Data Transfer

- Consider transferring an enormous file of L bytes from Host A to B using a MSS of 1460 bytes and a 66 byte header.
- What is the maximum value of L such that TCP sequence numbers are not exhausted?
- How long does it take to transmit the file over a 10 Mbps link? Ignore flow control and congestion control so A can pump segments out back to back continuously.

Solution

- You can transfer up to 2^32-2 bytes without wrapping over (we have a 32 bit sequence number, SYN and FIN both count as one “byte” of the sequence number space in TCP)
- Let:
  L = length of file = 2^32-2 bytes
  H = size of header = 1460 bytes
  R = transfer rate = (10*10^6 / 8) bytes per second

  Then:
  Total payload = L
  Total headers = ceil(L/H) * H

  So total time = (L + ceil(L/H)*H) / R

Prefix Aggregation

- Company X and Company Y connect to the same ISP, and they are assigned the prefixes:
  - X: 121.77.80/26
  - Y: 121.77.64/18
- The ISP has a single 3 port router: port 1 connects to the rest of the Internet, port 2 connects to X, and port 3 connects to Y. Write a routing table for this router, filling out as many fields as possible.
- What aggregated prefix can the ISP advertise to the rest of the Internet for companies X and Y?

Solution – Part A

Network Prefix	Subnet Mask	Next Hop	Outgoing Interface
0.0.0.0	0.0.0.0	$IP_{default\ route}$	1
121.77.80.0	255.255.255.192	IP_x	2
121.77.64.0	255.255.192.0	IP_y	3

The first entry is the default route: all IPs will match a 0.0.0.0/0 entry.

121.77.80/26 specifies a range such that IP_{dst} & mask == 121.77.80 & mask, where the mask starts with 26 zeros. So the network prefix is 121.77.80.0 and the netmask in binary is 11111111.11111111.11111111.11000000 = 255.255.255.192

The third entry can be calculated in the same manner.

We don't know the IPs of any of the routers we're connected to, so we just left them as free variables (IP_x, IP_y, etc.)

