END-TO-END CONNECTIVITY (TCP)

December 6, 2020

Outline

- UDP
- TCP
  - Headers
  - Connection Establishment
  - Transition Diagram
  - Data Transfer
  - Congestion Control
- Equilibria: A Model for TCP

UDP

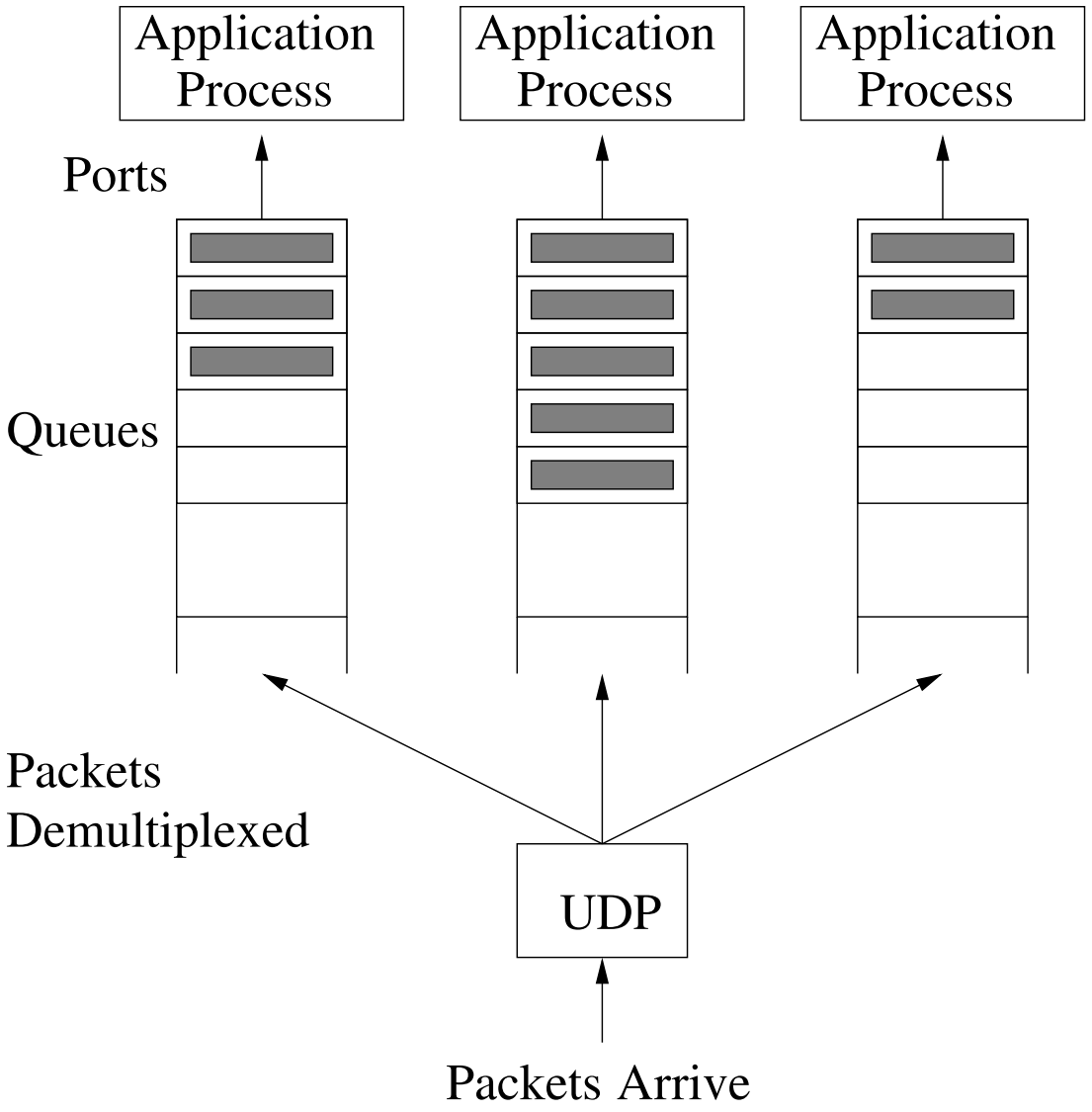
December 6, 2020

UDP

- UDP[a] (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP)
- It is used primarily for establishing low-latency and loss-tolerating connections between applications on the internet.
- Computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network.
- It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network; there is no guarantee of delivery, ordering, or duplicate protection.

[a] UDP was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP Example


Application Process
Application Process
Application Process
Ports
Queues
Packets Demultiplexed
UDP
Packets Arrive

UDP (A Simple Demultiplexer)

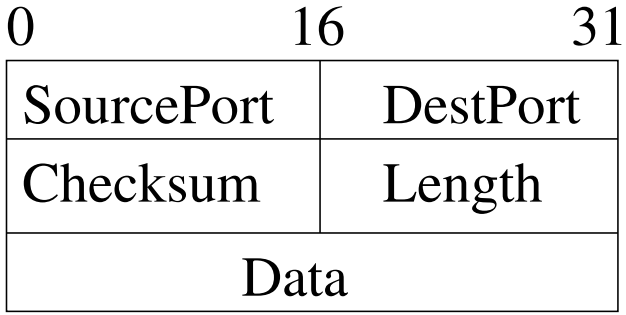
- Many processes are running on a given host.
- UDP extends the host-to-host delivery service into a process-to-process communication service and allows multiple application processes on each host to share the network.
- In UDP processes are identified by an abstract locator (also called port or mailbox).
- TCP is a connection-oriented protocol and UDP is a connection-less protocol. TCP establishes a connection between a sender and receiver before data can be sent. UDP does not establish a connection before sending data.

Ports

- Port numbers-mapping is published periodically in an RFC, and in UNIX they are available in file: /etc/services.
- Ports are implemented by message queues. Arriving messages are appended in the queue. No flow control is available.
- An application process removes messages from the queue, if there are any, otherwise the process blocks.
- How does one process learn the port number of another process?

UDP and Processes

- Servers accept messages in a well-known port:
  - Client process initiates message exchange with a server process: client's message has its port number in the message header and server can reply to it.
- Source process sends message to port, and application process receives message from port.



- At most $2^{16} \approx 64K$ ports are possible. Not enough, but they need to be intepreted only in a single host and not across entire internet.

UDP and Processes

- UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application;
- UDP avoids the overhead of such processing in the protocol stack.
- Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed due to retransmission, which may not be an option in a real-time system.

