TCP: TRANSPORT LAYER PROTOCOLS

Introduction

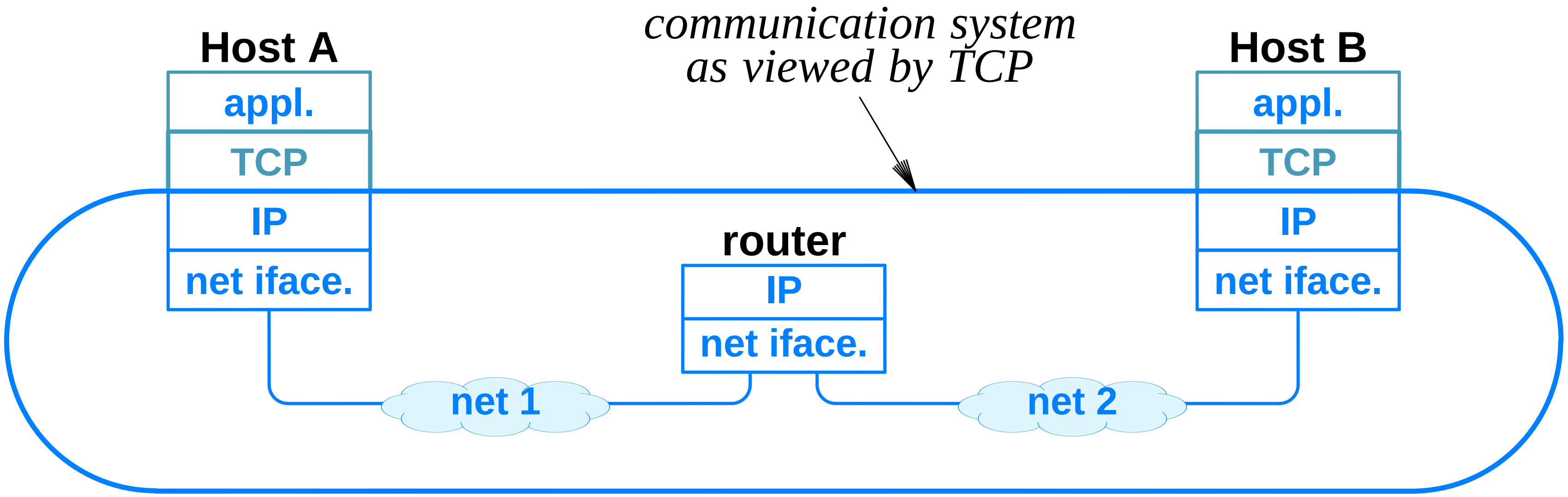
- TCP achieves a seemingly impossible task, this is because it uses the unreliable datagram service which is offered by IP to provide a reliable data delivery service to application programs
- TCP must compensate for delay in an internet, or loss, to provide efficient data transfer, i.e, the data must be delivered in exactly the same order that the data was sent, and there must be no loss or duplication
- TCP must do so without overloading the underlying networks and routers

Service Features Provided by TCP to Applications

- Connection oriented:
  A connection is requested, data is transfered, then the connection is released
- Point-to-point communication:
  TCP connection has exactly two endpoints
- Complete reliability:
  No data missing and no out of order
- Reliable connection startup:
  Three-way handshake
- Graceful connection shutdown:
  All data is delivered reliably before closing the connection
- Full duplex communication:
  Allows data to flow in either directions at any time
- Stream interface:
  Transferring a continuous sequence of bytes across a connection

