TD6 - Networks and protocols

TI602

In this tutorial we are interested in data transport and in the TCP protocol in particular.

Case study

We are interested in an online shooter video game platform where players control characters and compete in 3D space. Characters can interact with the environment and with other characters by attacking each other. This platform runs on a local network infrastructure, each player accesses the game on a machine on which his client application is running. A central server manages the synchronization of the game's views and actions: a client application sends data to the servers at a fairly high frequency on the position of the player's character, movements and the actions executed by the latter (in particular on the other players characters). The server after receiving this information, made a computation based on the actions and movements done (received) and distributes the updates to the other machines so that everyone can see everyone's state almost in real time locally on their machine. In this kind of games, the speed of the messages as well as the optimization of resources is very important in order to guarantee an optimal gaming experience (maximum data transmitted correctly and low latency for a view close to real time).

We have seen in the previous tutorial that UDP more suitable to transport game data in the context of our application. In this tutorial we are interested in the exchange of data when downloading updates and user data, which this time is done using the TCP transport protocol and FTP application protocol.

1. Connection Establishment

The establishment of a TCP connection is done by exchanging three messages (three-way handshake).

a. Why three messages? Wouldn't two messages have been enough?
b. Remind with a diagram the connection opening messages exchanged, specifying the fields essential to the establishment.
c. Why didn't the sequence number start at 0 ?
d. When the server receives two SYN segments from the same port of a client B, the second SYN can be a retransmission of the original SYN or a new connection request (in the event of a failure followed by a restart of B). How did the server tell the difference between these two cases?

2. Sequence number and Acknowledgement

One of the main properties of TCP is reliability (extracts from RFC 793):

`The TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the internet communication system. This is achieved by assigning a sequence number to each octet transmitted and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.`

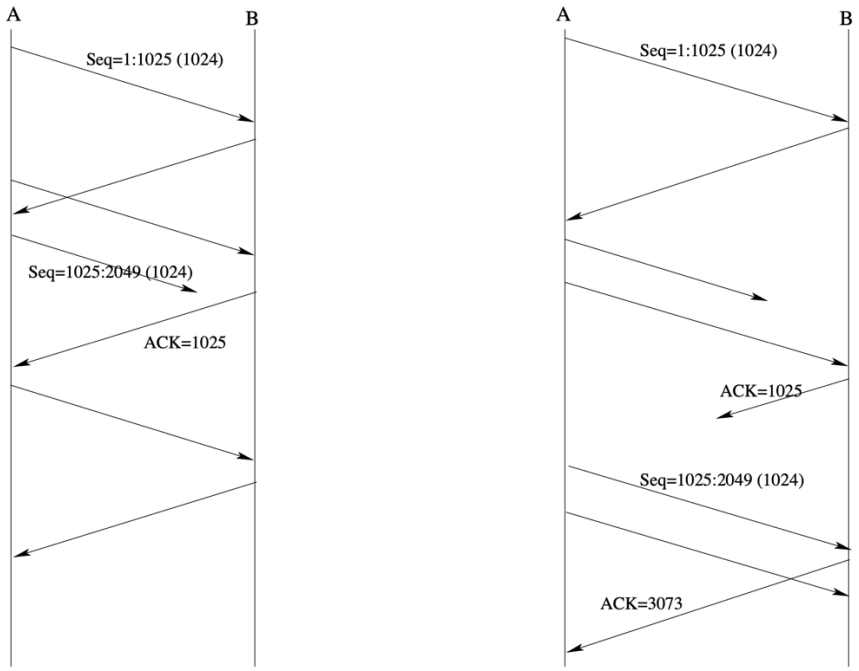
TCP therefore numbers data bytes and not its segments. The sequence number of the first byte of data is transmitted with the segment in the SequenceNumber field of the TCP header (32 bits), and is called the segment sequence number.

a. The 32 bits are sufficient to cover 4 billion bytes of data. Even considering that such an amount of data will never be transferred over a single connection, why is it possible to see the sequence number go from 2^{31}-1 to 0?

b. Assuming that the client is connected to the update server on a 1 Gbit/s link. How long does it take for the sequence number to complete loop, give a rough estimate?