UDP Usage

- Numerous key Internet applications use UDP, including:
  - the Domain Name System (DNS), where queries must be fast and only consist of a single request followed by a single reply packet,
  - the Simple Network Management Protocol (SNMP),
  - the Routing Information Protocol (RIP), and
  - the Dynamic Host Configuration Protocol (DHCP).
- UDP is best suited for applications that require speed and efficiency.
  - VPN tunneling, Streaming videos, Online games, Live broadcasts, Domain Name System (DNS), Voice over Internet Protocol (VoIP), Trivial File Transfer Protocol (TFTP)

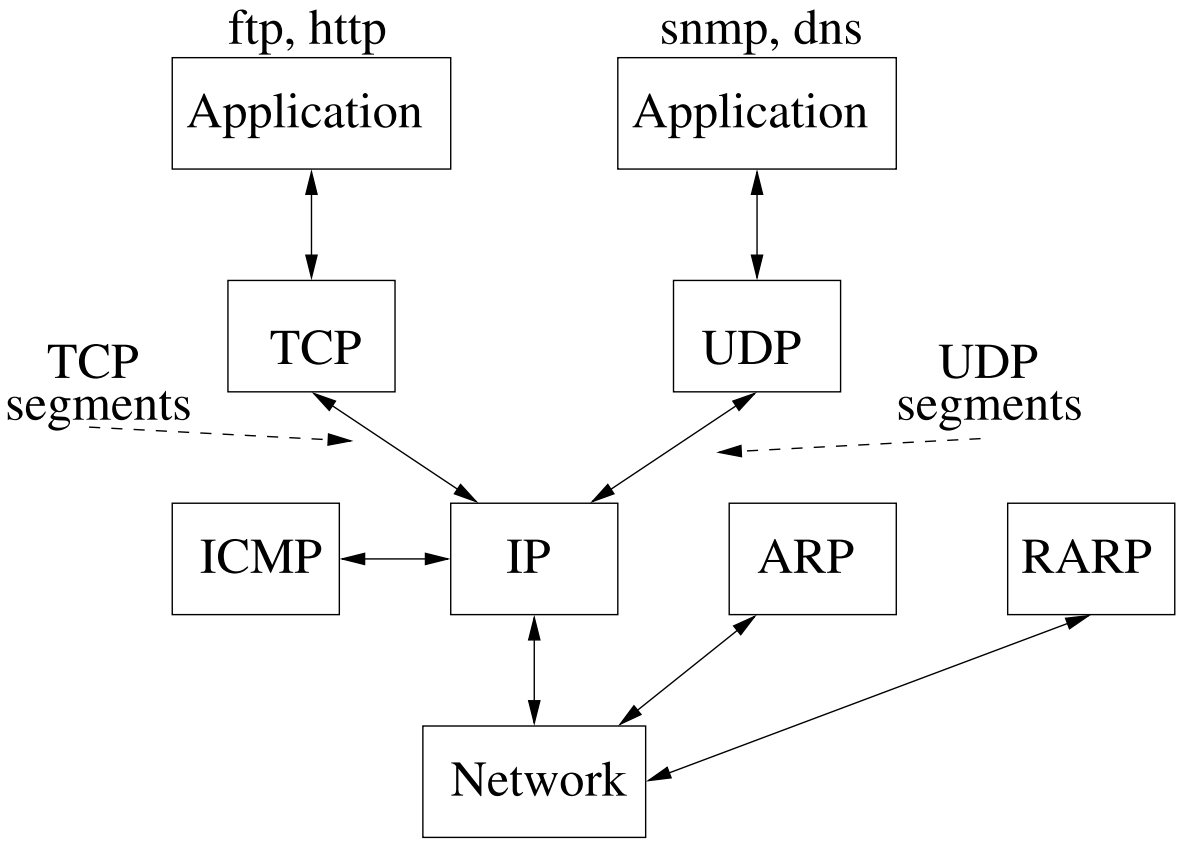
TCP

December 6, 2020

TCP

- TCP is based on the End-to-End connectivity paradigm: **functions should not be provided at lower system levels unless they can be correctly implemented at that level.**
- TCP is a reliable byte stream protocol. Its main features are
  1. A TCP sliding window protocol.
  2. TCP connections have very variable round trip times.
  3. TCP Packets may arrive out of order.
  4. TCP includes mechanism so that connections learn of each other's resources.
  5. TCP includes mechanisms to monitor congestion and control resource allocation.

