

Q1 When Does the Receiver ACK? Slides 88, 91

Names: _____

Host **A** is uploading a file to host **B** over a connection that is already established. Every segment A sends carries **100 bytes**. The capture uses relative sequence numbers, so the first byte of the connection is **byte 1**. B sends no data of its own and has plenty of free buffer space.

A sends six segments back to back. The third of them, carrying **bytes 201–300**, is **lost**; every other segment arrives in the order it was sent. An ACK number names the **next byte B expects**. B decides what to send back using four rules:

Rule	What arrives at B	What B does
R1	In-order segment, the byte B expected next. Nothing else is waiting to be ACKed.	Wait up to 500 ms for another segment. If none comes, send the ACK.
R2	In-order segment, the byte B expected next. One earlier segment is still waiting to be ACKed.	Immediately send one cumulative ACK covering both.
R3	Out-of-order segment: a higher sequence number than expected, so a gap has opened.	Immediately re-send the ACK for the last in-order byte.
R4	A segment that fills or partly fills the gap.	Immediately send an ACK, provided the segment starts at the low end of the gap.

1. For each segment that reaches B, give the rule that fires and the Ack number B puts in its reply. Write “none yet” where B does not send one. Row 1 is filled in as an example.

#	Arriving at B	Bytes it carries	Rule	Ack number B sends
1	1st segment	1–100	R1	none yet
2	2nd segment	101–200		
3	4th segment	301–400		
4	5th segment	401–500		
5	6th segment	501–600		
6	3rd segment, resent by A	201–300		

2. At the moment A decides to retransmit, which bytes does B actually have? Can A know that from the ACKs it has received?

Discuss with your partner

R1 delays an ACK for up to 500 ms to save traffic. What will happen if that delay is longer than A's retransmission timer but A only has one data segment to send?

Q2 Two Handshakes, One Bad Network Slides 99–103

A connection cannot start until both sides agree they want it and agree on the sequence number each will start from. **The network in this question may delay a message for a very long time without losing it**, and either side retransmits anything it thinks went missing. Nothing is corrupted and nothing is lost forever. Here are both protocols.

2-way handshake

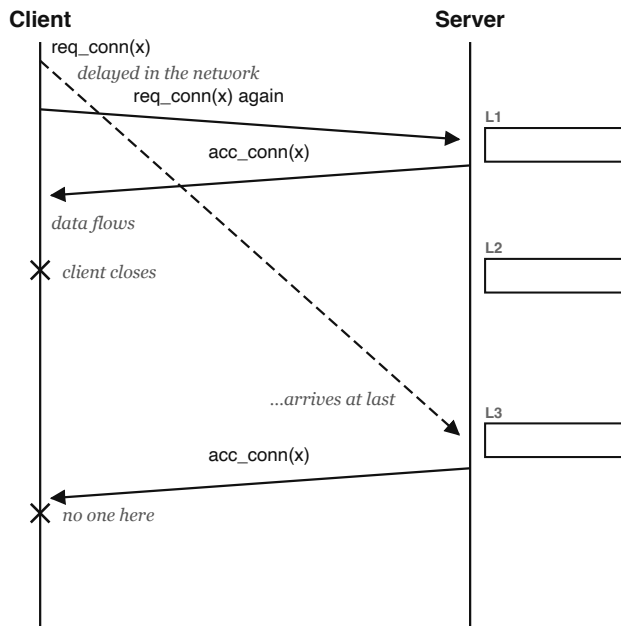
- **Client.** Picks a starting number x and sends `req_conn(x)`. Retransmits it if no reply comes back in time. Enters `ESTAB` when `acc_conn(x)` arrives.
- **Server.** On receiving `req_conn(x)`: records x , **allocates the connection's buffers**, enters `ESTAB` immediately, and replies `acc_conn(x)`.
- **Server states:** `LISTEN` $\rightarrow$ `ESTAB`. The server leaves `ESTAB` only when the connection is closed by a client, and **it runs no timer while in `ESTAB`**.

3-way handshake (TCP)

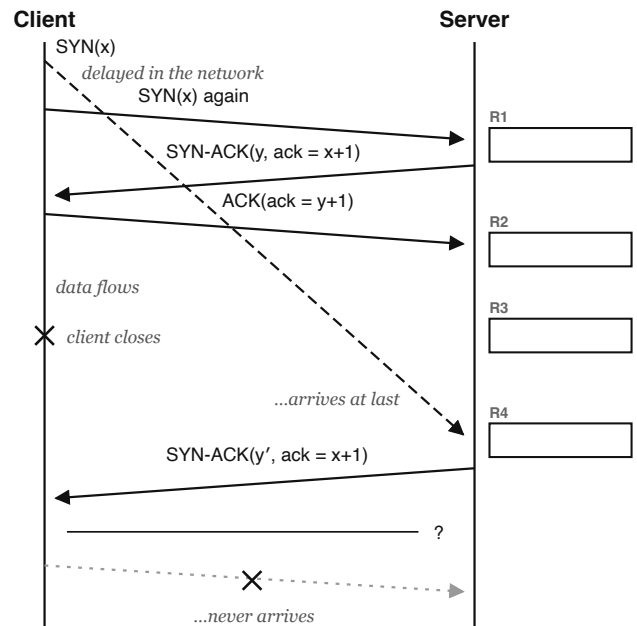
- **Client.** Picks x and sends `SYN(x)`. On receiving `SYN-ACK(y, ack = x+1)`: enters `ESTAB` and replies `ACK(ack = y+1)`.
- **Server.** On receiving `SYN(x)`: records x , **picks a brand-new number y that it has never used before**, replies `SYN-ACK(y, ack = x+1)`, and enters `SYN_RCVD`. It does **not** allocate the connection yet. It enters `ESTAB` only when an `ACK` carrying `ack = y+1` arrives.
- **Server states:** `LISTEN` $\rightarrow$ `SYN_RCVD` $\rightarrow$ `ESTAB`. A `SYN_RCVD` entry has a **timer running**, and is thrown away if no matching `ACK` arrives. After a client closes the connection, the server returns to `LISTEN`.

