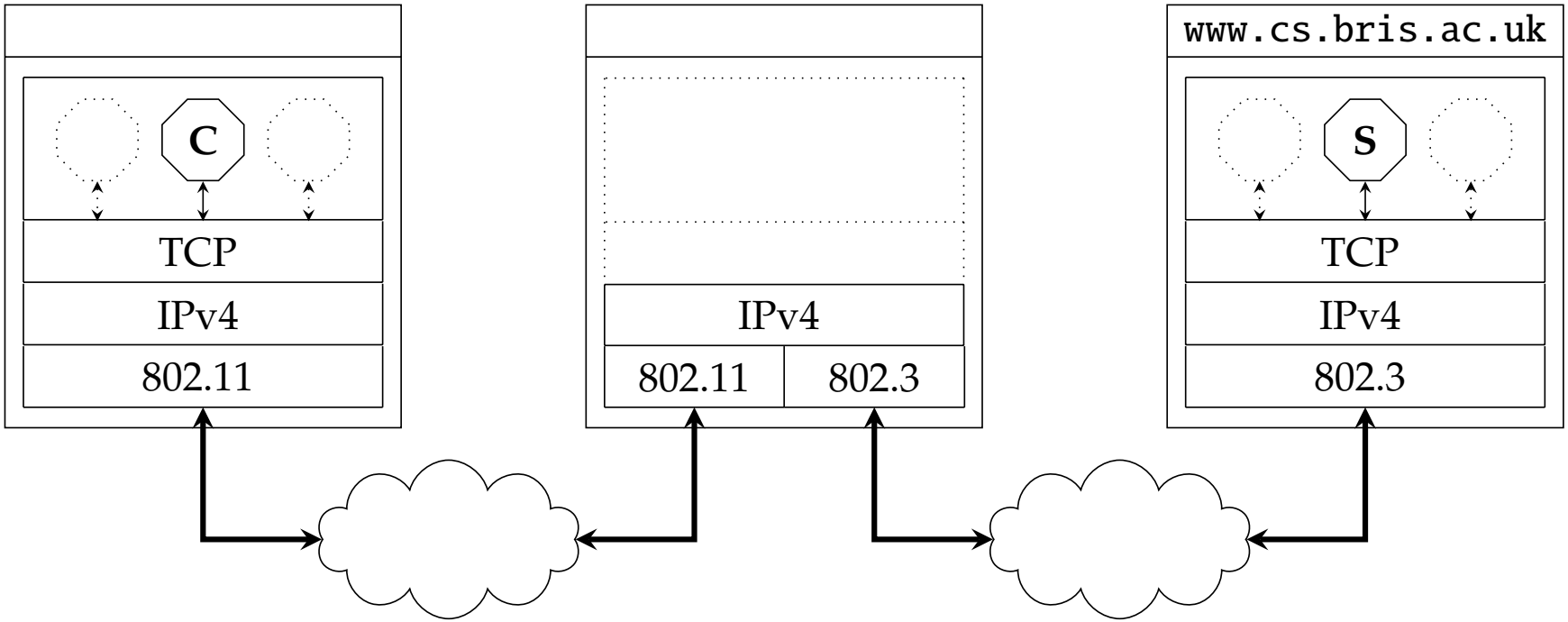
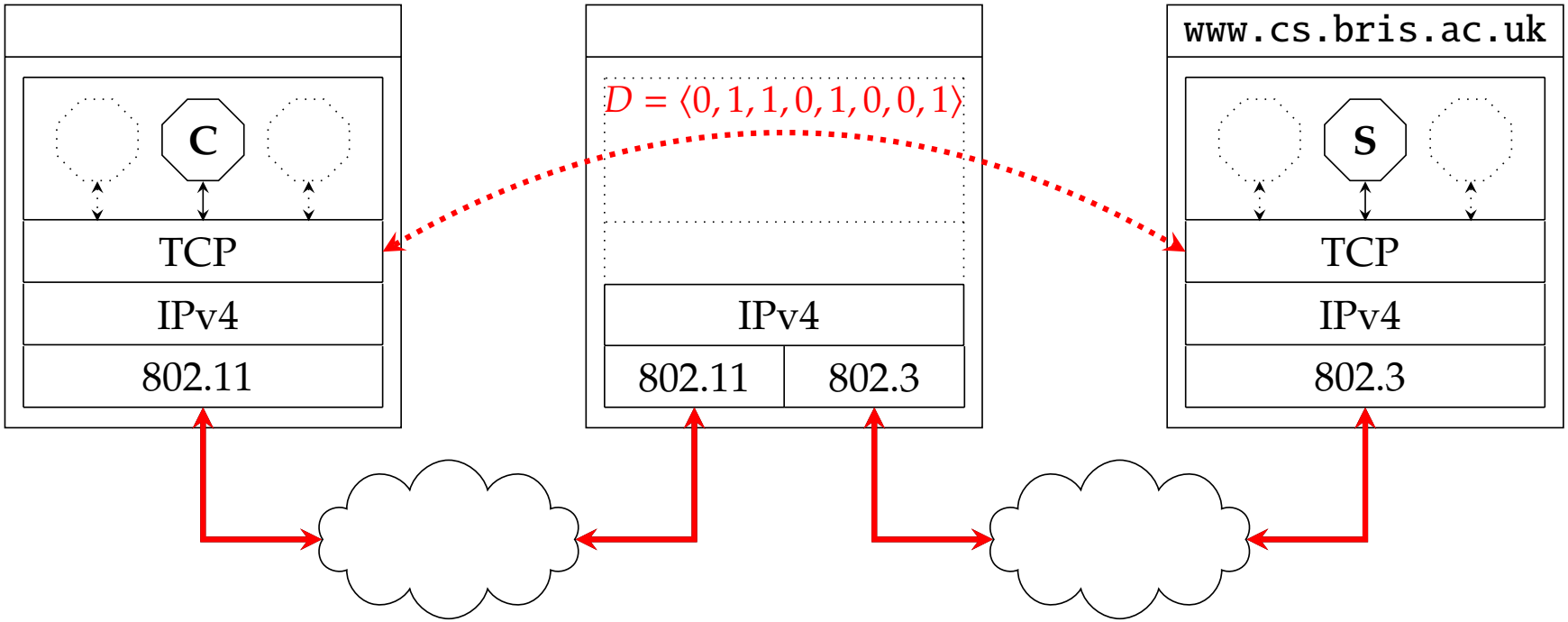


- Goal: investigate the **transport layer** e.g.,
  1. connection management,
  2. error, flow and congestion control

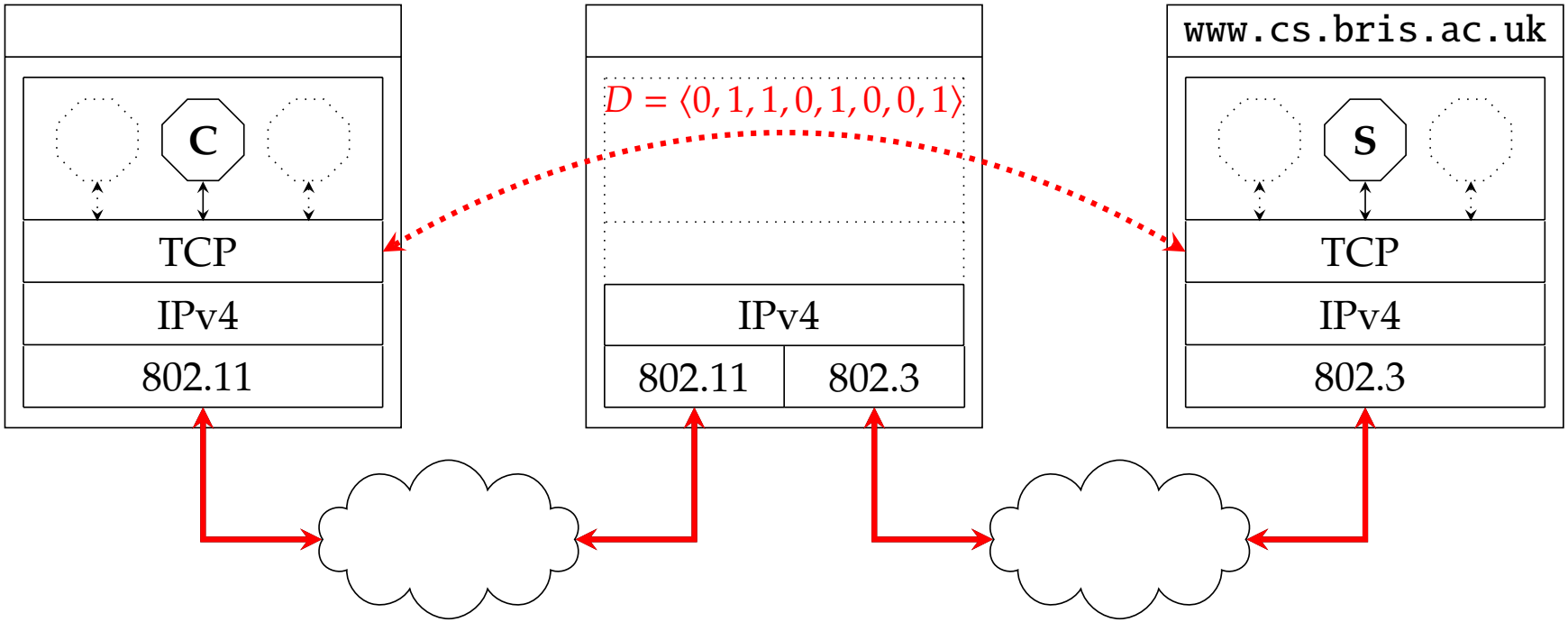
  st. applications can use end-to-end, **datagram**- or **stream**-based communication (by transmitting **datagrams** or **segments** respectively).



$F = H_{802.11} \parallel H_{IPv4} \parallel H_{TCP} \parallel \langle 0, 1, 1, 0, 1, 0, 0, 1 \rangle \parallel T_{802.11}$

- Goal: investigate the **transport layer** e.g.,
  1. connection management,
  2. error, flow and congestion control

  st. applications can use end-to-end, **datagram**- or **stream**-based communication (by transmitting **datagrams** or **segments** respectively).



$F = H_{802.11} \parallel H_{IPv4} \parallel H_{TCP} \parallel \langle 0, 1, 1, 0, 1, 0, 0, 1 \rangle \parallel T_{802.11}$

- Note that:
  - the transport layer need *only* be implemented by in hosts; routers *only* deal with network (and lower) layers,
  - this implies the transport layer offers an abstraction of the network itself, i.e., a **service model** (or API) ...

- Question: *which* service model should we opt for?
- Answer: it depends ...

Definition (datagram model)

For instance, **User Datagram Protocol (UDP)** [6] is

1. datagram (or message) oriented,
2. connection-less, unreliable,
3. allows unrestricted transmission,
4. limited length segment,
5. relatively simple, relatively efficient,
6. (mainly) used for stateless applications (e.g., DNS).

Definition (stream model)

For instance, **Transport Control Protocol (TCP)** [7] is

1. stream oriented,
2. connection-based, reliable,
3. applies flow and congestion control,
4. arbitrary length segment,
5. relatively complex, relatively inefficient,
6. (mainly) used for stateful applications (e.g., HTTP).

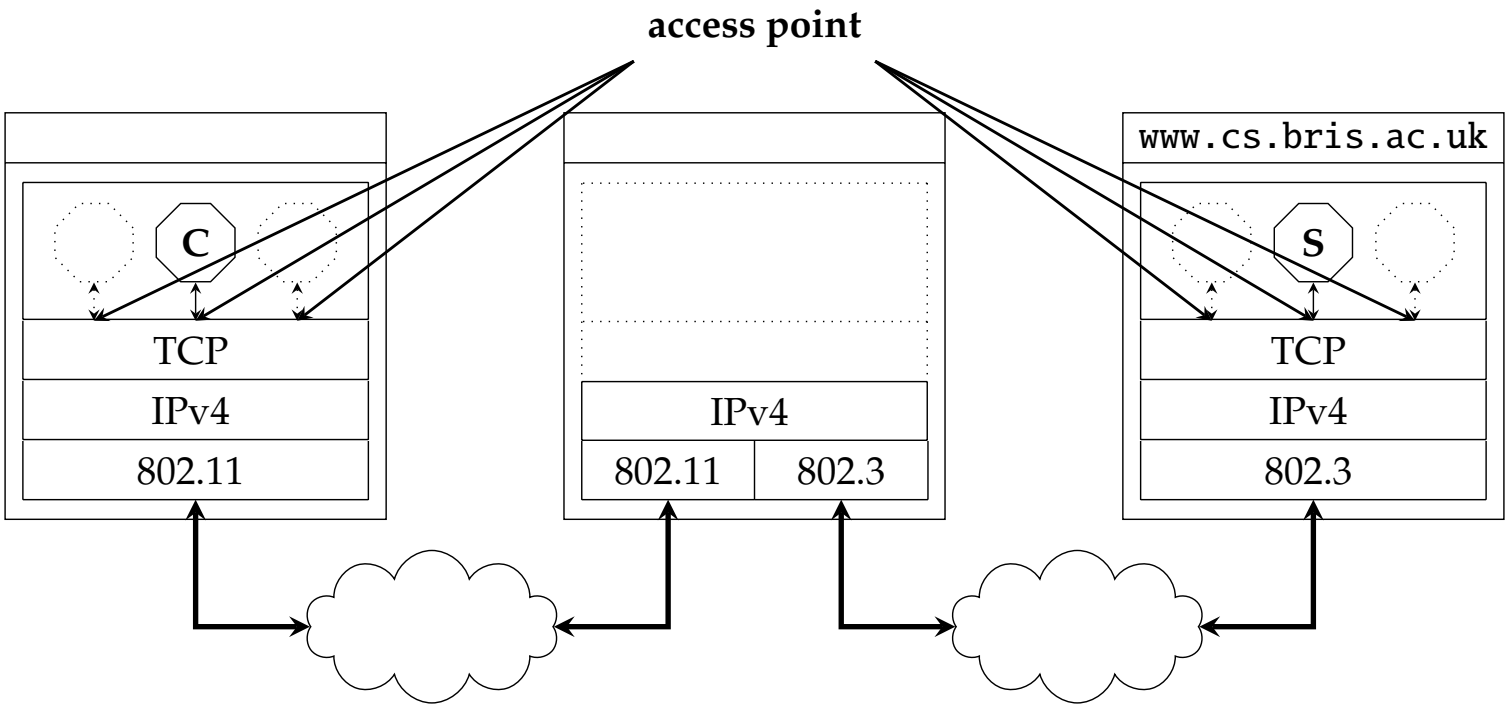
st. $\text{TCP} \gg \text{UDP} \simeq \text{IP} + \epsilon$.

Definition (**ports and sockets**)

Each transport layer **access point** is identified by a (local) **port number**; the tuple

(IP address, protocol, port number),

which is called a **socket**, therefore (globally) identifies the communication end-point (i.e., the application).



Definition (**ports and sockets**)

Note that

- a **socket pair** canonically identifies one connection,
- a **well-known port** is statically (pre-)allocated for specific (often system) applications (e.g., server instances),
- an **ephemeral port** is dynamically allocated for applications (e.g., client instances).

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

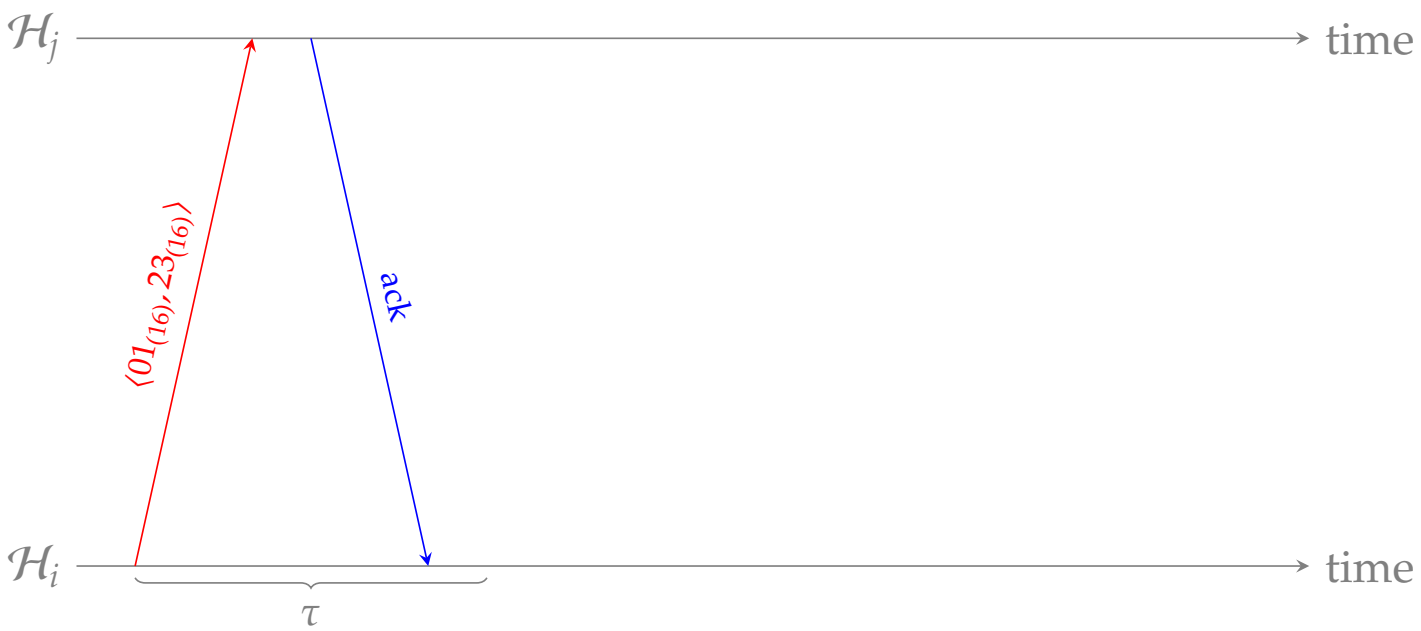
e.g., using the **stop-and-wait protocol** .

