Bucknell UNIVERSITY

Transport Protocols

CSCI 363 Computer Networks

Department of Computer Science

Expected Properties

- Guaranteed message delivery
- Message order preservation
- No duplication of messages
- Support for arbitrarily large messages
- Support for sender/receiver synchronization
- Receiver based flow control
- Support multiple application processes per host

Relationship with other layers

Application

What are the expectations of the functionality provided by underlying layers?

"Higher layers"

Add to the functionality provided by Transport

Transport

Provides end-to-end features

Network

May drop, reorder, or duplicate messages. May introduce arbitrarily long delays. May limit the size of messages to a maximum value.

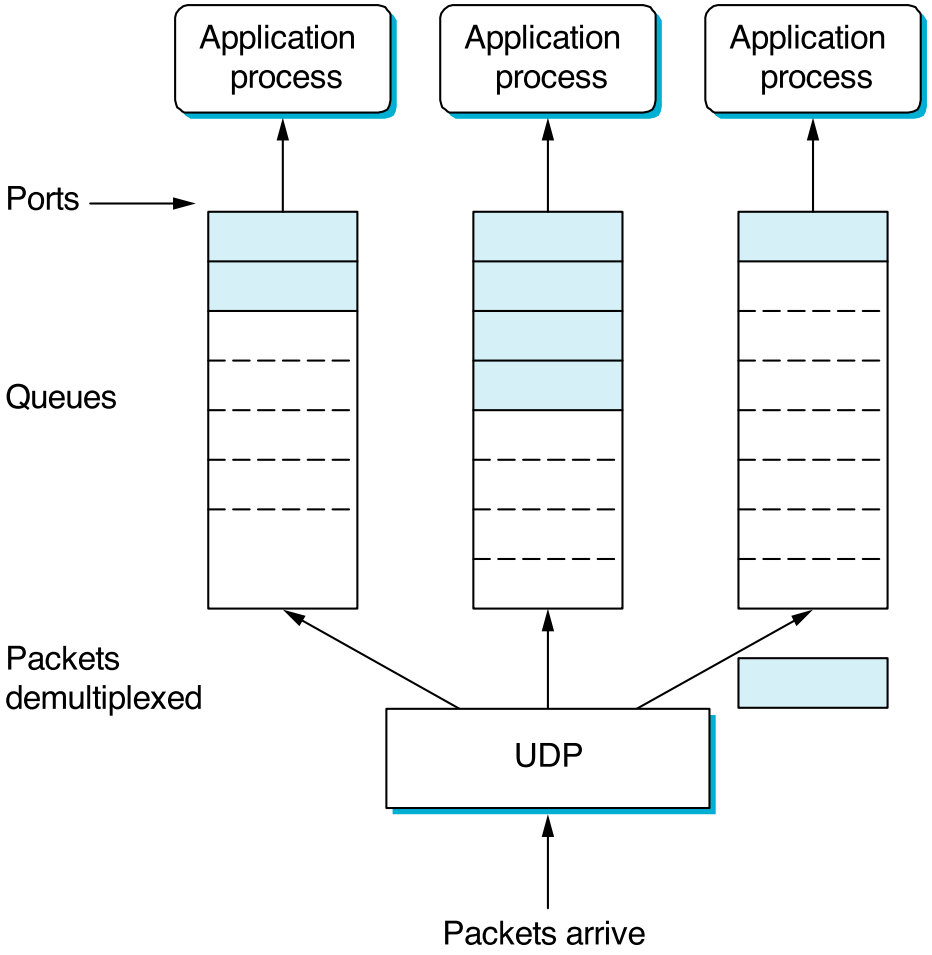
Transport Algorithms

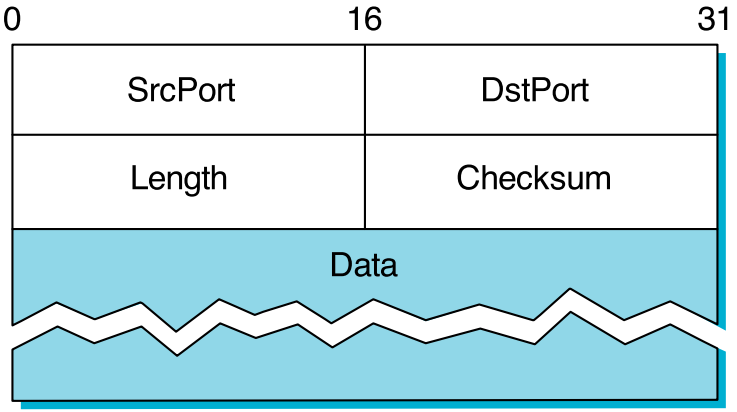
Simple asynchronous demultiplexing service

Reliable byte-stream service

Request/reply service

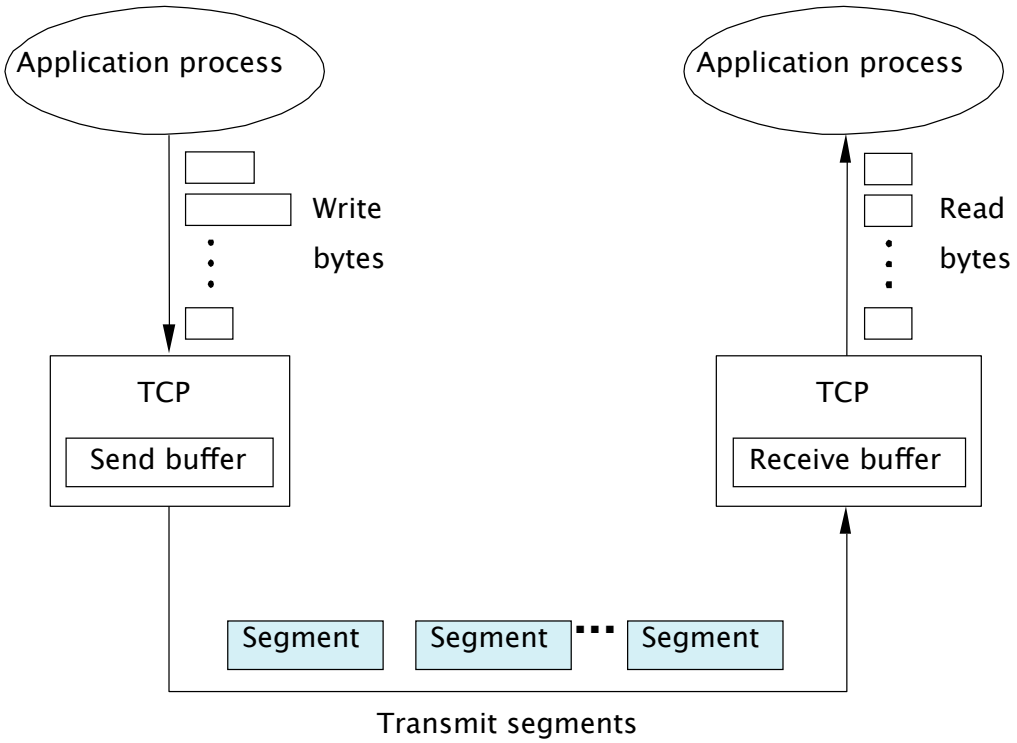
Simple Asynchronous Demultiplexing: UDP





UDP PDU format

Reliable Byte Stream: TCP



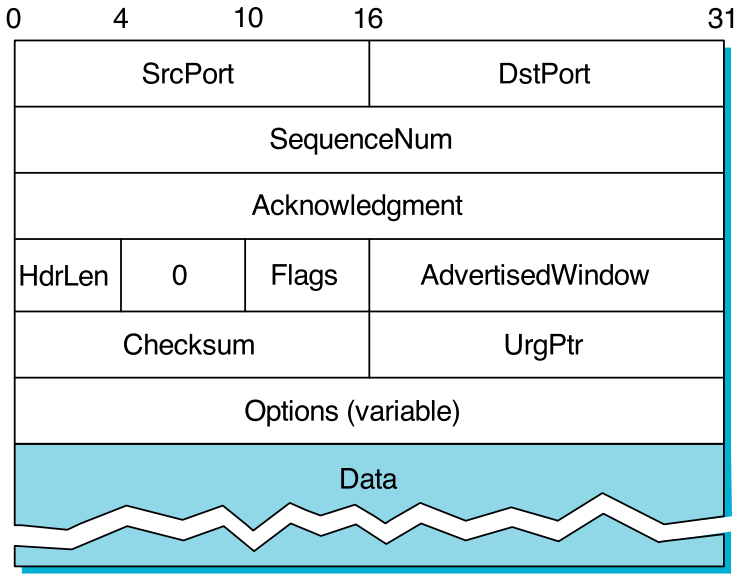
- Use the concept of ports to demultiplex streams to the same host.
- Guarantees reliable, in-order delivery of a ***stream of bytes*** without duplication.
- Implements flow-control allowing the receiver to throttle the volume of data transmitted by the sender.

