IP Routing

- An important function of the IP layer is routing
  - Provides the basic mechanism for routers to interconnect different physical networks
- Usually, IP routers function with partial information
- Two types of routing are possible
  - Direct and Indirect routing

IP Routing

- Direct Routing
  - If the destination host is attached to a physical network to which the source host is also attached, datagram is sent directly
    - Datagram is encapsulated into the physical frame and delivered directly to the destination host
- Indirect Routing
  - Indirect routing occurs when the destination host is not a network directly attached to the source host
    - One or more IP routers are needed to reach the destination host
  - Address of the first router is called indirect route in the context of the IP routing algorithm
    - The address of the first router is the only information needed by the source

Indirect Routing over Subnets

- In some cases, multiple subnets may be defined on the same network
- Even if the destination host is on different subnet than the source host, indirect routing is used
  - Therefore, there is a need for a router to forward traffic between subnets

IP Routing

- Whenever the IP routing software in a host or a gateway needs to transmit a datagram, it consults a routing table
  - Both hosts and gateways have routing tables typically containing :
    - The IP address of each possible destination network, and
    - The IP address of the next gateway along the path to the destination network
- Three types of mappings can be found
  - Direct routes, for locally attached networks
  - Indirect routes, for networks reachable via one or more routers
  - Defaut routes

Routing Table Configuration

- **Routing tables are dynamic**

Net Name	Net Address	Route
SATNET	4	Directly connected
Purdue	192.5.1	2 hops via 10.2.0.37
FIBERNET	24	3 hops via 10.3.0.72

Internet Routing Table for BBN Router

Routing Algorithm
Route(Datagram, Routing_Table)

Extract destination Netid
If (Netid is of a direct neighbor) then
  send datagram directly

Else if (route specified by host)
  route datagram as specified in the table
  (based on the information in the frame)

Else if (Netid is in routing table)
  route datagram as specified in the table

Else if a default route has been specified
  route datagram to default gateway

Else declare routing error

Routing With Masks
Route(Datagram, Routing_Table, Mask)

If (Dest_add AND Mask)=(Own_add AND Mask)
  Send datagram directly
Else
  Route datagram to next hop
  (As specified by the table)

Other Internet Protocols

- **ICMP, Internet Control Message Protocol, a required companion of IP**
  - **Allows hosts to interact with gateways**
    - **Provides feedback from gateways to hosts about problems**
    - **Destination unreachable, Time exceeded, Parameter problem, Source quench, Redirect message to update route to a shorter one, Echo and Echo reply, Times-tamps request and reply for sampling delays**

ICMP Messages

- ICMP message is encapsulated in an IP datagram

IP Datagrams

IP Header	ICMP Message

CMP Message Format

Msg Type	Code	Checksum Field
Type Based Content		

- There are 15 different types of messages
- Some types of messages use different code values
- Checksum covers the entire ICMP message

ICMP Message Types

Type	Code	Description	Query	Error
0	0	Ping Echo Reply	•	
8	0	Ping Echo Request	•	

- Packet InterNet Groper (PING), tests whether another host is reachable by sending ICMP echo request message to the host, expecting ICMP echo reply from the host

Destination Reachability

Type	Code	Description	Query	Error
3		Destination Unreachable		
	0	Network Unreachable		•
	1	Host Unreachable		•
	2	Protocol Unreachable		•
	3	Port Unreachable		•
	4	Fragmentation Needed, but ‘‘don’t_frag_bit’’ Set		•
	5	Source Route Failed		•
	6	Destination Network Unknown		•
	7	Destination Host Unknown		•
	8	Source Host Isolated (obsolete)		•
	9	Destination Network Administratively Prohibited		•
	10	Destination Host Administratively Prohibited		•
	11	Network Unreachable for TOS		•
	12	Host Unreachable for TOS		•
	13	Communication Prohibited by Filtering		•
	14	Host Precedence Violation		•
	15	Precedence Cutoff In Effect		•

Elementary Flow Control

Type	Code	Description	Query	Error
4	0	Source Quench		•

- **The ''source quench'' message contains an error that ''may'' be sent by a host or a router, when it receives data faster than it can handle**
  - **A system is not required to send a source quench message, even if it runs out of buffers and throws the datagram away**

Redirect Messages

Type	Code	Description	Query	Error
5		Redirect		
	0	Redirect For Network		•
	1	Redirect For Host		•
	2	Redirect for TOS and Network		•
	3	Redirect for TOS and Host		•

ICMP Messages

- **Router Advertisement**

Type	Code	Description	Query	Error
9	0	Router Advertisement	•	
10	0	Router Solicitation	•	