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

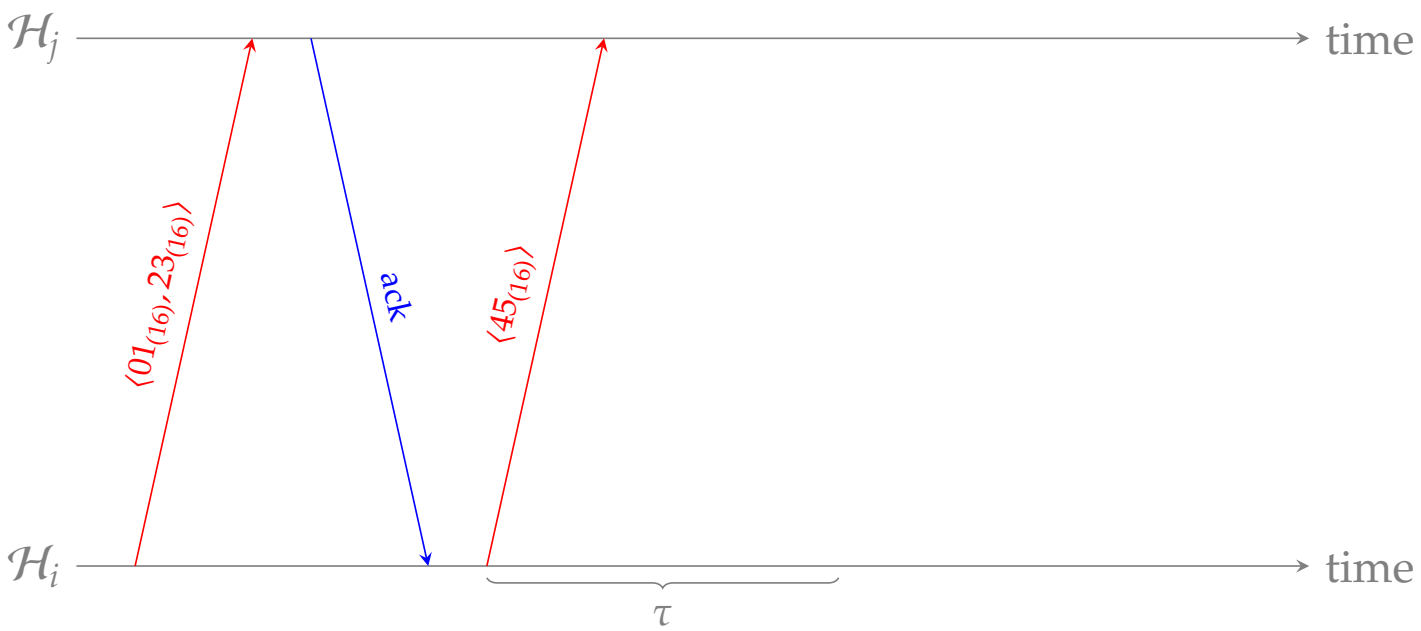


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

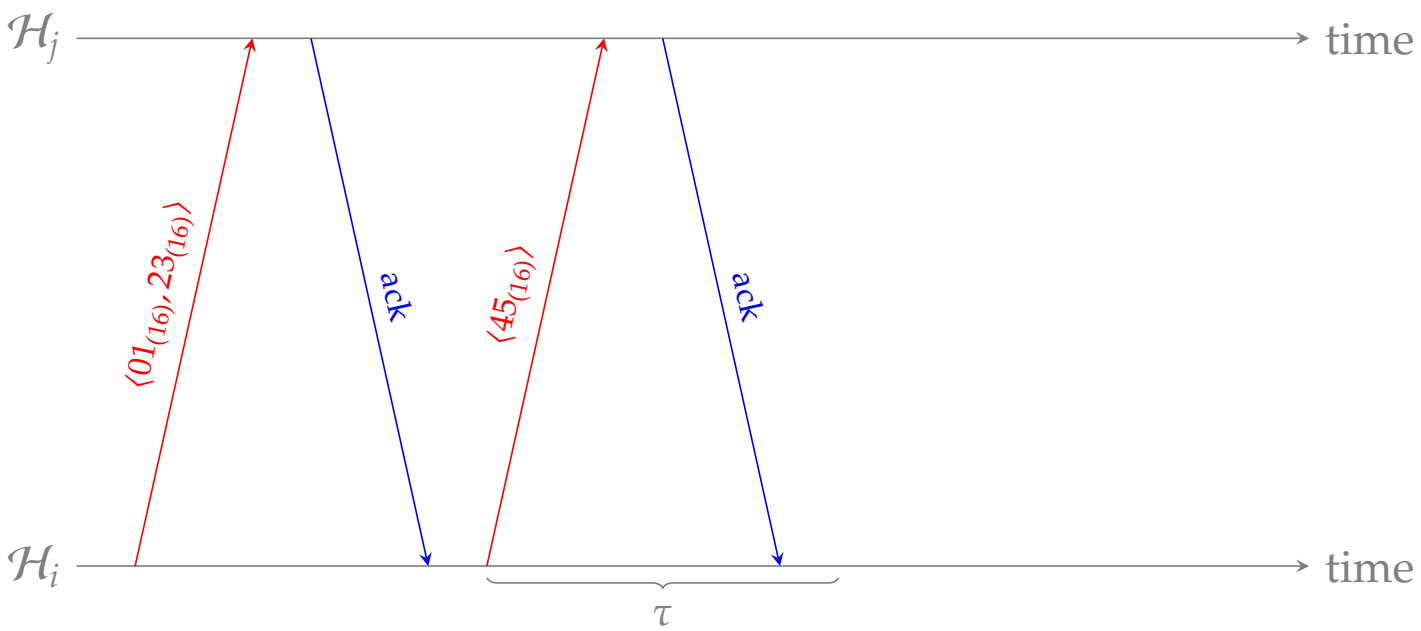


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .



Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

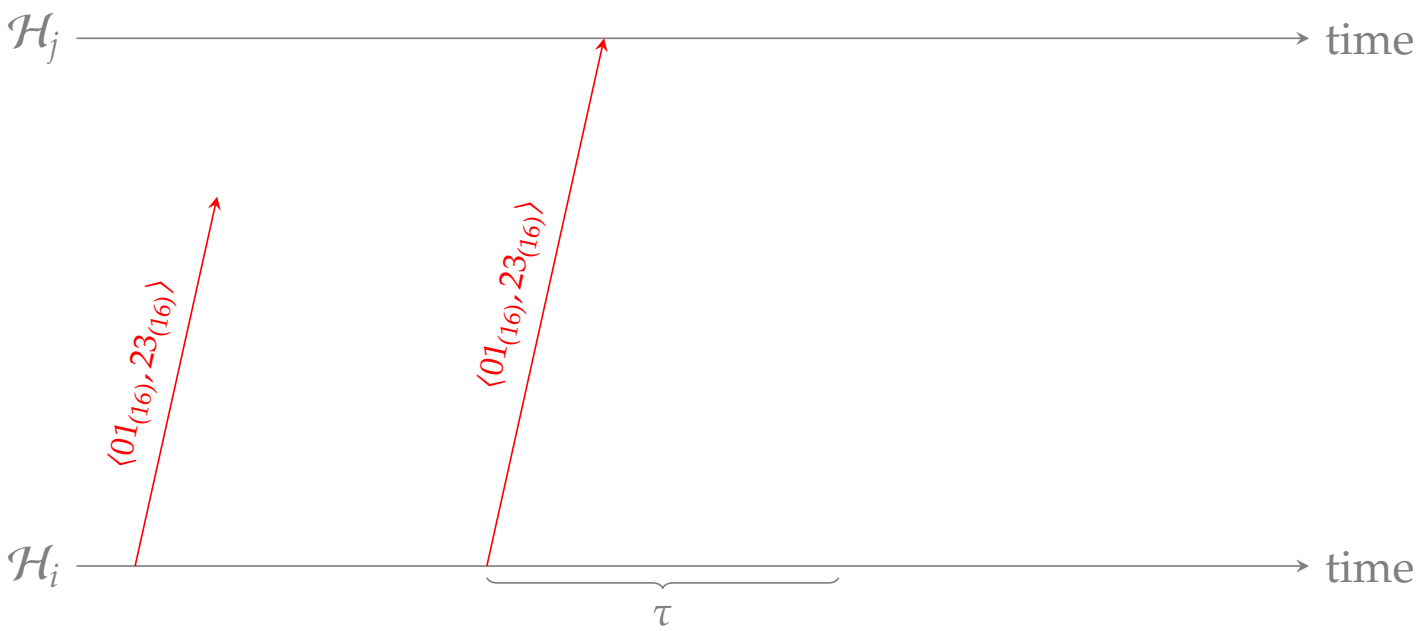
e.g., using the **stop-and-wait protocol** .

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

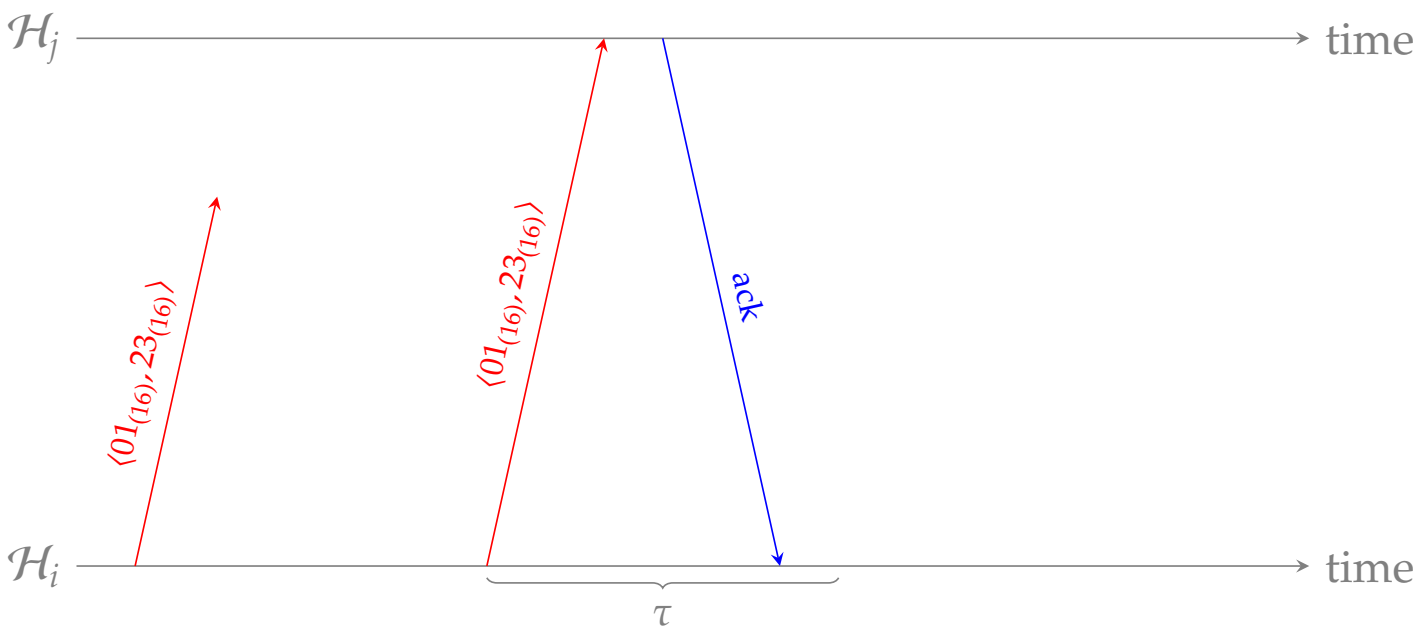


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

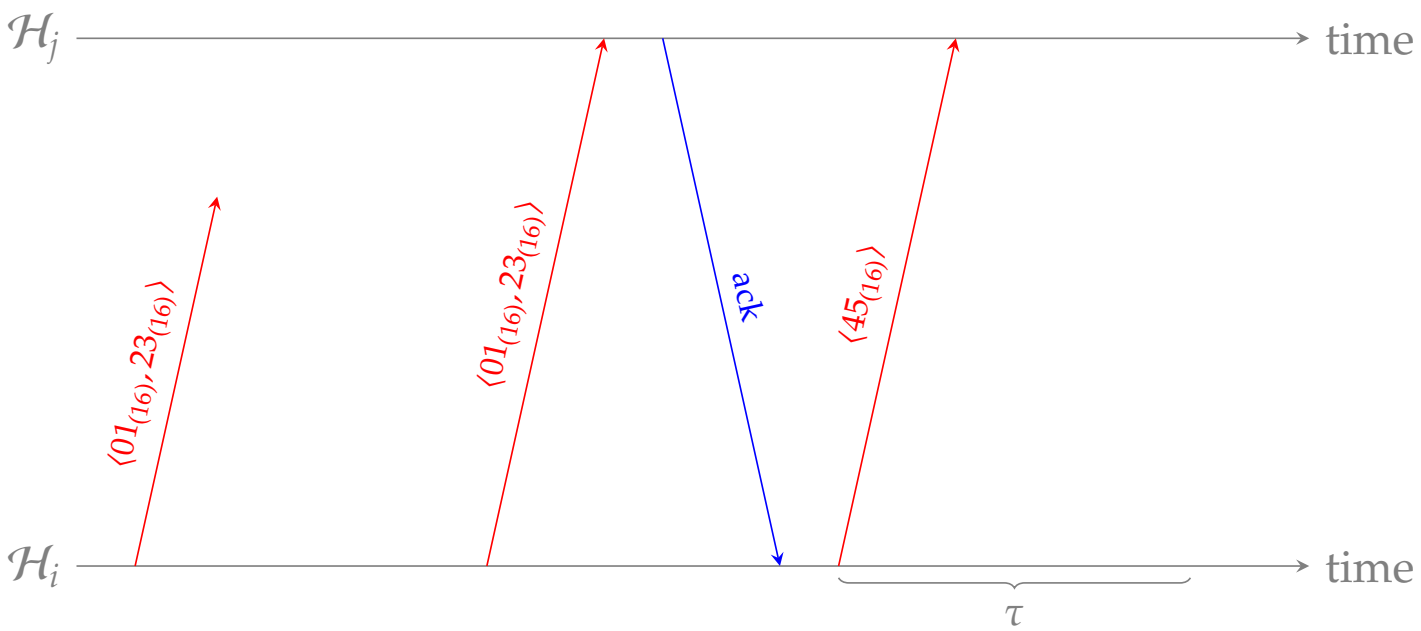


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

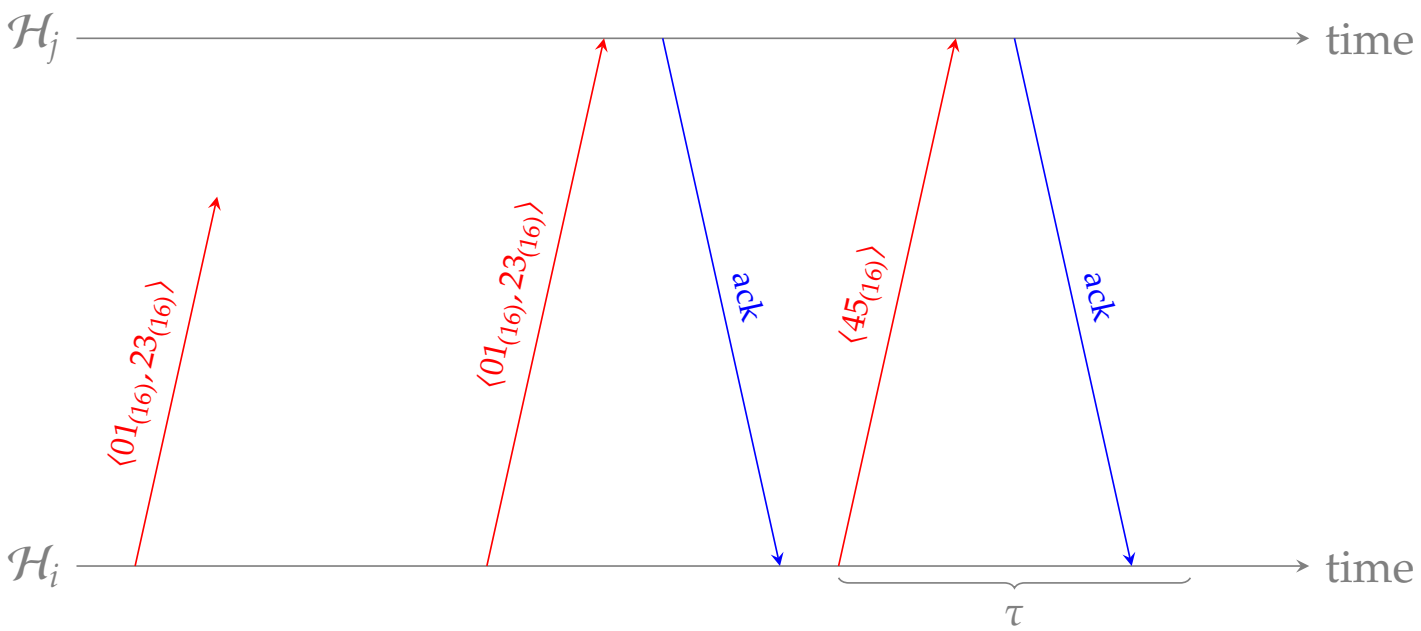


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .



Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

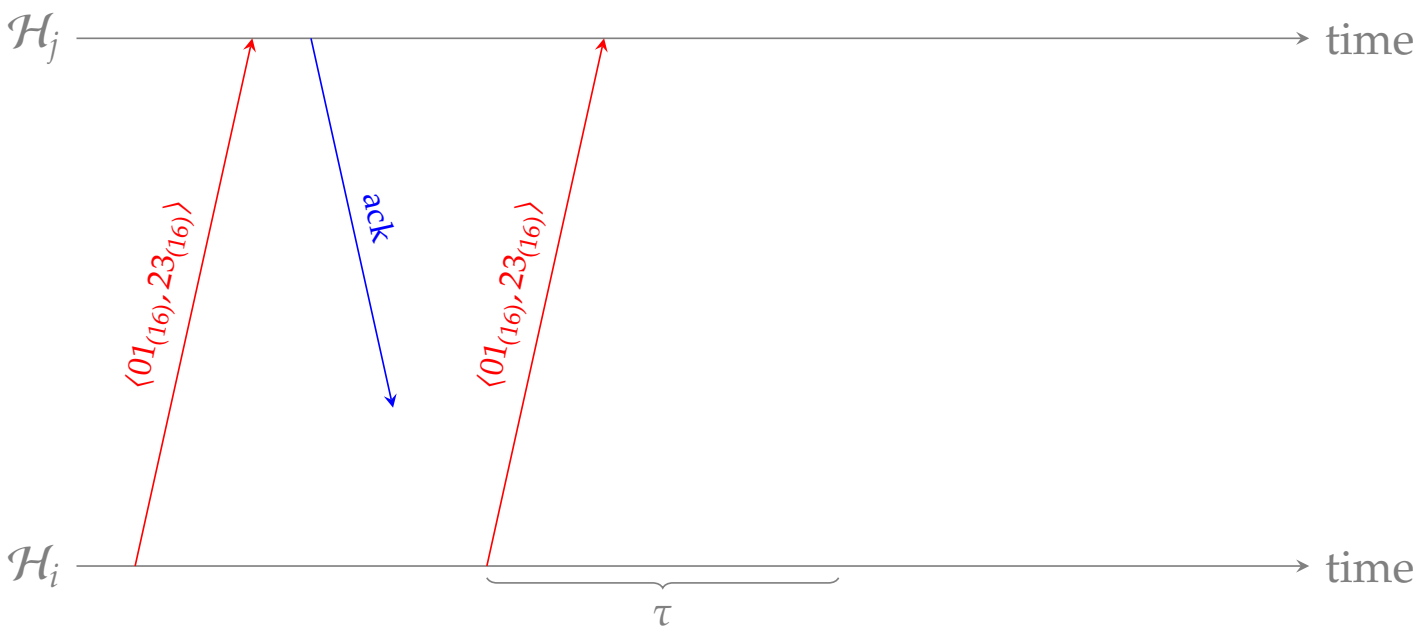
e.g., using the **stop-and-wait protocol** .

