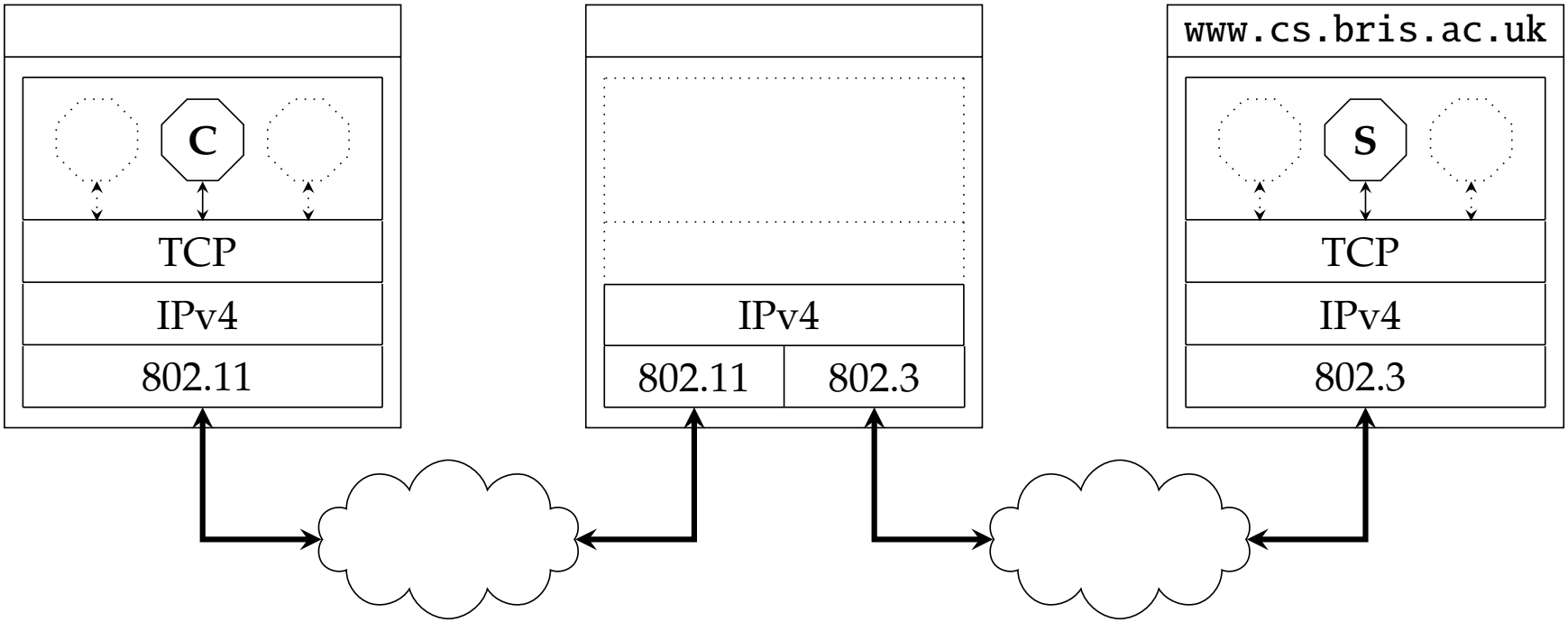
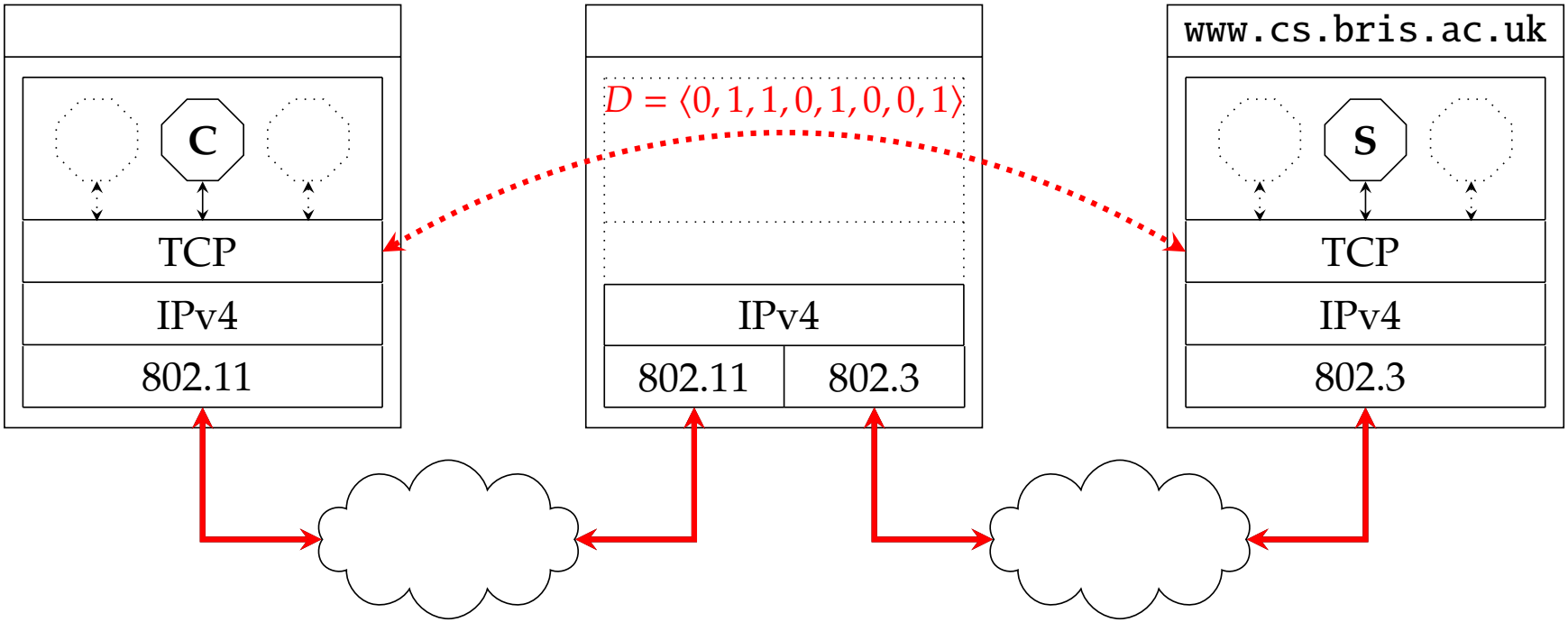


- Goal: investigate the **transport layer** e.g.,
  1. connection management,
  2. error, flow and congestion control

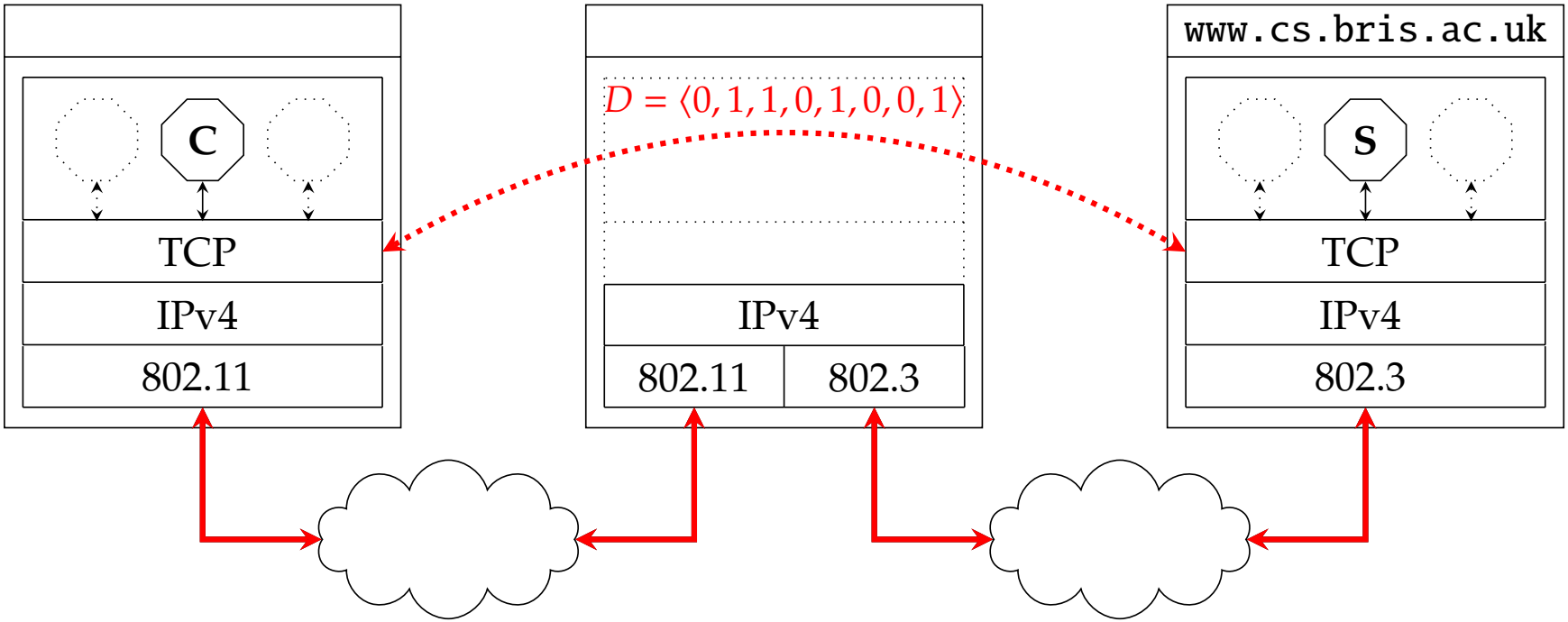
  st. applications can use end-to-end, **datagram**- or **stream**-based communication (by transmitting **datagrams** or **segments** respectively).



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

- 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

For instance, **User Datagram Protocol (UDP)** [11] is

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

Definition

For instance, **Transport Control Protocol (TCP)** [10] 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).

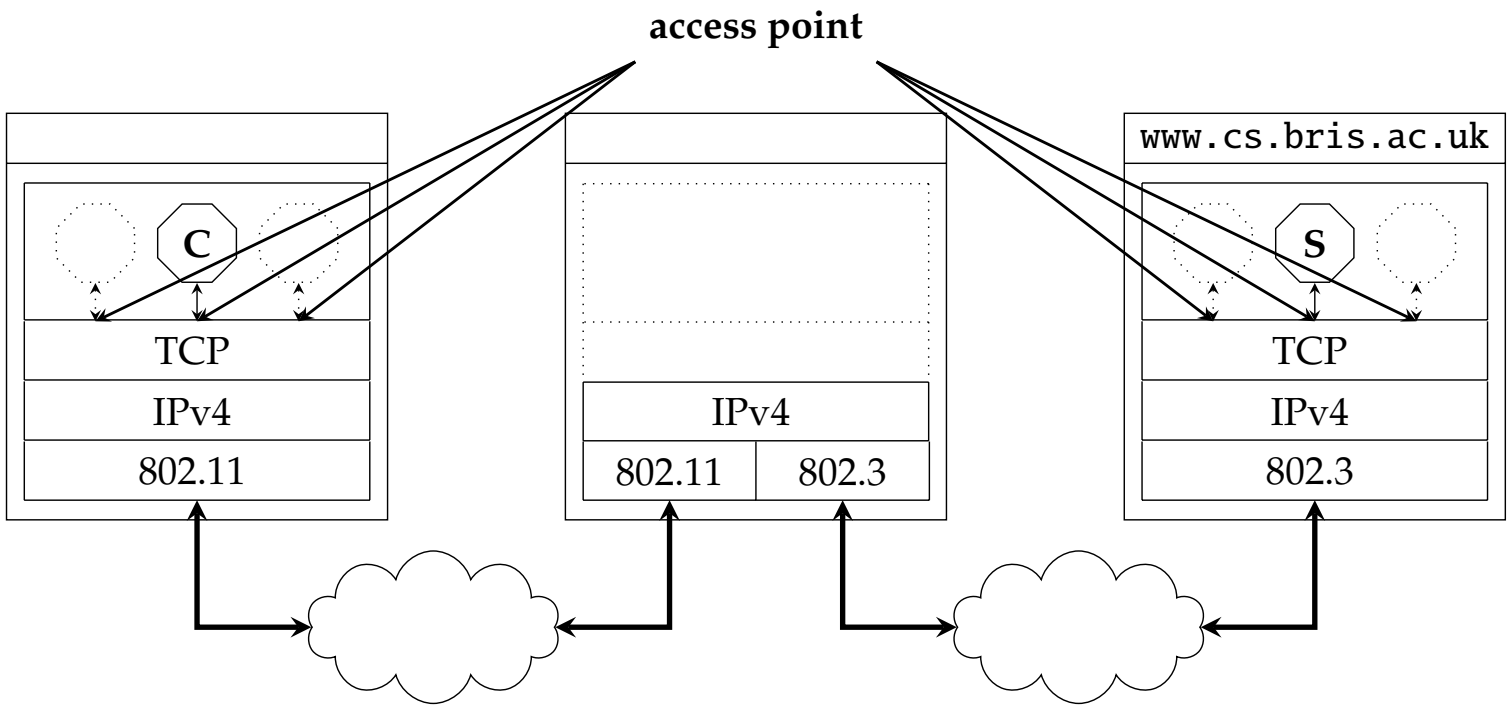
st. $\text{TCP} \gg \text{UDP} \simeq \text{IP} + \epsilon$.

Definition

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

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).

Concepts (2)

Definition

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) 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)

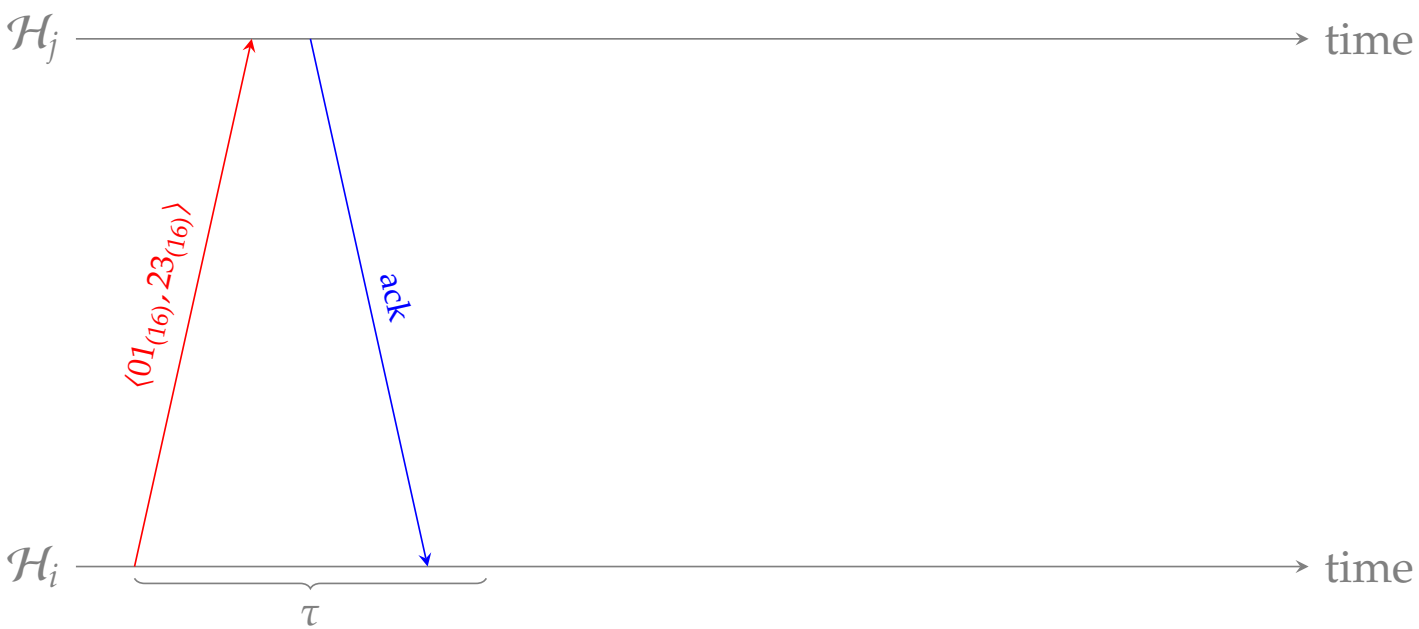
e.g., using the **stop-and-wait protocol** .

Definition

Automatic Repeat reQuest (ARQ) 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** .

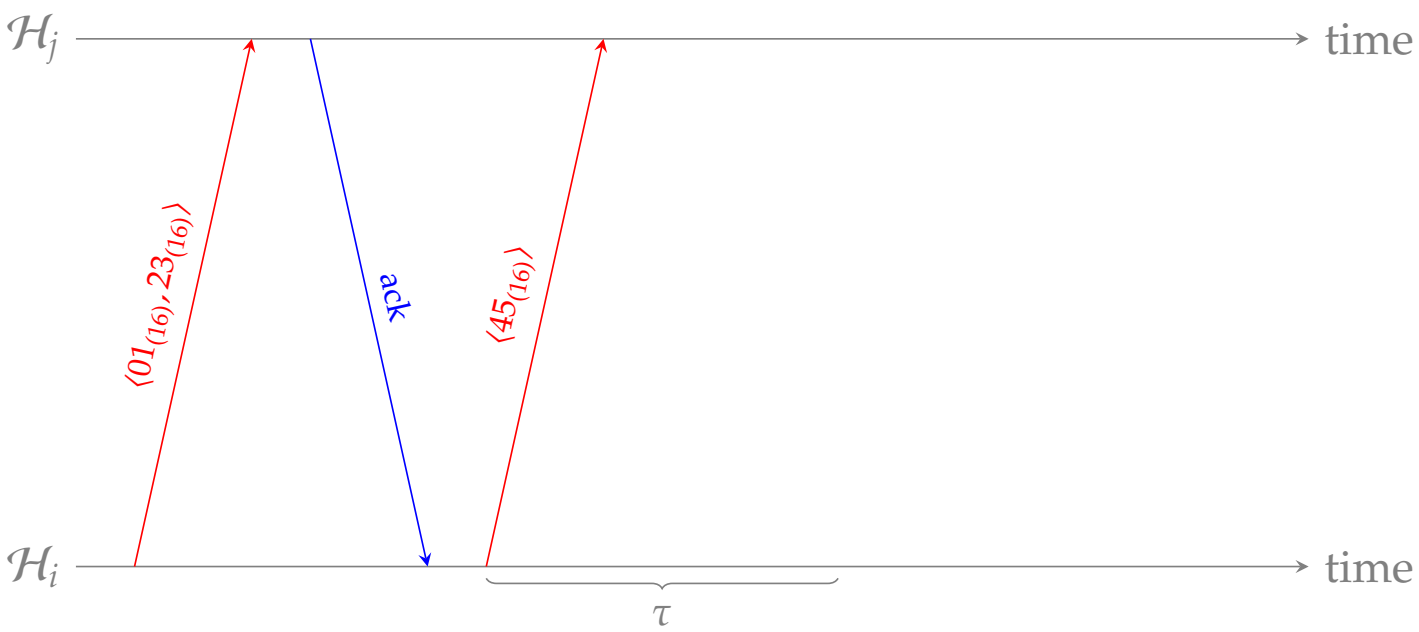


Definition

Automatic Repeat reQuest (ARQ) 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** .

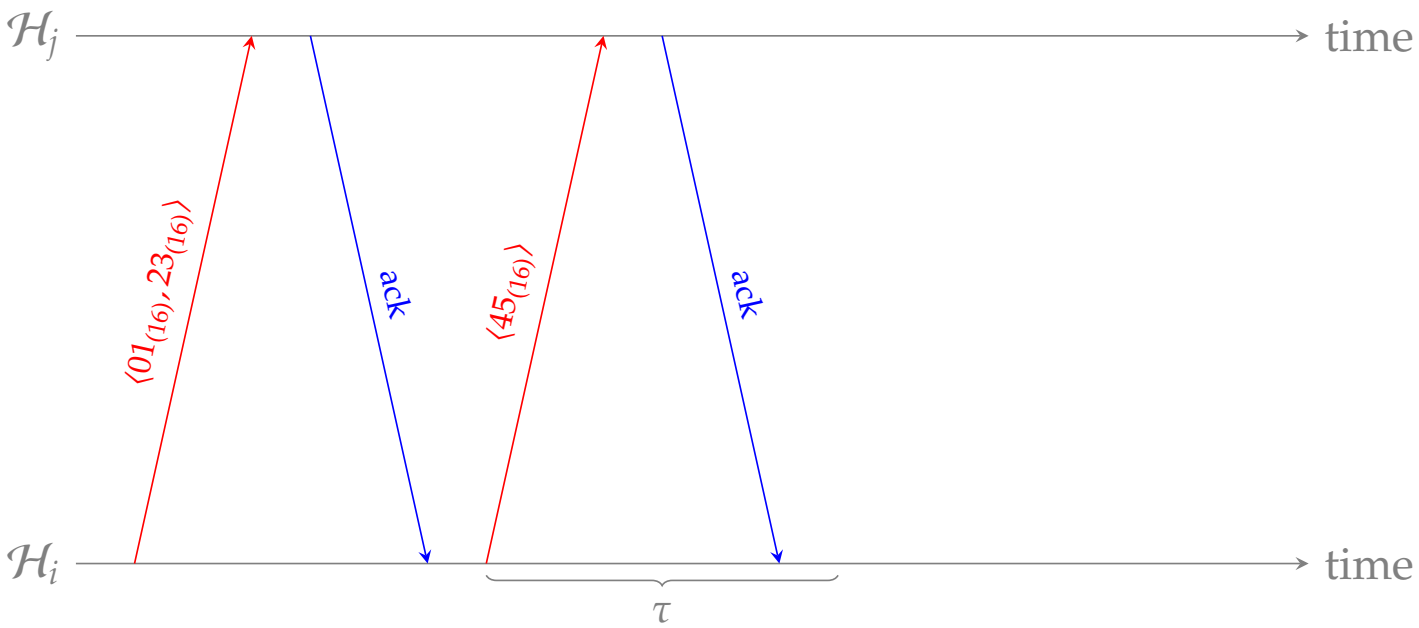


Definition

Automatic Repeat reQuest (ARQ) 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** .



Concepts (2)

Definition

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) 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)

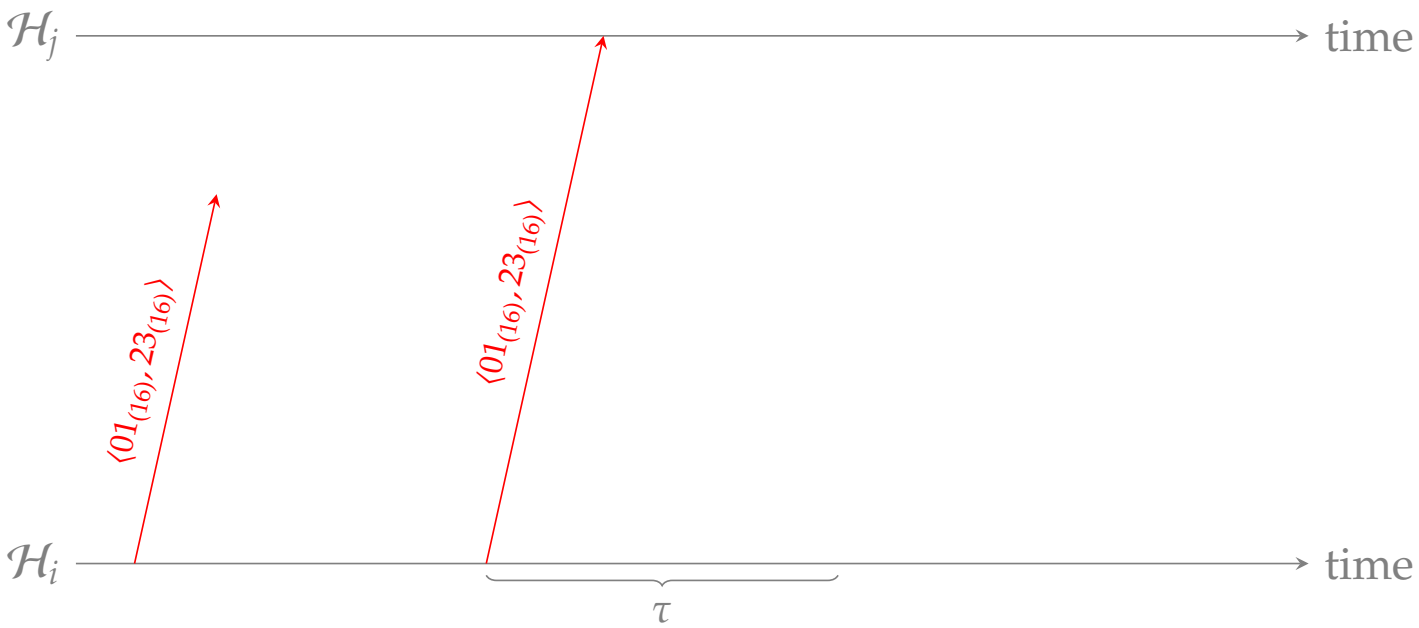
e.g., using the **stop-and-wait protocol** .

Definition

Automatic Repeat reQuest (ARQ) 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** .

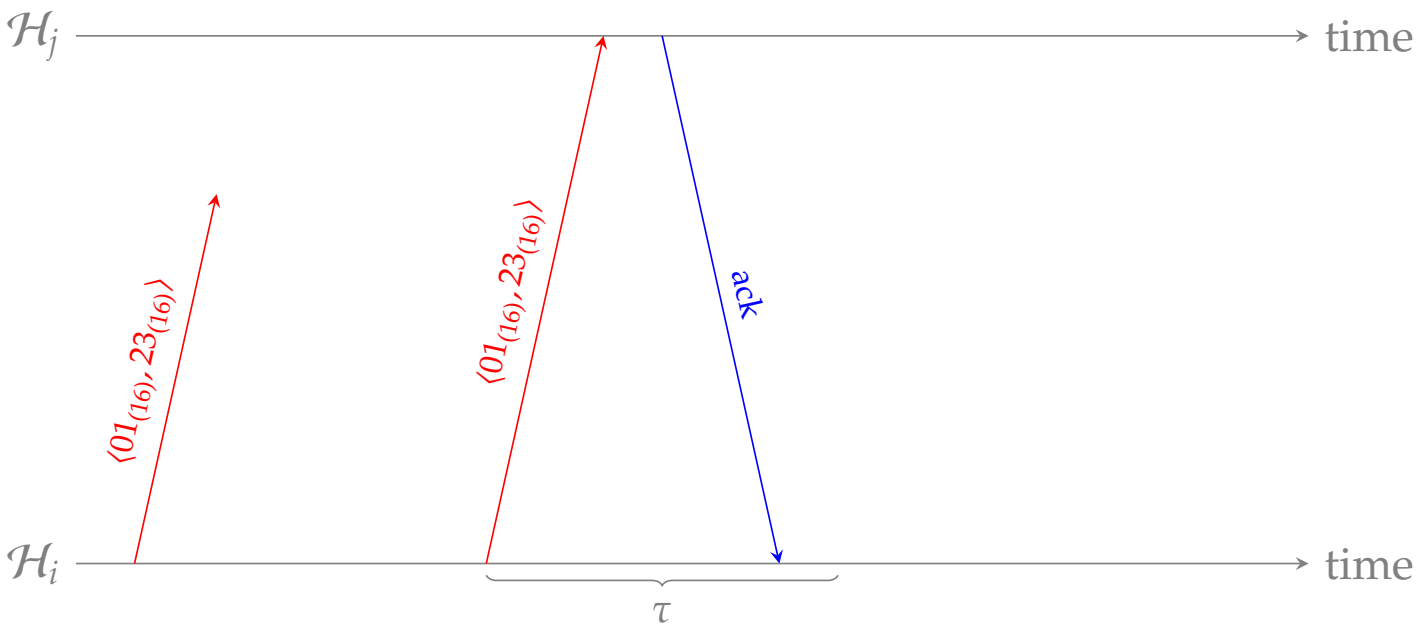


Definition

Automatic Repeat reQuest (ARQ) 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** .

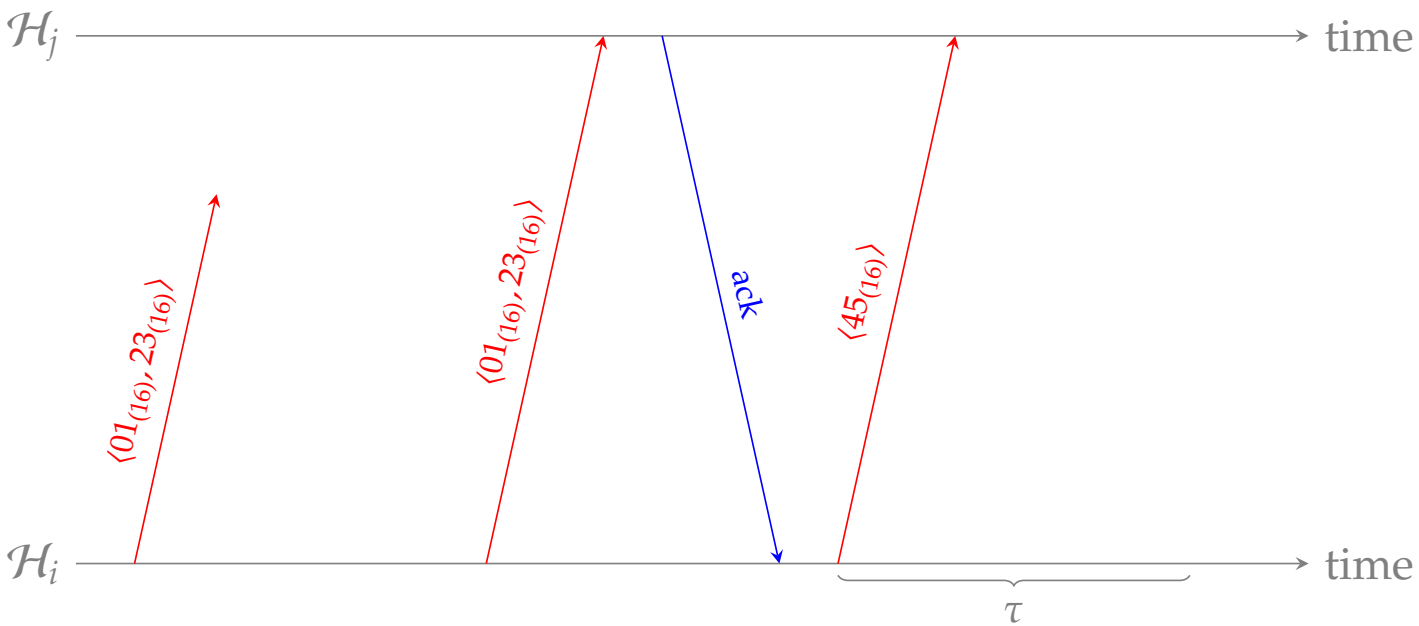


Definition

Automatic Repeat reQuest (ARQ) 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** .

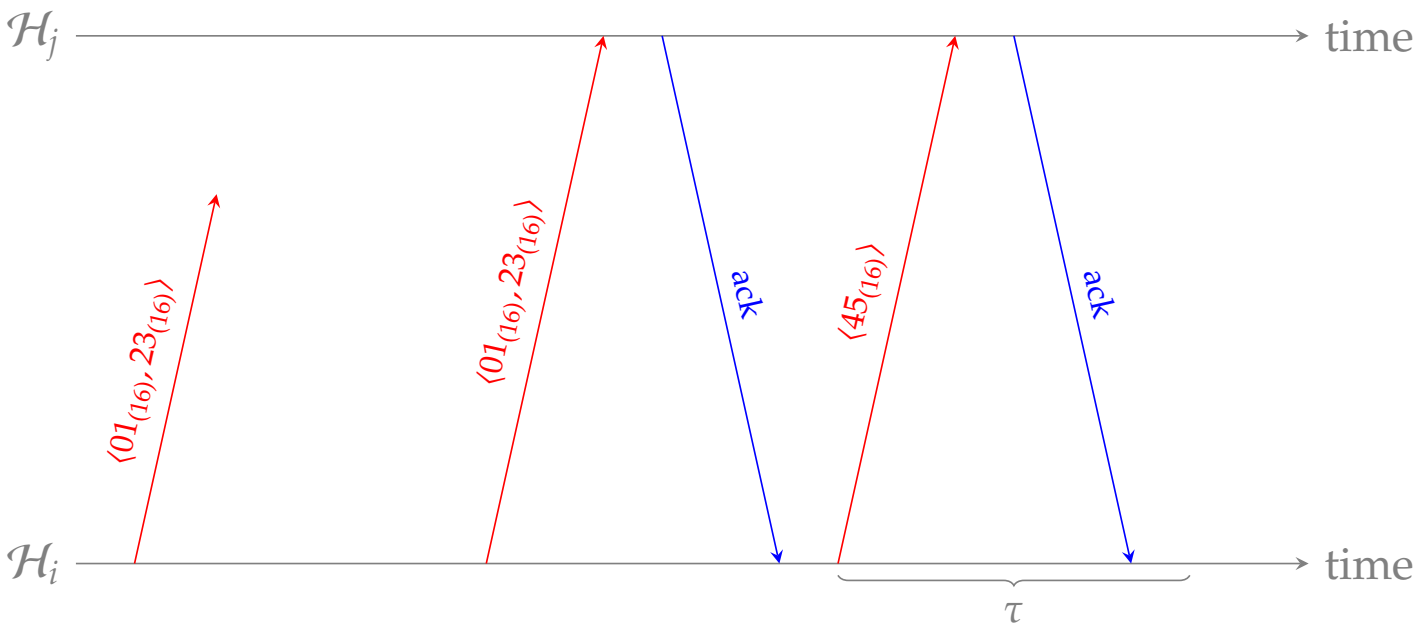


Definition

Automatic Repeat reQuest (ARQ) 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** .



Concepts (2)

Definition

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) 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)

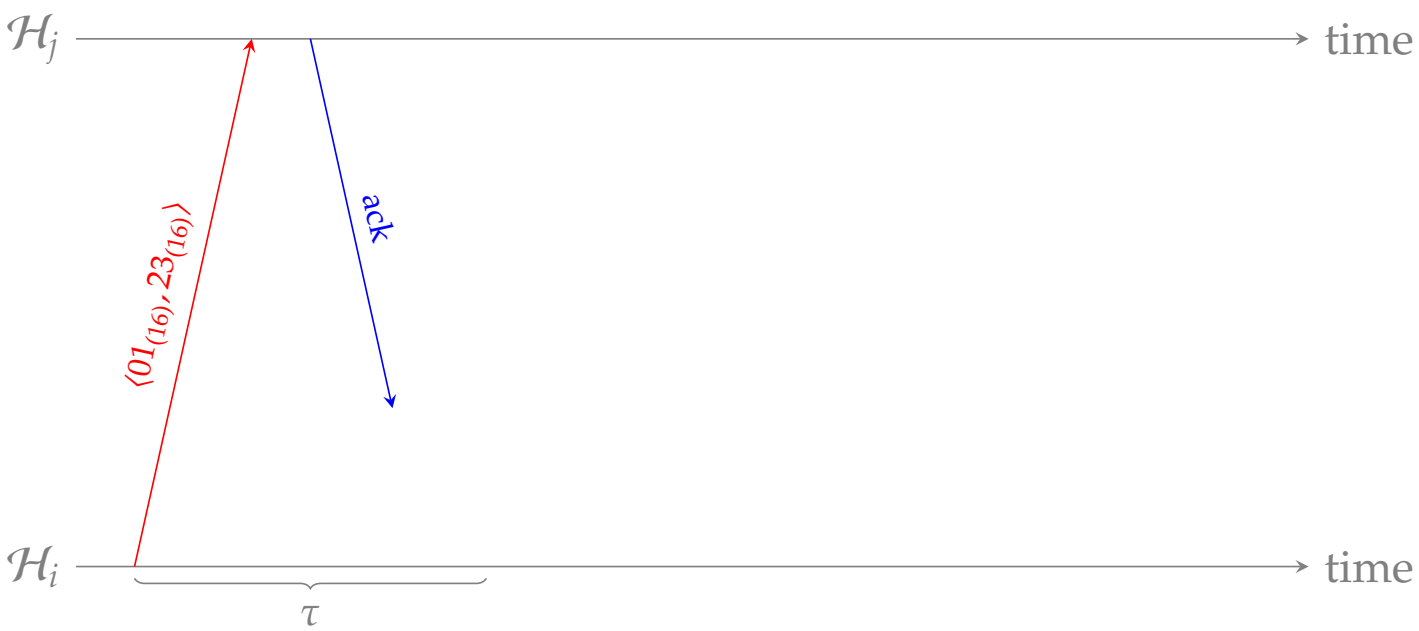
e.g., using the **stop-and-wait protocol** .

Definition

Automatic Repeat reQuest (ARQ) 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** .

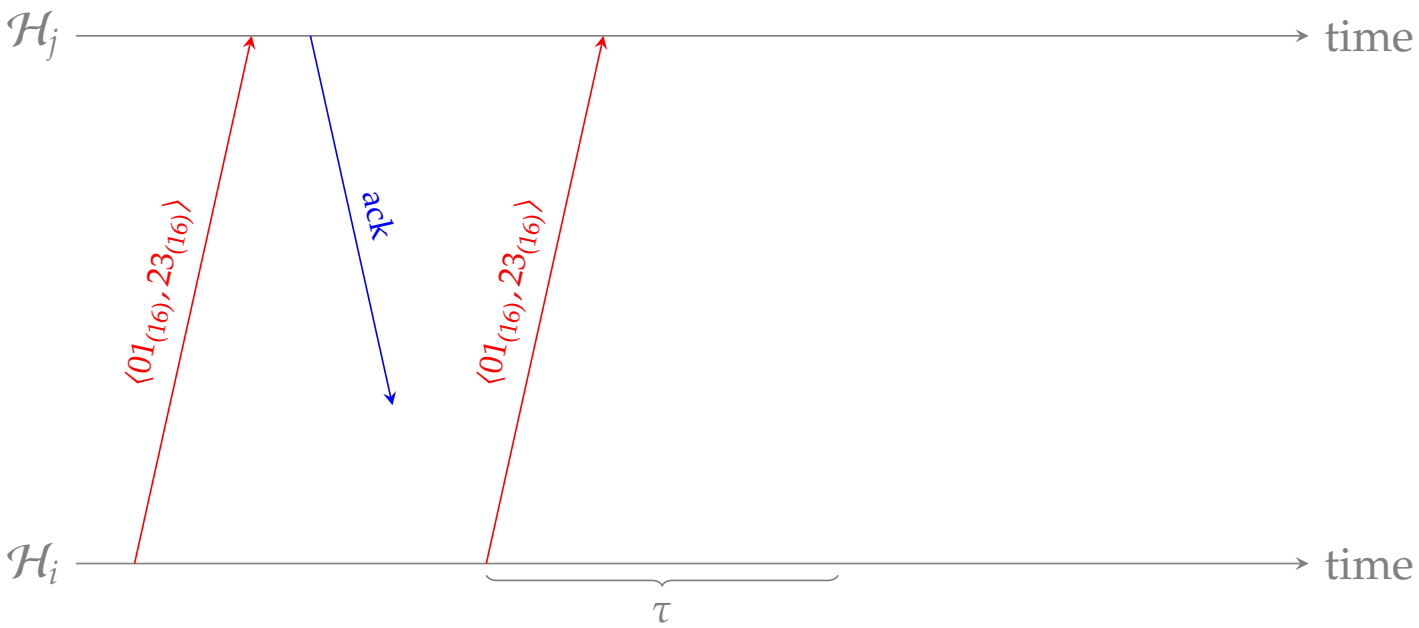


Definition

Automatic Repeat reQuest (ARQ) 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** .