- **Time Expiration**

Type	Code	Description	Query	Error
11		Time Exceeded		
	0	Time_To_Live is Zero During Transit		•
	1	Time_To_Live is Zero During Reassembly		•

Parameter Errors

Type	Code	Description	Query	Error
12		Parameter Error		
	0	Bad IP Header (catchall error)		•
	1	Required Option Missing		•

Timestamps

Type	Code	Description	Query	Error
13	0	Timestamp Request	•	
14	0	Timestamp Reply	•	

Information

Type	Code	Description	Query	Error
15	0	Information Request (obsolete)	•	
16	0	Information Reply (obsolete)	•	

Address Mask

Type	Code	Description	Query	Error
17	0	Address Mask Request	•	
18	0	Address Mask Reply	•	

ICMP Rules

- **ICMP error messages are never generated in response to an ICMP error message, to avoid recursive error generation**
- **When an ICMP error message is sent, the message always contains the IP header and the first 8 bytes of the IP datagram that caused the error**
  - **Allows the ICMP receiver to uniquely identify the protocol, and the protocol port number associated with the message**

ICMP Rules

- ICMP error message is never generated for:
  - a datagram destined to an IP broadcast address, or
  - a datagram sent as a link-layer broadcast, or
  - a fragment other than the first, or
  - a datagram whose source address does not define a single host (zero address, loopback address, broadcast address or multicast address)
- These rules are meant to prevent ‘‘broadcast storms’’

IP User Datagram Protocol (UDP)

- UDP provides unreliable connectionless delivery service using IP to transport messages
- UDP does not enhance the best effort service provided by IP
  - Reliability is relegated to the application
- UDP provides a mechanism to distinguish among multiple destinations within a host
  - The mechanism is based on the notion of protocol ports

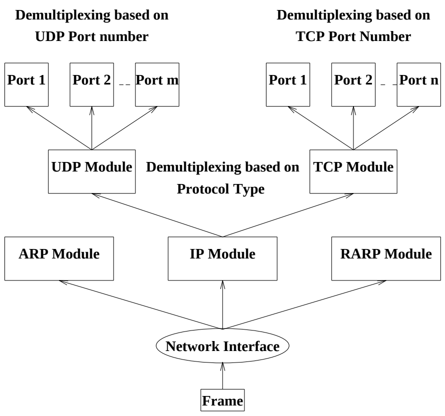
IP Communication Ports

- Specifying a process Id as the ultimate destination address may not be adequate
  - Processes are dynamic
  - Functions identify destinations, not processes
- Port, an abstract communication point
  - The ultimate destination of IP datagrams
  - Ports are usually synchronous
  - OS manages ports and provide an interface to them

IP Communication Ports

- To communicate, a sender needs to know :
  - IP address of the remote host and
  - Port number within that host

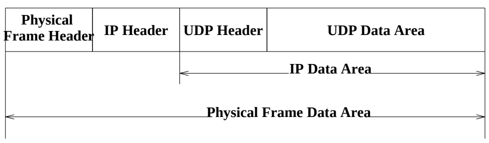
Multiplexing in UDP



User Datagram Format

UDP Source Port	UDP Destination Port
UDP Message Length	UDP Checksum
Data	
....	

UDP Frame Encapsulation



- **Source port optional and set to 0 if not used.**

UDP Checksum

- **UDP checksum is optional and need not to be used (set to 0)**
  - **Using UDP checksum option is useful, however, since IP checksum does not cover the data portion of the datagram**
  - **It provides the only way that data has arrived intact and should be used**

UDP Checksum

- **The UDP checksum covers more information than is present in the UDP datagram**
  - **A pseudo-header is prepended to the UDP datagram, and a checksum over the entire object is computed**
    - **Pseudo-header contains source and destination IP addresses, IP protocol type code (17 for UDP) and UDP datagram length (pseudo-header not included)**
    - **It guarantees that the datagram has reached the proper destination**

Transmission Control Protocol
TCP

- TCP provides a reliable virtual circuit service
  - TCP ‘‘guarantees’’ ordered delivery of a stream of data without loss or duplication
    - Despite unreliable network packet delivery service

TCP

- TCP assumes little about the underlying communications system
- TCP be used with a variety of packet delivery services:
  - IP datagram delivery service
  - Dialup telephone lines
  - Local Area Networks
  - Lower speed long haul networks
  - High speed fiber optics

Transmission Control Protocol

- TCP can be accommodated to a specific delivery service with minor modifications
  - Passing parameters to IP for inclusion in the IP datagram header
  - Incorporating ‘‘pseudo-headers’’

