer is advanced by one slot (circularly)



Protocols for Gigabit Networks

18

- **Problems of old network protocols on Gigabit networks**
 - Sequence number recycling
 - The assumption that the time to use up the entire sequence space would greatly exceed the maximum packet lifetime no longer hold true
 - Many protocols use 32-bit sequence numbers. It take as little as 34 sec to cycle through, well under the 120 sec maximum packet lifetime on the Internet
 - Communication speeds have improved much faster than computing speeds
 - Fast network traffic can easily saturate CPU

8 November 2005

EEC484/584

Wenbing Zhao

Protocols for Gigabit Networks

19

- **Problems of old network protocols on Gigabit networks (continued)**
 - go back n protocol performs poorly on lines with a large bandwidth-delay product
 - Gigabit lines are fundamentally different from megabit lines in that long gigabit lines are delay limited rather than bandwidth limited
 - For many gigabit applications, such as multimedia, the variance in the packet arrival times is as important as the mean delay itself

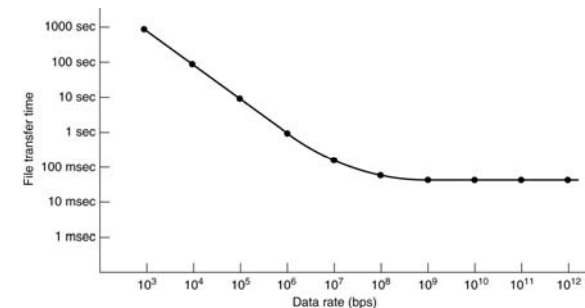
8 November 2005

EEC484/584

Wenbing Zhao

Protocols for Gigabit Networks

- **Time to transfer and acknowledge a 1-megabit file over a 4000-km line**
 - Latency is determined by the speed of light for high enough bandwidth



Protocols for Gigabit Networks: Design Principle

21

- Design for speed, not for bandwidth optimization
- Feedback issues
 - Use rate-based protocol in stead of sliding window protocol
 - Resource reservation based protocol in stead of slow start algorithm
- Header should contain as few fields as possible, and each field should be large enough

8 November 2005

EEC484/584

Wenbing Zhao

Protocols for Gigabit Networks: Design Principle

22

- The header and data should be separately checksummed
 - First, to make it possible to checksum the header but not the data
 - Second, to verify that the header is correct before copying the data into user space
 - It is desirable to do the data checksum at the time the data are copied to user space

8 November 2005

EEC484/584

Wenbing Zhao

Protocols for Gigabit Networks: Design Principle

23

- The maximum data size should be large, to permit efficient operation even in the face of long delays
 - 1500 bytes is too small
- Send a normal amount of data along with the connection request
- On protocol software
 - concentrating on the successful case
 - Minimize copying

8 November 2005

EEC484/584

Wenbing Zhao

Review for Midterm #2

24

- Chapter 5: Network layer
- Chapter 6: Transport layer

8 November 2005

EEC484/584

Wenbing Zhao

Network Layer

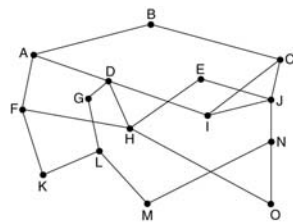
- Main concern: end-to-end transmission
 - Perhaps over many hops at intermediate nodes
- Network Layer Design Issues
 - Services Provided to the Transport Layer
 - Routing
 - Congestion control
 - Internetworking – connection of multiple networks
 - Goals – services should
 - Be independent of subnet technologies
 - Shield transport layer from number, type, topology of subnets
 - Uniform network addresses across LAN/WAN

Routing Algorithms

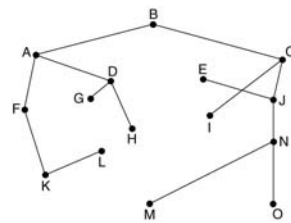
- Sink Tree
- Shortest Path Routing
- Flooding
- Distance Vector Routing
- Link State Routing
- Broadcast Routing