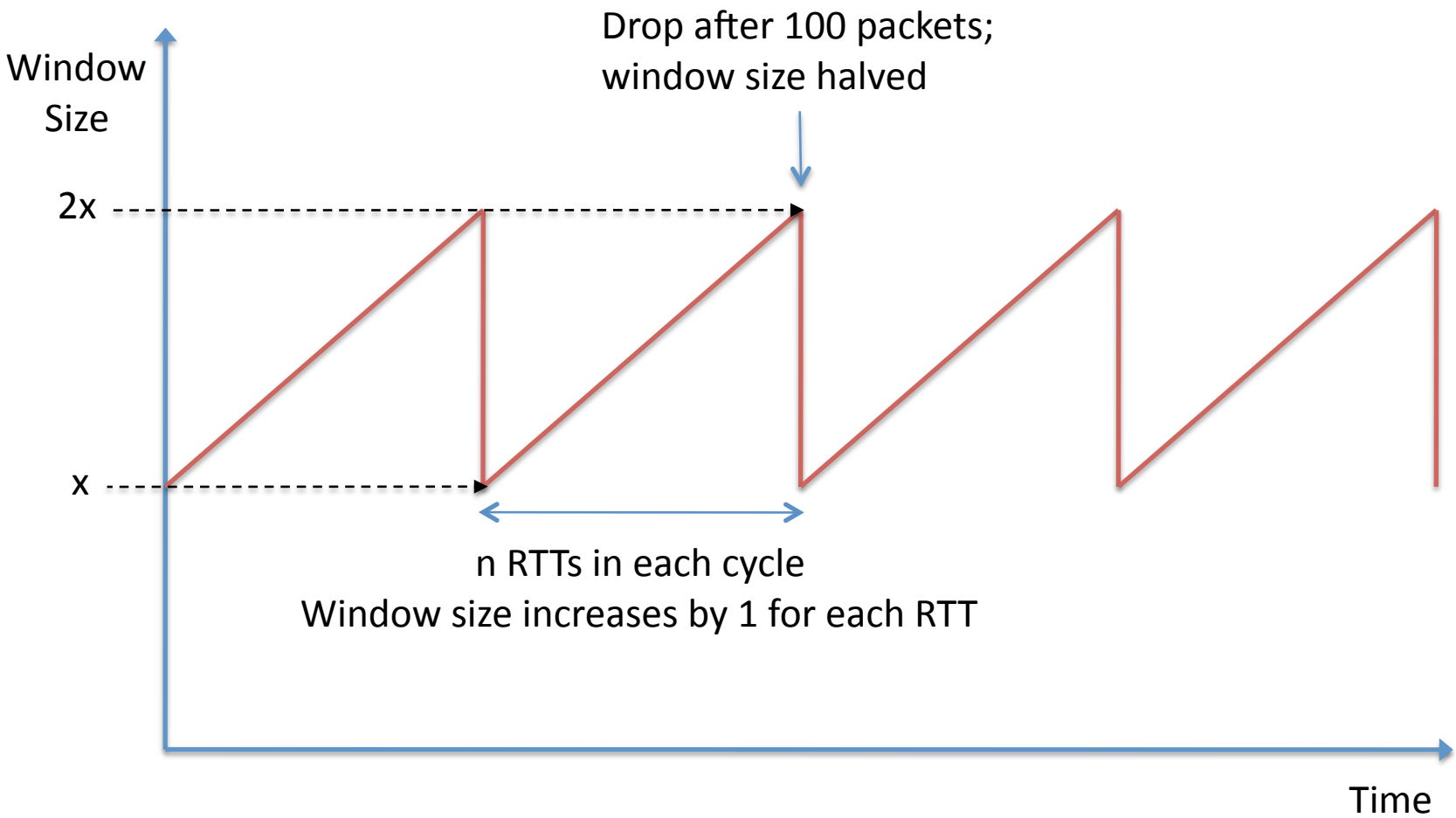
Solution – Part B

- 121.77.80.0/26 = **01111001.01001101.01**010000.00000000
- 121.77.64.0/18 = **01111001.01001101.01**010000.00000000
- Aggregate prefix is the bolded section, as it fully encloses both ranges but doesn't include any additional addresses: **01111001.01001101.01**010000.00000000 = 121.77.64.0/18
- Note: You can't always get an aggregated prefix between two address ranges (for example, if there is a hole between the regions)

Congestion Control

- Suppose a hardware error causes one out of every 100 packets to drop (99 packets sent correctly, then one drops, then 99 packets sent correctly, then one drops…). Assume we attempt to use TCP with simple AIMD over this link (W halved at a loss event, otherwise W++ for every RTT)
- How many RTTs can we expect between successive packet losses?
- With 1000 byte TCP segments (including header) over a 1 Gbps link with 100 ms RTT, what is the maximum throughput of the connection?

Illustration of Steady-State Behavior



Solution

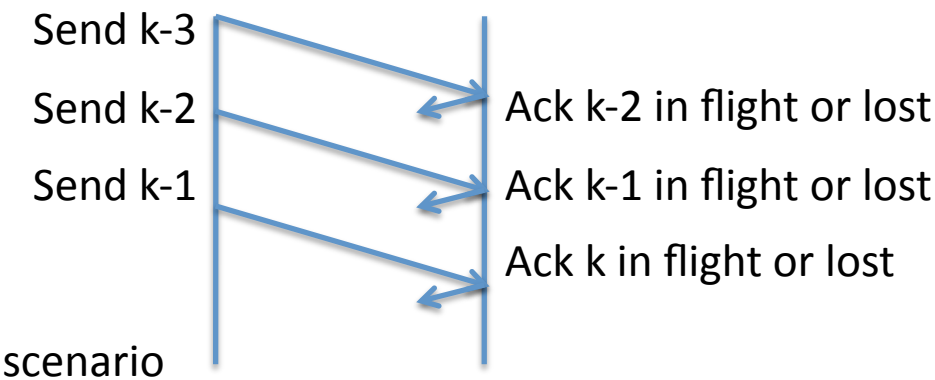
- For every cycle in the sawtooth, 100 segments are sent.
- For each RTT, the # segments sent is the window size, which starts at x. The window size increases by 1 for each RTT. After n RTTs, we have:
  $x + (x+1) + (x+2) + (x+(n-1)) \ldots = nx + (n*(n-1))/2 = 100$
- Also, when we get a loss event after n RTTs, the window size will have doubled:
  Final window size $= x + (n-1) = 2x$
- Solving the simultaneous equation for n and x, we get:
  # RTTs $= n = 8.68 \approx 9$ RTTs
  Initial window size $= x = 7.68 \approx 8$ segments

Sequence Numbers (3-P18)

- Consider a GBN protocol with a window size of 3 and sequence number range of 1024. Suppose at time t, the next in-order packet that the receiver is expecting has a sequence number k. Also assume the network does not re-order messages.
- What are the possible sets of sequence numbers inside the sender's window at time t?
- What are all possible values of the ACK field in all possible messages currently propagating back to the sender that could affect the sender window?