TCP/IP/UDP Architecture

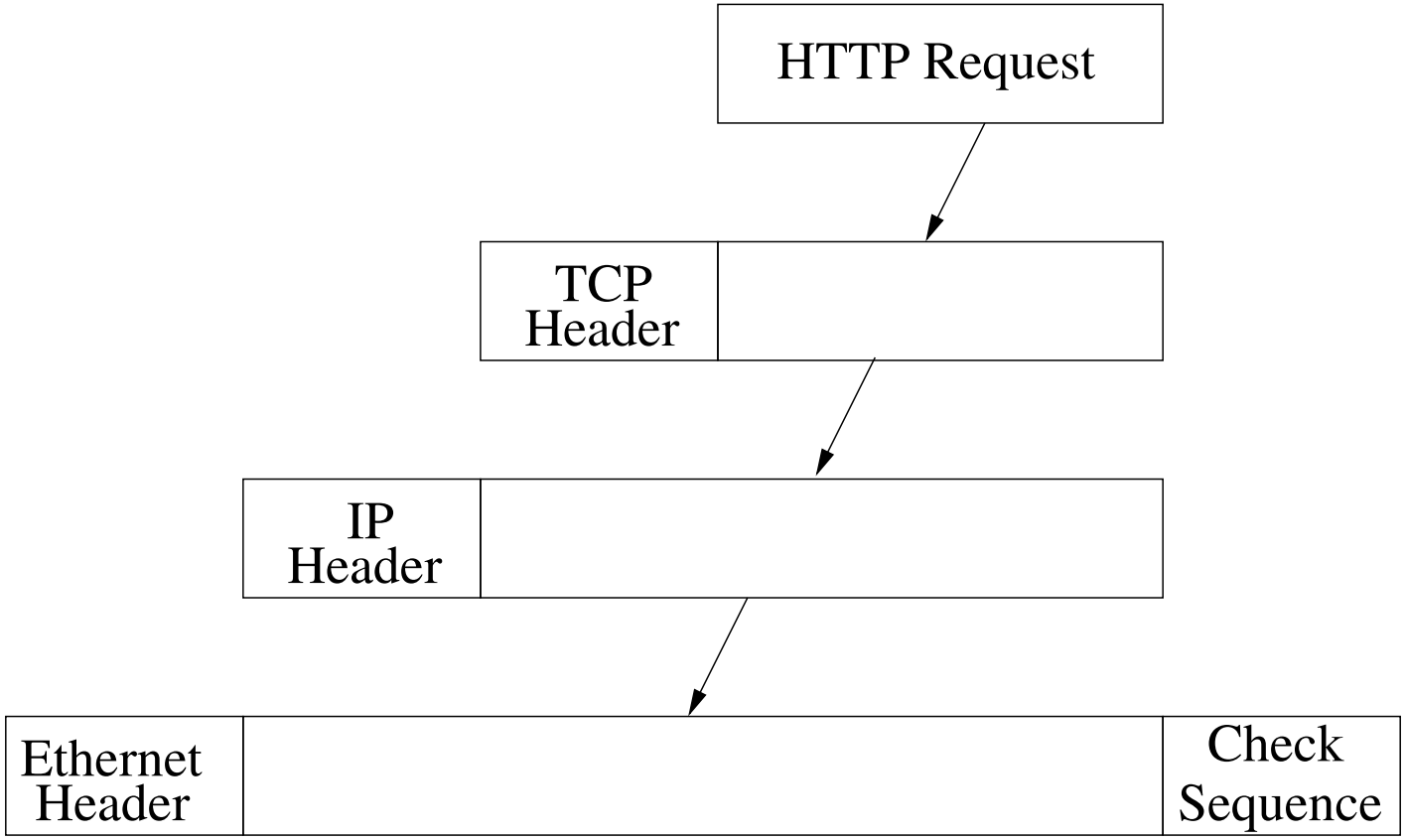


ARP = Address Resolution Protocol

RARP = Reverse Address Resolution Protocol

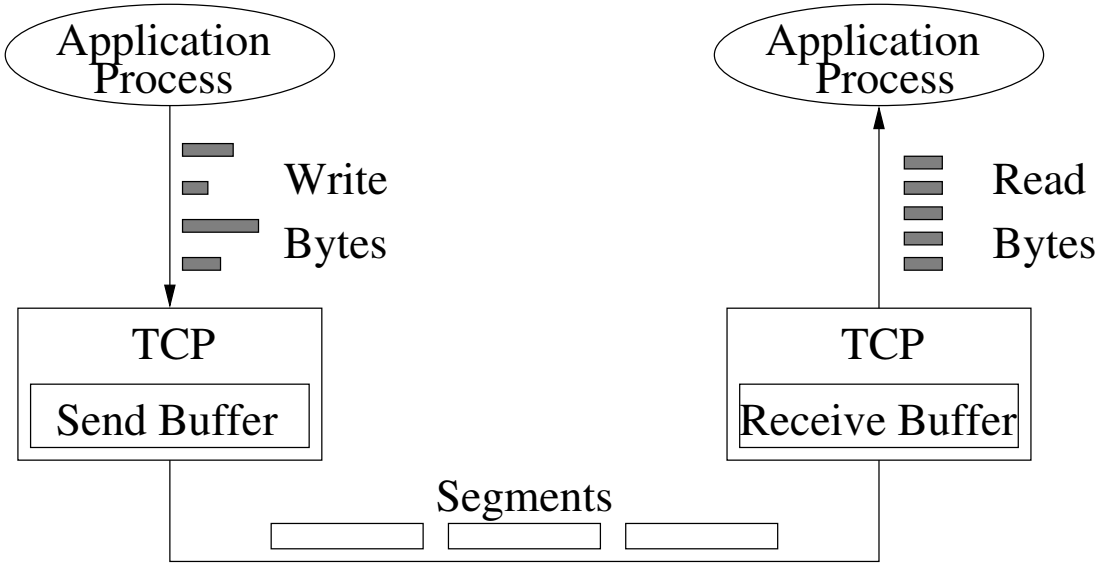
ICMP = Internet Control Message Protocol

TCP/IP Packet Encapsulation

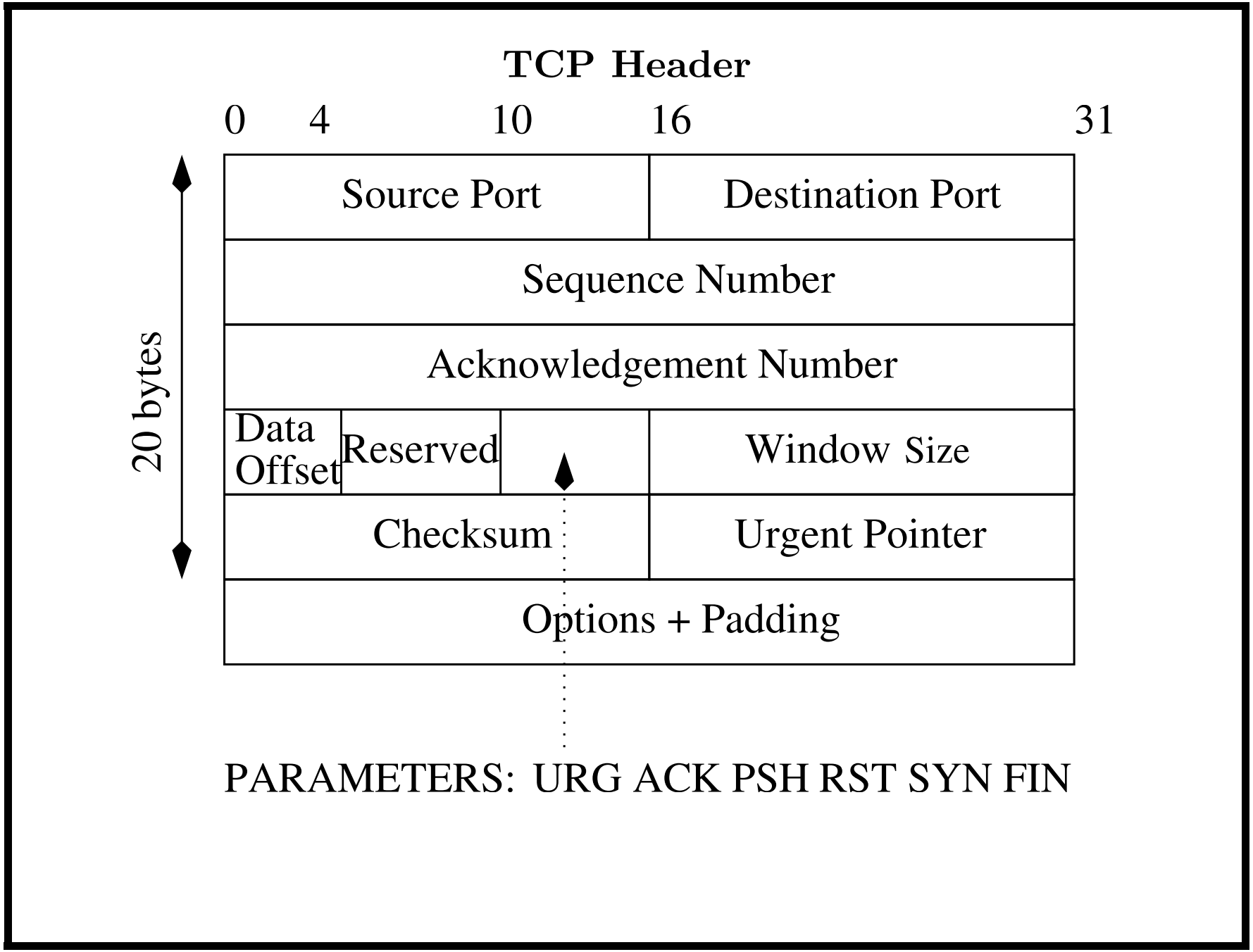


TCP Segment Transmission

- TCP is a byte oriented protocol: sender writes bytes into a TCP connection and receiver reads them out of the TCP connection.