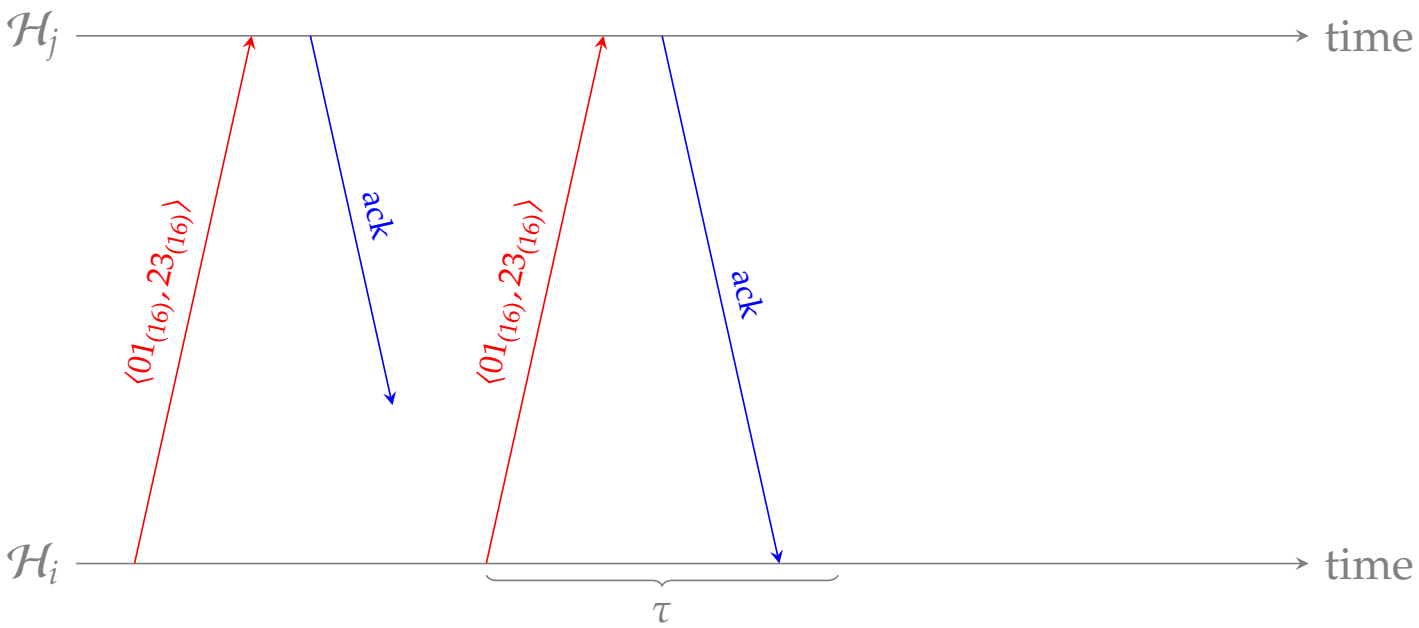


Definition

Automatic Repeat reQuest (ARQ) 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** .

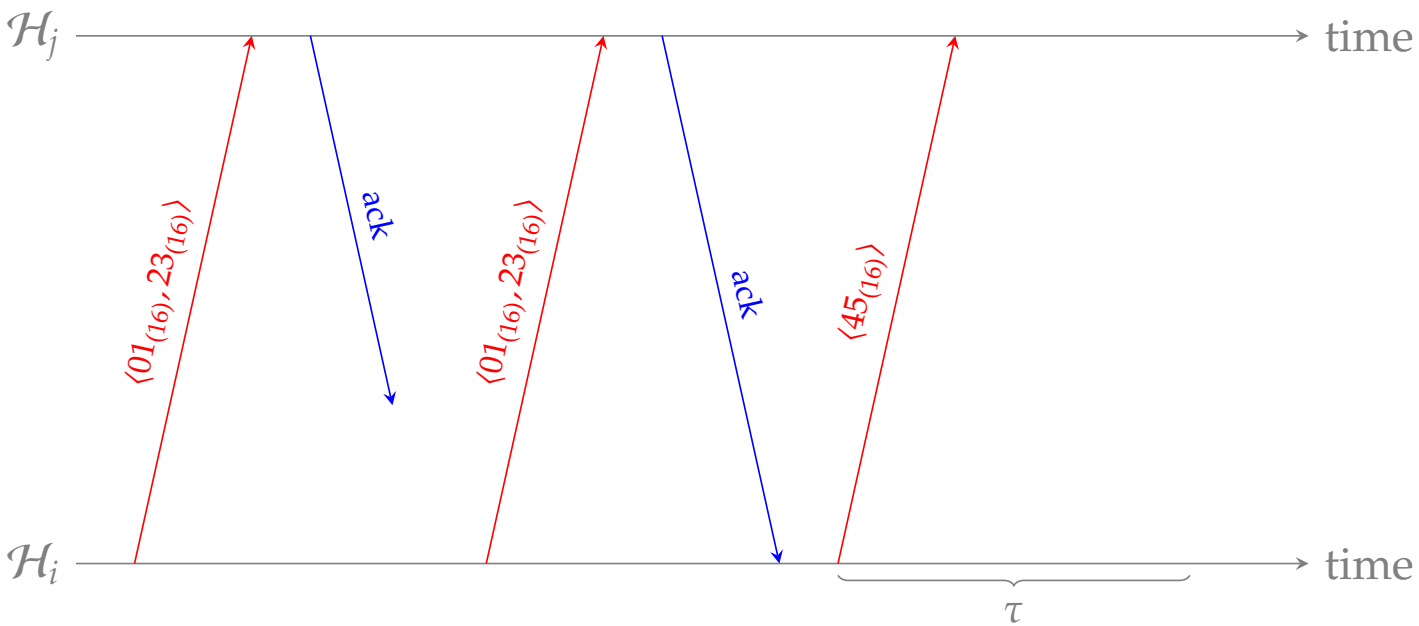


Definition

Automatic Repeat reQuest (ARQ) 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** .

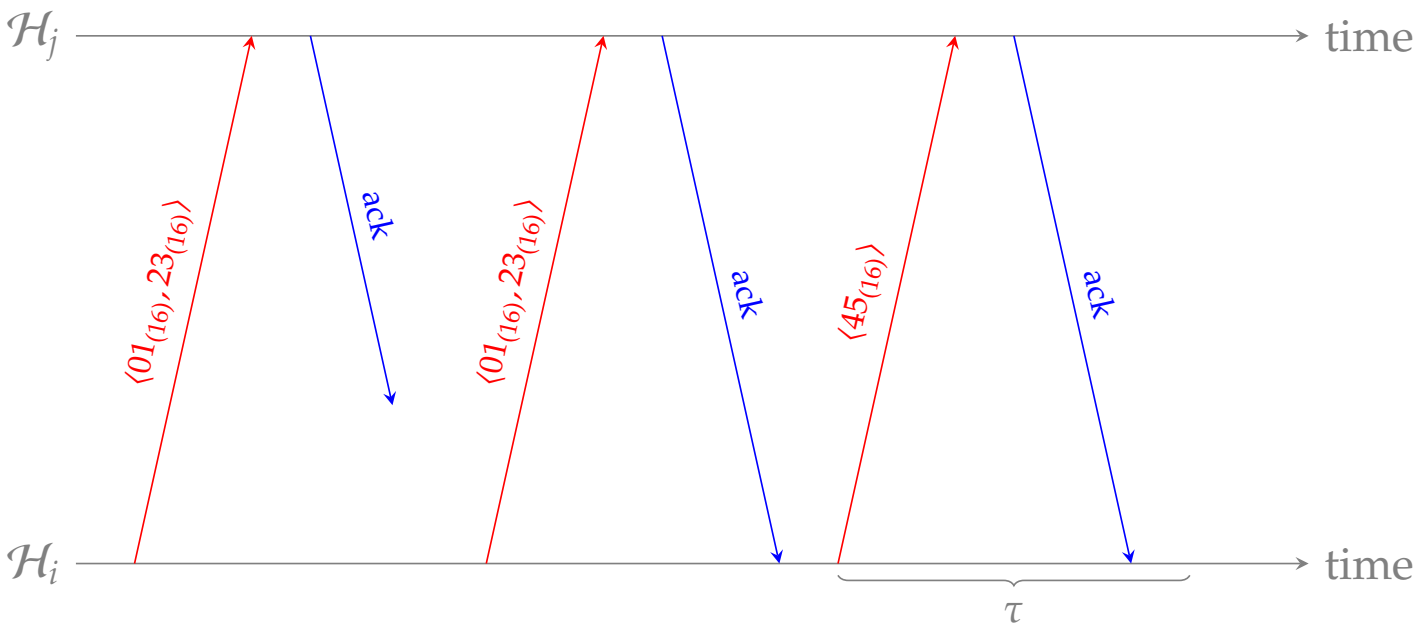


Definition

Automatic Repeat reQuest (ARQ) 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** .



Concepts (2)

Definition

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) 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** .

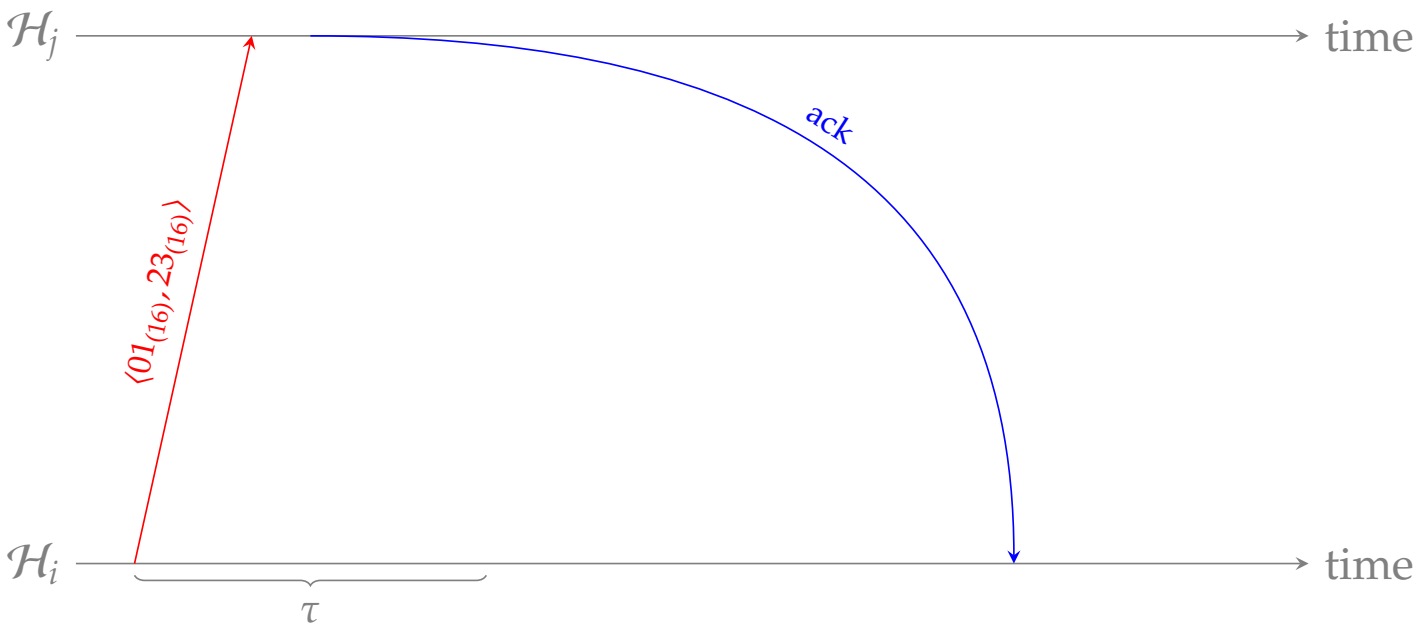
Concepts (2)

Definition

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** .



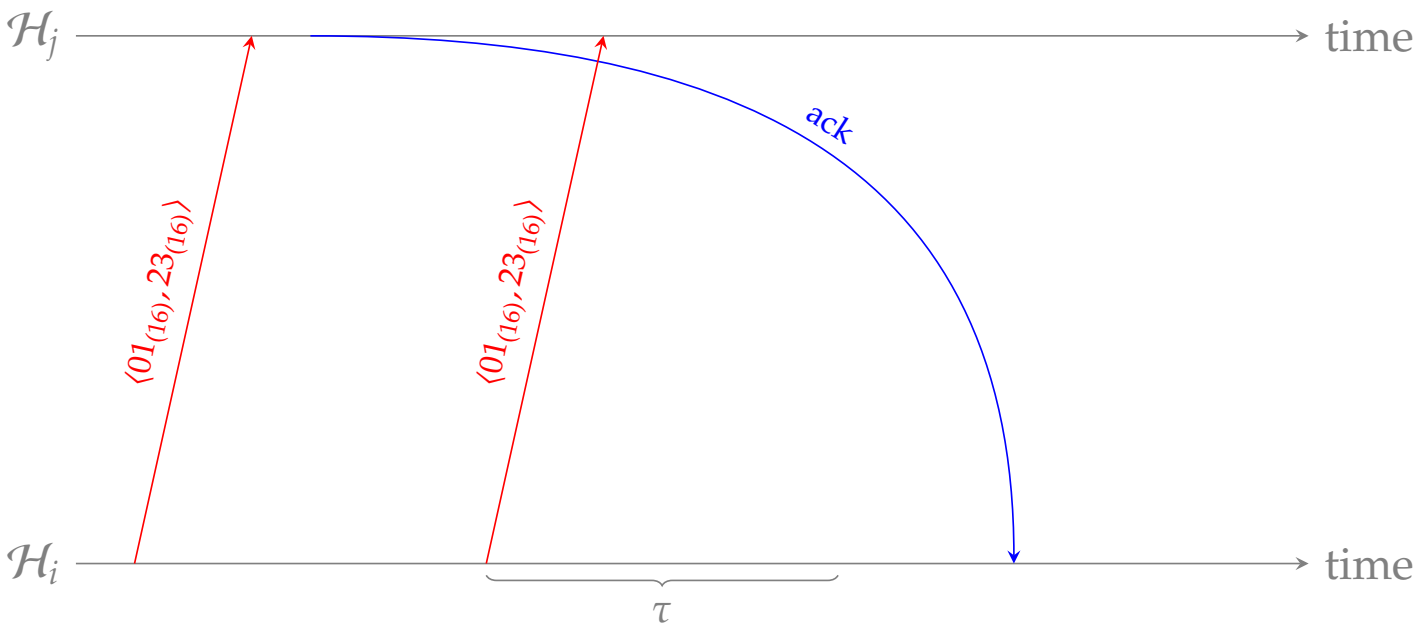
Concepts (2)

Definition

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** .

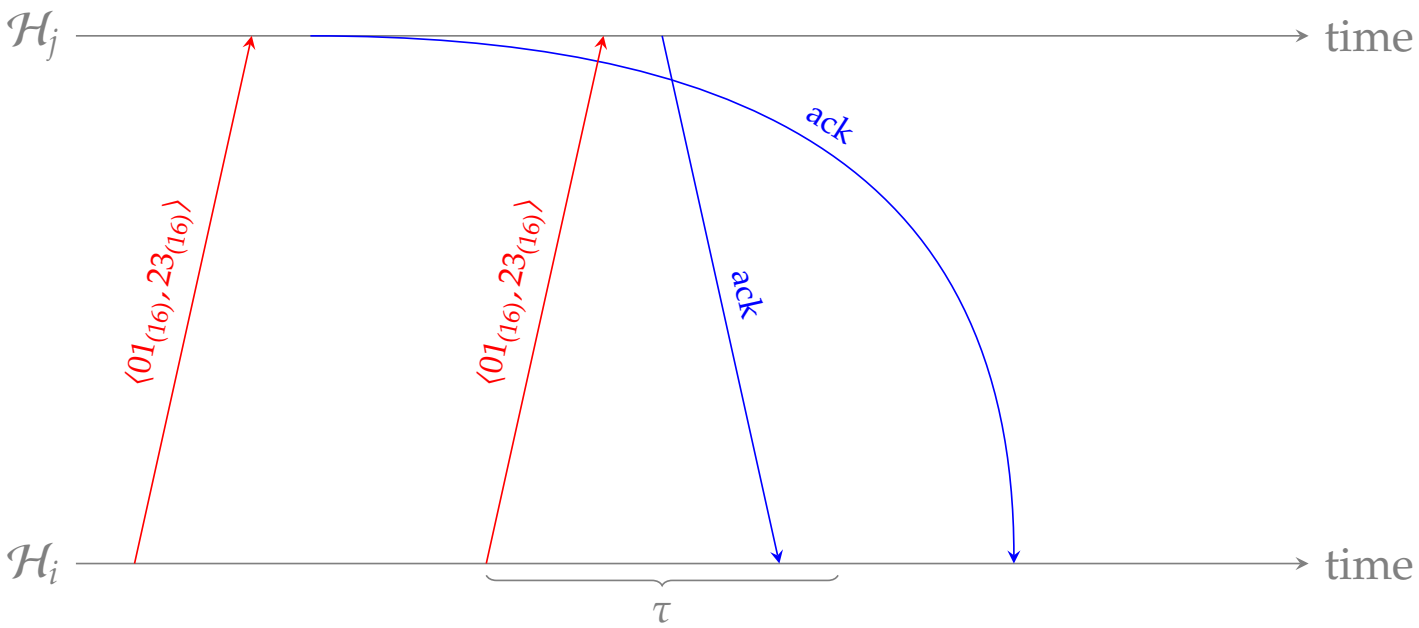


Definition

Automatic Repeat reQuest (ARQ) 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** .

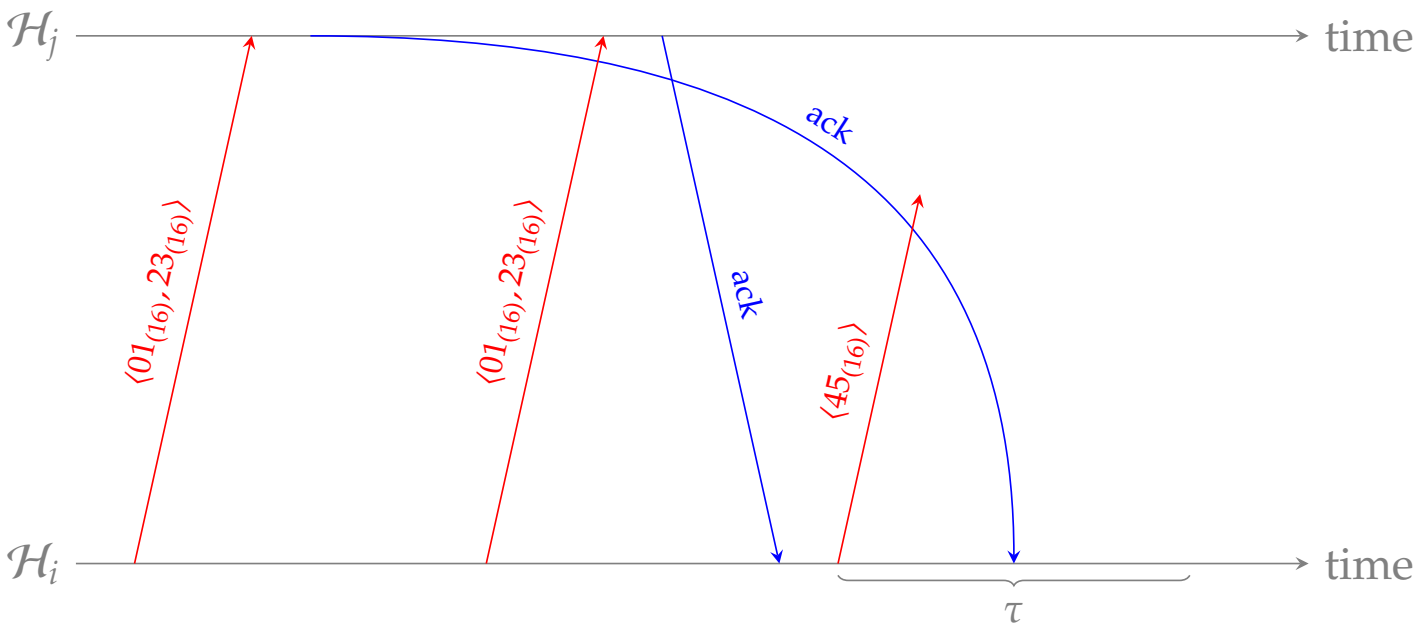


Definition

Automatic Repeat reQuest (ARQ) 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** .

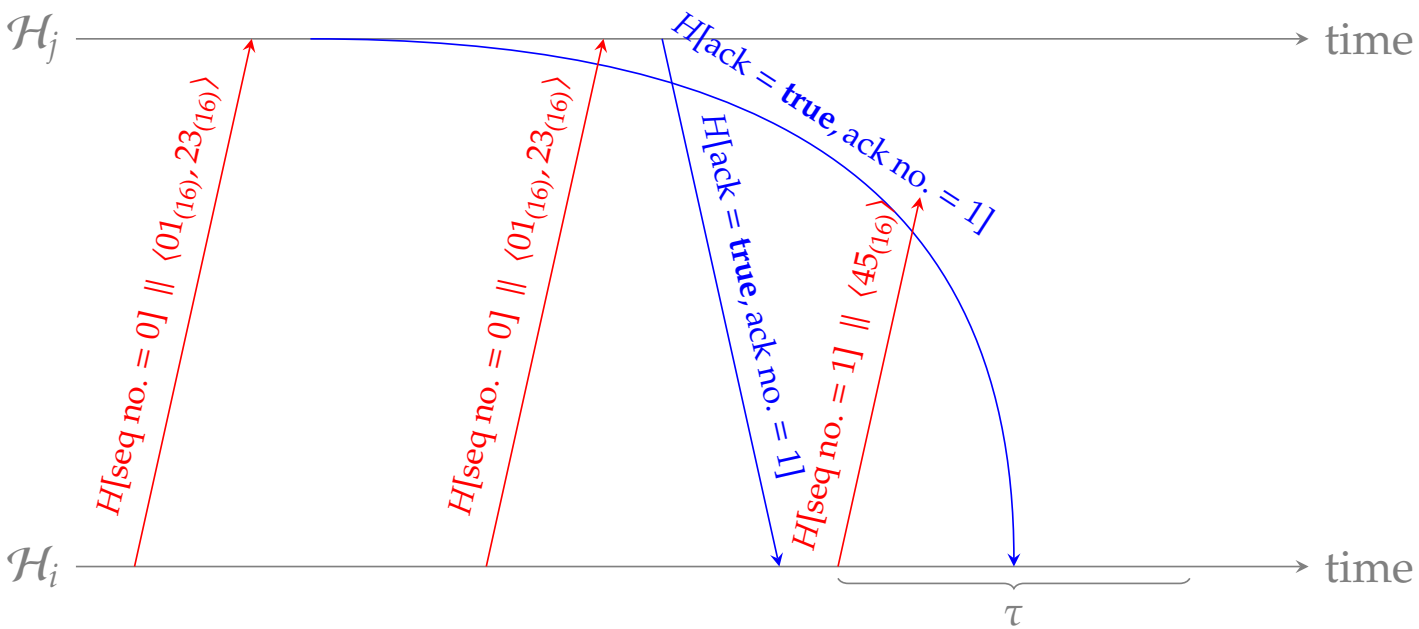


Definition

Automatic Repeat reQuest (ARQ) 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**).

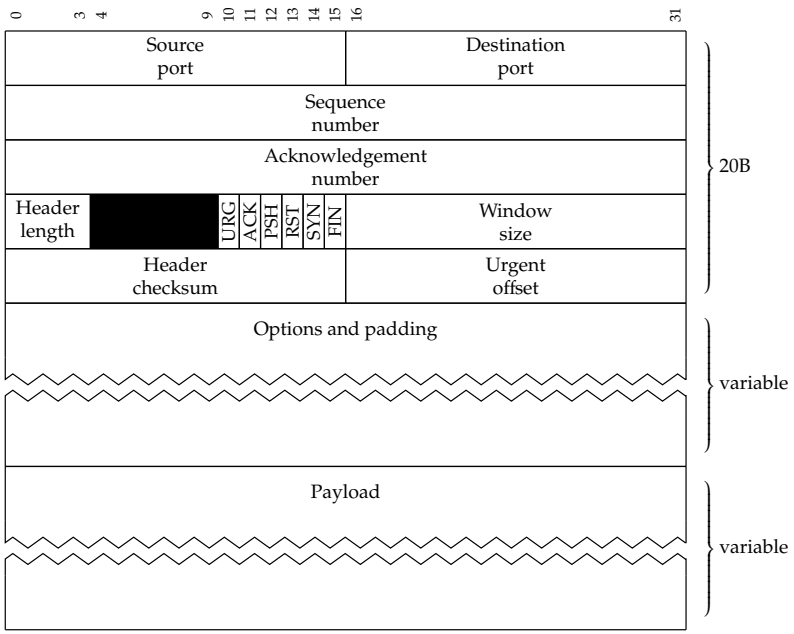


TCP (1)

- 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**.

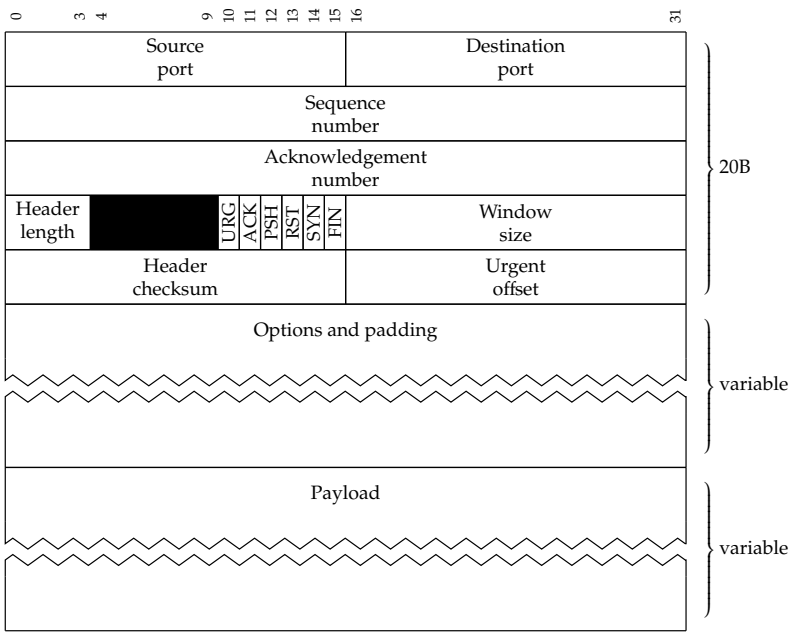
Data Structure (**TCP segment** [10, Section 3.1])



The data structure includes:

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

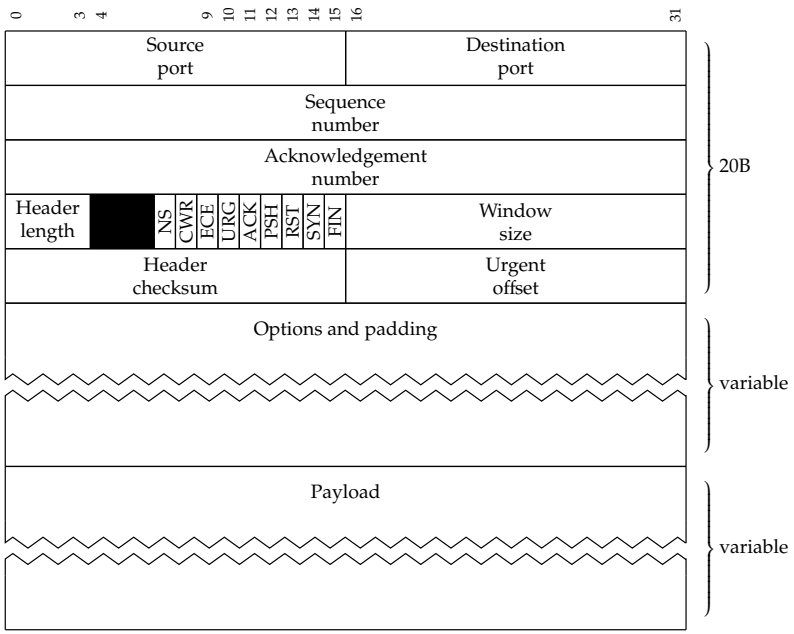
Data Structure (**TCP segment** [10, Section 3.1])



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** [10, Section 3.1]+[12, Section 6]+[13, Section 9])

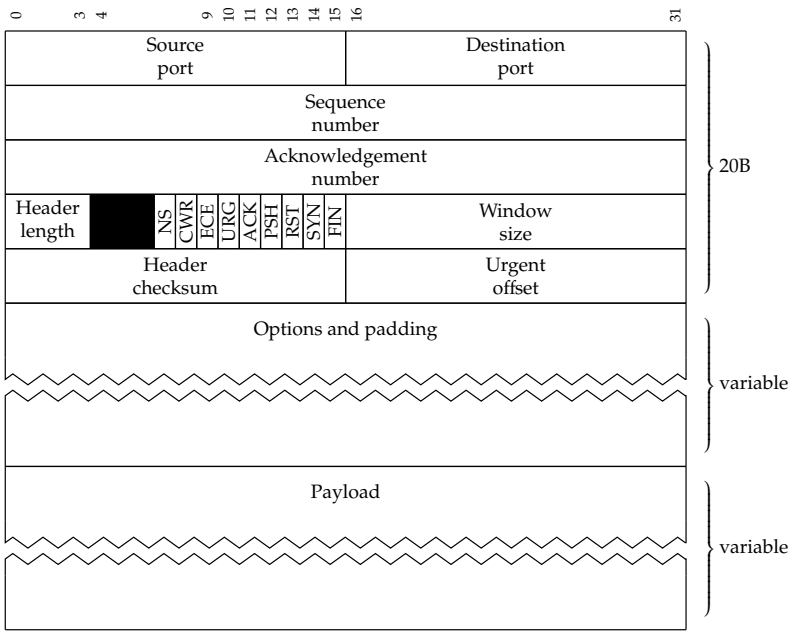


The data structure includes:

- 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)**.

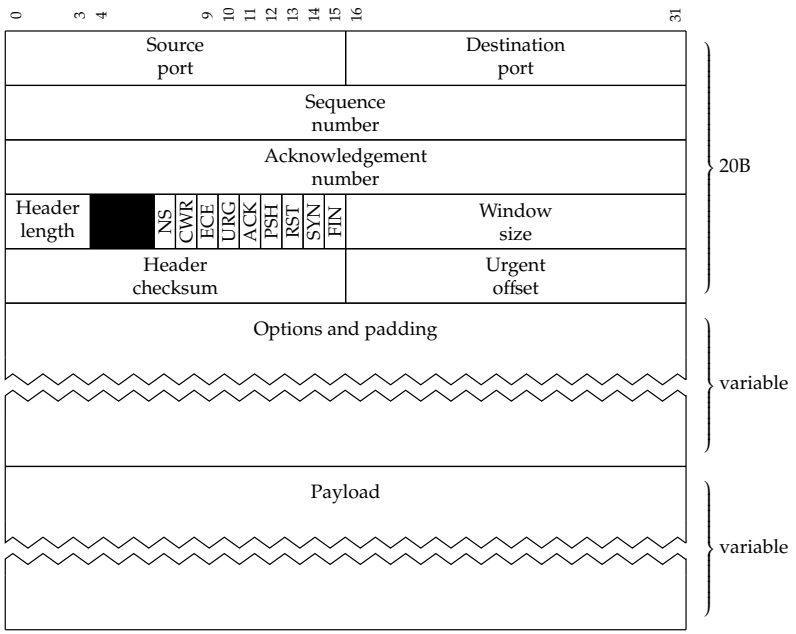
Data Structure (**TCP segment** [10, Section 3.1]+[12, Section 6]+[13, Section 9])



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.

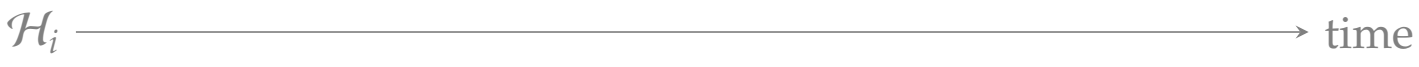
Data Structure (**TCP segment** [10, Section 3.1]+[12, Section 6]+[13, Section 9])



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)

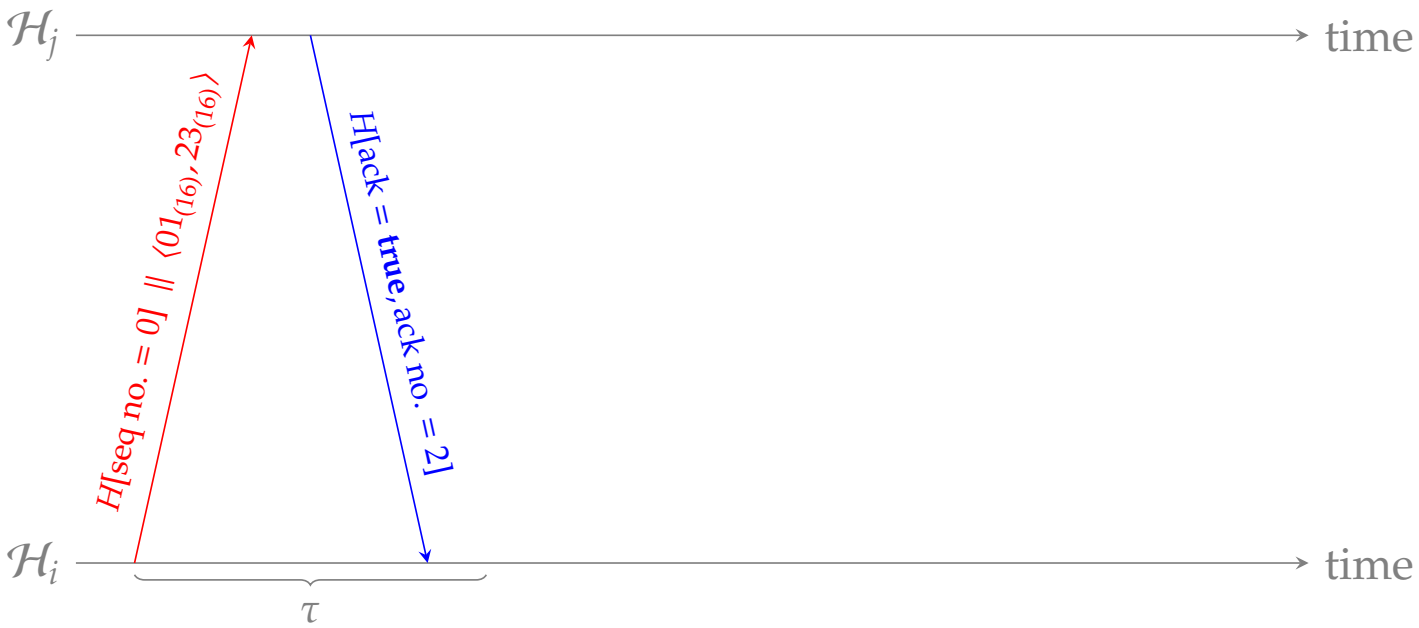


- Question: why have sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)

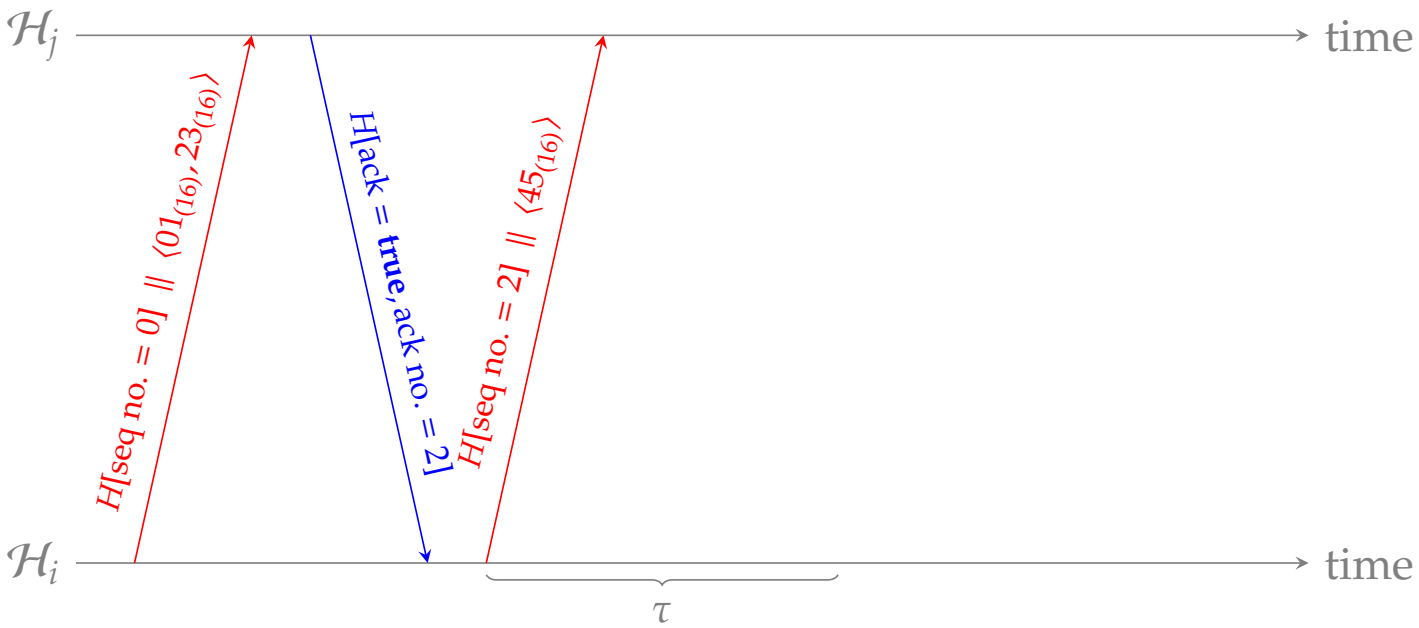
- Question: why have sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



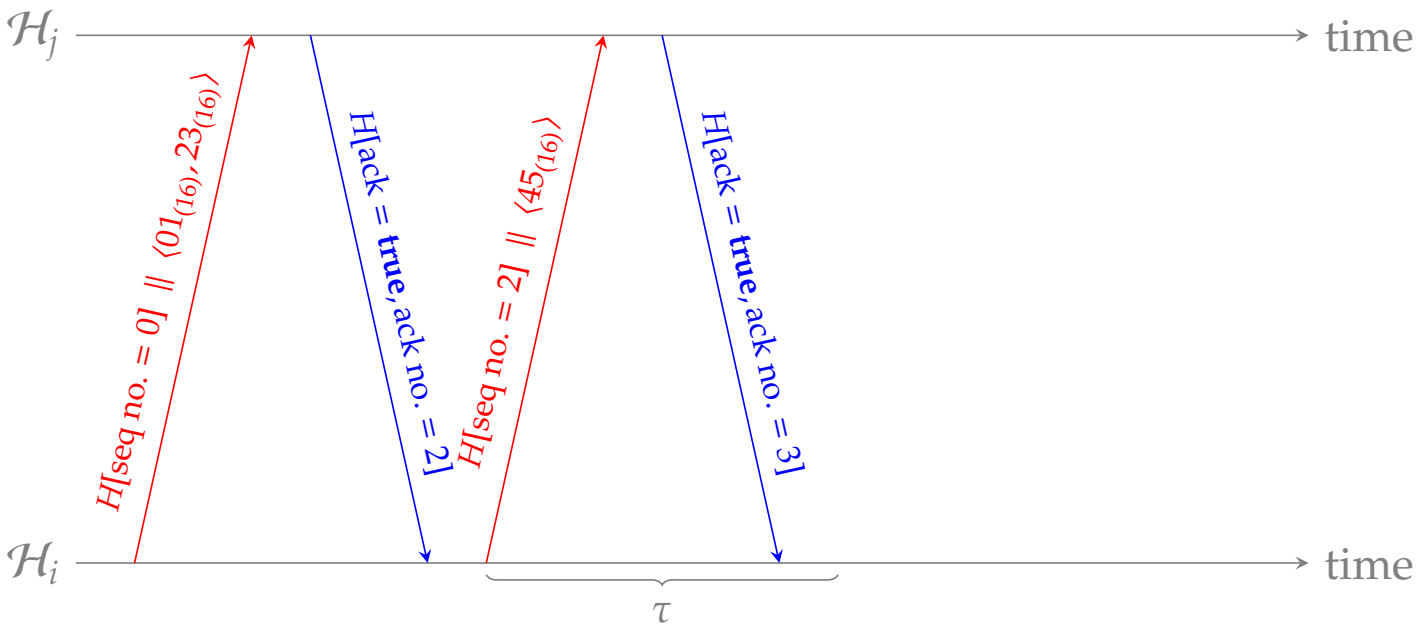
- Question: why have sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