Question: What is the difference between flow-control and congestion-control?

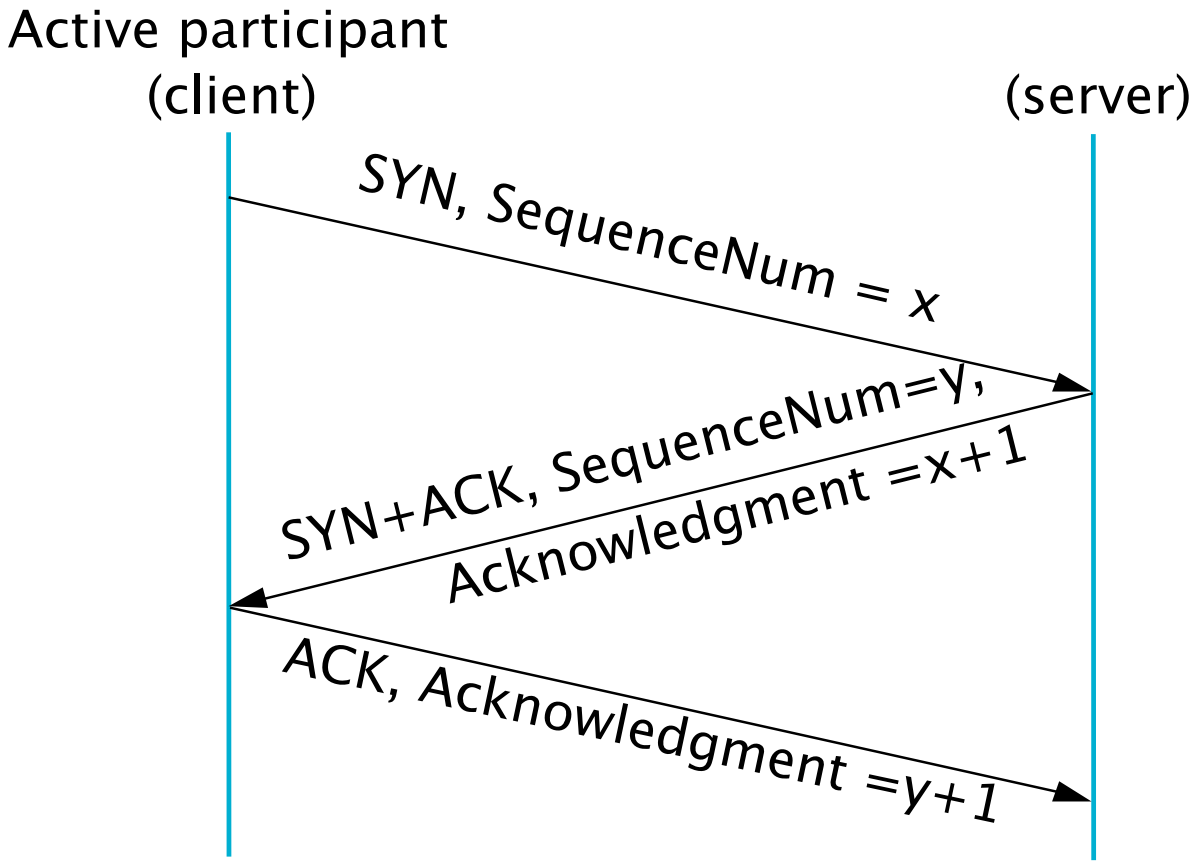
Segment Format

TCP is a byte-oriented protocol.

Question: What are the performance considerations that drive the implementation of such a protocol?