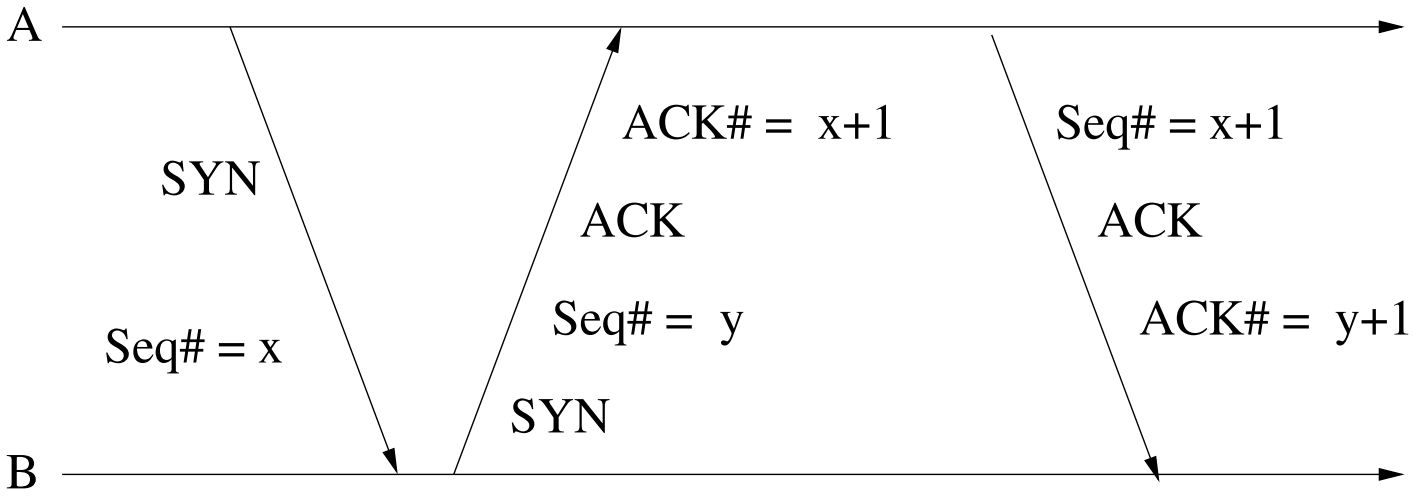
- TCP maintains a variable MSS (Max Segment Size). It decides to send when it has collected "enough" (= MSS) bytes and/or sending process explicitely requests packets. A timer can also trigger transmissions.


TCP Header
0
4
10
16
31
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Data Offset
Reserved
Window Size
Checksum
Urgent Pointer
Options + Padding
20 bytes
PARAMETERS: URG ACK PSH RST SYN FIN

TCP Connection Establishment

- Three-way handshake:

  1. A sends sends request by setting its SYN bit and registers initial sequence number.



  2. B responds by setting its own SYN bit and sends its own sequence number.

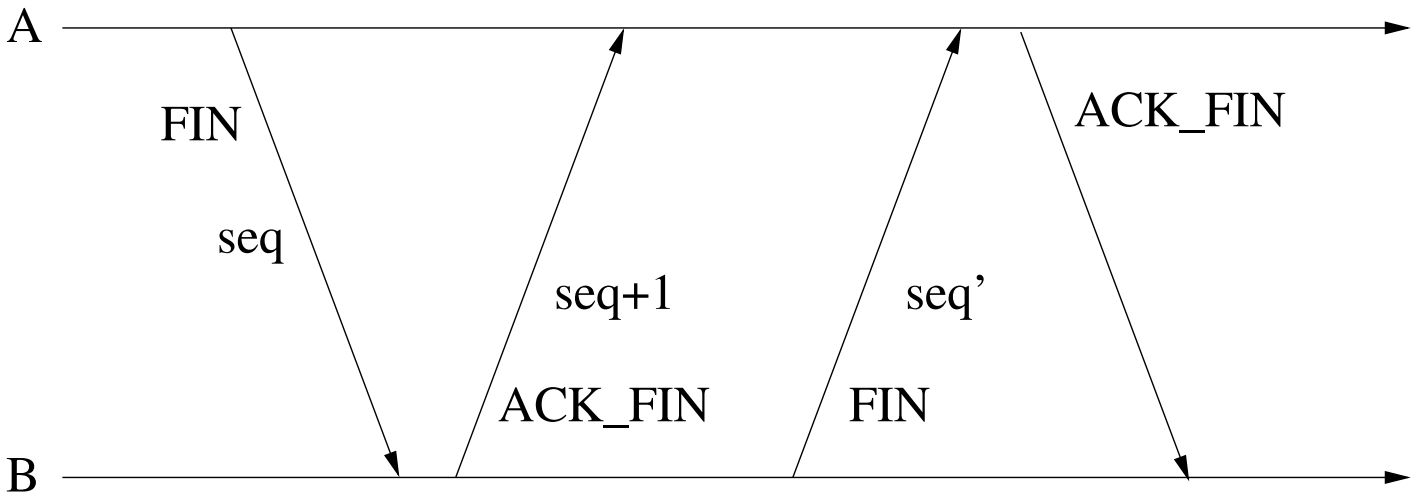
  3. A acknowledges.

Parameters

- A and B want to agree on the connection parameters.
  1. A sends to B the starting segment number it plans to use.
  2. B acknowledges and sends to A its own starting segment number.
  3. A responds by acknowledging B's segment number.
- In addition a timer is scheduled and if expected response is not received the segment is retrasmitted.

