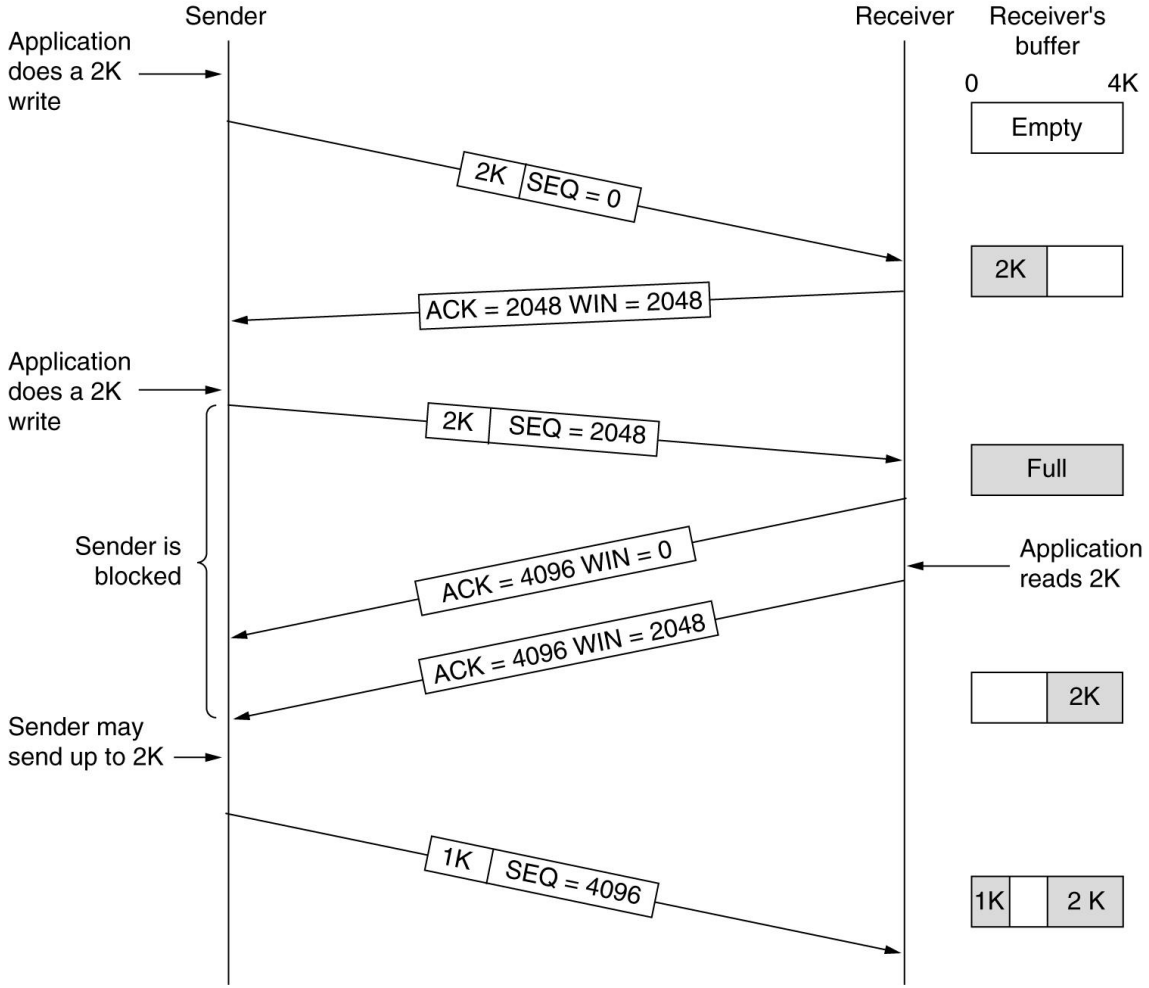
Transmission Policy

TCP transmission policy



- Window management in TCP, starting with the client having a 4096 bytes buffer

TCP transmission policy

- When window size is 0 the sender can't send segments with two exceptions
  - Urgent data may be sent (i.e. to allow the user to kill the process running on the remote machine)
  - The sender may send 1 byte segment to make the receiver re-announce the next byte expected and window size
- Senders are not required to send data as soon as they get it from the application layer;
  - i.e. when the first 2KB of data came in from the application, TCP may have decided to buffer it until the next 2KB of data would have arrived, and send at once a 4KB segment (knowing that the receiver can accept 4KB buffer)
  - This leaves space for improvements
- Receivers are not required to send acknowledgements as soon as possible