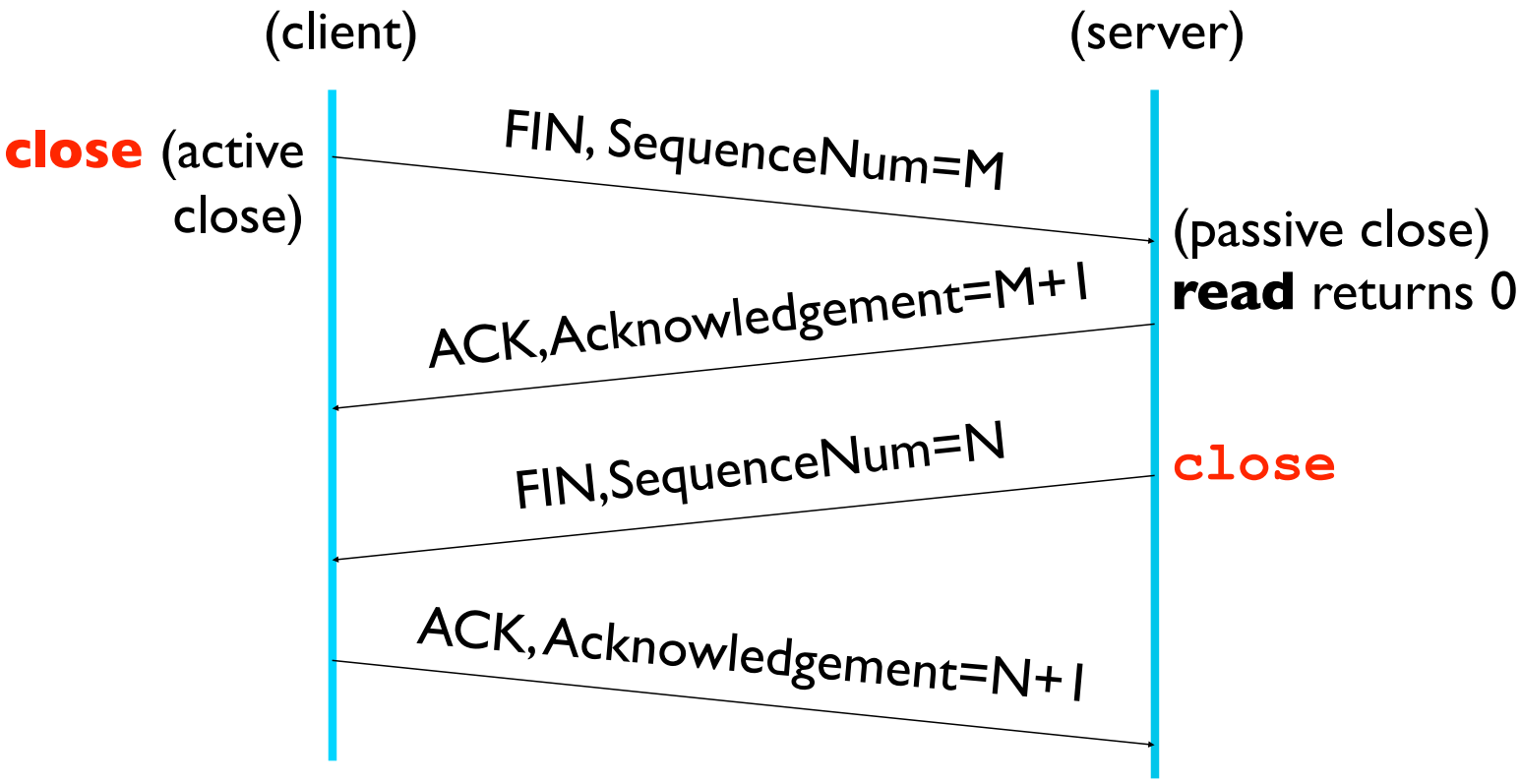
TCP Connection Establishment



Three-Way Handshake

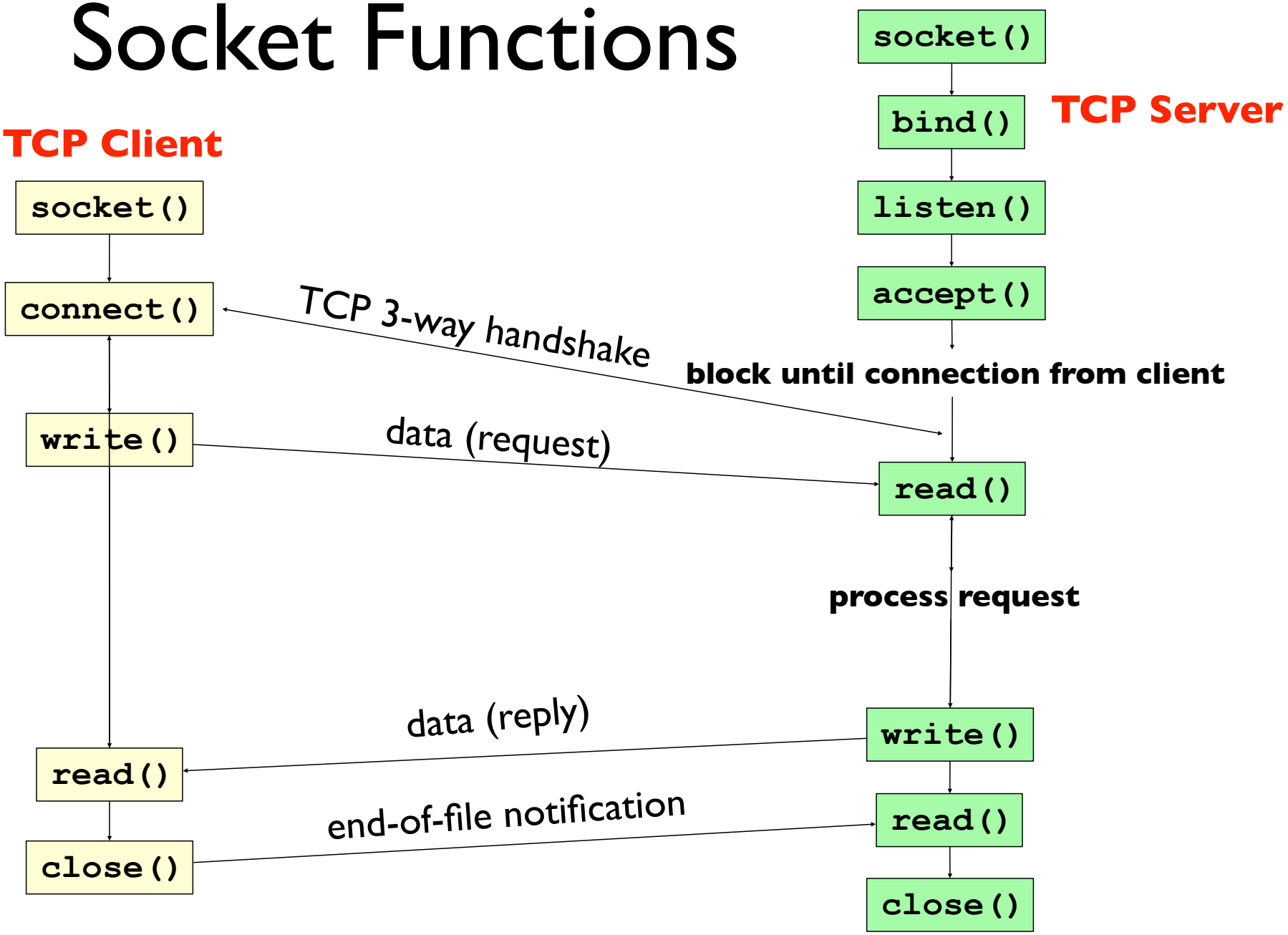
It takes 3 TCP segments to establish a connection.

TCP Connection Termination

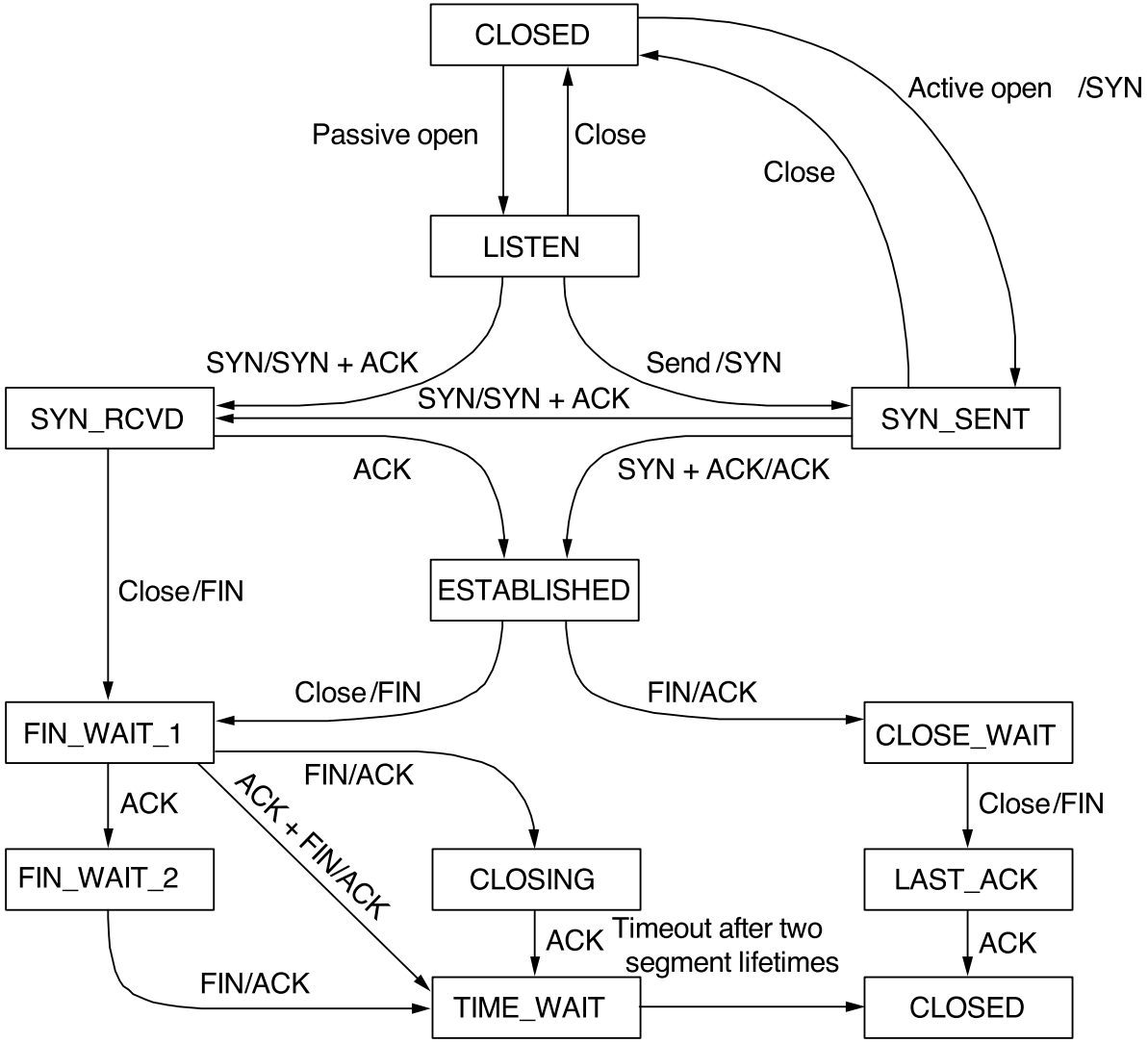


It takes 4 TCP segments to terminate a connection.

