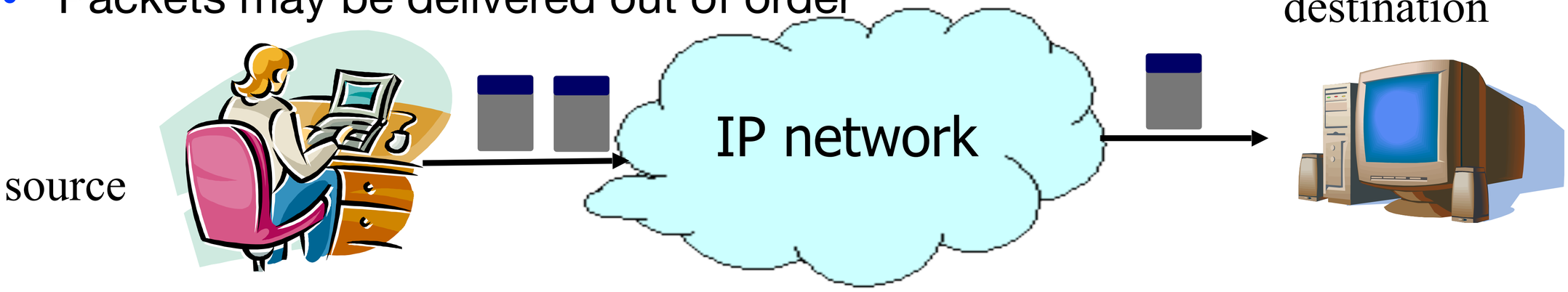
IP: "*Best Effort*" Packet Delivery

- Routers inspect destination address, locate "next hop" in forwarding table
  - Address = ~unique identifier/locator for the receiving host
- Only provides a "*I'll give it a try*" delivery service:
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order

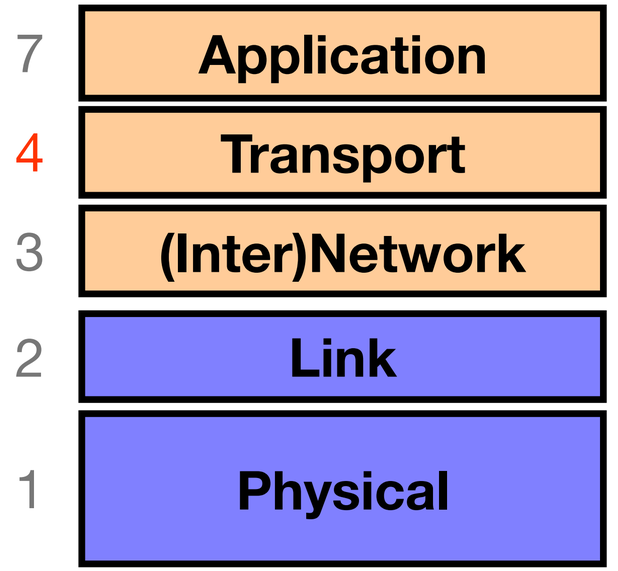


“Best Effort” is Lame! What to do?

- It’s the job of our Transport (layer 4) protocols to build services our apps need out of IP’s modest layer-3 service

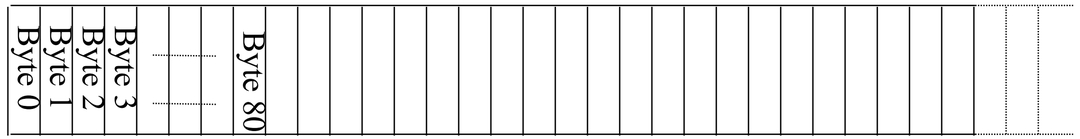
“Best Effort” is Lame! What to do?

- #1 workhorse: TCP (Transmission Control Protocol)
- Service provided by TCP:
  - Connection oriented (explicit set-up / tear-down)
    - End hosts (processes) can have multiple concurrent long-lived communication
  - **Reliable**, in-order, *byte-stream* delivery
    - Robust detection & retransmission of lost data



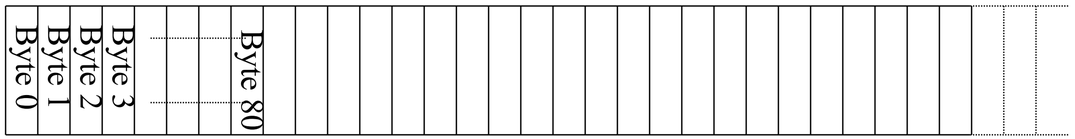
TCP “Bytestream” Service

Process A on host H1

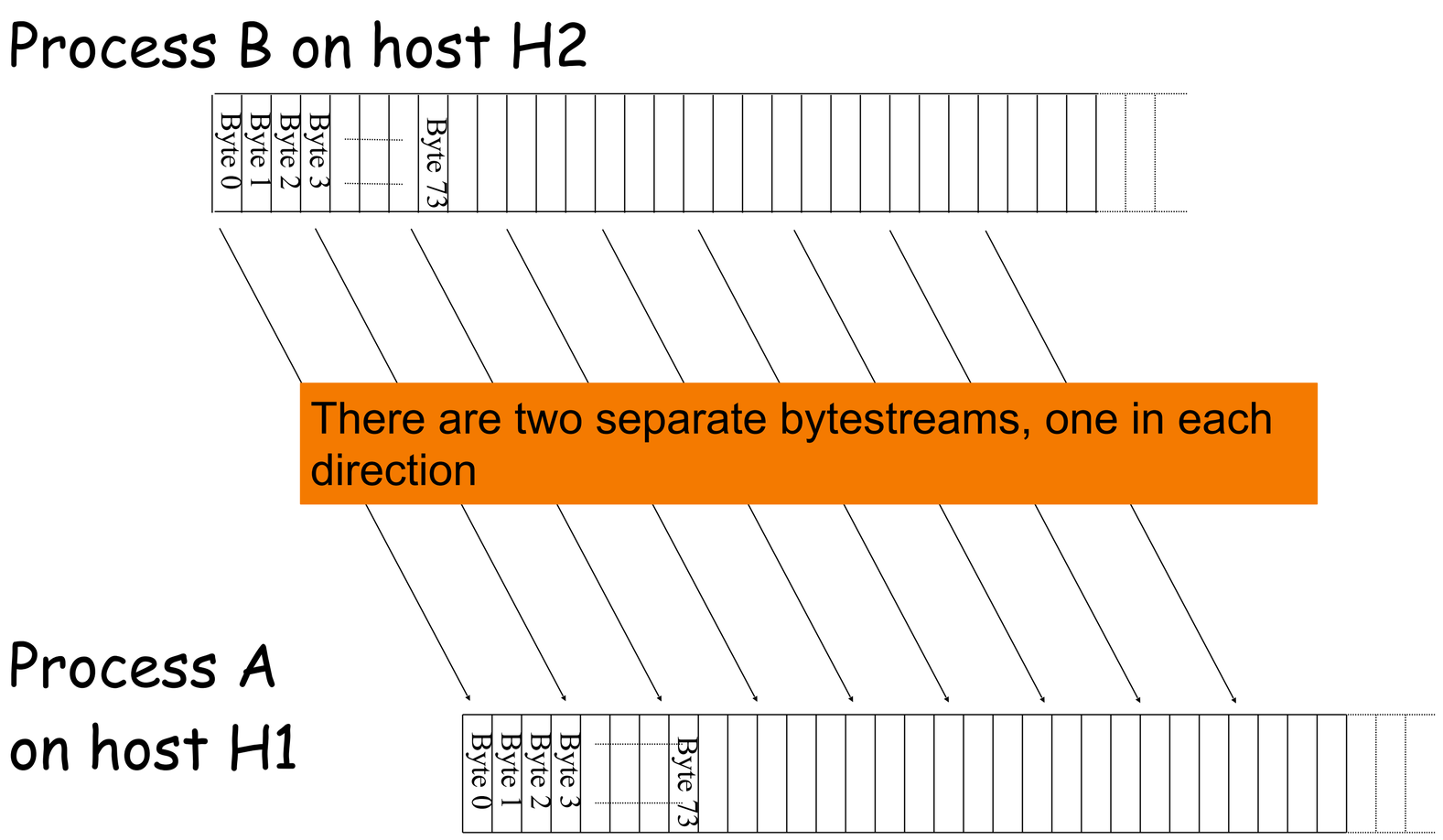


Hosts don’t ever see packet boundaries, lost or corrupted packets, retransmissions, etc.

Process B on host H2



Bidirectional communication:



Ports: Analogy

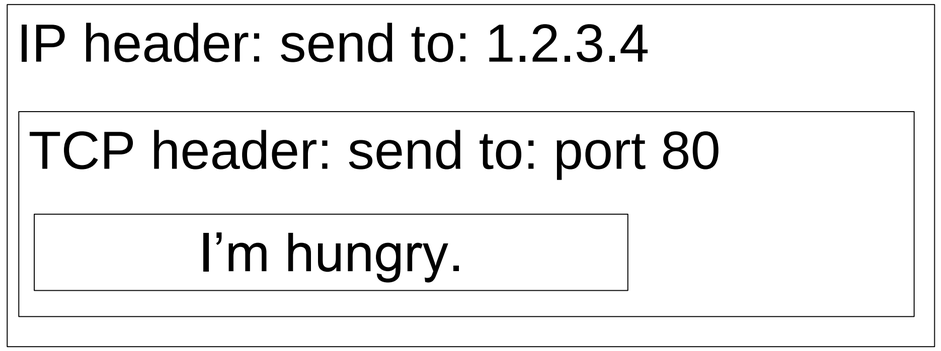
- Alice is pen pals with Carol. Alice's roommate Bob is also pen pals with Carol.
- Carol's replies are addressed to the same global (IP) address. How to tell which letters are for Bob and which are for Alice?