Closing a TCP Connection

1. A sets and sends its FIN bit with a sequence number.



2. B acknowledges, sets its FIN bit and sends it to A

3. A acknowledges.

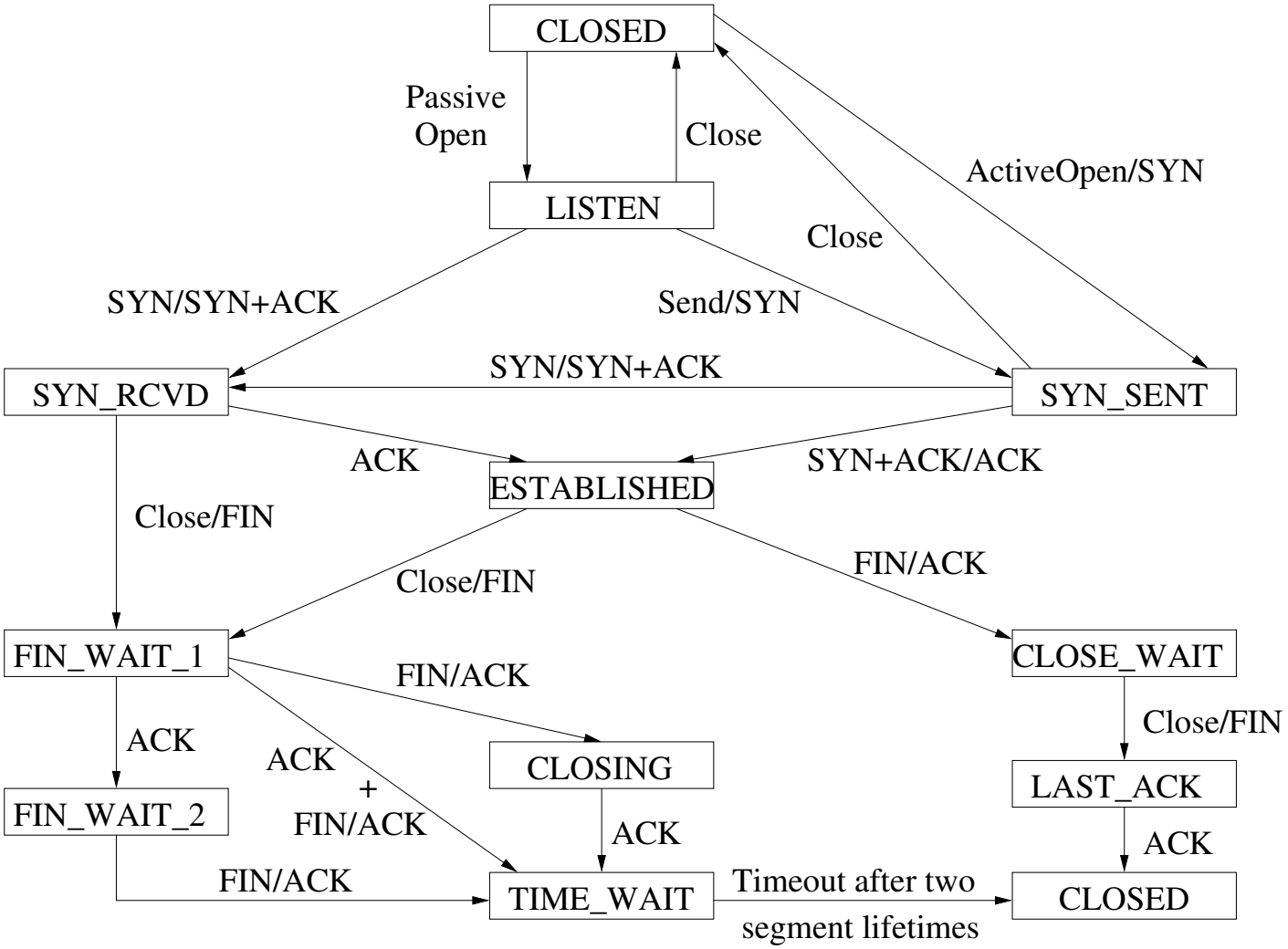
TCP Transition Diagram

- The TCP Transition Diagram defines the semantics of peer-to-peer and service interface.
- The Syntax is given by the TCP headers.
  - All connections start in the CLOSED state.
  - Connections move from state to state through the arcs.
  - Tags on arcs are of the form Event/Action. E.g., SYN/SYN+ACK means when SYN arrives transition is made and a reply ACK+SYN given.
  - Passive open causes TCP to move to LISTEN, followed by an active open and eventually to ESTABLISHED state.

TCP Transition Diagram

- Transitions are triggered when
  1. a segment arrives from a peer.
  2. an operation (e.g. Active Open) on TCP is invoked by local application process.
- Passive open causes TCP to move to LISTEN, followed by an active open and eventually to ESTABLISHED state.

TCP Transition Diagram



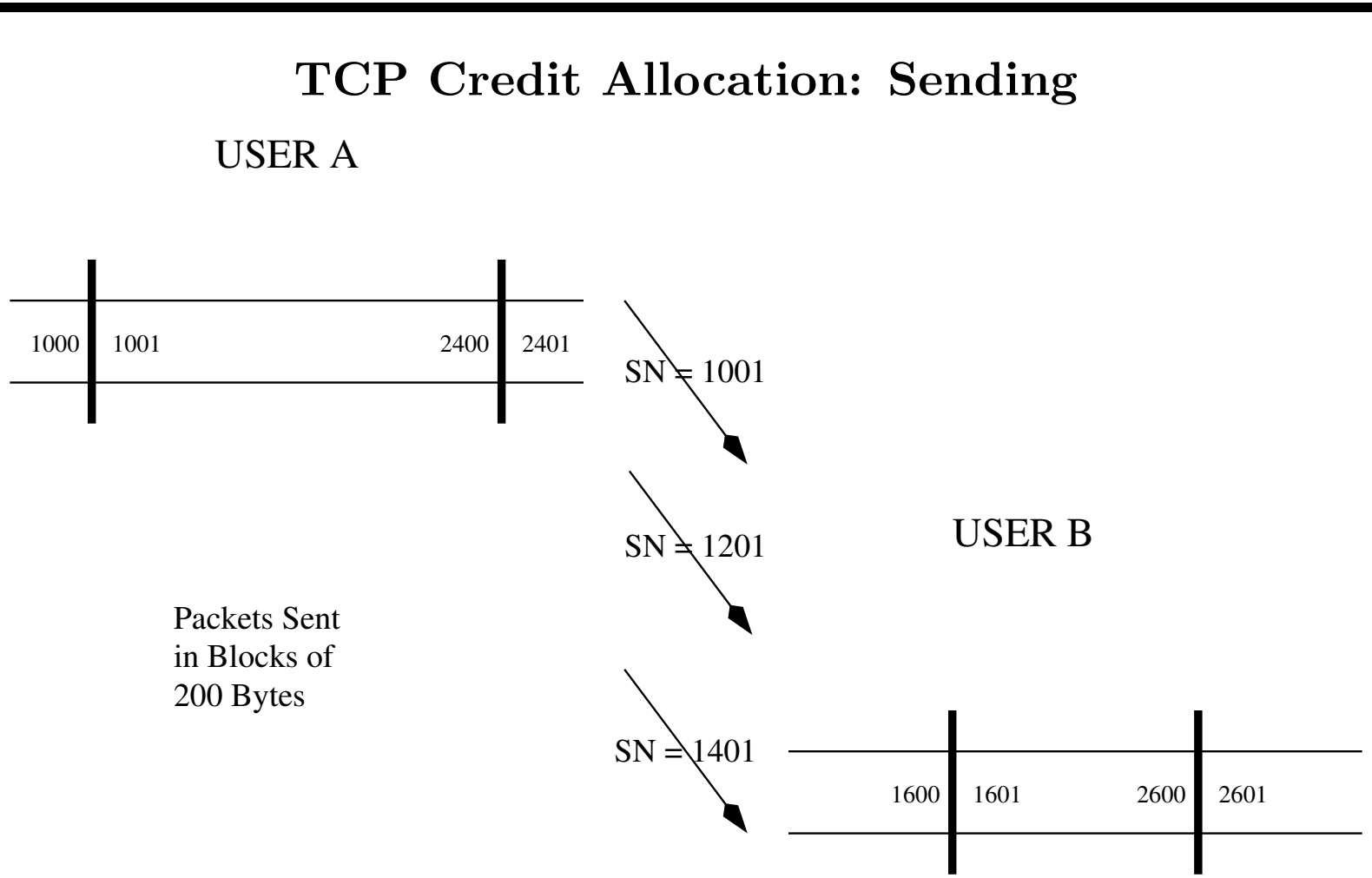
TCP Sequence Numbers

- In a TCP session, the client (on either side) maintains a 32-bit sequence number which it uses to keep track of how much data it has sent.
- This sequence number is included on each transmitted packet, and acknowledged by the opposite host as an acknowledgement number to inform the sending host that the transmitted data was received successfully.
- When a host initiates a TCP session, its initial sequence number may be any value $0 \ldots 2^{32} - 1$. Sequence numbers are relative to this initial sequence number of that stream.