End-to-End Service

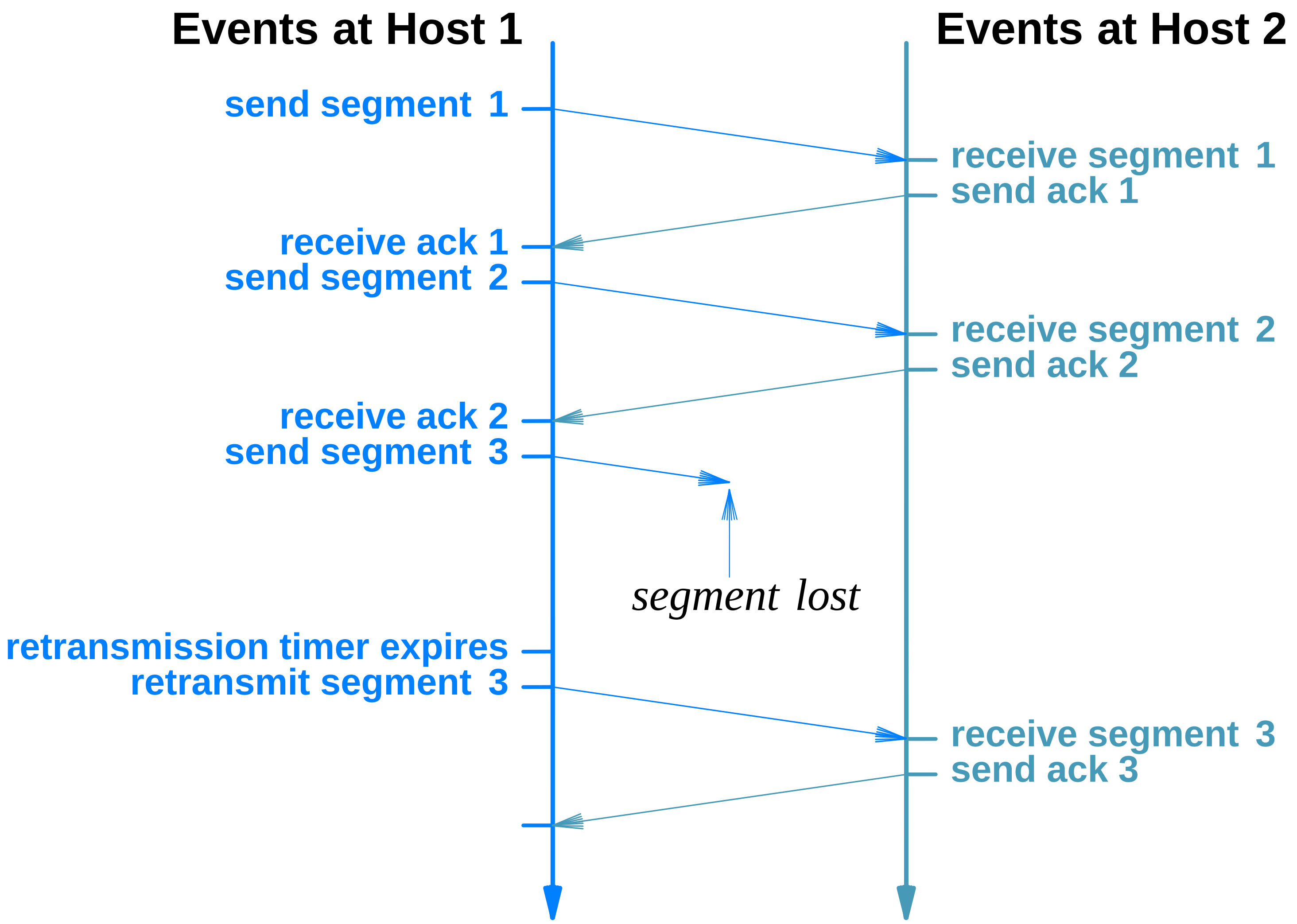
- TCP provides a connection directly from an application on one computer to an application on a remote computer, i.e., *End-to-End Service*
- The connections provided by TCP are called *virtual connections* because they are achieved in software
- TCP software is needed at each end of the virtual connection, but not on intermediate routers