TCP Interface Characteristics

- Stream orientation, data is viewed as a stream of bytes
- Virtual Circuit connection
- Buffered transfer
  - TCP collects enough data to fill a reasonably large datagram before transmission
  - TCP provides a push mechanism to force transfer, if needed

TCP Interface Characteristics

- TCP provides an unstructured stream
  - TCP does not mark boundaries between streams
  - Application's responsibility to understand stream contents
- Full duplex connection
- TCP specifications describe generally how applications use TCP, but does not dictate details of an interface

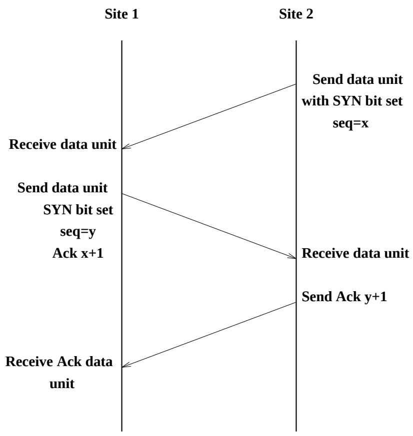
TCP Connection establishment

- Initially, a service user at site 1 issues a passive open function
  - Willingness to accept connections from other users
  - TCP returns a passive open confirm with local connection name to use when a connection is set up
  - Service user at site 1 can start ''listening'' for requests

Connection Establishment

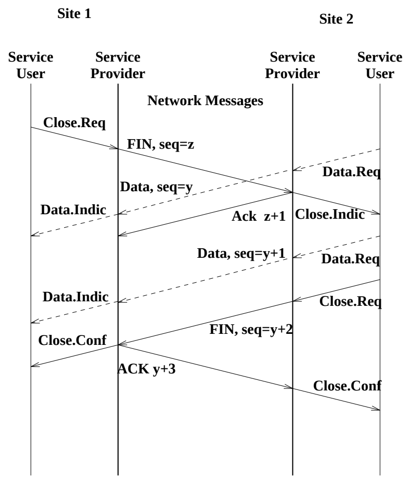
- To establish a connection with service at site 1, user at site 2 issues an active open
  - Active open contains data needed to set up the connection
    - This begins a three-way handshake

Three-way Handshake



Connection Termination

- The normal means for closing a connection is a graceful close



TCP Segment Format

<table>
<tr><td colspan="3">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="5">Sequence Number</td></tr>
<tr><td colspan="5">Acknowledgement Number</td></tr>
<tr><td>HLen</td><td>Reserved</td><td>Code Bits</td><td colspan="2">Window</td></tr>
<tr><td colspan="3">Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="4">Options (If Any)</td><td>Padding</td></tr>
<tr><td colspan="5">Data</td></tr>
<tr><td colspan="5">.........</td></tr>
</table>

TCP Segment Fields

- Sequence number identifies position in sender’s byte stream of the data in the segment
- The Ack number identifies the number of the octet the source expects to receive next
- HLen specifies the length of the segment header
  - Required because option field is variable

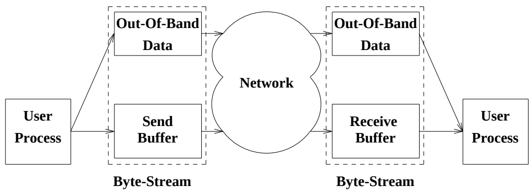
TCP Segment Fields

- Code bits (6) determine the purpose of the segment content
  - ACK (100000) : Acknowledgment field is valid
  - URG (010000) : Urgent pointer field is valid
  - PSH (001000) : This segment requests a push
  - RST (000100) : Reset connection
  - SYN (000010) : Synchronize sequence numbers
  - FIN (000001) : Sender end byte stream is reached
- Window, a 16-bit flow control credit allocation, in octets

Out of Band Data

- Out of band data is needed to handle abort or program interrupt signals
  - These signals should not wait for the octets already in the TCP stream to be transmitted
- TCP uses URGENT feature to accommodate out-of-band data

Out of Band Data



Out of Band Data

- TCP notifies associated application of the beginning and end of urgent mode
  - How TCP informs the application depends on the OS
- Urgent data is detected by the URG code set and retrieved by Urgent Pointer

TCP Segment Fields

- TCP uses a checksum field to verify that the segment has reached its correct destination (IP address and Port number)
  - A pseudo-header is used. The procedure to compute the checksum is similar to the IP and UDP procedure
    - The pseudo-header is prepended to the segment, and padding is added to make segment multiple of 16 bits
    - A 0 is first stored in the checksum field then a 16-bit one's complement sum of the entire object including the pseudo-header, TCP header and user data is accumulated, and the resulting sum is stored in the checksum.
  - Paddings and pseudo-header are neither counted in the segment length nor transmitted