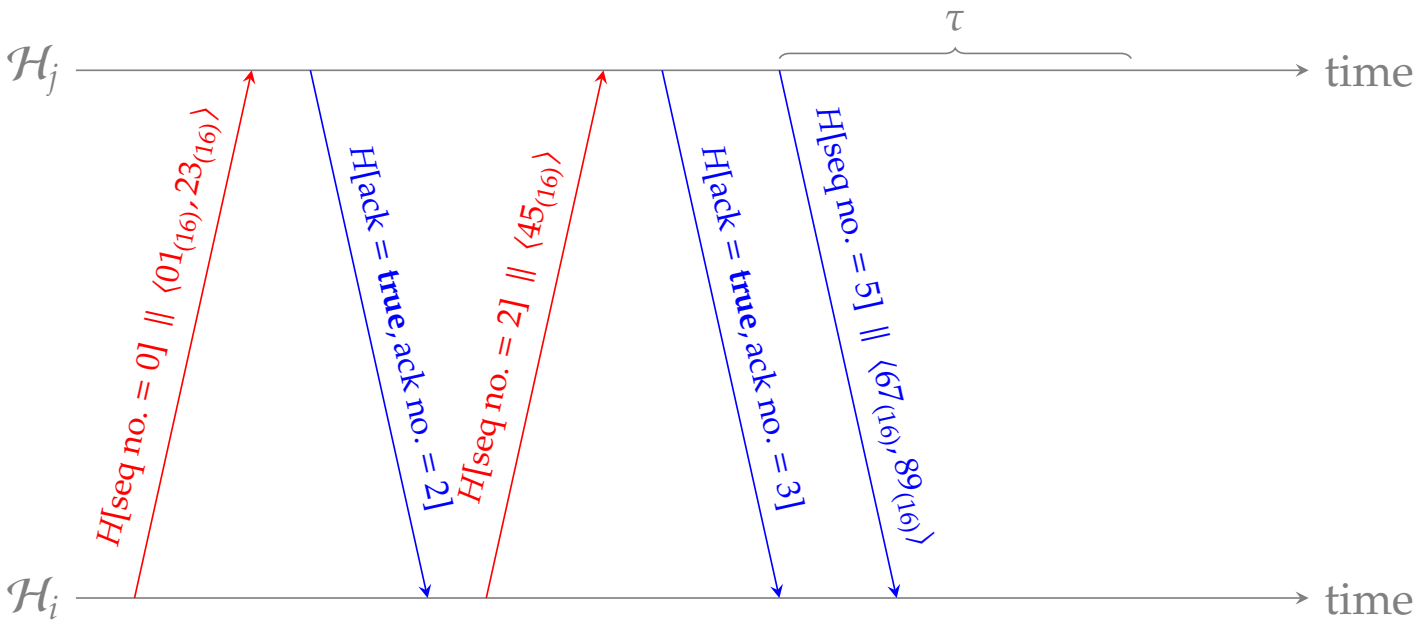
- Question: why have sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



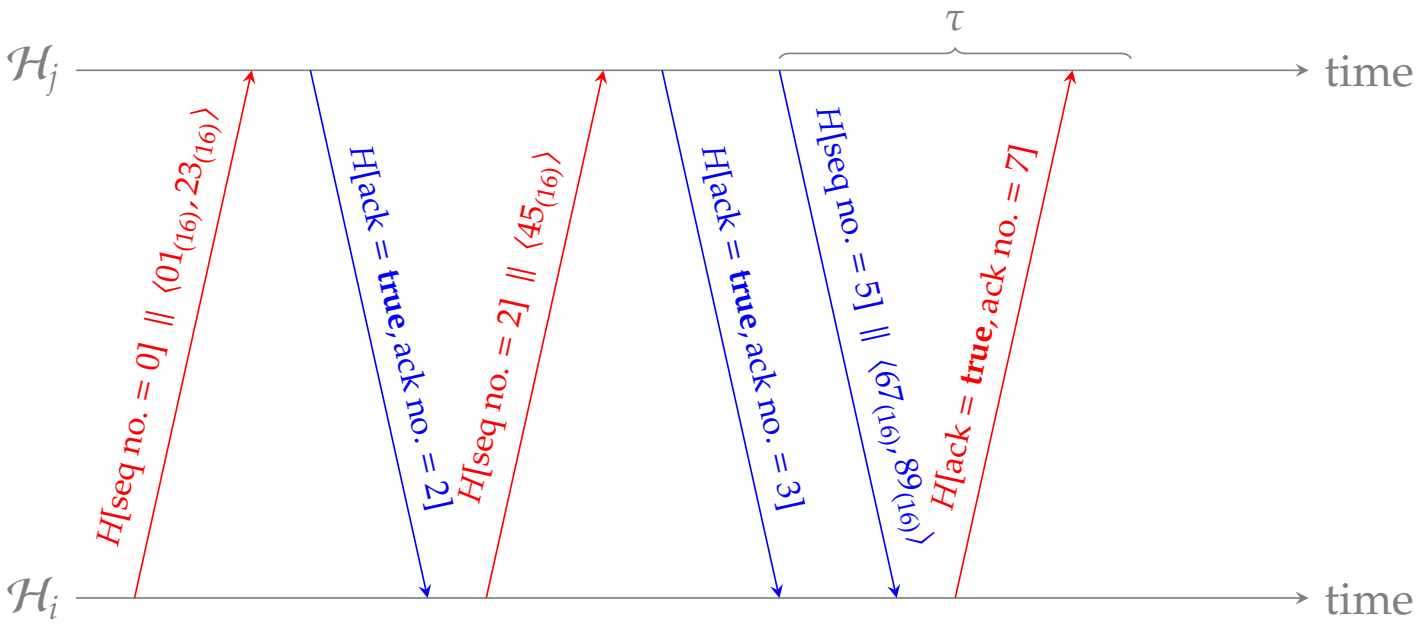
- Question: why have sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



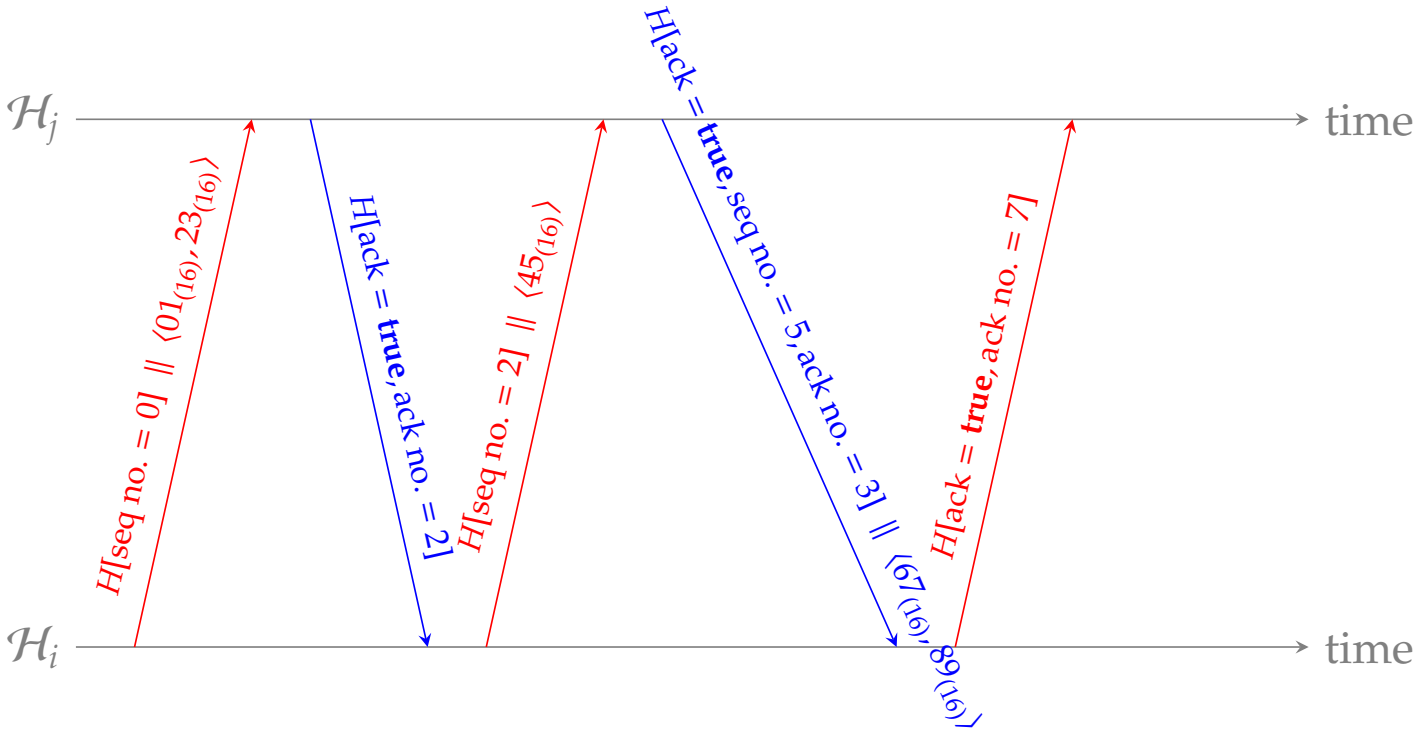
- Question: why have sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

TCP (5)



- Question: why have 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.

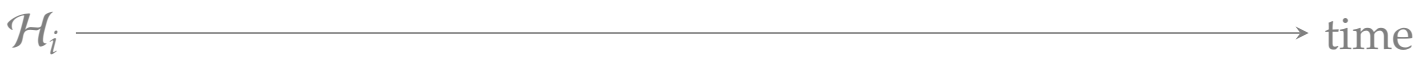
TCP (5)



- Question: why have 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:

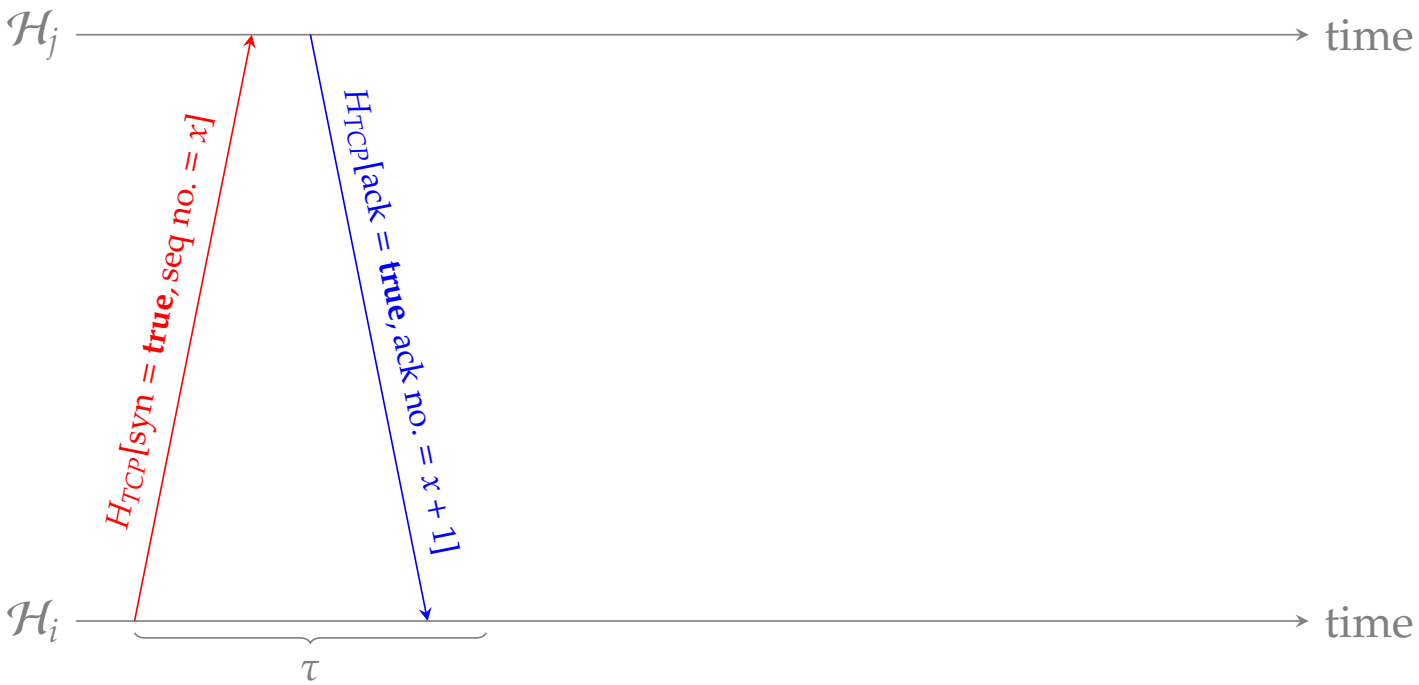


TCP (6)

- Example:
  1. connection establishment,

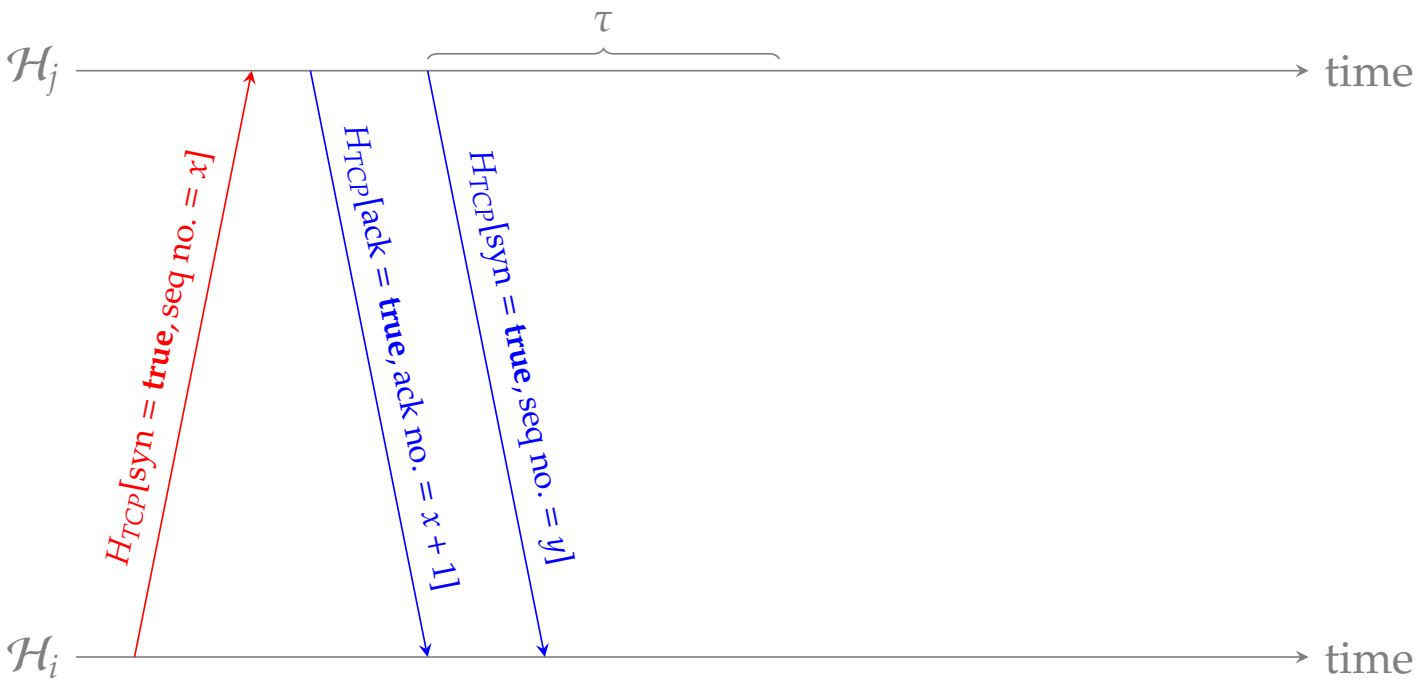
TCP (6)

- Example:
  1. connection establishment,



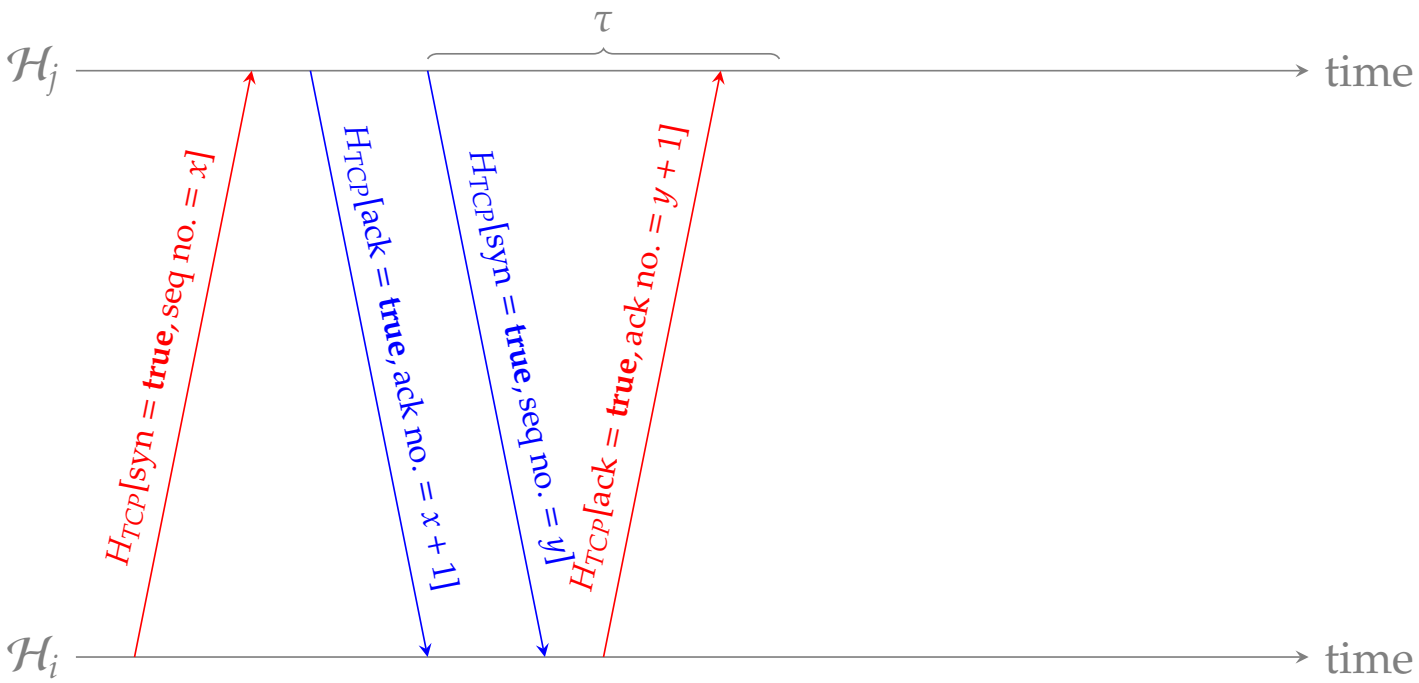
TCP (6)

- Example:
  1. connection establishment,



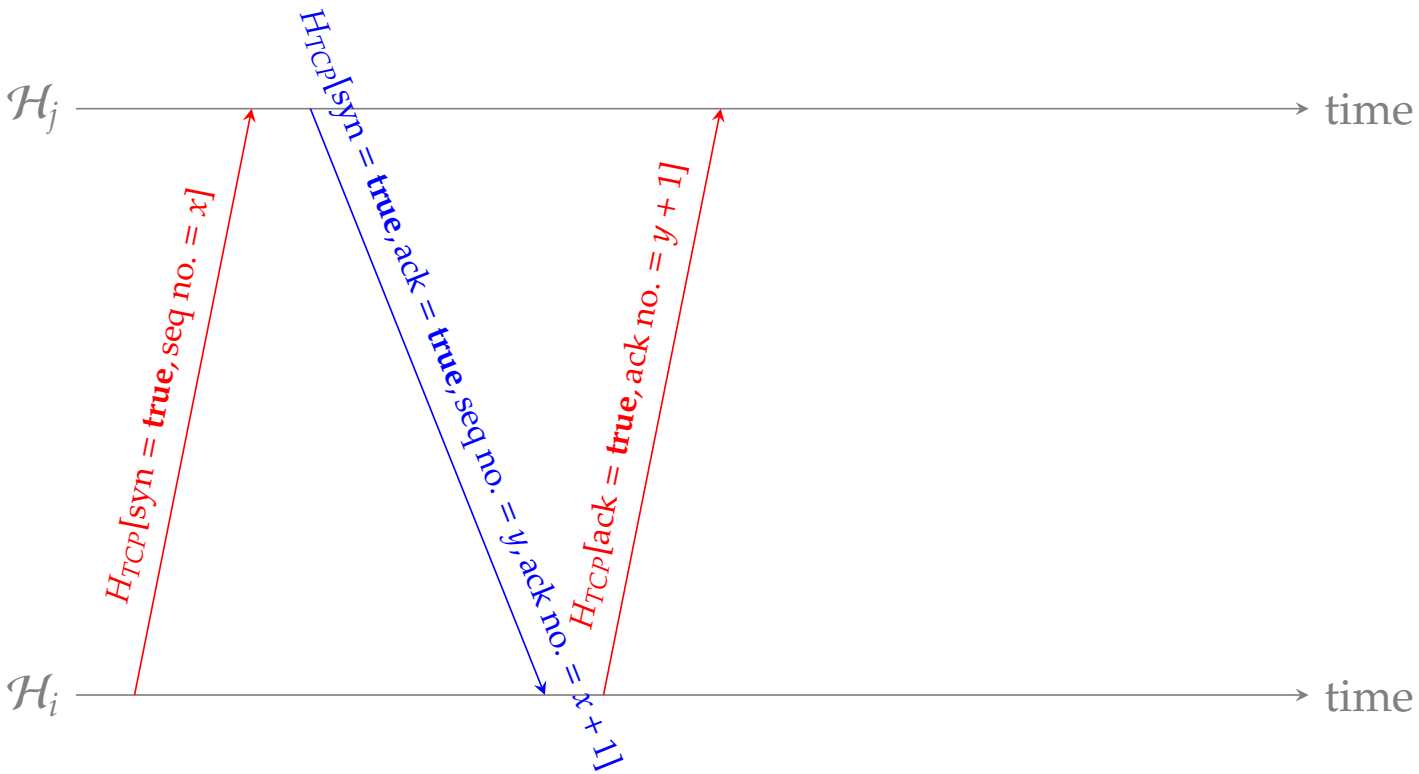
TCP (6)

- Example:
  1. connection establishment,



TCP (6)

- Example:
  1. connection establishment,

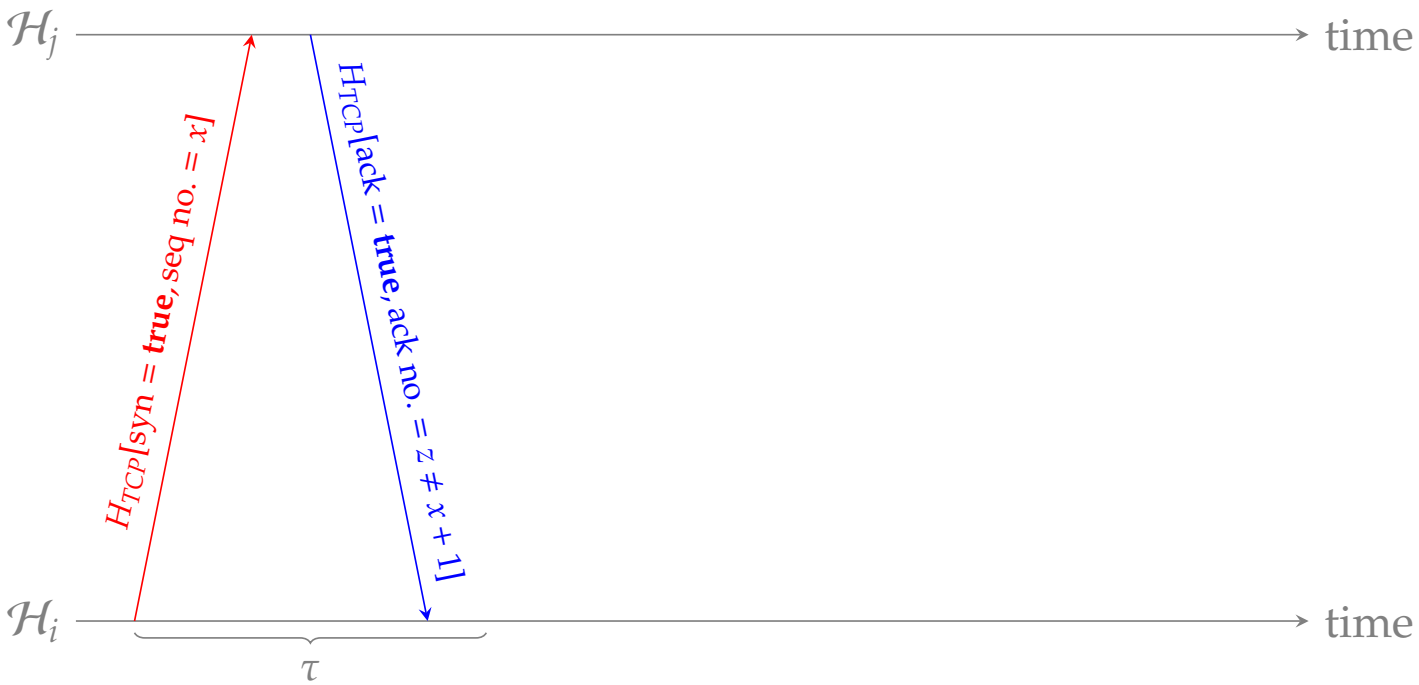


TCP (6)

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

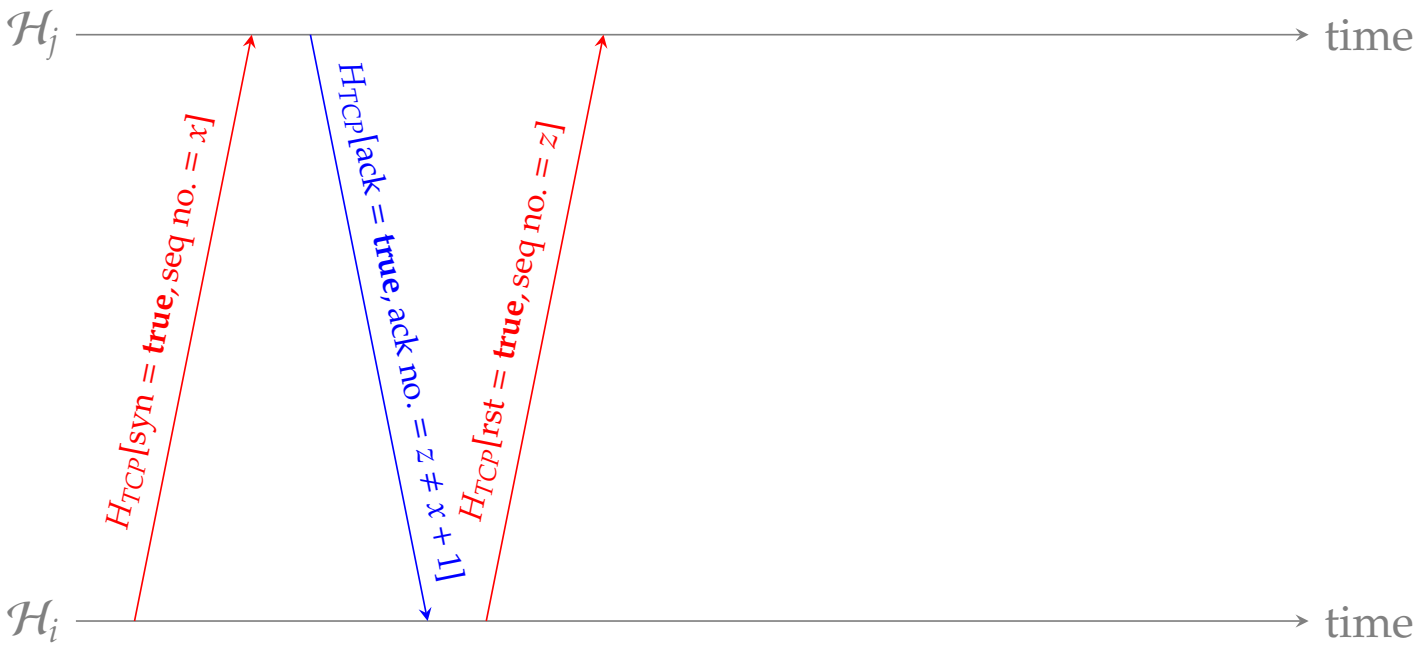
TCP (6)

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



TCP (6)

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

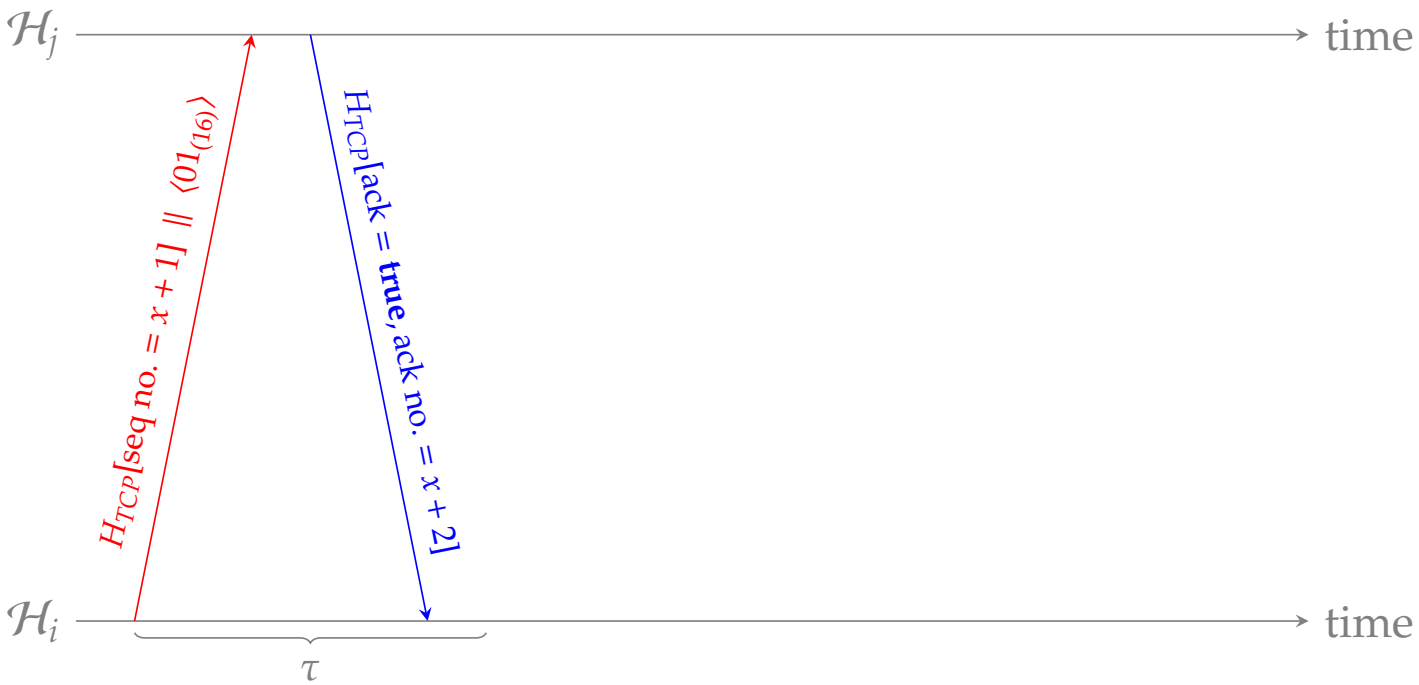


TCP (6)

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

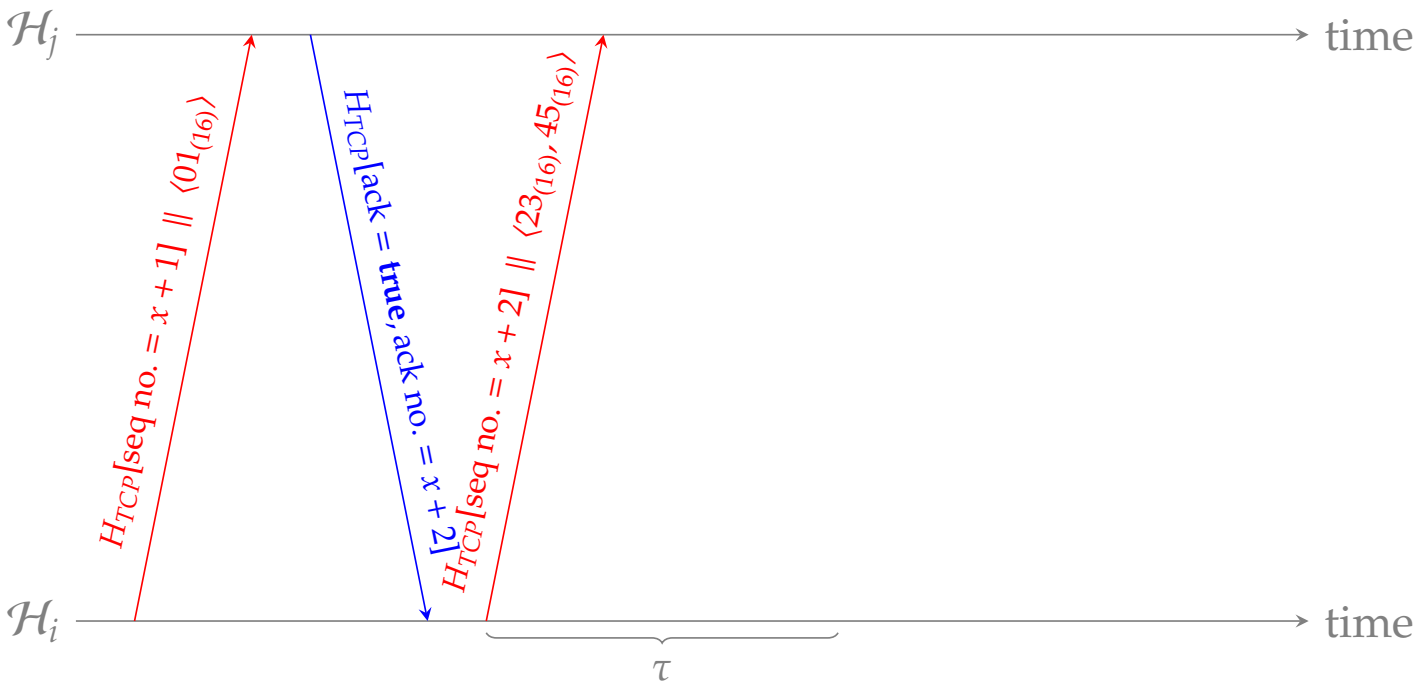
TCP (6)

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



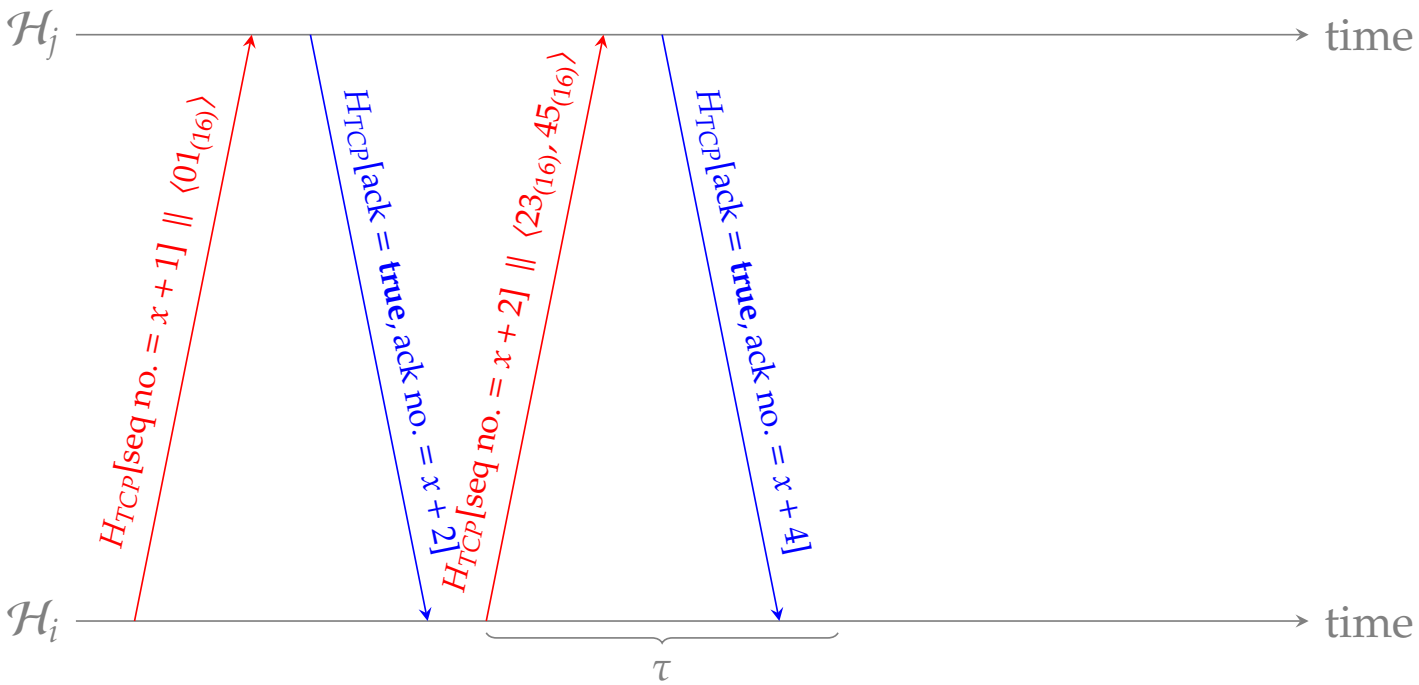
TCP (6)

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



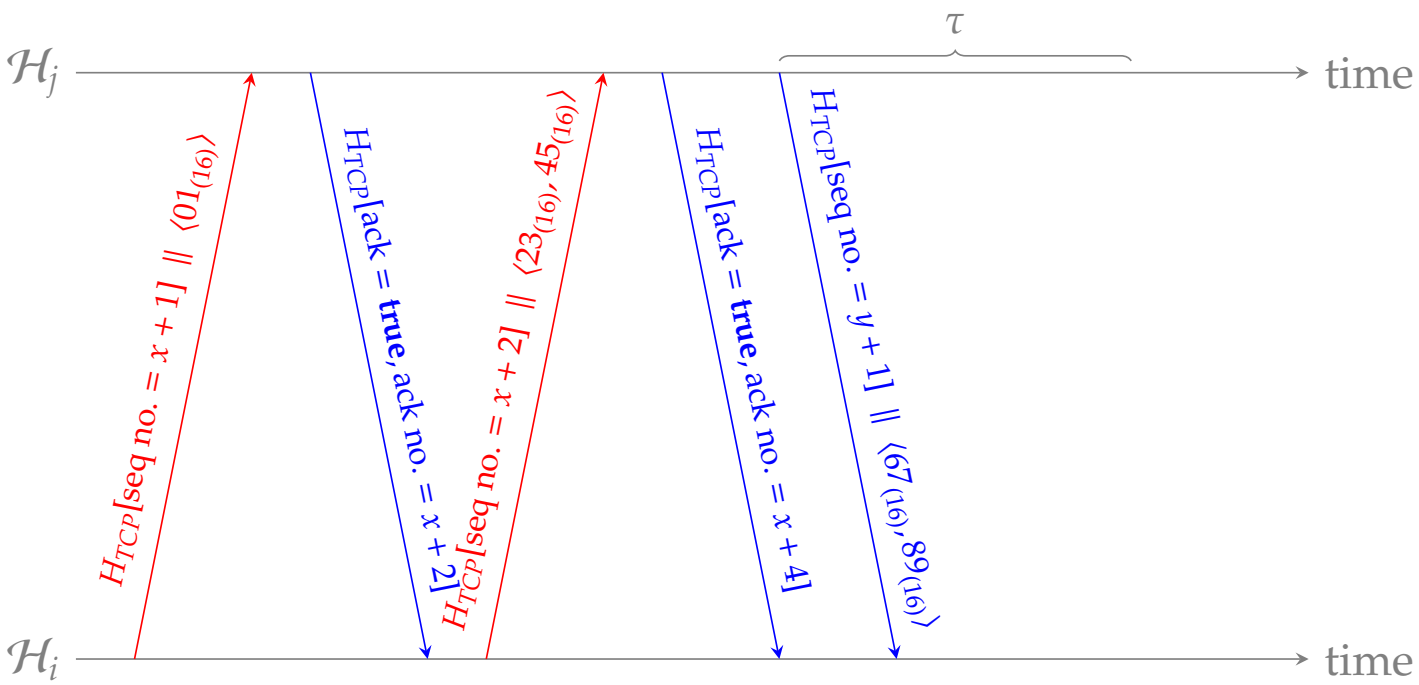
TCP (6)