Ports: Analogy

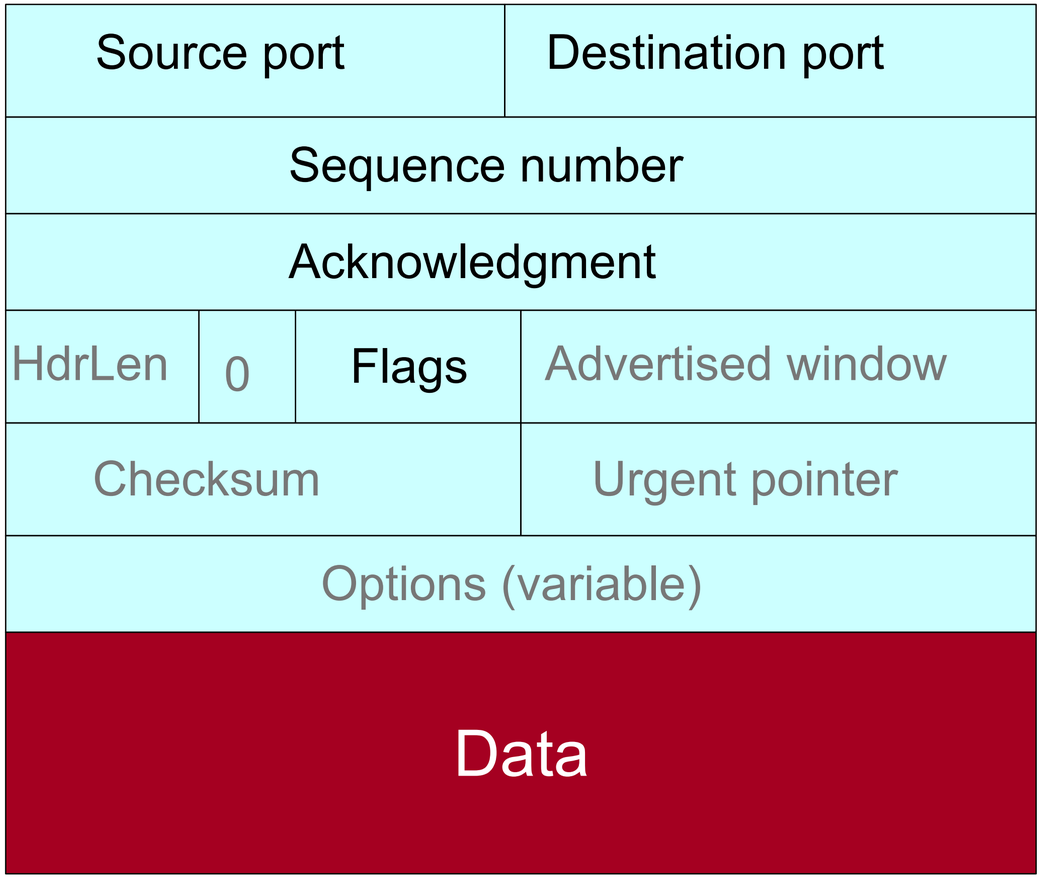
- Solution: Add a room number (port) inside the letter.
- In private homes like Alice/Bob, the port numbers are meaningless.
- In a public office (server) like Cory Hall, the port numbers are constant and known.

Ports

- Ports help us distinguish between different applications on a computer or server
- Remember: TCP is built on top of IP, so the IP address is still there

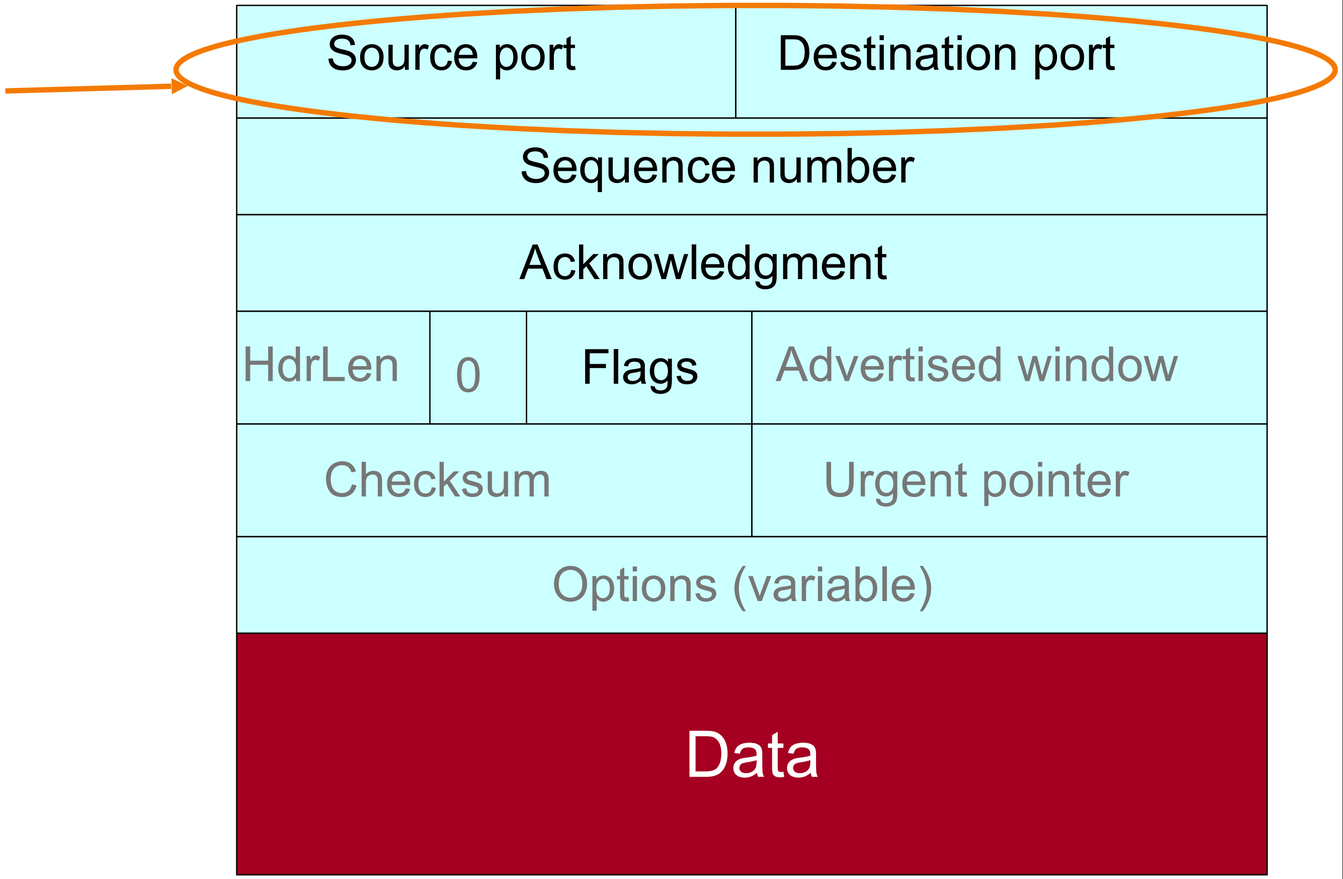


TCP Header


Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data

TCP Header

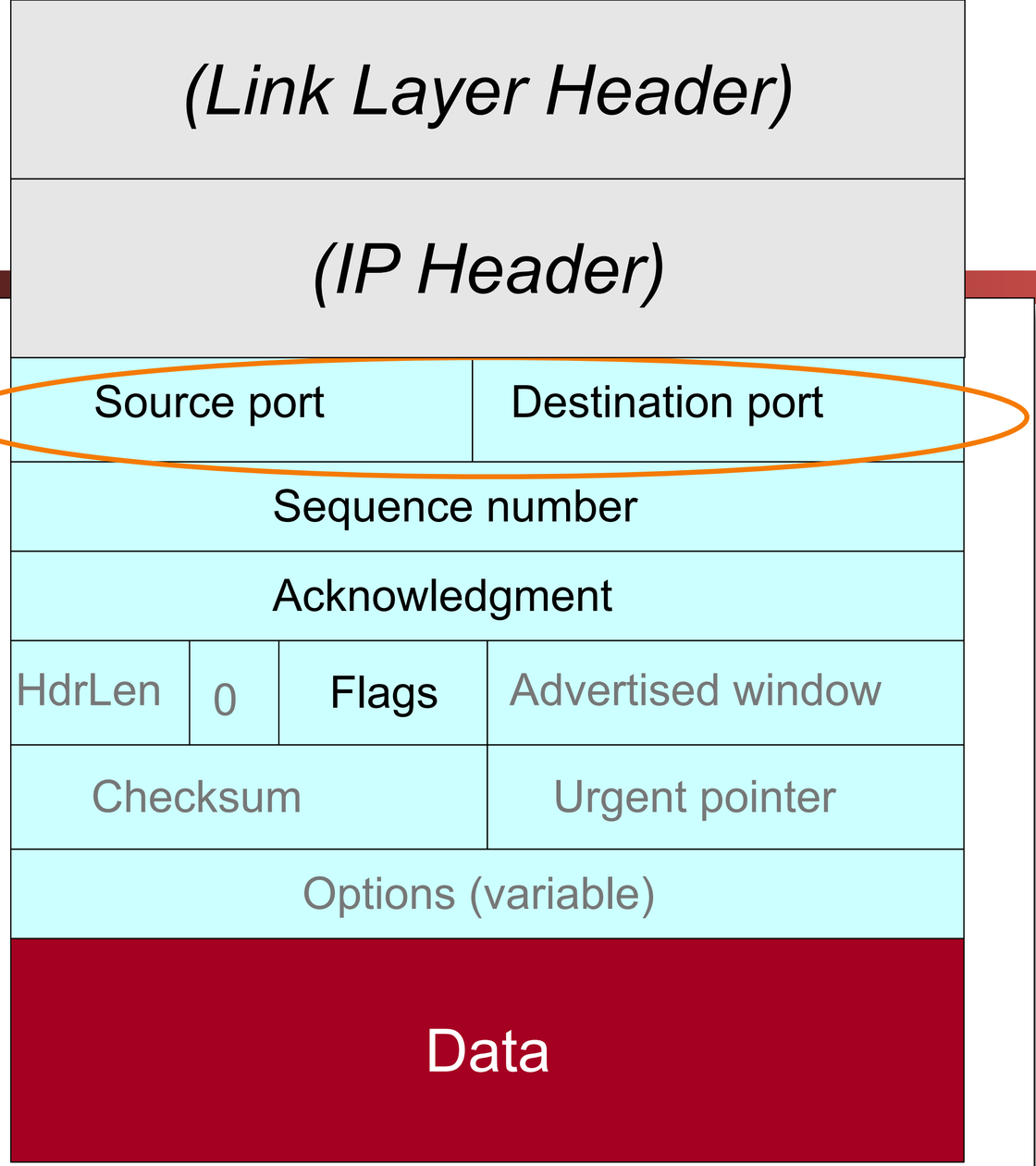
Ports are associated with OS processes



TCP Header

Ports are associated with OS processes

IP source & destination addresses plus TCP source and destination ports uniquely identifies a TCP connection