TCP Options

- **Originally, one option, maximum segment size was defined**
  - **The 16-bit option may be used only in the initial connection request segment**
- **More recently, two other options have gained widespread acceptance**
  - **Window scale factor**
  - **Timestamps**

TCP Options

End of Option

kind=0
1 Byte

No Operation : Padding

kind=1
1 Byte

Maximum Segment Size

kind=2	len=4	MSS
1 Byte	1 Byte	2 Bytes

Window Scale Factor

kind=3	len=3	shift count
1 Byte	1 Byte	1 Byte

Timestamp

kind=8	len=10	timestamp value	timestamp echo reply
1 Byte	1 Byte	4 Byte	4 Byte

Window Scale Option

- **This option increases the definition of the window from 16 to 32 bits**
  - **A much better alternative than changing the TCP header to accommodate the need for larger windows**
- **When window flow option is used, the value of the field is multiplied by 2^F, where $0 < F \leq 14$ is a scale option specified in the initial connection request segment**
  - **A value $F = 14$ results in a window of 1,073,725,440 bytes (65535 2^{14})**
- **Internally, TCP maintains a ''real'' window size of 32 bits**
- **The option can only appear in a SYN segment**
  - **Thus, scale factor is agreed upon and fixed in each direction at connection setup**

Timestamps Option

- **This option allows a sender to place a timestamp value in every segment**
  - **The receiver reflects this value in the ack, allowing the sender to calculate an RTT for each received Ack**
  - **Larger window sizes may require more than one RTT update per window**
    - **This option allows the sender to compute multiple RTTs per window**

Timestamps Option

- Timestamp is a monotonically increasing value
  - The receiver echoes the value received and usually does not care about the time unit used
    - Thus, clock synchronization between sender and receiver is not required
- Some systems increment the timestamp clock once every 500 ms and reset it to 0 on a reboot

Timestamps and Acks

- An ack segment may cumulatively acknowledge more than one segment at a time
  - This leads to potential ambiguity
    - If the receiver sends an Ack to acknowledge two received data, which received timestamp is sent back in the timestamp echo reply field?
  - May require some state information to be kept at the end systems

Timestamps and Acks

- To minimize the amount of state information, only a single timestamp is kept per connection
- TCP uses two variables to decide when to update the timestamp value
  - ‘‘tsrecent’’ keeps track of the timestamp value to send in the next ACK
  - ‘‘lastack’’ keeps track of the ack sequence number from the last Ack that was sent
    - This is the sequence number expected next by the receiver

Timestamps Update Algorithm

- When a segment arrives, if the segment contains the byte number lastack, then the timestamp value from the segment is saved in tsrecent
- Whenever a timestamp option is sent, tsrecent is sent as the timestamp echo reply and the sequence number field is saved in lastack

TCP Data Flow Control

- TCP uses a variable size window protocol to regulate flow control
- Its unique feature is that is decouples acknowledgements of received data units from the granting of a permission to send additional data
  - Each acknowledgment contains a window advertisement
    - Increased window advertisement, allows the sender to proceed with sending octets not acknowledged yet
    - Decreased window advertisement, causes the sender to stop at the boundary of the window

Acknowledgments and Retransmission

- TCP uses a cumulative acknowledgment scheme
  - Ack reports on the number of octets so far accumulated and specifies the next octet expected
- Advantages
  - Acks are easy to generate unambiguously
  - Lost Acks do not necessarily force retransmission
- Disadvantages
  - Receiver does not report on the segments successfully received, if one intermediate segment is lost
- Timeout mechanism becomes very crucial

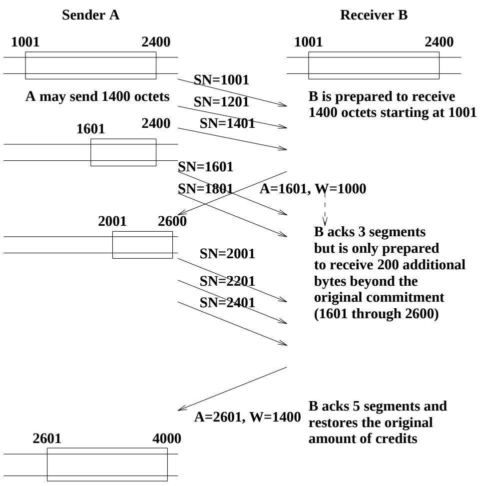
TCP Window Mechanisms

- TCP uses a credit allocation scheme
  - The sender includes the sequence number of the first octet in the segment data field
  - The receiver acks the incoming segment with a message of the form (Ack=i, Win=j)
    - All octets up to i-1 are acked
    - Permission is granted to send an additional window of j octets starting at i through i+j-1