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



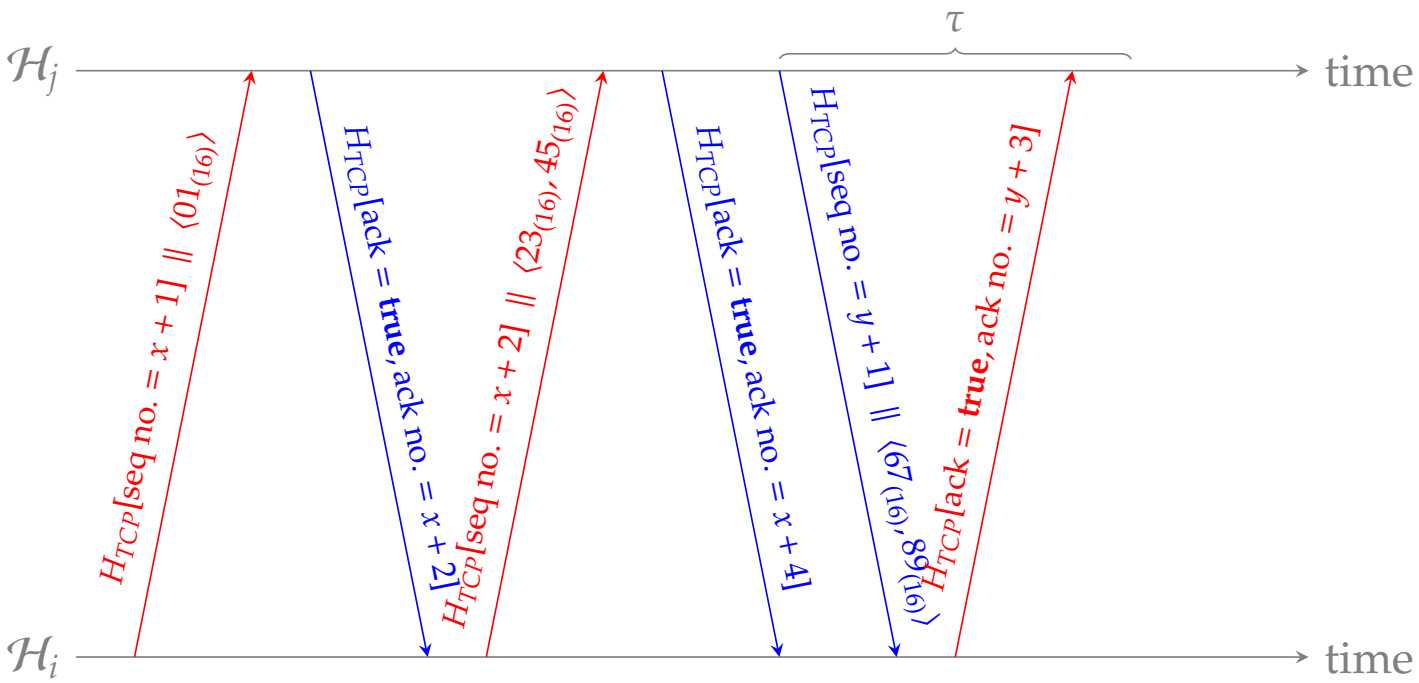
TCP (6)

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



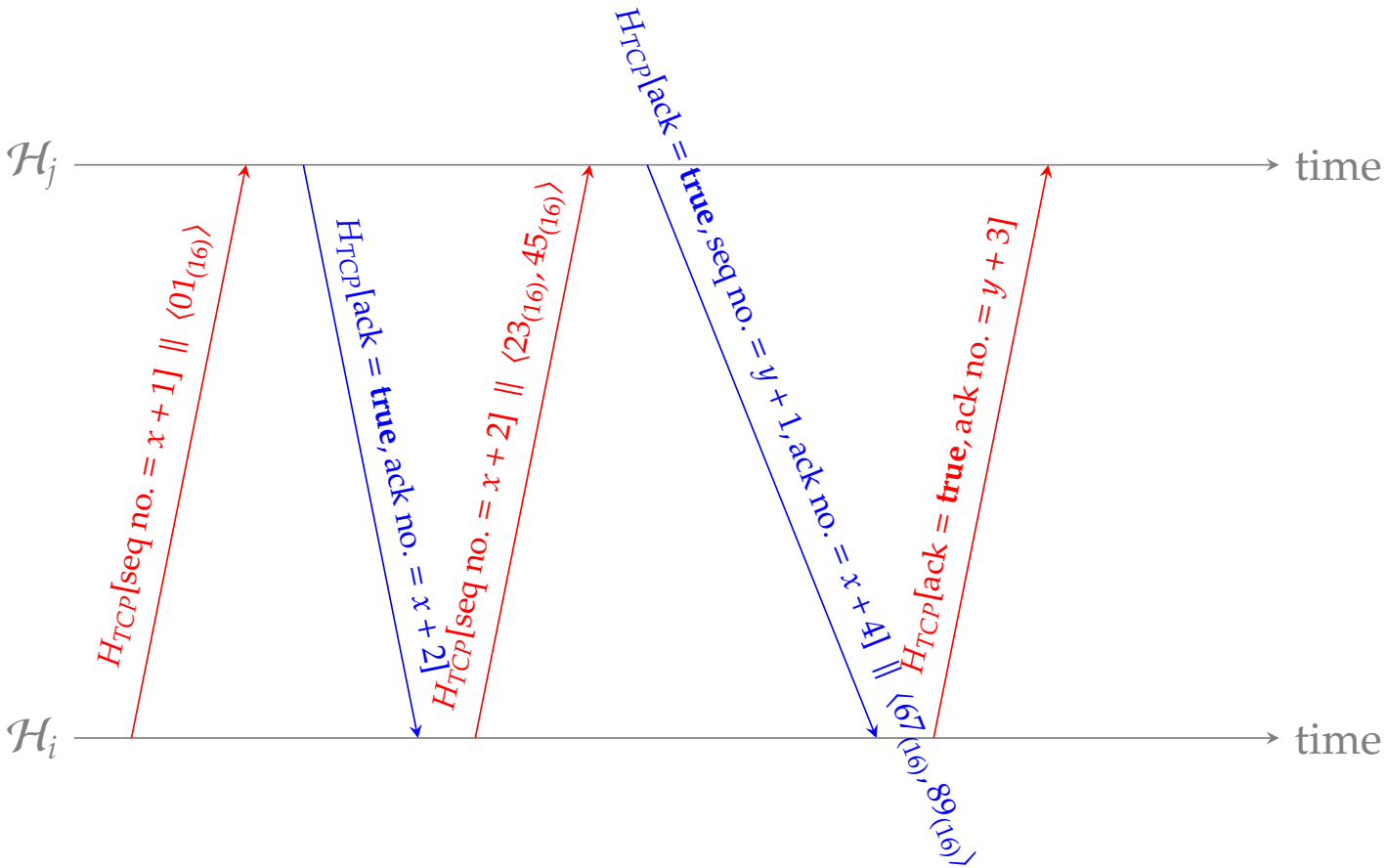
TCP (6)

- 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

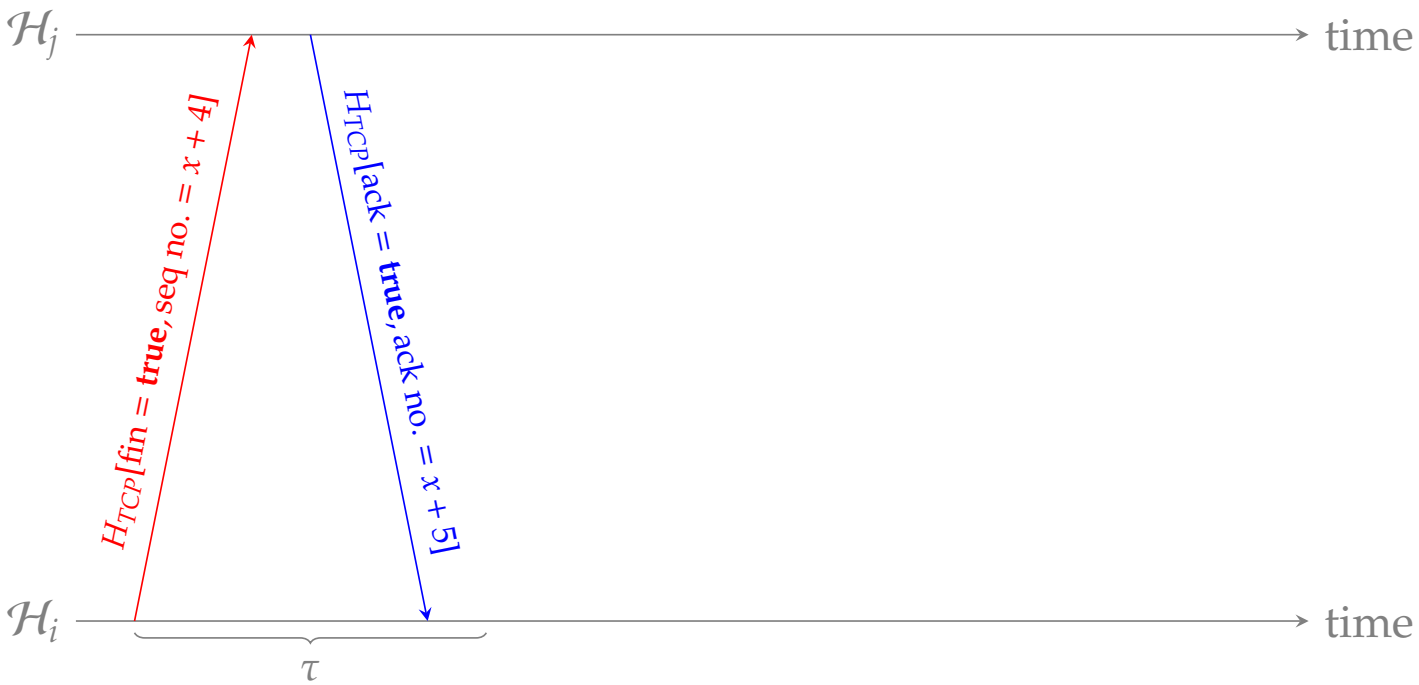


TCP (6)

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

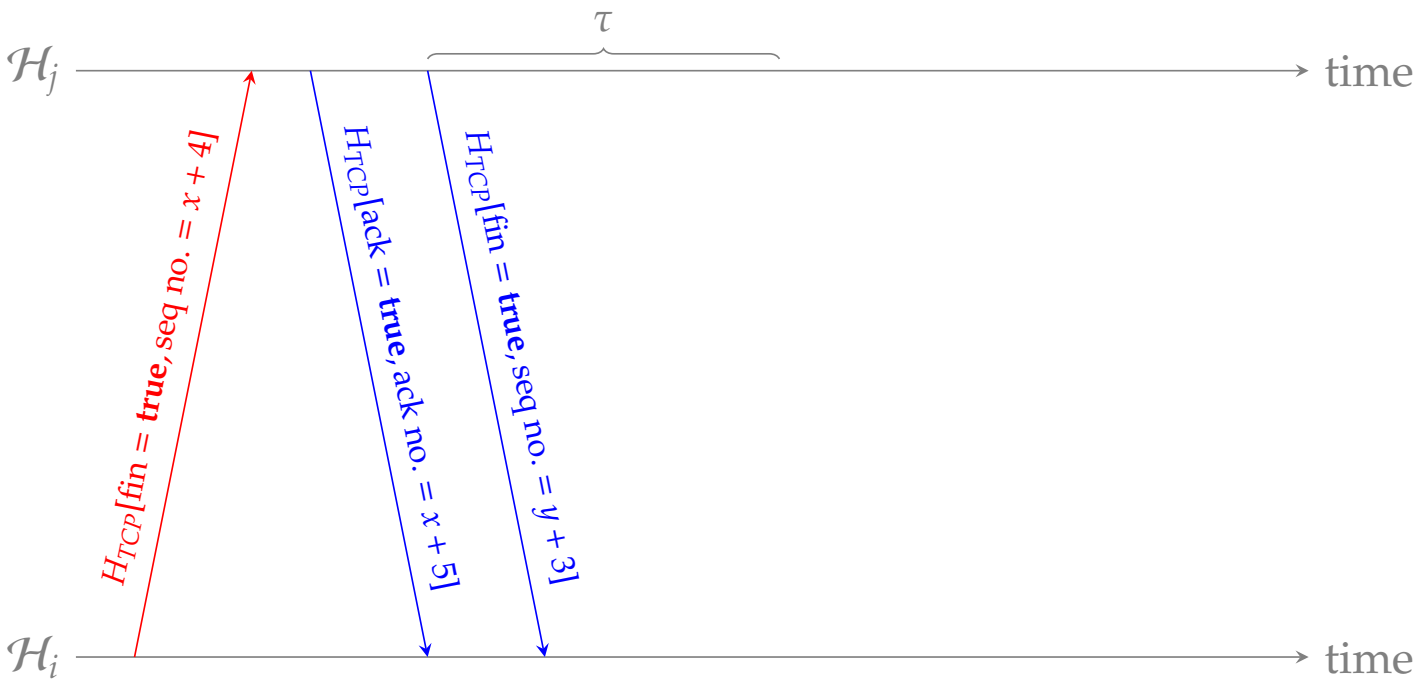
TCP (6)

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



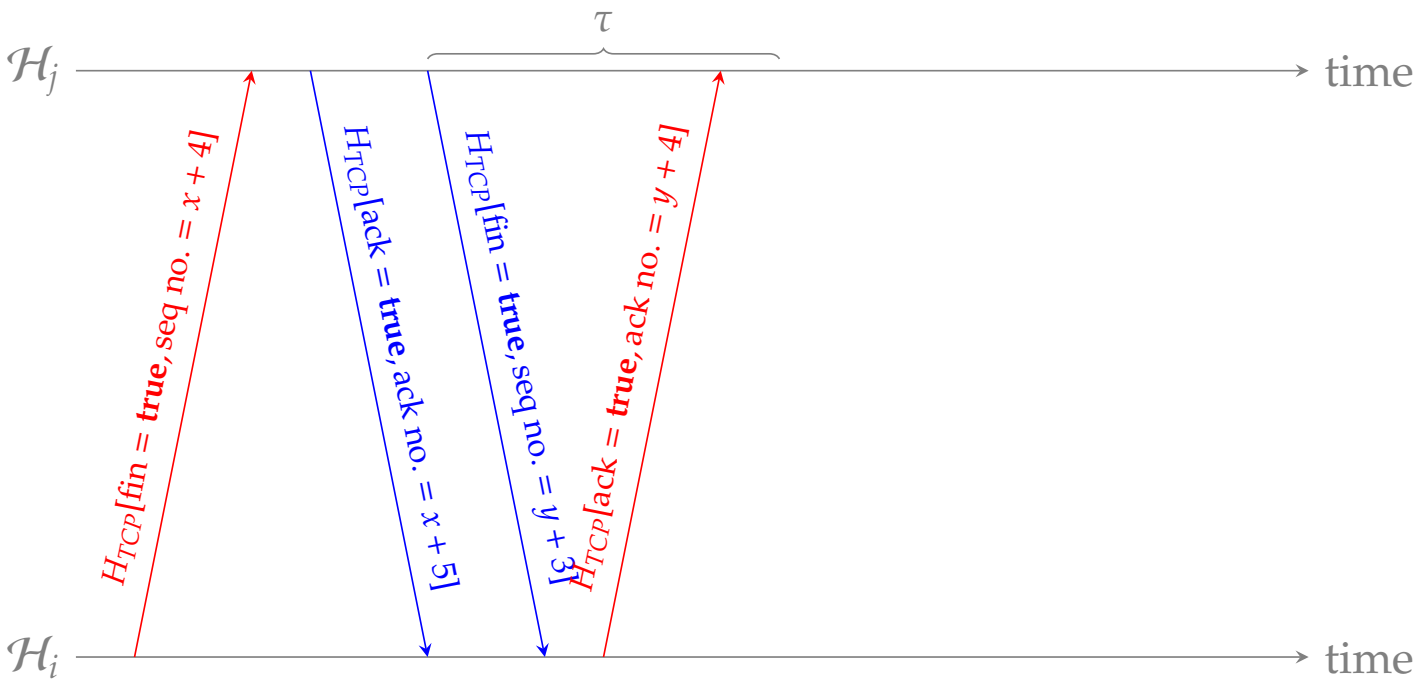
TCP (6)

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



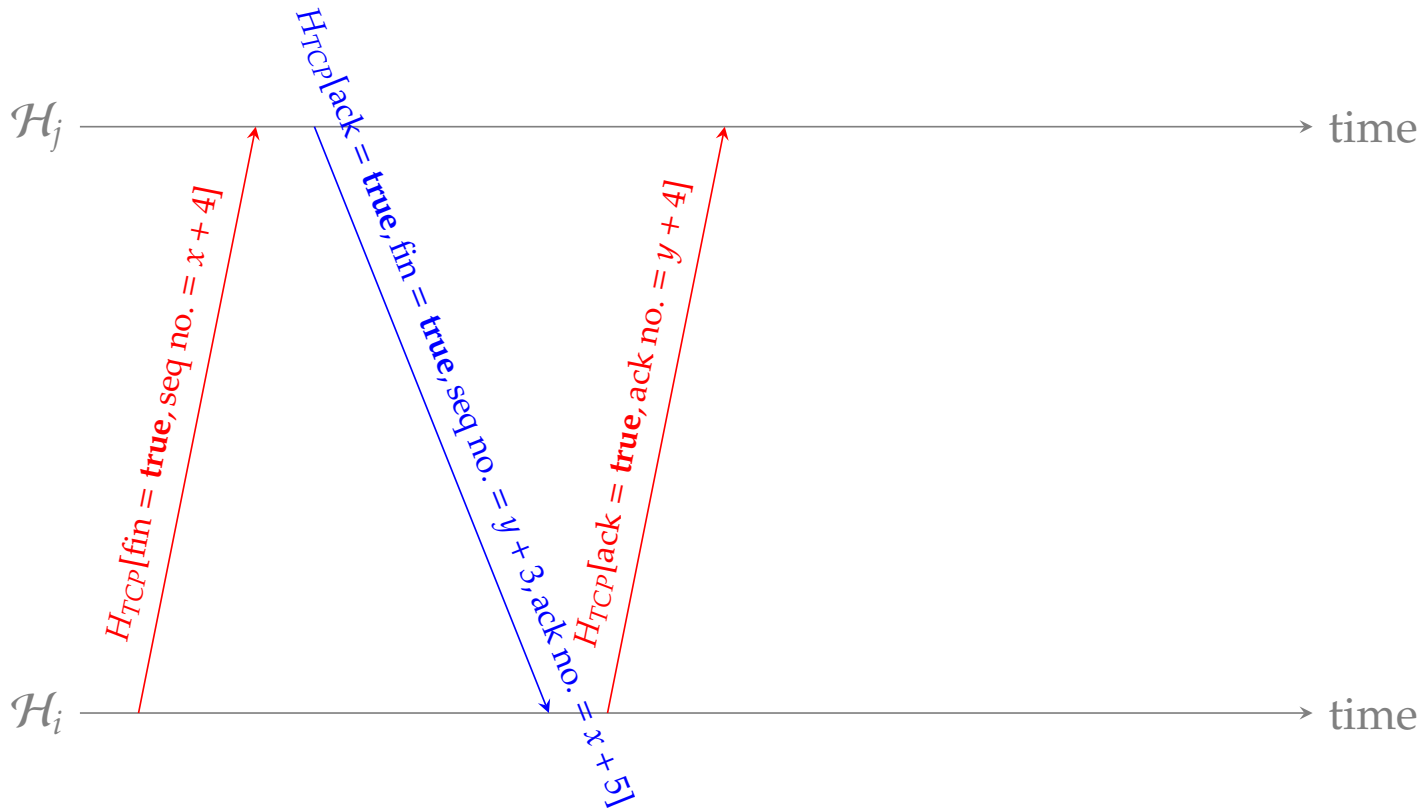
TCP (6)

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

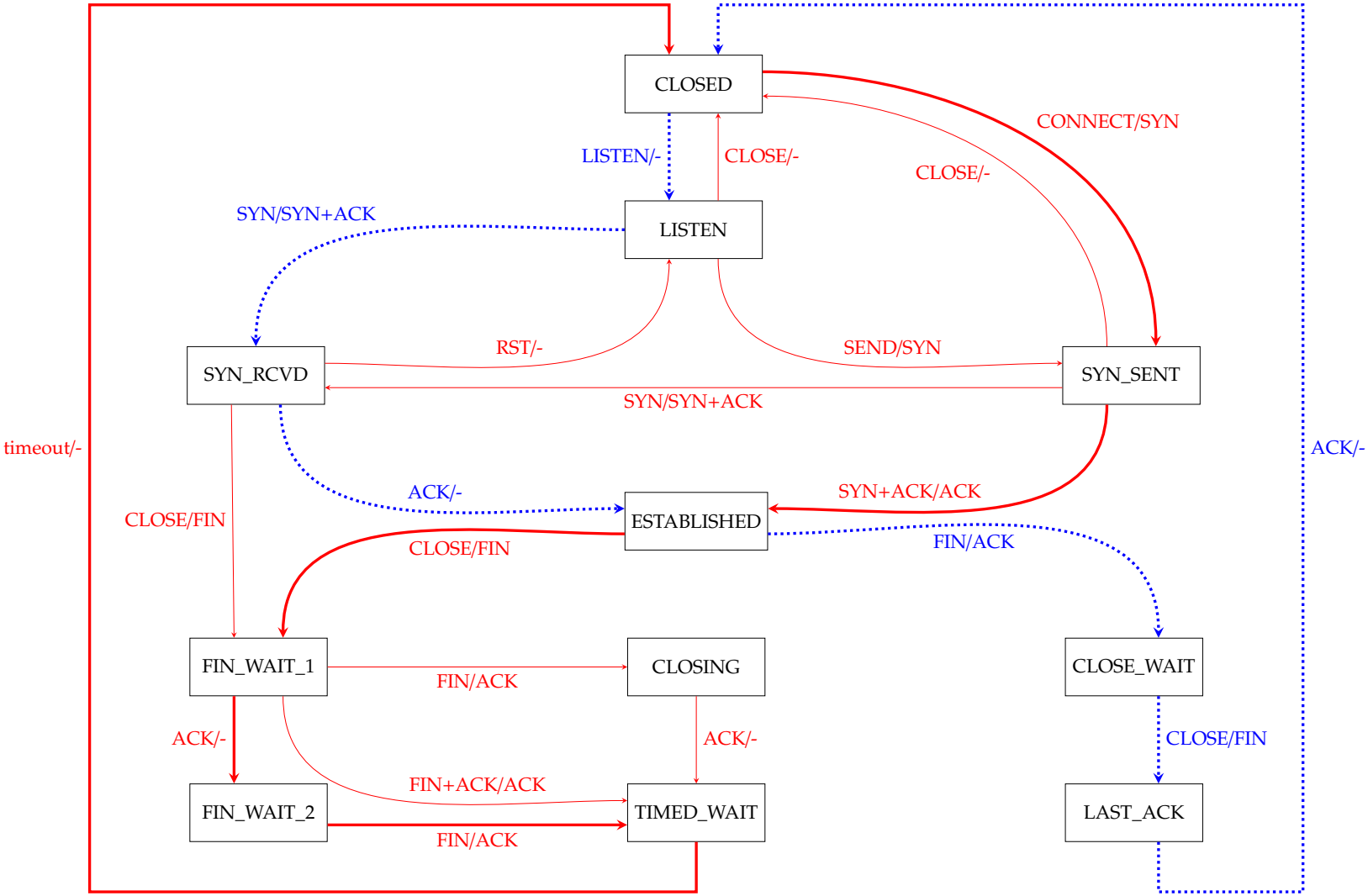


TCP (6)

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



Algorithm


CLOSED
LISTEN/-
CLOSE/-
CONNECT/SYN
CLOSE/-
SYN/SYN+ACK
LISTEN
RST/-
SEND/SYN
SYN_RCVD
SYN_SENT
SYN/SYN+ACK
timeout/-
ACK/-
ACK/-
SYN+ACK/ACK
CLOSE/FIN
ESTABLISHED
CLOSE/FIN
FIN/ACK
FIN_WAIT_1
CLOSING
CLOSE_WAIT
FIN/ACK
ACK/-
ACK/-
CLOSE/FIN
FIN+ACK/ACK
FIN_WAIT_2
TIMED_WAIT
LAST_ACK
FIN/ACK

TCP (8)

- ... so far so good, *but*, for example
  - Question: can we improve the efficiency of stop-and-wait?
  - Question: are other improvements/optimisations possible?
  - Question: how do we select τ, the time-out threshold?

TCP (8)

- ... so far so good, *but*, for example
  - Question: can we improve the efficiency of stop-and-wait?
  - Answer: yes, via **sliding-window** based on either
    - **go-back-n**, or
    - **selective-repeat**

    which *also* offer a neat solution for flow control.
  - Question: are other improvements/optimisations possible?

- Question: how do we select τ, the time-out threshold?

TCP (8)

- ... so far so good, *but*, for example
  - Question: can we improve the efficiency of stop-and-wait?
  - Answer: yes, via **sliding-window** based on either
    - **go-back-n**, or
    - **selective-repeat**

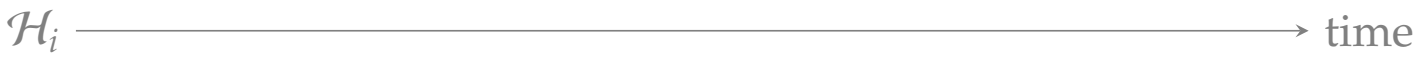
    which *also* offer a neat solution for flow control.
  - Question: are other improvements/optimisations possible?
  - Answer: yes, lots, e.g.,
    - **cumulative ACKs**,
    - **selective ACKs**,
    - **delayed ACKs**,
    - ...
  - Question: how do we select τ, the time-out threshold?

TCP (8)

- ... so far so good, *but*, for example
  - Question: can we improve the efficiency of stop-and-wait?
  - Answer: yes, via **sliding-window** based on either
    - **go-back-n**, or
    - **selective-repeat**

    which *also* offer a neat solution for flow control.
  - Question: are other improvements/optimisations possible?
  - Answer: yes, lots, e.g.,
    - **cumulative ACKs**,
    - **selective ACKs**,
    - **delayed ACKs**,
    - ...
  - Question: how do we select τ, the time-out threshold?
  - Answer: using a moving average of measured RTT.

TCP Improvements (1)
Sliding-window ARQ



- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

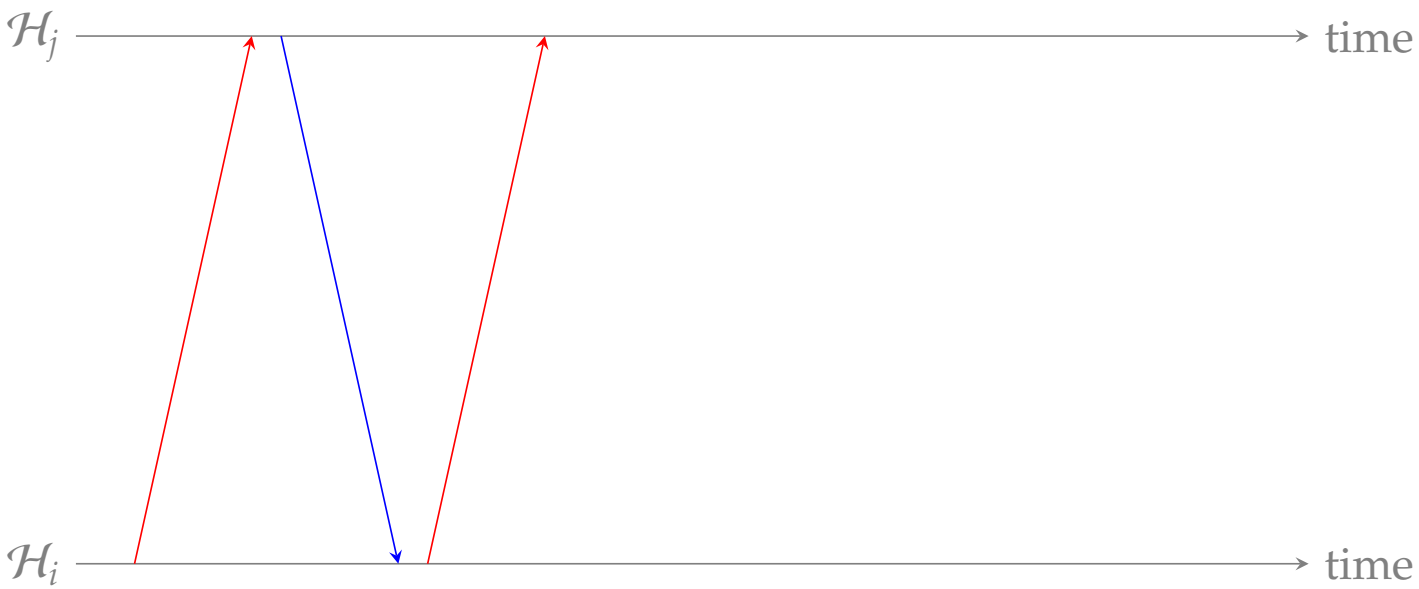
TCP Improvements (1)
Sliding-window ARQ

- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP Improvements (1)
Sliding-window ARQ

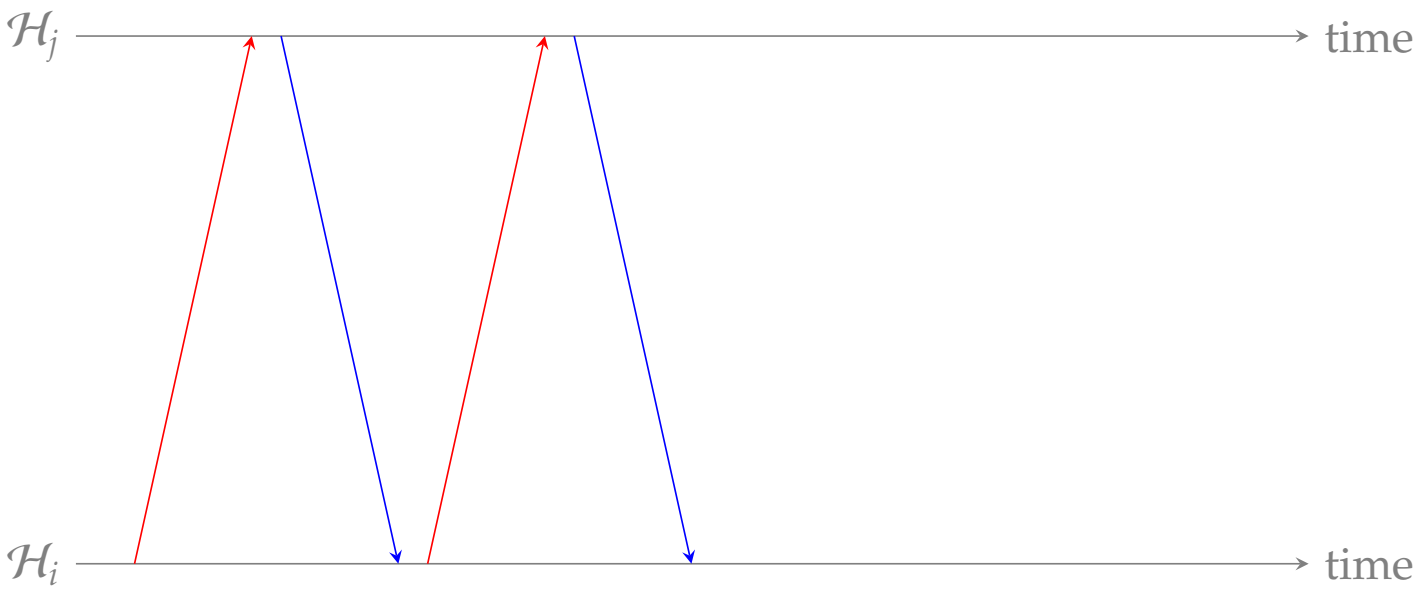
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP Improvements (1)
Sliding-window ARQ



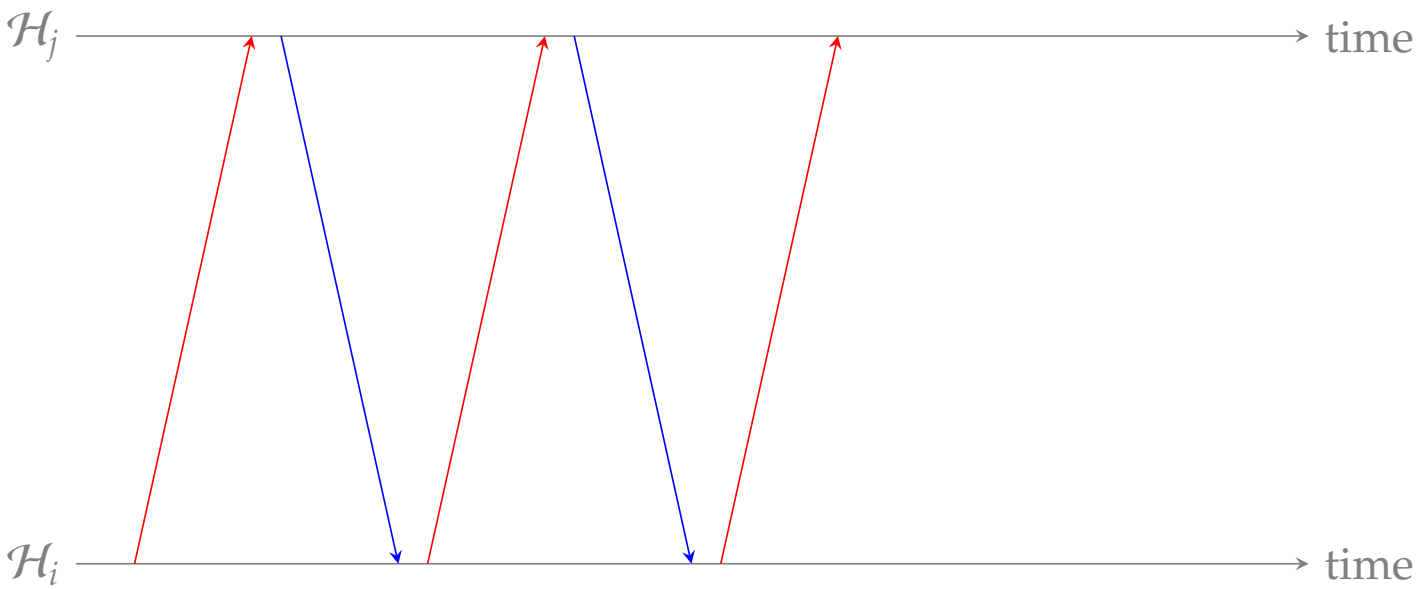
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP Improvements (1)
Sliding-window ARQ



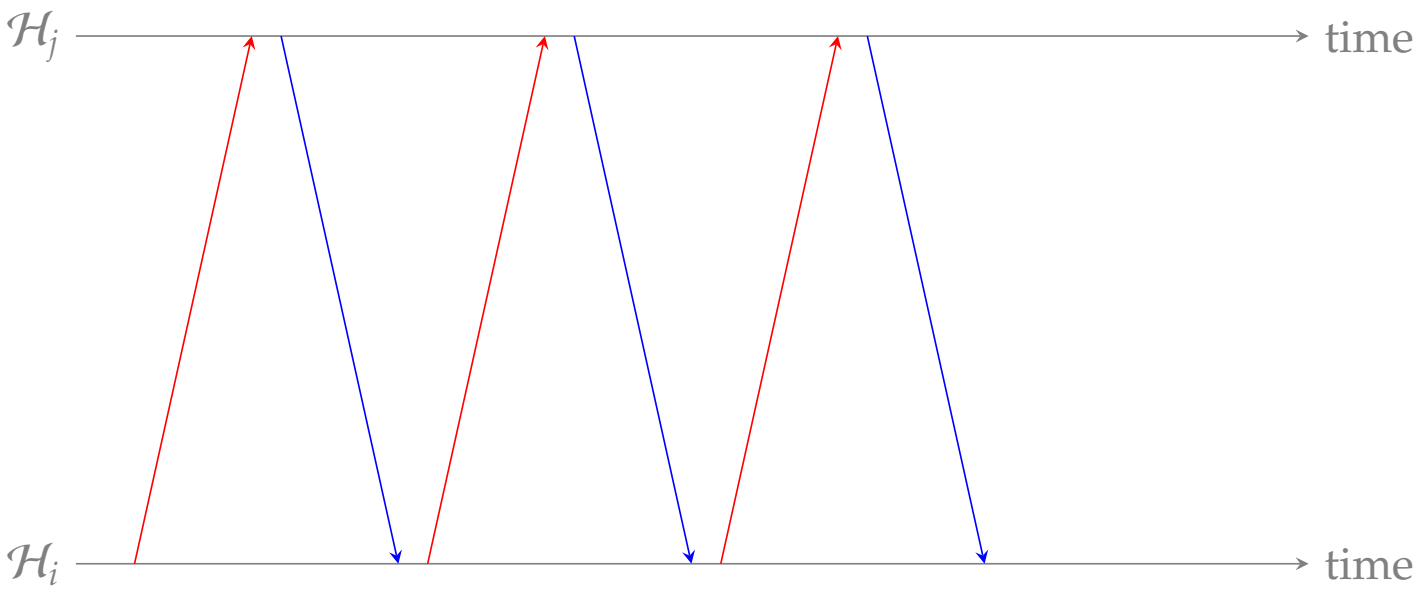
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP Improvements (1)
Sliding-window ARQ



- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP Improvements (1)
Sliding-window ARQ



- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.