TCP is an end-to-end transport protocol

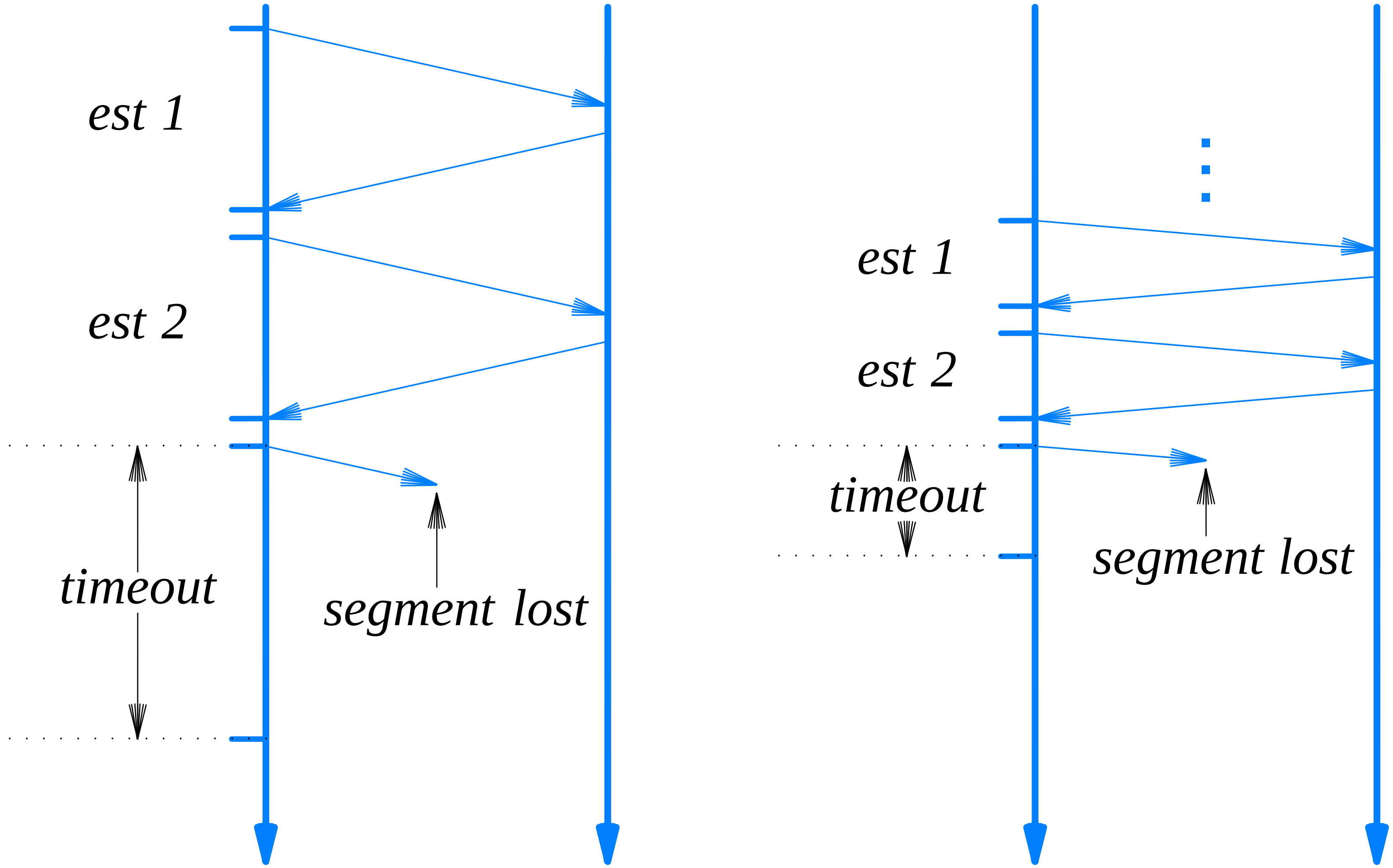
Lost Packets and Retransmission

- Whenever a segment arrives undamaged, the receiving protocol software sends a small segment back to report successful reception
- the sender takes responsibility for ensuring that each packet is transferred successfully
  - Whenever a sender sends a packet, its protocol starts a timer
  - If an acknowledgment arrives before the timer expires, the software protocol cancels the timer
  - If the timer expires before an acknowledgment arrives, the software protocol
    * Sends another copy of the packet
    * Starts timer again


Events at Host 1
Events at Host 2
send segment 1
receive segment 1
send ack 1
receive ack 1
send segment 2
receive segment 2
send ack 2
receive ack 2
send segment 3