Sink Tree

- **Sink tree** – the set of optimal routes from all sources to a given destination form a tree rooted at the destination



A subnet



A sink tree for router B

Shortest Path Routing

- Shortest in what sense?
 - Number of hops, geographical distance, least queueing and transmission delay
- Dijkstra's Algorithm
 - Each node labeled with distance from source node along best known path
 - Initially, no paths known all nodes labeled with infinity
 - As algorithm proceeds, labels may change reflecting shortest path
 - Label may be tentative or permanent, initially, all tentative
 - When label represents shortest path from source to node, label becomes permanent

Flooding

- Idea: every incoming packet is sent out on every outgoing line except the one it arrived on
- **Adv:** flooding always chooses shortest path, no other algorithm has shorter delay
- **Disadv:** generates lots of duplicates
- Methods to damper this effect
 - Header of each packet has hop counter that is decremented at each hop; packet discarded when counter reaches 0
 - Source router puts sequence number in each packet it receives from its hosts
 - Each router has list for each source router telling which sequence numbers originating at that source have already been seen
 - Each list augmented with counter k, meaning all sequence numbers up through k have been seen
 - If packet is duplicate, router discards it

Distance Vector Routing

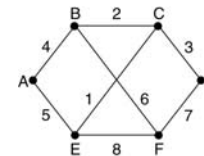
- Also called Bellman-Ford or Ford-Fulkerson. Used in ARPANET
- Idea: each router maintains a table (a vector), giving best known distance to each destination and which line to use to get there
 - Table is updated by exchanging info with neighbors
 - Table contains one entry for each router in network with
 - Preferred outgoing line to that destination
 - Estimate of time or distance to that destination
 - Once every T msec, router sends to each neighbor a list of estimated delays to each destination and receives same from those neighbors

Link State Routing – Used in Internet

- Idea: Each router must do the following:
 - Discover its neighbors, learn their network address
 - Measure the delay or cost to each of its neighbors
 - Construct a packet telling all it has just learned
 - Send this packet to all other routers
 - Compute the shortest path to every other router

Building Link State Packets

- Packets contain identity of sender, sequence number, age, list of **neighbors** and delay to that neighbor
- When are link state packets constructed?
 - Periodically at regular intervals
 - When link or nodes goes down or comes back



A subnet

Link		State		Packets	
A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

The link state packets for this subnet

Possible Strategies for Broadcast Routing

- Source sends separate copy of packet to all destinations
- Flooding
- Multi-destination routing
- **Spanning tree** - the sink tree (no loops) rooted at source that includes all routers. A router copies an incoming broadcast packet onto all the spanning tree lines except the one it arrived on
- **Reverse path forwarding**: if router gets packet on optimal (reverse) path to root, then it forwards the packet

Warning Bit

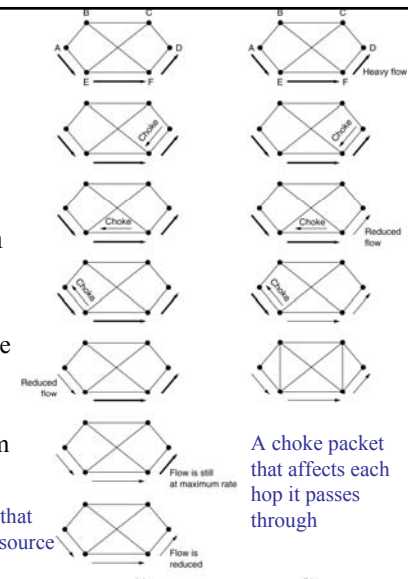
- Signal the warning state by setting a special bit in the packet's header
 - When packet arrived at its destination, transport entity copied the bit into the next ack sent back to the source
 - The source then cut back on traffic
 - Every router along the path could set the warning bit, traffic increases only when no router is in trouble