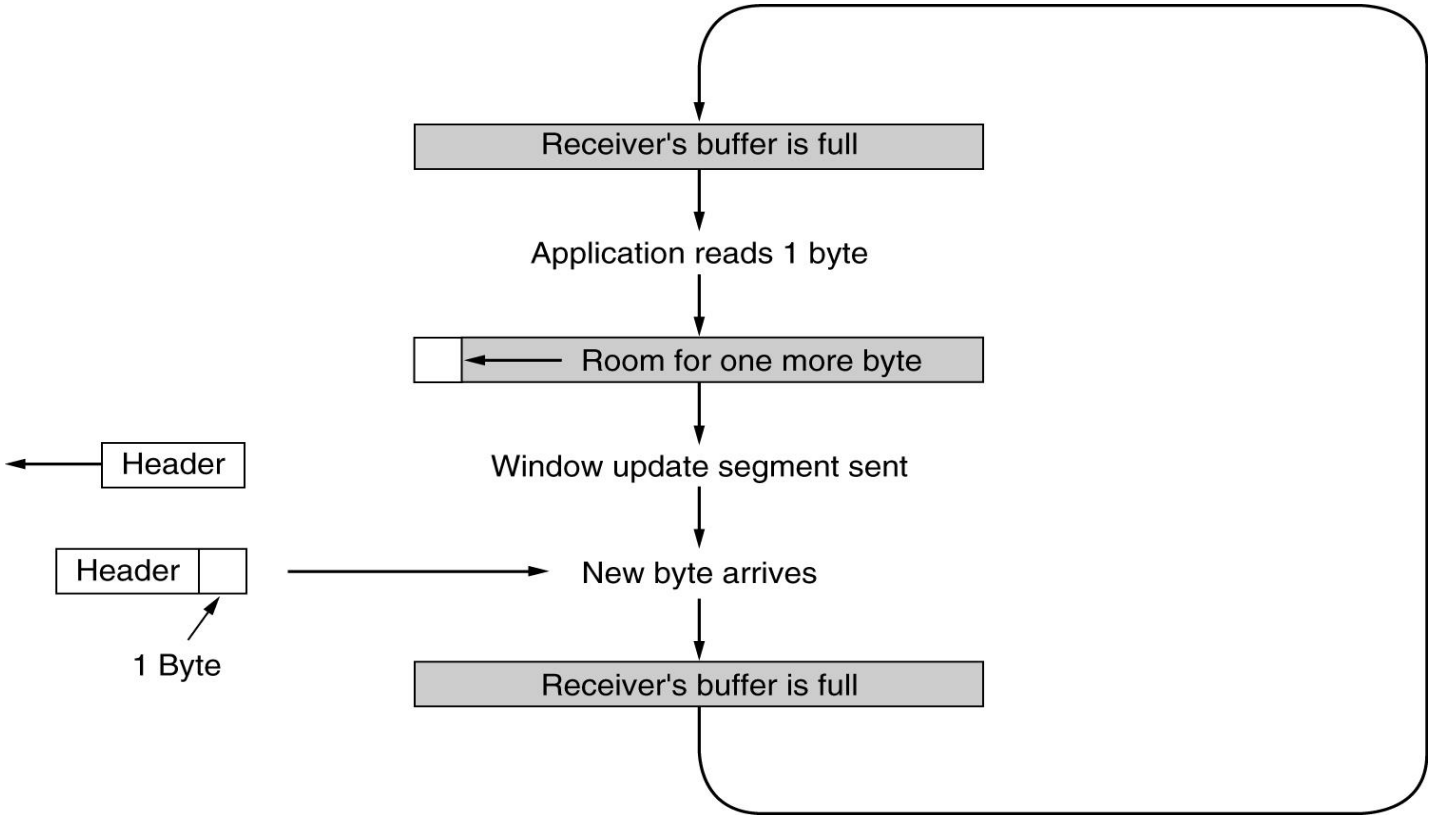
TCP optimizations – delayed ACK

- One solution that many TCP implementation use to optimize this situation is to delay acknowledgements and window updates for 500ms
  - The idea is the hope to acquire some data that will be bundled in the ACK or window update segment
  - This solution deals with the problem at the receiver end, it doesn't solve the inefficiency at the sending end

TCP optimizations – Nagle's algorithm

- Operation:
  - When data come into the sender TCP one byte at a time, just send the first byte as a single TCP segment and buffer all the subsequent ones until the first byte is acknowledged
  - Then send all the buffered characters in one TCP segment and start buffering again until they are all acknowledged
  - The algorithm additionally allows a new segment to be sent if enough data has accumulated to fill half the window or a new maximum segment
- If the user is typing quickly and the network is slow, then a substantial number of characters may go in each segment, greatly reducing the usage of the bandwidth
- Nagle's algorithm is widely used in TCP implementations; there are some times when it is better to disable it:
  - i.e. when an X-Window is run over internet, mouse movements have to be sent to remote computer; gathering them and send them in bursts, make the cursor move erratically at the other end.