Socket Functions



State Transition Diagram



Question: How do your implement such a complex protocol?

Question: How do you verify that your implementation conforms to the specifications?

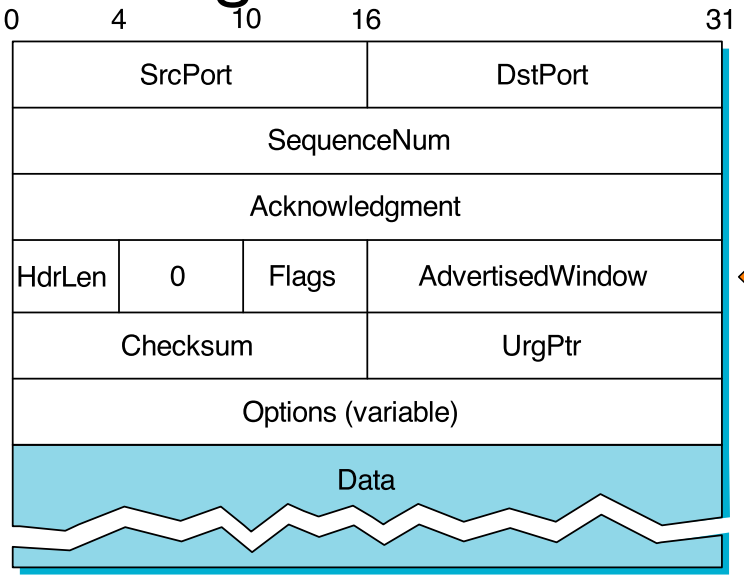
Note: Arcs are labeled with *event/action*.

Sliding Window

TCP's sliding window algorithm:

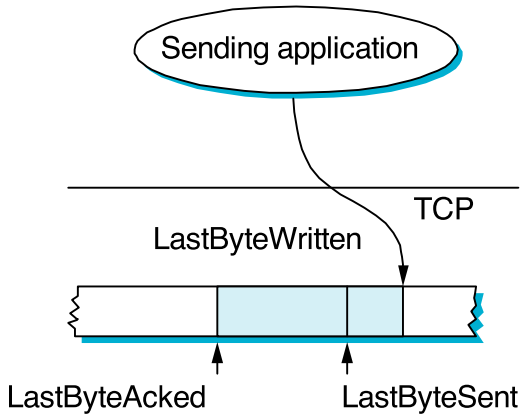
1. Guarantees reliable delivery.
2. Guarantees in-order delivery.
3. Enforces flow control.

Look again at the TCP segment header:

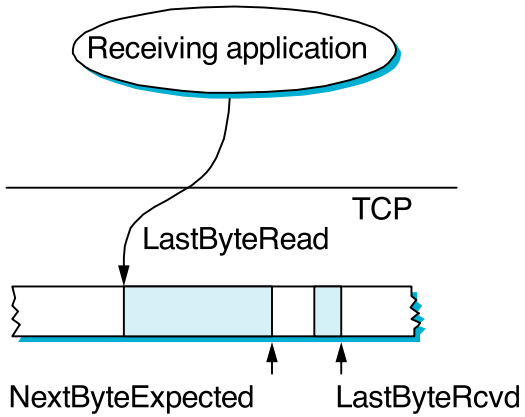


The receiver tells the sender the number of unacknowledged bytes of data it will allow (based on buffer size).

Flow Control



(a)



(b)

LastByteAcked<=LastByteSent

LastByteSent<=LastByteWritten

LastByteSent-LastByteAcked<=AdvertisedWindow

EffectiveWindow=AdvertisedWindow-
(LastByteSent-LastByteAcked)

LastByteWritten – LastByteAcked<=MaxSendBuffer

LastByteRead<NextByteExpected

NextByteExpected<=LastByteRcvd+1

LastByteRcvd – LastByteRead<=MaxRcvBuffer

AdvertisedWindow =MaxRcvBuffer –
((NextByteExpected-1)-LastByteRead)

When The Window Hits Zero

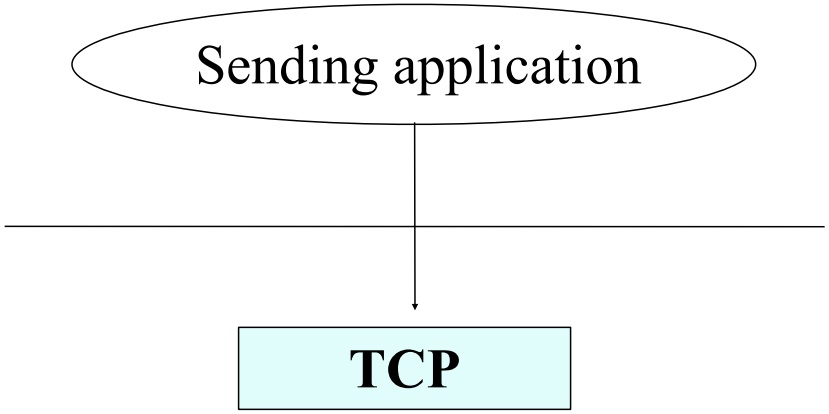
If the AdvertisedWindow reaches zero, the sender is not permitted to send any more data.

TCP ***always*** sends a segment in response to a received data segment with the latest values for Acknowledge and AdvertisedWindow. The receiver doesn't, however, send a spontaneous segment with this information.

Question: How does would the sender learn that the **AdvertisedWindow** has become greater than zero?