Choke Packets

- Tell the source directly to reduce traffic
 - Each router monitors utilization of each of its output lines
 - When utilization becomes greater than threshold, output line enters warning state
 - For each newly arriving packet, check if output line in warning state
 - If so, router sends choke packet back to source giving it the destination found in the packet
 - Packet is tagged so does not generate any more choke packets and forwarded as usual
 - When source receives choke packet, it must reduce traffic to specific destination

Hop-by-Hop Choke Packets

- The router that receives a choke packet must reduce the flow to its downstream router
 - This is achieved by allocating more buffer to the incoming flow
 - The router also passes the choke packet to its upstream router



A choke packet that affects only the source

A choke packet that affects each hop it passes through

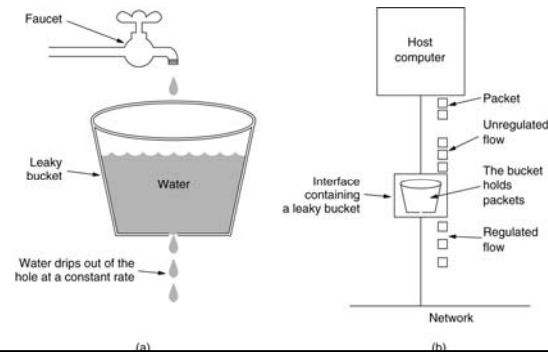
Load Shedding

■ RED - Random Early Detection

- Having routers drop packets before situation becomes hopeless
- Routers maintain a running average of their queue lengths
- When average queue length on some line exceeds a threshold, the line is said to be congested and action is taken
- Just discard selected packet and not report it.
- Sender respond to lost packets by slowing down transmission rate
- Appropriate for wired networks

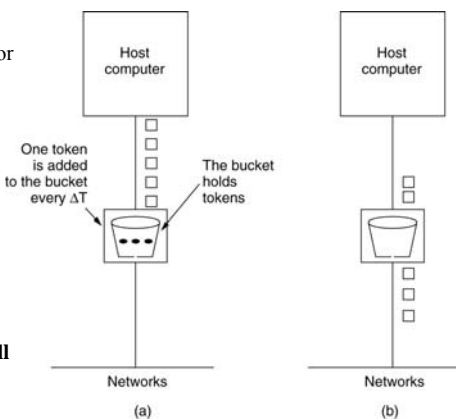
The Leaky Bucket Algorithm

- No matter what rate water enters the bucket, water flows out of the bucket at constant rate ρ (provided bucket is not empty)
 - When bucket is full, water slops over sides and is lost
 - Analogous to finite queue in network switch



The Token Bucket Algorithm

- The bucket hosts tokens in stead of data packets
- A token represents a packet or k bytes
- Tokens are added into the bucket with a constant rate
- Bucket has certain capacity. If bucket is full, new tokens are thrown away
- A data packet can be transmitted only if enough tokens present in bucket
- **Token bucket algorithm allows some burstiness because a full bucket of tokens saved can be used all at once**

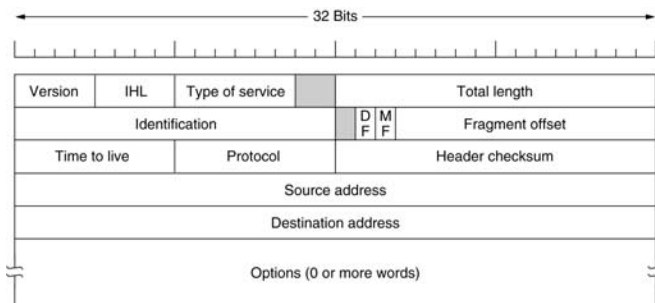


Tunneling

- Tunneling - common solution for interconnecting multiple heterogeneous networks **when the source and destination hosts are on the same type of network**
 - Multiprotocol router extracts the IP packet, inserts it in the payload field of the WAN network layer packet, and addresses the latter to the WAN address of the destination multiprotocol router
 - Destination multiprotocol router removes the IP packet and sends it the destination host