TCP Improvements (1)
Sliding-window ARQ

- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ

- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

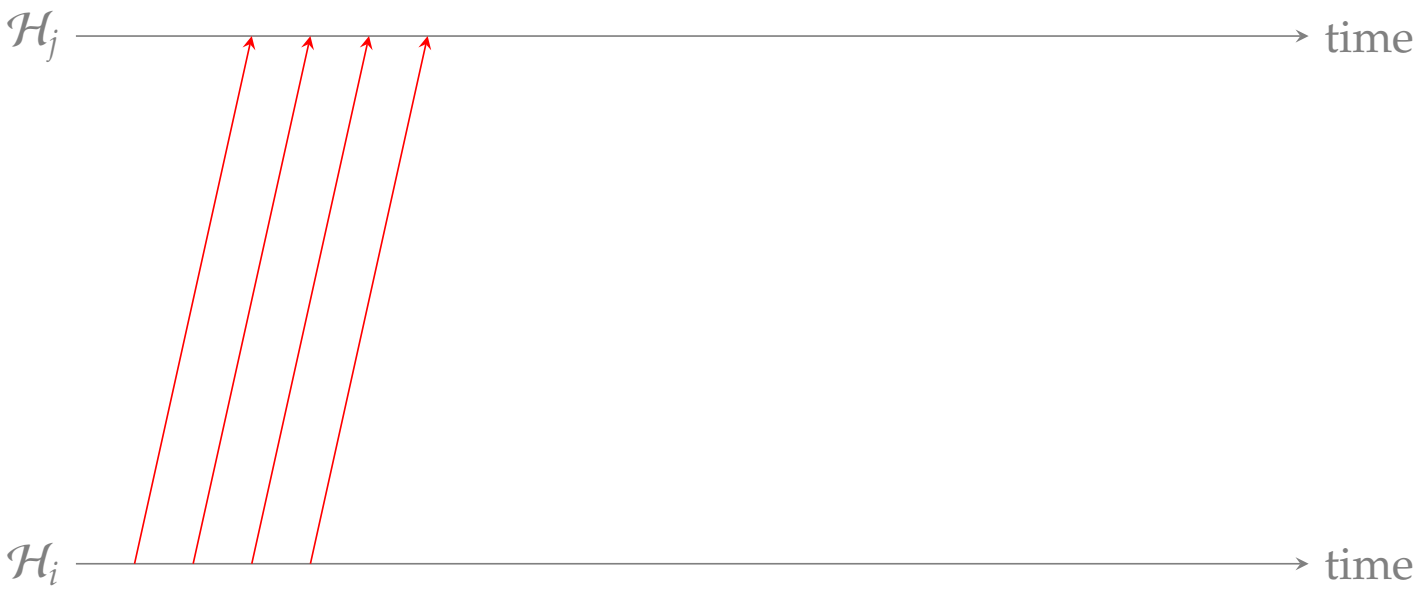
TCP Improvements (1)
Sliding-window ARQ

- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ

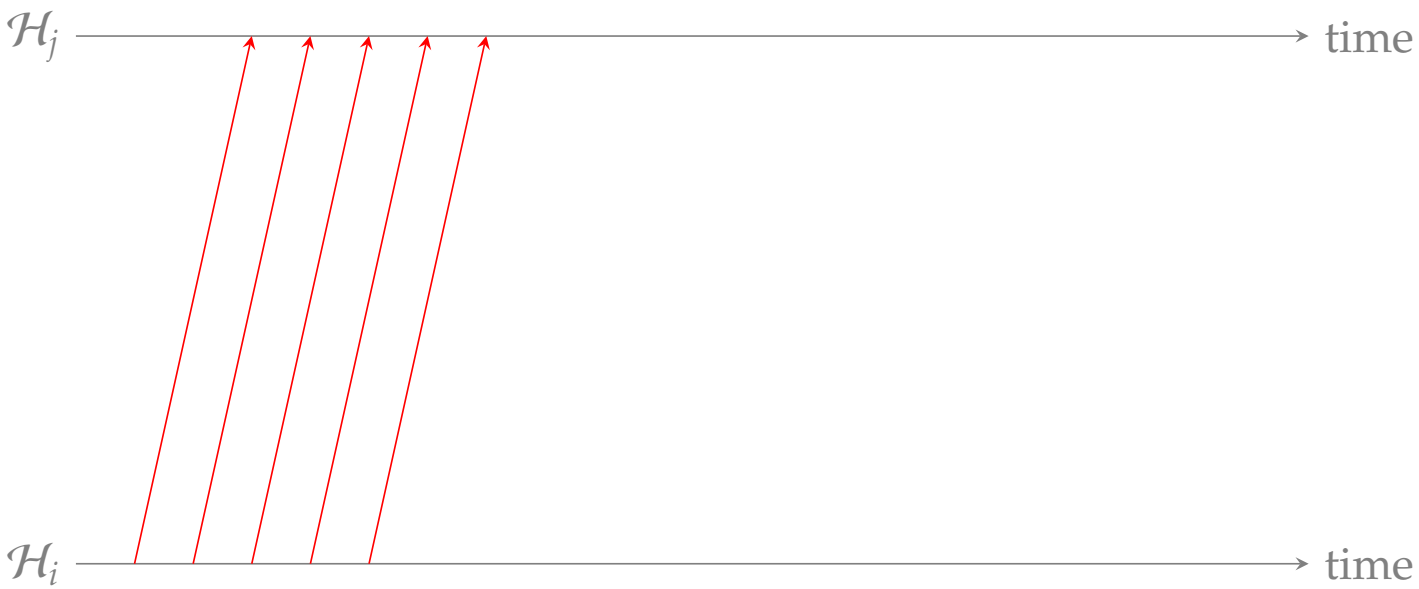
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



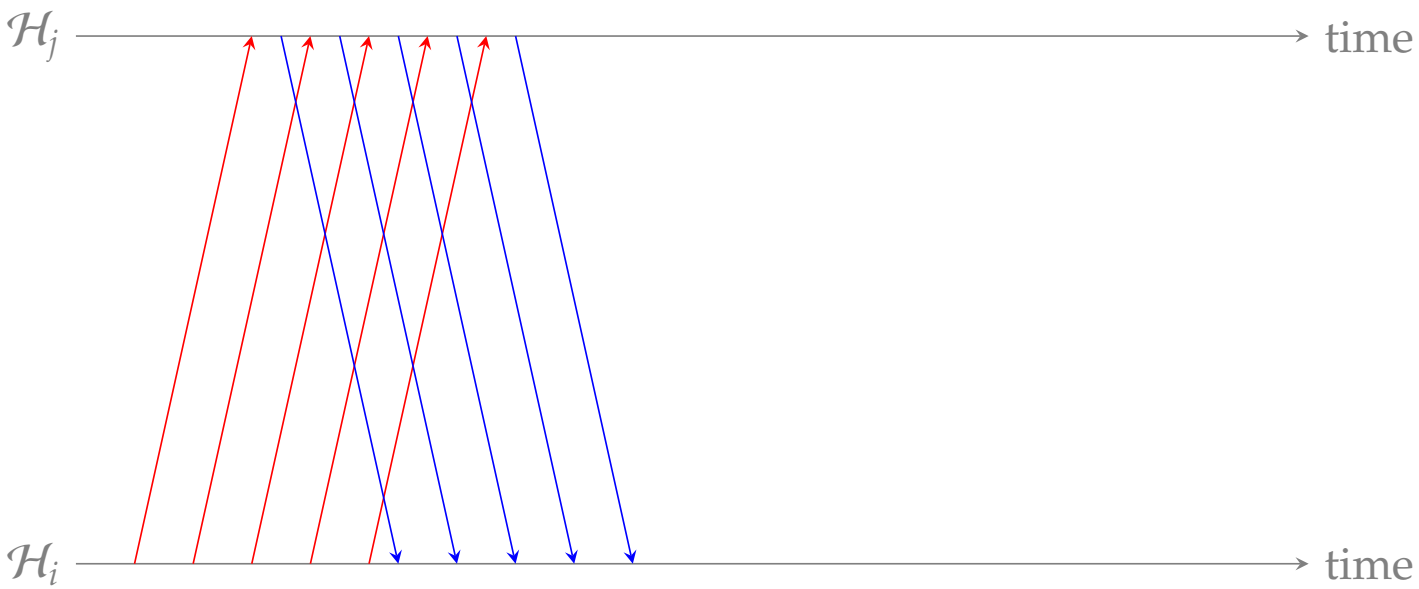
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



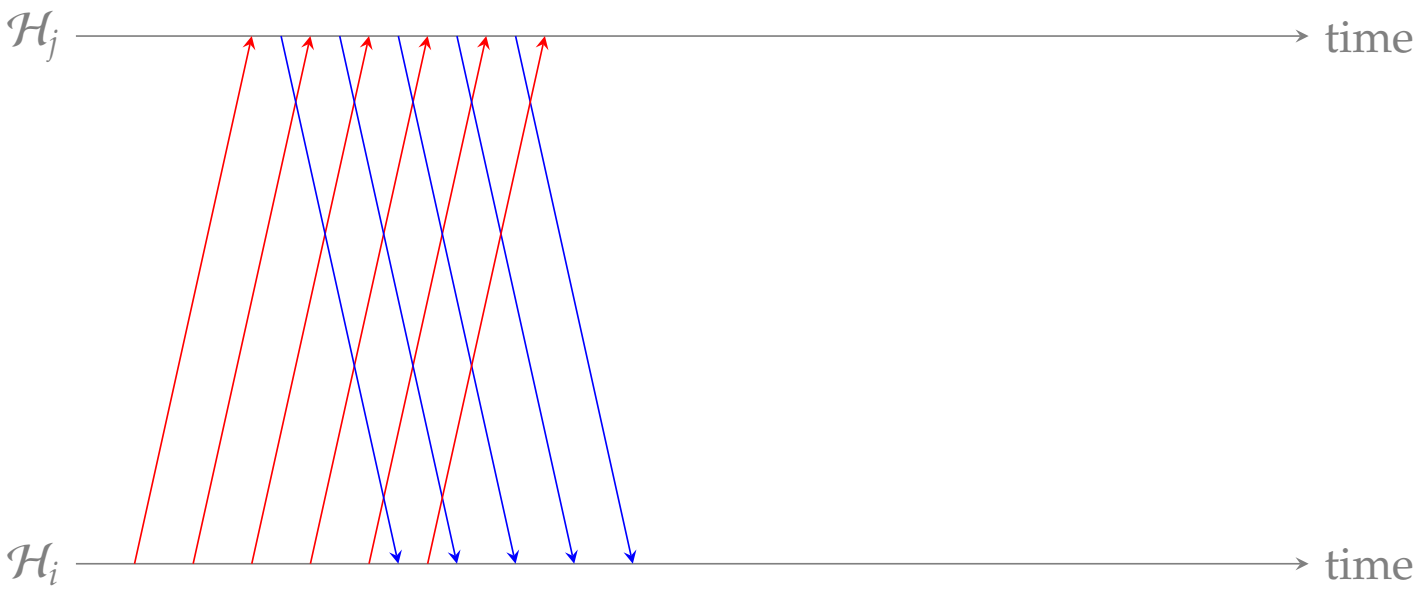
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



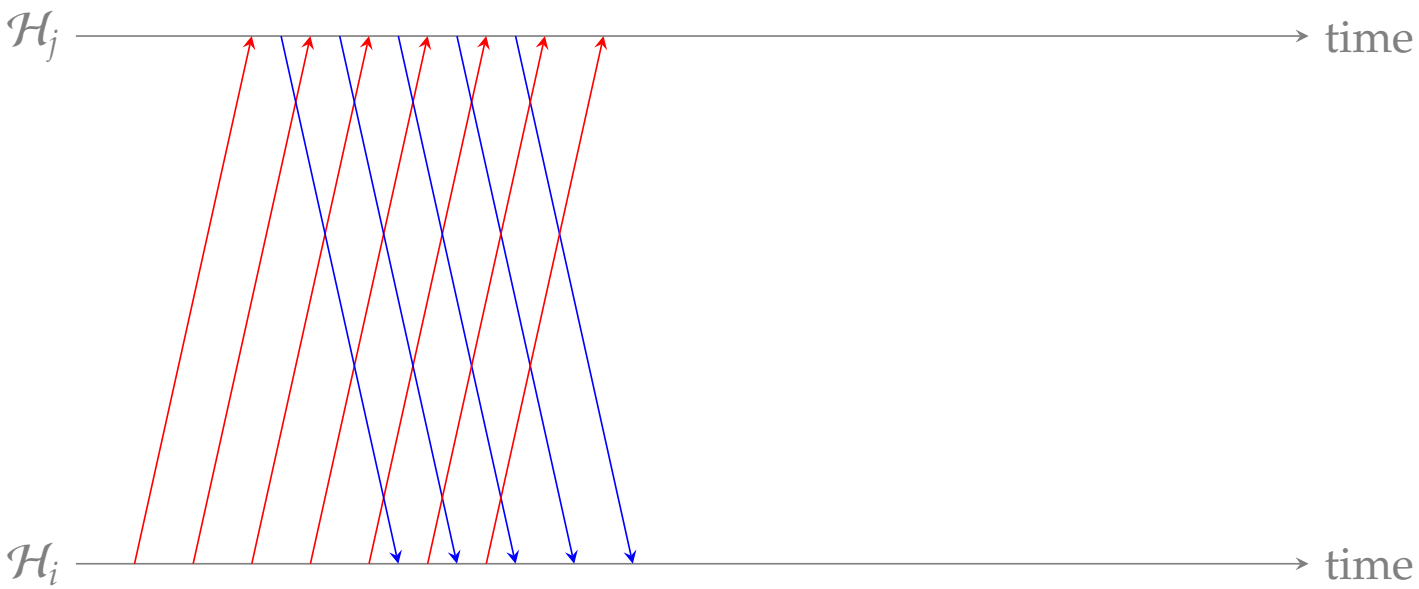
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



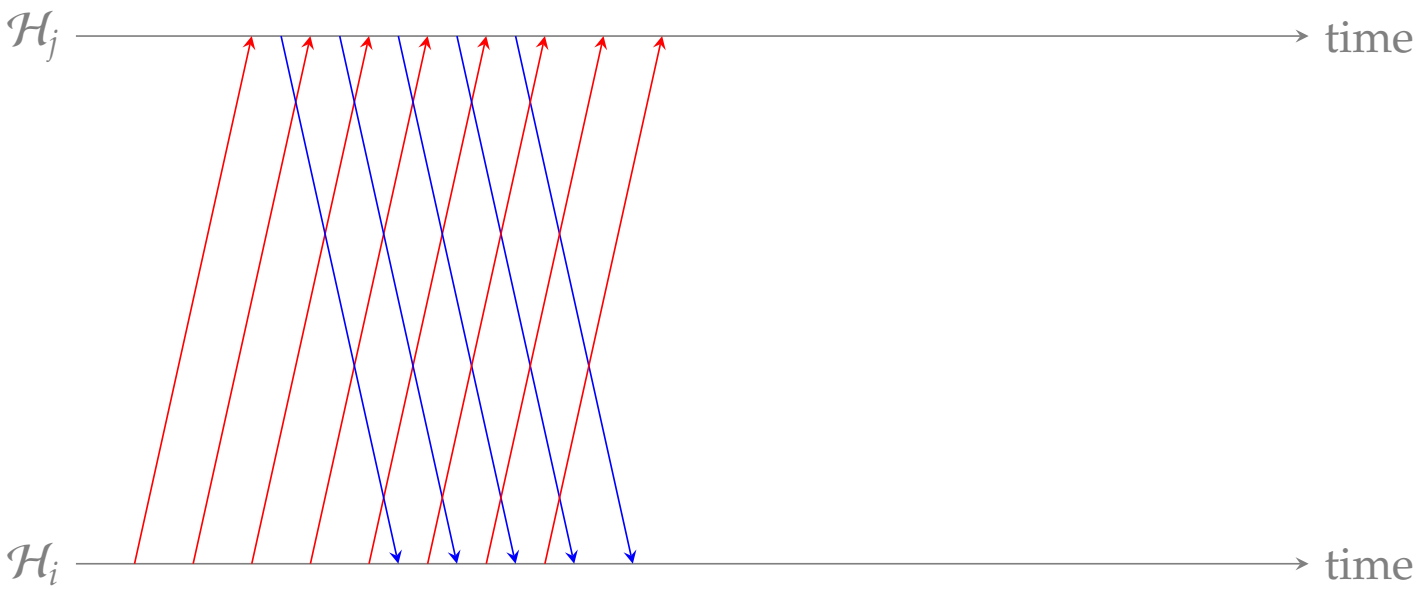
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



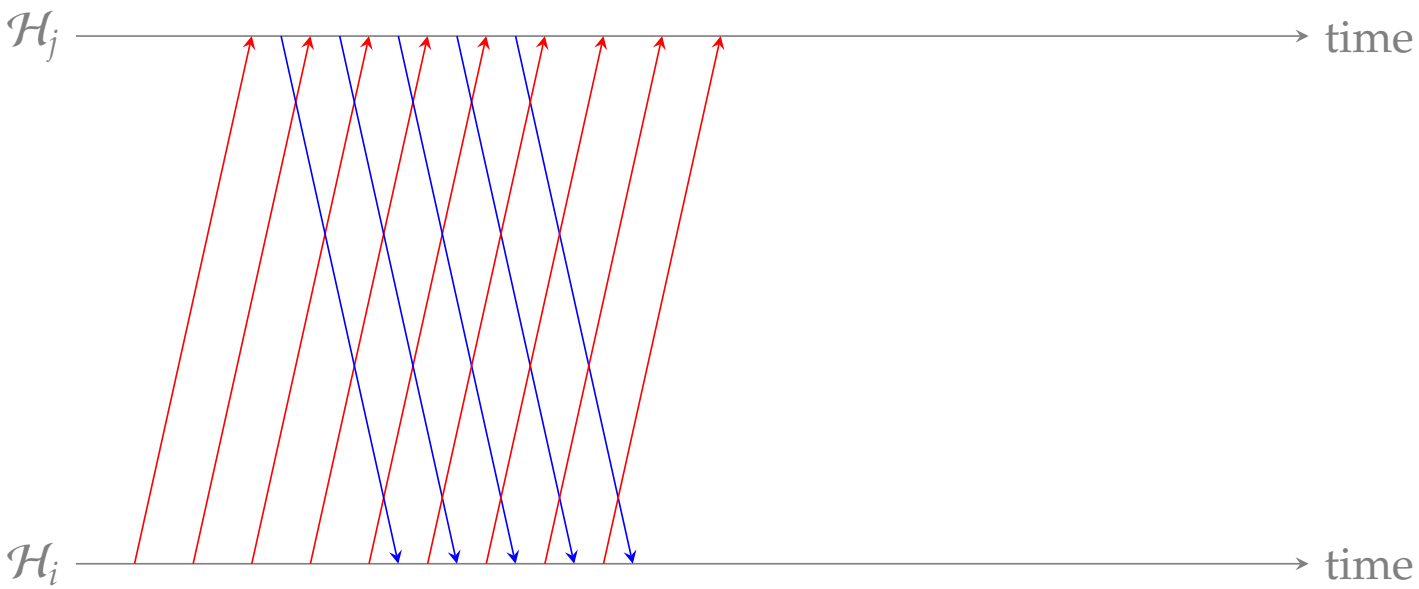
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



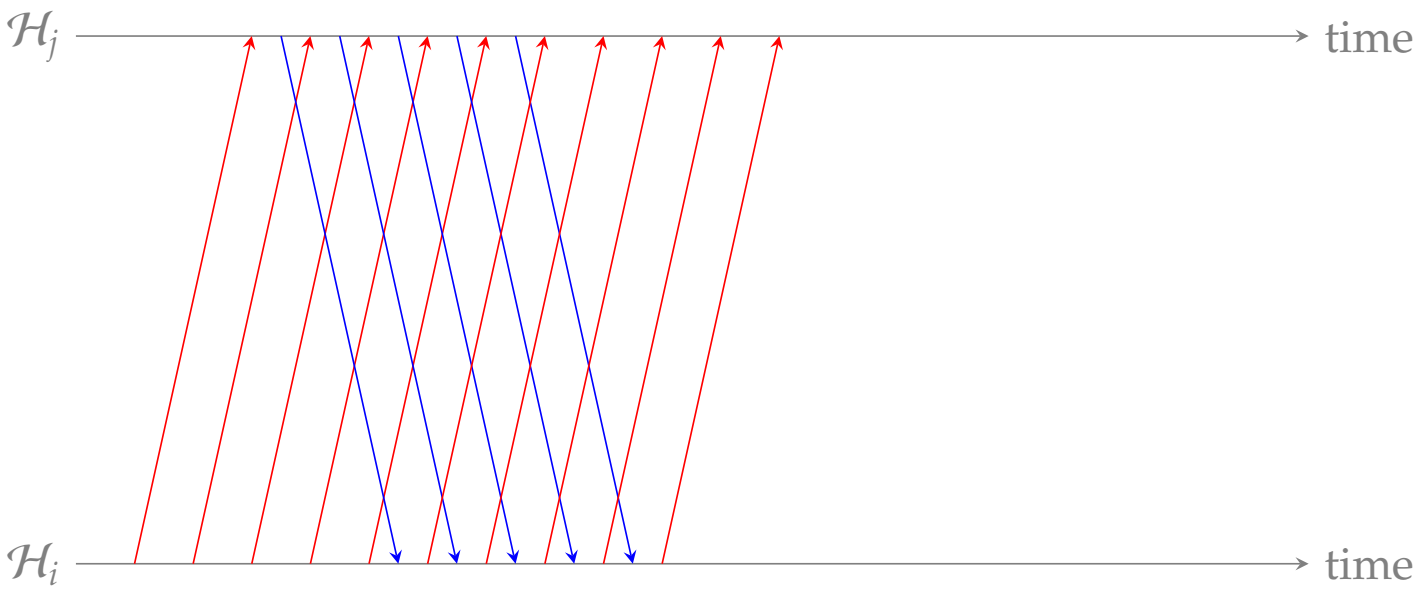
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



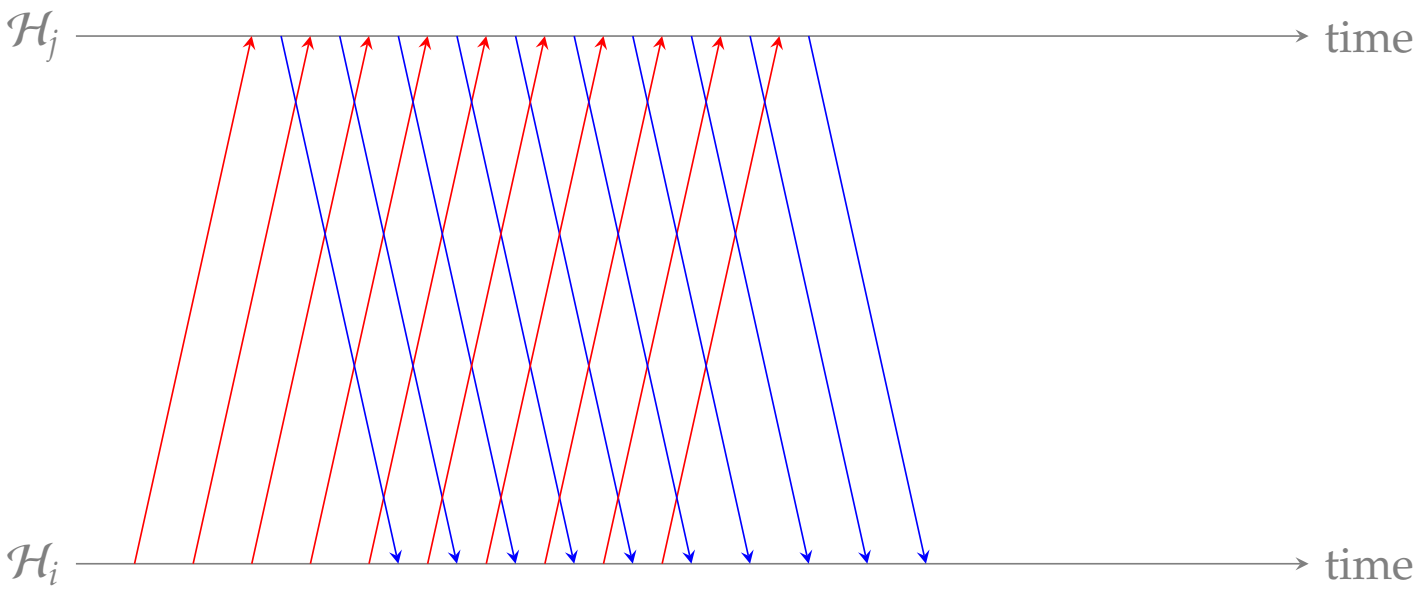
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



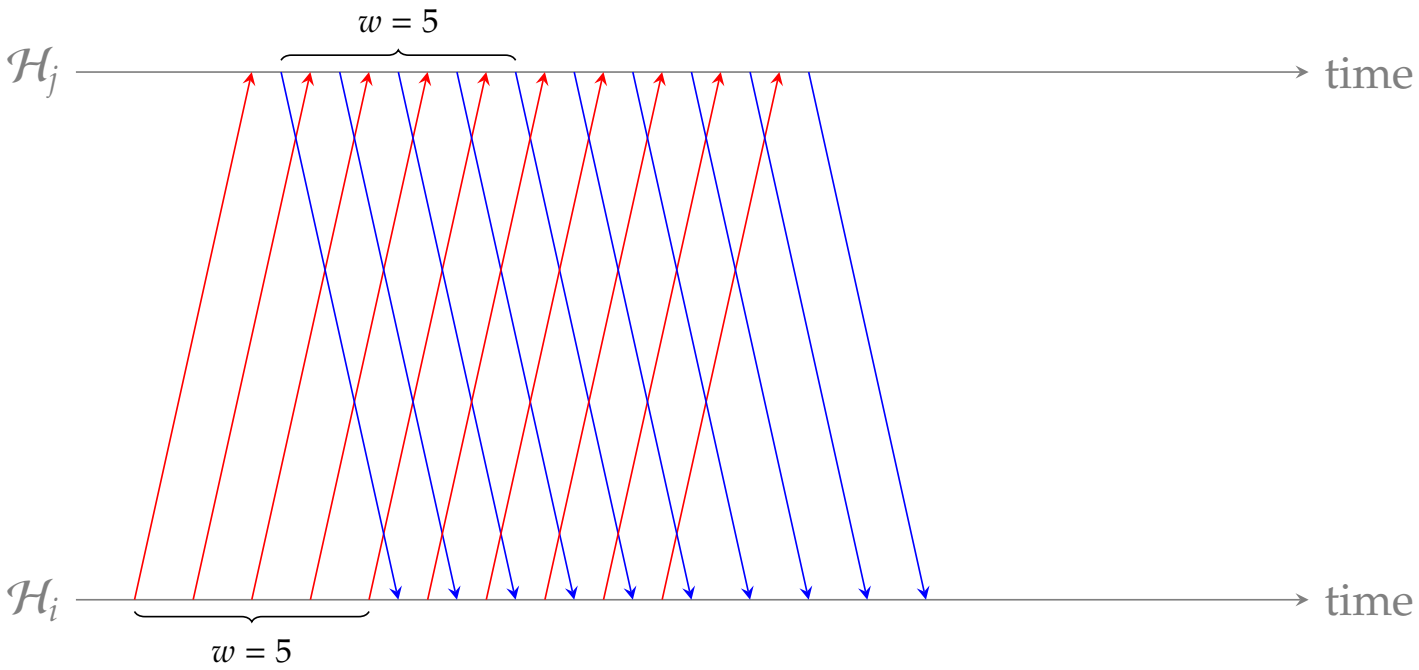
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



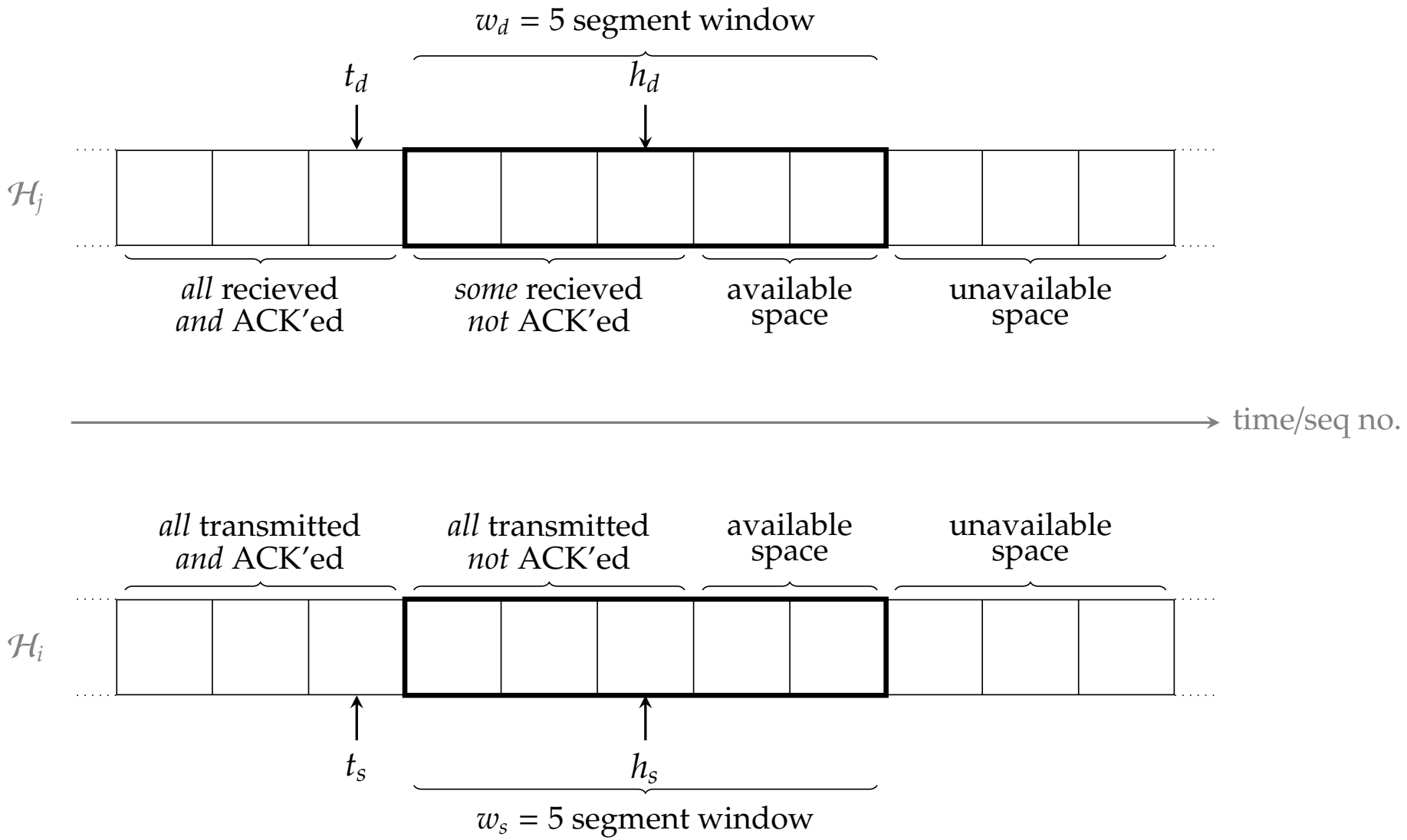
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (1)
Sliding-window ARQ



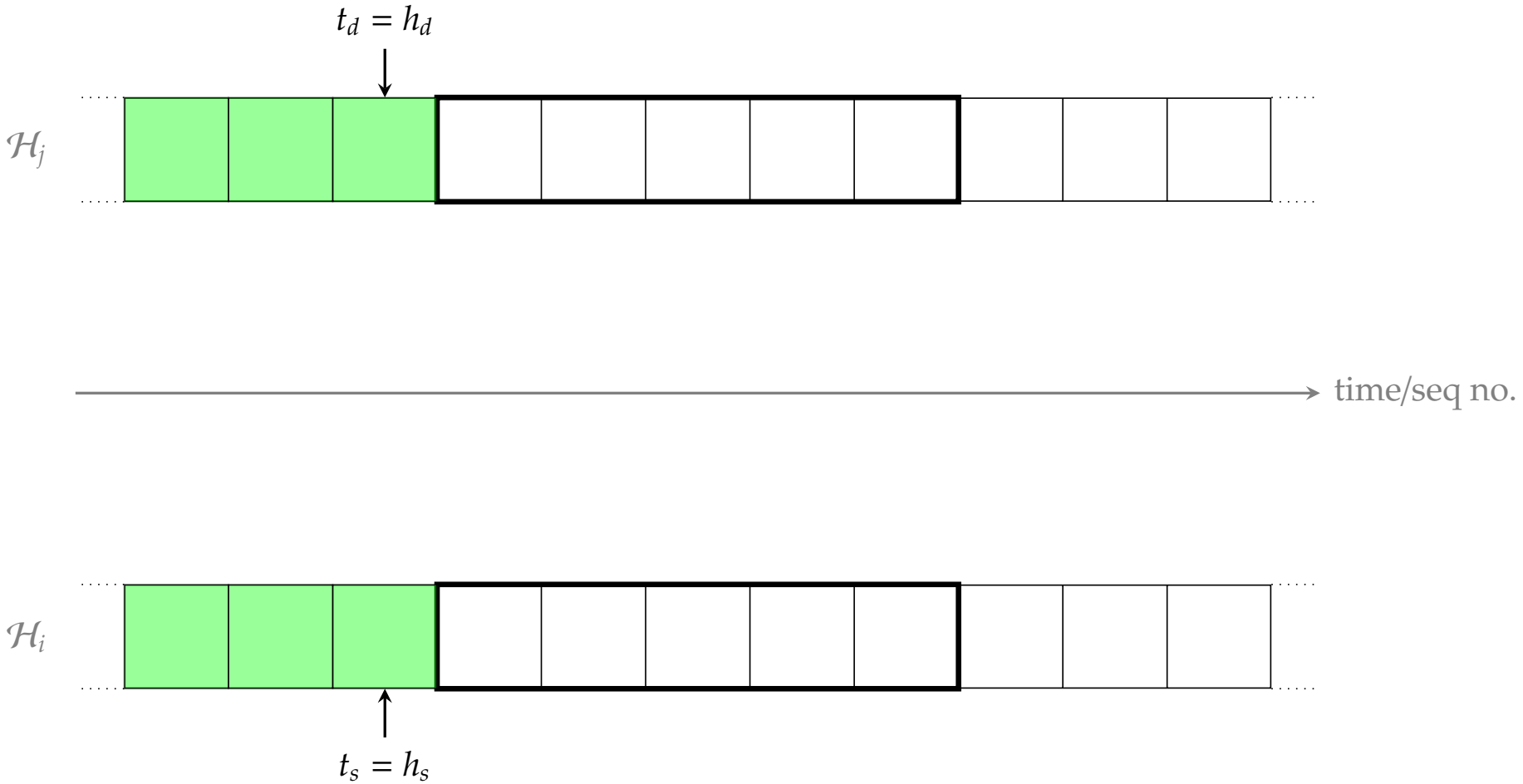
- Problem: stop-and-wait allows $w = 1$ un-ACK'ed segment
  - for a LAN this is fine, but
  - for an inter-network this is not fine, due to the high(er) bandwidth-latency product.
- Solution: generalise to $w > 1$ via **sliding-window**.

TCP Improvements (2)
Sliding-window ARQ

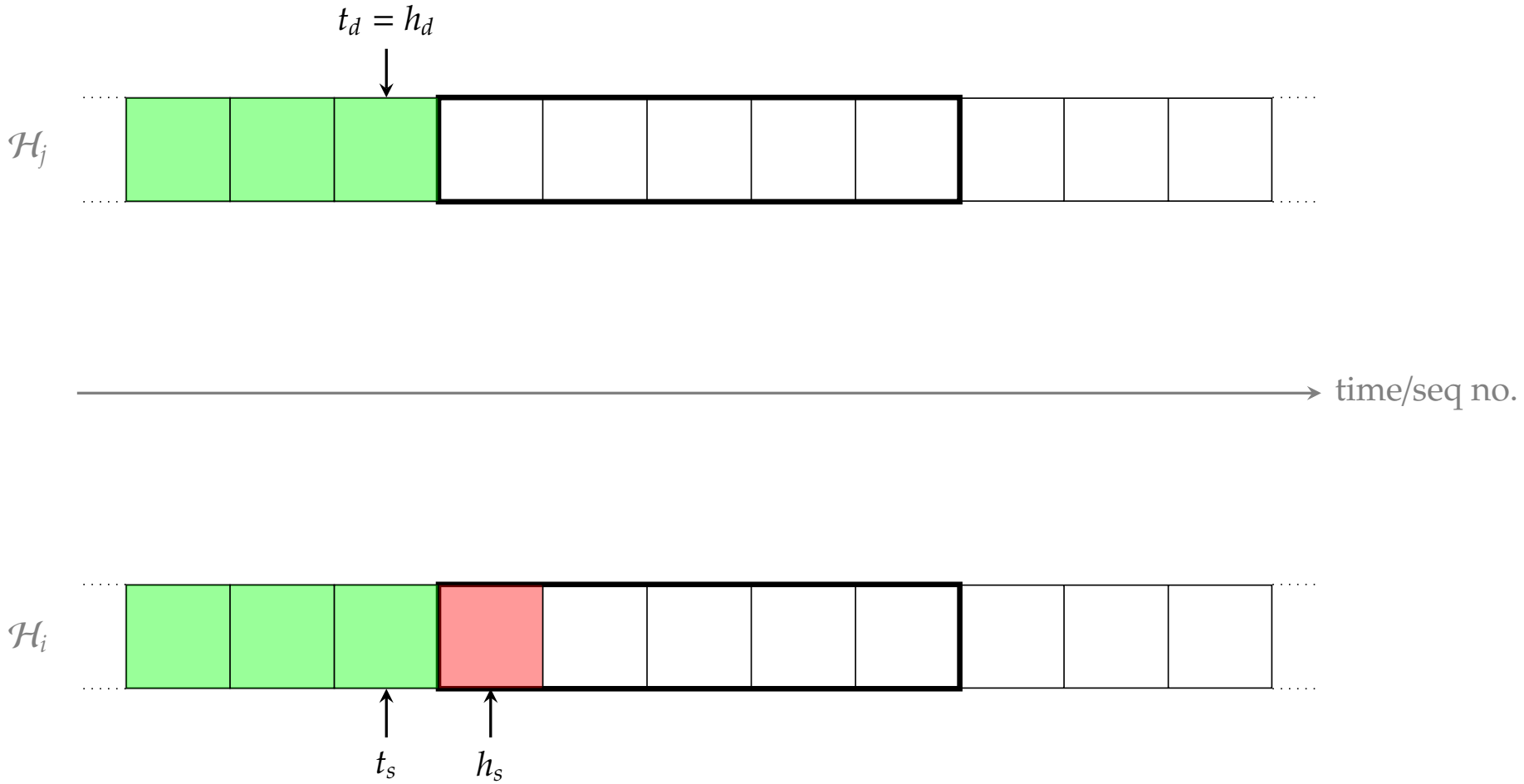


TCP Improvements (2)
Sliding-window ARQ

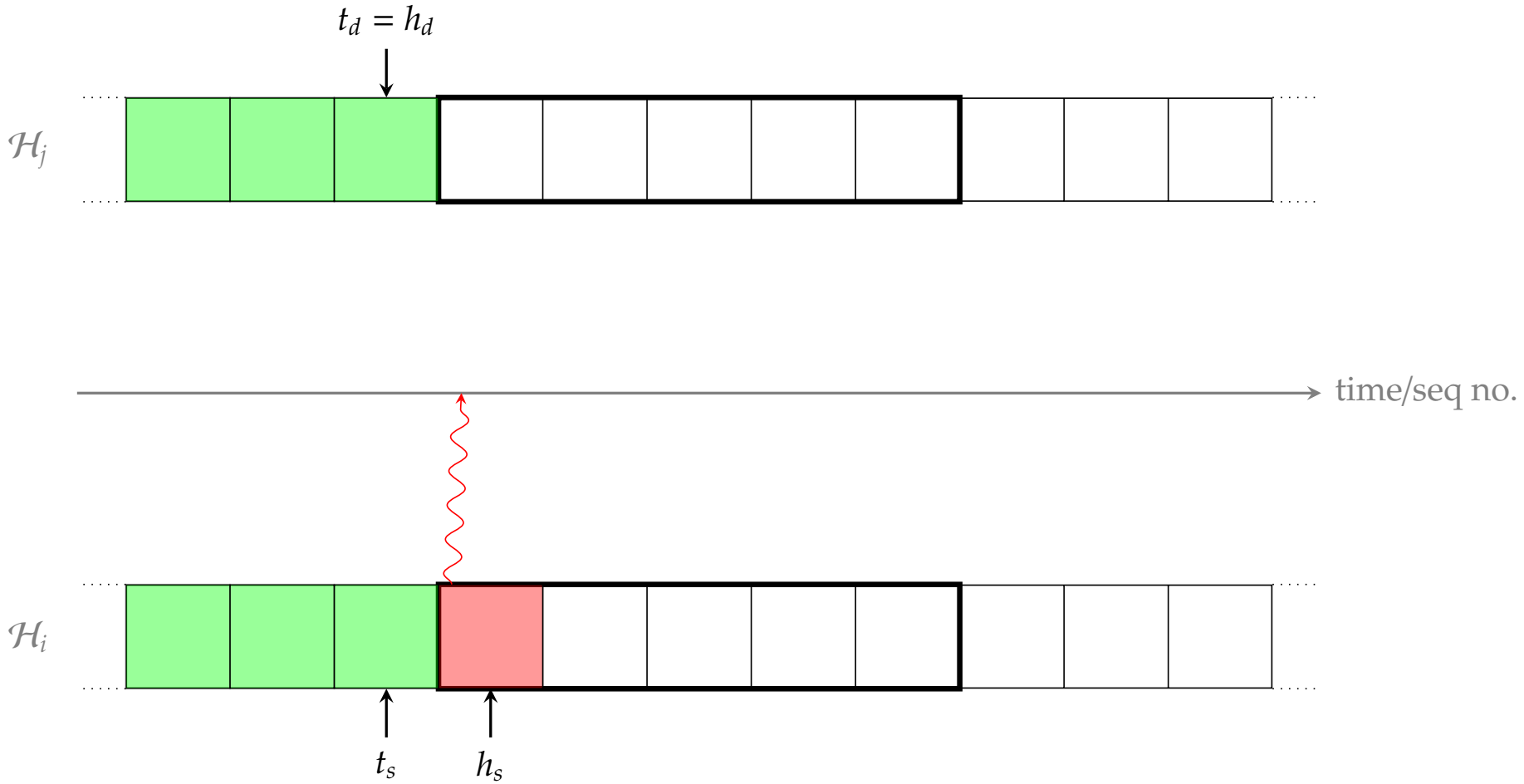
- Example:



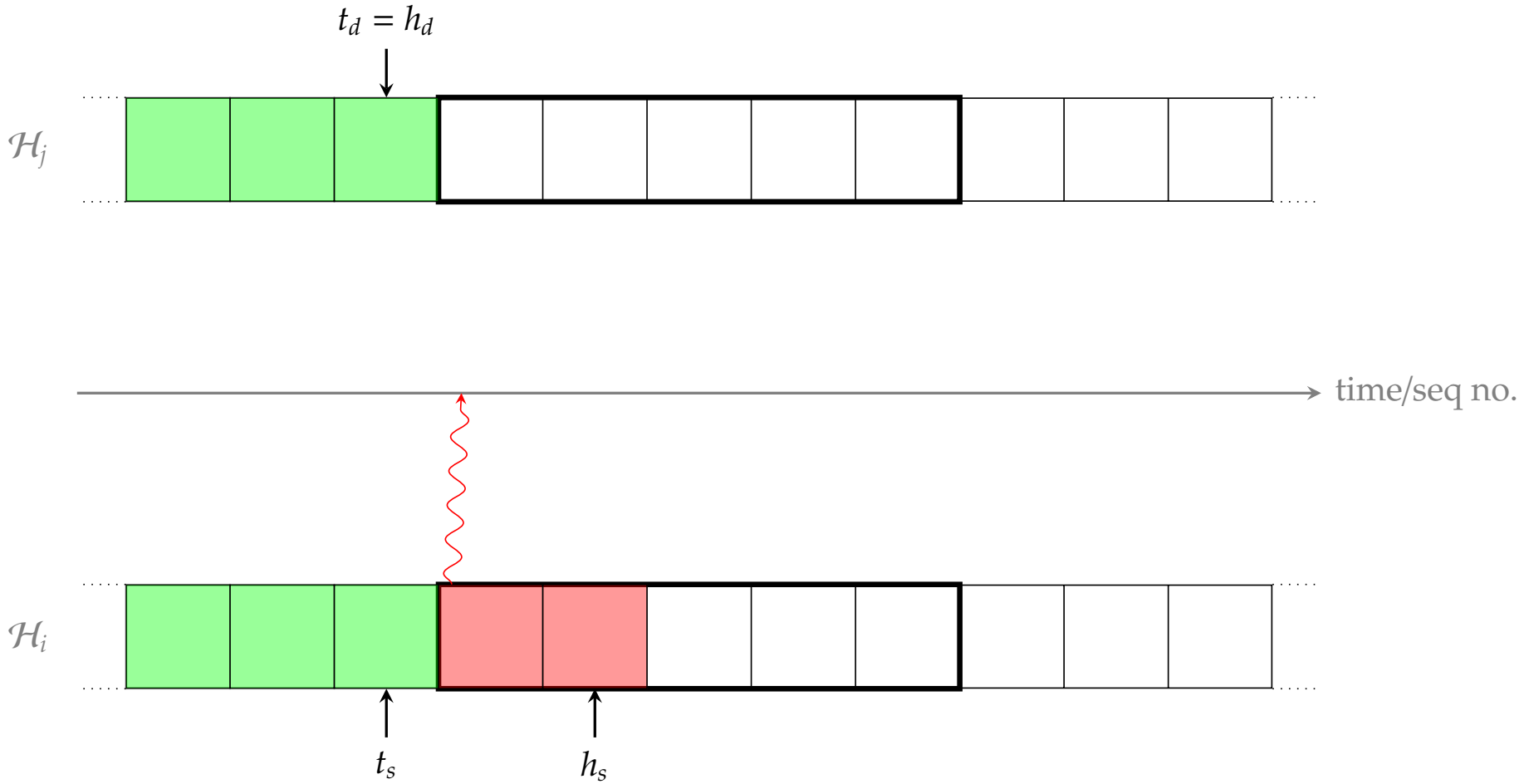
- Example: application layer (on source) invokes send.