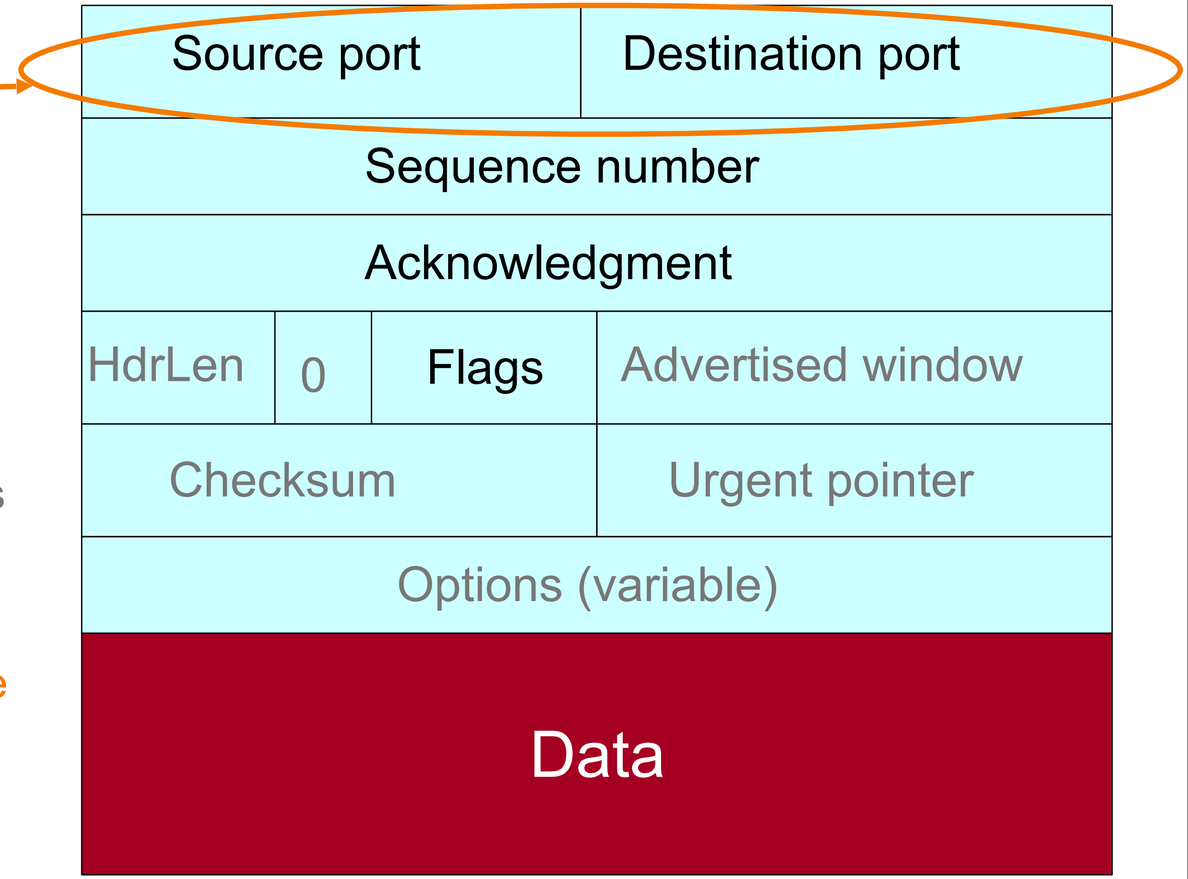


TCP Header

Ports are associated with OS processes

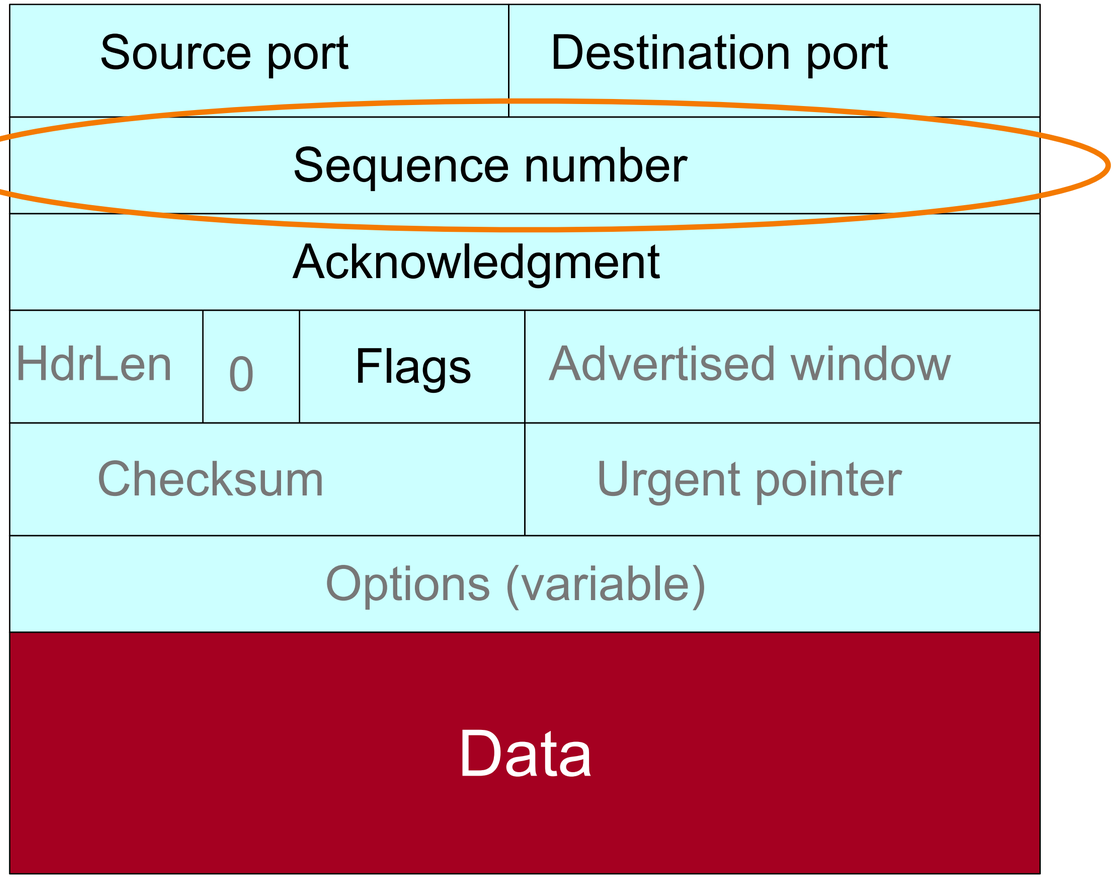
IP source & destination addresses plus TCP source and destination ports uniquely identifies a TCP connection