The IP Protocol

The IPv4 (Internet Protocol) header



CIDR – Classless InterDomain Routing

- Shortage of IP addresses caused by the classful addressing
 - A class is obviously too large for any organization
 - C class is too small (only 256 addresses available)
 - B class is requested and allocated, but it is still too large for most organizations => many IP addresses are wasted
- Solution - for the remaining IP addresses, CIDR is used
 - Allocate remaining IP addresses in variable-sized blocks, without regard to the classes
 - **The starting address must fall on the boundary of the block size**
 - E.g., if a site needs, say, 2000 addresses, it is given a block of 2048 addresses on a 2048-byte boundary

Routing with CIDR

- Each routing table is extended by giving it a 32-bit mask
- A single routing table for all networks consisting of an array of (IP address, subnet mask, outgoing line) triples
- When a packet comes in, its destination IP address is first extracted
- Then, the routing table is scanned entry by entry, masking the destination address and comparing it to the table entry looking for a match
- It is possible that multiple entries (with different subnet mask lengths) match, in which case the longest mask is used
 - E.g., if there is a match for a /20 mask and a /24 mask, the /24 mask is used

NAT – Network Address Translation

- Temporary solution - network address translation
 - Each company or family is assigned a single IP address (or a small number of them)
 - Within the company, every computer gets a unique private IP address, which is used for routing intramural traffic
 - When a packet exits the company and goes to the ISP, an address translation takes place
 - Three ranges of IP addresses have been declared as private:
 - 10.0.0.0 - 10.255.255.255 (16,777,216 hosts)
 - 172.16.0.0 - 172.31.255.255/12 (1,048,576 hosts)
 - 192.168.0.0 - 192.168.255.255/16 (65,536 hosts)

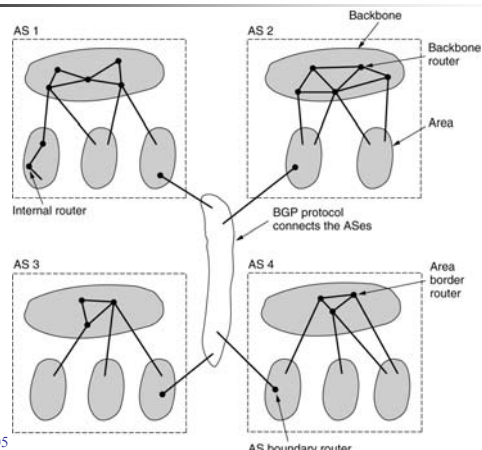
NAT – What about the Incoming Traffic?

- **Solution is based on the assumption all traffic is TCP/UDP**
- TCP/UDP has two port fields, one for source port, the other for destination port, each 16 bits wide
- The source port is used as an index to an internal table maintained by the NAT box
- The internal sender's private IP and original port info are stored in the table
- When the reply comes back, it will carry the index as the destination port, the NAT box then translates the address back
- For both outgoing and incoming address translations, the TCP/UDP and IP header checksums are recomputed

Internet Control Protocols

- ICMP – Internet Control Message Protocol
- ARP – find the mapping of IP addresses to data link layer addresses
- RARP – This protocol allows a newly-booted diskless-workstation (e.g., X terminal) to broadcast its Ethernet address and ask for its IP address
 - **RARP server** responds to a RARP request with the assigned IP address
- BOOTP – uses UDP
- DHCP – Allows both manual IP address assignment and automatic assignment. DHCP has largely replaced RARP and BOOTP

Internet Routing Protocols



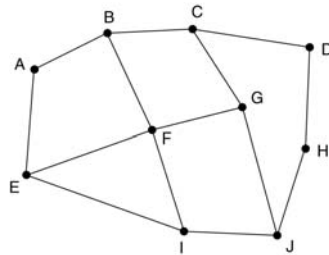
OSPF

- Within area, each router has same link state database
- Each router periodically floods LINK STATE UPDATE packets to each of its **adjacent** routers
- Each router constructs the graph for its area and computes shortest path
- Backbone router also does:
 - Accepts info from area border routers
 - Computes best route from each area border router to every other area border router
 - This info propagated back to area border routers which advertise it within their areas
- Using this info, router about to send interarea packet selects best exit router to backbone