TCP performance issues (2)



- Silly window syndrome (Clark, 1982)
  - Data is passed to the sending TCP entity in large blocks
  - Data is read at the receiving side in small chucks (1 byte)

TCP optimizations – Clark's solution

- Clark's solution:
  - Prevent the receiver from sending a window update for 1 byte
  - Instead have the receiver to advertise a decent amount of space available; specifically, the receiver should not send a window update unless it has space to handle the maximum segment size (that has been advertised when the connection was established) or its receiving buffer its half empty, which ever is smaller
  - Furthermore, the sender can help by not sending small segments; instead it should wait until it has accumulated enough space in the window to send a full segment or at least one containing half of the receiver's buffer size (which can be estimated from the pattern of window updates it has received in the past)

Nagle's algorithm vs. Clark's solution

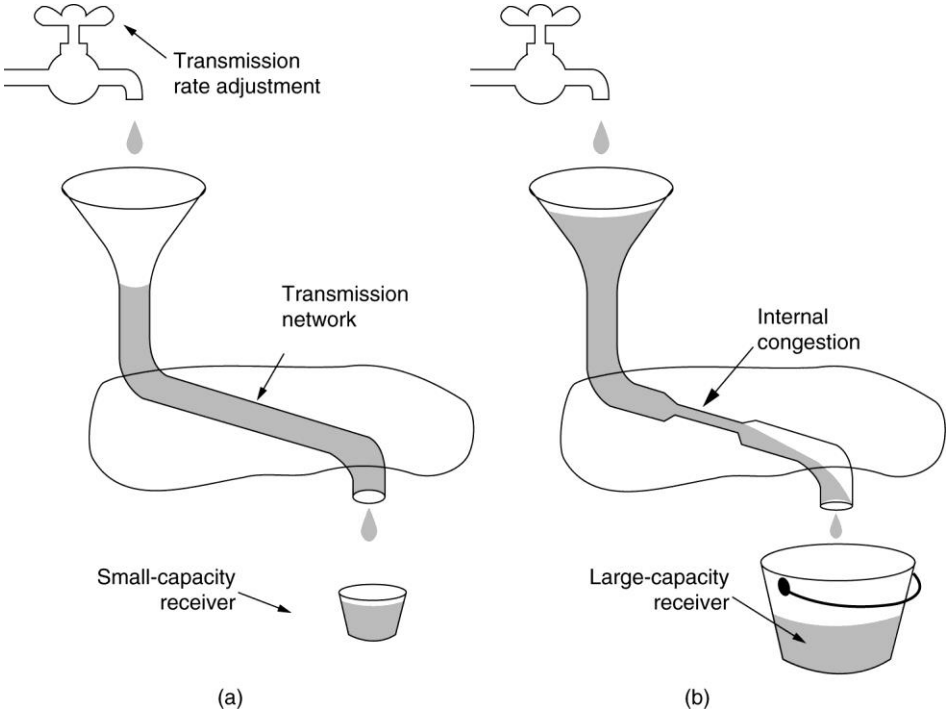
- Clark's solution to the silly window syndrome and Nagle's algorithm are complementary
  - Nagle was trying to solve the problem caused by the sending application deliver data to TCP one byte at a time
  - Clark was trying to solve the problem caused by the receiving application reading data from TCP one byte at a time
  - Both solutions are valid and can work together. The goal is for the sender not to send small segments and the receiver not to ask for them
- The receiving TCP can also improve performance by blocking a READ request from the application until it has a large chunk of data to provide:
  - However, this can increase the response time.
  - But, for non-interactive applications (e.g. file transfer) efficiency may outweigh the response time to individual requests.

Congestion Control

TCP congestion control

- TCP deals with congestion by dynamically manipulating the window size
- First step in managing the congestion is to detect it
  - —A timeout caused by a lost packet can be caused by
    - Noise on the transmission line (not really an issue for modern infrastructure)
    - Packet discard at a congested router
  - —Most transmission timeouts are due congestion
- All the Internet TCP algorithms assume that timeouts are due to congestion and monitor timeouts to detect congestion

TCP congestion control



- Two types of problems can occur:
  - Network capacity
  - Receiver capacity
- When the load offered to a network is more than it can handle, congestion builds up
  - (a) A fast network feeding a low capacity receiver.
  - (b) A slow network feeding a high-capacity receiver.