Some port numbers are "well known" / reserved e.g. port 80 = HTTP



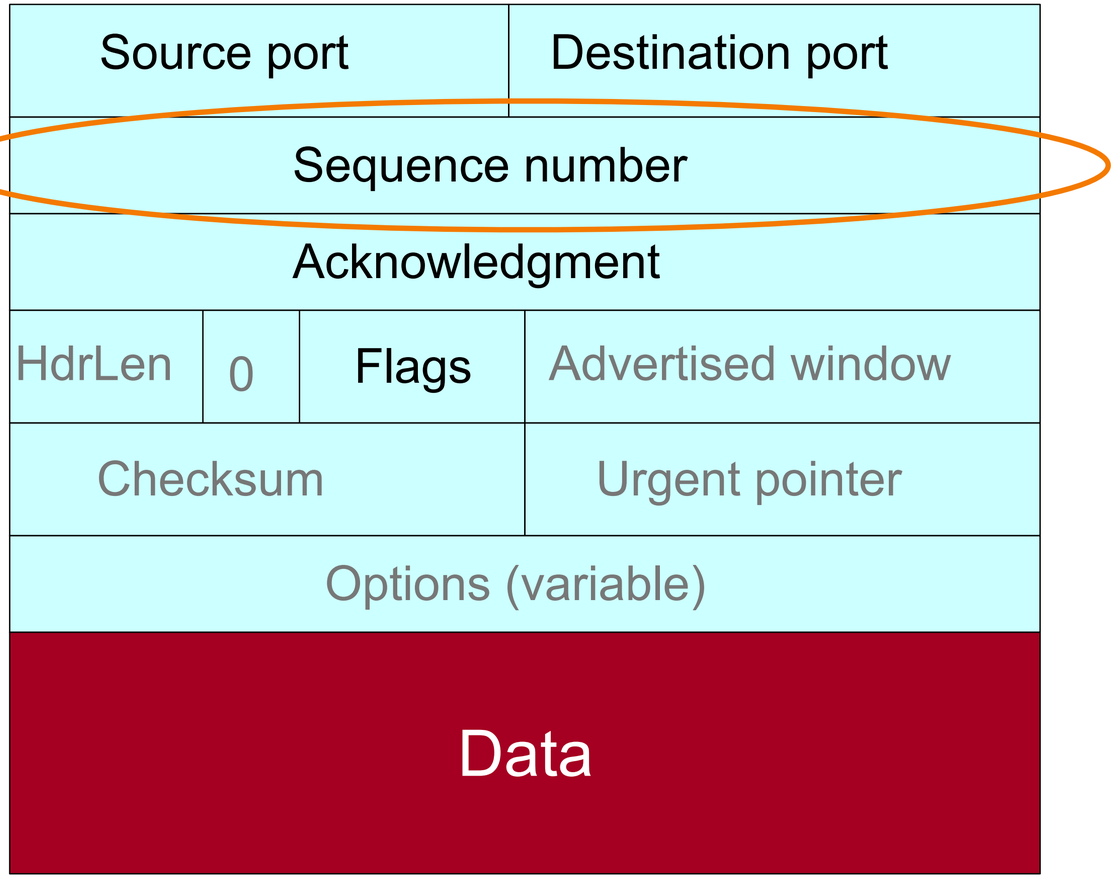
TCP Header

Starting sequence number (byte offset) of data carried in this packet



TCP Header

Starting sequence number (byte offset) of data carried in this packet

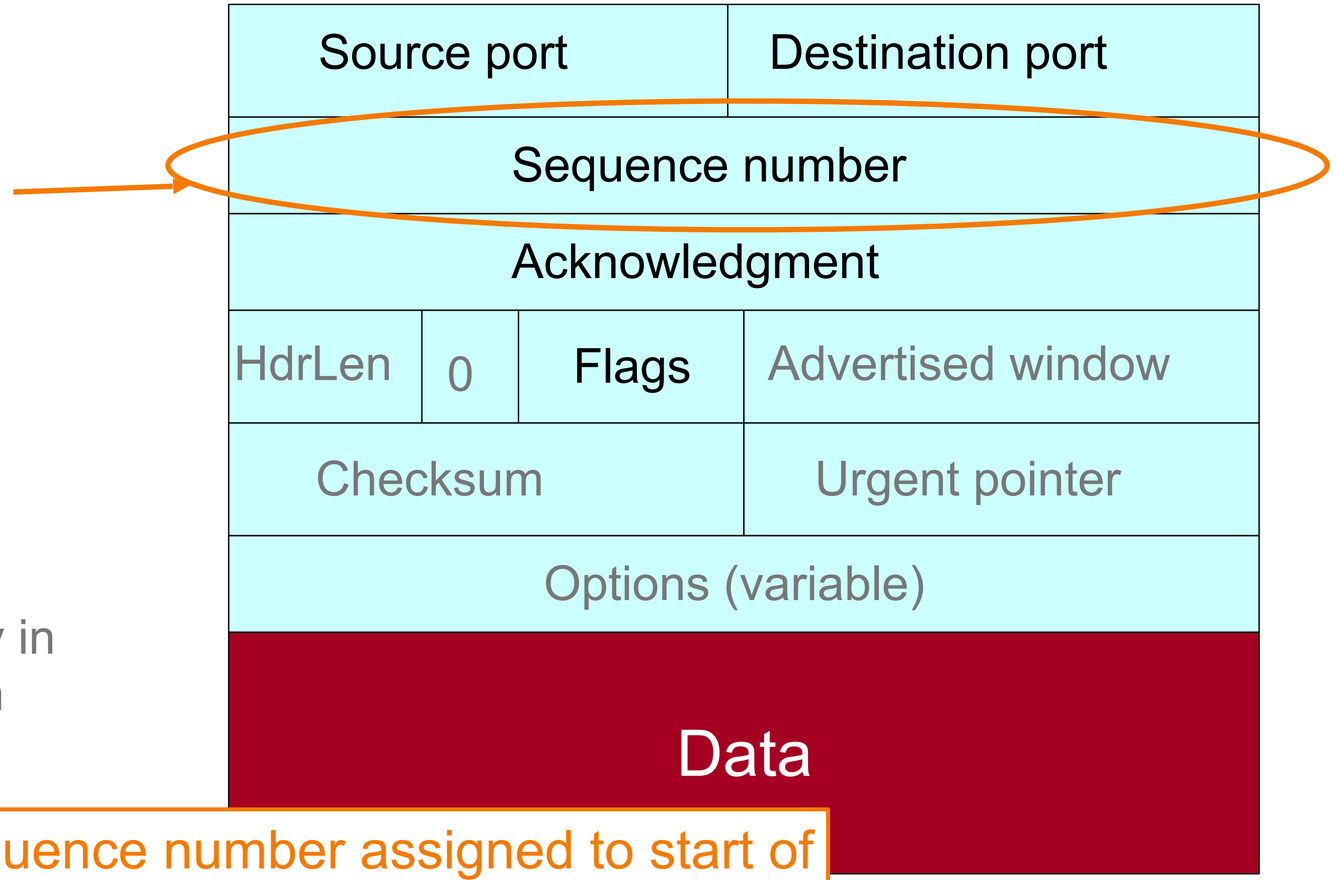


Byte streams numbered independently in each direction

TCP Header

Starting sequence number (byte offset) of data carried in this packet

Byte stream numbered independently in each direction

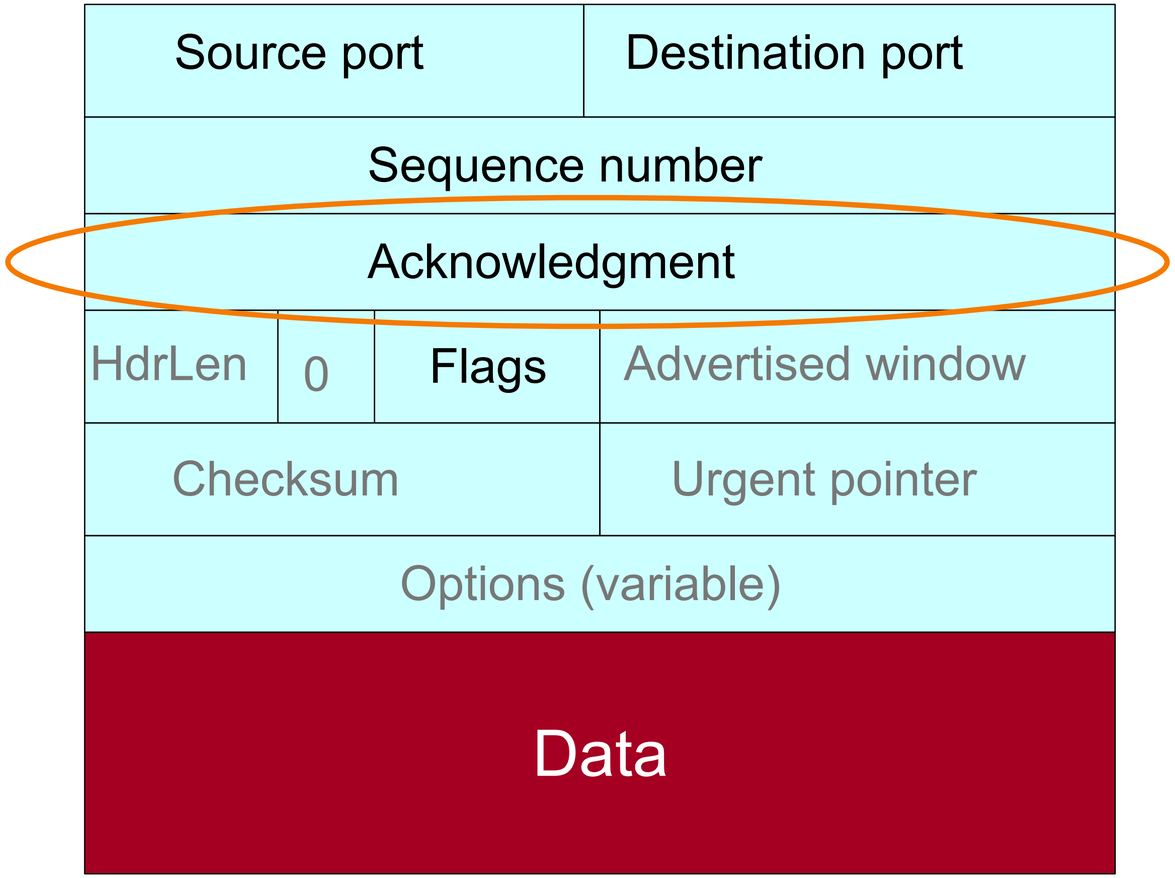


Sequence number assigned to start of byte stream is picked when connection begins; **doesn't** start at 0

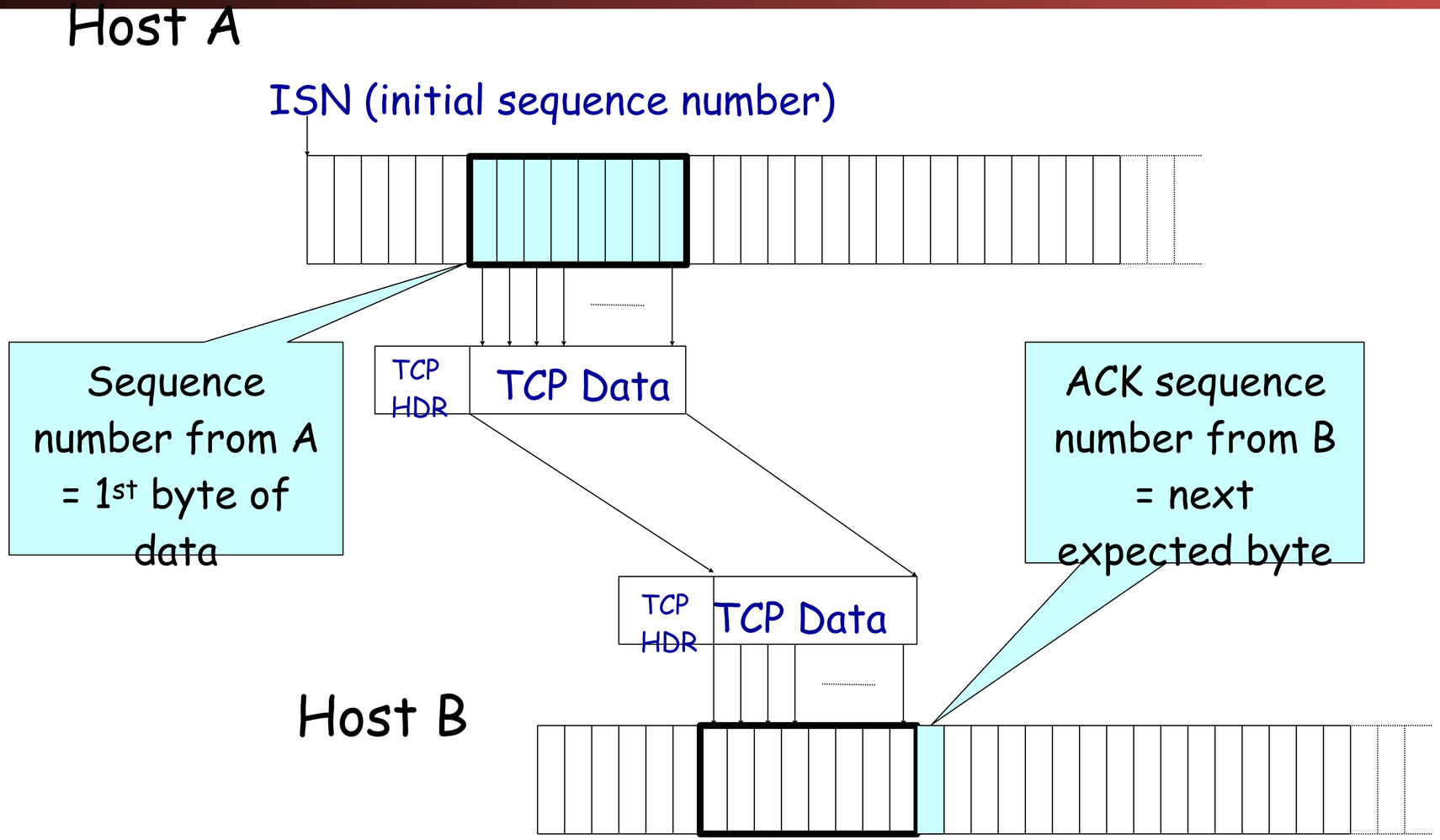
TCP Header

Acknowledgment gives seq # **just beyond** highest seq. received **in order**.

If sender sends **N** bytestream bytes starting at seq **S** then "ack" for it will be **S+N**.