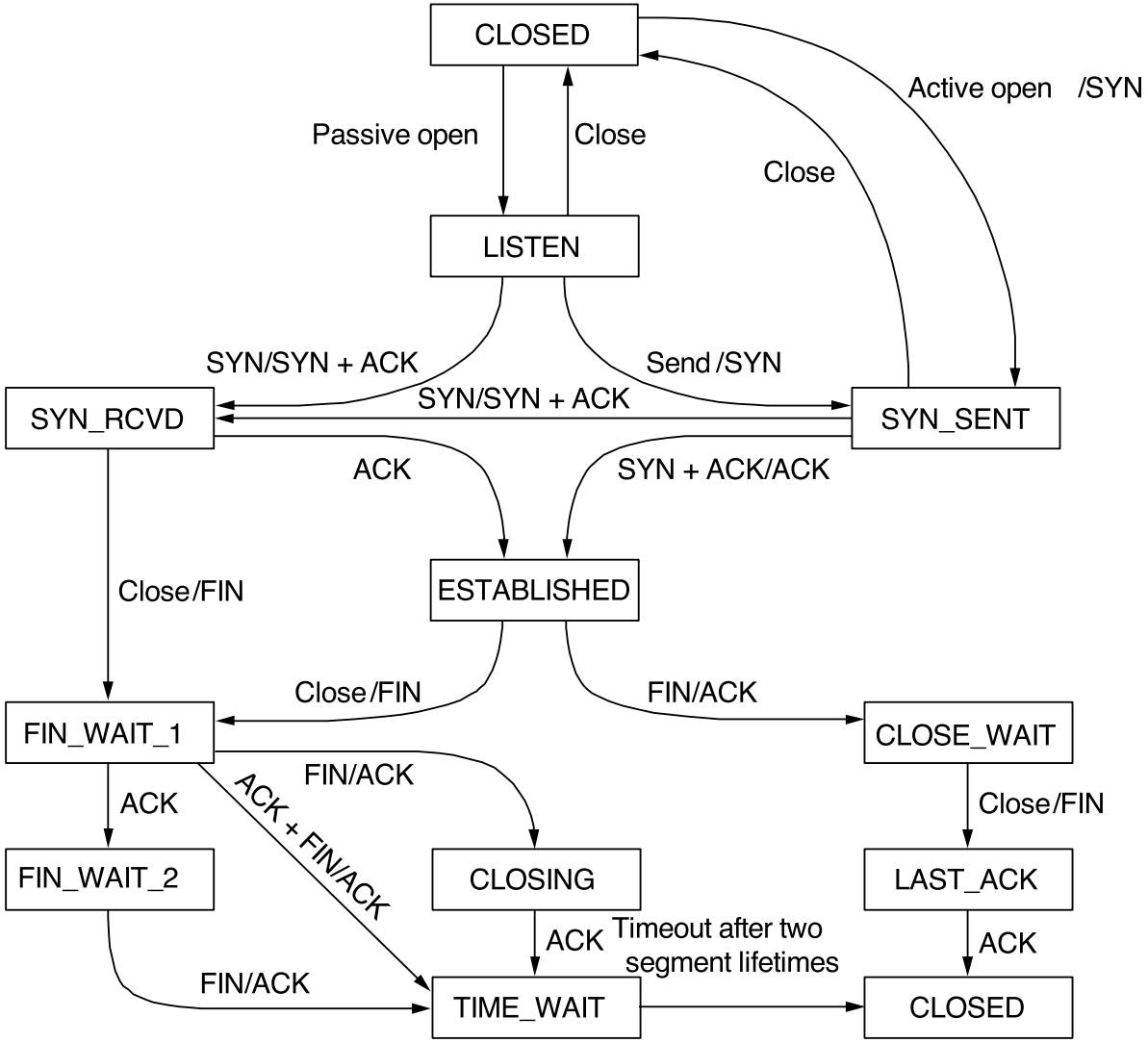
Triggering Transmission

Question: How should TCP decide to send a ***segment***?



1. As soon as MSS (maximum segment size) bytes have been collected from sender.
2. Whenever the sending application explicitly requests it (***push***).
3. Whenever a "timer" fires and then however many bytes have been buffered so far are sent.

State Transition Diagram



Question: How do your implement such a complex protocol?

Question: How do you verify that your implementation conforms to the specifications?

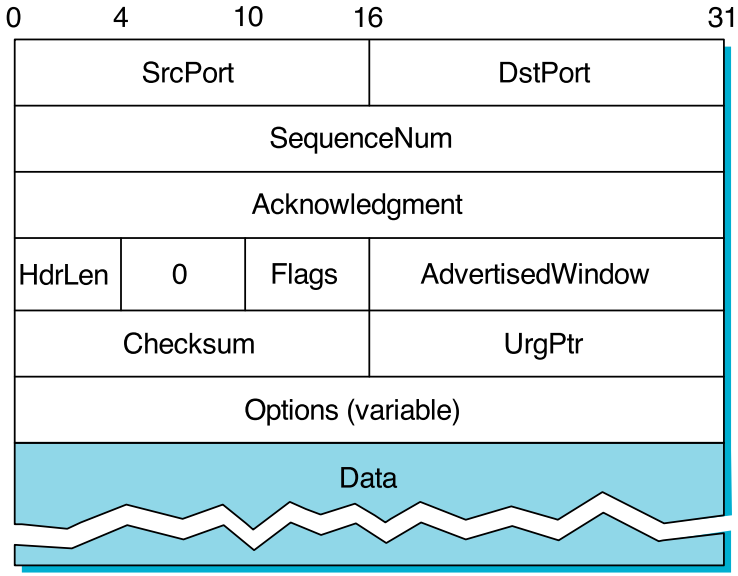
Note: Arcs are labeled with *event*/*action*.

Sliding Window

TCP's sliding window algorithm:

1) **Guarantees reliable delivery.**
2) **Guarantees in-order delivery.**
3) **Enforces flow control.**

Look again at the TCP segment header:



The receiver tells the sender the number of unacknowledged bytes of data it will allow (based on buffer size).

When The Window Hits Zero

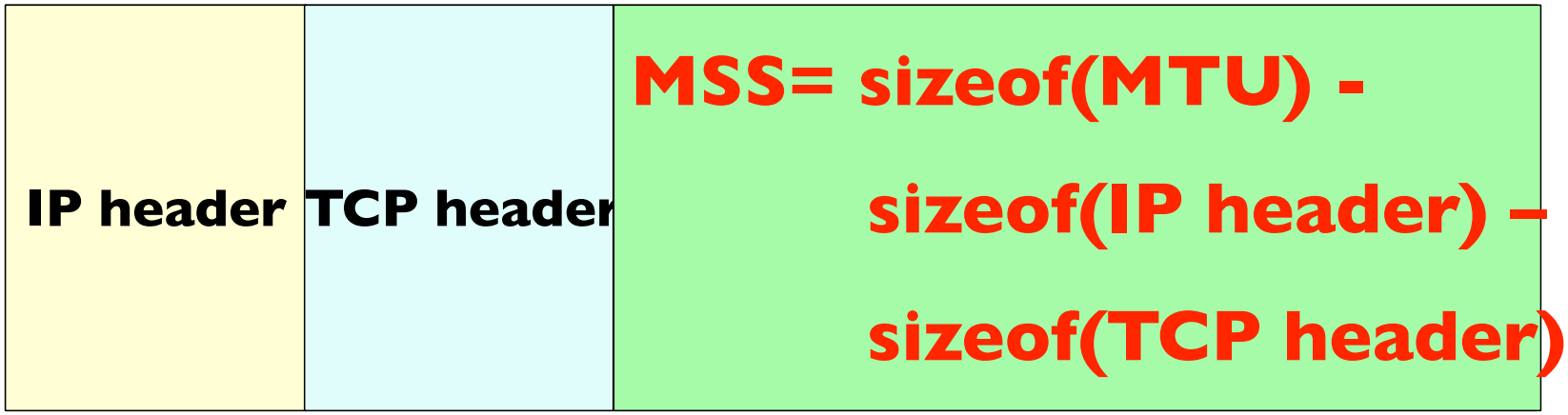
If the AdvertisedWindow reaches zero, the sender is not permitted to send any more data.

TCP ***always*** sends a segment in response to a received data segment with the latest values for Acknowledge and AdvertisedWindow. The receiver doesn't, however, send a spontaneous segment with this information.

Question: How does would the sender learn that the AdvertisedWindow has become greater than zero?

Maximum Segment Size (MSS)

Rule of thumb: Usually set to the size of the largest segment TCP can send without causing the local IP to fragment.



MTU of the directly connected network

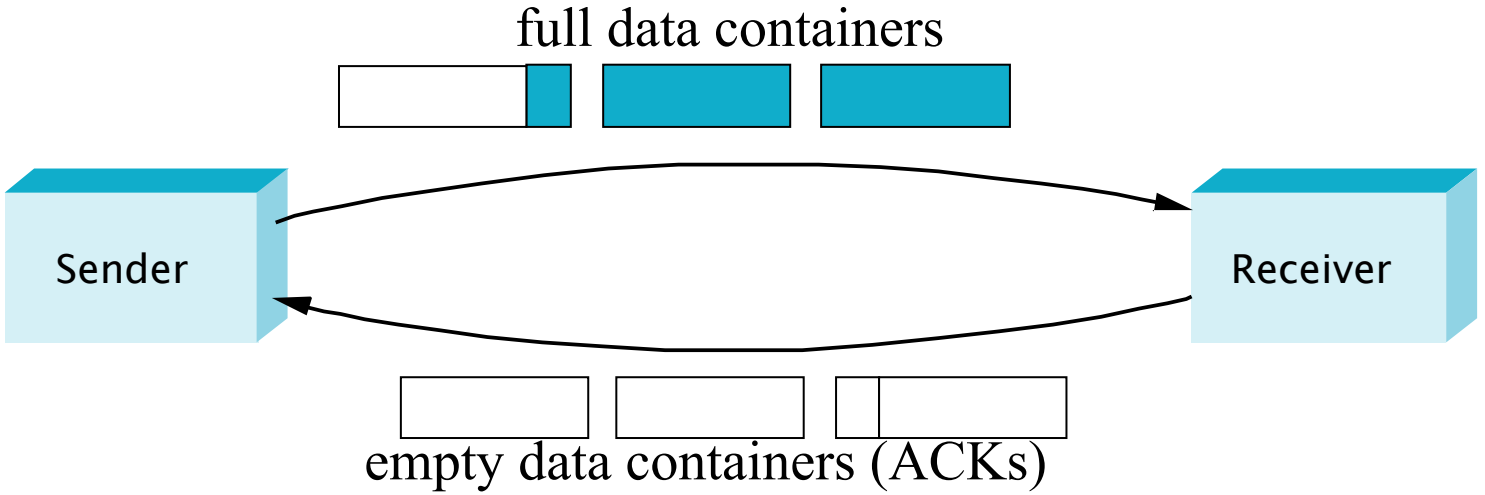
"Aggressive Send"

Now, consider what happens in the presence of flow control:

- AdvertisedWindow=0, so the sender is accumulating bytes to send.
- ACK arrives and AdvertisedWindow=MSS/2.
- **Question:** Should the sender go ahead and send MSS/2 or wait for AdvertisedWindow to increase all the way to MSS?