   TCP segments also carry an acknowledgment number in the AcknowledgmentNumber field, which represents the sequence number of the next expected data byte in the reverse data stream.

c. Complete the following diagrams, knowing that for the first one (left), the message has a size of 2048 bytes and that for the second (right), the message has a size of 3072 bytes. Remember that the (tcpdump) notation a: b (c) means that we send a segment with a as the sequence number, b is the implicit end sequence number, and finally c gives the size of the data in bytes. Note that c = b - a.



d. Assuming an inactive TCP connection (no data transfer in progress) between a client A and the server. A third party has hacked the communication and has learned the sequence numbers used on both sides. Suppose the third party, pretending to be the server, sends a segment containing 100 bytes of data to A. What happens?

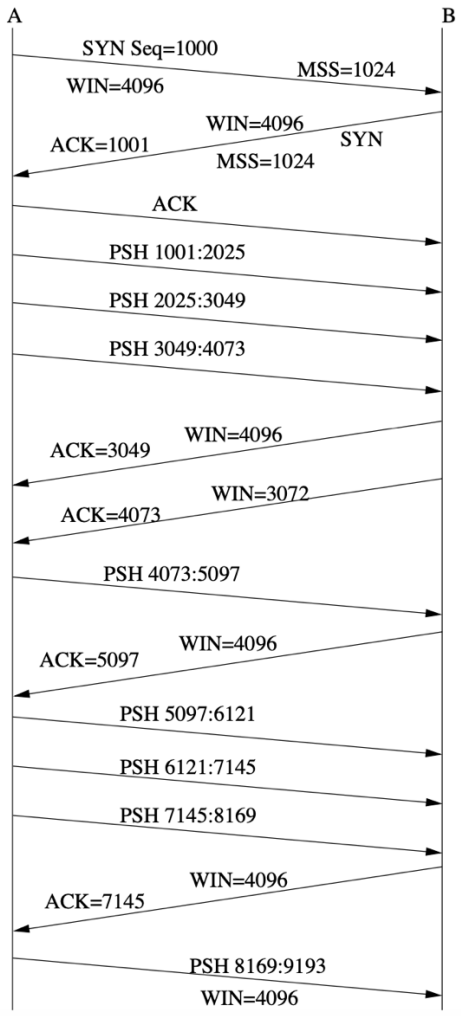
3. Flow Control

For flow control, TCP uses a sliding window mechanism of variable size:

`TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.`

a. What are the parameters that change the size of the window? How are the actors informed?

b. In the following figure, give after each sent and received message the amount of TCP data that machine A can still send without receiving an acknowledgment from B.

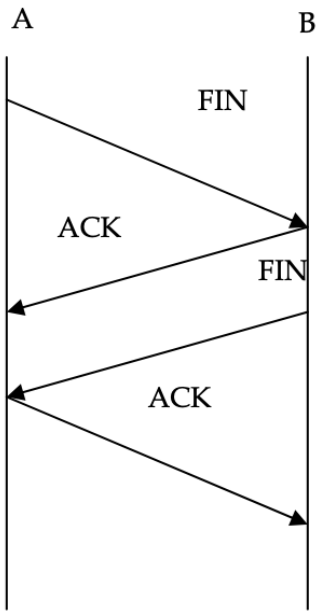


We aim to design a reliable, byte-stream oriented, sliding window protocol. This protocol is intended to operate on a 100 Mbit/s network. The network RTT (Round-Trip Time) is 100 ms, and the MSL (Maximum Segment Lifetime) is 60 s.

c. How many bits should the AdvWin field of the protocol be?
d. How many bits should the SeqNum field of the protocol be encoded?
e. Among the values given in the statement, which ones can be obtained reliably?
f. A client on a TCP connection that receives an AdvWin at 0 periodically probes the server to determine when the window opens. Why would the server need a timer if it was responsible for explicitly and spontaneously signaling the window opening?
g. TCP is byte-oriented: the numbering and the window relate to bytes. When does TCP decide to build and send a segment?

4. Connection Closing

The segment exchange for a connection closing is shown below:



a. When TCP sends a [SYN, SeqNum = x] or a [END, SeqNum = x], the corresponding ACK segment carries AckNum = x + 1. In other words, the SYN and FIN segments consume one unit of the numbering space. Is it really necessary? Could we have had [ACK, AckNum = x] instead?