segment lost
retransmission timer expires
retransmit segment 3
receive segment 3
send ack 3

Adaptive Retransmission Timeout

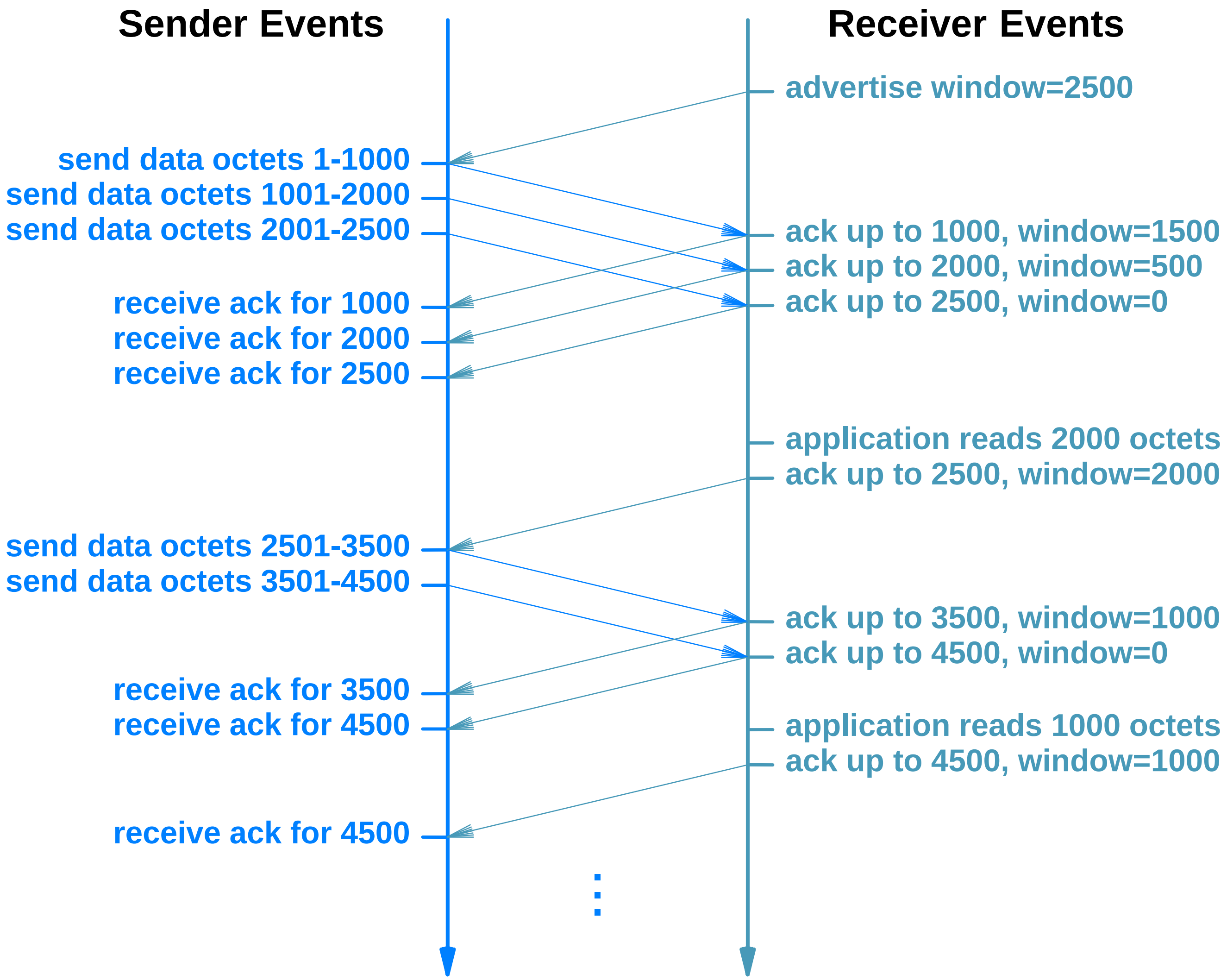
- Time for acknowledgment to arrive depends on
  - Distance to destination
  - Current traffic conditions
- Traffic conditions change rapidly
- Multiple connections can be open simultaneously
- To handle these variety of delays, TCP introduces the following *adaptive retransmission* scheme
  - Keep estimate of round trip times on each connection
  - Use current estimate to set the retransmission timer
    - If the estimate delay is large, TCP uses a large retransmission
    - If the estimate delay is small, TCP uses a small retransmission
- The goal is to wait long enough to determine that a packet was lost, without waiting longer than necessary



