The Transport Protocols

Hui Chen [a]

[a]CUNY Brooklyn College

October 5, 2023

Outline

Transport Services

- Connectionless service (a.k.a., datagram service)
- Connection-oriented service

Outline

UDP Service[1]

- A multiplexing service above the Internet protocol
- Connectionless service
- Unreliable service
- Reduce overhead of the protocol
  - Inward data collection (e.g., sensor data).
  - Outward data dissemination (e.g., multicast/broadcast messages to network users).
  - Request-reponse.
  - Real-time application.

[1] Postel, *User Datagram Protocol*.

UDP PDU: UDP Datagram

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Source Port																Destination Port															
Length																Checksum															

Connection-Oriented Services

Building connection-oriented service.

- Above reliable sequencing network service
- Above unreliable network service

Above Reliable Sequencing Network Service

Underlying network service,

- accepts messages of arbitrary length,
- has virtually 100 percent reliability, and
- delivers messages in sequence to the destination.

What are the design issues,

- Addressing
- Mulltiplexing
- Flow control
- Connection establishment
- Connection termination

Above Unreliable Network Service

Underlying network service is unreliable,

- there is packet loss,
- packets may arrive out of order,
- there may be duplicate packets.

What are the design issues,

- Addressing
- Multiplexing
- Ordered delivery
- Retransmission strategy
- Duplicate detection
- Flow control
- Connection establishment
- Connection termination
- Failure recovery

Outline

TCP Services[2]

- Data stream push and urgent data signaling
- TCP service request primitives
- TCP service response primitives

[2] Postel, *Transmission Control Protocol*.

TCP Service Request Primitives

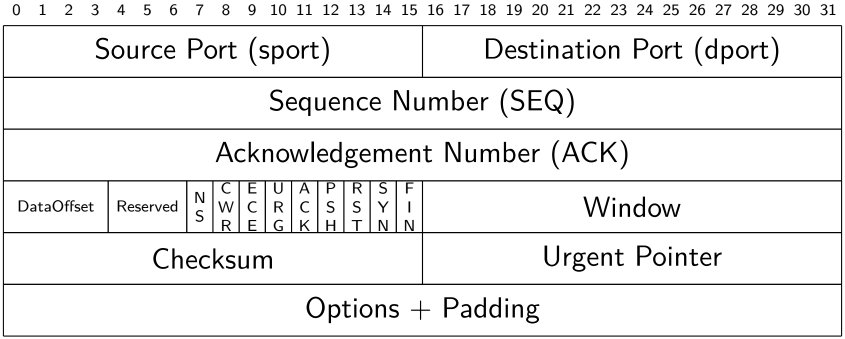
Primitive	Description
Unspecified passive open	Listen for connection attempt from any destination
Fully specified passive open	Listen for connection attempt from specified destination
Active open	Request connection at a specified destination
Active open with data	Request connection at and transmit data to a specified destination
Send	Transfer data
Allocate	Issue incremental allocation for received data
Close	Close connection gracefully
Abort	Close connection abruptly
Status	Query connection status

TCP Service Response Primitives

Primitive	Description
Open ID	informs TCP users of connection name
Open Failure	reports failure of an active open request
Open Success	reports completion of pending Open request
Deliver	reports arrival of data
Closing	reports TCP users has issued a Close and all data has been delivered
Terminate	reports that the connection has terminated
Status Response	reports current status of connection
Error	reports service-request or internal error

TCP PDU: TCP Segment

TCP uses only a single type of protocol data unit called a TCP segment



Design of TCP Segment Header

- Each connection identified with 4-tuple:

  `(SrcPort, SrcIPAddr, DstPort, DstIPAddr)`
- Flow control. Credit allocation/sliding window

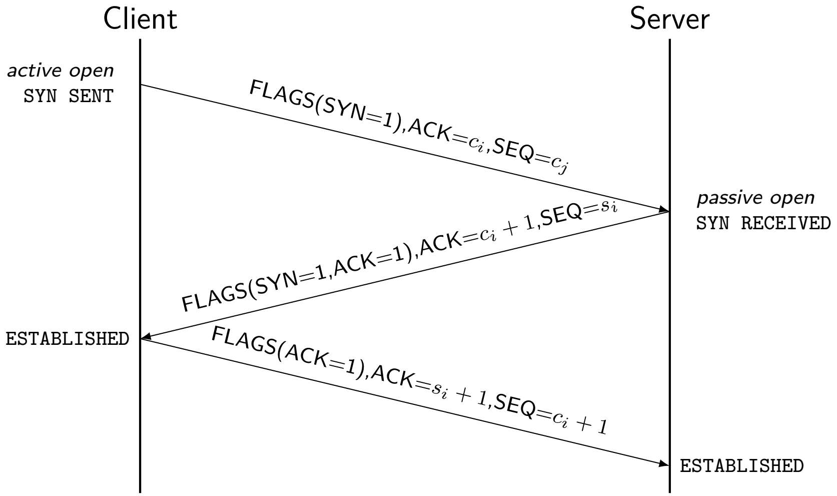
  `AcknowledgmentNum, SequenceNum, AdvertisedWinow`
- Connection and signaling (Flags).