TCP Credit Allocation Scheme

- The credit allocation scheme is flexible
- Assume last message issued by receiver carries ACK=i and Win=j
  - To increase credit an amount k>j, when no additional data have arrived, receiver issues Ack=i and Win=k
  - To ack an incoming segment containing m octets of data (m<j) without granting additional credits, receiver issues Ack=i+m and W=j-m
- Receiver can issue cumulative acks, and is not required to issue an ack immediately after receiving a segment

TCP Window



Effect of Window Size

- W=TCP Window Size (bytes)
- R=Source Data Rate (bps)
- D=Propagation Delay (sec)
- S, Maximum Achievable Normalized Throughput, can be expressed as:
  - S=1 if W > 2 $\times$ RD bits or RD/4 bytes
  - S=4W/RD bytes if W<RD/4

TCP Actual Throughput

- TCP performance can be affected by several complicating factors
  - In most cases, a number of TCP connections are multiplexed over the same network interface, so each connection is only allocated a fraction of bandwidth
    - This reduces R, and therefore S as well
  - TCP connections usually involve a hop across multiple networks
    - Router delay becomes the biggest contributor to D
  - Actual data rate may be reduced below R if a bottleneck router exists along the path
  - If any segment is lost and must be retransmitted, throughput is reduced

TCP Flow Control

- The rate at which a TCP entity can send data is determined by the rate of incoming ACKs to previous segments
  - This rate is determined by the bottleneck in the round trip path between the source and destination

TCP Window Control

- TCP sliding window must not only control end-to-end flow but congestion control as well
- TCP relies on ACKs to control flow and congestion
- Absence of and ACK within the timeout interval signals congestion

TCP Window Control

- TCP does not rely on an explicit negative acknowledgment to detect errors
  - TCP relies exclusively on positive acks and retransmission when an ack does not arrive within a given timeout period
- Retransmission TimeOut (RTO) should be on the order of RTT
  - RTT and its statistics, however, vary considerably as the Internet load and network change
    - Thus, timers play a major role in TCP

Timer Values

- The timeout timer should be set to a value somewhat longer that the Round Trip Time (RTT)
- Two strategies can be used to set the timer value:
  - Static value timers
  - Adaptive value timers

Static Value Timers

- A fixed timer value based on the Internet typical behavior is used for RTO value
- The strategy suffers from the inability to respond to Internet changing conditions
  - A high value leads to sluggish service and unnecessary long time to respond to a lost packet
  - A low value leads to a feedback condition that causes more retransmission in case of congestion

Adaptive Timers

- TCP keeps track of the time taken to ack the data segments and sets its retransmission timers based on the average of the observed delays
- Can this be trusted?
  - Cumulative ack by the peer may delay acknowledgment
  - If a segment has been retransmitted, the sender cannot determine which segment, first or second, does the ack correspond to.
  - Internet load may change suddenly
- Still, tracks better RTT variation than fixed strategy

Adaptive Retransmission Timers

- The retransmission strategy must:
  - Estimate RTT based on observed past values of RTTs
  - Set the Retransmit Time Out (RTO) value to reflect current level of network load
    - Find suitable functions, f(), of estimated RTT
      - Set RTO = f(Estimated RTT)
      - It is clear that RTO should be at least $2 \times$RTT

Adaptive Retransmission Timers
Equally Weighted Average RTT Estimate

- One approach would simply be to average the observed RTTs over a number of segments

$$ARTT(k+1) = \frac{1}{k+1}\sum_{i=1}^{k+1} RTT(i)$$

$$ARTT(k+1) = \frac{K}{k+1} ARTT(k) + \frac{1}{k+1} RTT(k+1)$$

- Each term of the summation is weighted equally
  - However, most recent values are more likely to reflect future behavior
    - Greater weight must be given to most recent values

Adaptive Retransmission Timers
Smoothed Average

- An estimated value of RTT is computed as a smoothed average

$$SRTT(k+1) = \alpha SRTT(k) + (1-\alpha) RTT(k+1),$$
$$0 \le \alpha \le 1$$

- The smaller the value of α is the greater the weight is given to more recent observations
  - With small values of α, the average quickly reflects a rapid change in the observed quantity
  - Disadvantage, a brief surge in the observed valued followed by a return to the average value causes fluctuations
    - $\alpha \approx 0.9$ recommended