Sequence Numbers

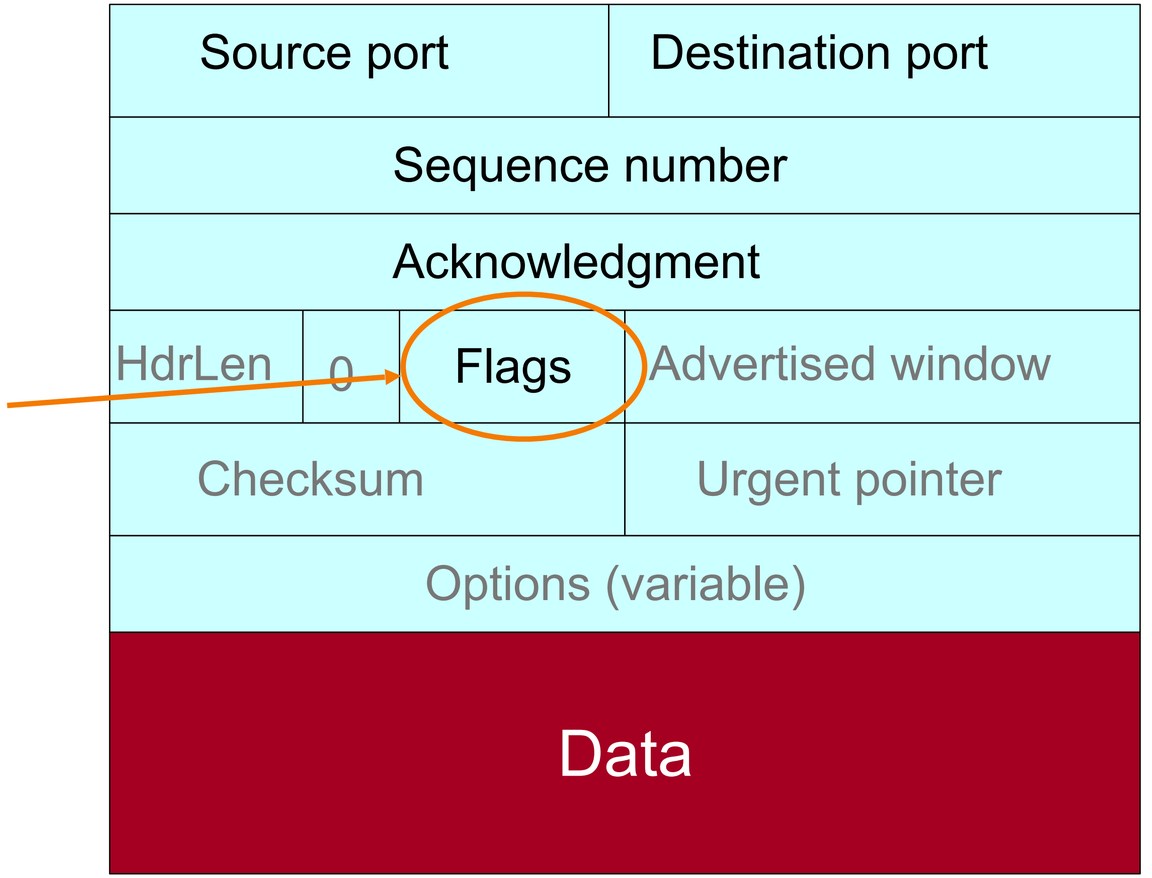


TCP Header

Uses include:

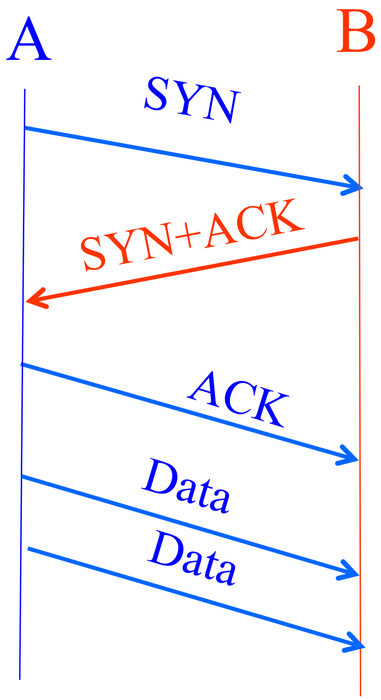
acknowledging data (“**ACK**”)

setting up (“**SYN**”) and closing connections (“**FIN**” and “**RST**”)



Establishing a TCP Connection

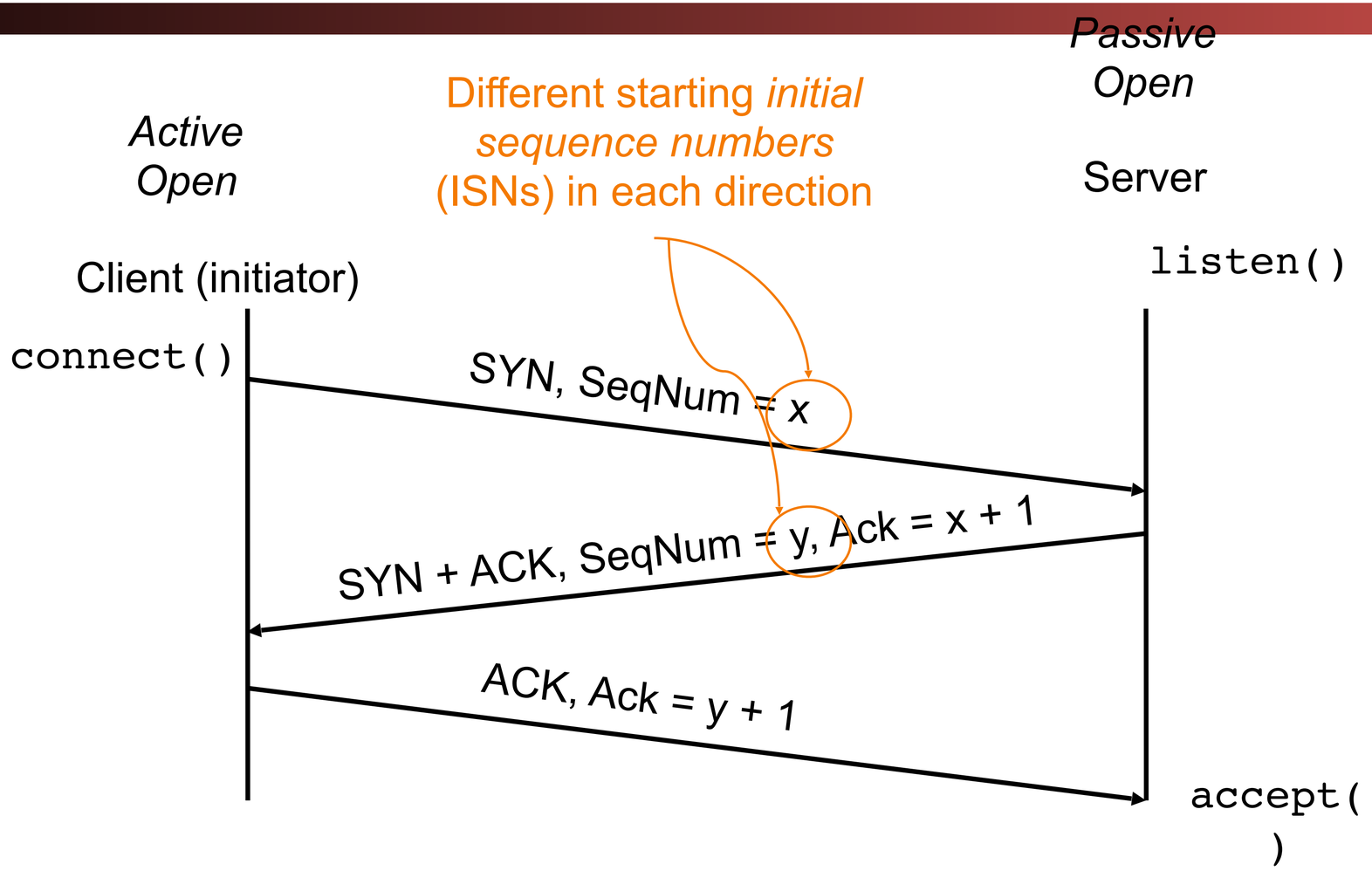
- Three-way handshake to establish connection
  - Host A sends a **`SYN`** (open; "synchronize sequence numbers") to host B
  - Host B returns a SYN acknowledgment (**`SYN+ACK`**)
  - Host A sends an **`ACK`** to acknowledge the `SYN+ACK`



Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on local clock)

Timing Diagram: 3-Way Handshaking



UDP

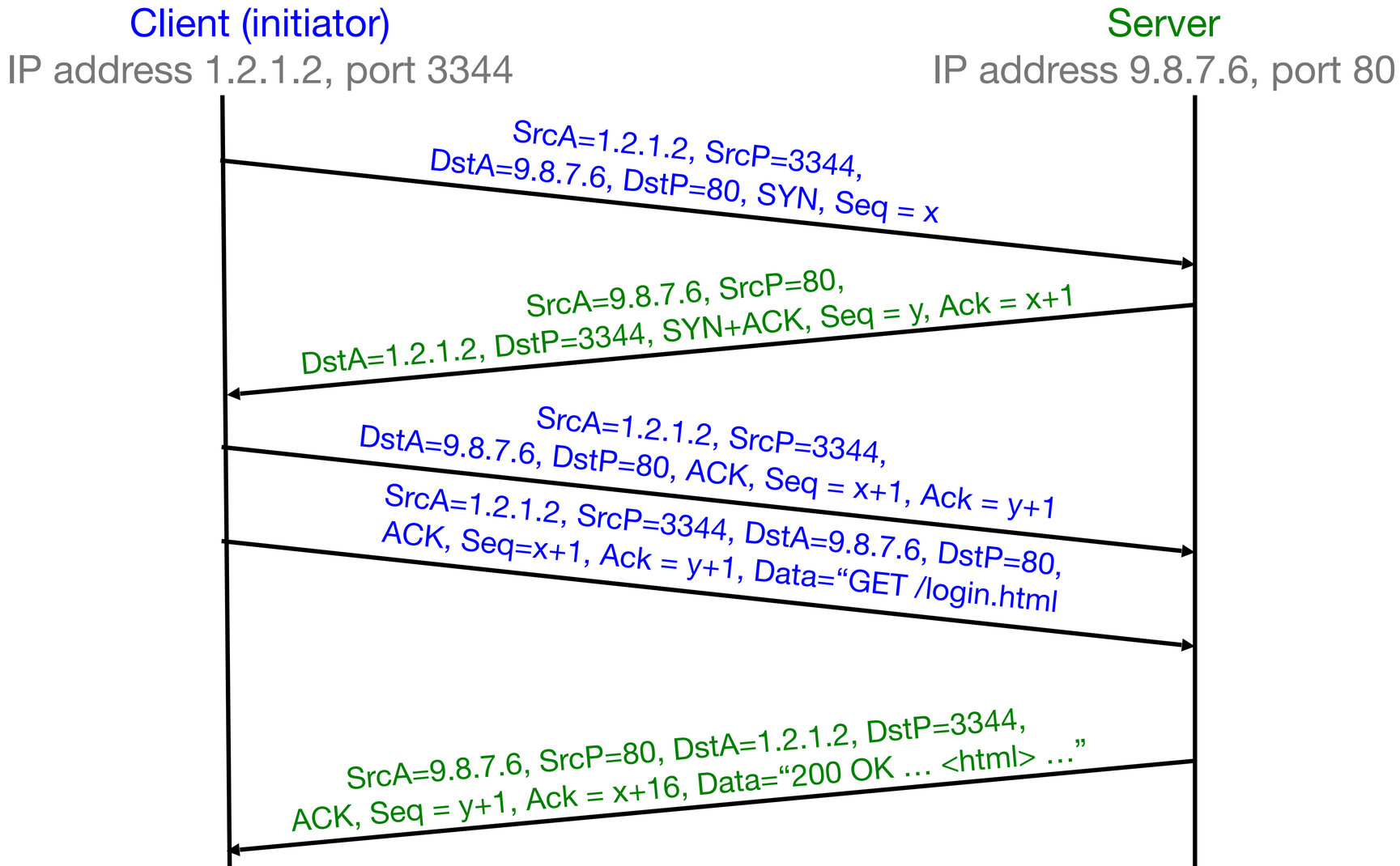
- UDP (User Datagram Protocol) is an alternative to TCP
- At the transport layer (layer 4), you have to choose TCP **or** UDP