Timeout depends on current round-trip estimate

Flow Control (Sliding Window Protocol)

- Receiving side
  - Establishes multiple buffers
  - *Advertises* the sender with the size of these buffers
  - When a packet is successfully arrived in order, the receiving side
    - Sends an acknowledgment signal to the sender
    - Slides the window
- Sending side
  - Transmits packets for all available buffers
  - When getting acknowledgment
    - Slides the window
    - Transmits more packets
- The window tells how many bytes can be sent
- The window moves as acknowledgments arrive
- Each acknowledgment is sent along with the amount of the available buffer space, i.e., the available window size (*window advertisement*)
- When a window advertisement is equal to zero, we call it *closed window*



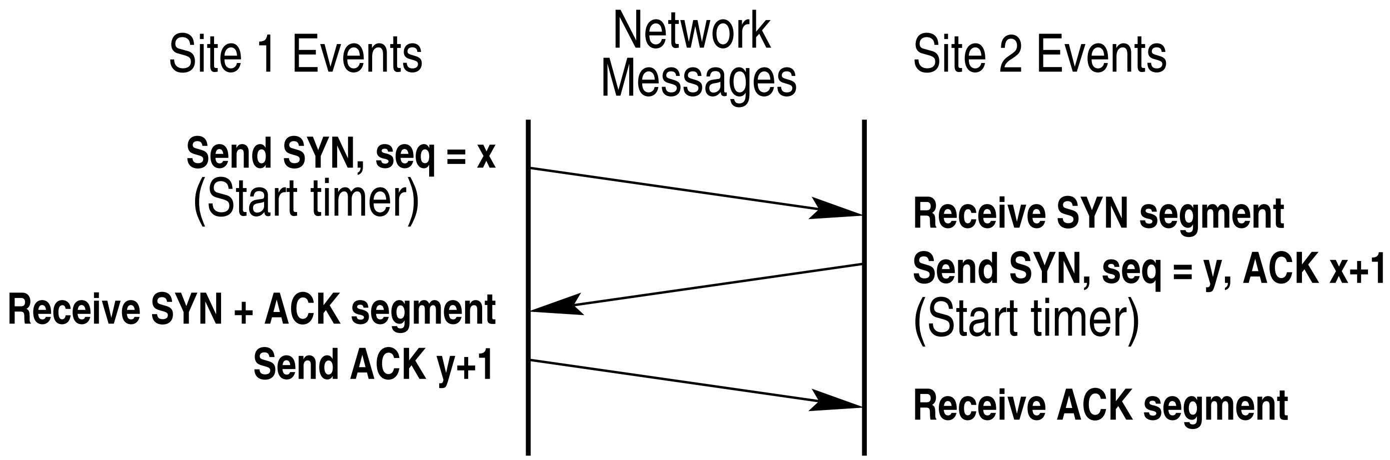
Interpretation: I acknowledge receiving up to byte No. X, and can take Y more bytes

Startup/Shutdown Connection

- Connection startup must be reliable
- Connection shutdown must be graceful
- Startup/shutdown connections are difficult tasks, this is because
  - Segments can be
    - Lost
    - Duplicated
    - Delayed
    - Delivered out of order
    - Either side can crash
    - Either side can reboot
  - Duplicate "*shutdown*" segment must be prevented from affecting later connection
- To guarantee that connections are established properly, TCP uses a *three-way handshake*
- To guarantee that connections are shutdown properly, TCP uses a *three-way handshake*, or a *modified three-way handshake*, in which the middle segment is split into two halves