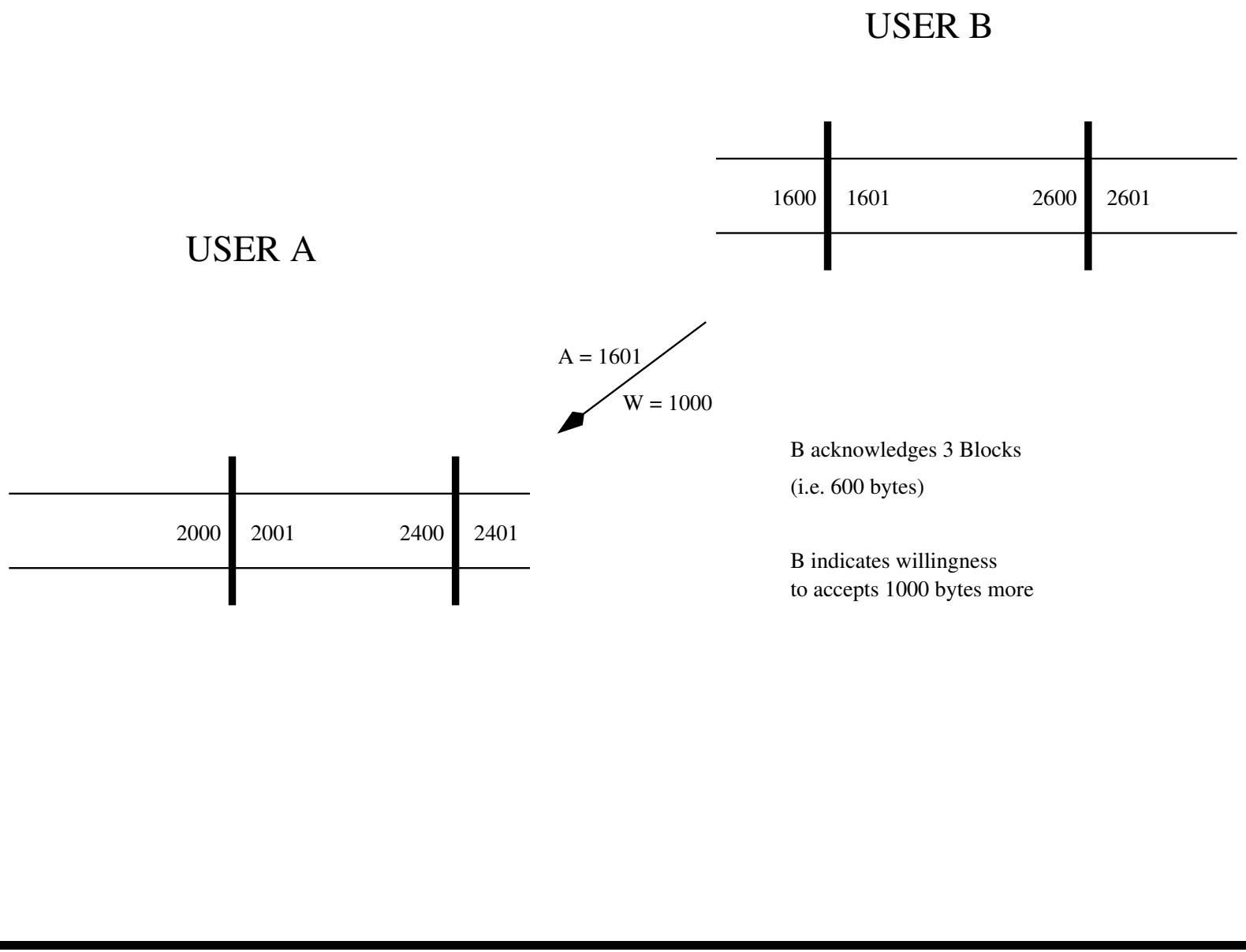
TCP Data Transfer (Sliding Window)

- Each byte transmitted has a sequence number.
- When sending a segment sender includes a sequence number of the first byte in segment data field.
- Receiver acknowledges an incoming segment with a message of the form $(A = i, W = j)$, meaning
  - all bytes through sequence number $i - 1$ are acknowledged, next expected byte has sequence number i.
  - permission is granted to send an additional window W of bytes of data, i.e. the j bytes corresponding to sequence numbers $i..i + j - 1$.
- Protocol known as Credit Allocation Scheme

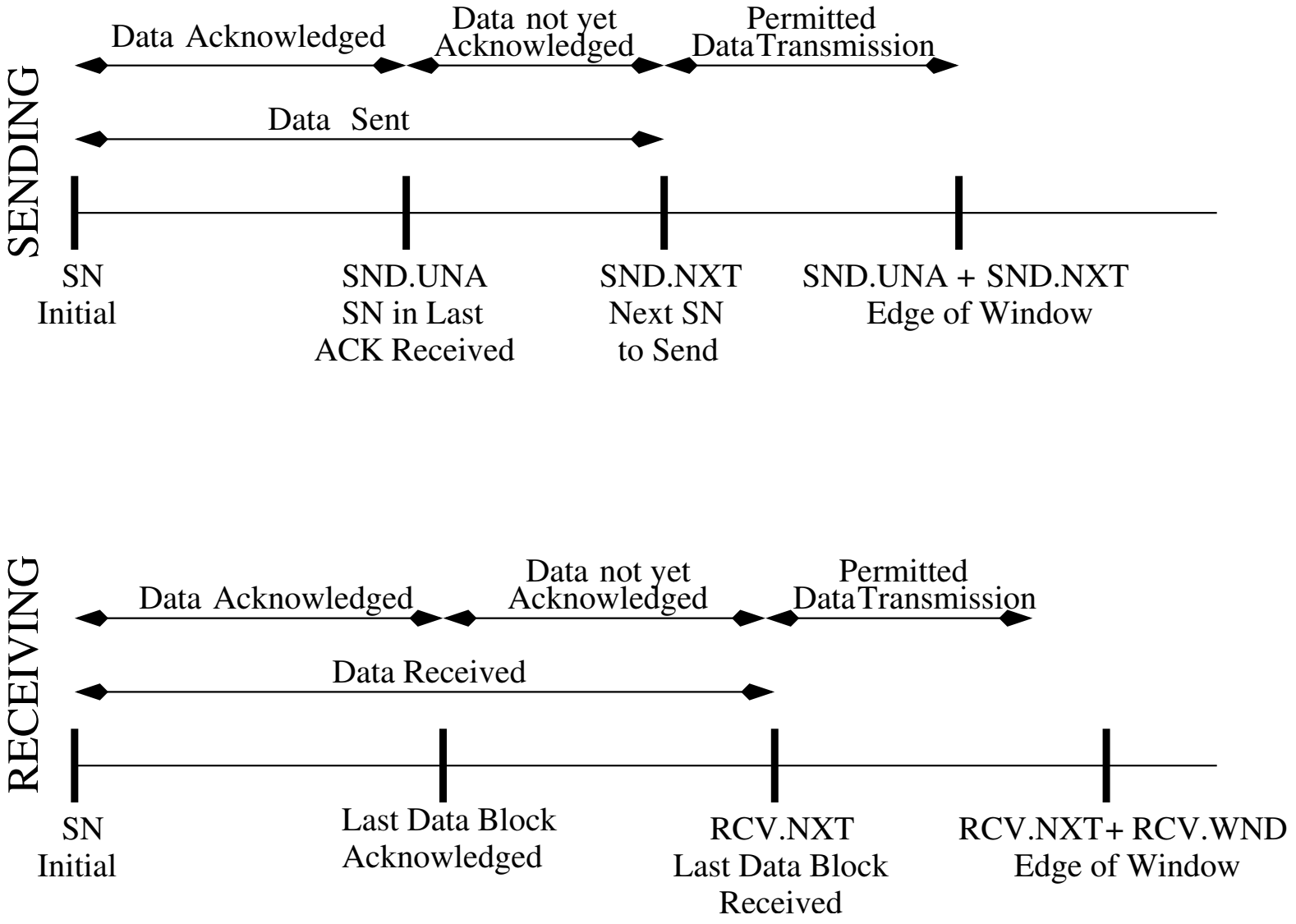
TCP Credit Allocation: Sending



TCP Credit Allocation: Acknowledging



TCP Credit Allocation: Acknowledging



Flexibility of Credit Allocation Scheme

- Assume last message issued by B was $(A = i, W = j)$ and the last byte of data reserved by B was the byte numbered $i - 1$.
  - To increase credit to an amount k $(k > j)$ when no additional data have arrived B issues $(A = i, W = k)$.
  - To acknowledge an incoming segment containing m bytes of data $(m < j)$ without granting additional credit B issues $(A = i + m, W = j - m)$.
  - Receiver is not required to acknowledge data received immediately but can issue cummulative acknowledgement.