Adaptive Retransmission Timers

- Given an SRTT, what should the RTO value be?
- Use a constant value:
  $$RTO(k+1) = SRTT(k+1) + \Delta$$
- Δ is not proportional to SRTT
  - If SRTT(k+1) is large, Δ may become insignificant relatively
    - As a consequence, fluctuations in the RTT causes unnecessary retransmissions
  - If SRTT(k+1) is small, Δ may become relatively large
    - As a consequence, fluctuation in the RTT causes unnecessary delays in retransmission

Adaptive Retransmission Timers

- An alternative is to use timer values that are proportional to SRTT

$$RTO(k+1) = Min(U, Max(L, \beta SRTT(k+1)))$$

  - U: Preselected fixed upper bound
  - L: Preselected fixed lower bound
  - β constant
- Originally, $0.8 \le \alpha \le 0.9$ and $1.3 \le \beta \le 2.0$ were suggested (RFC 793)
  $$RTO(k+1) = \beta SRTT(k+1)$$
  - How effective is this scheme in case of network congestion?

TCP Congestion Control

- Congestion control in the Internet is complex
- IP is a connectionless, stateless protocol
  - No provision for determining congestion, let alone control it
- TCP provides only end-to-end flow control and relies on implicit feedback to deduce the presence of a congestion
  - ICMP quench messages too crude to provide any effective control
- No cooperative, distributed algorithm exists between different TCP entities
  - To the contrary, TCP entities may be selfish and do not cooperate to maintain some level of control

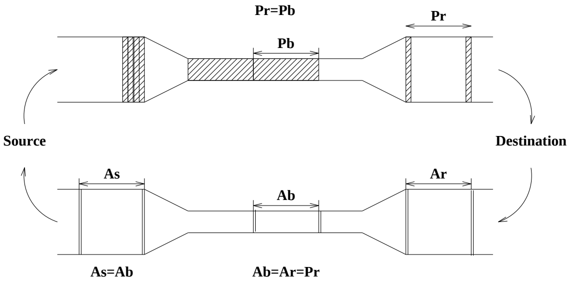
TCP Congestion Control

- The only tool is the sliding-window flow control and error control mechanisms
  - Not enough, in a dynamic environment
- A number of clever techniques have been developed and added to control congestion in the Internet
  - Mechanisms for congestion detection
  - Mechanisms for congestion avoidance
  - Mechanisms for congestion recovery

TCP Flow and Congestion Control

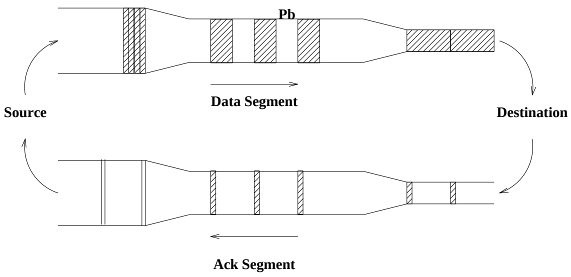
- **TCP is Self-Clocking: returning of acks functions as pacing signals**
  - **After an initial burst, the sender's segment rate matches the arrival rate of acks**
    - **Sender rate equals the rate of the slowest link on the path**
      - **This way TCP senses the bottleneck and regulates its rate accordingly**

TCP Flow and Congestion Control Network Bottleneck



- **Pb is the minimum segment spacing on the slowest link**
- **Pr is the segment spacing at the receiver**
- **Ar is the Ack spacing at the receiver**
- **Ar is the Ack spacing at the bottleneck**

TCP Flow and Congestion Control Receiver Bottleneck



- **The self-clocking behavior works equally well if the bottleneck is the receiver**

TCP Flow and Congestion Control

- **Problem, the sender does not know where is the bottleneck: network or receiver**
  - **If Acks arrive relatively slow due to network congestion then the sender must transmit segments at a rate slower than the Acks**
  - **On the other hand, if slow pace is due to the receiver, this pace should dictate the transmission policy**
- **Thus, TCP sliding window must be enhanced to factor in network congestion**

Improving TCP Congestion Effectiveness

- **Three techniques have been suggested for retransmission timer management**
  - **RTT Variance Estimation**
  - **Exponential Backoff**
  - **Karn’s Algorithm**
- **Four techniques have been suggested for window management**
  - **Low Start**
  - **Dynamic window sizing on congestion**
  - **Fast retransmission**
  - **Fast Recovery**

RTT Variance Estimate Factors of Influence

- **RTT exhibits relatively high variance due to three sources**
  - **A low data rate on the TCP connection relative to propagation time makes transmission delay relatively important in RTT estimation**
    - **Thus, high variation in datagram sizes induces high variation in RTT**
    - **SRTT estimator can be heavily influenced by characteristics of the data, that have nothing to do with the network**
  - **Abrupt changes in Internet load and condition**
  - **TCP receiver may not ack each segment individually, but cumulatively**