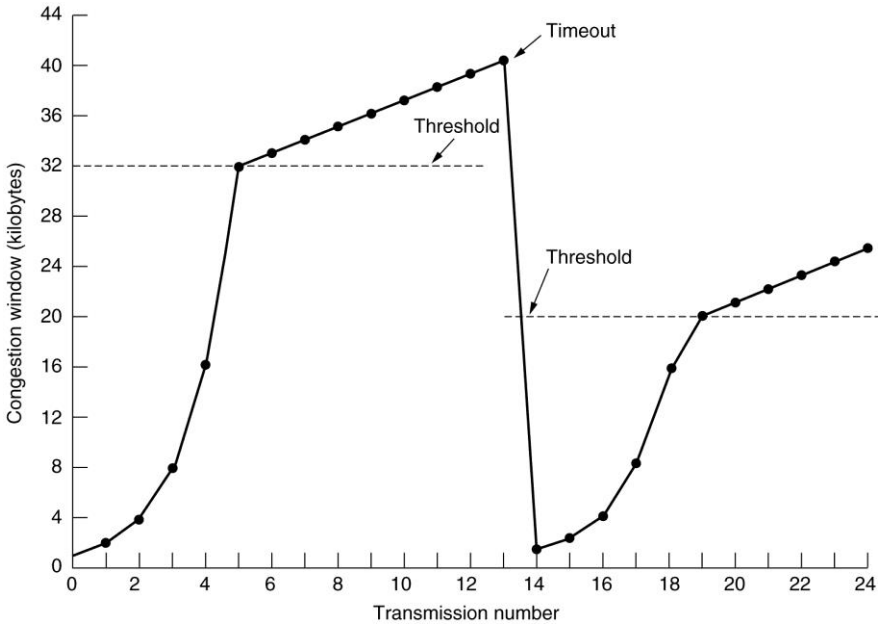
TCP congestion control

- TCP deals with network capacity congestion and receiver capacity congestion separately; the sender maintains two windows
  - The window that the receiver has guaranteed
  - The *congestion window*
- Both of the windows reflect the number of bytes that the sender may transmit; the number that can be transmitted is the minimum of the two windows
  - If the receiver says "send 8K" but the sender knows that more than 4K will congest the network, it sends 4K
  - On the other hand, if the receiver says "send 8K" and the sender knows that the network can handle 32K, then it sends the full 8K
  - Therefore the effective window is the minimum between what the sender thinks is all right and the receiver thinks is all right

Slow start algorithm (Jacobson, 1988)

- When a connection is established, the sender initializes the congestion window to the size of a the maximum segment in use; it then sends a maximum segment
  - If the segment is acknowledged in time, the sender doubles the size of the congestion window (making it twice the size of a segment) and sends two segments, that have to be acknowledged separately
  - As each of those segments is acknowledged in time, the size of the congestion window is increased by one maximum segment size (in effect, each burst successfully acknowledged doubles the congestion window)
- The congestion window keeps growing until either a timeout occurs or the receiver's window is reached
- The idea is that if bursts of size, say 1024, 2048, 4096 bytes work fine, but burst of 8192 bytes timeouts, congestion window remains at 4096 to avoid congestion; as long as the congestion window remains at 4096, no larger bursts than that will be sent, no matter how much space the receiver grants

Slow start algorithm (Jacobson, 1988)



- Max segment size is 1024 bytes
- Initially the threshold was 64KB, but a timeout occurred and the threshold is set to 32KB and the congestion window to 1024 at transmission time 0

- The internet congestion algorithm uses a third parameter, the **threshold**, initially 64K, in addition to the receiver and congestion windows.
  - When a timeout occurs, the threshold is set to half of the current congestion window, and the congestion window is reset to one maximum segment size.
- Each burst successfully acknowledged doubles the congestion window.
  - It grows exponentially until the threshold value is reached.
  - It then grows linearly until the receivers window value is reached.

TCP timer management

- TCP uses multiple timers to do its work
  - —The most important is the *retransmission timer*
    - When a segment is sent, a retransmission timer is started
    - If the segment is acknowledged before this timer expires, the timer is stopped
    - If the timer goes off before the segment is acknowledged, then the segment gets retransmitted (and the timer restarted)
    - The big question is how long this timer interval should be?
  - —*Keepalive timer* is designed to check for connection integrity
    - When goes off (because a long time of inactivity), causing one side to check if the other side is still there

TCP timer management

- TCP uses multiple timers to do its work
  - *Persistence timer* is designed to prevent deadlock
    - Receiver sends a packet with window size 0
    - Latter, it sends another packet with larger window size, letting the sender know that it can send data, but this segment gets lost
    - Both the receiver and transmitter are waiting for the other
    - Solution: persistence timer on the sender end, that goes off and produces a probe packet to go to the receiver and make it to advertise again its window
  - *TIMED WAIT state timer* used when a connection is closed; it runs for twice the maximum packet lifetime to make sure that when a connection is closed, all packets belonging to this connection have died off.