TCP Retransmission Strategies

- TCP has no explicit negative acknowledgement. It relies instead on positive ACK.
  - A timer is associated with each segment as it is sent. If timer expires before acknowledgement then it must retransmit.
  - What is the appropriate value of the timer?
    - Too small? We have unnecessary retransmissions!
    - Too large? We have slugginess!
    - Right value? Slightly above round-trip delay!
- Possible strategies: Adaptive and Nonadaptive.

TCP Average Retransmission Timers

- Most TCP implementations attempt to estimate the current round trip delay by observing the pattern of delay for recently transmitted segments.
- How does TCP determines timers?
- It runs a complex algorithm which updates periodically values. For example, define the parameters
  - $RTT(i)$ = round trip time observed for ith transmitted segment.
  - $ARTT(k)$ = average round trip time for first k segments, where

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

- Segment is the unit of transmission used by TCP.

TCP Average Retransmission Timers

- Therefore we have the formula

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

- It might be expensive to compute each time for k, the following recursive formula is useful

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

- Therefore the computation of $ARTT(k+1)$ involves only $ARTT(k)$ and $RTT(k+1)$.

Exponential Averaging

- We can smooth the previous paramater $ARTT(k)$ by using

$$SRTT(k+1) = (1-\alpha)\sum_{i=0}^{k} \alpha^i RTT(k+1-i) + \alpha^{k+1} RTT(0)$$

- From this we get

$$\begin{aligned} & SRTT(k+1) \\ & = (1-\alpha)\sum_{i=0}^{k} \alpha^i RTT(k+1-i) + \alpha^{k+1} RTT(0) \\ & = \alpha \left((1-\alpha)\sum_{i=1}^{k} \alpha^{i-1} RTT(k-(i-1)) + \alpha^k RTT(0) \right) \\ & \quad + (1-\alpha) RTT(k+1) \end{aligned}$$

Exponential Averaging

- $SRTT(k)$ is called smoothed round trip time estimate.
- As shown abobe it is defined by the recursion

$$SRTT(k+1) := \alpha SRTT(k) + (1-\alpha)RTT(k+1) \qquad (2)$$

- By choosing $0 < \alpha < 1$ we observe that the further from k the less weight assigned.
- Equation (2) used by RFC 793 to make TCP estimates.

TCP Congestion Control

- IP is stateless and connectionless and has no provision for detecting congestion.
- TCP provides only end-to-end flow control and can deduce presence of congestion by indirect means.
- In general its knowledge of the network conditions is unreliable.
- There is no distributed control to bind together the various TCP entities.
- The only tool used for control is the sliding window mechanism.

Determining TCP Future Flow Rate

- TCP determines future flow rate by the rate of arrival of incoming ACKs which in turn depends on bottlenecks. Such bottlenecks may be due to
  - the internet
  - receiver
  - sender
- This gives rise to the need for retransmission timer management.
- In the sequel we consider three proposals (Algorithms).

TCP Traffic Variance

- Equation (2) enables TCP to adapt to round trip time changes but it is not always effective:
- TCP-peers vary in performance, traffic conditions may change abruptly, etc.
- To make estimates we try to use the probablistic methodology of expectation, variance and standard deviation, i.e., we can "pretend"

$$X \leftarrow RTT \text{ and } E[X] \leftarrow ARTT$$

- and and make use of the formula for mean deviation[a]

$$MDEV(X) = \sqrt{E\left[|X - E[X]|^2\right]}$$

[a] This comes from the well-known formula for the variance.

TCP Traffic Variance

- Then we estimate the sample standard deviation of RTT as follows

$$\begin{aligned} AERR(k+1) &= RTT(k+1) - ARTT(k) \\ ADEV(k+1) &= \frac{1}{k+1}\sum_{i=1}^{k+1} |AERR(i)|^2 \\ &= \frac{1}{k+1}\sum_{i=1}^{k+1} |RTT(i) - ARTT(i-1)|^2 \end{aligned}$$

with $ARTT(0) = 0$

EXPONENTIAL RTO (Retransmission Timer) Backoff

- When a TCP sender timesout on a segment it must retransmit.
- Since timeout is probably due to network congestion it is best to reuse the RTO value.
- This may cause problems when many TCP connections are active.

EXPONENTIAL RTO (Retransmission Timer) Backoff

- The following strategy is recommended.
  - TCP source increases its RTO value each time the same segment is retransmitted, this is referred to as the backoff process.
  - TCP source will have to wait longer time for a second retransmission.
  - Updating is done according to formula

$$RTO = qRTO$$

    Usually $q = 2$.

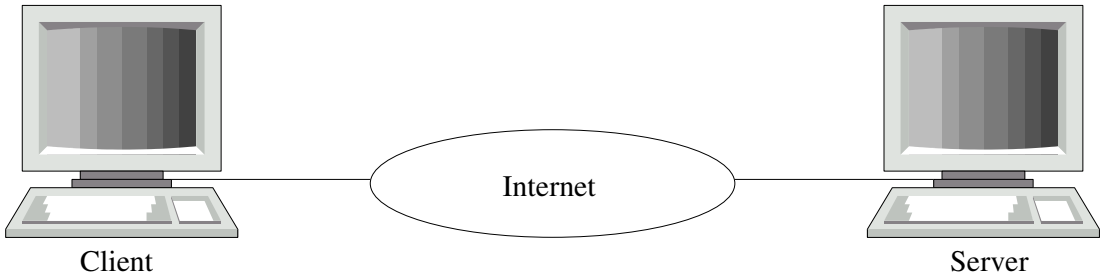
- Same technique used in Ethernet Carrier Sensing protocols.

Karn's Algorithm

- If no segments are retransmitted Jacobson's algorithm works well.
- When segments timeout and need to be retransmitted the RTT values in Jacobson's algorithm are discarded.
- A new suggestion is
  - Do not use the measured RTT for a retransmitted segment to update SRTT and SDEV.
  - Calculate the backoff RTO when a retransmission occurs.
  - Use the backoff RTO value for succeeding segments until an ACK arrives for a segment that has not been retransmitted.
  - When such an acknowledgement is received Jacobson's algorithm is used to compute future RTO values.

TCP and Client/Server

- It is not computets/users that communicate with each other, but rather the end-to-end application programs.



  Each computer must know
  - Its own IP address.
  - Its subnet mask.
  - The IP address of a router.
  - The IP address of a server.

- Stored in a config file, accessed by the bootstrap process (BOOTP).