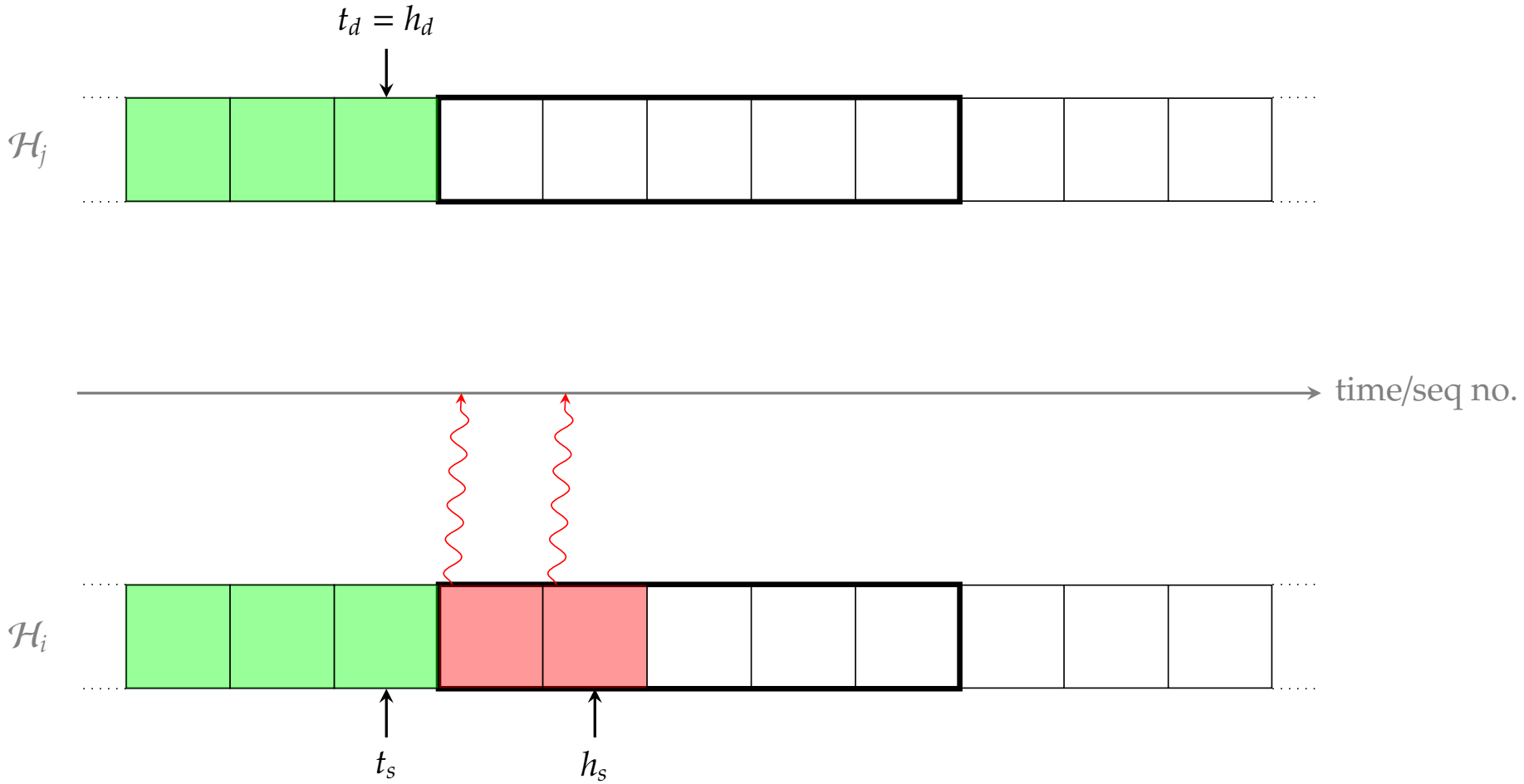
- Example: transport layer (on source) sends segment.



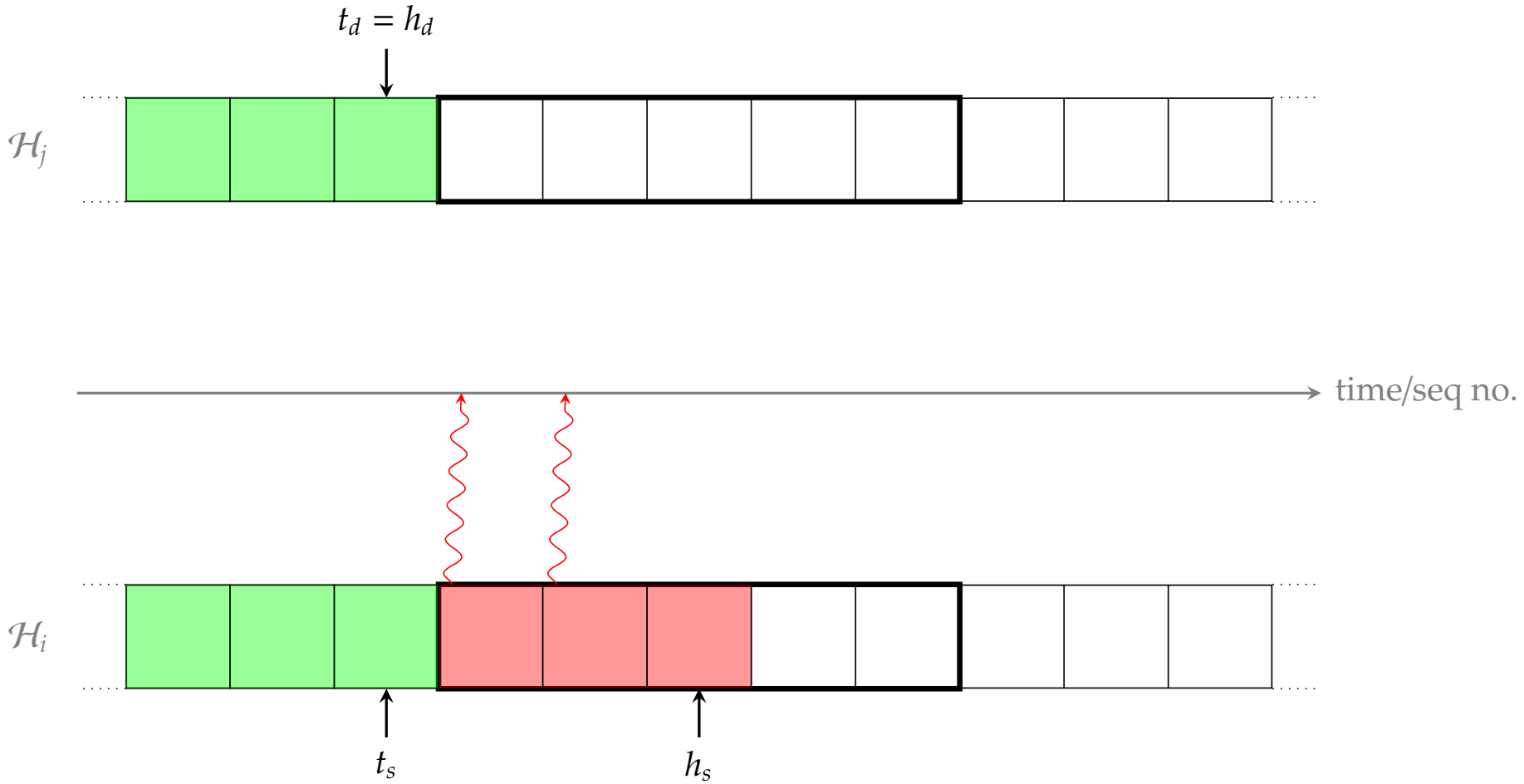
- Example: application layer (on source) invokes send.



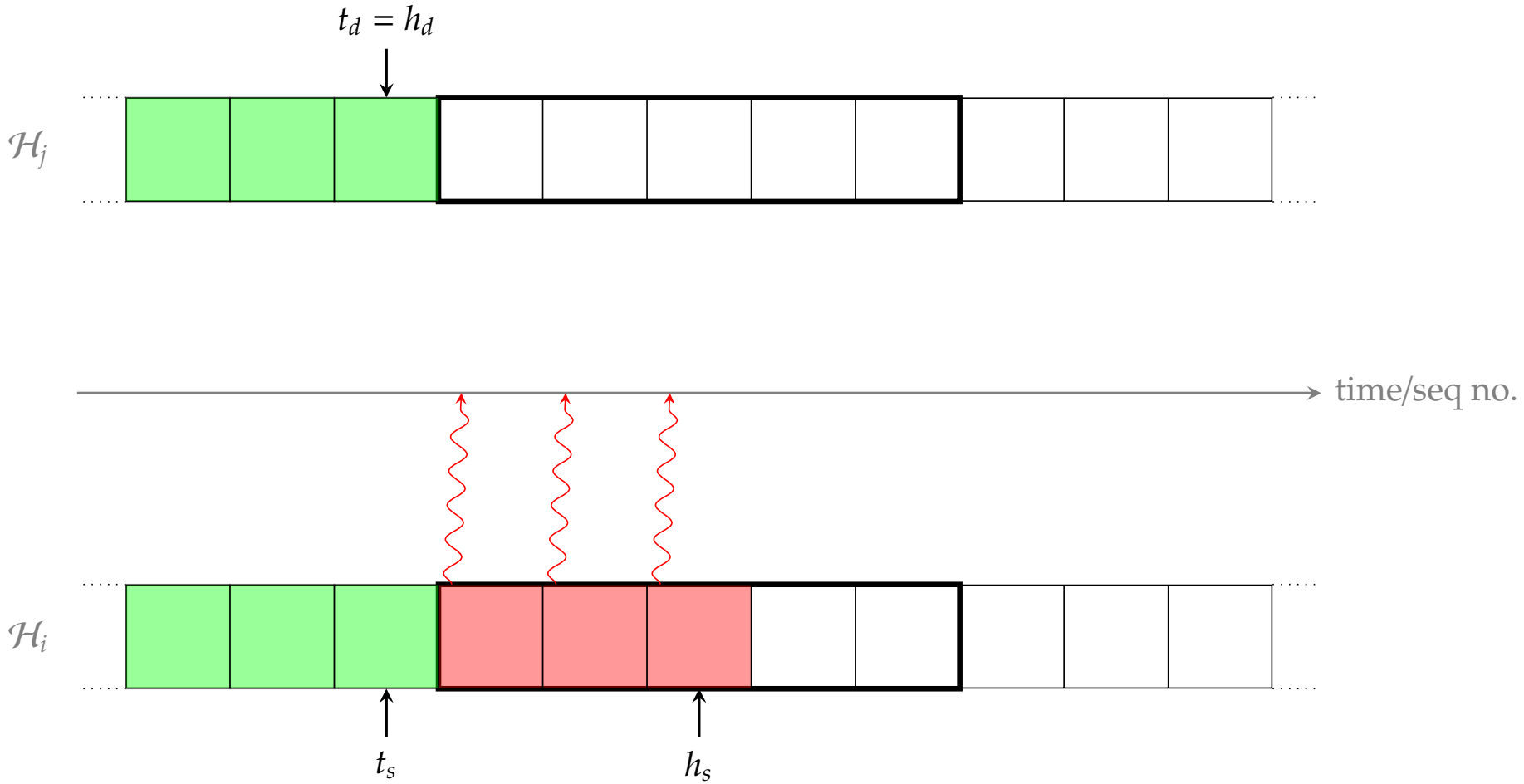
- Example: transport layer (on source) sends segment.



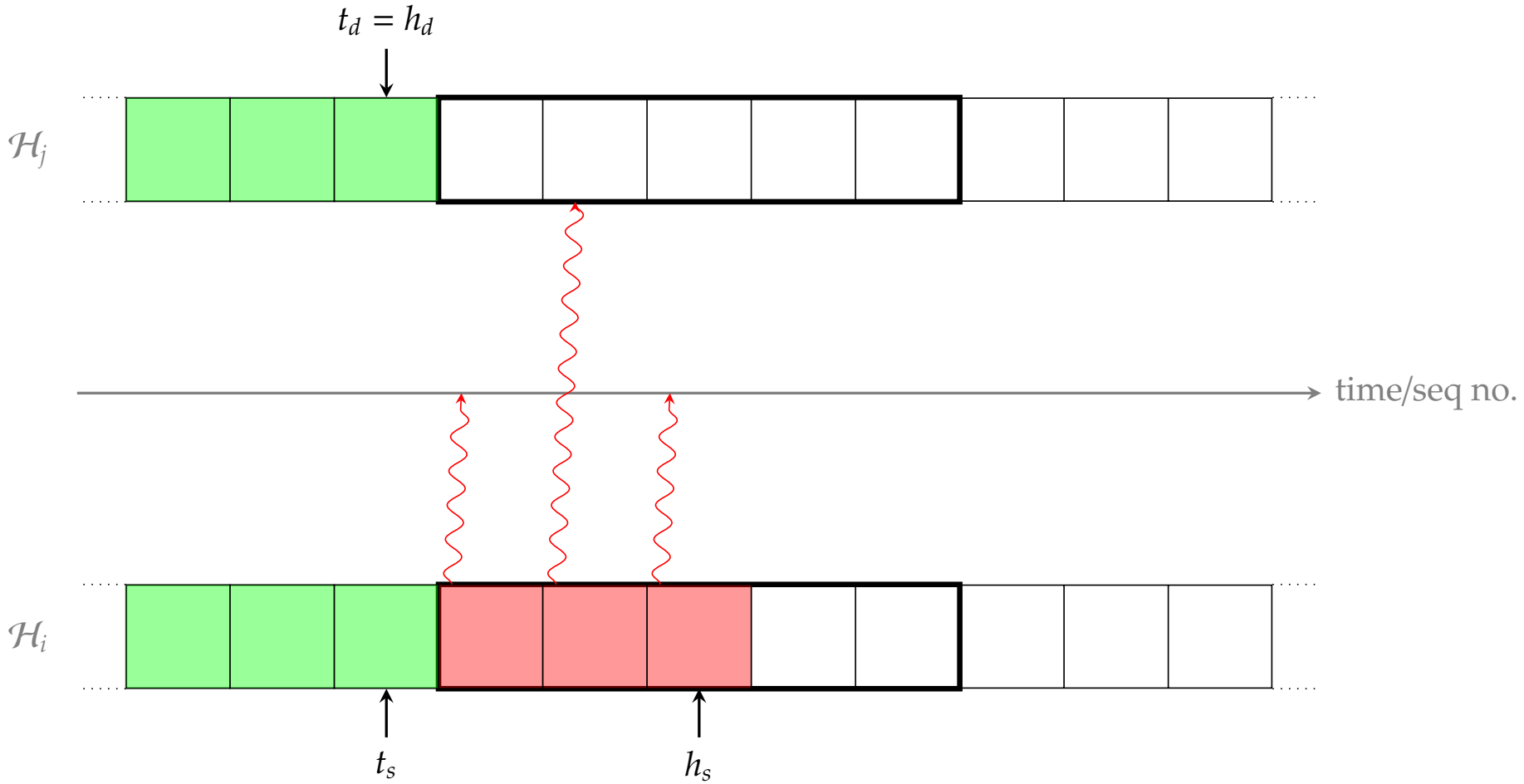
- Example: application layer (on source) invokes send.



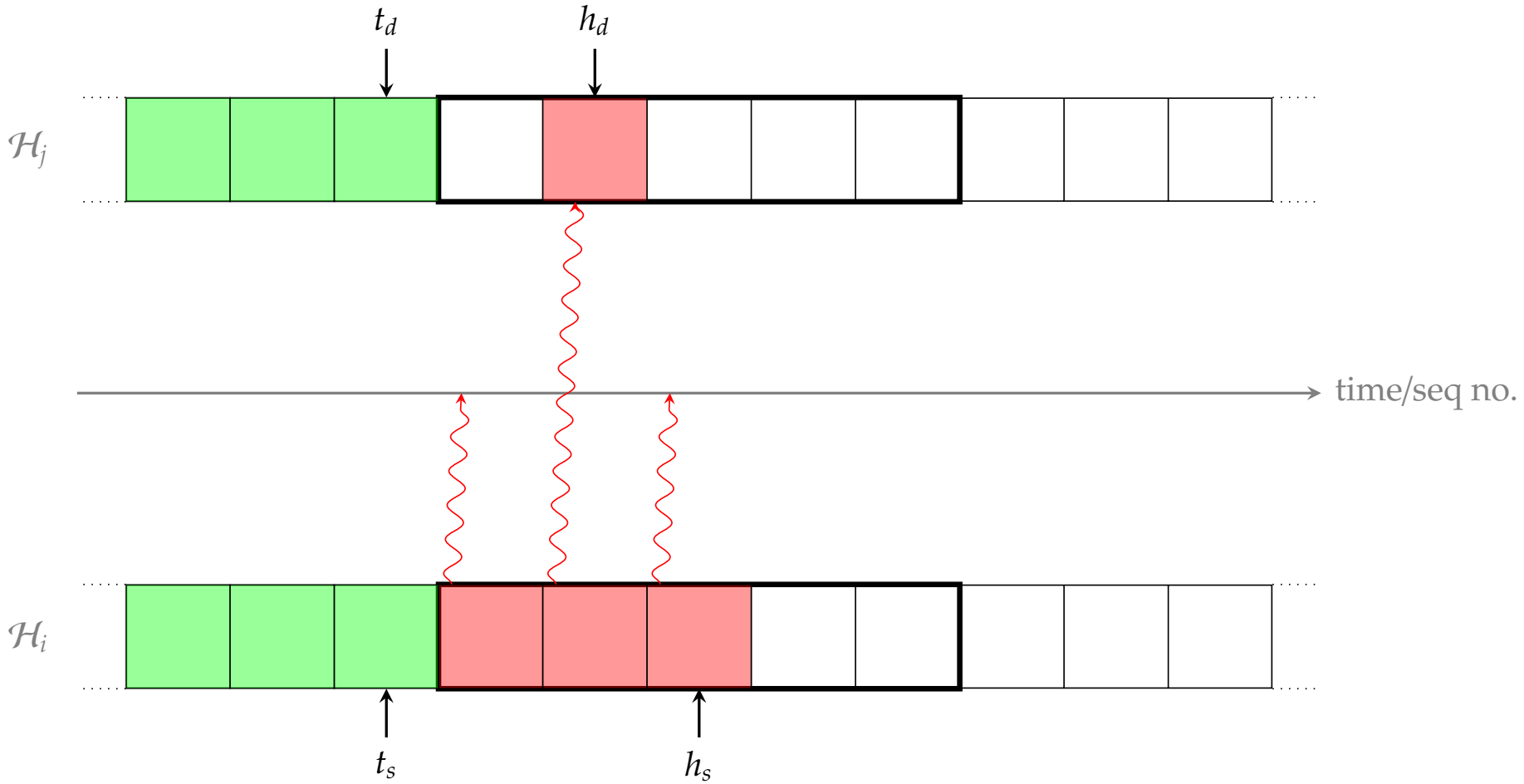
- Example: transport layer (on source) sends segment.



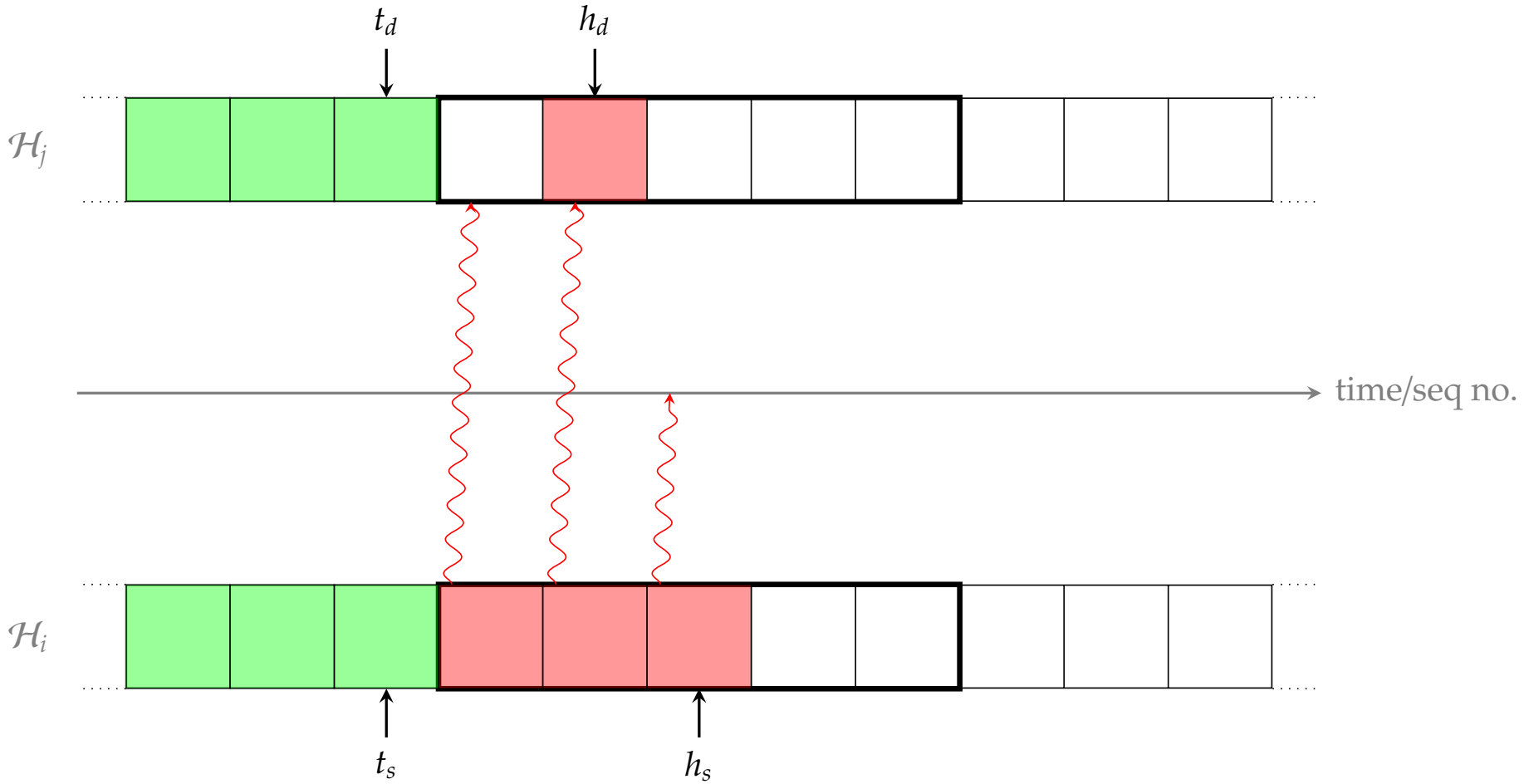
- Example: transport layer (on destination) recieves segment.



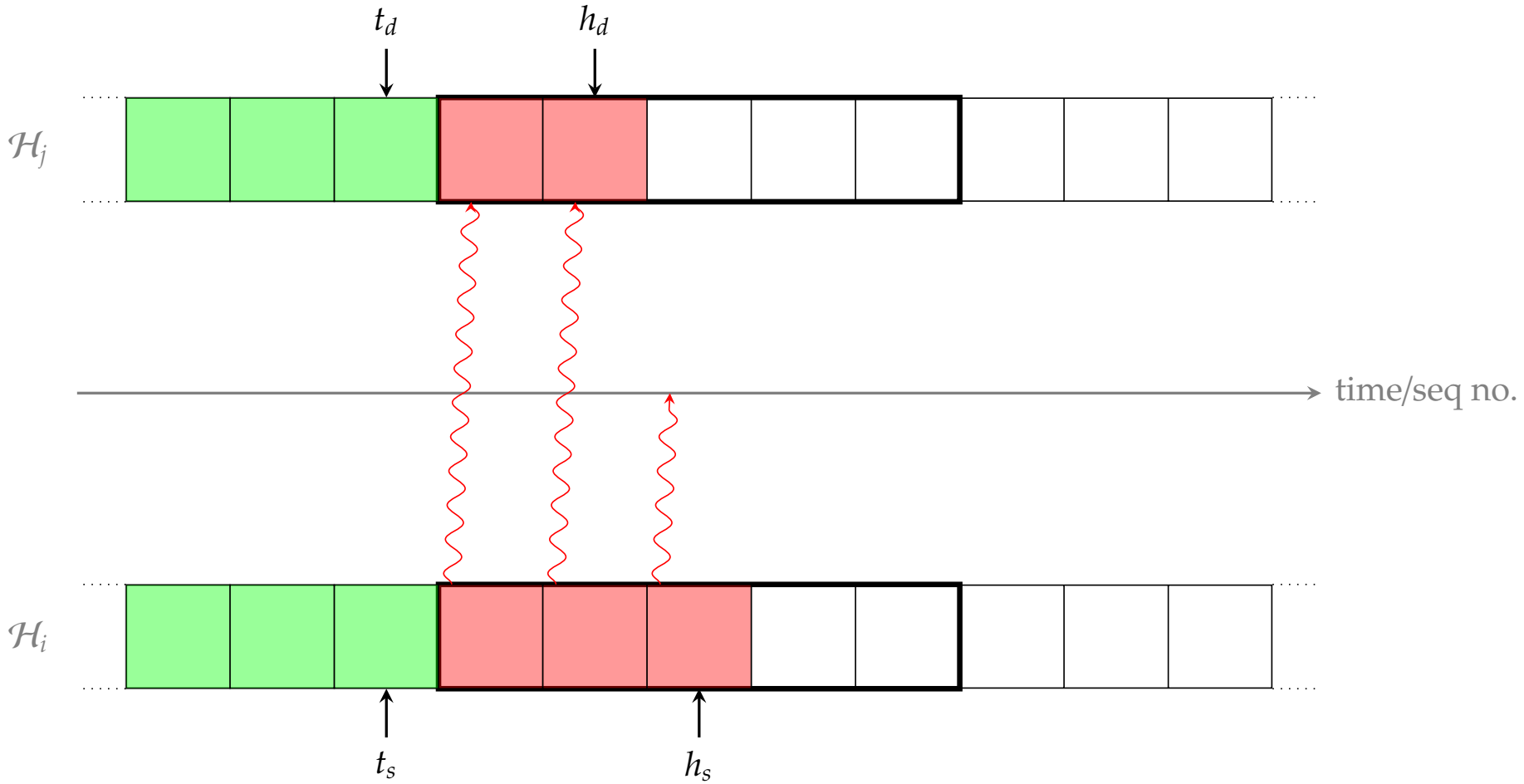
- Example: transport layer (on destination) recieves segment.



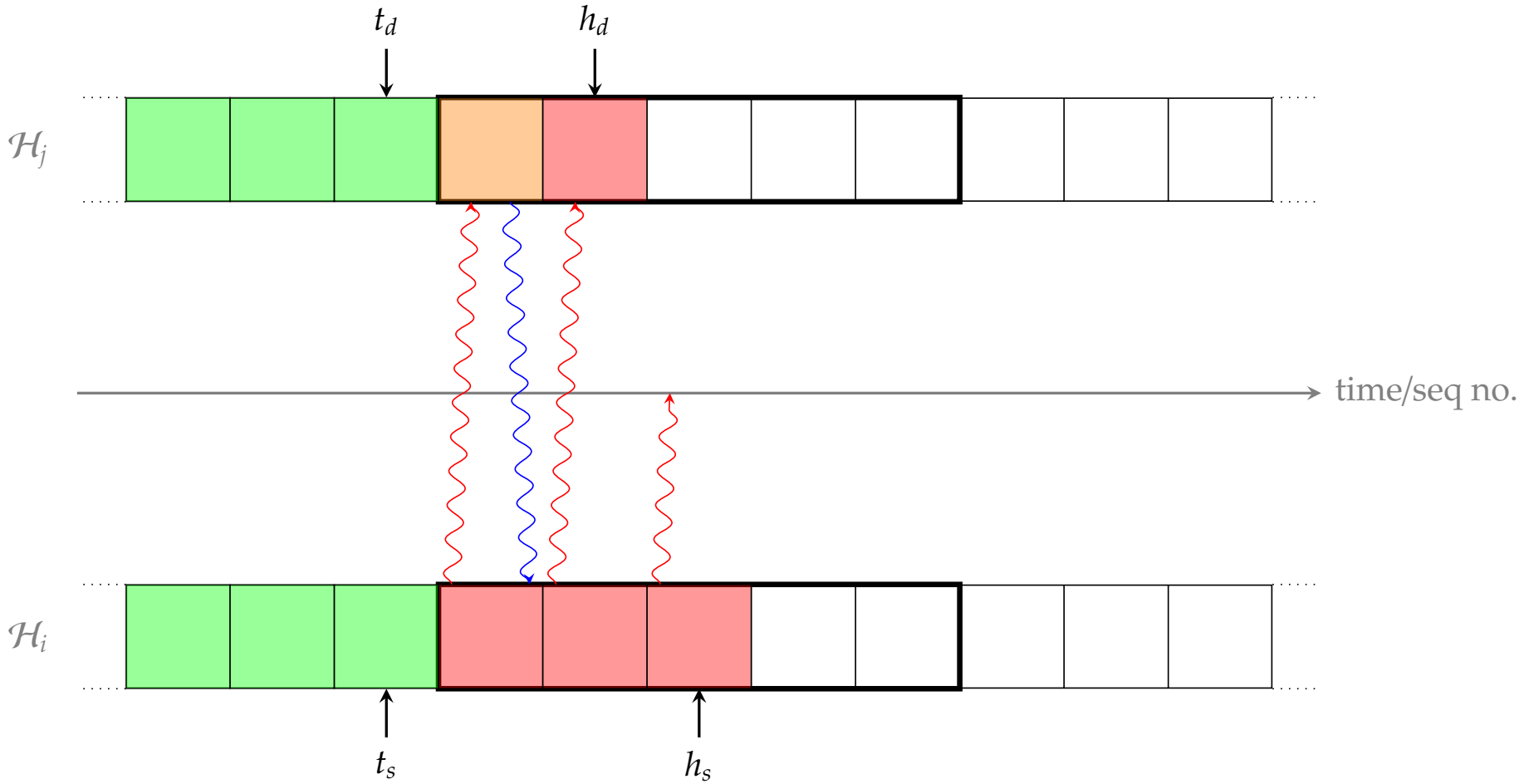
- Example: transport layer (on destination) recieves segment.



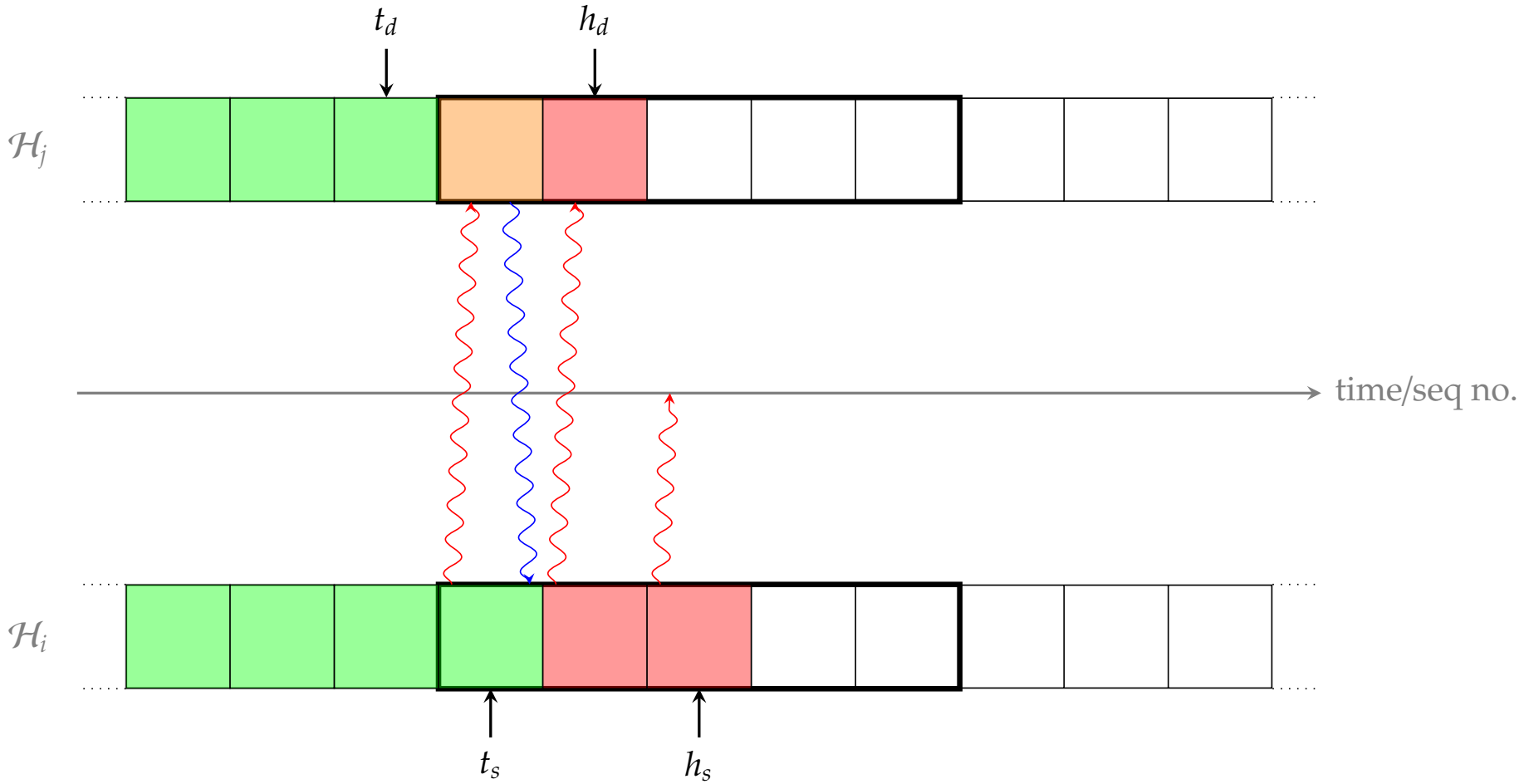
- Example: transport layer (on destination) recieves segment.



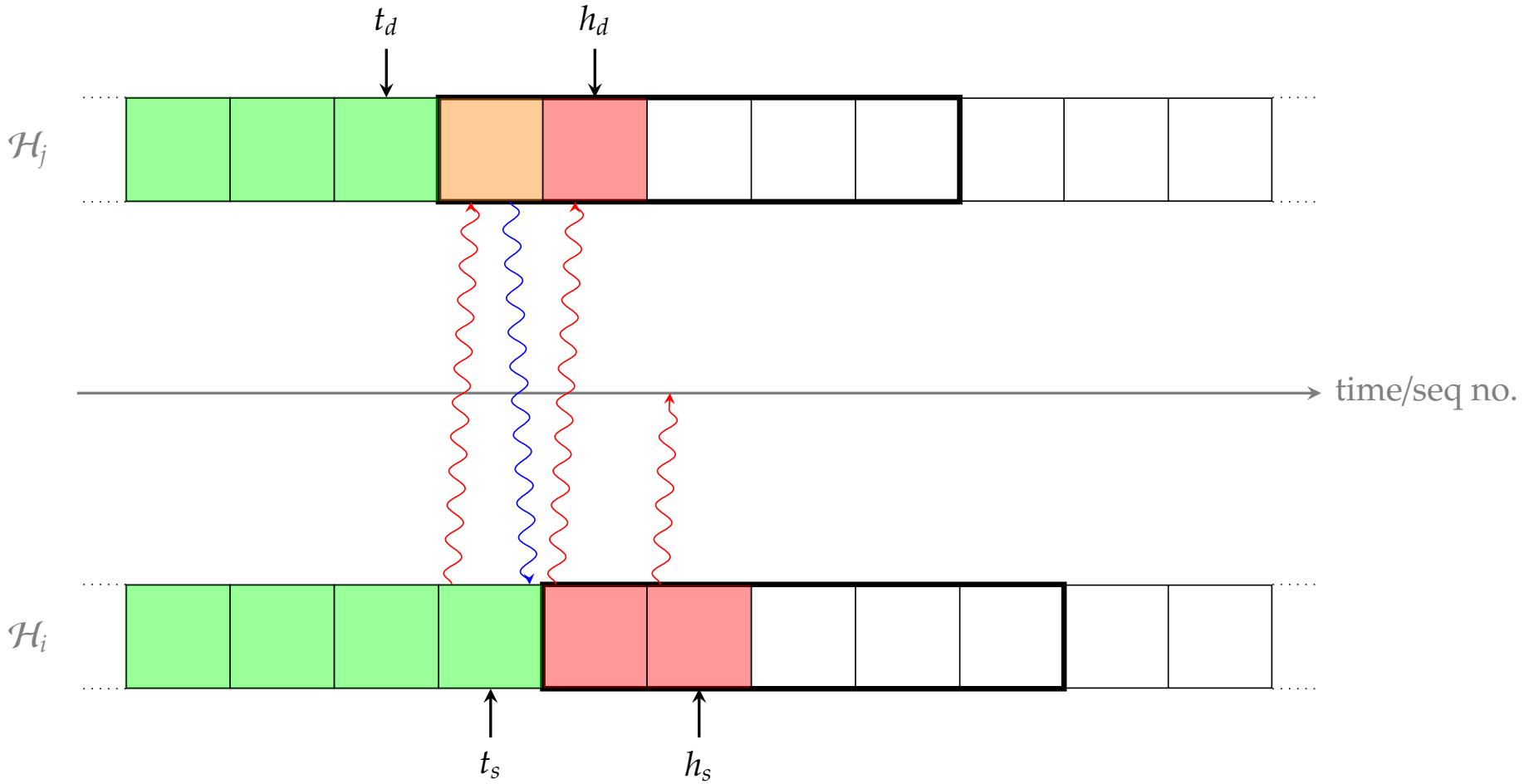
- Example: transport layer (on destination) sends ACK.