  `SYN, FIN, RESET, PUSH, URG, ACK`
- Error detection. From data, TCP header, and pseudo header (important fields from IP header and TCP header to compute,

  `Checksum`

Connection Establishment

TCP uses a 3-way handshake to do connection establishment.

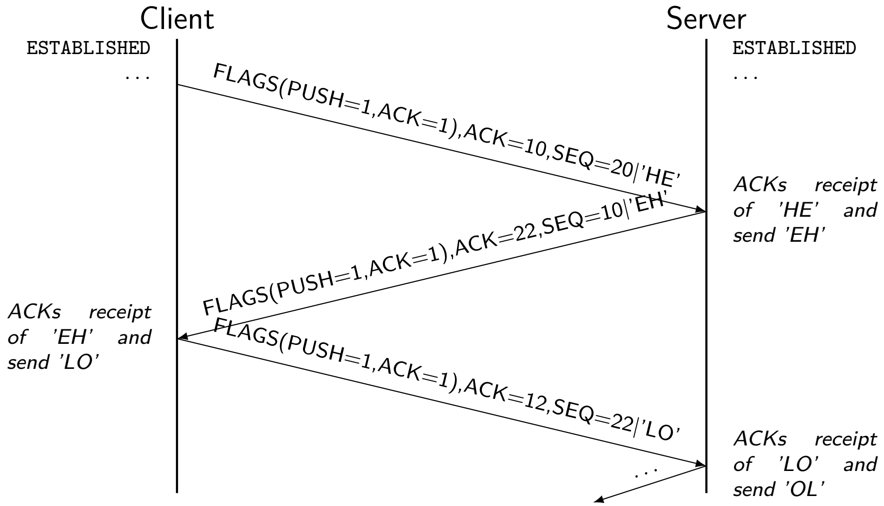

Client
Server
active open
SYN SENT
FLAGS(SYN=1),ACK=c_i,SEQ=c_j
passive open
SYN RECEIVED
FLAGS(SYN=1,ACK=1),ACK=$c_i + 1$,SEQ=s_i
ESTABLISHED
FLAGS(ACK=1),ACK=$s_i + 1$,SEQ=$c_i + 1$
ESTABLISHED

Data Transfer

- Logically a stream of 8-bit bytes (octets).
  - Every octet is numbered, modulo 2^{32}.
  - 32-bit sequence number in TCP segment is the sequence number of the first octet in the data field.
  - 32-bit acknowledgement number cumulatively acknowledges the octets received.
- Flow control. The credit allocation scheme (sliding window $+$ dynamic buffer allocation)
- Both transmission and reception ends buffers data.
  - Normally constructs TCP segments or release data to the user based on its own discretion.
  - The PUSH flag is used to force the data transfer or passing-on to the user.
  - A user may specify a block of data as urgent. The end of block is marked urgent (in TCP header). TCP alerts the user the arrival of the urgent data.

Example Scenario of Data Transfer

Let's consider a "telnet" like application.


Client
Server
ESTABLISHED
. . .
ESTABLISHED
. . .
FLAGS(PUSH=1,ACK=1),ACK=10,SEQ=20|'HE'
ACKs receipt of 'HE' and send 'EH'
FLAGS(PUSH=1,ACK=1),ACK=22,SEQ=10|'EH'
ACKs receipt of 'EH' and send 'LO'
FLAGS(PUSH=1,ACK=1),ACK=12,SEQ=22|'LO'
ACKs receipt of 'LO' and send 'OL'
. . .

What if there are lots of data to transmit?

Let's examine the following example.

- Host A sends a file of 500,000 bytes over a TCP connection with Maximum Segment Size (MSS) as 1,000 bytes to host B How many segments?