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

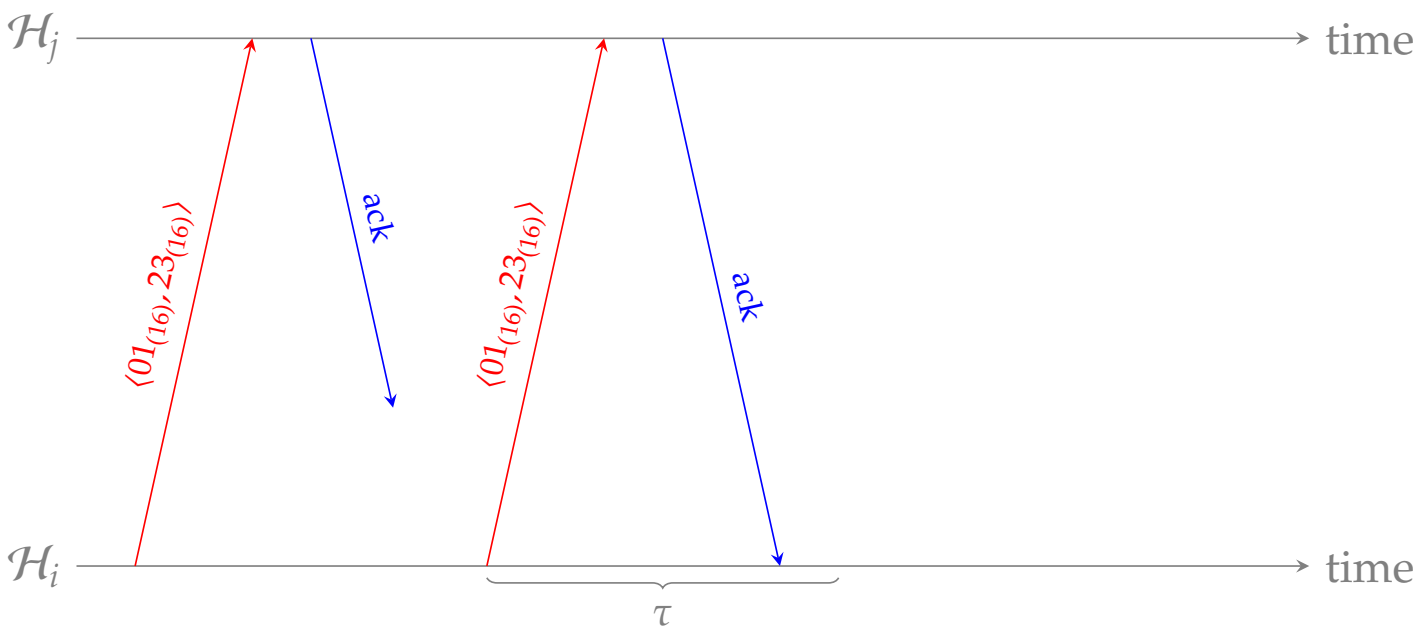


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

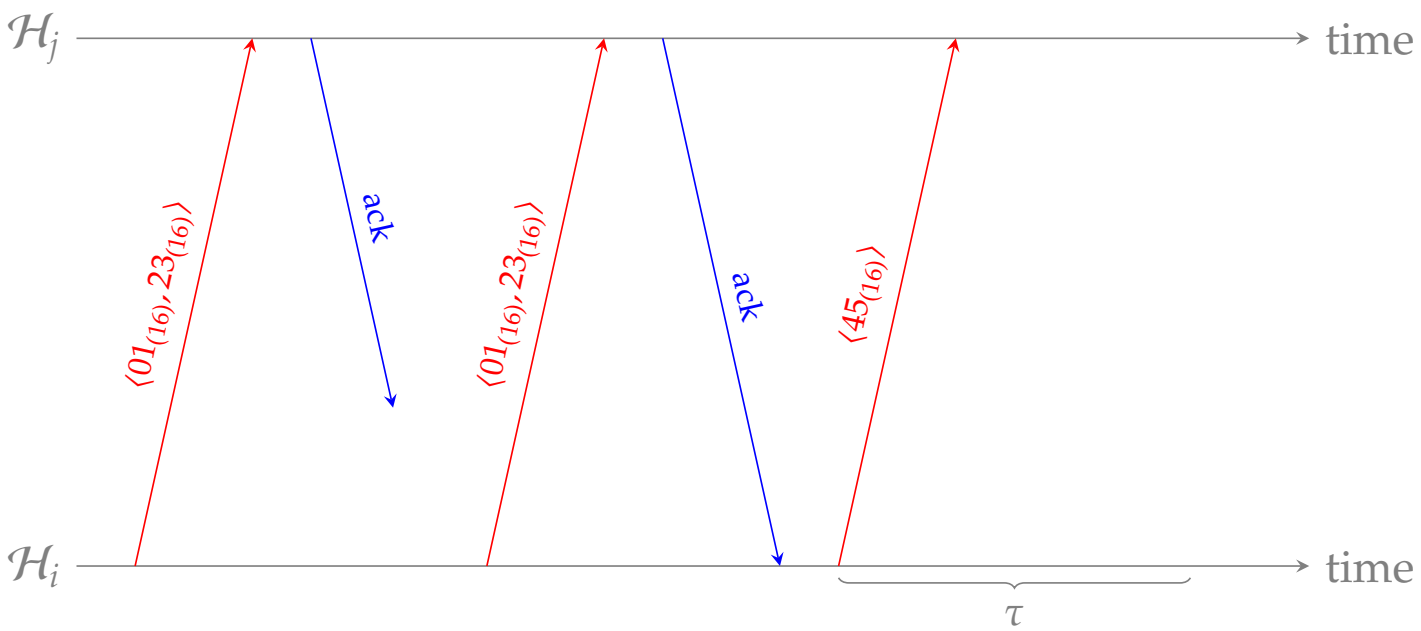


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

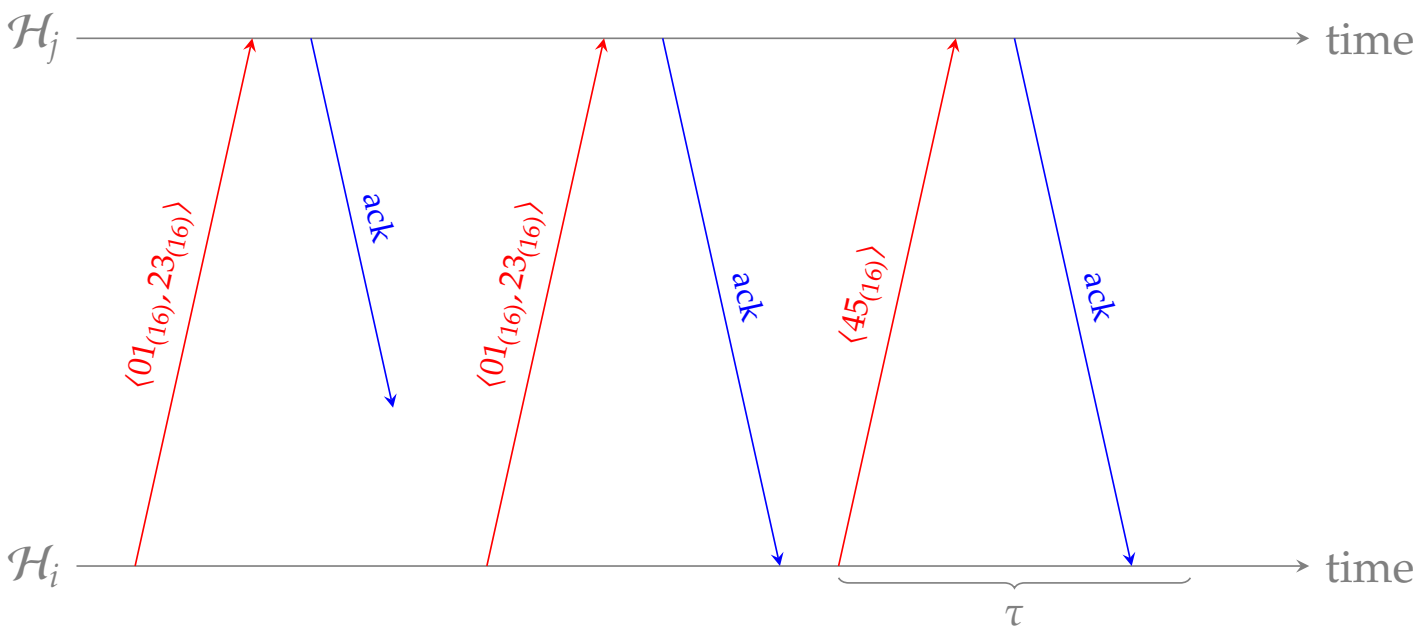


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .



Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

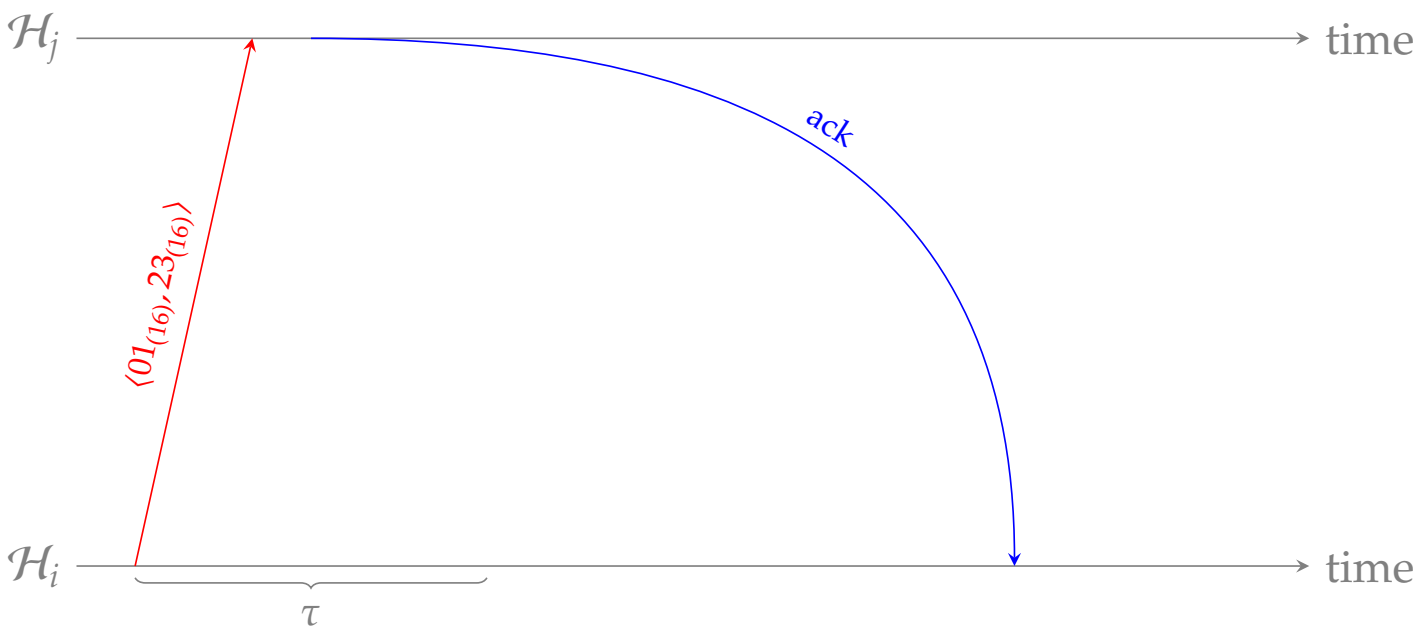
e.g., using the **stop-and-wait protocol** .

Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

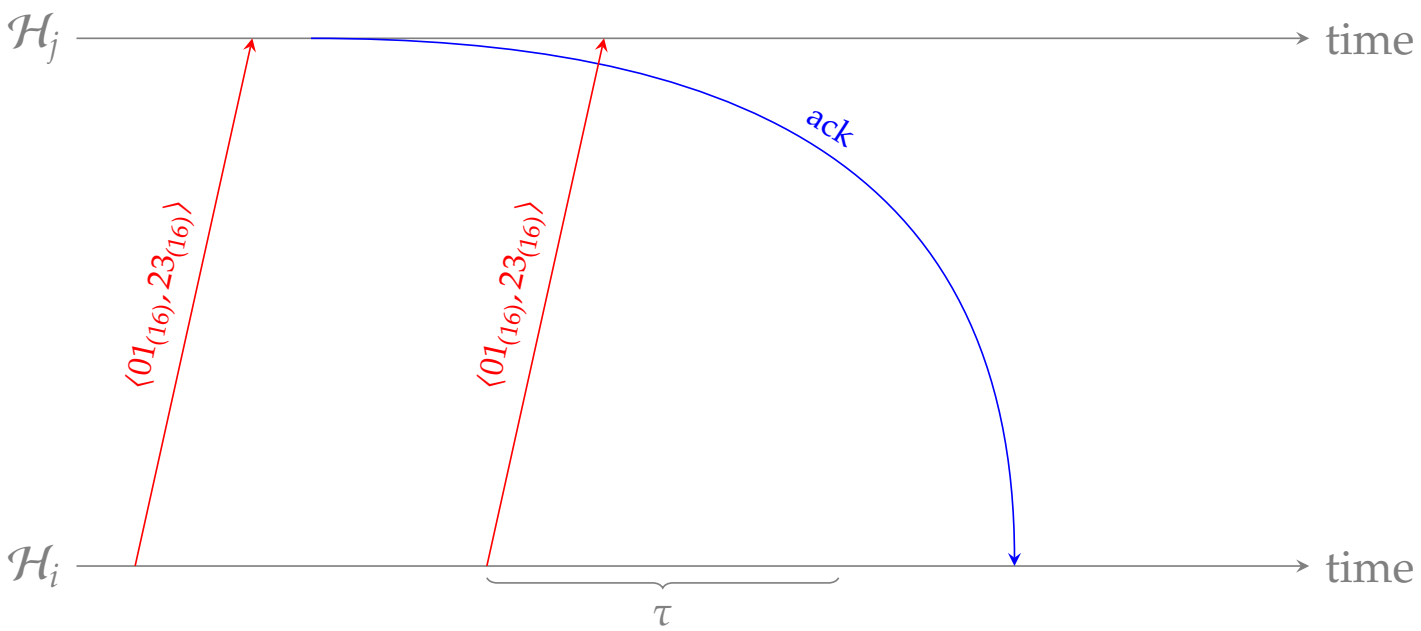


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

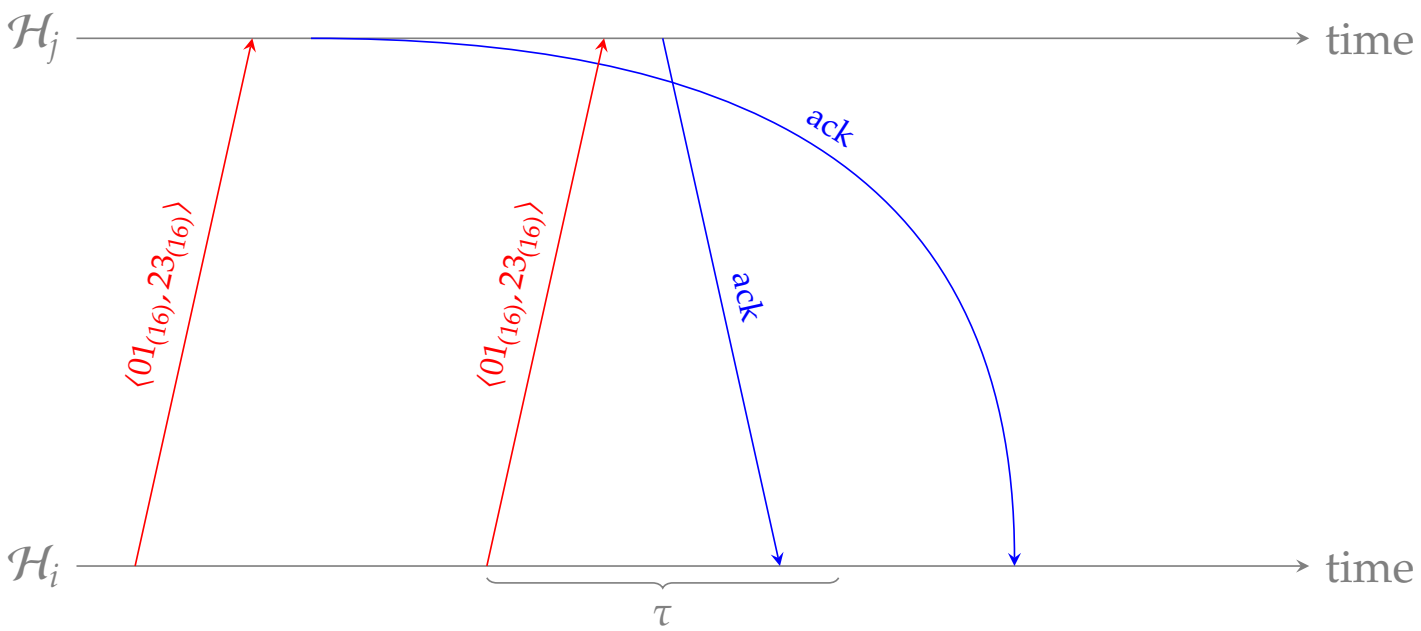


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

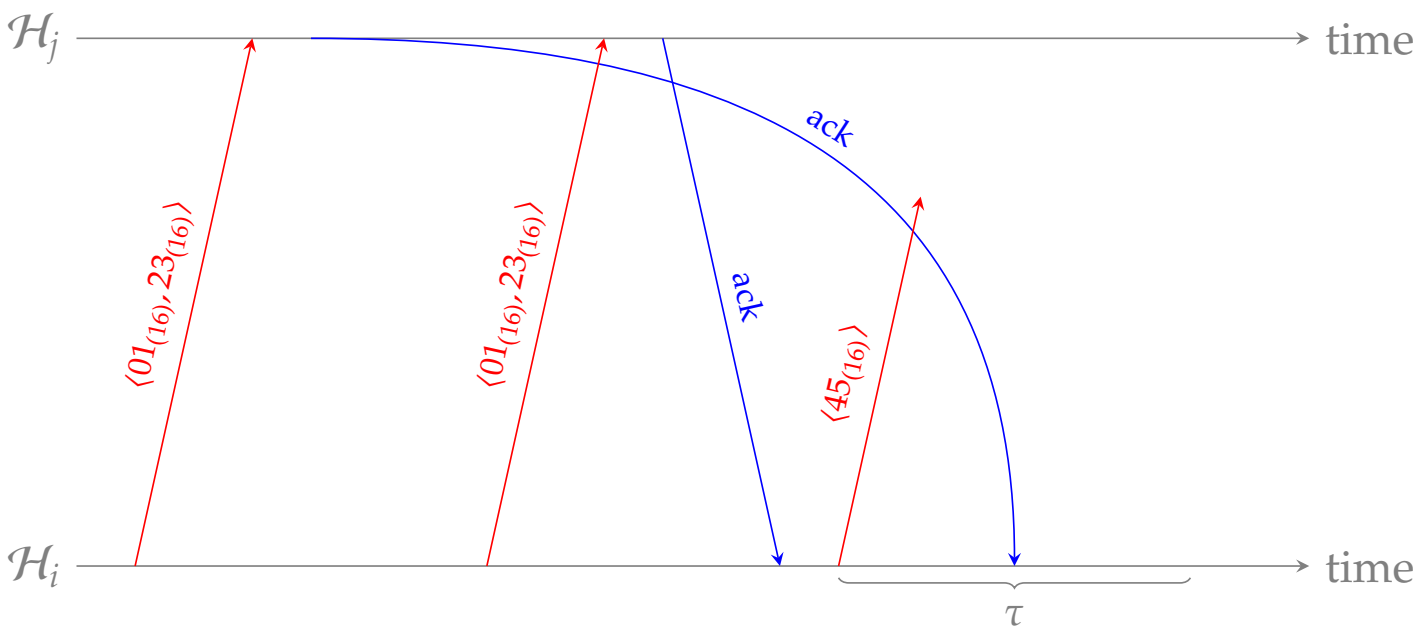


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** .