Exterior Gateway Routing Protocol

■ Border Gateway Protocol (BGP)

- Used between autonomous systems
- Main concerns: politics, security, economic
- Uses distance vector routing except keeps track of exact path instead of cost to destination and periodically tells its neighbors that path



Information F receives
from its neighbors about D

From B: "I use BCD"
From G: "I use GCD"
From I: "I use IFGCD"
From E: "I use EFGCD"

50

Transport Layer

- **Transport Entity** - The hardware and/or software within the transport layer that does the work. The transport entity can be located in
 - Operating system kernel, a separate user process, a library package bound into network applications, or conceivably on the network interface card
- The transport layer fulfills the key function of isolating the upper layers from the technology, design, and imperfections of the subnet
 - **Transport service provider** - The bottom four layers
 - **Transport service user** - the upper layer(s)

8 November 2005

EEC484/584

Wenbing Zhao

The TCP Service Model

- Requires both sender and receiver to create socket
- Each socket has socket number (address) consisting of IP address of host and 16-bit port number
- Port is TCP name for TSAP
- Connections are identified by the socket identifiers at both ends, i.e., (socket1, socket2)
- Port numbers below 1024 are called **well-known ports** and are reserved for standard services
 - E.g., 21 for FTP, 23 for Telnet
- Full-duplex, point-to-point

8 November 2005

EEC484/584

Wenbing Zhao

51

TCP Protocol

- Sequence numbers used for ACKs and also for window mechanism
- Sender and receiver TCP entities exchange data in the form of segments
- **Segment** - fixed 20 byte header + optional part followed by data
- TCP software decides size of segments, accumulate data from several writes into 1 segment, or split data from one write over multiple segments

8 November 2005

EEC484/584

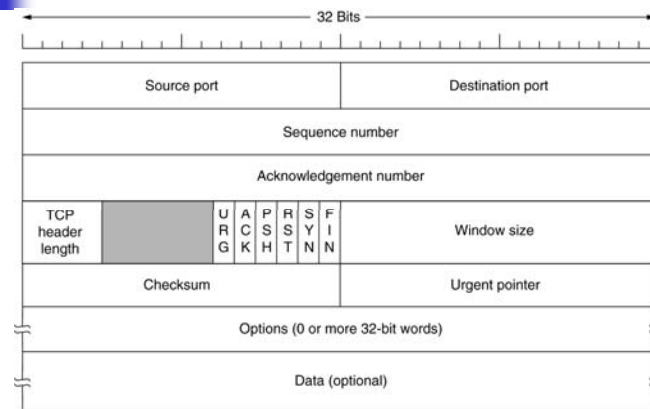
Wenbing Zhao

52

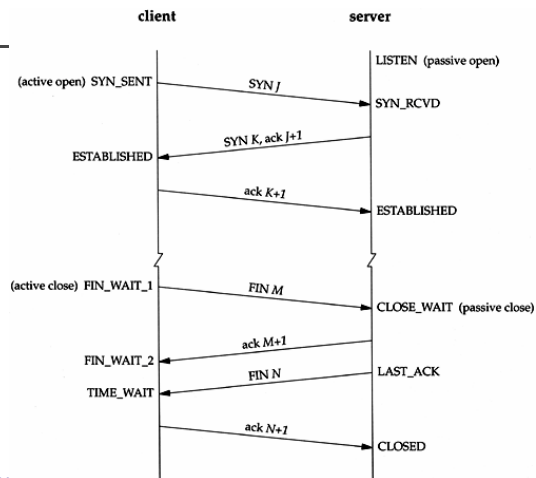
TCP Protocol

- Uses sliding window for flow control
 - Resembles go-back-n protocol
 - No selective ack, or nack, e.g., if 1-1024 and 2049-3072 are received, can only ack 1025
 - RFC 1106 propose a solution, using TCP options

The TCP Segment Header



TCP Connection Release



TCP Transmission Policy

- **Nagle's algorithm:** solution for the 1-byte-at-a-time sender problem
 - When sender application passes data to TCP one byte at a time
 - Send first byte
 - Buffer the rest until first byte ACKed
 - Then send all buffered bytes in one TCP segment
 - Start buffering again until all ACKed
 - Implemented widely in TCP, can be disabled/enabled by using socket options
 - For some application, it is necessary to disable the Nagle's algorithm, e.g., X Windows program over Internet, to avoid erratic mouse movement, etc.