RTO Scheme Scheme Revisited

- **TCP original scheme suggests to compute RTO as follows:**

$$RTO(k+1) = \beta \times SRTT(k+1),\ \beta = 2$$

- **if RTT has low variance, it results in unnecessary high values of RTO**
- **In unstable environments, it may be inadequate to protect against retransmissions**

RTT Variance Estimates

- **A more effective approach is to estimate RTT standard deviation**
  - **Too costly as it involves square and square root calculation**
- **A less expensive variation measure is to estimate the mean deviation**

$$MDEV(X) = E\left[\ \left| X - E(X) \right|\ \right]$$

- **A simple average could be used to estimate MDEV**

$$AERR(k+1) = RTT(k+1) - ARTT(k)$$

$$ARTT(k) = \frac{1}{k+1}\sum_{i=1}^{k} RTT(i)$$

$$ADEV(k+1) = \frac{1}{k+1}\sum_{i=1}^{k+1} \left|AERR(k+1)\right|$$

$$ADEV(k+1) = \frac{k}{k+1}ADEV(k) + \frac{1}{k+1}\left|AERR(k+1)\right|$$

RTT Estimates
ADEV Analysis

- **All the terms of ADEV are given equal weight**
  - **Each term is multiplied by** $\frac{1}{k+1}$
- **Again, it may be desirable to give greater weight to more recent instants, because they are more likely to reflect the future**

Dynamic Estimate of RTT Variability

- **The algorithm to dynamically estimate the variability in estimating RTT uses an exponential smoothing technique**
  - **SRTT(k+1) = (1 - g) × SRTT(k+1) + g × RTT(k+1)**
  - **SERR(k+1) = SRTT(k+1) - SRTT(k)**
  - **SDEV(k+1) = (1 - h) × SEDV(k) + h × SERR(k+1)**
  - **RTO(k+1) = SRTT(k+1) + f×SEDV(k+1)**
    - **g=0.125, h=0.25 and f=4**

TCP Congestion Control

- **RTT variance estimation is efficient in detecting data loss**
- **It may not be sufficient in case of retransmission**
- **Two other factors must be taken into consideration when managing retransmissionP timers in case of congestion**
  - **What RTO value should be used?**
    - **Exponential RTO Backoff**
  - **What RTT samples should be used in case of retransmission?**
    - **Karn's Algorithm**

Exponential RTO Backoff

- **Timeouts are usually due to congestion**
  - **Maintaining the same RTO value for retransmission is ill advised**
    - **May cause sustained congestion due to the development of a similar pattern of behavior of all active TCP connections**
    - **All sources wait for local RTO time and retransmit again**
  - **A more efficient scheme must try to increase RTO values to give the Internet more time to clear the congestion**
    - **Exponential RTO Backoff scheme**
    - **RTO = q × RTO (q=2, typical value)**

RTT Samples
Karn's Algorithm

- Ambiguity may occur when a timeout occurs and the same segment is retransmitted
  - Upon receiving an ACK, the sender has no way of determining which segment, the first or the second, has been acked
- Feeding the resulting RTT into the RTO algorithm may lead to false measurements
  - False measurements lead to oscillations

Karn's Algorithm

- Karn's Algorithm uses the following rules:
  - Do not use the measured RTT for a retransmitted segment to update SRTT and SDEV
  - Calculate the backoff RTO when a retransmission occurs
    - RTO = q $\times$ RTO
    - Use the backoff RTO value for succeeding segment until an acknowledgment arrives for a segment that has not been retransmitted
      - When the ack is received to an unretransmitted segment, use the normal algorithm to update SRTT and SDEV and compute future RTO values

Window Management Techniques

- In addition to techniques for improving the effectiveness of the retransmission timer, a number of strategies to effectively manage the sending window have been suggested
  - Slow Start
  - Dynamic window sizing on congestion
  - Fast retransmission
  - Fast recovery
- These techniques are incorporated in all modern implementations of TCP

Slow Start

- TCP is self-clocking and paces itself approximately to rate of the bottleneck
  - At the initialization phase, no such pacing is available for guidance
- Sending at full window speed is ill-advised as it may cause large packet drops
  - Some means of gradually expanding the initial window until pacing takes over is needed
  - Slow start is the procedure recommended