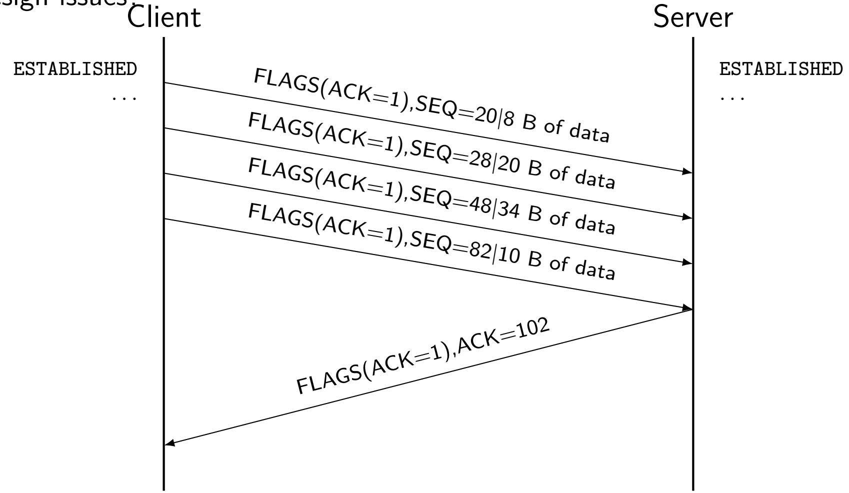
  ```
  500,000/1,000 = 500
  Sequence number assignments:
  Sequence number of 1st segment? 0
  Sequence number of 2nd segment? 1,000
  Sequence number of 3rd segment? 2,000
  ```
- But reality is quite complex due to "unreliable" network.

Effects of “Unreliable” Network

- Scenario 1. Host B received all bytes numbered 0-1,999 from host A
  - What would host B put in the acknowledgement number field of the segment it sends to A?
  - 2,000: the sequence number of the next byte host B is expecting
- Scenario 2a. Host B received two segments containing bytes from 0-999, and 2,000-2,999, respectively.
  - What would host B put in the acknowledgement number field?
  - 1000: TCP only acknowledges bytes up to the first missing byte in the stream, and it is the next byte host B is expecting
- Scenario 2b. Host B received two segments containing bytes from 0-999, and 2,000-2,999, respectively.
  - What does host B in this case that the segments arrive out of order (segment 3 arrived earlier than segment 2)?
  - TCP does not specify. Up to the implementation.
    - Option 1: Host B immediately discards out-of-order segment (simple receiver design)
    - Option 2: Host B keeps the out-of-order segment and waits for missing bytes to fill in the gaps (more efficient on bandwidth utilization, taken in practice)

Flow Control and Channel Efficiency

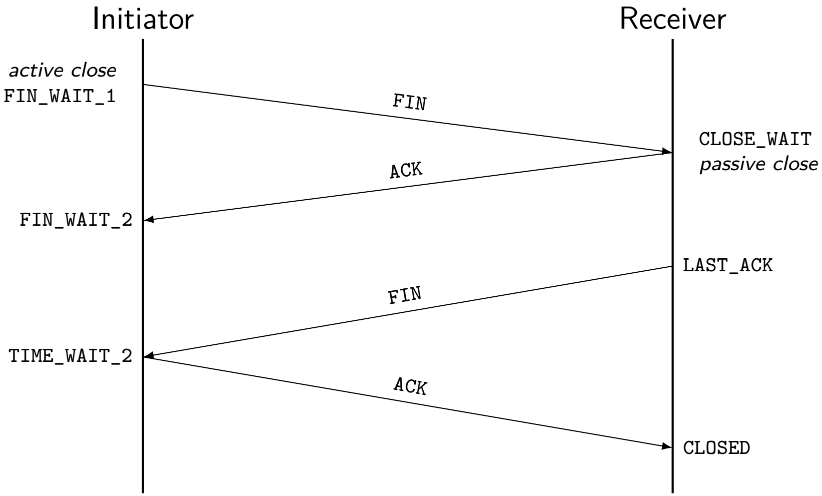
Send multiple TCP segments one after the another, but what are the design issues?


Client
Server
ESTABLISHED
...
ESTABLISHED
...
FLAGS(ACK=1),SEQ=20|8 B of data
FLAGS(ACK=1),SEQ=28|20 B of data
FLAGS(ACK=1),SEQ=48|34 B of data
FLAGS(ACK=1),SEQ=82|10 B of data
FLAGS(ACK=1),ACK=102

Connection Termination

- Graceful close.
  - Each TCP user must issues a CLOSE primitive.
  - TCP sets the FIN bit on the last segment it sends out.
- Abrupt termination.
  - It occurs when the user issues an ABORT primitive.
  - An RST segment is sent to the other end.
  - All attempts to send or receive data are abandoned, and buffered data are discarded.

Connection Termination



Scenarios of Connection Termination

- This side closes first

$$ESTABLISHED \rightarrow FIN\_WAIT\_1 \\ \rightarrow FIN\_WAIT\_2 \rightarrow TIME\_WAIT$$

- Other side closes first

$$ESTABLISHED \rightarrow CLOSE\_WAIT \\ \rightarrow LAST\_ACK \rightarrow CLOSED$$

- Both sides close at the same time

$$ESTABLISHED \rightarrow FIN\_WAIT\_1 \\ \rightarrow CLOSING \\ \rightarrow TIME\_WAIT \rightarrow CLOSED$$

Implementation Policy Options

- Send policy
- Deliver policy
- Accept policy
- Retransmit policy
- Acknowledge policy