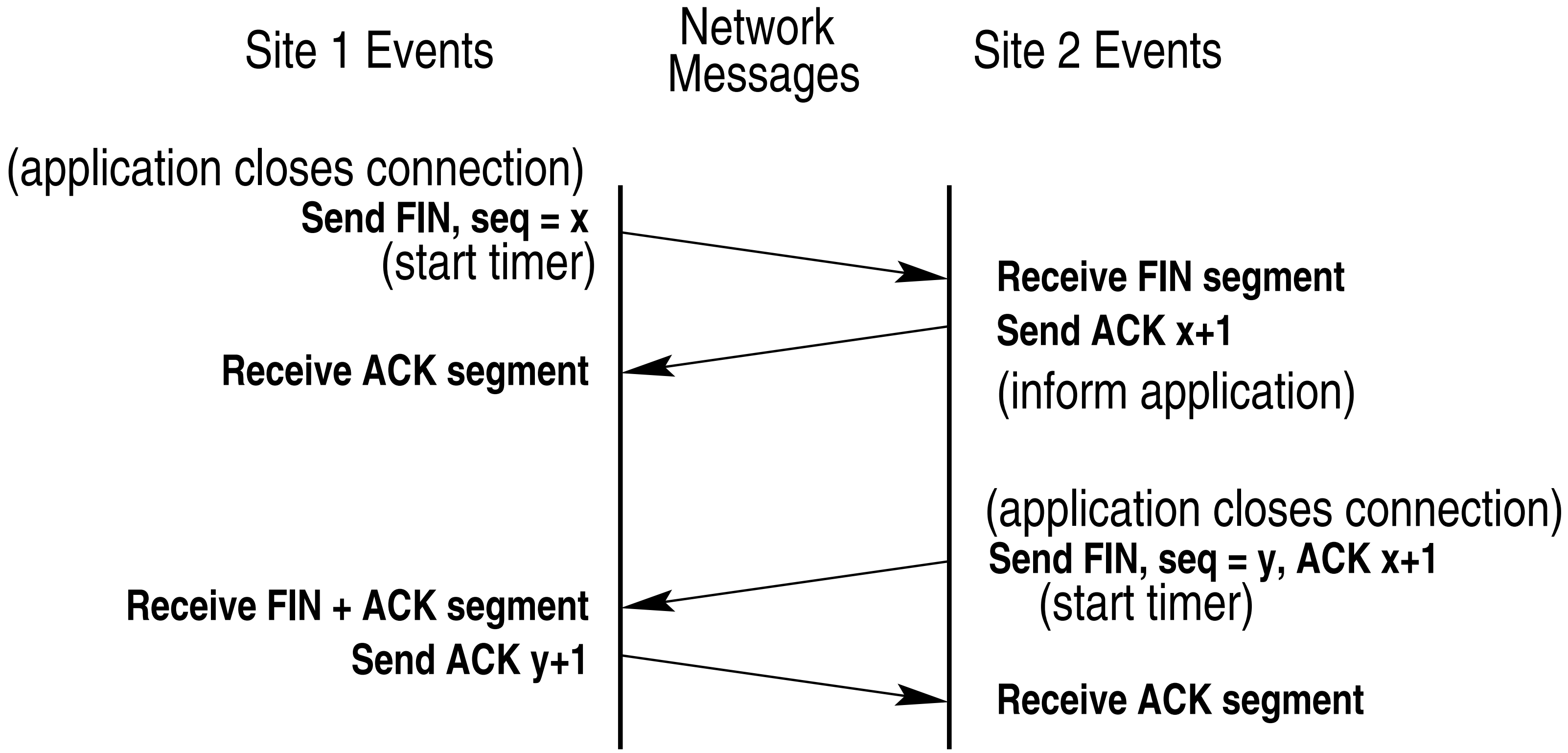
Startup Connection

- One machine initiates a startup connection, by sending a *synchronization* (SYN) segment, as well as an initial *sequence number* (seq = ??)
- The other machine sends another SYN, initializes and sends its *sequence number*, and *acknowledges* (ACK) getting the sent segment
- ACK specifies the next expected sequence number
- The first machine *acknowledges* getting the segment
- Once the connection is established, data can flow in both directions equally well, i.e., there is no master/slave relationship