Slow Start

- TCP makes use of a congestion window, cwnd, measured in segments, rather than in octets
- At any point in time, the allowed window is constrained by the following relationship
  - awnd = Min [credit, cwnd]
    - awnd is the allowed window in segments and represents the number of segments the sender is allowed to send without waiting for an Ack
    - cwnd is the congestion window, in segments
    - credit is the amount of unused credit granted in the most recent ack, in segments
      - When an Ack is received, the value is computed as window/(|segment|), where window contained in the incoming TCP segment represents the amount of data the receiver is willing to accept

Slow Start

- Initially, when a new connection is opened cwnd is set to 1
- On receipt of an Ack, TCP increments cwnd by one and sends two segments
- On receipt of the Acks, TCP increments cwnd by two, one for each Ack, and sends 4 segments
- This process continues until the window allowed by the network and the receiver is reached

Slow Start Analysis

- In essence, slow start mechanism probes the network to find out how many segments can be accepted by the network
- Exponential, rather than linear increase of cwnd size
  - Slow in the sense that the sender is not allowed to send the advertized window worth of segments, when no segment is in transit

Dynamic Window Sizing on Congestion

- In case of a congestion, the exponential growth of cwnd under slow start may be too aggressive and may worsen the congestion
  - It is easy to drive the network into congestion and hard to recover from it
    - Once congestion occurs, it may take a long time for congestion to clear
- In case of congestion, it is recommended to use slow start but with a linear growth in cwnd

Dynamic Window Sizing on Congestion Rules

- When a timeout occurs, use the following rules
  - Set a slow-start threshold, ssthresh, equal to half cwnd
    - ssthresh = cwnd/2
  - Set cwnd=1 and perform slow start until cwnd=ssthresh
    - In this phase, cwnd is increased only by 1 for every Ack received
  - For cwnd $\geq$ ssthreh, increase cwnd by 1 for each round-trip time

Fast Retransmit

- Both the original and enhanced scheme set the RTO value at a value ‘‘noticeably greater than the SRTT
- Several factors justify this strategy
  - RTO is computed based on the prediction of the next RTT, estimated from past values of RTT
    - Delay fluctuation may cause the estimated RTT to be smaller than the actual RTT
  - Delay fluctuation at the receiver may also make the estimated RTT unreliable
  - Cumulative Acks contributes to RTT fluctuations
- These factors can cause TCP to become slow to retransmit, when segments are lost

Segment Loss

- Many segments can be lost, based on the policy adopted by the destination TCP
  - An In-order accept policy at the destination may result in a large packet drops
  - If an in-window accept policy is used at the destination, slow retransmission can cause problems
    - Sender sends a sequence of segments, the first of which is lost
      - As long as the send window is not empty and RTO does not expire, sender can continue sending
    - Receiver receives the remaining segments but must buffer them until the first one arrives
      - If retransmission of first segment is delayed too long, the receiver will have to discard incoming segments

Dealing with Slow Retransmission

- To enhance the performance provided by RTO, two procedures have been suggested
  - Fast retransmit
  - Fast recovery

Fast Retransmit Rules
Receiver Side

- If the received segment is out of order, the receiver immediately issues an Ack for the last segment received in order
  - The receiver continues to send this Ack until the missing segment arrives
- When the missing segment arrives, the receiver sends a cummulative Ack for all of the in-order segments received so far

Fast Retransmit Rules
Sender Side

- A sender may receives a duplicate Ack for two reasons:
  - 1. The segment following the Acked one was delayed, but may still arrive although out of order
    - The sender should not retransmit the segment
  - 2. The segment is lost and must be retransmitted
- To avoid unnecessary retransmissions, it is recommended that the sender waits until it receives 3 duplicate Acks to the same segment
  - Total of 4 Acks for the same segment

Fast Recovery

- When a sender retransmits a segment using fast retransmit, it assumes that a segment was lost, even though it has not timeout yet on that segment
  - Accordingly, sender should take congestion avoidance measures
    - Set ssthresh to cwnd/2
    - Set cwnd=1
    - Begin exponential slow-start until cwnd=ssthresh and then increase cwnd linearly
- This approach may be unnecessary since the return of the Acks indicates that segments are getting through fairly regularly
  - Fast recovery is suggested

Fast Recovery Rules

- 1. When the third duplicate Ack arrives
  - Set ssthresh = cwnd/2
  - Retransmit missing segment
  - Set cwnd= ssthresh + 3
    - Adding 3 accounts for the number of segments, received and buffered by the receiver, which triggered the transmission of duplicate Acks

  2. Each time an additional duplicate Ack for the same segment arrives, increment cwnd by 1 and transmit a segment if possible
  - This accounts for the last segment that triggered the last duplicate Ack
- 3. When the next Ack for a new data arrives, set cwnd=ssthresh