UDP

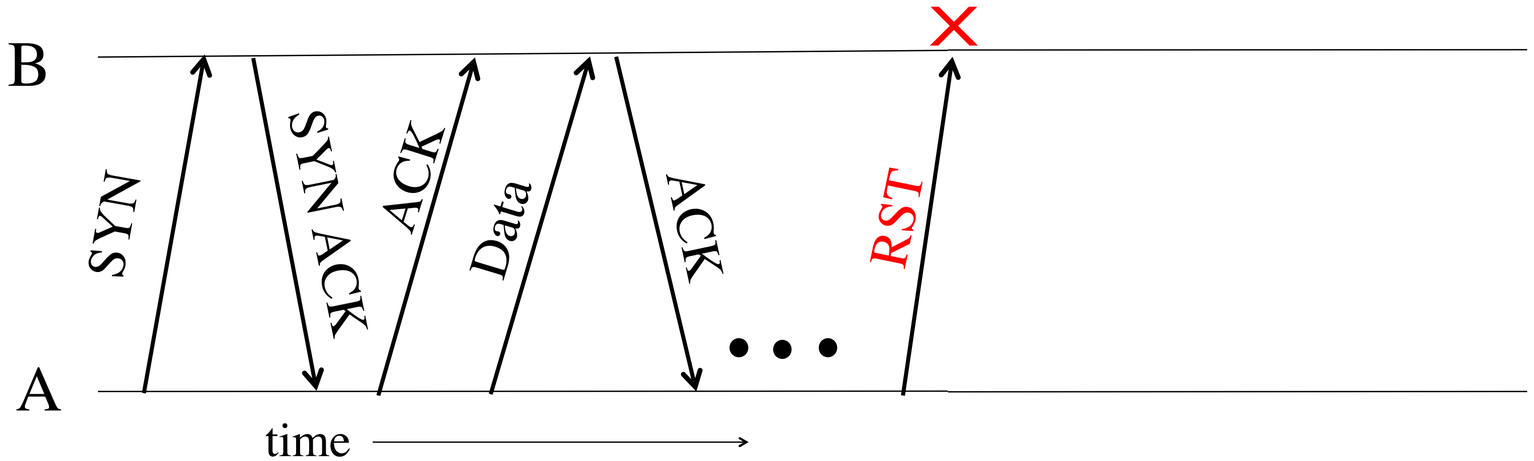
- UDP offers no reliability guarantees (still best-effort), but it adds ports
- Benefit: much faster than TCP (no handshake required)
- UDP header:

0	**16-bit source port**	**16-bit destination port**
32	16-bit length field	16-bit checksum
64	**Payload: arbitrary data**	

TCP Conn. Setup & Data Exchange



Abrupt Termination



- If A sends a TCP packet with RST flag to B and sequence number fits, connection is terminated
  - Unilateral, and takes effect immediately

TCP Threat: Disruption aka RST injection

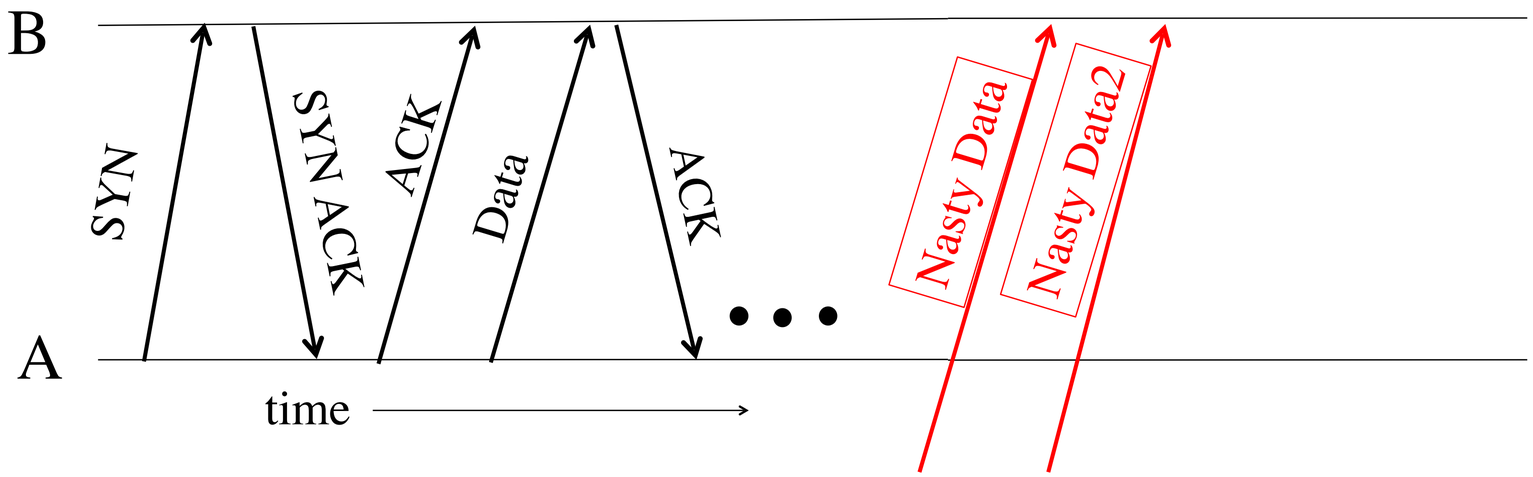
- The attacker can inject RST packets and block connection
  - TCP clients must respect RST packets and stop all communication

- Who uses this?
  - China: The Great Firewall does this to TCP requests
  - A long time ago: Comcast, to block BitTorrent uploads
  - Some intrusion detection systems: To hopefully mitigate an attack in progress

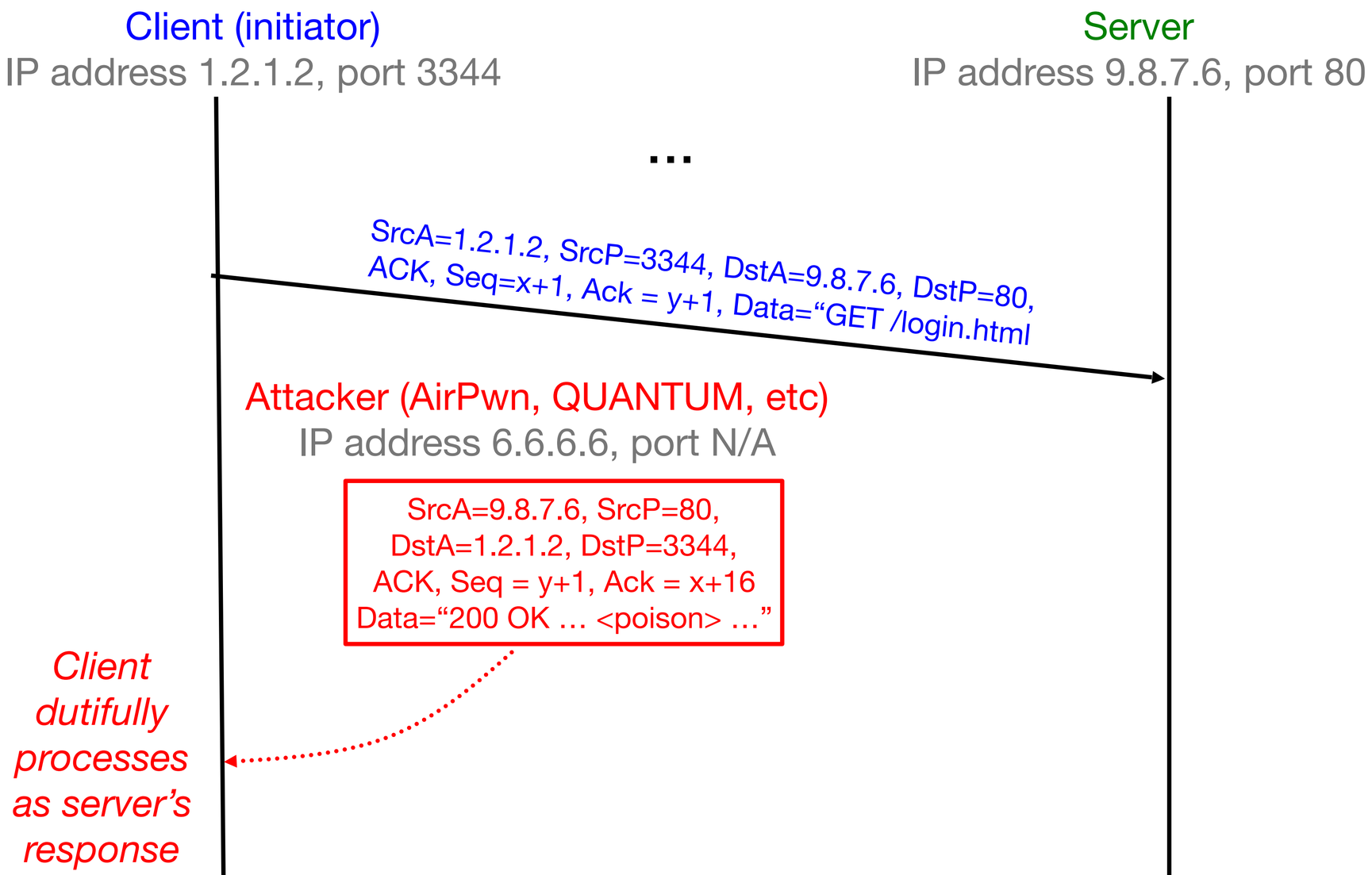
Discuss with a partner: Who can do RST injection? (a) off-path attacker, (b) on-path attacker, (c) man-in-the-middle