Equilibria:
A Model for TCP

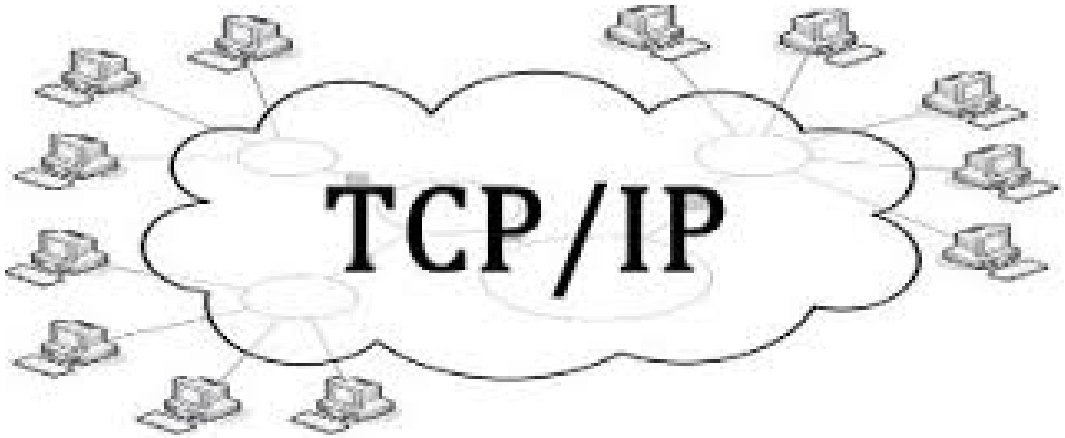
December 6, 2020

A TCP Model

- What's going on with TCP?
- Two things must happen:
  - You measure RTT to learn the behaviour of the traffic.
  - You react appropriately with RTO (timeouts) to improve performance.
- Lets take a high level approach.
- We'll try to build a clear model that illuminates the behaviour of TCP.

Transferring Packets

- When a file is requested over the Internet, the computer that hosts that file breaks it into small packets of data that are then transferred across the network by the transmission control protocol of the Internet, known as TCP.



- The rate at which packets enter the network is controlled by TCP, which is implemented as software on the two computers that are the source and destination of the data.

General Algorithm: "Increase/Decrease"

- The general approach is as follows (Jacobson 1988).
  - When a link within the network becomes overloaded, one or more packets are lost; loss of a packet is taken as an indication of congestion, the destination informs the source, and the source slows down.
  - TCP then gradually increases its sending rate until it again receives an indication of congestion.
- This cycle of increase and decrease enables the source computers to discover and use the available capacity, and to share it between different flows of packets.

Works Very Well!

- TCP has been very successful as the Internet has evolved from a small-scale research network to today's interconnection of hundreds of millions of endpoints and links.
- Each of a large but indeterminate number of flows is controlled by a feedback loop that can know only of that flow's experience of congestion.
- A flow does not know how many other flows are sharing a link on its route, or even how many links are on its route.
- The links vary in capacity by many orders of magnitude, as do the numbers of flows sharing different links.
- It is remarkable that so much has been achieved in such a rapidly growing and heterogeneous network with congestion controlled just at the endpoints.

Why does "Increase/Decrease" Algorithm work so well?

- In recent years researchers have shed some light on TCP's success, by interpreting the protocol as a decentralized distributed algorithm that solves an optimization problem, just as the decentralized choices of drivers in a road network solve an optimization problem.

How TCP Works: ACKs (1/2)

- Packets transferred by TCP across the Internet contain sequence numbers indicating their order, and they should arrive at their destination in that order.
- When a packet is received at the destination, it is acknowledged: an acknowledgment is a short packet sent by the destination back to the source.
- If a packet has been lost in the transfer, the source can tell this from the sequence numbers contained in the acknowledgments.
- The source keeps a copy of each packet sent until it has been positively acknowledged; these copies form what is called a sliding window, and allow packets lost in transfer to be sent again by the source.

How TCP Works: Increase/Decrease (2/2)

- Stored in the source computer there is a numerical variable known as the *congestion window* and denoted by w.
- The congestion window directs the size of the sliding window in the following sense:
  1. if the size of the sliding window is less than w, then the computer increases it by sending out a packet;
  2. if the size of the sliding window is greater than or equal to w, then it waits for positive acknowledgments to come in, which have the effect of reducing the size of the sliding window and, as we shall see, increasing w as well.
- The size of the sliding window continually changes, moving in the direction of a target size that is given by the congestion window.

How Much to Increase/Decrease!

- How much should you increase the congestion window w?
  - By a function $I(w)$ of w, i.e.,

$$w \leftarrow w + I(w)$$

- How much should you decrease the congestion window w?
  - By a function $D(w)$ of w, i.e.,

$$w \leftarrow w - D(w)$$

- The choice of $I(w)$ and $D(w)$ should reflect how aggressive/cautious you want to be!

Basic Rules: Algorithm

- The congestion window itself is not a fixed number:
  - rather, it is constantly being updated, and the precise rules for how this is done are critical for TCP's sharing of capacity.
- The rules currently used are as follows:
  1. Every time a positive acknowledgment comes in, w is increased by $I(w)$, and
  2. Every time a lost packet is detected, w is decreased by $D(w)$.
- Thus, if the source computer detects a lost packet, it realizes that there has been some congestion and backs off for a while, but if all its packets are getting through then it allows the rate at which it sends packets to inch up again.

Using Statistics and Probability

- The way one applies this methodology is by monitoring traffic parameters, like packet loss, and congestion at the nodes.
- For example, we can measure the probability that a packet is lost.
  - We can do this by measuring traffic at the nodes over longer periods of time.
  - Of course, the values we obtain will depend on time.

Algorithm

- Let p be the probability that a packet is lost.
- We execute the following algorithm:
- If the current window size is w then with probability
  1. $1-p$ increase the congestion window by $I(w)$, and
  2. p decrease the congestion window by $D(w)$.
- $I(w), D(w)$ are as yet undefined!

Equilibrium

- The expected change in the congestion window w per update step is therefore

$$E[Change] = I(w)(1-p) + (-D(w))p$$

- The expected change will be positive for small values of w, but will become negative if w is big enough.
- Equilibrium for w might arise when

$$E[Change] = 0$$

- Hence, this expression is zero when

$$I(w)(1-p) = D(w)p \tag{3}$$

- If $I(w), D(w)$ are interesting functions of w we get information on "how window size relates to the packet loss probability p.

Example

- Assume that the packet error rate is p: this can be measured in the TCP protocol. Let

$$I(w) = \frac{1}{w} \text{ and } D(w) = \frac{w}{2}$$

- At equilibrium (see Equation (3)) $I(w)(1-p) = D(w)p$, i.e.,

$$\frac{1}{w}(1-p) = \frac{w}{2}p$$

- So solving for w,

$$w^2 = \frac{2(1-p)}{p}$$

- Therefore

$$w = \left(\frac{2(1-p)}{p}\right)^{1/2} = \sqrt{2}\left(\frac{1}{p} - 1\right)$$

  So w is inversely proportional to p, which is what you expect!

Exercies[a]

1. Would it be reasonable for TCP to set the retransmission strategies between hosts by negotiation prior to a session of transmission of packets?
2. Discuss how TCP makes measurements using Retransmission Strategies so as to determine optimal congestion strategies.
3. What are similarities and differences of exponential backoff algorithms as used in TCP and in Ethernet?
4. Compare the effectiviness of Jacobson's and Karn's algorithms for TCP retransmission control.
5. Here are some choices for $I(w), D(w)$:
   (a) **[Useless:]** $I(w) = Aw$, and $D(w) = Bw$

[a] Do not submit!

(b) **[Aggressive:]** $I(w) = Aw^k$, $k \geq 2$, and $D(w) = Bw$

(c) **[Conservative:]** $I(w) = Aw$, and $D(w) = Bw^k$, $k \geq 2$

(d) **[About Right:]** $I(w) = 1/w$, and $D(w) = w/2$

What equilibrium conditions do they give rise to?

6. Why does TCP use the last algorithm?