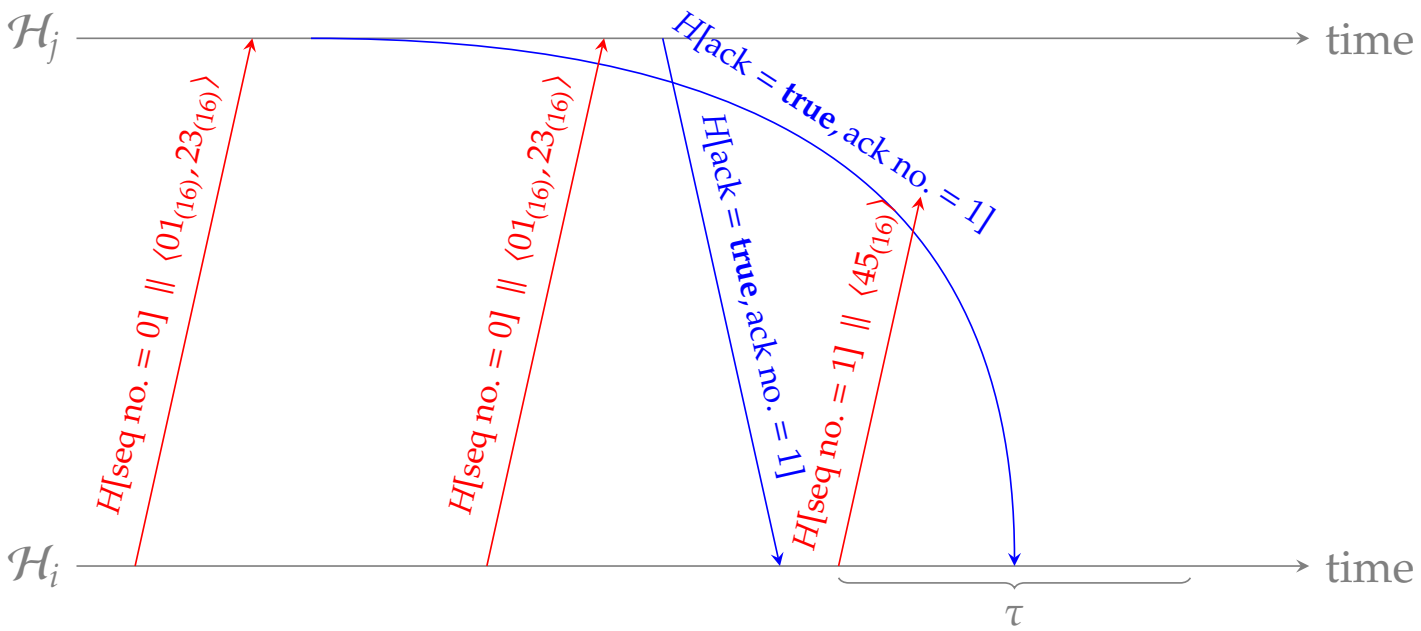


Definition (**Automatic Repeat reQuest (ARQ)**)

Automatic Repeat reQuest (ARQ) is a (generic) mechanism used to provide reliability: it means that the

- destination (automatically) transmits acknowledgement (or **ACK**) on receipt of valid data,
- source (automatically) retransmits segment after time-out (i.e., after τ time units)

e.g., using the **stop-and-wait protocol** (plus **sequence numbers**).



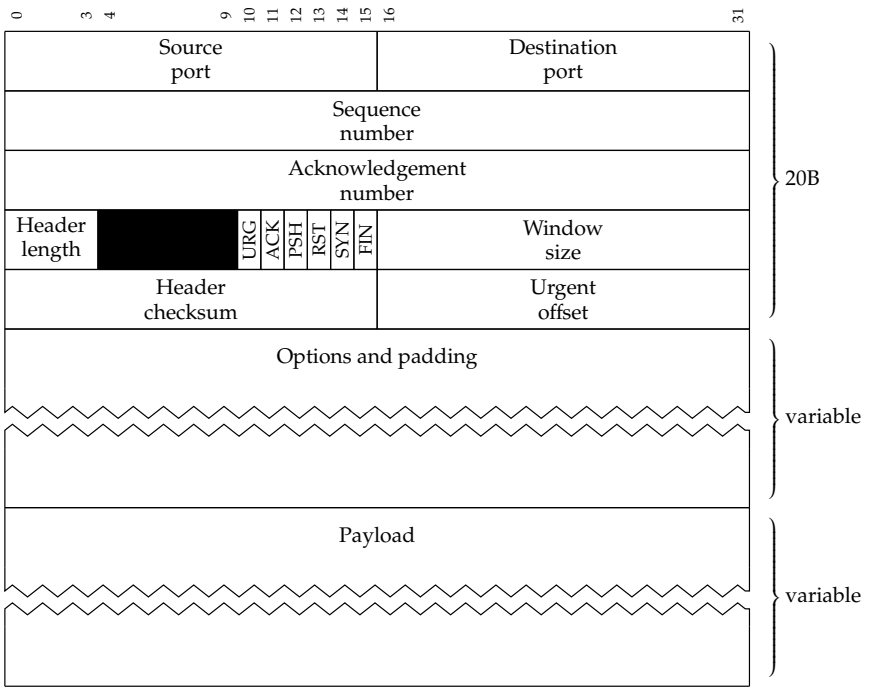
TCP (1)

- Normal TCP-based communication occurs in three phases, namely
  1. **connection establishment**
     - 1.1 exchange signalling parameters,
     - 1.2 allocate resources,
     - 1.3 synchronise ready to communicate
  2. full-duplex (i.e., 2-way), unicast (i.e., with precisely two end-points) communication via the established connection, then eventually
  3. **connection termination**
     - 3.1 complete pending communication,
     - 3.2 release resources

  plus various auxiliary actions, e.g., **connection reset**.

TCP (2)

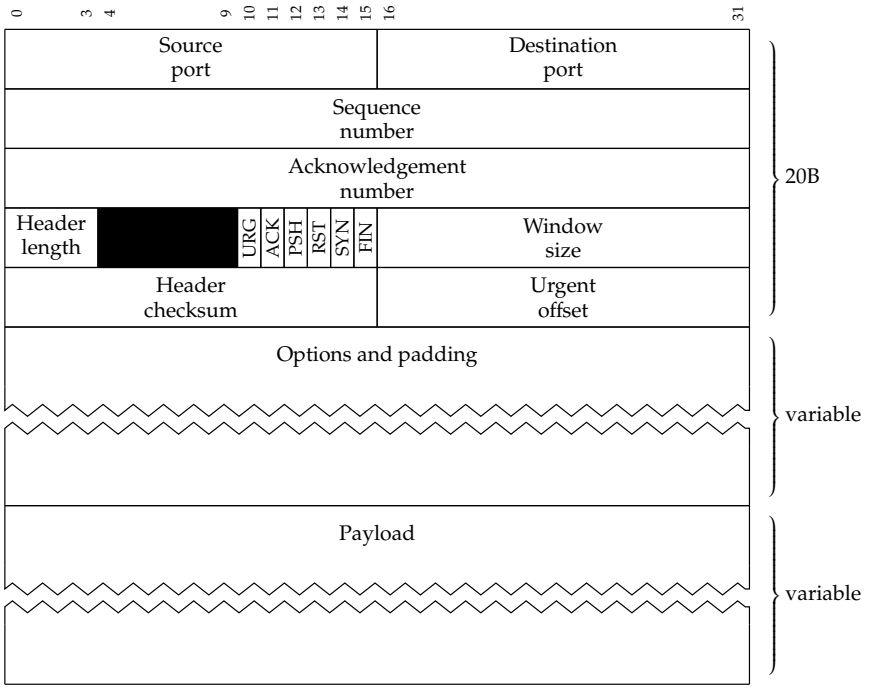
Data Structure (**TCP segment** [7, Section 3.1])



The data structure includes:

- A 16-bit source port.
- A 16-bit destination port.
- A 32-bit sequence number.
- A 32-bit acknowledgement number.

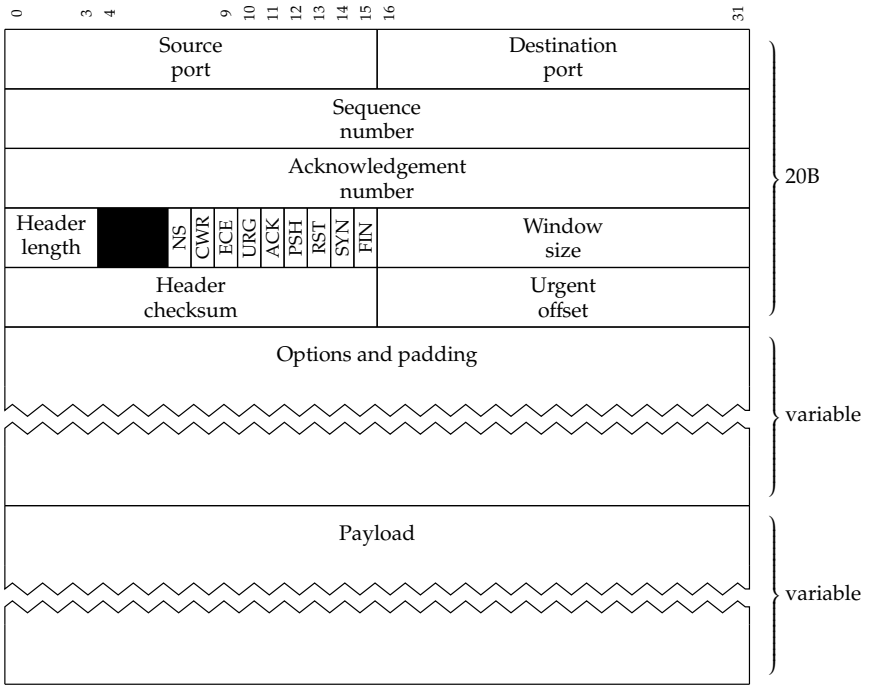
Data Structure (**TCP segment** [7, Section 3.1])



The data structure includes:

- A set of flags, including
  - 1-bit **URGent (URG)** flag, which marks the urgent pointer field as significant.
  - 1-bit **ACKnowledgement (ACK)** flag, which marks the acknowledgement field as significant.
  - 1-bit **PuSH (PSH)** flag, which means "transmit *now*: don't buffer".
  - 1-bit **ReSeT (RST)** flag, used for connection control.
  - 1-bit **SYNchronise (SYN)** flag, used for connection control.
  - 1-bit **FINish (FIN)** flag, used for connection control.

Data Structure (**TCP segment** [7, Section 3.1]+[8, Section 6]+[9, Section 9])



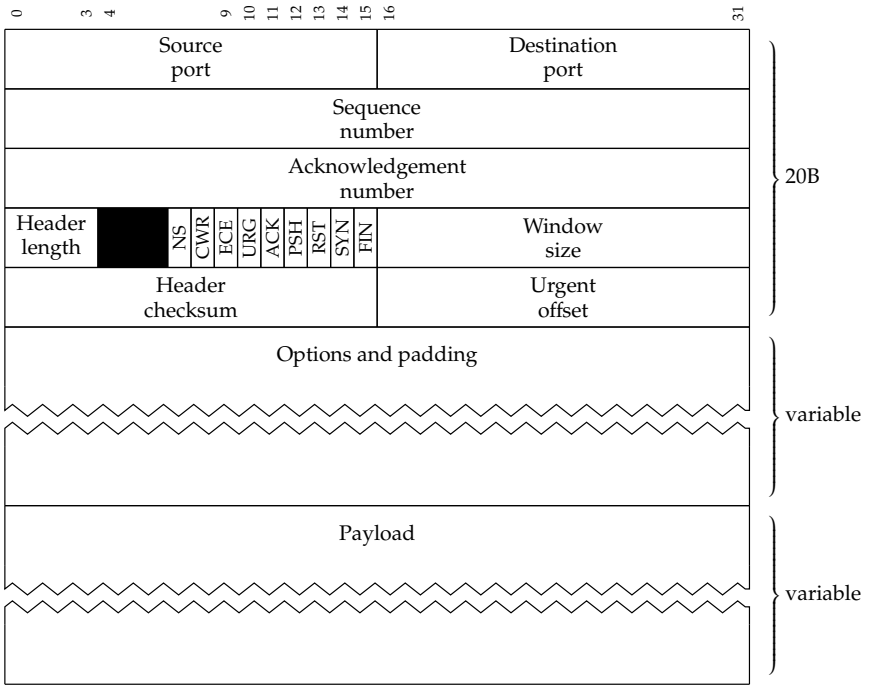
The data structure includes:

- A set of flags, including
  - 1-bit **URGent (URG)** flag, which marks the urgent pointer field as significant.
  - 1-bit **ACKnowledgement (ACK)** flag, which marks the acknowledgement field as significant.
  - 1-bit **PuSH (PSH)** flag, which means "transmit *now*: don't buffer".
  - 1-bit **ReSeT (RST)** flag, used for connection control.
  - 1-bit **SYNchronise (SYN)** flag, used for connection control.
  - 1-bit **FINish (FIN)** flag, used for connection control.

  plus some extras for **Explicit Congestion Notification (ECN)**.

TCP (2)

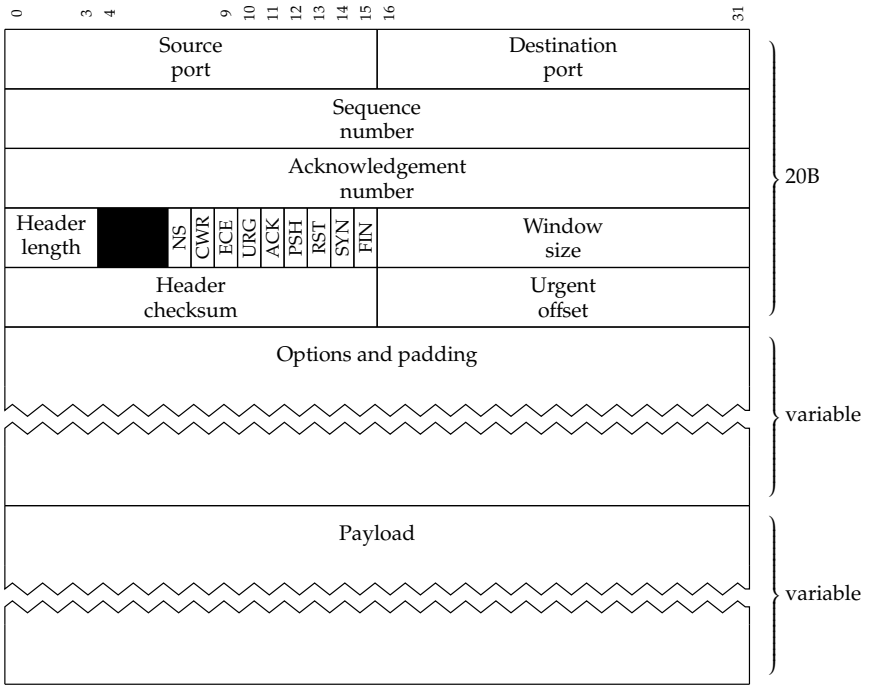
Data Structure (**TCP segment** [7, Section 3.1]+[8, Section 6]+[9, Section 9])



The data structure includes:

- A window size, used for flow control.
- A 16-bit checksum (on whole segment) used to detect errors
- An urgent offset, used for flow control.

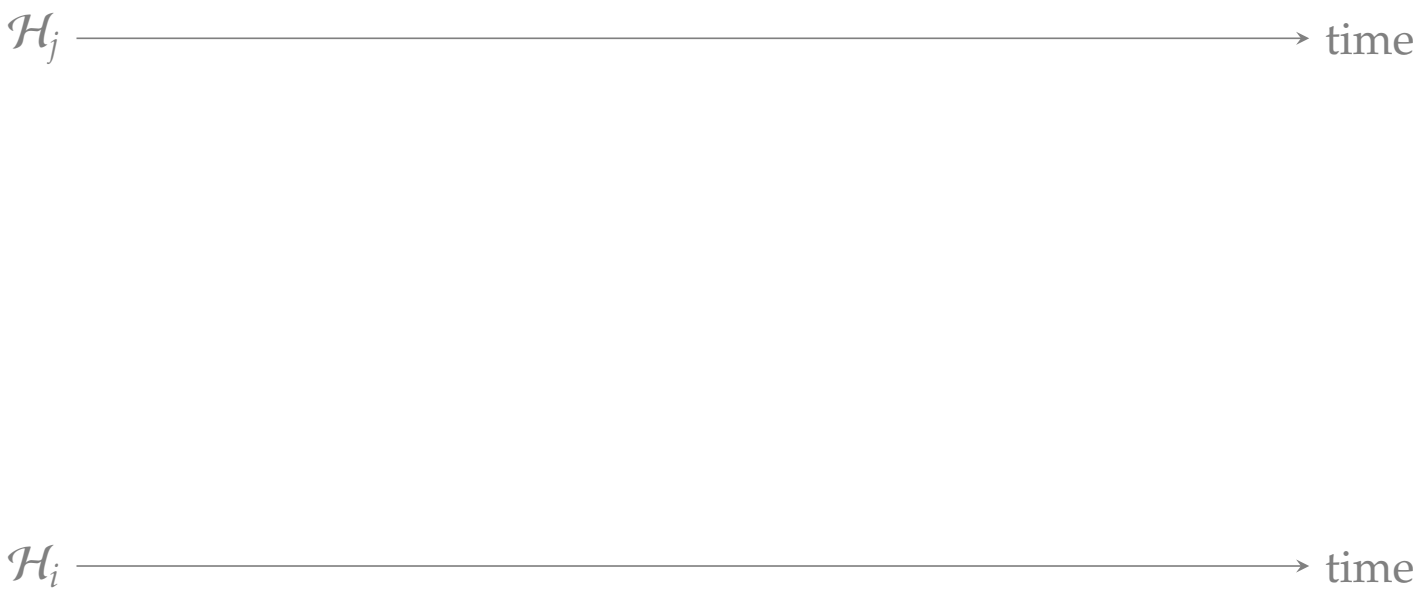
Data Structure (**TCP segment** [7, Section 3.1]+[8, Section 6]+[9, Section 9])



The data structure includes:

- A set of options (allowing protocol extensibility).
- Any padding required to ensure the header is a multiple of 32 bits.
- The payload.