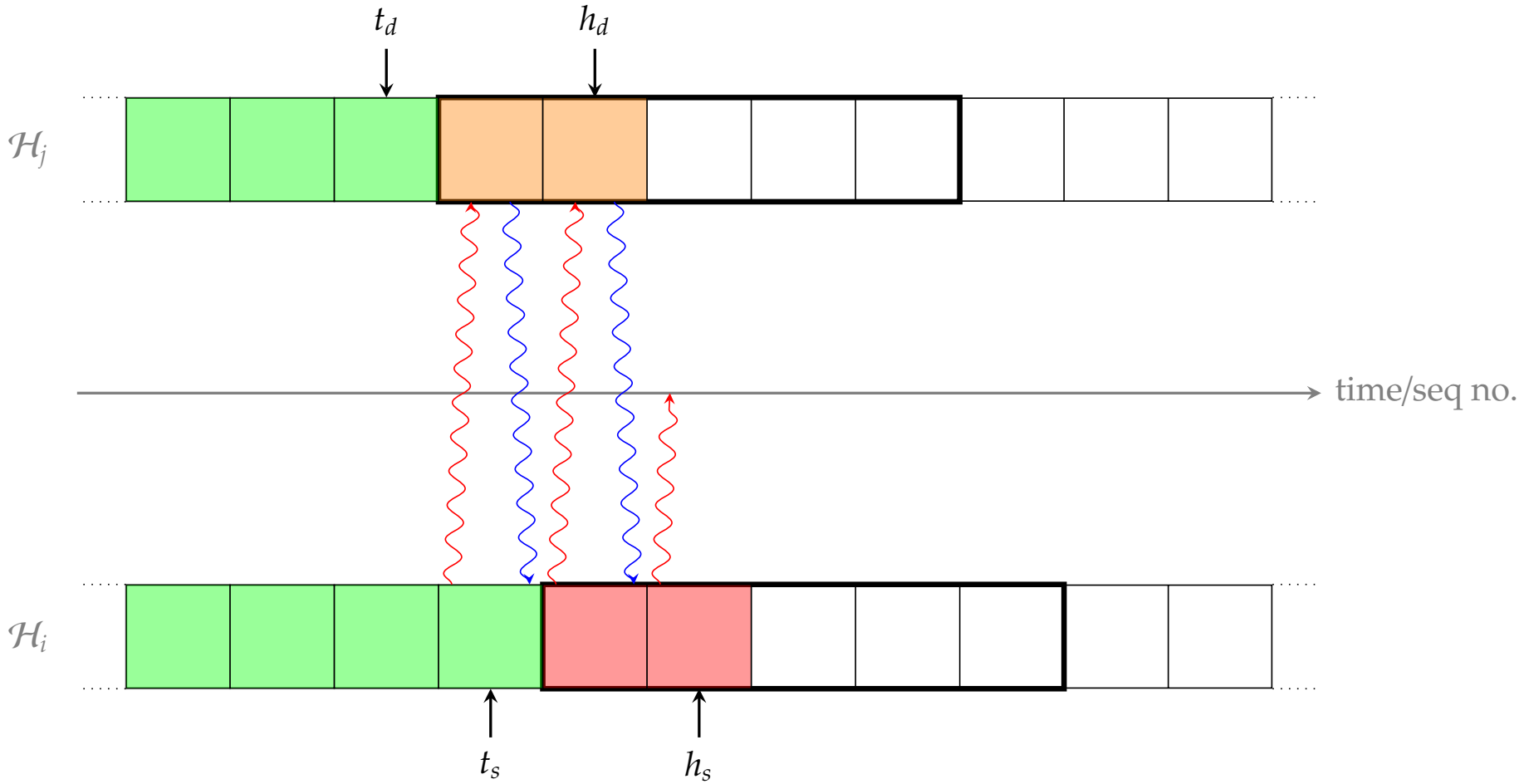
- Example: transport layer (on source) recieves ACK.



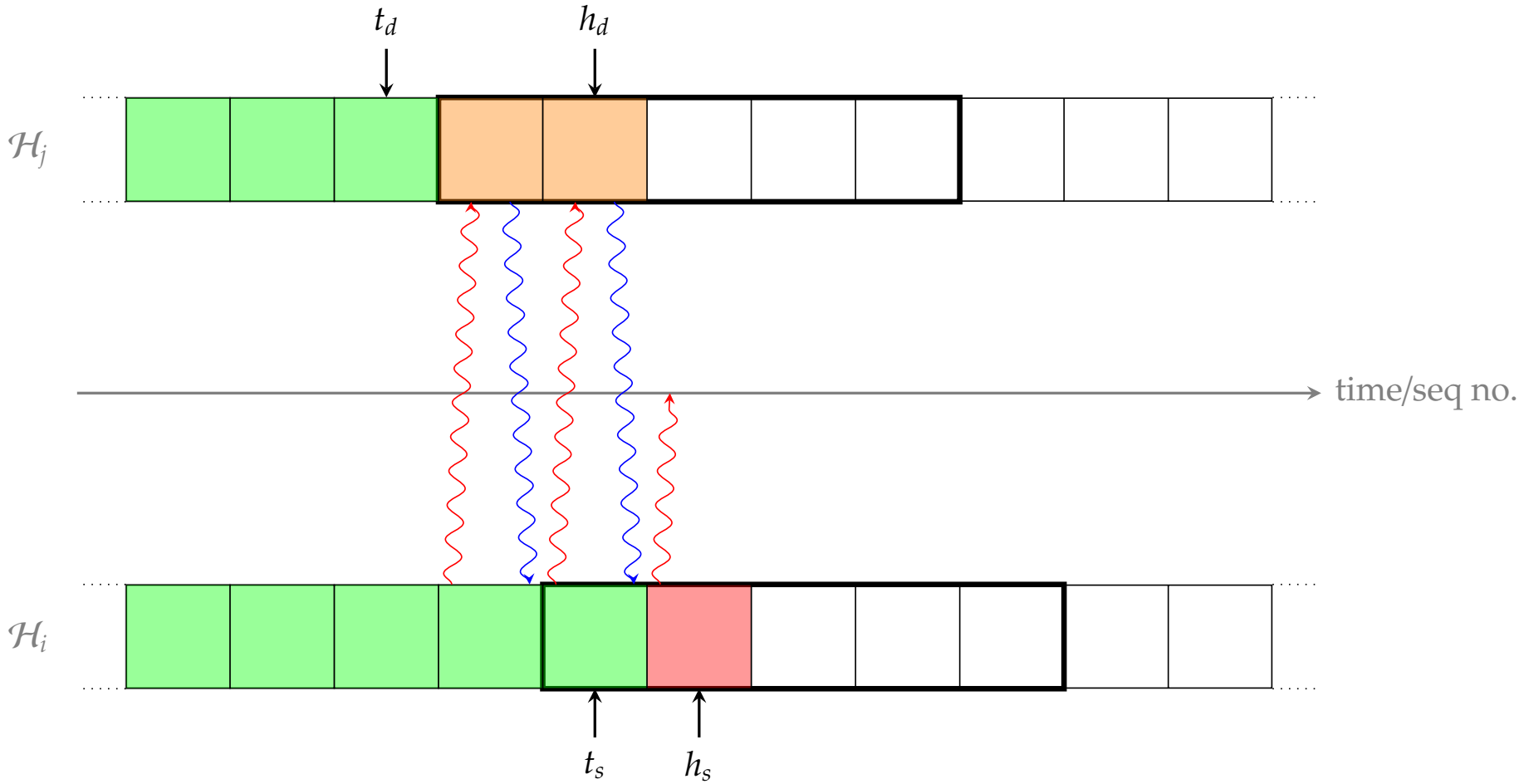
- Example: transport layer (on source) updates window.



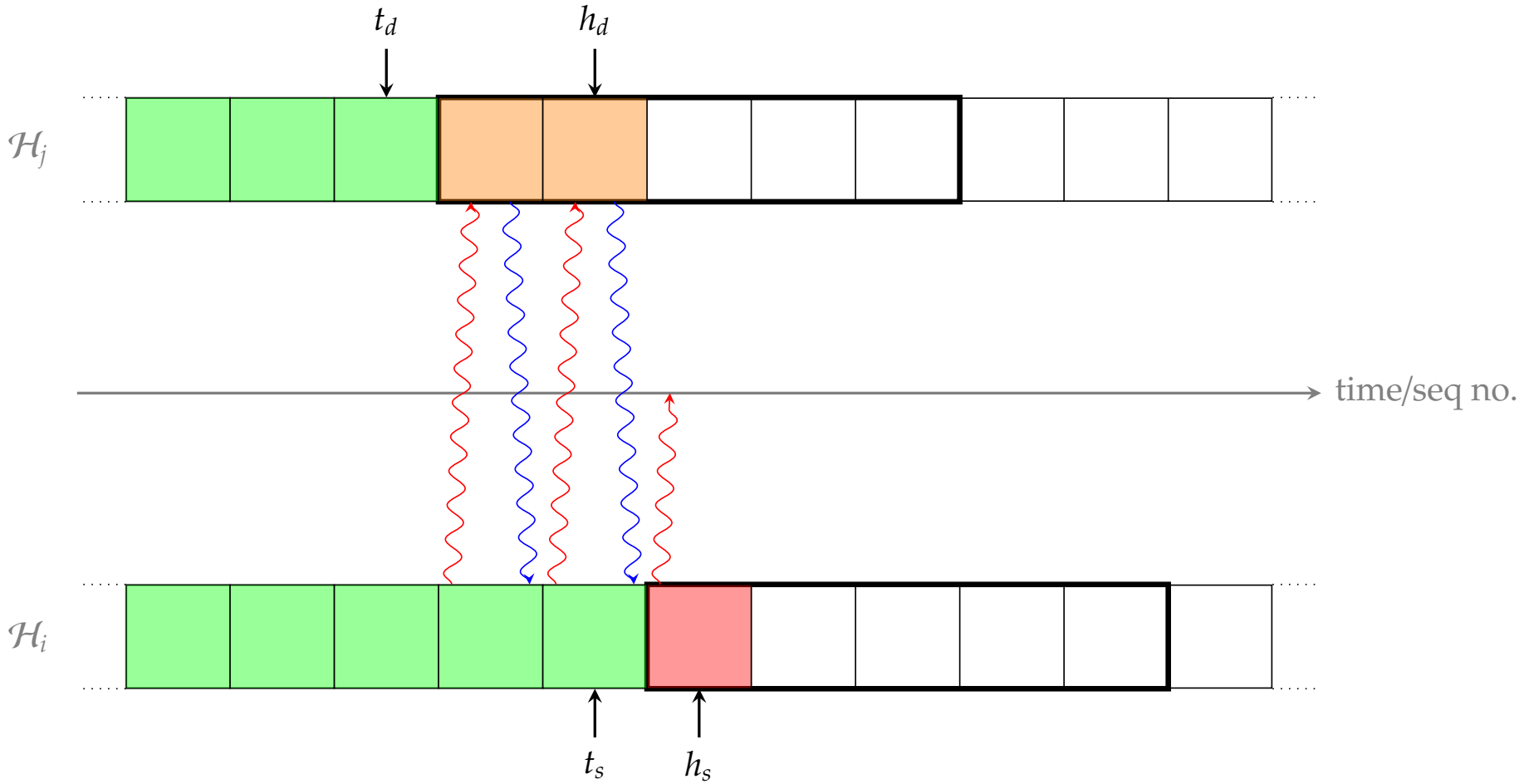
- Example: transport layer (on destination) sends ACK.



- Example: transport layer (on source) recieves ACK.

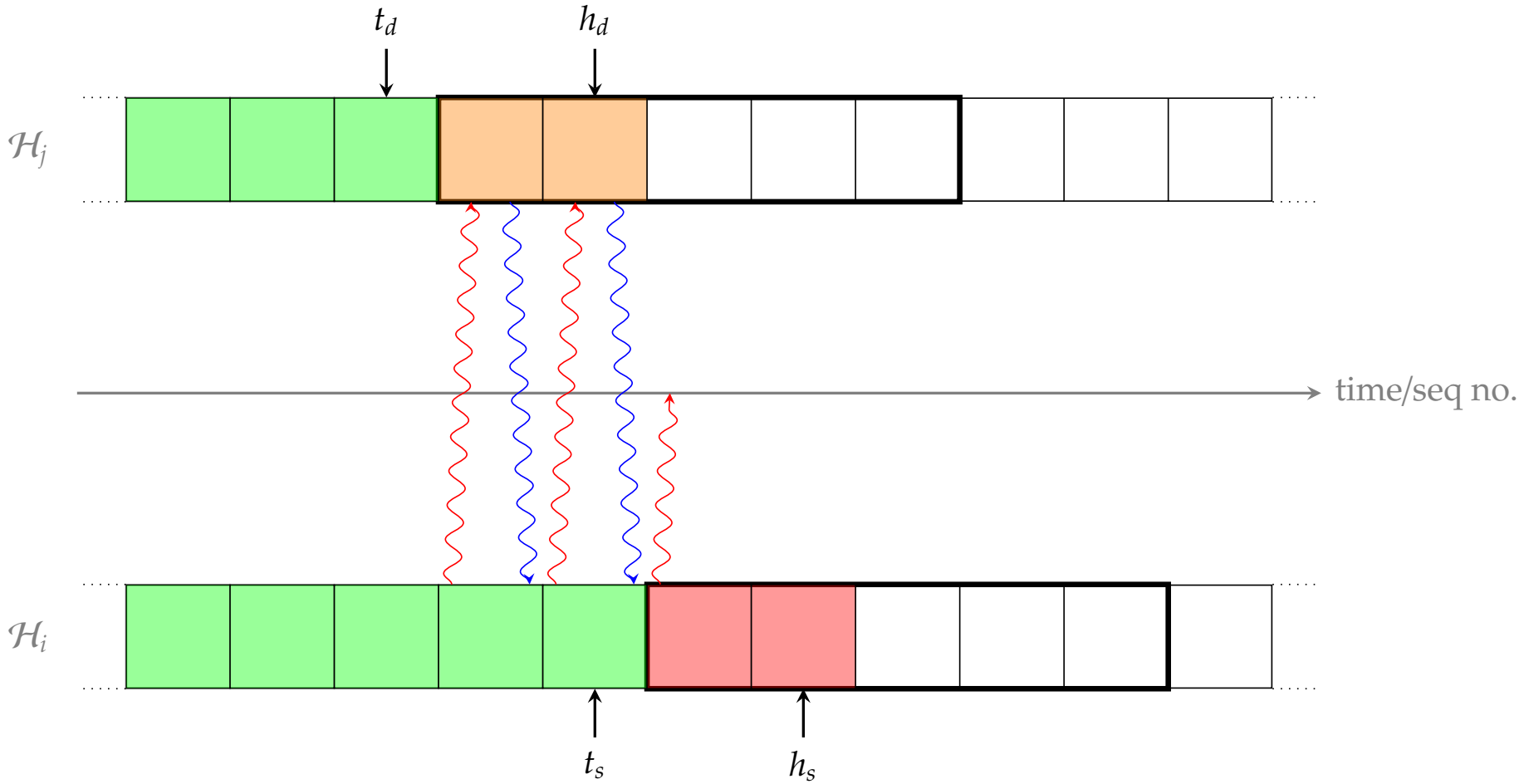


- Example: transport layer (on source) updates window.

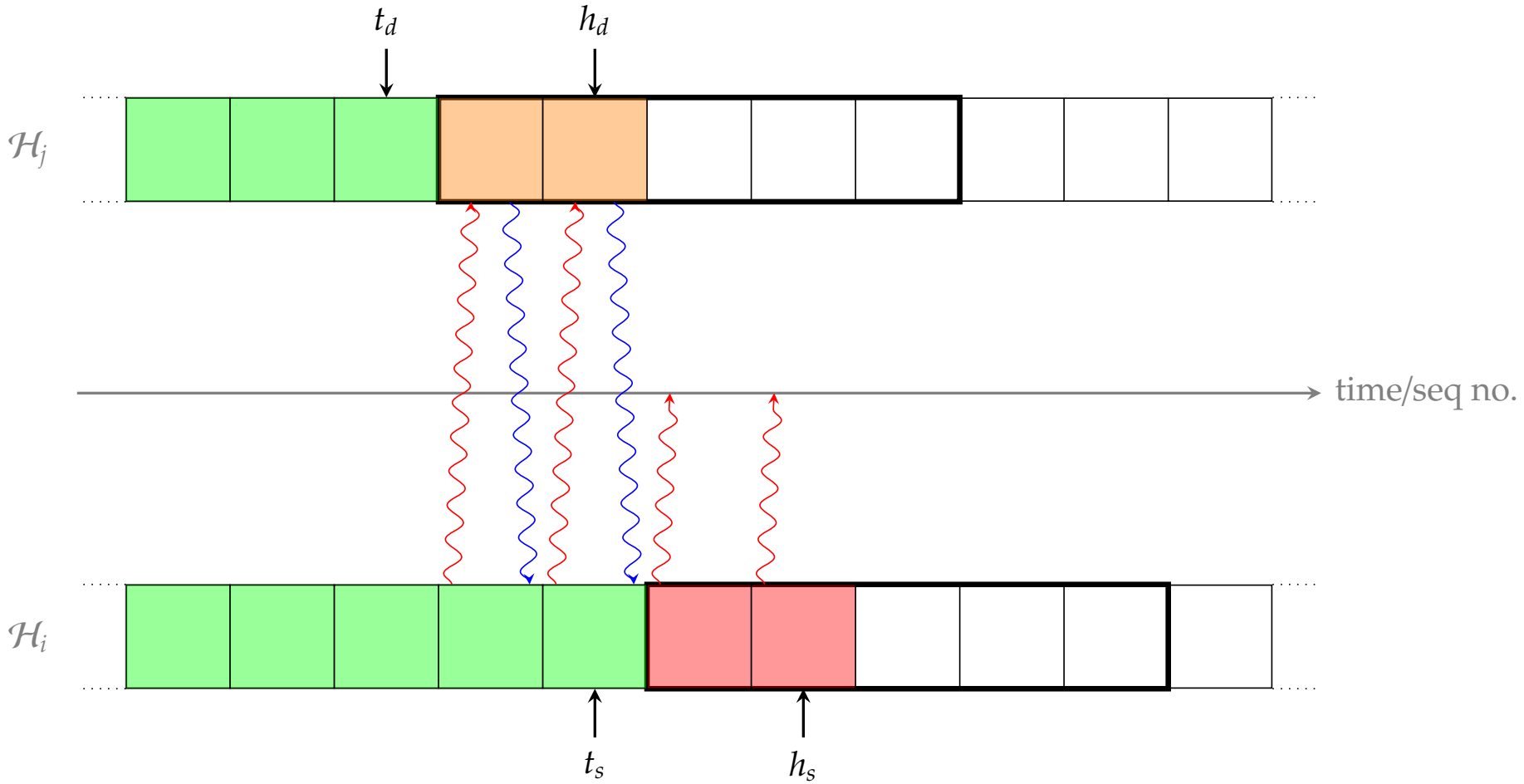


TCP Improvements (2)
Sliding-window ARQ

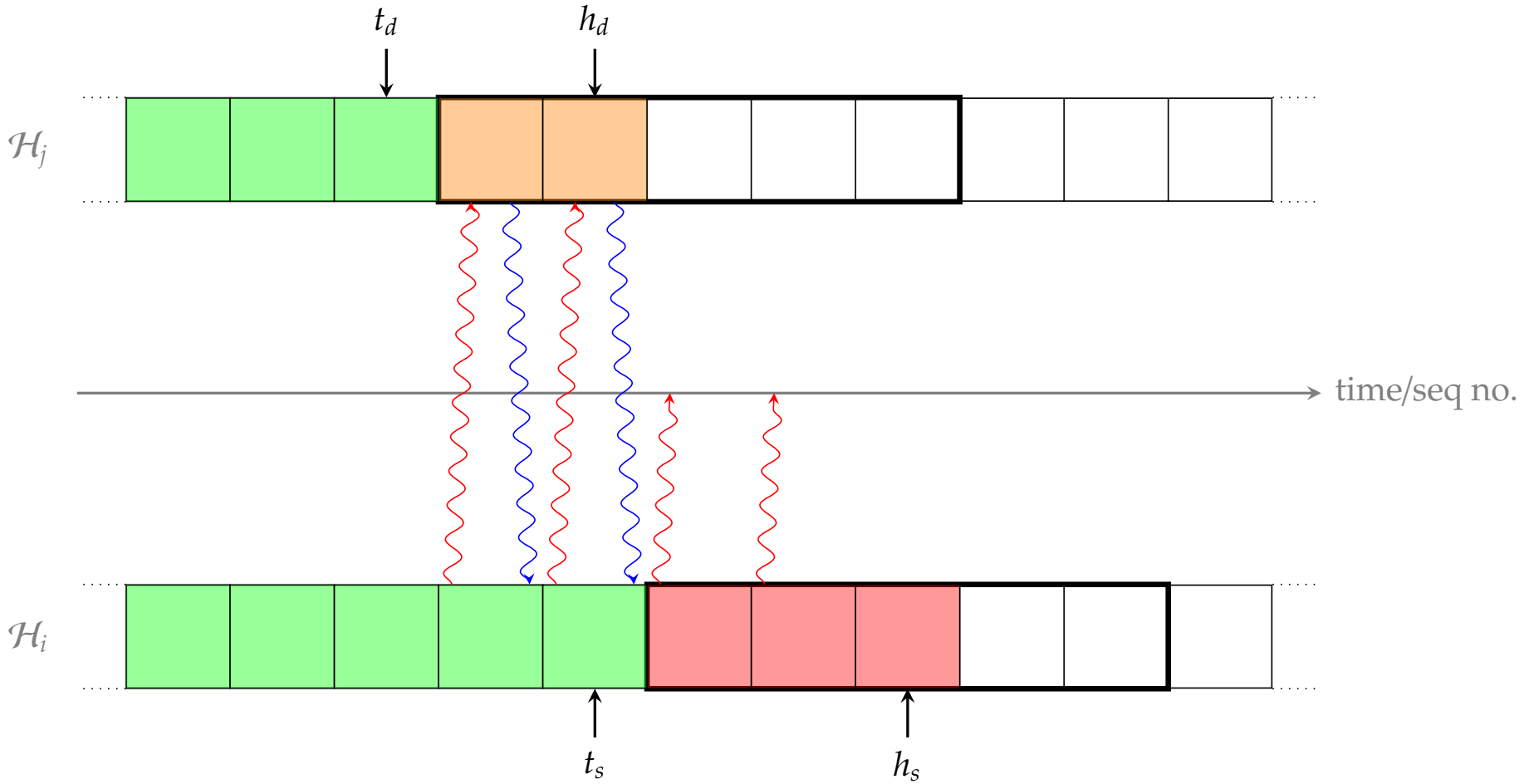
- Example: application layer (on source) invokes send.



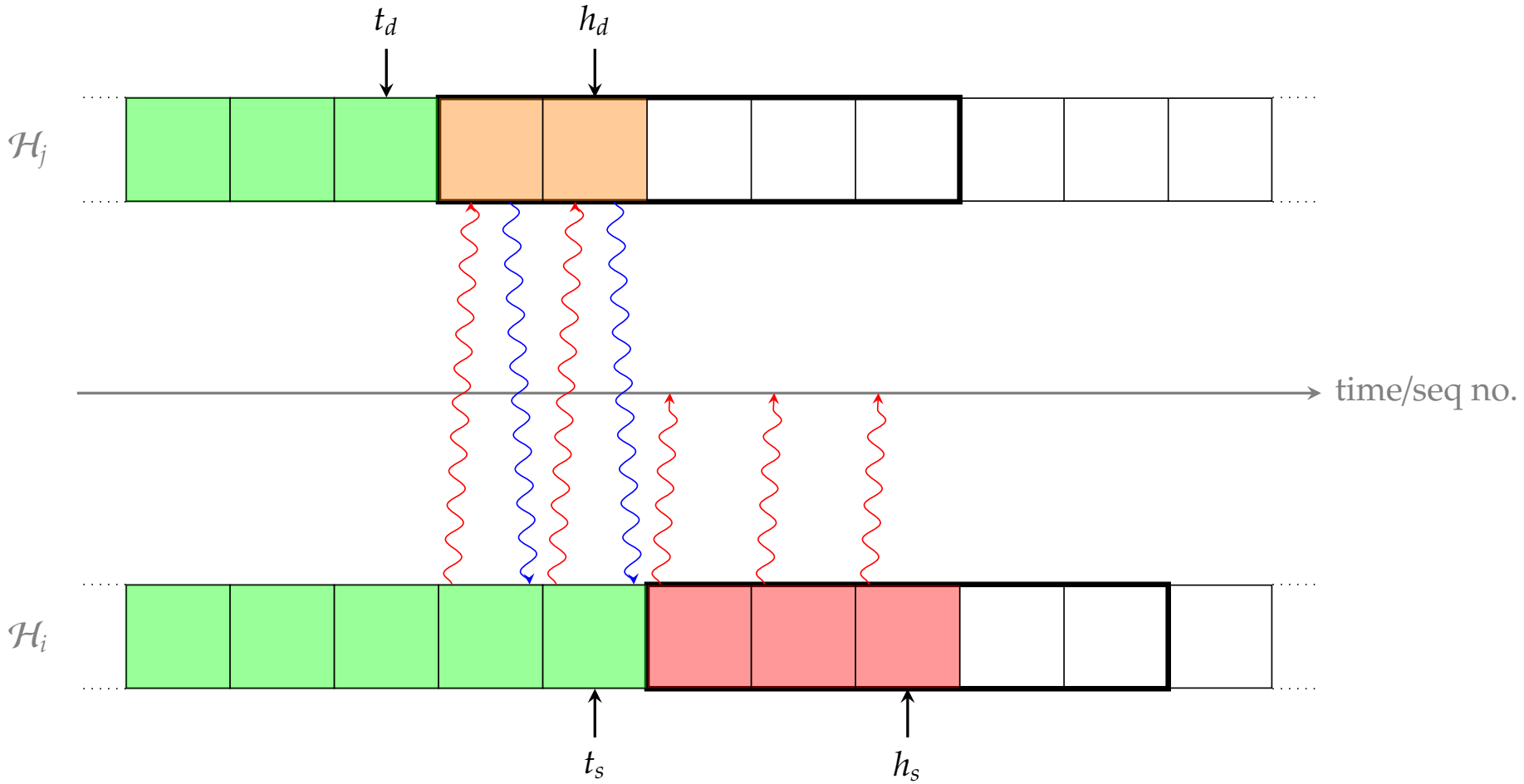
- Example: transport layer (on source) sends segment.



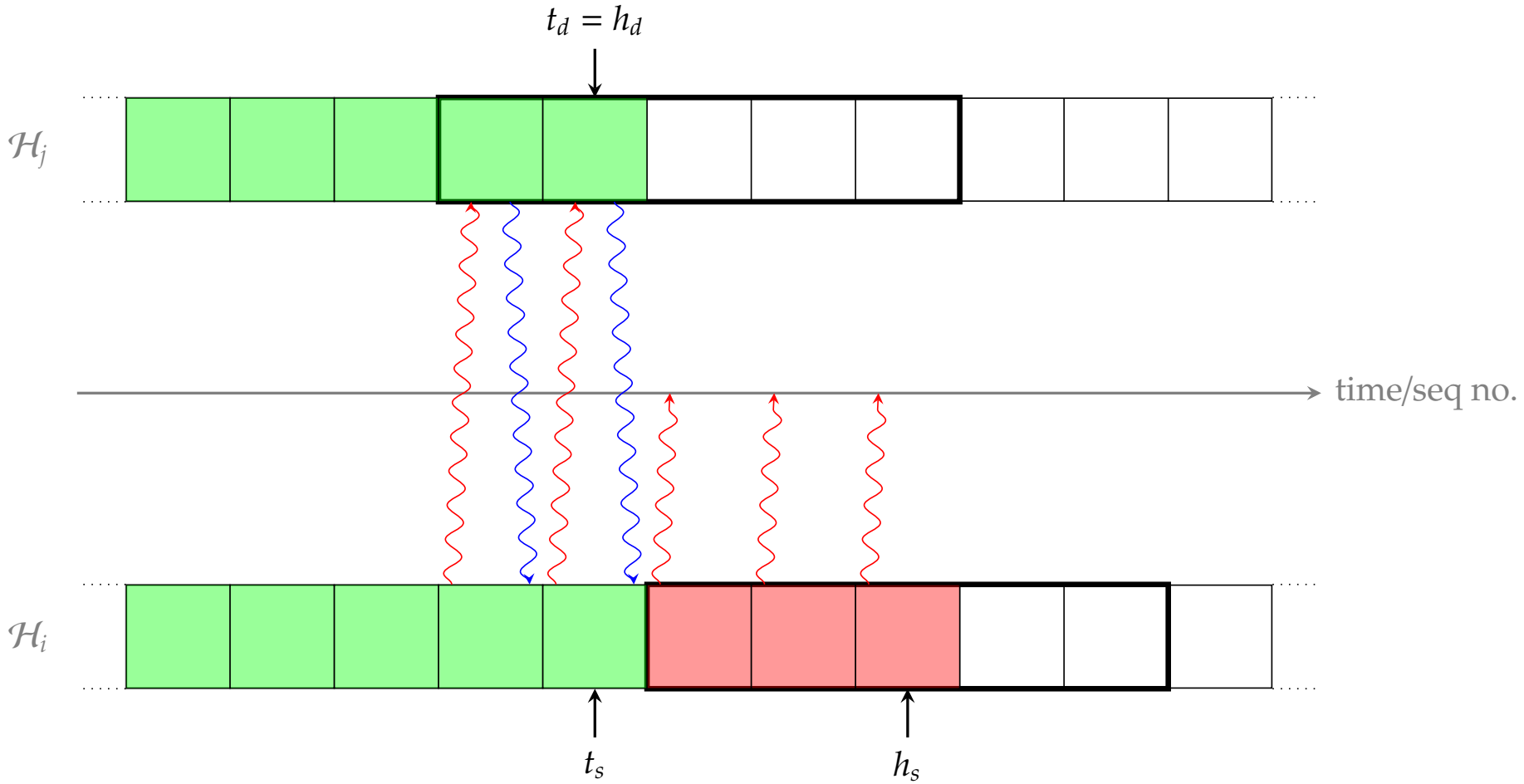
- Example: application layer (on source) invokes send.



- Example: transport layer (on source) sends segment.

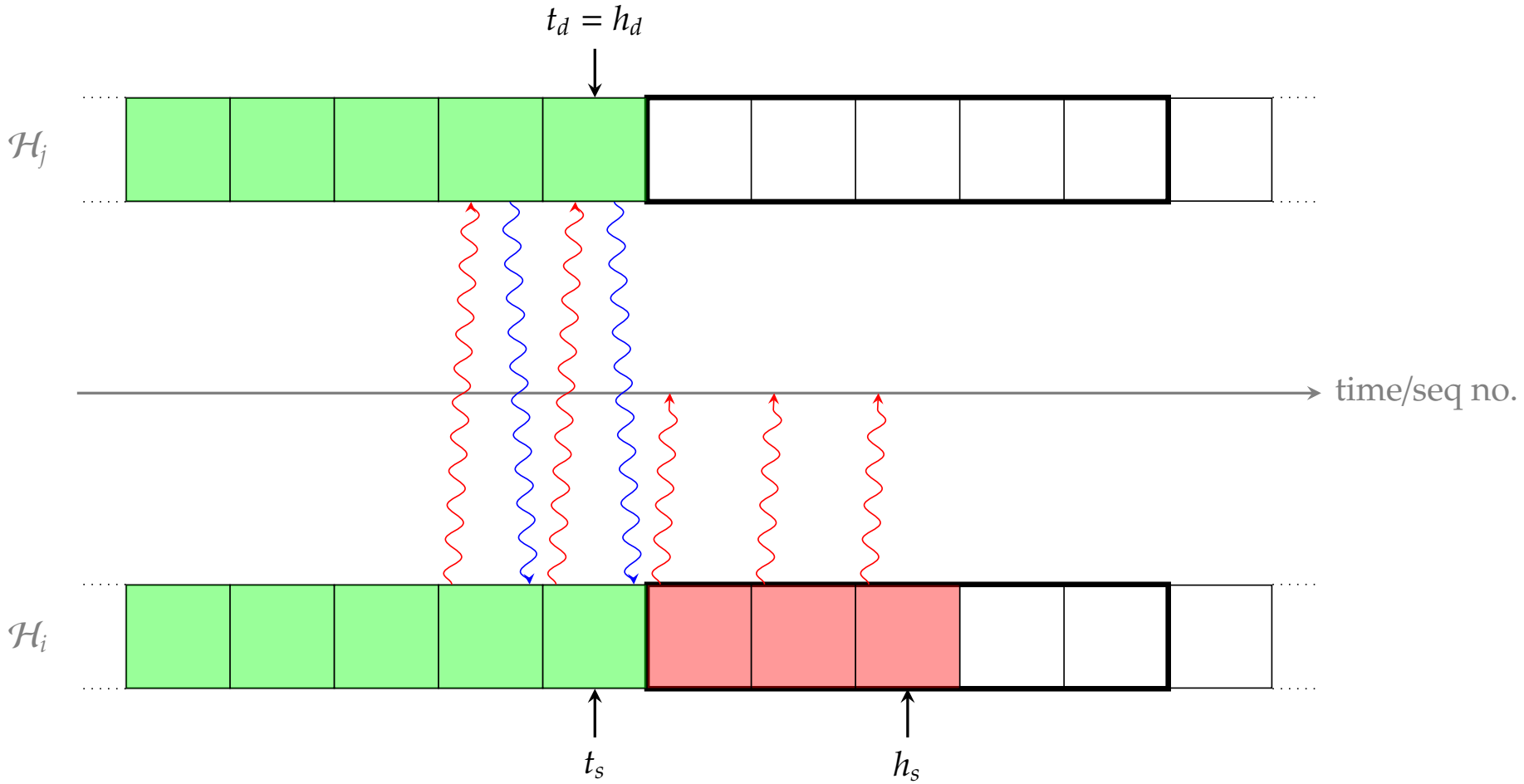


- Example: application layer (on destination) invokes `recv`.



TCP Improvements (2)
Sliding-window ARQ

- Example: transport layer (on destination) updates window.



Algorithm (source)

Assuming the source maintains

- a w_s-element (cyclic) buffer,
- t_s and h_s pointers into the buffer, and
- w_s time-out timers

then various events can occur:

1. Application layer invokes `send`:
   - 1.1 if $h_s < t_s + w_s$
     - copy segment into buffer at h_s,
     - set $h_s \leftarrow h_s + 1$, then
     - transmit segment and start time-out timer
   - 1.2 otherwise block transmission.
2. Transport layer receives ACK:
   - set $t_s \leftarrow t_s + 1$, then
   - unblock upto one pending transmission.
3. Transport layer times-out:
   - *re*transmit segment wrt. timer that timed-out.

Algorithm (destination)

Assuming the destination maintains

- a w_d-segment (cyclic) buffer, and
- t_d and h_s pointers into the buffer

then various events can occur:

1. Application layer invokes `recv`:
   - copy upto m in-order segments from buffer at $t_d + 1$ onward,
   - set $t_d \leftarrow t_d + m$, then
   - transmit ACK(s).
2. Transport layer receives data:
   - 2.1 if $t_d <$ seq no. $\leq t_d + w_d$, buffer segment,
   - 2.2 otherwise drop segment.

TCP Improvements (4)
Sliding-window ARQ $\rightsquigarrow$ flow control

- ... it get's even better still:
  - imagine we allow w_s grow or shrink dynamically ...
  - ... smaller w_s "throttles" the source, i.e., reduces how many un-ACK'ed segments it can transmit: if we allow the *destination* to set w_s, this yields a flow control mechanism.
- Idea:
  1. destination includes w_a the **advertised window size**

     $$w_a = w_d - (h_d - h_a)$$

     in each ACK (via the window size field), and
  2. source uses **effective window size**

     $$w_e = \min(w_s, w_a)$$

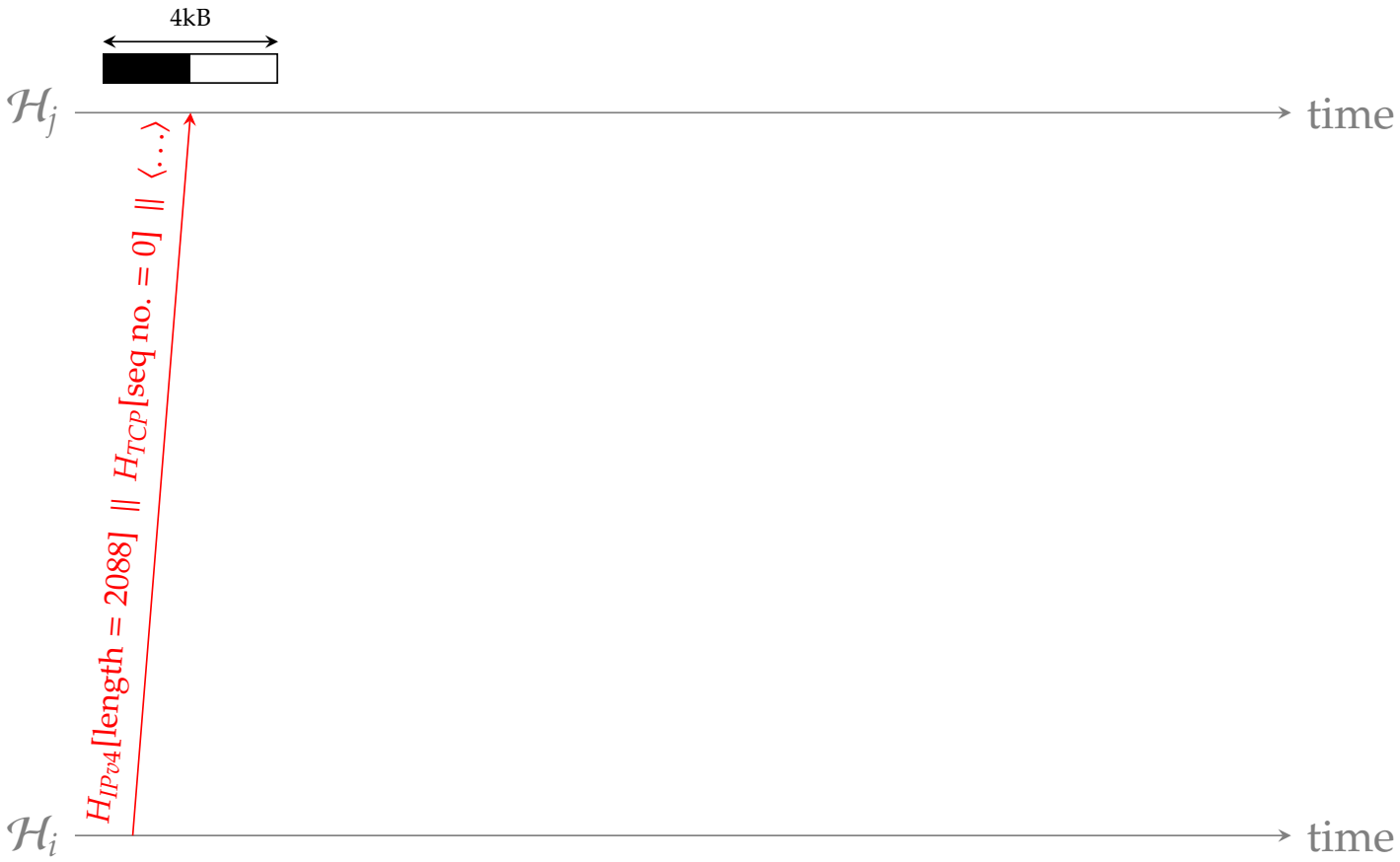
     instead of w_s.

TCP Improvements (5)
Sliding-window ARQ $\rightsquigarrow$ flow control

- Example: imagine the destination has a 4kB buffer.

TCP Improvements (5)
Sliding-window ARQ ⇝ flow control

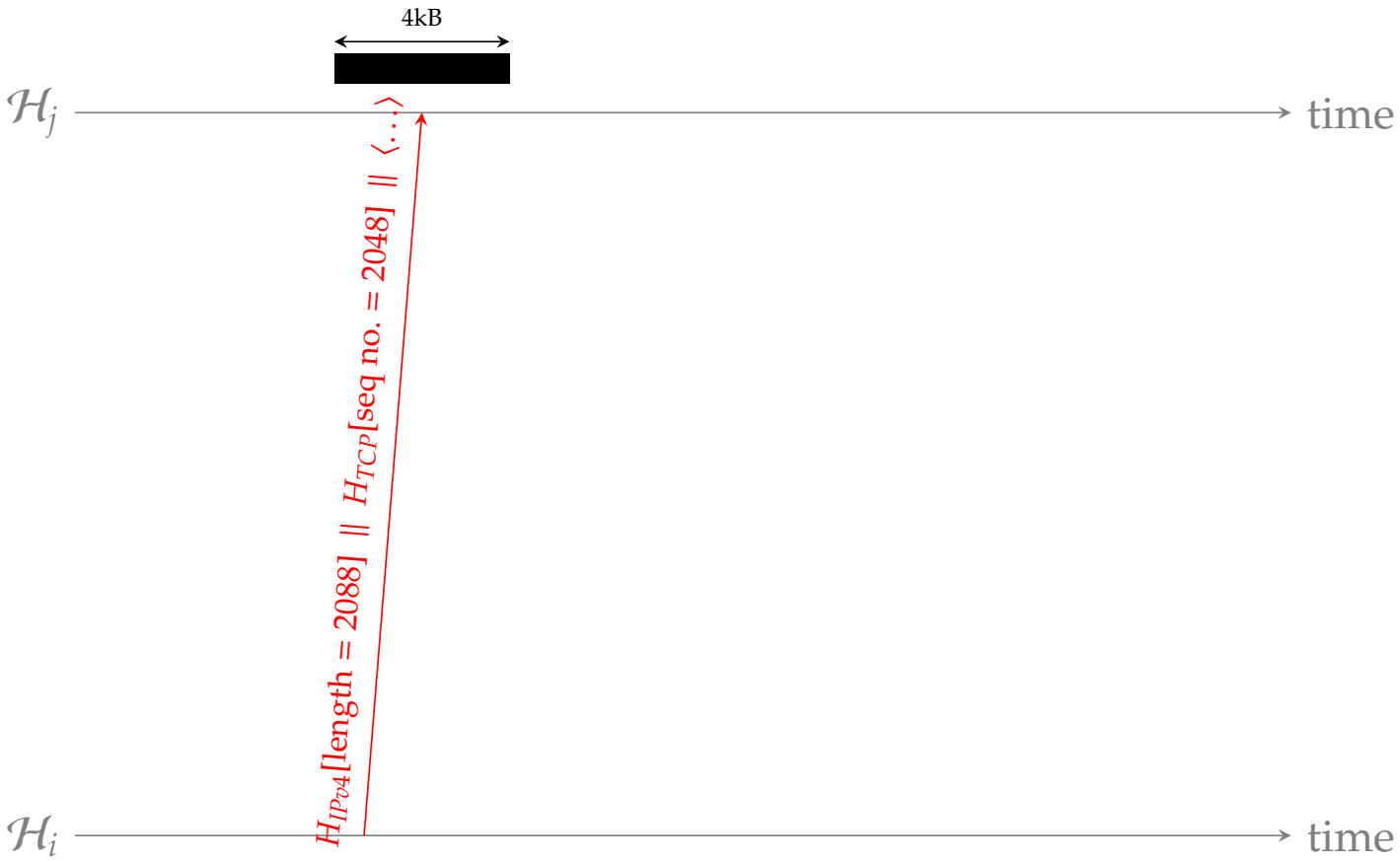
- Example: imagine the destination has a 4kB buffer.