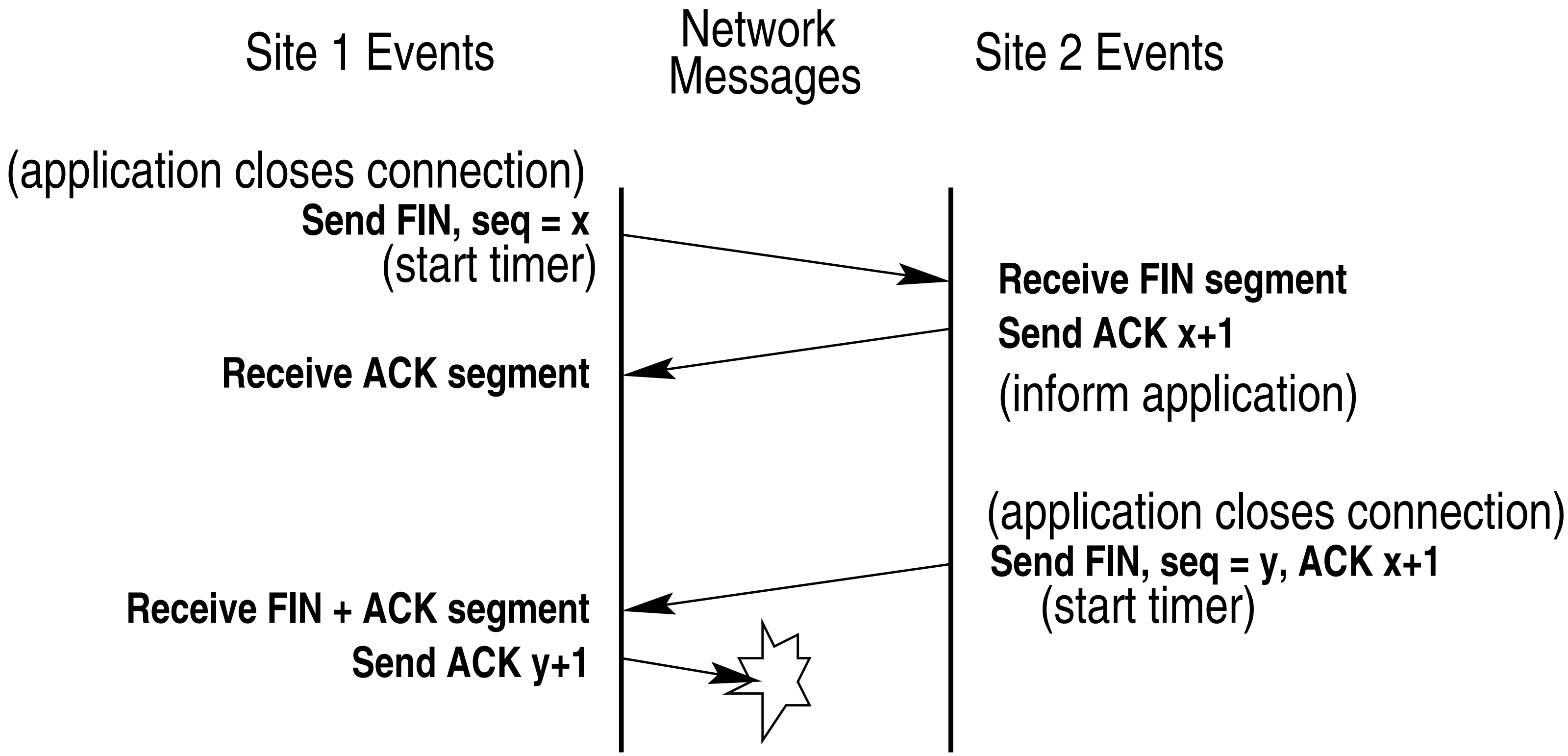


Shutdown Connection

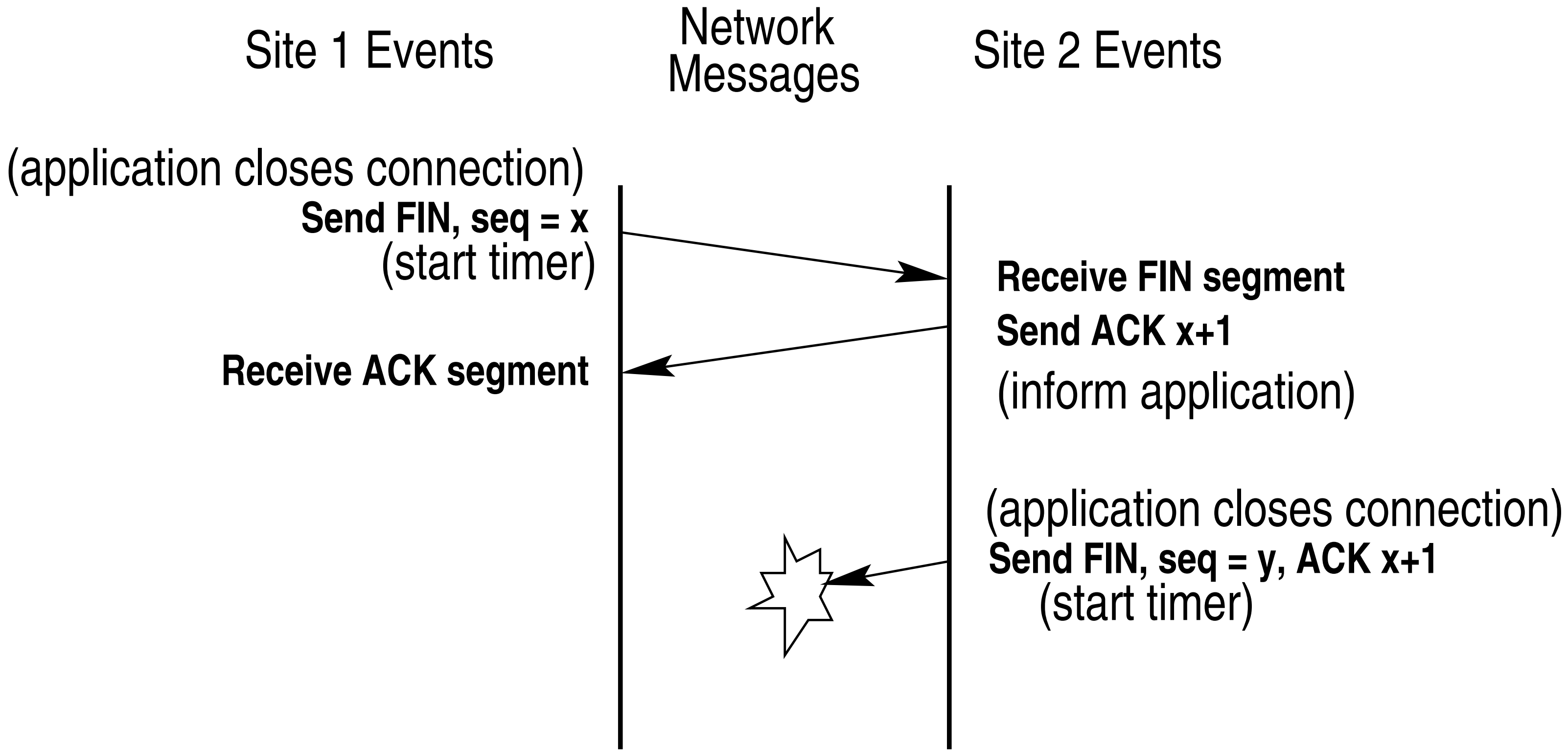
- One machine sends a *Finish* (FIN) segment
- The other machine acknowledges the FIN segment and inform the application program on its end that no more data is available
- Note that TCP connections are full duplex, i.e., two independent stream transfers, one going in each direction
- Once a connection has been closed in a given direction, TCP refuses to accept more data for that direction
- Meanwhile, data can continue to flow in the opposite direction until the sender closes it
- Of course acknowledgments continue to flow back to the sender, even after a connection is closed
- When both directions is closed, the TCP software at each endpoint deletes its record of the connection and releases whatever reserved buffers
- This scheme is a modified three-way handshake, this is because the second machine may send the FIN acknowledgment and its own FIN request separately in two different segments


Site 1 Events
Network Messages
Site 2 Events
(application closes connection)
Send FIN, seq = x
(start timer)
Receive FIN segment
Send ACK x+1
Receive ACK segment
(inform application)
(application closes connection)
Send FIN, seq = y, ACK x+1
(start timer)
Receive FIN + ACK segment
Send ACK y+1
Receive ACK segment