Silly Window Syndrome

- Consequence of aggressively taking advantage of any available window. Think of the TCP stream as a conveyor belt:



If the sender fills an empty container as soon as it arrives, the any small container introduced into the system remains indefinitely: it is immediately filled and emptied in each end.

Nagle's Algorithm

Goal: Make the sender application "wait long enough" so that small containers are coalesced into larger ones, but not so much that interactive applications will suffer.

Algorithm

```
when (application has data to send) {
    if (available data>MSS) && (AdvertisedWindow>MSS)
                send full segment
    else
                if (unACKed data in transit)
                  buffer the new data until an ACK arrives
                else
                  send all the new data immediately
}
```

PS: On by default on a TCP socket; option **TCP_NODELAY** turns it off.

Adaptive Retransmissions

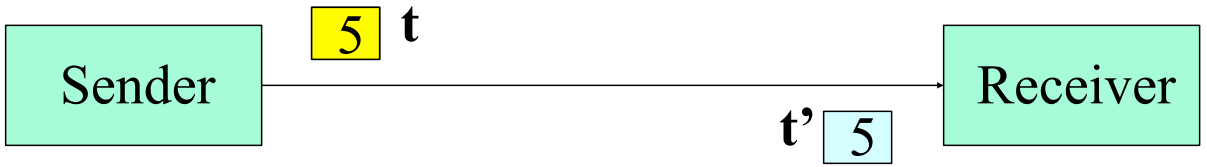
Sender starts a timer, pushes the packet to receiver, and waits for ACK. If timer expires, the sender retransmits the packet. ACK arrives at Sender; if timer running, then turn off timer.

Question: If the packets sent from the sender are identical, to which one does the ACK correspond?

Question: What should the setting of the timer be?

Original Timeout Algorithm

Goal: To keep a running average of the RTT between two host and use this value to set the timer.



$$\text{SampleRTT} = t - t'$$

$$\text{EstimatedRTT} = \alpha \times \text{EstimatedRTT} + (1 - \acute{a}) \times \text{SampleRTT}$$

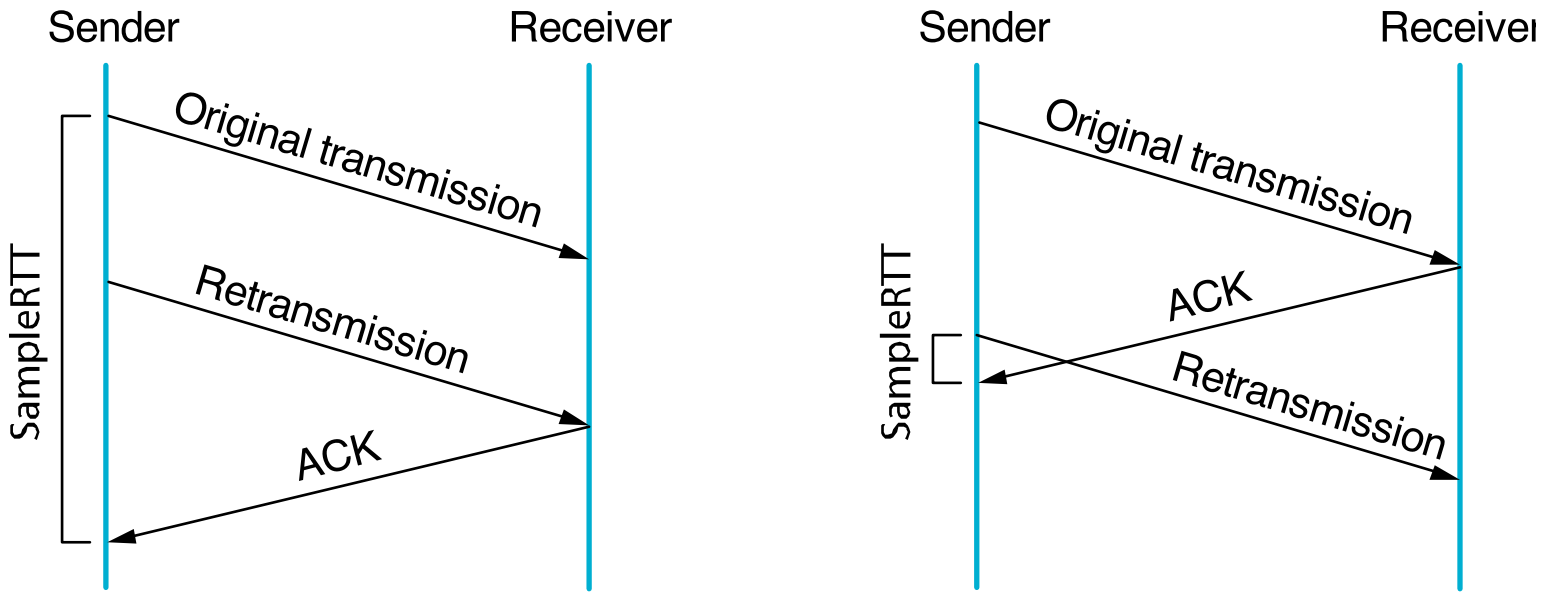
$$\text{TimeOut} = 2 \times \text{EstimatedRTT}$$

Question: What should α be?

Karn/Partridge Algorithm

ACK doesn't acknowledge a particular transmission.
ACK acknowledges the ***receipt*** of data.

Question: If you don't know which transmission is being ACKed, how do you compute SampleRTT?



Karn/Partridge Algorithm

Solution:

- Compute RTT only for the first transmission of a packet.
- Each time TCP retransmits a packet, set the timeout value to 2xTimeOut.

Rationale: If packets are being lost, this is likely to be the consequence of congestion, so the sender should be less aggressive.

Jacobson/Karels Algorithm

RTT variance in the original computation should be considered: if it's small then EstimatedRTT is a trusted number and there is no need to use 2xTimeOut on retransmissions.

$$\text{SampleRTT} = t - t'$$

$$\text{Difference} = \text{SampleRTT} - \text{EstimatedRTT}$$

$$\text{EstimatedRTT} = \text{EstimatedRTT} + (\delta \times \text{Difference})$$