TCP Threat: Data Injection

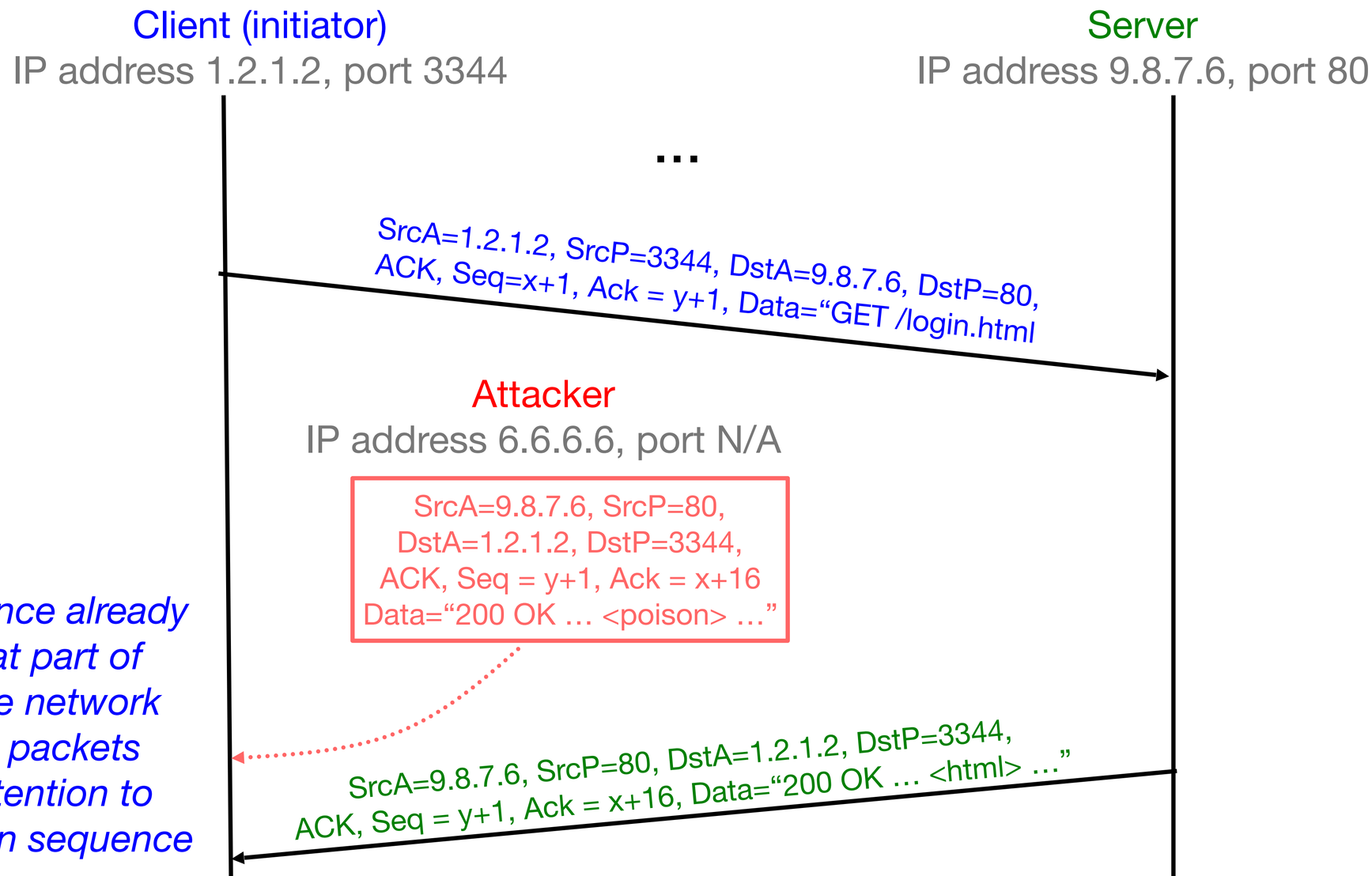
- If attacker knows ports & sequence numbers (e.g., on-path attacker), attacker can inject data into any TCP connection
  - Receiver B is *none the wiser!*
- Termed TCP connection hijacking (or "*session hijacking*")
  - A general means to take over an already-established connection!
- We are toast if an attacker can see our TCP traffic!
  - Because then they immediately know the port & sequence numbers



TCP Data Injection



TCP Data Injection



Client ignores since already processed that part of bytestream: the network can duplicate packets so only pay attention to the first version in sequence

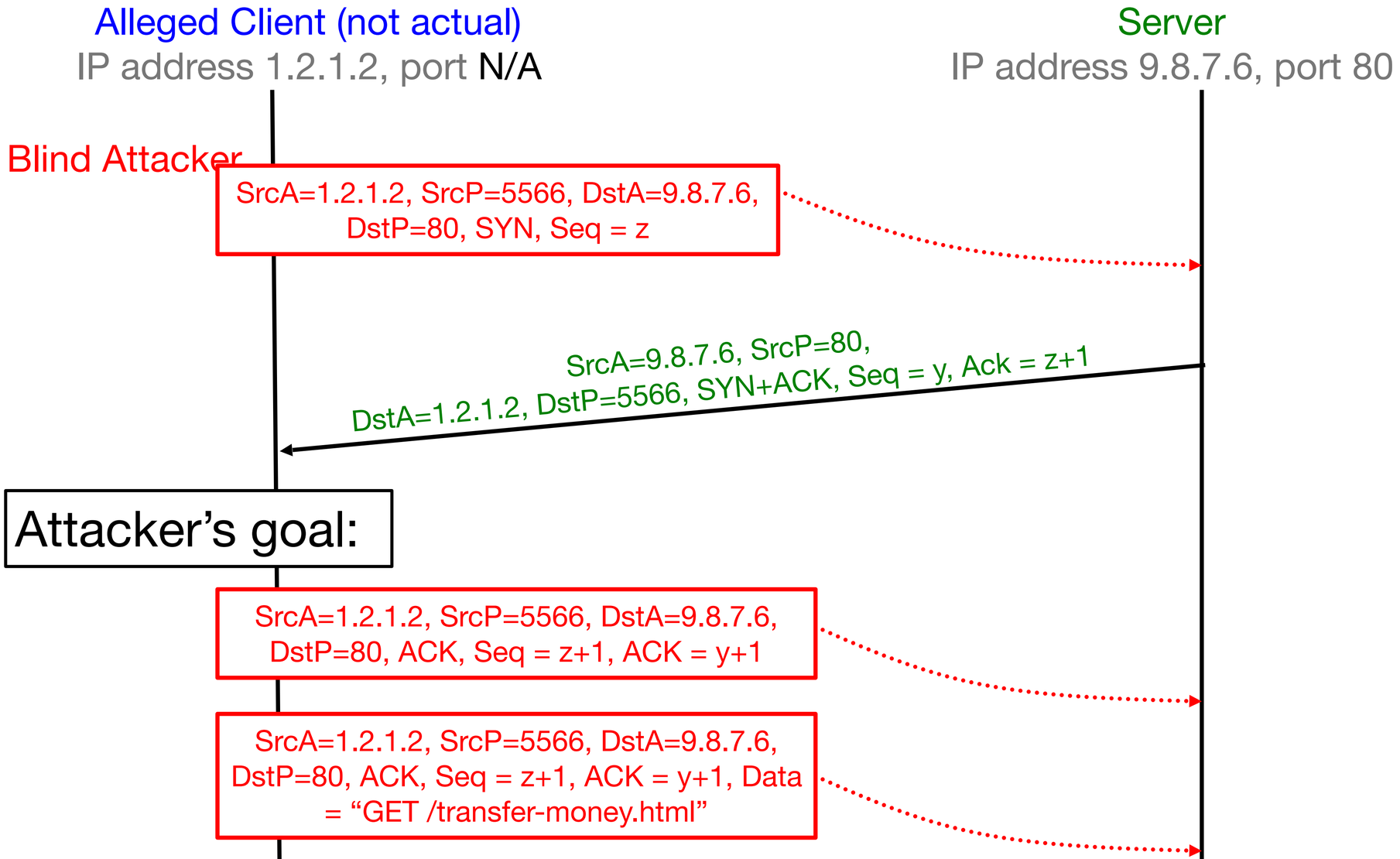
TCP Threat: Blind Hijacking

- Is it possible for an off-path attacker to inject into a TCP connection even if they can’t see our traffic?
- YES: if somehow they can infer or guess the port and sequence numbers

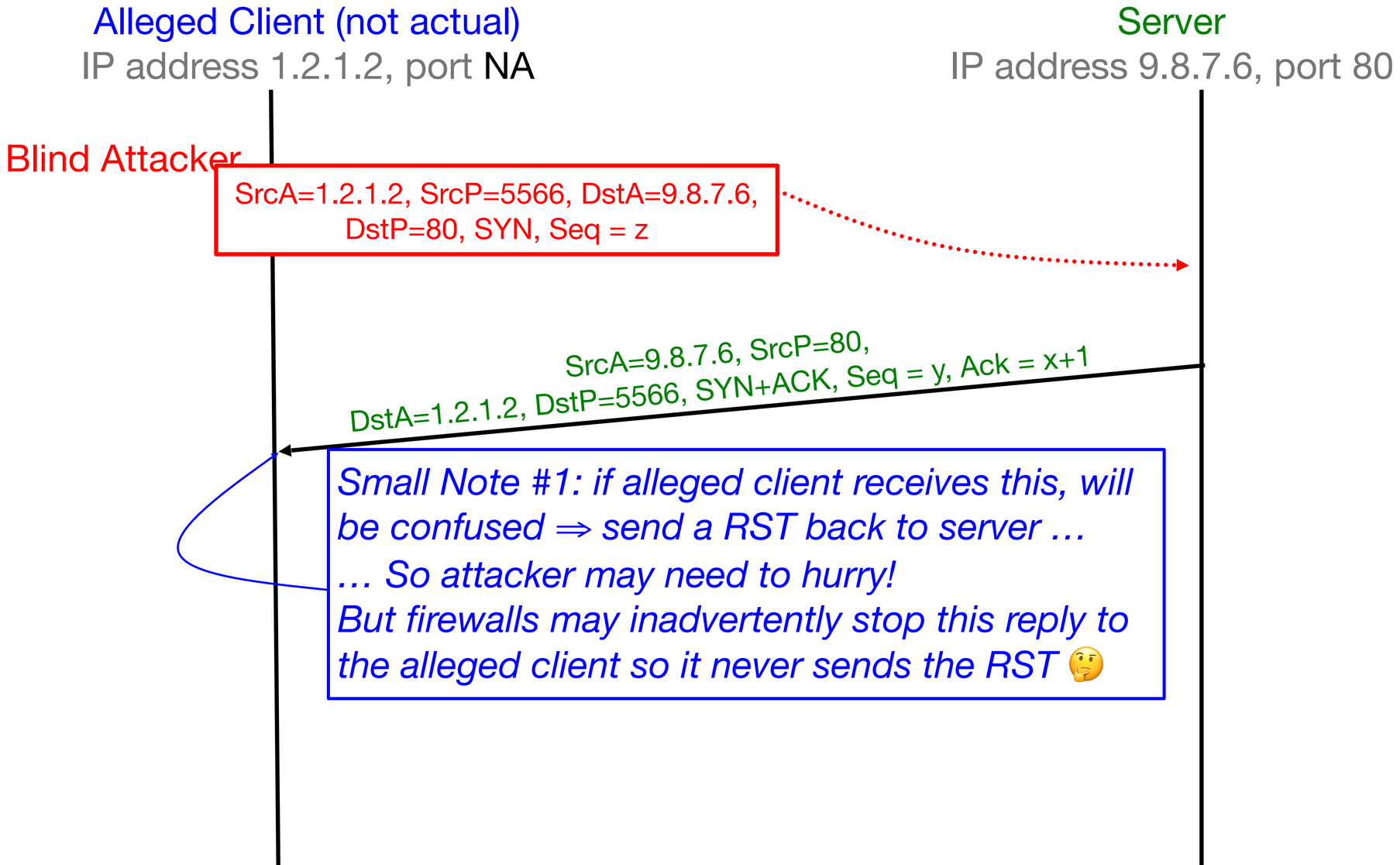
TCP Threat: Blind Spoofing

- Is it possible for an off-path attacker to create a fake TCP connection, even if they can't see responses?
- Yes if somehow they can infer or guess the TCP initial sequence numbers
- Why would an attacker want to do this?
  - Perhaps to leverage a server's trust of a given client as identified by its IP address
  - Perhaps to frame a given client so the attacker's actions during the connections can't be traced back to the attacker