Solution (Part A)

- Sender window = max # of unack'd packets outstanding
- Ack # = next sequence # expected (receiver has received contiguous bytes up to ack – 1)
- In this problem, ack = k
- Worst case: receiver is just sending "ack k" ("ack k-1" and "ack k-2" are lost or in flight). Then sender's window is full and it has just sent us [k-3, k-2, k-1]. See the diagram below.
- Best case: sender has just received "ack k", and therefore starts to send send off [k, k+1, k+2]. Draw out the diagram to convince yourself.
- If you vary the sequence of dropped acks, you will see that the sender can have this set of sequence numbers in its window:
  - [k-3, k-2, k-1]
  - [k-2, k-1, k]
  - [k-1, k, k+1]
  - [k, k+1, k+2]



Worst case scenario

Solution (Part B)

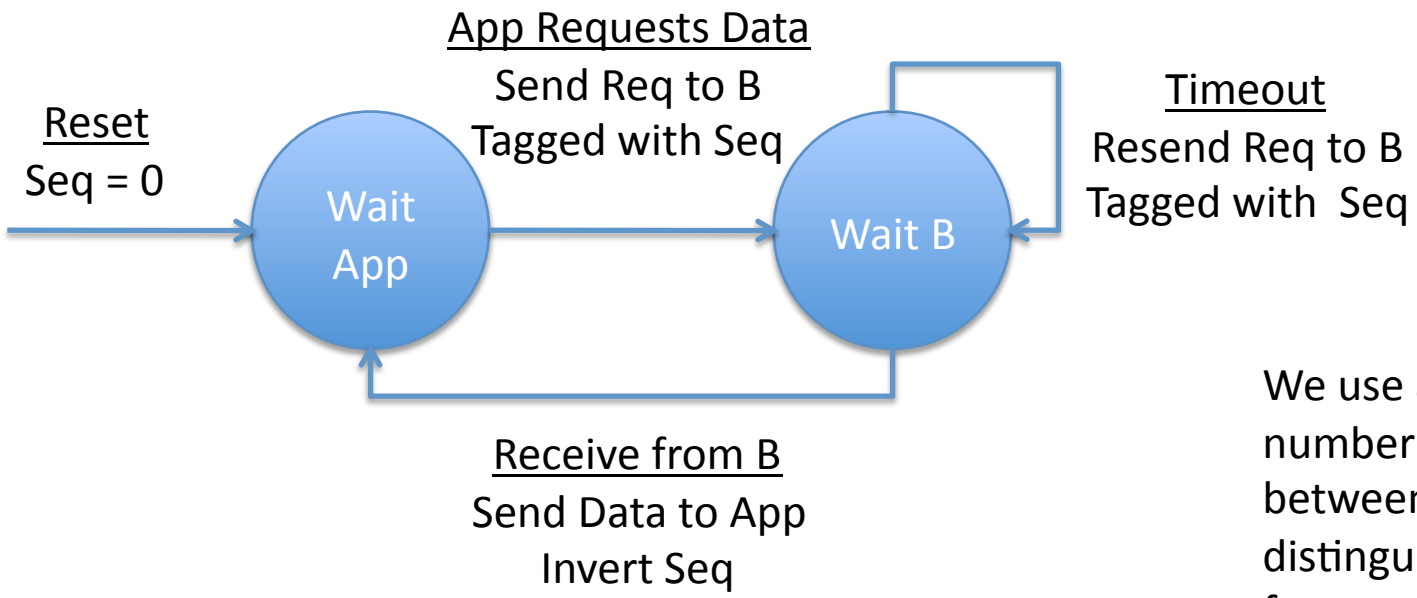
- If the receiver is expecting to receive k next, then the largest ack that can be in the network can be at most k by definition.
- However, from the diagram on the last page, clearly acks of k-1 and k-2 can still be in flight, due to the sender window of size 3.
- Acks of seq #s < k-2 are possible in the network, but won't affect the sender window, since it has already advanced to at least k-2.
- Therefore, there can be ack messages acking the seq #s [k-2, k-1, and k] in the network which will affect the sender window.

Reliable Transfer (3-P19)

- Suppose we have two network entities, A and B. B has a supply of data messages that will be sent to A according to the following conventions:
- When A gets a request from the app to get the next data message from B, it sends a request message to B
- B then responds with the data message
- A then delivers exactly one copy of this message to the client
- Link from A to B is lossy, but does not corrupt or duplicate packets
- Link from B to A is perfectly reliable
- Give an FSM description of a protocol that incorporates only the needed mechanisms to guarantee reliable, in-order message passing to the app given these constraints.

Solution

Sender



We use a single sequence number that alternates between 0 and 1 to distinguish retransmissions from new requests.

Receiver