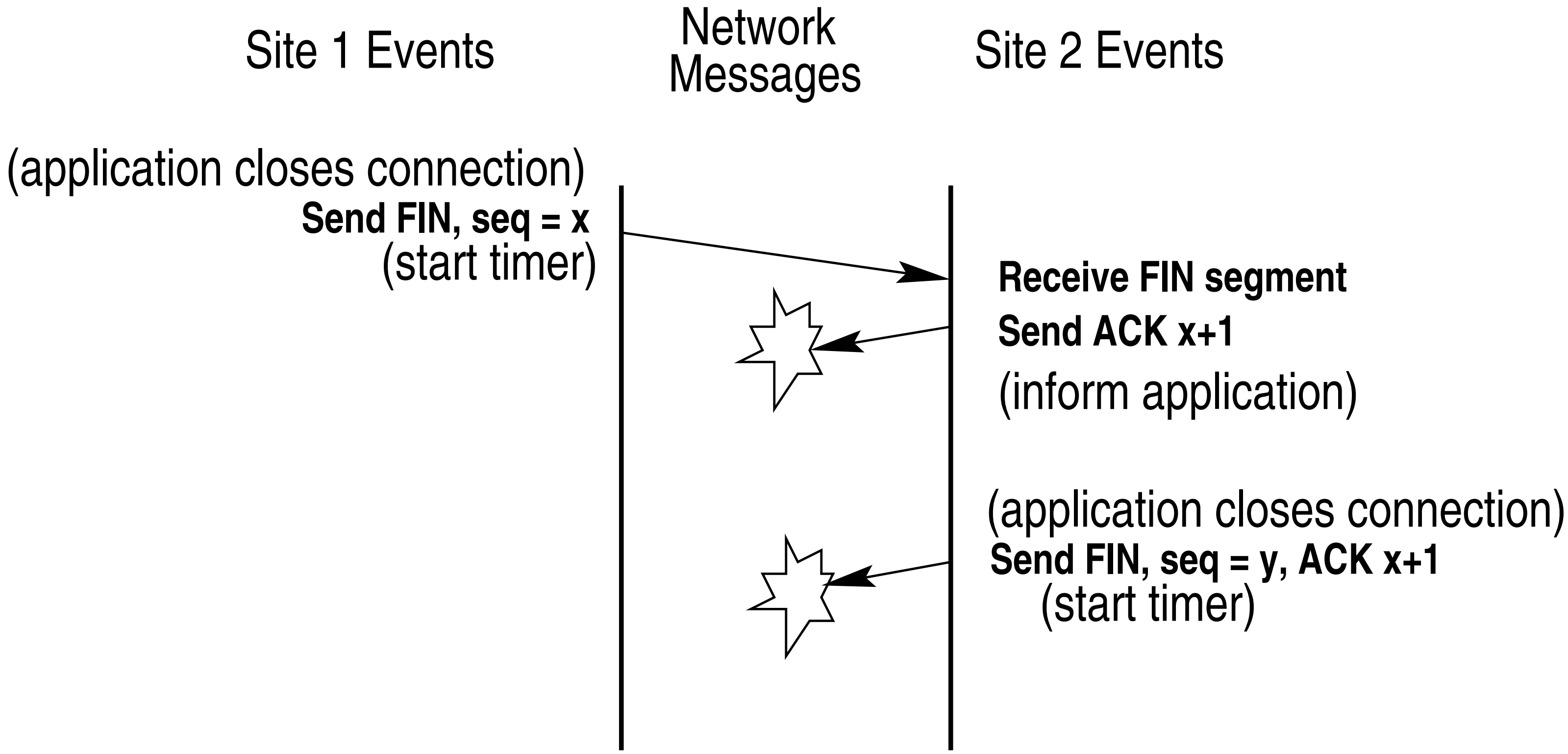
- TCP uses a timer to solve the problem of lost segment



What should the sender/receiver do in this case?



What should the sender/receiver do in this case?



What should the sender/receiver do in this case?

Congestion Control

- When more data arrive than can be sent, the queue grows and the effective delay increases (principle cause of delay)
- If congestion persists, the TCP software may run out of memory and begin discarding packets
- In a sever congestion cases, the entire system may become unusable (congestion collapse)
- To prevent this problem, TCP software begins congestion control whenever a segment is lost
  - Send just a single segment containing data
  - If its acknowledgment arrives, doubles the amount of data being sent, and so on
  - This exponential increase continues until sending half of the receiver's advertised window, at which slows down the rate of increase

The TCP Segment Header Format

<table>
<tr><th>0</th><th>4</th><th>10</th><th>16</th><th>24</th><th>31</th></tr>
<tr><td colspan="3">SOURCE PORT</td><td colspan="3">DESTINATION PORT</td></tr>
<tr><td colspan="6">SEQUENCE NUMBER</td></tr>
<tr><td colspan="6">ACKNOWLEDGEMENT NUMBER</td></tr>
<tr><td>HLEN</td><td>NOT USED</td><td>CODE BITS</td><td colspan="3">WINDOW</td></tr>
<tr><td colspan="3">CHECKSUM</td><td colspan="3">URGENT POINTER</td></tr>
<tr><td colspan="6">OPTIONS (if any)</td></tr>
<tr><td colspan="6">BEGINNING OF DATA
⋮</td></tr>
</table>

- SOURCE PORT: Identifies the application program that sent the data
- DESTINATION PORT: Identifies which application program on the receiving computer should receive the data
- SEQUENCE NUMBER: Identifies the position in the sender's byte stream of the data in the segment
- ACKNOWLEDGMENT NUMBER: Identifies the byte number that the source expects to receive next

- HLEN: Header length (multiple of 32-bit words)
- CODE BITS: Determines the purpose and contents of the segment
  - URG: Urgent pointer field is valid
  - ACK: Acknowledgment field is valid
  - PSH: This segment requests a push
  - RST: Reset the connection
  - SYN: Synchronize sequence number
  - FIN: Sender has reached end of its byte stream
- WINDOW: Advertises how much data the TCP software is willing to accept
- CHECKSUM: A 16-bit integer checksum used to verify the integrity of the data as well as the TCP header
- URGENT POINTER: Indicates a byte offset from the current sequence number at which urgent data are to be ended. URG/URGENT POINTER act as an interrupt message to send data without waiting for the program at the other end of the connection to consume bytes already in the stream. Such signals are most often needed when a program on the remote machine fails to operate correctly.

- Note that:
  - PSH and URG are different; PSH means the sender kindly requests from the receiver to deliver the data to the application upon arrival, and not to be buffered until a full buffer is received, however data still deliver to the application in order, i.e.,
    * PSH means flush the buffer
    * URG means interrupt signal
  - RST is used to
    * Reset a connection that has become confused due to a host crash, or some other reason
    * Refuse an attempt to open a connection
  - In general, if you get a segment with the RST bit on, you have a problem on your hands