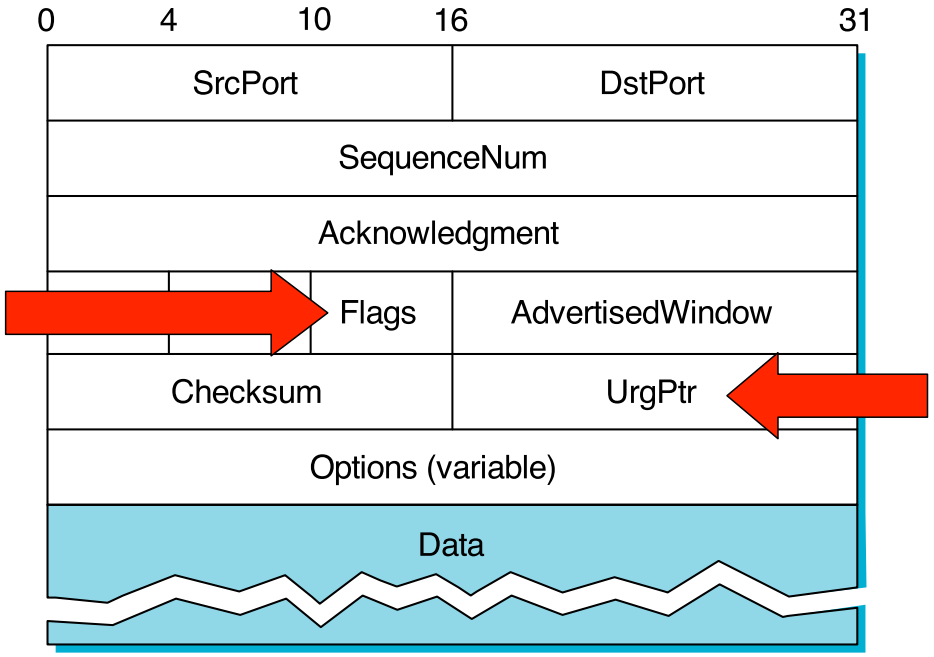
$$\text{Deviation} = \text{Deviation} + \delta(|\text{Difference}| - \text{Deviation})$$

$$\text{TimeOut} = \mu \times \text{EstimatedRTT} + \phi \times \text{Deviation}$$

$$\delta \in [0,1],\ \mu = 1,\ \text{and}\ \phi = 4$$

Record Boundaries

Problem: If TCP provides the abstraction of a byte stream, how can the receiver identify individual records?



1. Use the UrgentPtr and URG flag to mark the record boundary.
2. Use the PUSH flag: this means that whatever bytes sent to the receiver must be flushed out of the buffer. (The receiving side must support this option – note that the socket API doesn't.
3. The application can insert its own markers in the data stream.

TCP Extensions

- Since the header has variable length, it can be used to carry additional information.

- Not all hosts need to recognize the extensions, but when both sender and receiver do, they should agree to use them during the connection establishment.

Extension 1: Better RTT Estimate

- Write a 32-bit timestamp on the outgoing segment.
- The receiver echoes the timestamp back to the sender in ACK segments.
- The sender can read its clock when it receives the ACK and computer a more accurate estimate of RTT.

Extension 2: SeqNo Wrap Around

- Write a 32-bit timestamp on the outgoing segment.
- Using the 32-bit SequenceNumber field together with the timestamp increases the range to 64-bits.

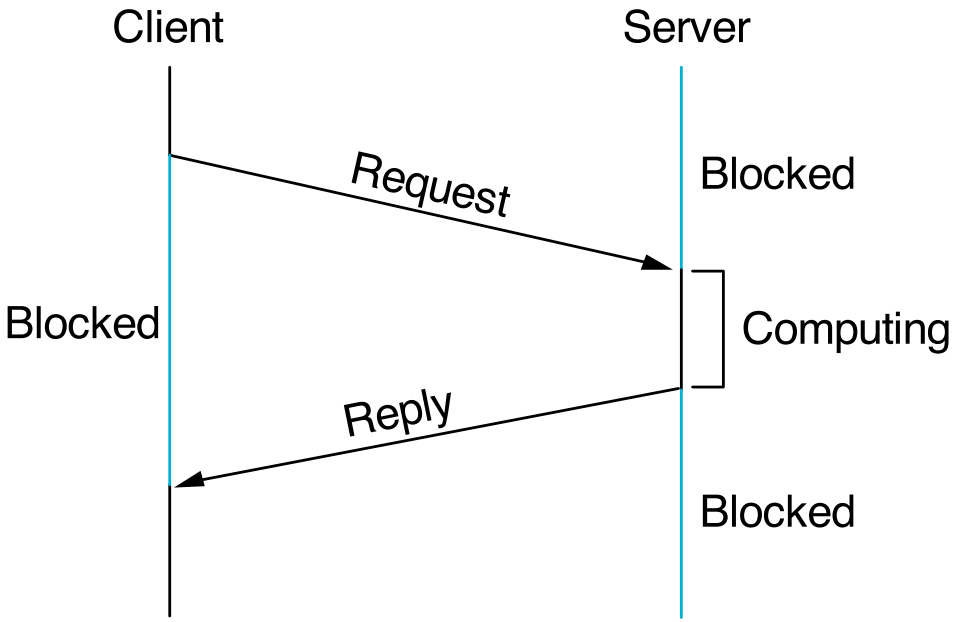
Note that timestamp is monotonically increasing and so it can be used to detect two different incarnations of the same SequenceNumber.

Extension 3: Scale AdvertisedWindow

- High-speed networks have larger delay x bandwidth pipes. In order to keep them full, one may need more the a 32-bit AdvertisedWindow.
- A scaling factor can be used together with AdvertisedWindow.
- If the scaling factor is 2, then AdvertisedWindow states how many unacknowledged 16-bits the sends can push out.

Remote Procedure Call

RPC Concepts

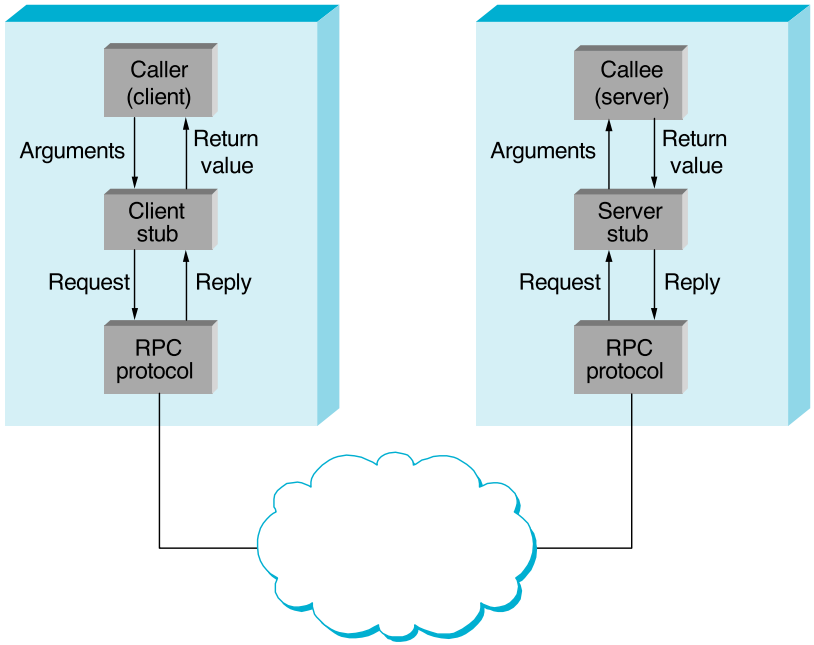


Remote Procedure Call is a mechanism for structuring distributed systems. It is based on the semantics of a local procedure call:

- The application program makes a call intro a procedure without regard for whether it is executed locally or remotely and blocks until the call returns.

Remember that the network between caller and callee (client and server) may not be reliable (duplicated or lost messages) and that there may be significant architectural differences between the two hosts (different data representation).

RPC Mechanism



The major components in RPC are:

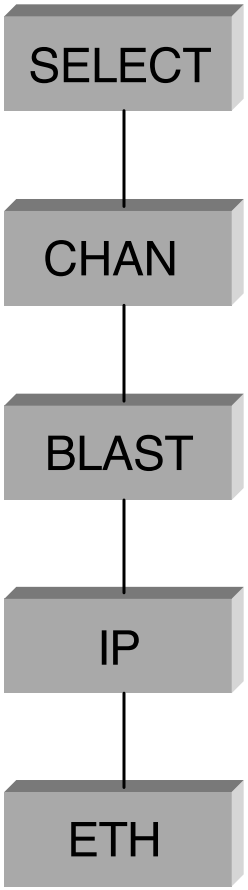
1. A protocol between caller and callee that makes up for the undesirable properties of the network.
2. Programming language and compiler support: the arguments of a call must be put into a request message and then translated to the callee's conventions. The returned data goes through a similar process.