Run this scenario for both protocols: The client's *first* request is delayed in the network. It retransmits, the connection is established, data is exchanged, and the client closes. Long afterwards, the delayed packet finally reaches the server.

2-WAY HANDSHAKE



3-WAY HANDSHAKE



1. Write the server's state in each of the seven boxes, using only `LISTEN`, `SYN_RCVD` and `ESTAB`.
2. Compare **L1** and **R1**. The same message has just arrived at both servers. What resources has the 2-way server committed that the right-hand one has not?
3. On the 2-way side, the server ends in the state you wrote in **L3**. Will it release any reserved resources?
4. On the 3-way side, write on the dashed arrow the message that would have to arrive for the server to reach `ESTAB`. Give **two** reasons it never will — one about the client, and one about the number y' itself.

Q3 Why y Must Be New Slides 104–105

Names: _____

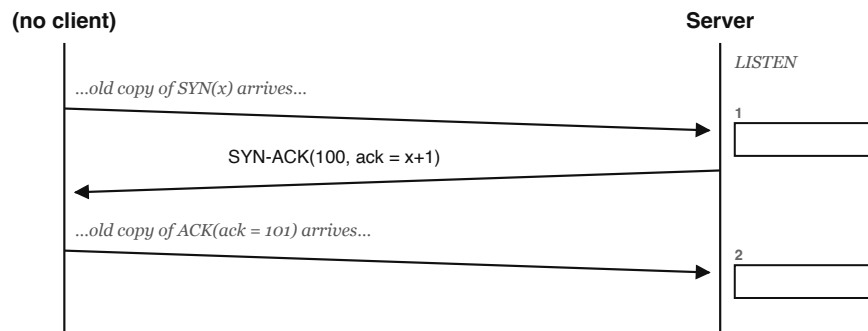
Q2 showed that the 3-way handshake survives an old duplicate. This question is about *why*, and about what would break if one detail changed.

- For each message of the 3-way handshake, what does the side that **receives** it now know for certain?

Message	Received by	What the receiver now knows for certain
SYN(x)	server	
SYN-ACK(y , ack = $x+1$)	client	
ACK(ack = $y+1$)	server	

- Messages 2 and 3 do the same job in opposite directions. Say what that job is in one sentence, and explain why message 1 cannot do it for either side.

- A lazy server always uses $y = 100$ for every connection instead of picking a new number each time. A client opened a connection to this server earlier, so copies of SYN(x), SYN-ACK(100, ack = $x+1$) and ACK(ack = 101) have all crossed the network at some point — and the network may still deliver an old copy of any of them. Complete the timeline by filling in the boxes.



4. Three messages were exchanged in part 3, exactly as the protocol requires, and the server still ended up connected to nobody. What property must y have for the 3-way handshake to do its job?

What we mean by an attack

Everything so far went wrong *by accident* — an old copy turned up at a bad moment. Now add a third machine that is doing it deliberately.

There are three parties: the **server**; a **client the server trusts**, perhaps because the server only accepts connections from inside the university; and an **attacker** somewhere else on the Internet. Neither the client nor the server is misbehaving — the attacker is a stranger to both. What the attacker wants is for the server to open a connection that it believes belongs to the trusted client, so that it can then send data the server would never accept from the attacker's own address.

The attacker can write **any source address** into the packets it sends, so it puts the **trusted client's** address on a $\text{SYN}(x)$. But it has stolen the address, not the path: the server's reply, $\text{SYN-ACK}(y, \text{ack} = x+1)$, is delivered to the **trusted client**, not to the attacker. **The attacker never sees y .** So the whole attack comes down to one question — can it produce $\text{ack} = y+1$ without having seen message 2? If it can, the server completes the handshake and believes the trusted client is on the other end. If it cannot, the attacker is stuck and the server's SYN_RCVD entry simply times out.

5. Which of these choices of y would be safe? For each, say what an attacker or an old copy would have to do to defeat it.

Choice of y	Safe?	Why / how it is defeated
$y = x$ (reuse the client's number)		
A counter the server increases by 1 per connection		
A number the server picks at random		

6. After message 3 arrives, the server knows the client is live. Does the **client** know that the server received its ACK ? If not, why is the handshake still finished?