TCP (5)

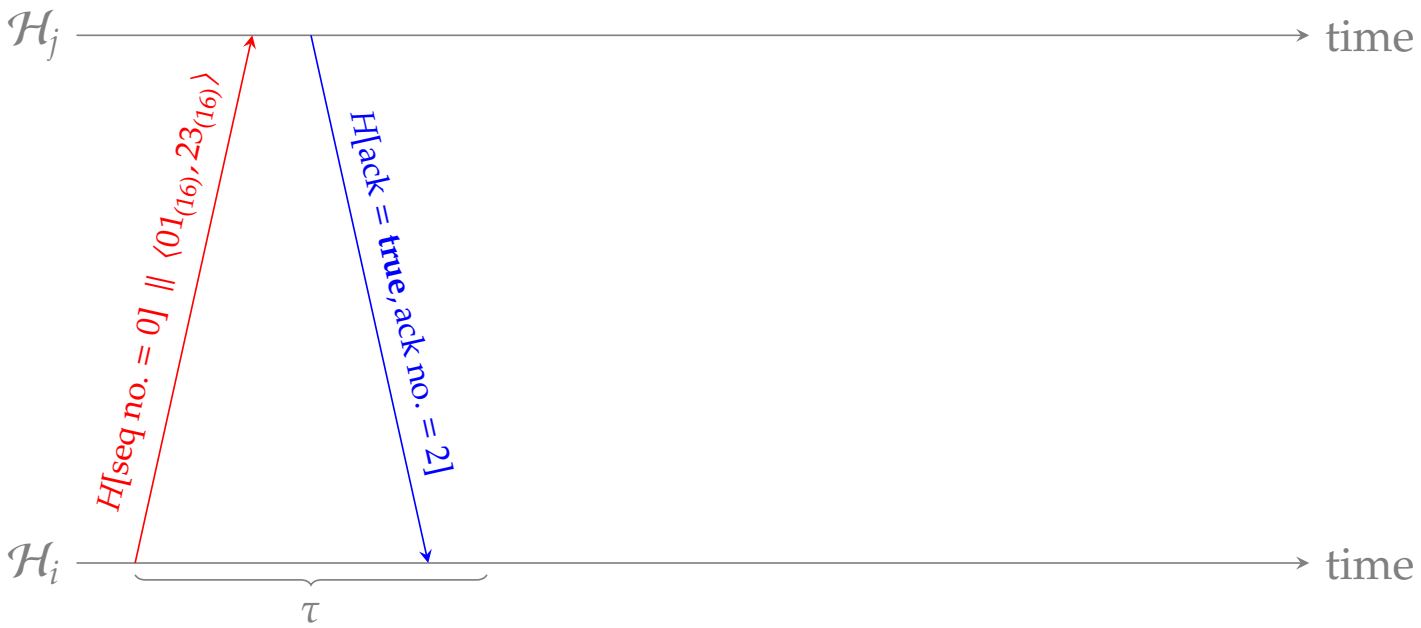


- Question: why have a sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)

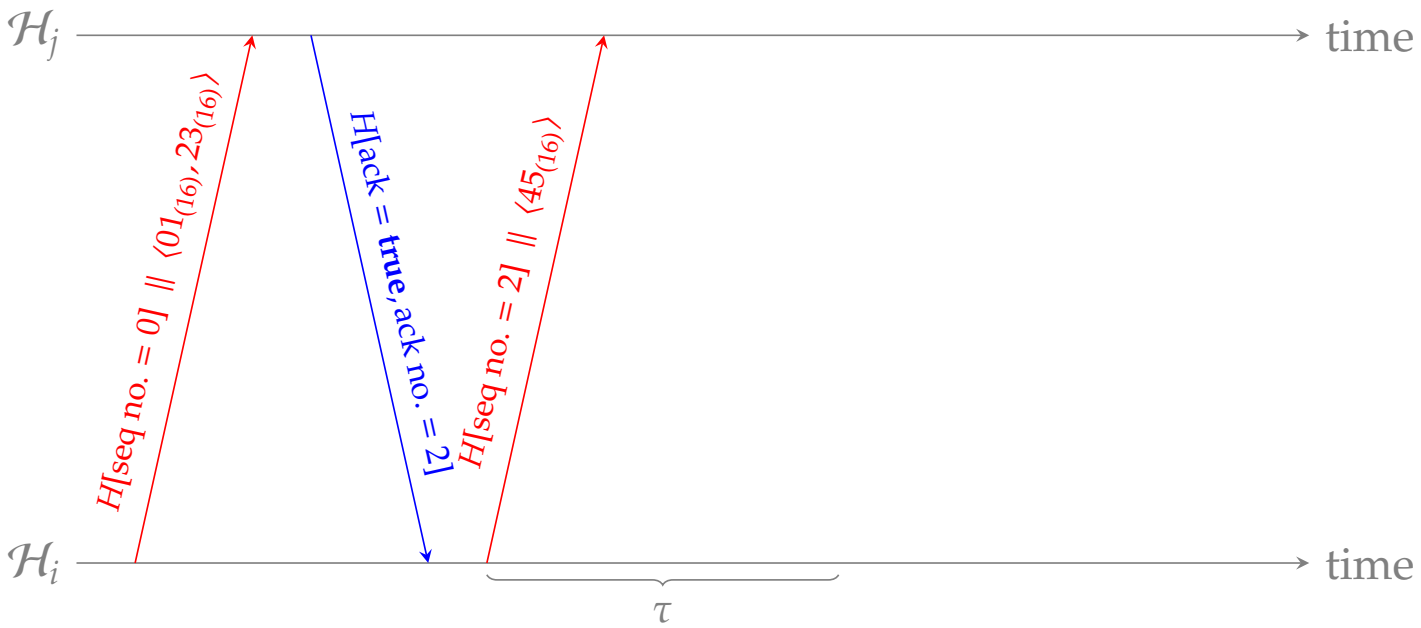
- Question: why have a sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



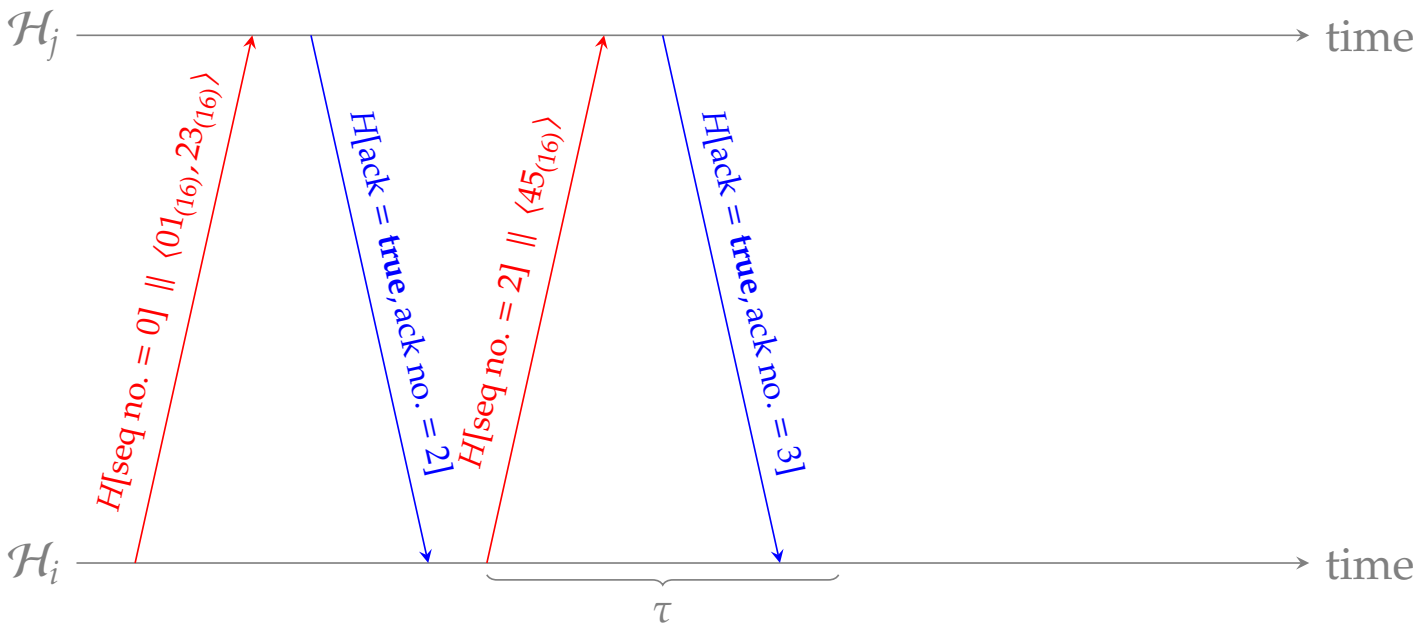
- Question: why have a sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



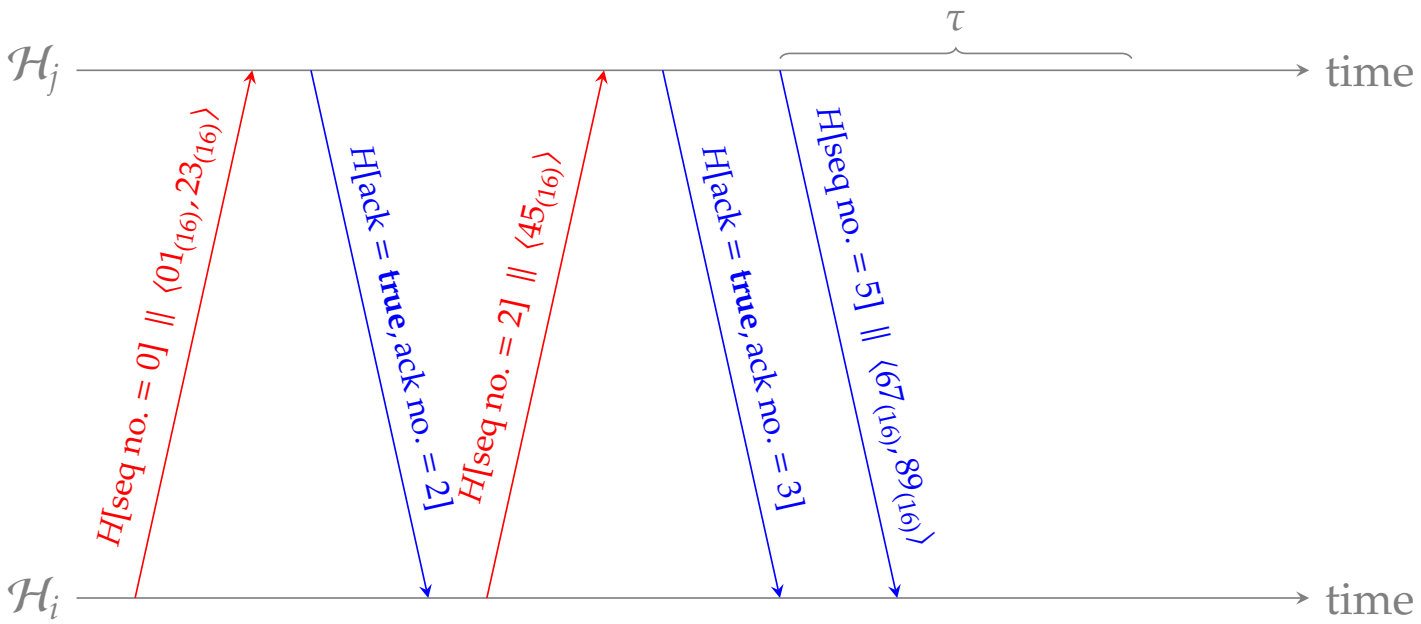
- Question: why have a sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



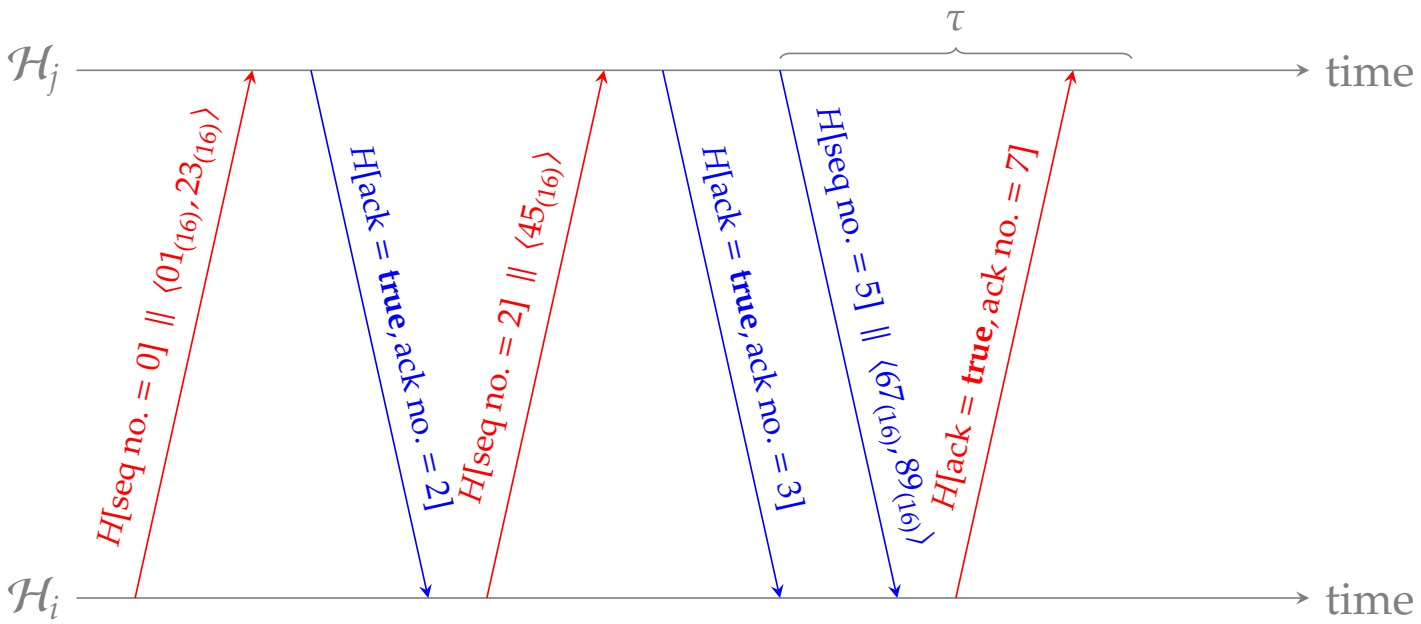
- Question: why have a sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



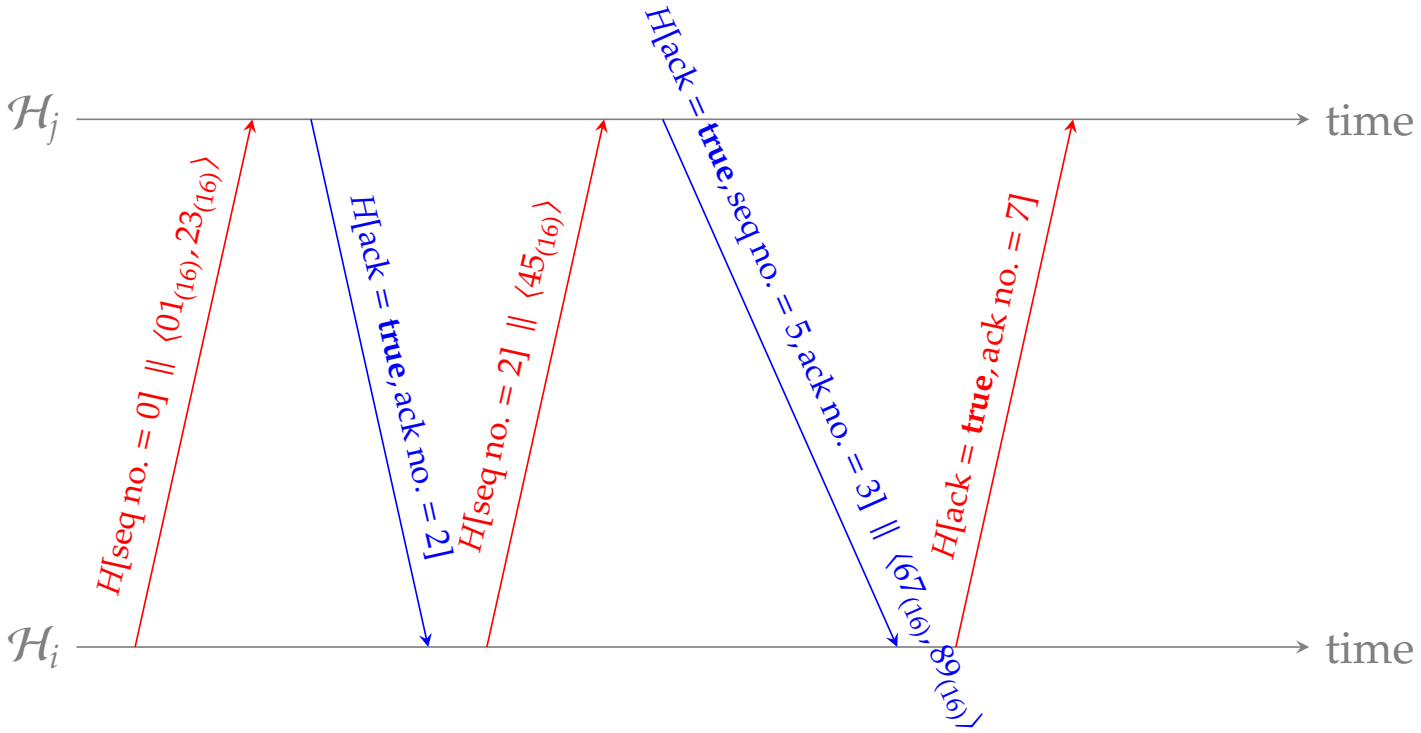
- Question: why have a sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



- Question: why have a sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

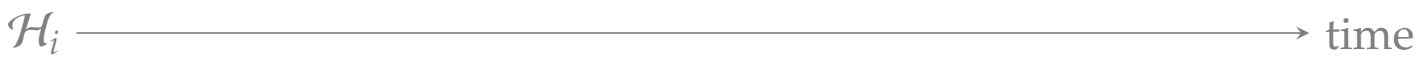
TCP (5)



- Question: why have a sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (6)

- Example:

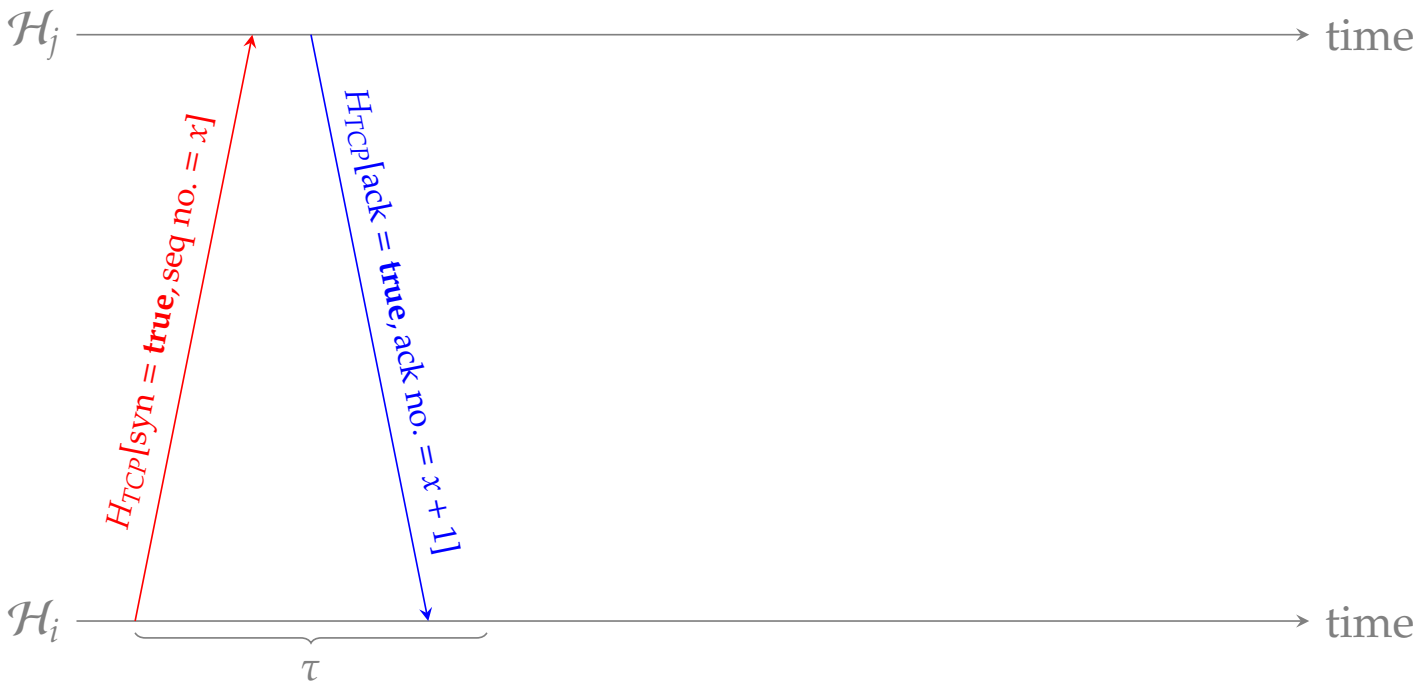


TCP (6)

- Example:
  1. connection establishment,

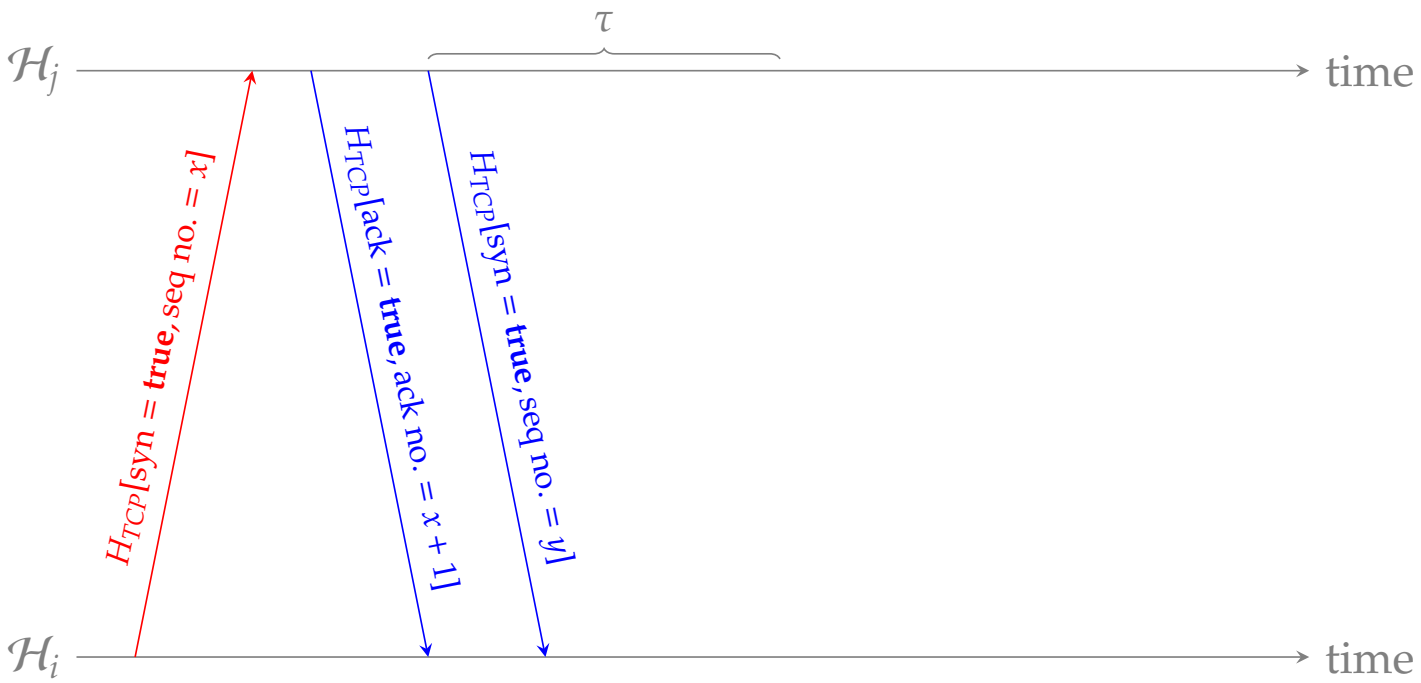
TCP (6)

- Example:
  1. connection establishment,



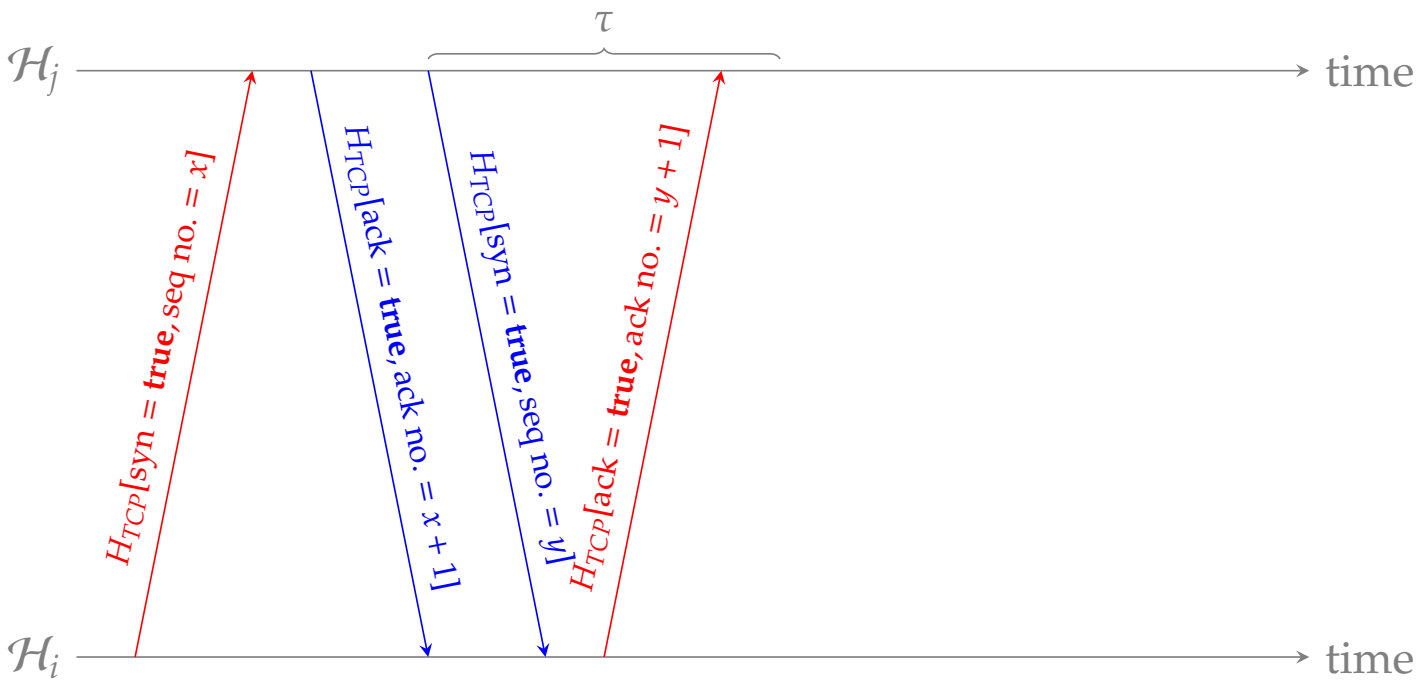
TCP (6)

- Example:
  1. connection establishment,



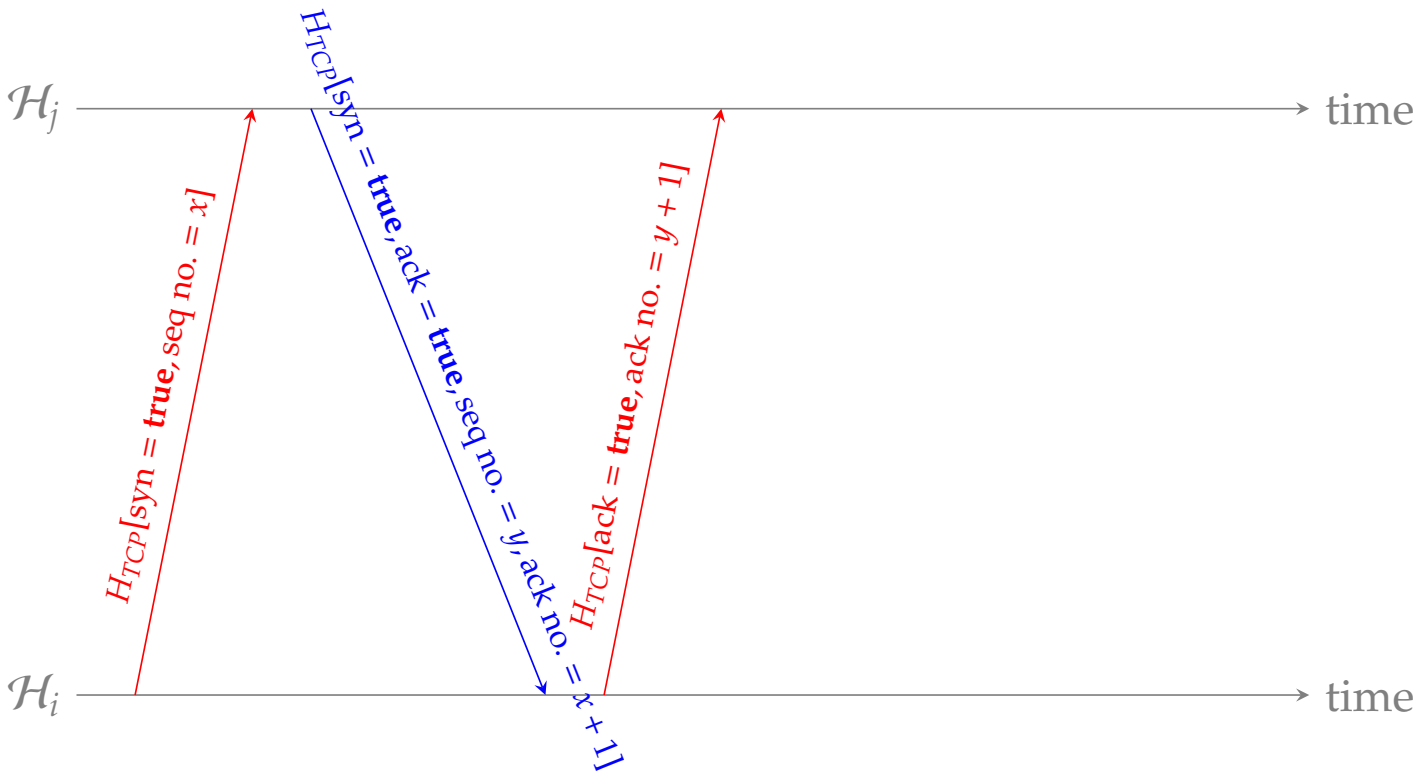
TCP (6)

- Example:
  1. connection establishment,



TCP (6)

- Example:
  1. connection establishment,

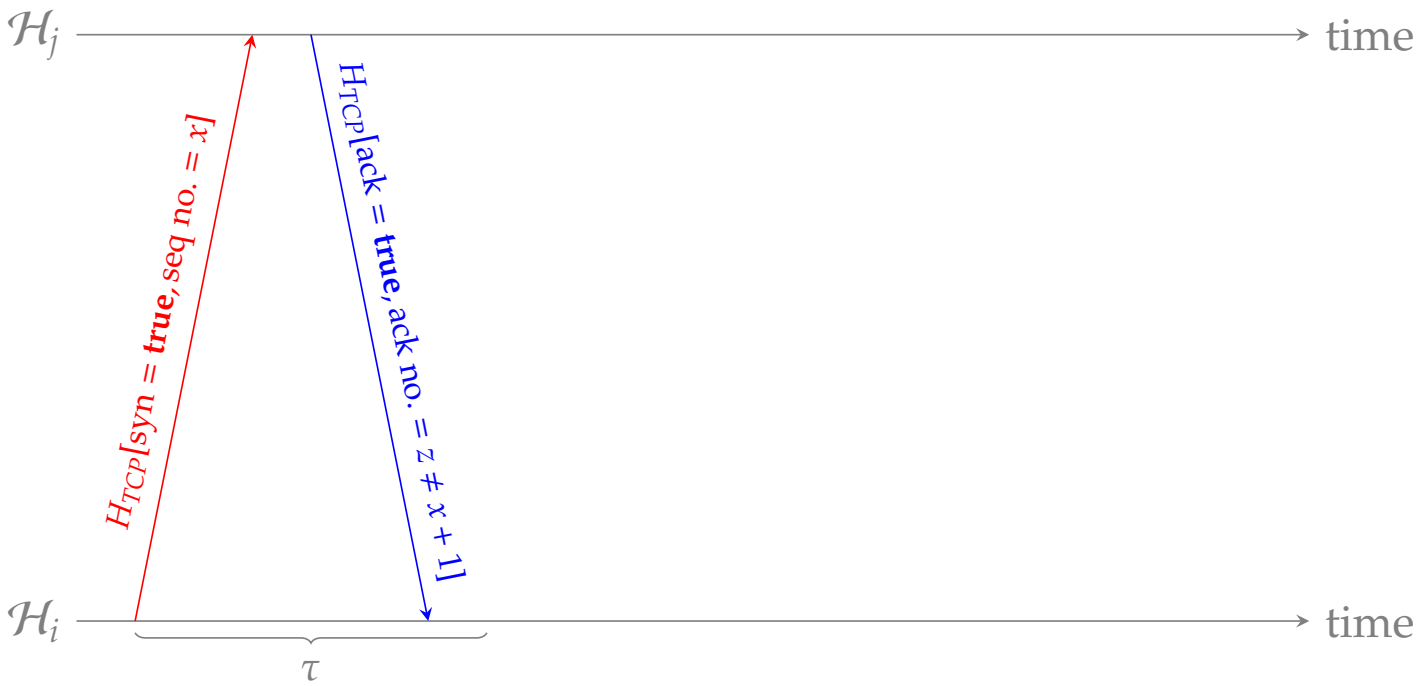


TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,

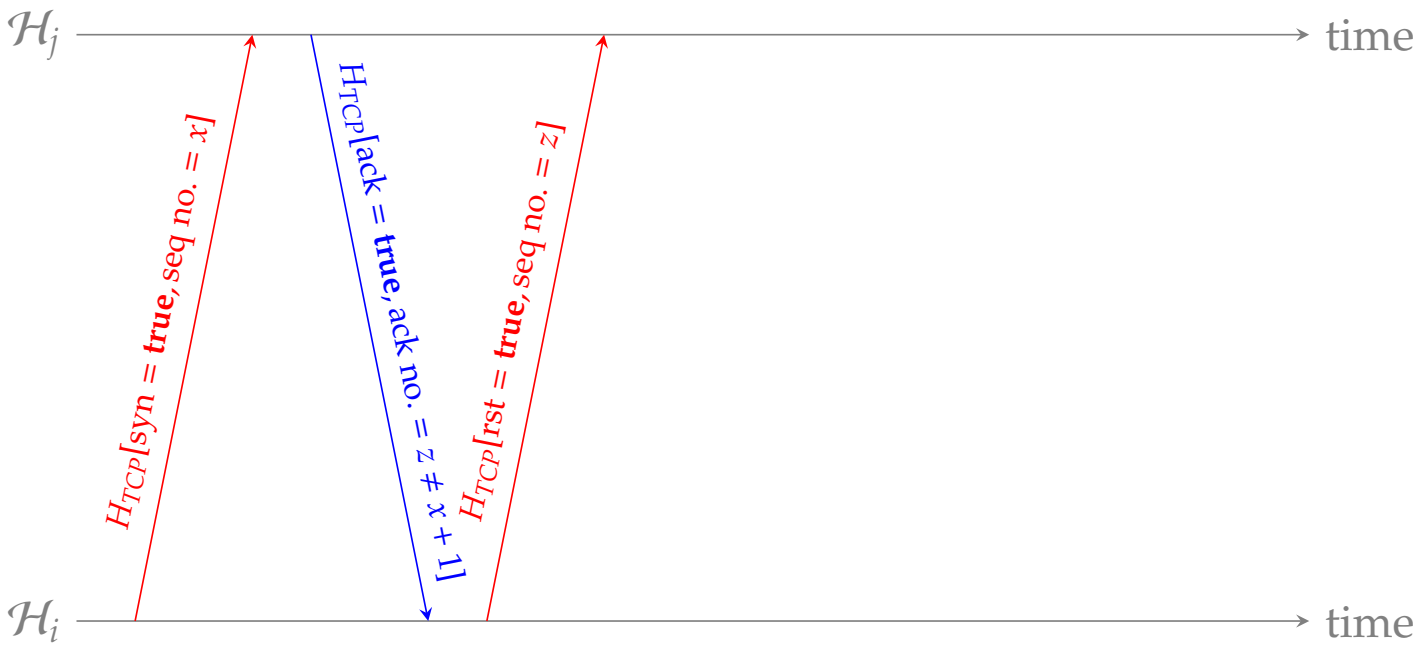
TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,



TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,

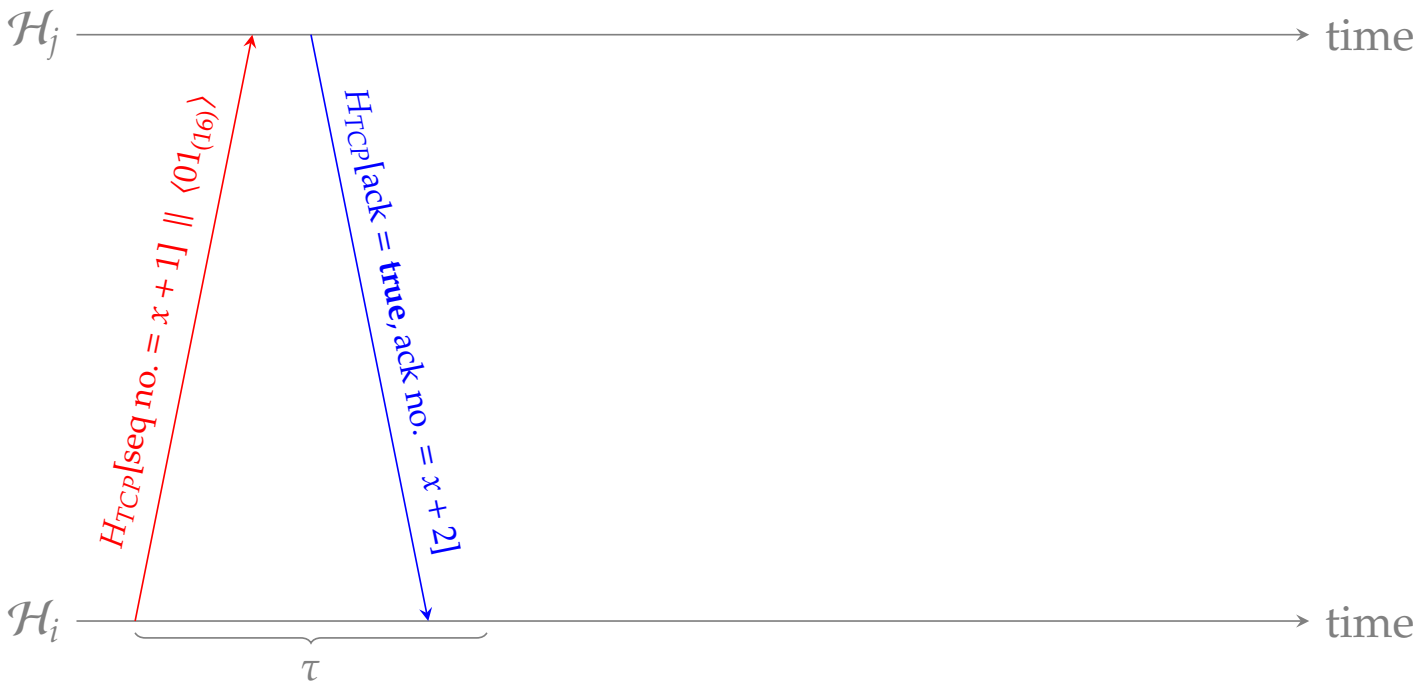


TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and

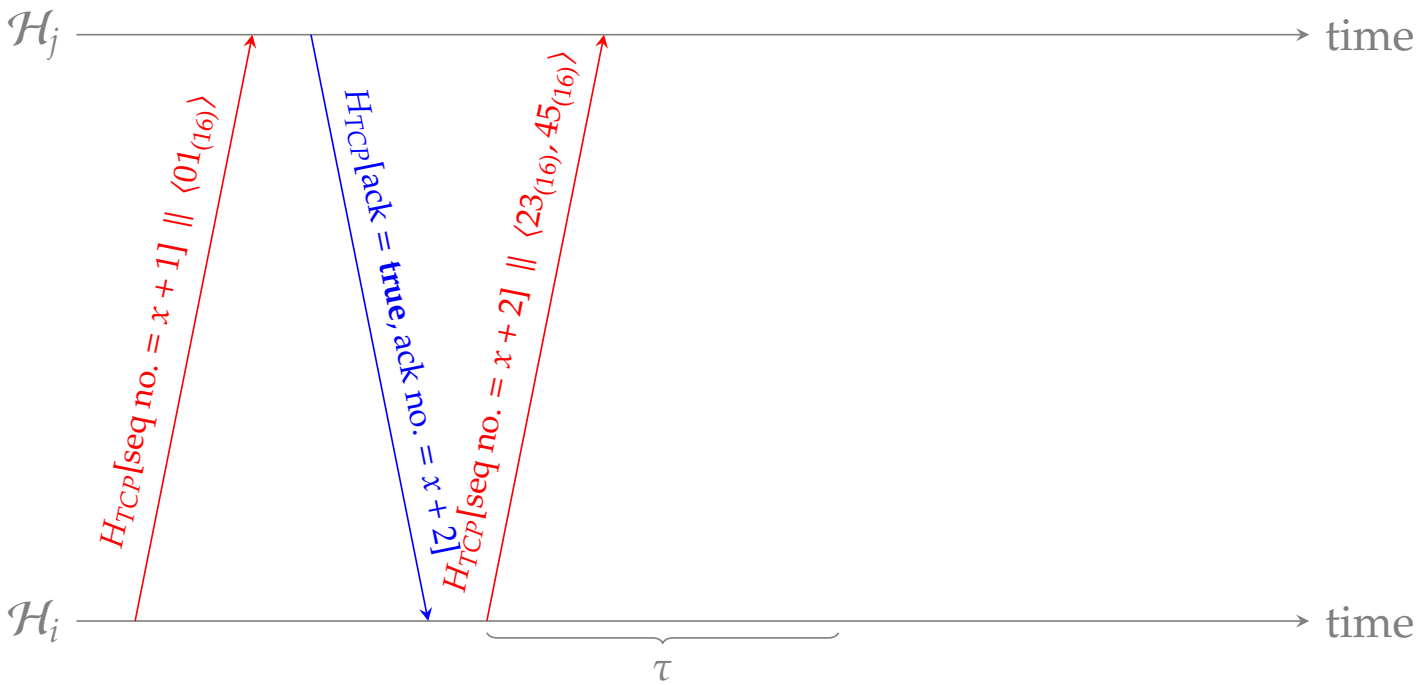
TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and



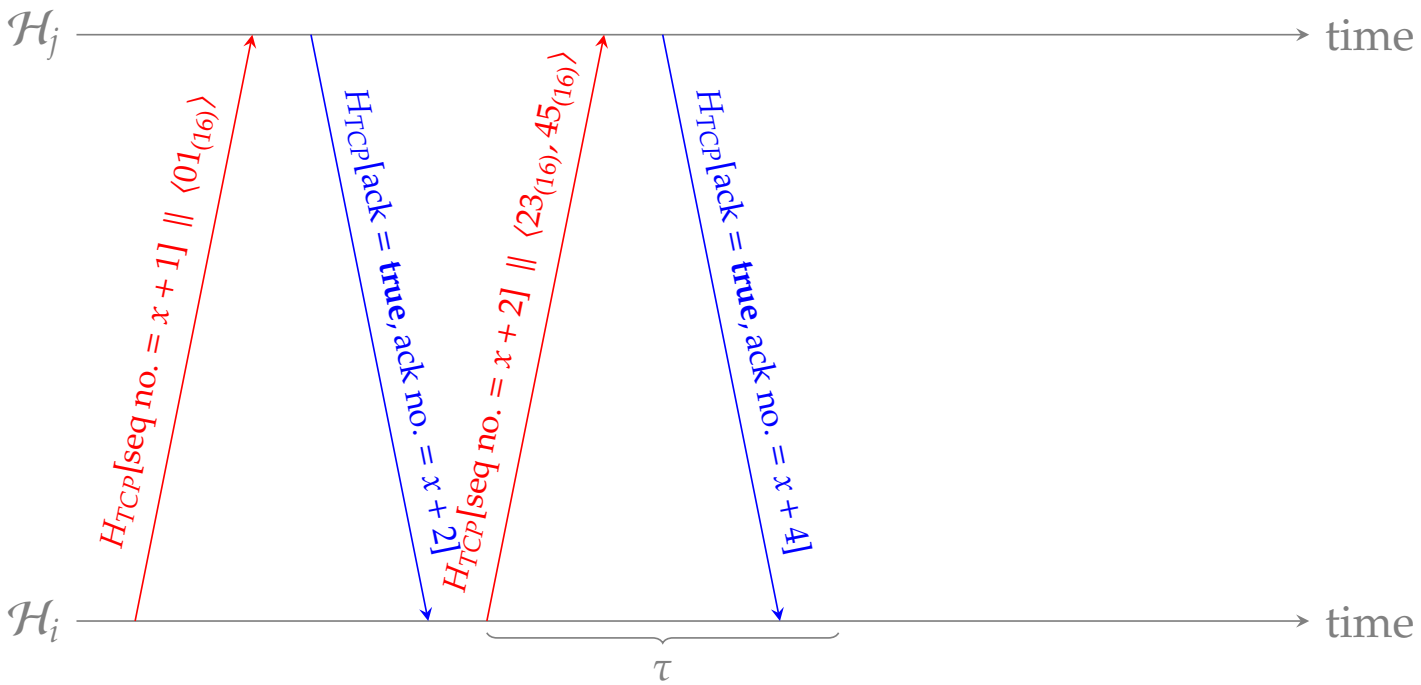
TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and



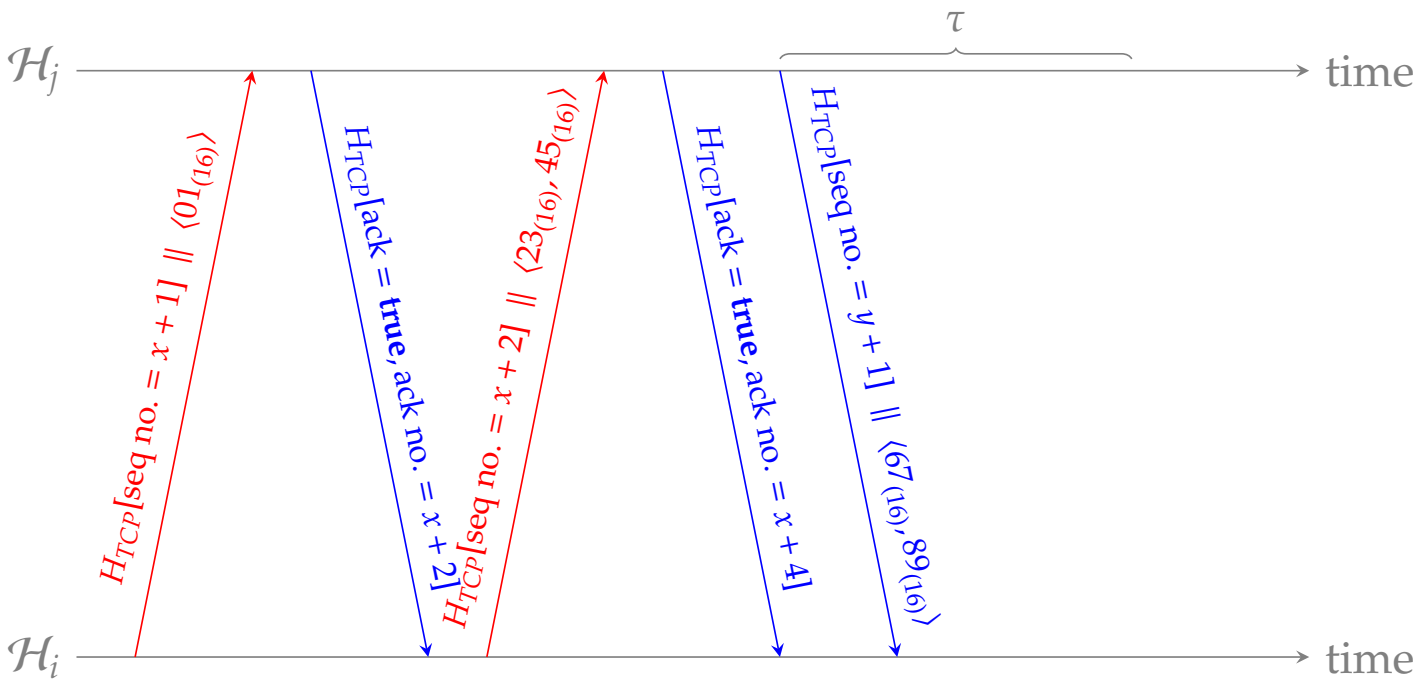
TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and



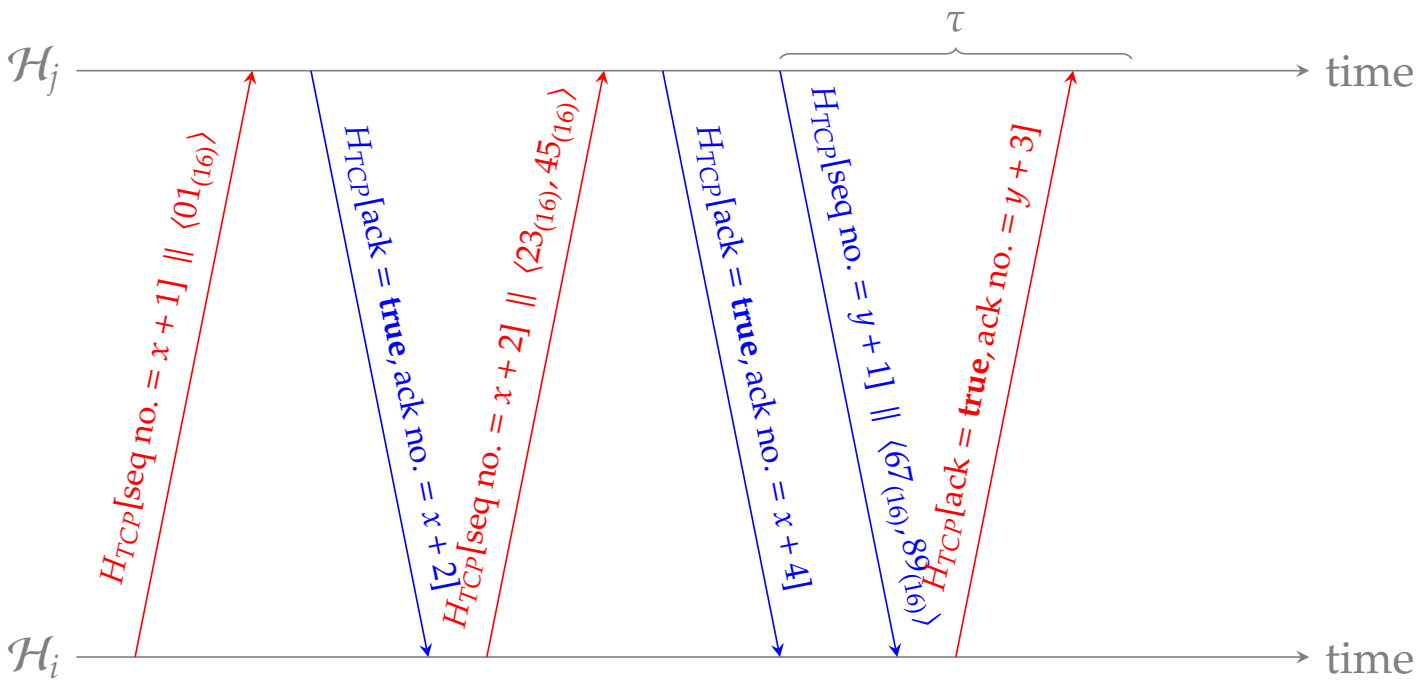
TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and



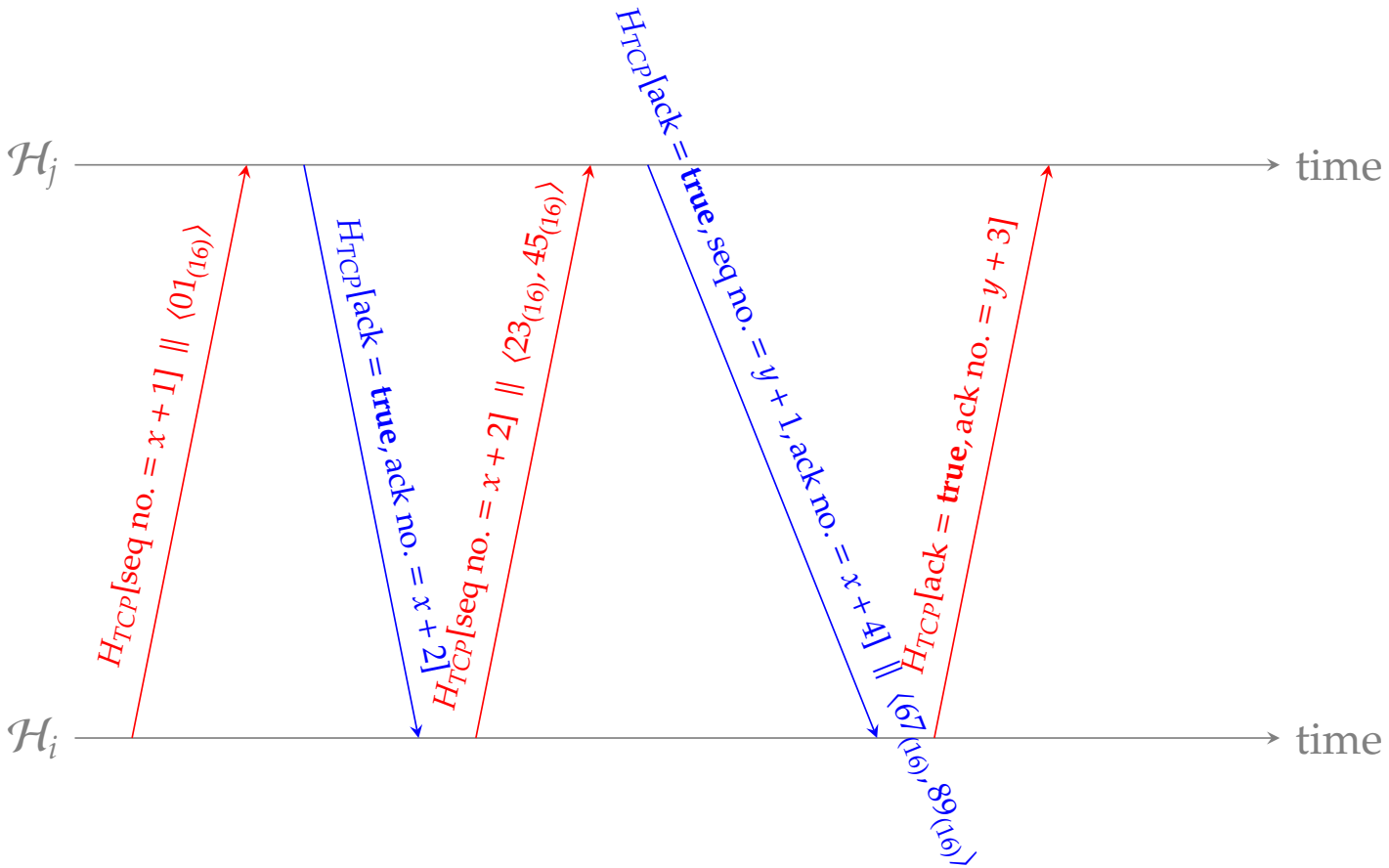
TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and



TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and

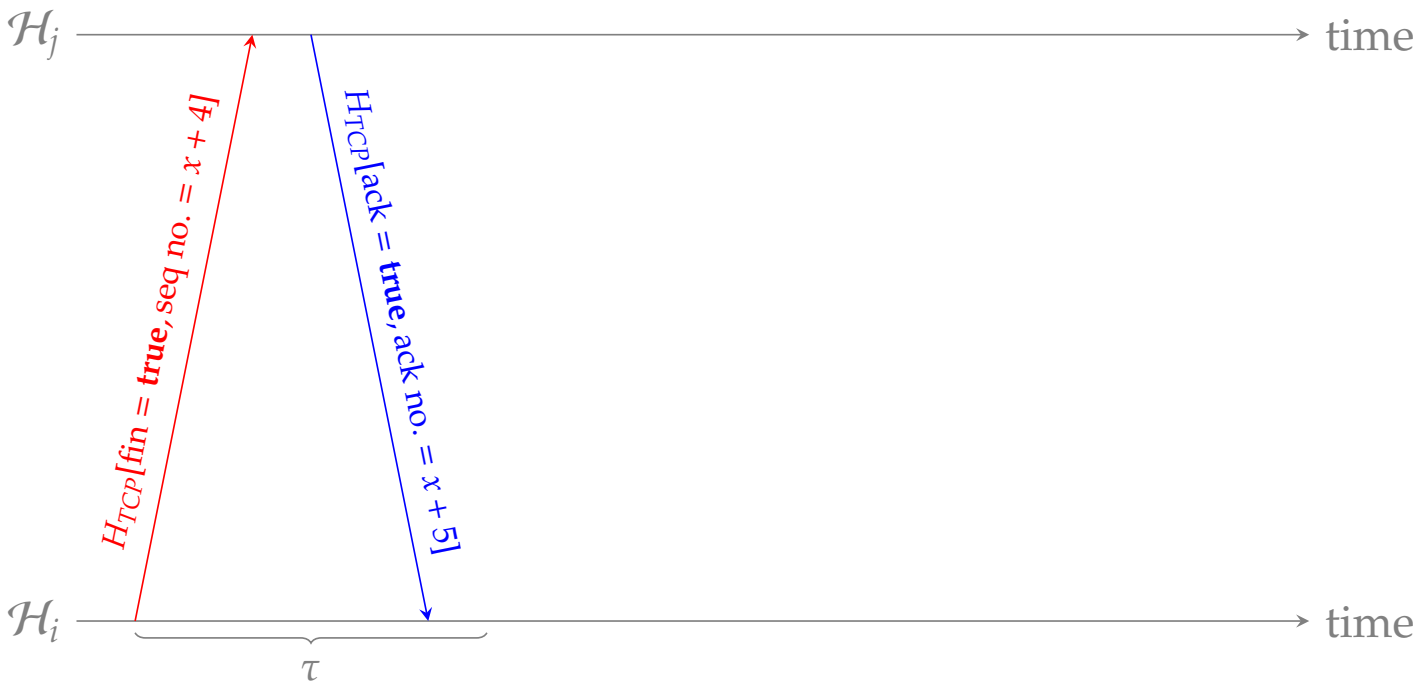


TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and
  4. connection termination.

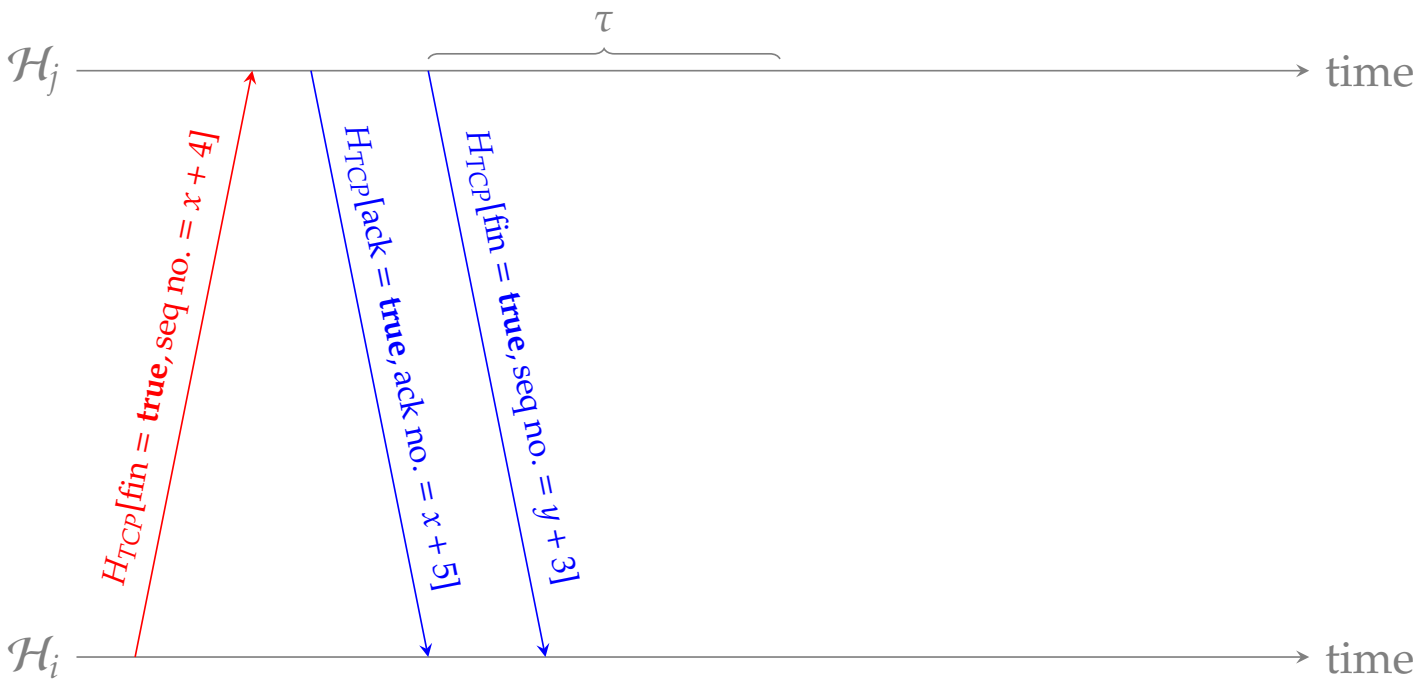
TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and
  4. connection termination.



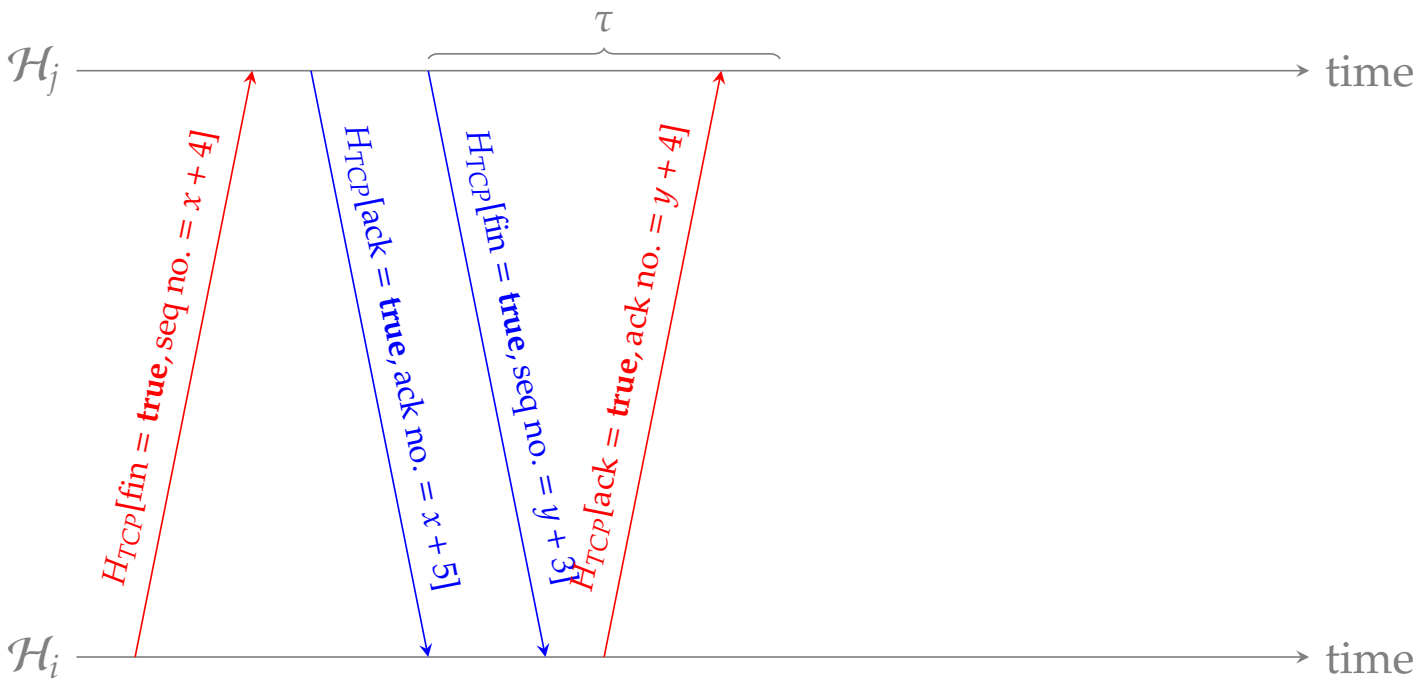
TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and
  4. connection termination.



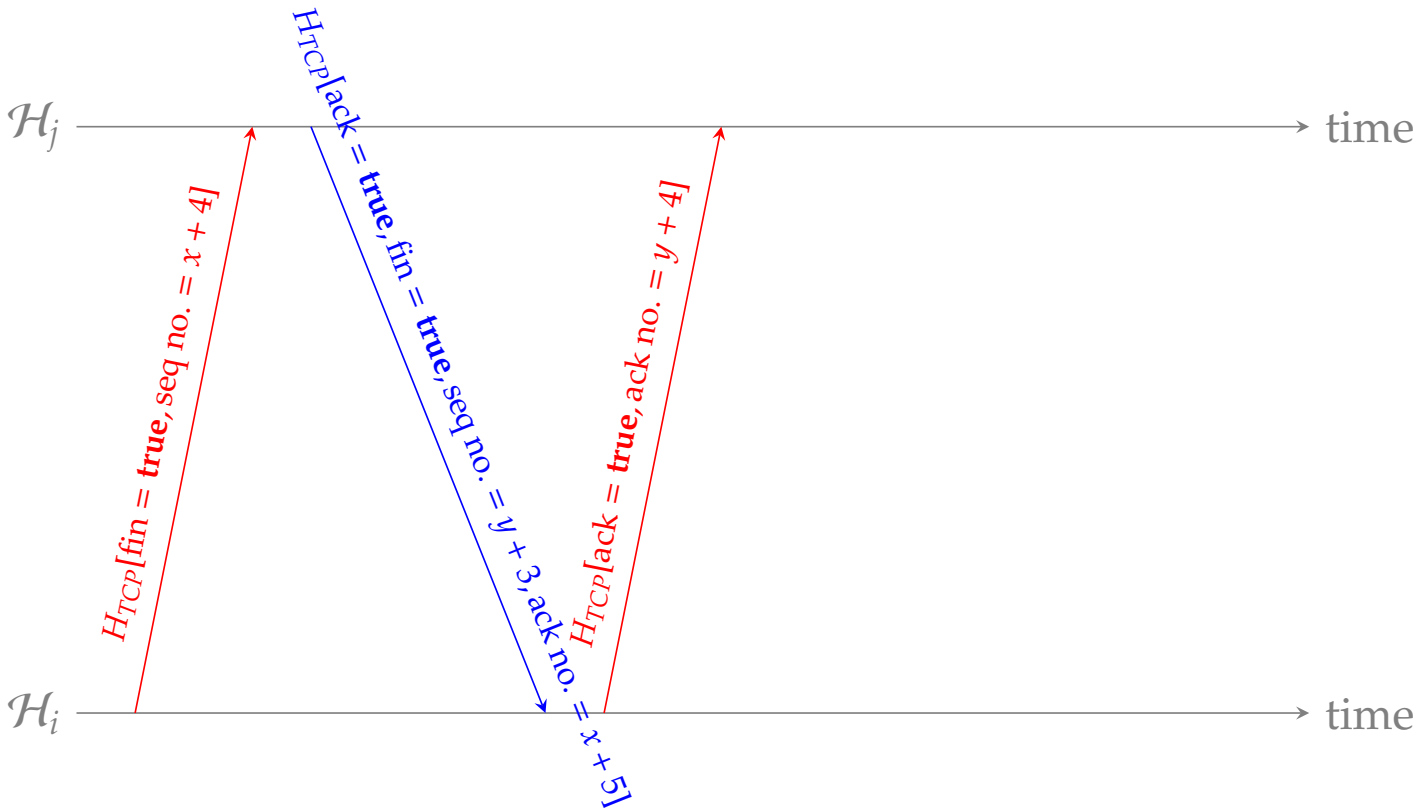
TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and
  4. connection termination.

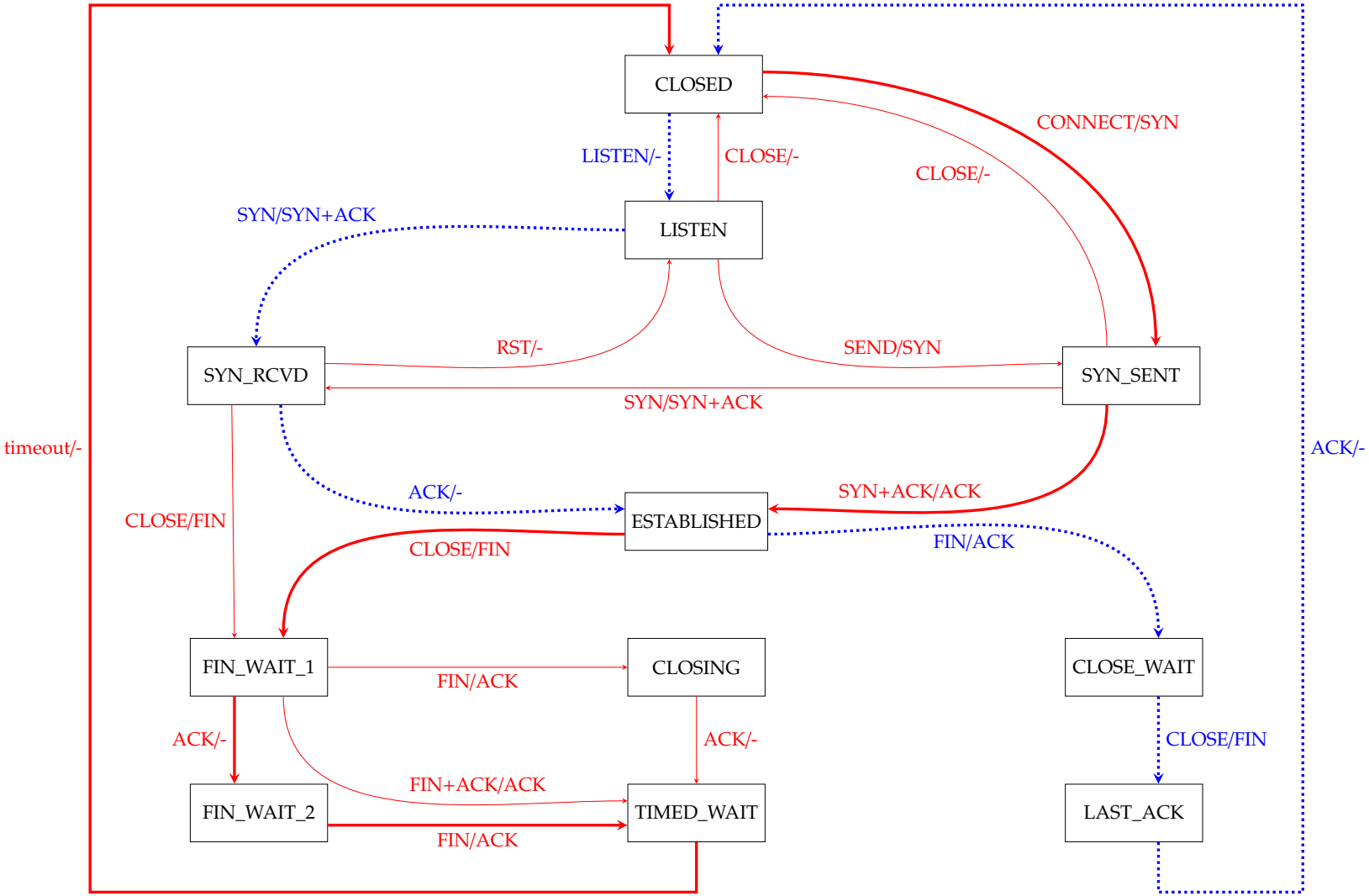


TCP (6)

- Example:
  1. connection establishment,
  2. connection reset,
  3. full-duplex communication, and
  4. connection termination.



Algorithm (**TCP state machine**)



Continued in next lecture ...

References

[1] Wikipedia: Automatic repeat request.
http://en.wikipedia.org/wiki/Automatic_repeat_request.

[2] Wikipedia: Port.
http://en.wikipedia.org/wiki/Port_(computer_networking).

[3] Wikipedia: Socket.
http://en.wikipedia.org/wiki/Socket.

[4] Wikipedia: Transmission control protocol.
http://en.wikipedia.org/wiki/Transmission_Control_Protocol.

[5] Wikipedia: User datagram protocol.
http://en.wikipedia.org/wiki/User_Datagram_Protocol.

[6] J. Postel.
User Datagram Protocol.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 768, 1980.
http://tools.ietf.org/html/rfc768.

[7] J. Postel.
Transmision Control Protocol.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 793, 1981.
http://tools.ietf.org/html/rfc793.

References

[8] K. Ramakrishnan, S. Floyd, and D. Black.
The addition of Explicit Congestion Notification (ECN) to IP.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 3168, 2001.
http://tools.ietf.org/html/rfc3168.

[9] N. Spring, D. Wetherall, and D. Ely.
Robust Explicit Congestion Notification (ECN) signaling with nonces.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 3540, 2003.
http://tools.ietf.org/html/rfc3540.

[10] W. Stallings.
Chapter 23: Transport protocols.
In *Data and Computer Communications*. Pearson, 9th edition, 2010.