RPC Protocol Design

The RPC protocol must provide several different functions, mainly:

- Fragmenting and reassembling large messages,
- Synchronizing request and reply messages, and
- Dispatching request messages to the correct process.

A Simple RPC Protocol Stack

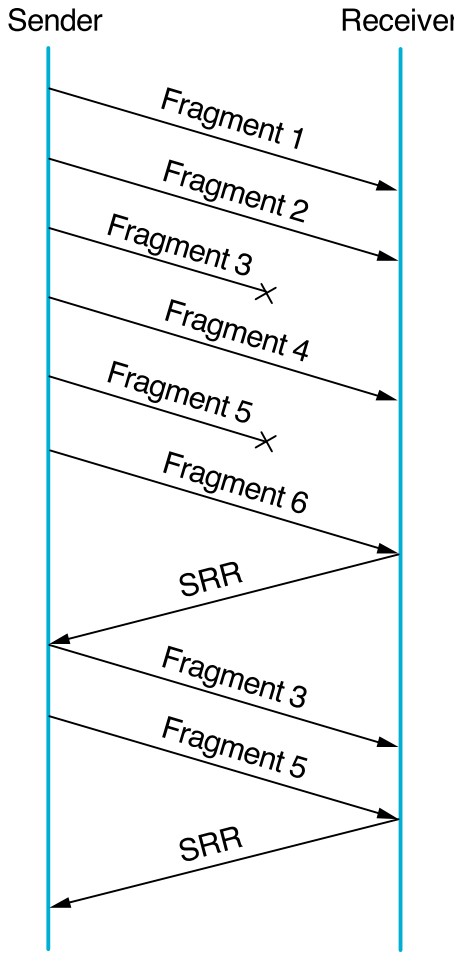


RPC is a generic term, not a specific standard like UDP, or TCP. This gives us the opportunity to design our own RPC protocol from scratch.

Our design will be based on three microprotocols:

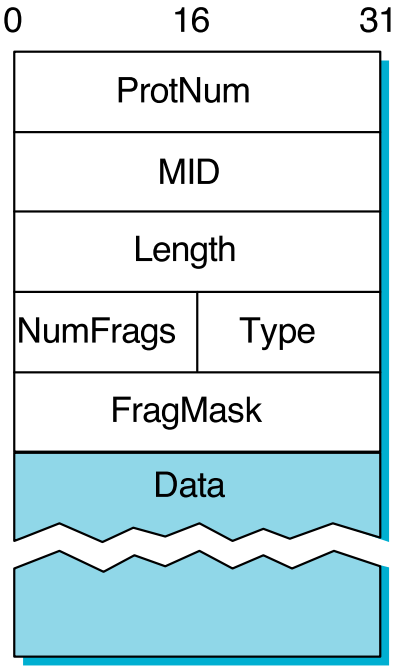
- **BLAST**: fragmentation and reassembly,
- **CHAN**: synchronization of reply and request messages, and
- **SELECT**: dispatching of request messages to the correct process.

BLAST

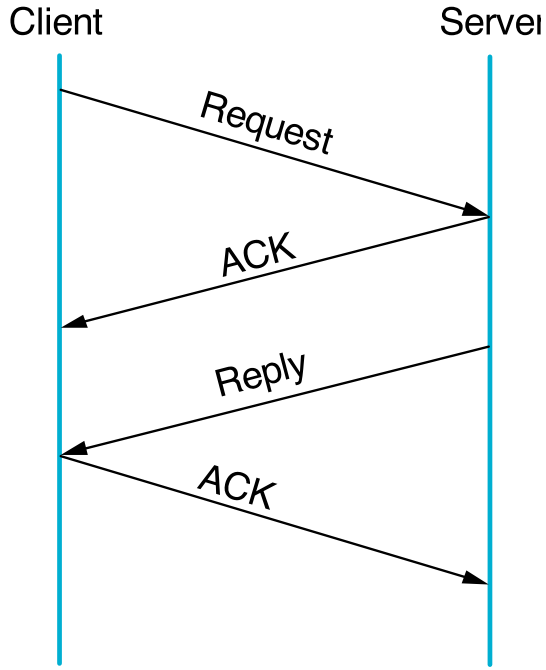


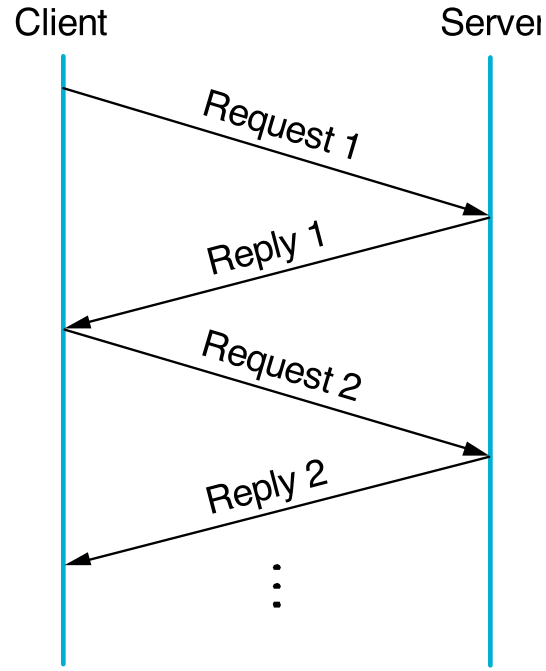
The design principle behind **BLAST** is to send all fragments to the receiver without waiting for each one to be acknowledged individually. The receiver uses a selective retransmission request (SRR) as partial acknowledgements.

Format for BLAST message header:



Spend a few minutes discussing with your group how we can make BLAST work. Also, think of what you would need to store in its message header.





CHAN

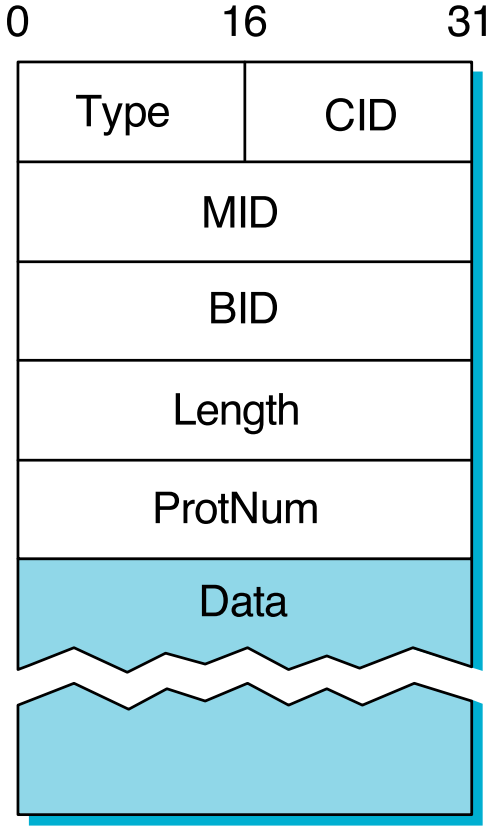
Microprotocol CHAN implements synchronization in the request/reply algorithm. It must also guarantee reliable message delivery (no loss, no duplication).

The protocol preserves the semantics ***at-most-once***: for every request message, either zero or one copy is delivered to the server.

When replies come almost immediately after request, ***implicit acknowledgements*** can be used.

CHAN

Format for CHAN message header:



Spend a few minutes brainstorming with your group. Try to provide answers to the following questions:

- What message types CHAN needs?
- How can CHAN distinguish a dead server from a slow server?
- If you use message identifiers, how would CHAN deal with hosts that reboot and reset the identifier to the same old starting value?
- How would CHAN implement synchronization?

SELECT

Microprotocol SELECT must dispatch request messages to the appropriate procedure in the server host.

Brainstorm with your group and attempt to identify the issues that drive an implementation of SELECT. Two important points to consider are:

- The addressing scheme used to identify each procedure.
- SELECT is a good place to manage concurrency: if client (or caller) is multi-threaded, since CHAN allows only one outstanding call at a time, SELECT should allow for calls made from independent threads to execute concurrently rather than sequentially.