TCP Transmission Policy

- **Clark's algorithm** (to avoid the silly window syndrome)
 - Receiver should not send window update until it can handle max segment size it advertised when connection established or its buffer is half empty, whichever is smaller
 - Sender should wait until it has accumulated enough space in window to send full segment or one containing at least half of receiver's buffer size
- Nagle's algorithm and Clark's algorithm are complementary

Slow Start Algorithm

- When connection is established, sender initializes congestion window size to size of max segment in use on connection, it then sends one max segment
- If segment is acked before timer goes off, it increases congestion window size by max segment size, it then sends two segments
- Doubling each time (congestion window size is increased exponentially) until timeout occurs or receiver's window size is reached

Internet Congestion Control Algorithm

- Three parameters
 - Receiver's window size
 - Sender's congestion window size
 - Threshold, initially 64KB
- When timeout occurs,
 - Threshold is set to half **current** congestion window size
 - Congestion window size is set to one max segment size
- Use slow start algorithm but stop when threshold is reached, then increase congestion window size linearly

Fast Retransmission and Recovery

- Fast retransmission algorithm
 - An ACK is generated and sent immediately when an out-of-order segment is received to notify the sender
 - When 3 or more duplicated ACKs are received in a row, it is taken as strong evidence that a segment has been lost
 - The segment that appears to have been lost is retransmitted before the retransmission timer expires
- Fast recovery algorithm
 - On fast retransmission, slow start algorithm is not used
 - Why fast recovery ?
 - Data are still flowing between two ends

Fast Recovery Algorithm

- When third duplicate ACK is received
 - Set threshold to half the current congestion window
 - Set congestion window to (threshold + 3 × segment size)
- Each time another duplicate ACK arrives
 - Increment congestion window by the segment size
 - Transmit a packet (if allowed by new congestion window)
- When the next ACK arrives that acknowledges new data, **set congestion window to threshold value**
 - This should be the ACK of the retransmission from step 1
 - Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK

TCP Timer Management

- The algorithm generally used by TCP is due to Jacobson (1988)
- For each connection, TCP maintains a variable, *RTT*, that is the best current estimate of the round-trip time to the destination in question
- *RTT* is updated according to formula

$$RTT = \alpha RTT + (1 - \alpha)M$$
 - *M* is the newly measured roundtrip time
 - α is a smoothing factor that determines how much weight is given to the old value. Typically $\alpha = 7/8$

TCP Timer Management

- Timeout = βRTT
 - Constant value was inflexible because it failed to respond when the variance went up. Need another smoothed variable, *D*, the deviation
 - Whenever an acknowledgement comes in, the difference between the expected and observed values, $|RTT - M|$, is computed.
 - A smoothed value of this is maintained in *D* by the formula

$$D = \alpha D + (1 - \alpha)|RTT - M|$$
 - Time out is set to $RTT + 4 \times D$
- **Karn's algorithm:** The timeout is doubled on each failure until the segments get through the first time
 - Do not update *RTT* on any segments that have been retransmitted

TCP Timer Management

- **Persistence timer** - used on the sending side. Designed to prevent the deadlock caused by the loss of window update packet when receiver's window is 0
- **Keepalive timer** - when a connection has been idled for a long time, the keepalive timer may go off to cause one side to check whether the other side is still there. If it fails to respond, the connection is terminated.
- The last timer used on each TCP connection is the one used in the *TIMED WAIT* state while closing. It runs for twice the maximum segment lifetime to make sure that when a connection is closed, all packets created by it have died off