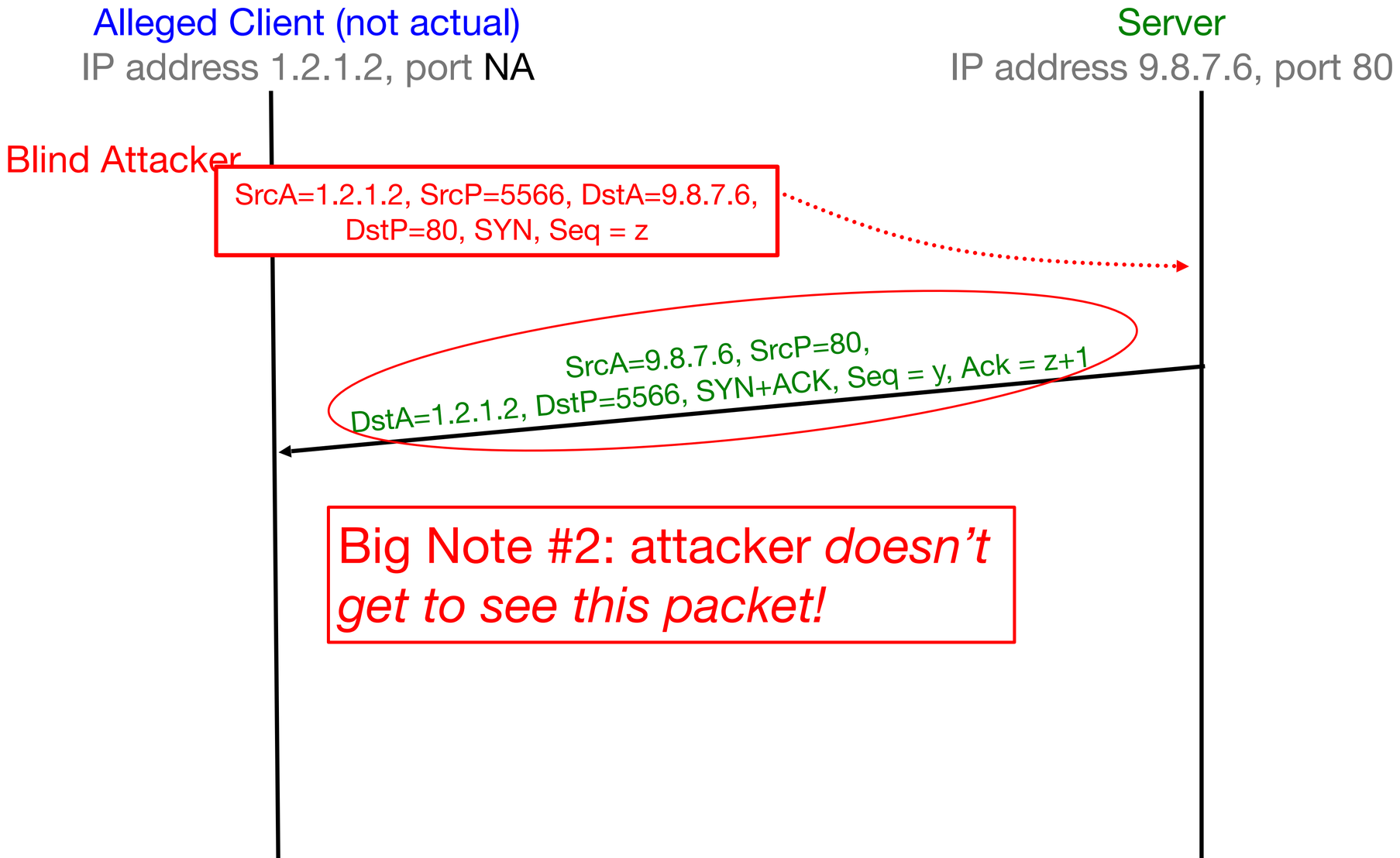
Blind Spoofing on TCP Handshake



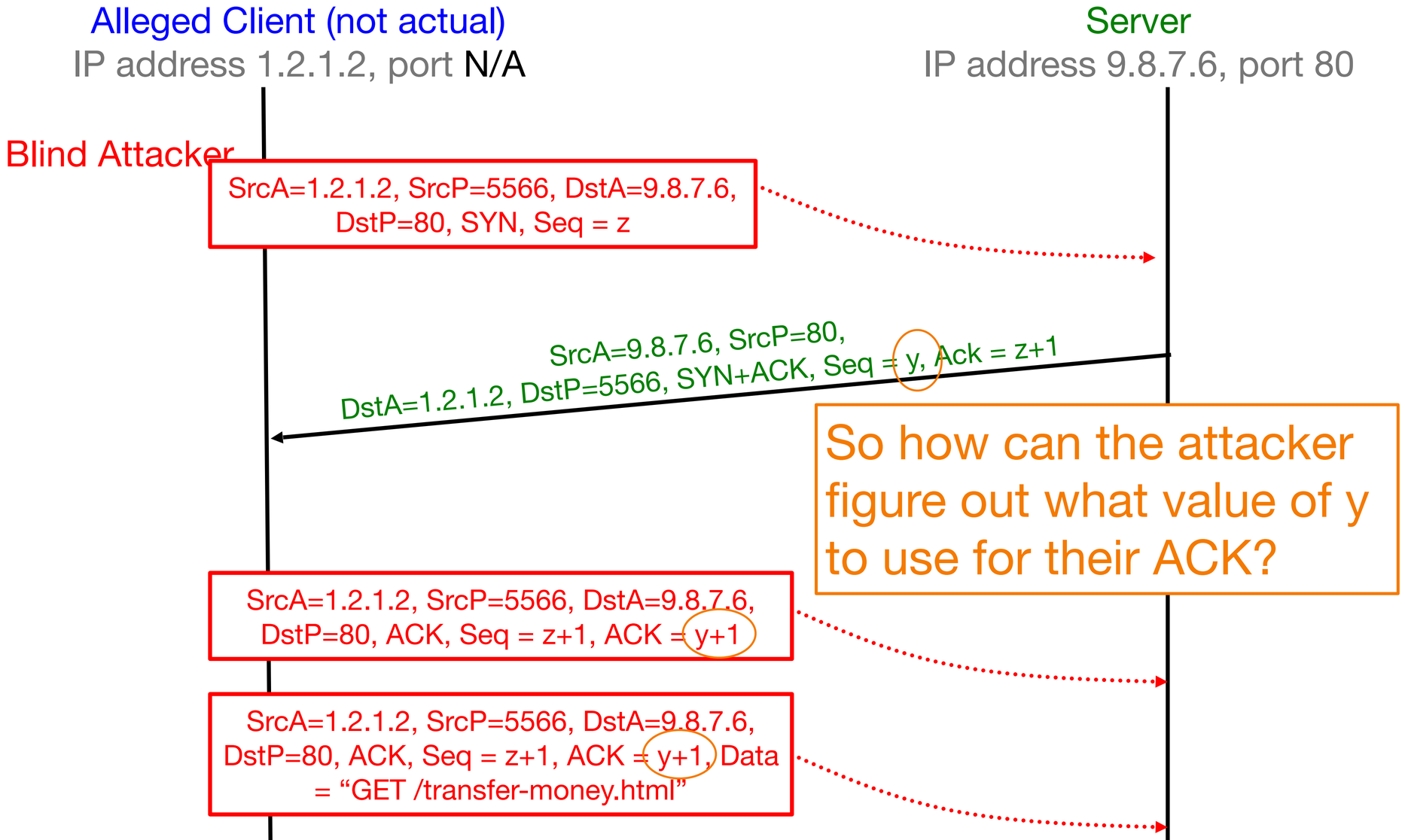
Blind Spoofing on TCP Handshake



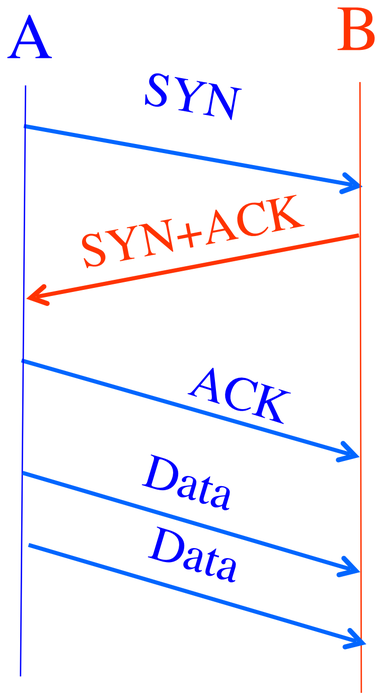
Blind Spoofing on TCP Handshake



Blind Spoofing on TCP Handshake



Reminder: Establishing a TCP Connection



Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on local clock)

Hmm, any way for the attacker to know *this?*

How Do We Fix This?

Use a (Pseudo)-*Random* ISN

Sure – make a non-spoofed connection *first*, and see what server used for ISN y then!

Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully terminate by forging a RST packet
  - Inject (spoof) data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - Remains a major threat today
- Blind spoofing no longer a threat
  - Due to randomization of TCP initial sequence numbers

Ghost of blind spoofing...

- CVE-2016-5696
  - "Off-Path TCP Exploits: Global Rate Limit Considered Dangerous" Usenix Security 2016 https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cao
- Key idea:
  - RFC 5961 added some global rate limits that acted as an ***information leak***:
    - Could determine if two hosts were communicating on a given port
    - Could determine if your guess at the sequence number is “in window”
  - Once you get the sequence #s, you can then inject arbitrary content into the TCP stream
- Fixed today