- Example: imagine the destination has a 4kB buffer.

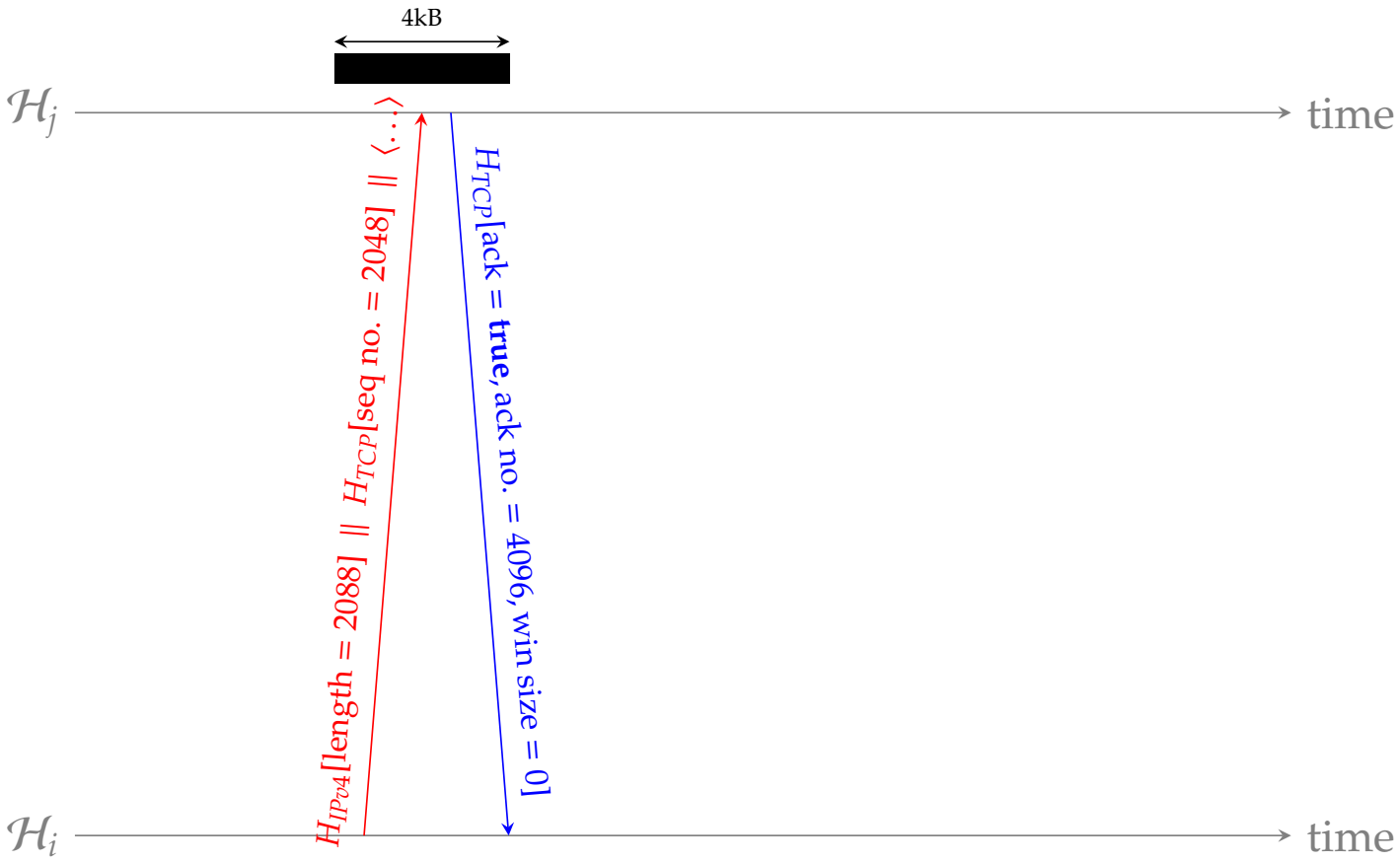
TCP Improvements (5)
Sliding-window ARQ ⇝ flow control

- Example: imagine the destination has a 4kB buffer.

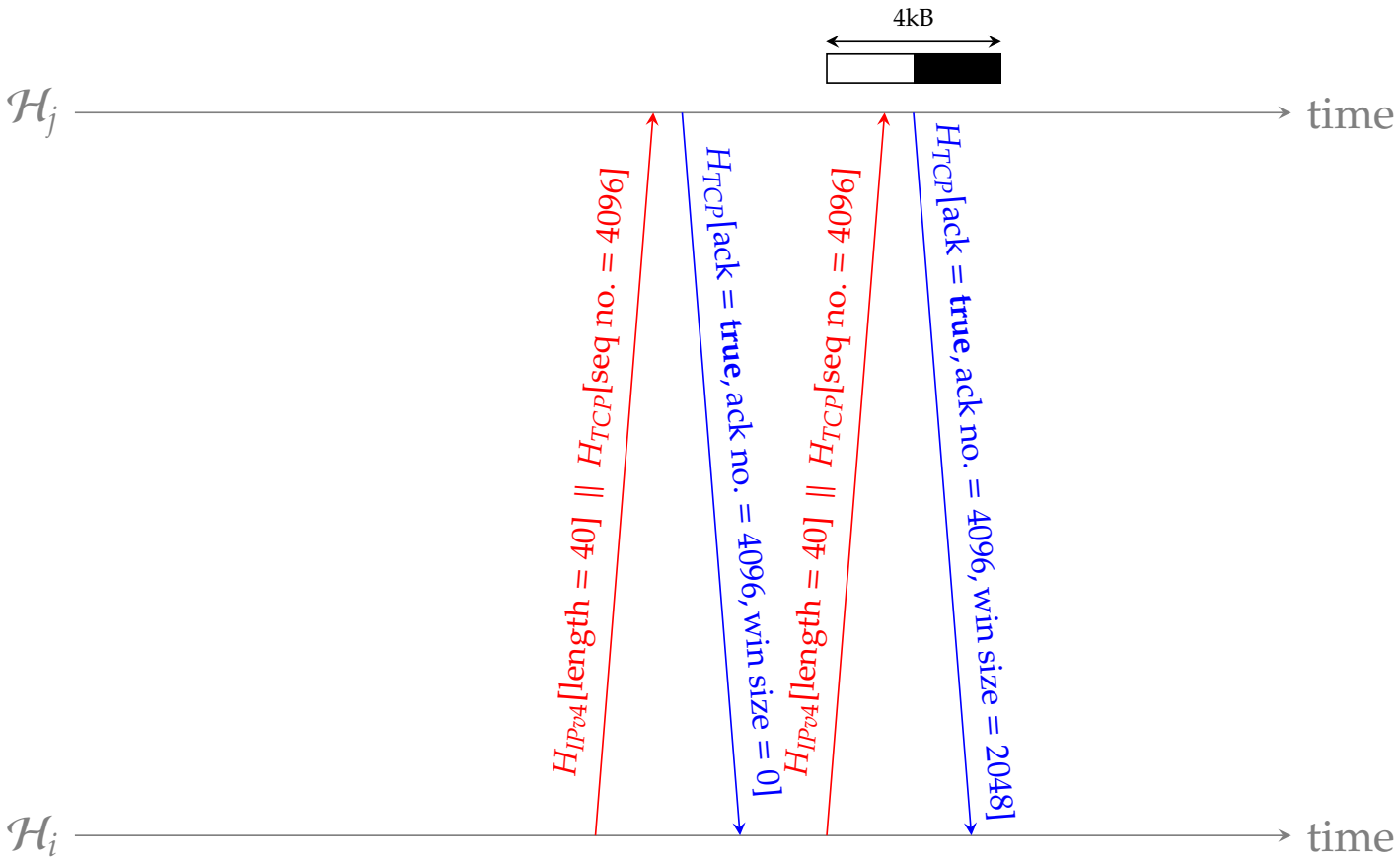


TCP Improvements (5)
Sliding-window ARQ ⇝ flow control

- Example: imagine the destination has a 4kB buffer.



- Example: imagine the destination has a 4kB buffer.

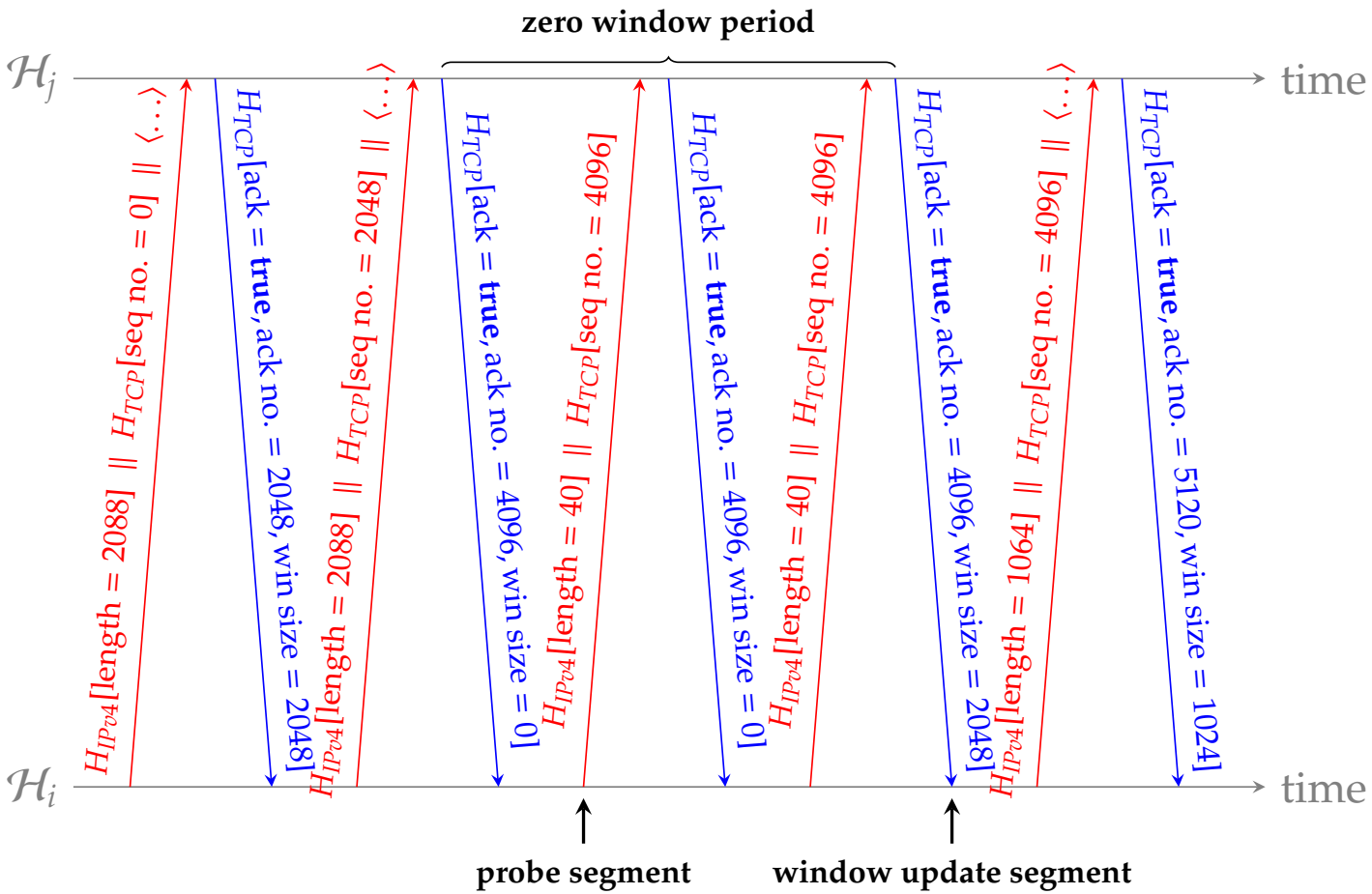


TCP Improvements (5)
Sliding-window ARQ ⇝ flow control

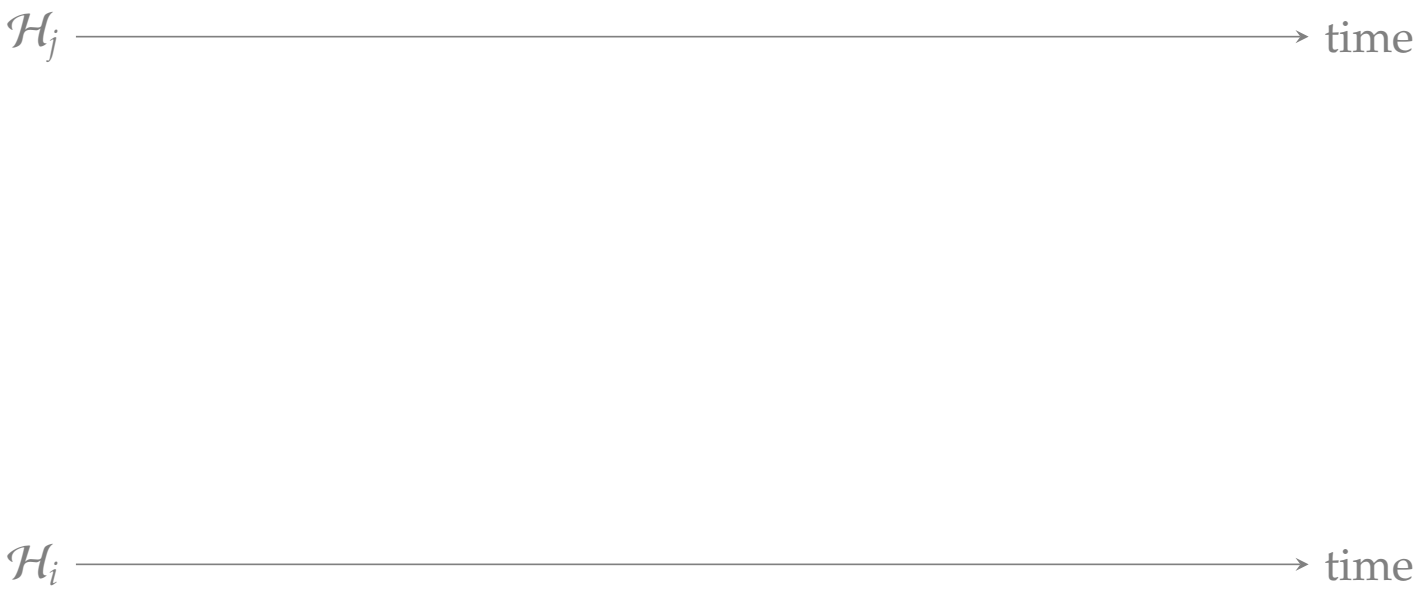
- Example: imagine the destination has a 4kB buffer.

- Example: imagine the destination has a 4kB buffer.

- Example: imagine the destination has a 4kB buffer.



TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



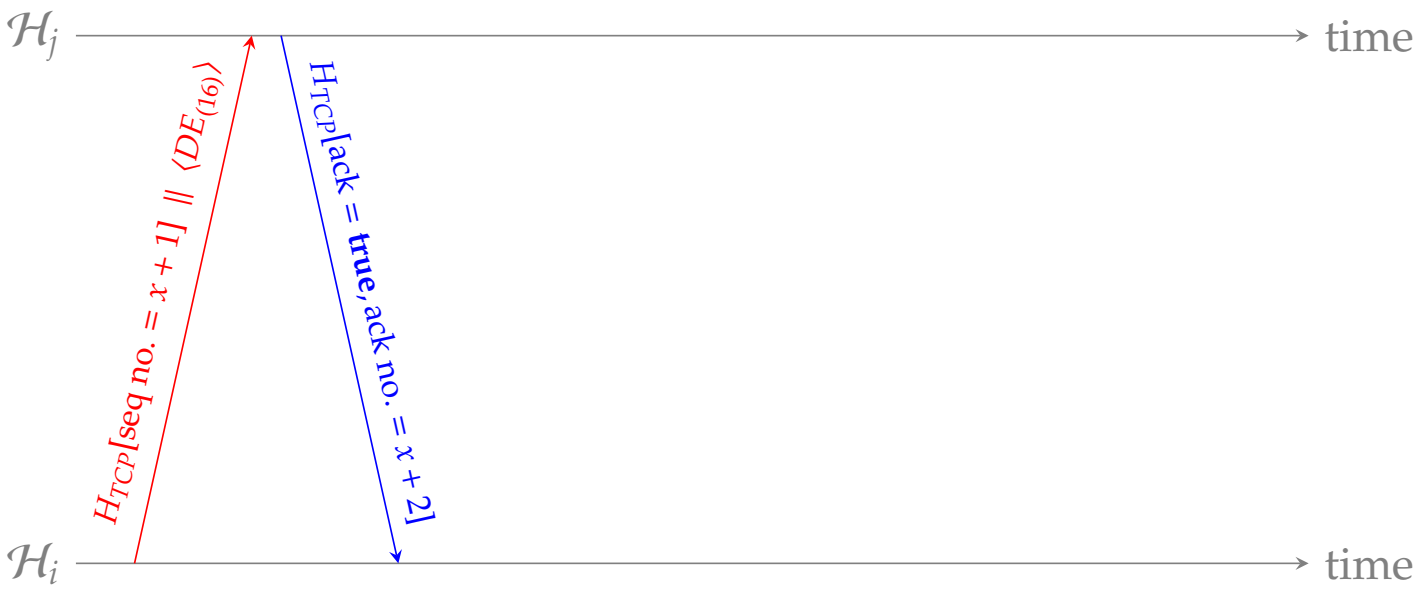
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP Improvements (6)

Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

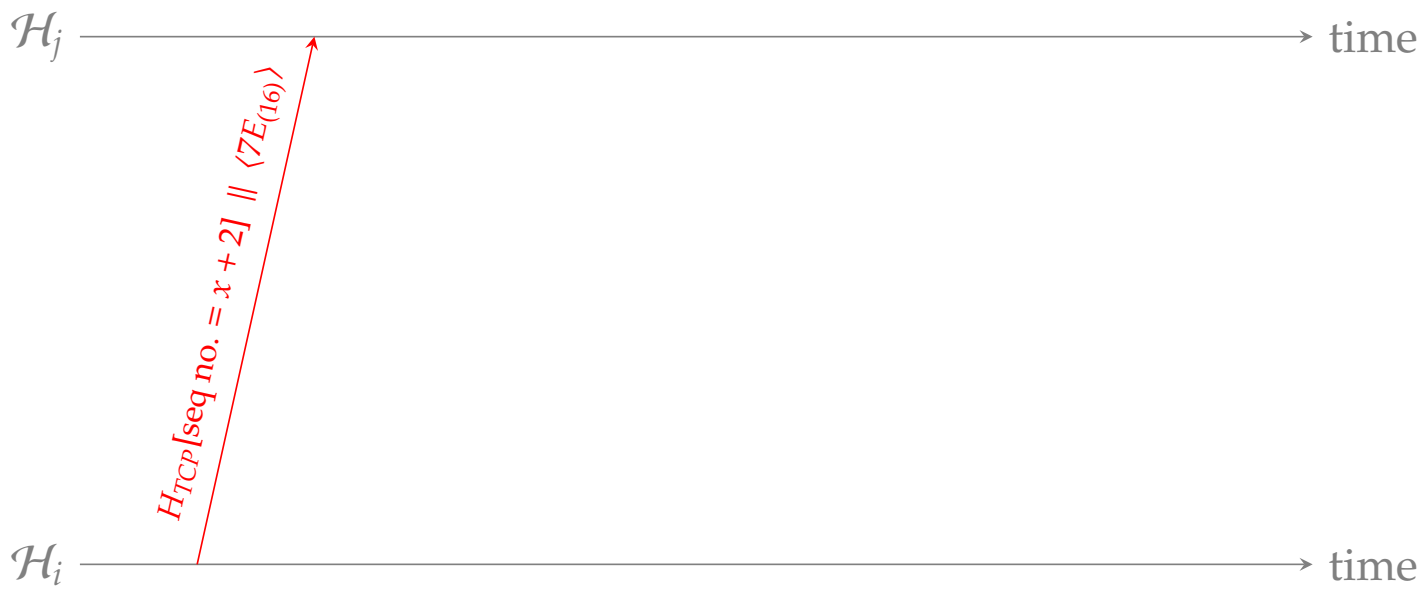
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



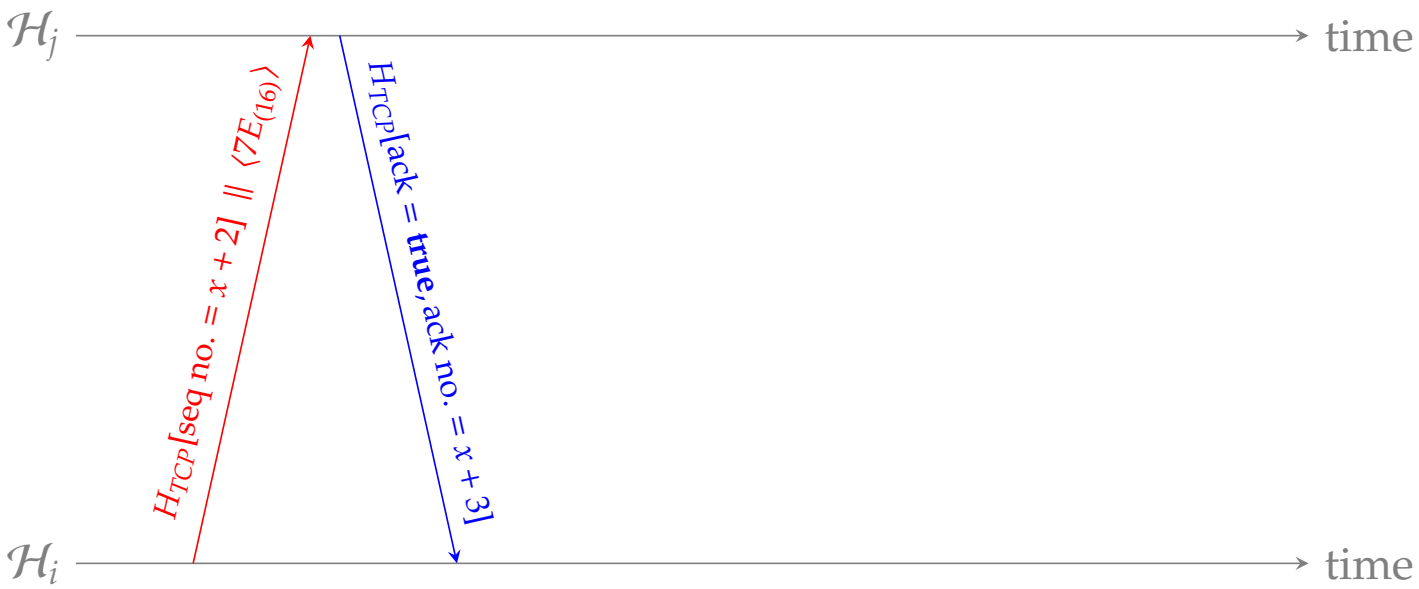
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

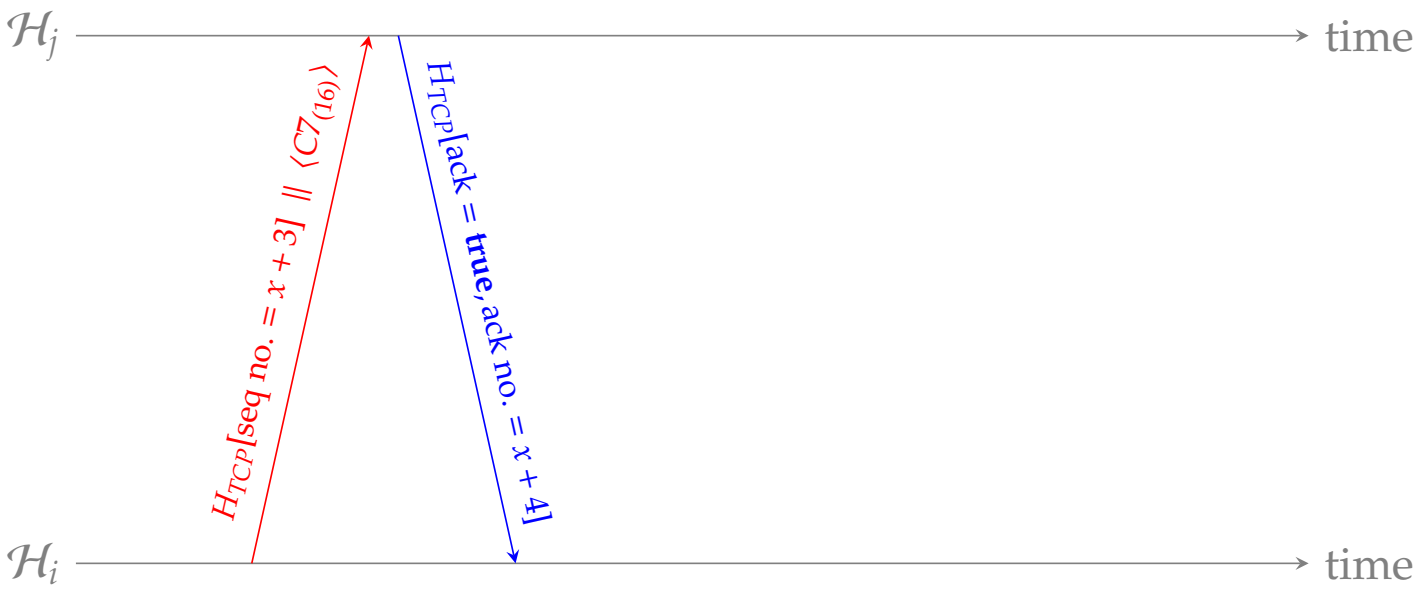


- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs

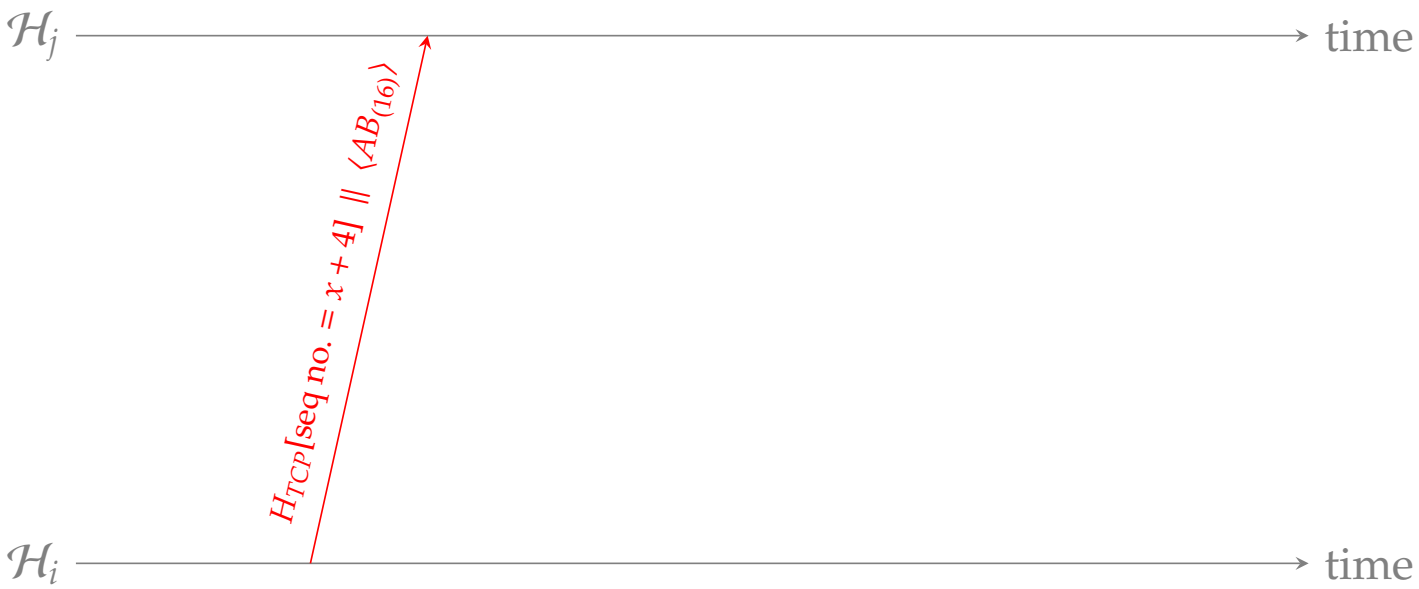
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



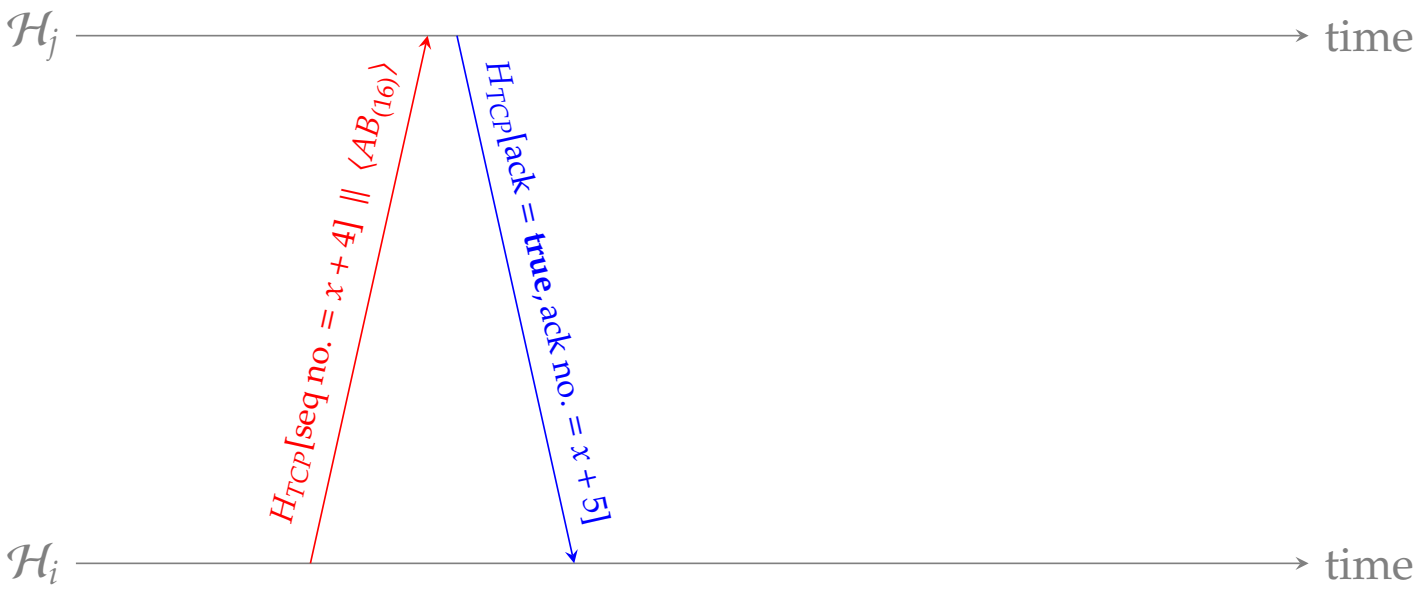
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



- Problem: naively, we send one ACK per segment (whether piggybacked or not).

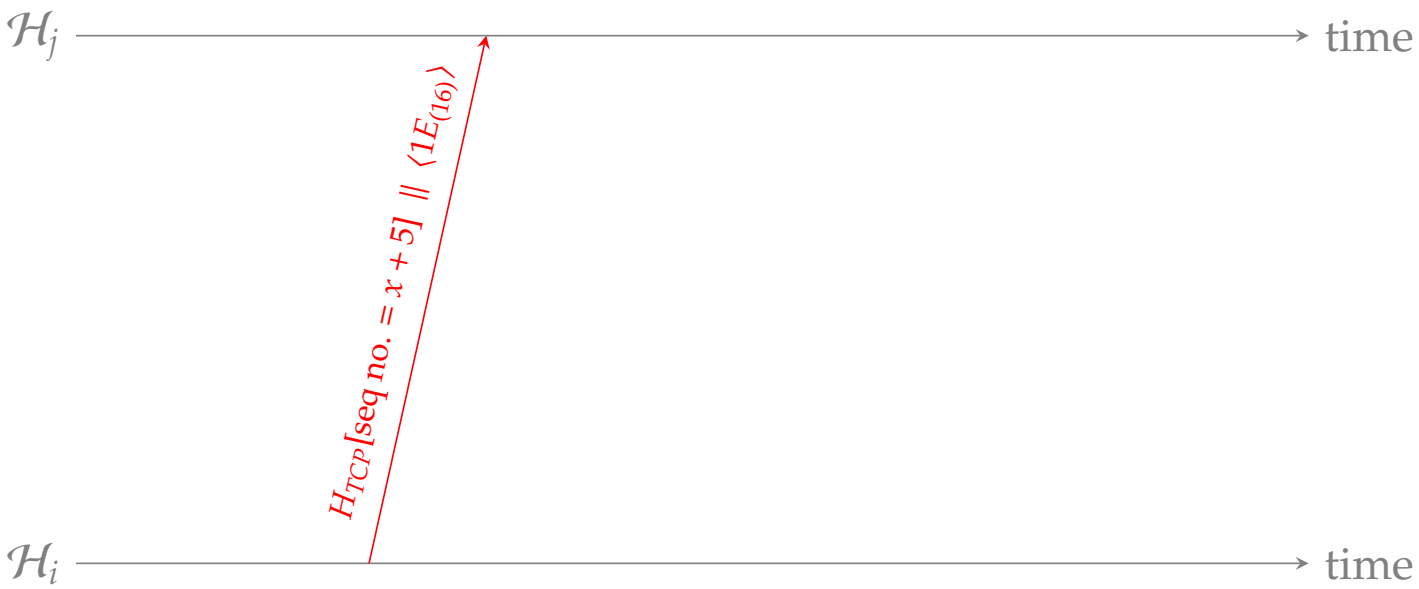
TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



- Problem: naively, we send one ACK per segment (whether piggybacked or not).

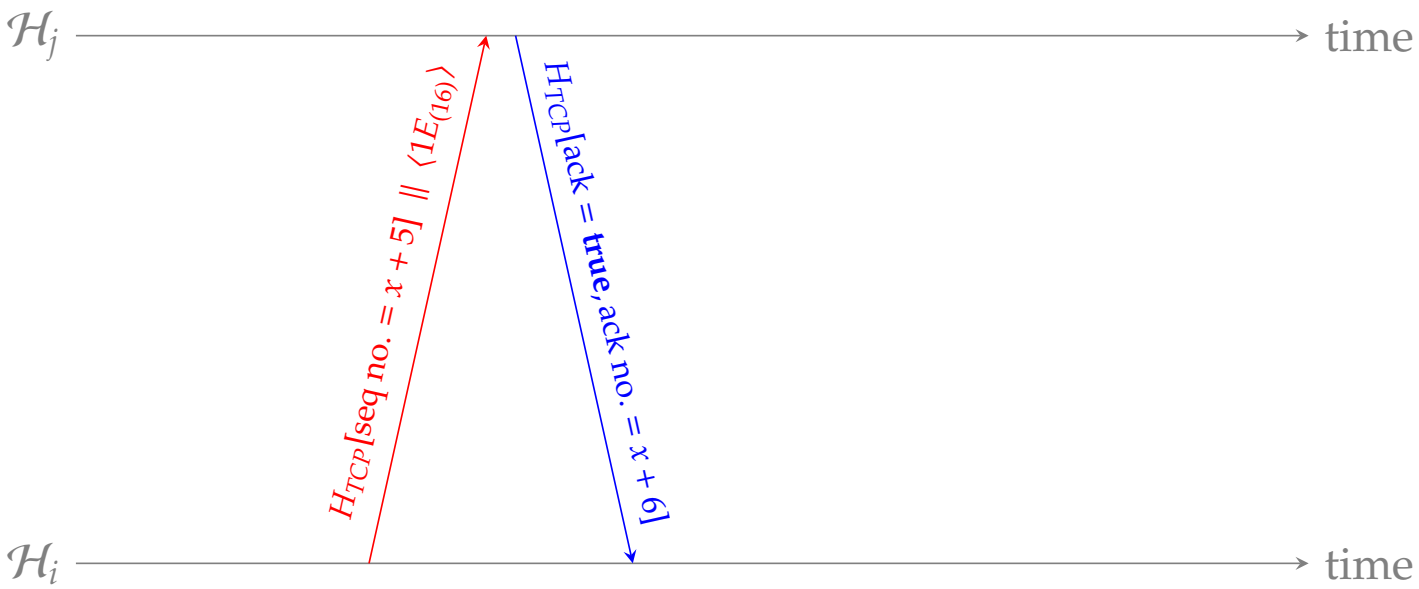
TCP Improvements (6)

Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



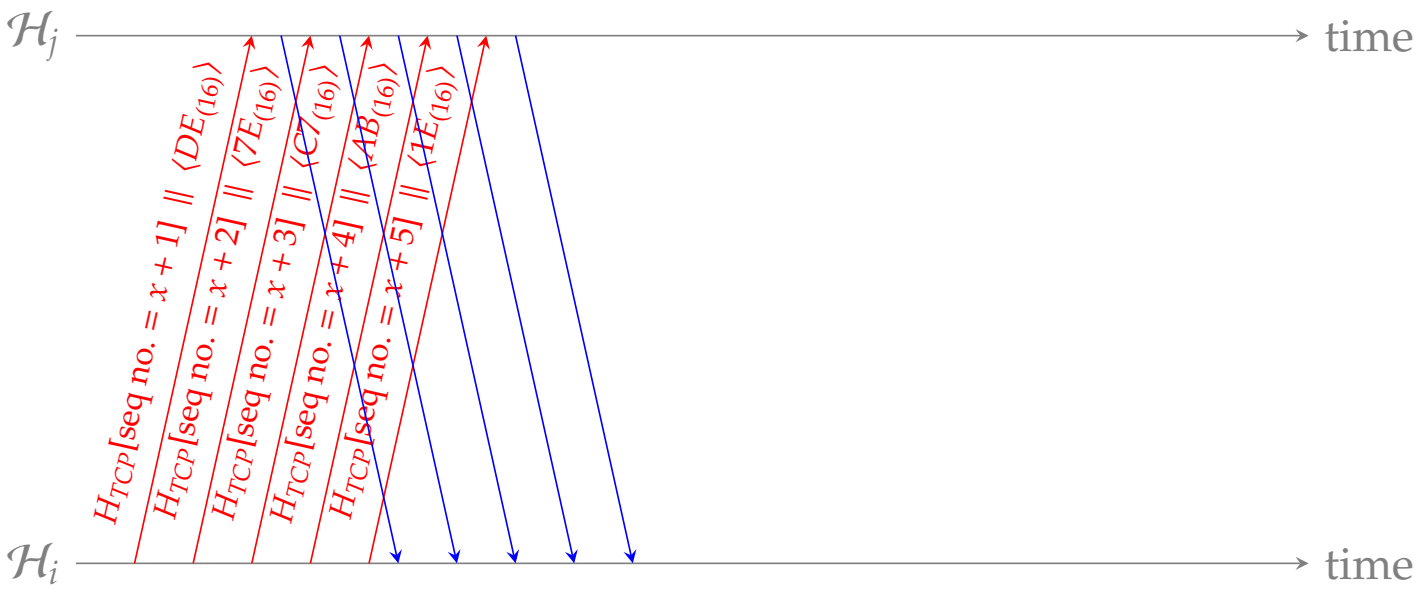
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



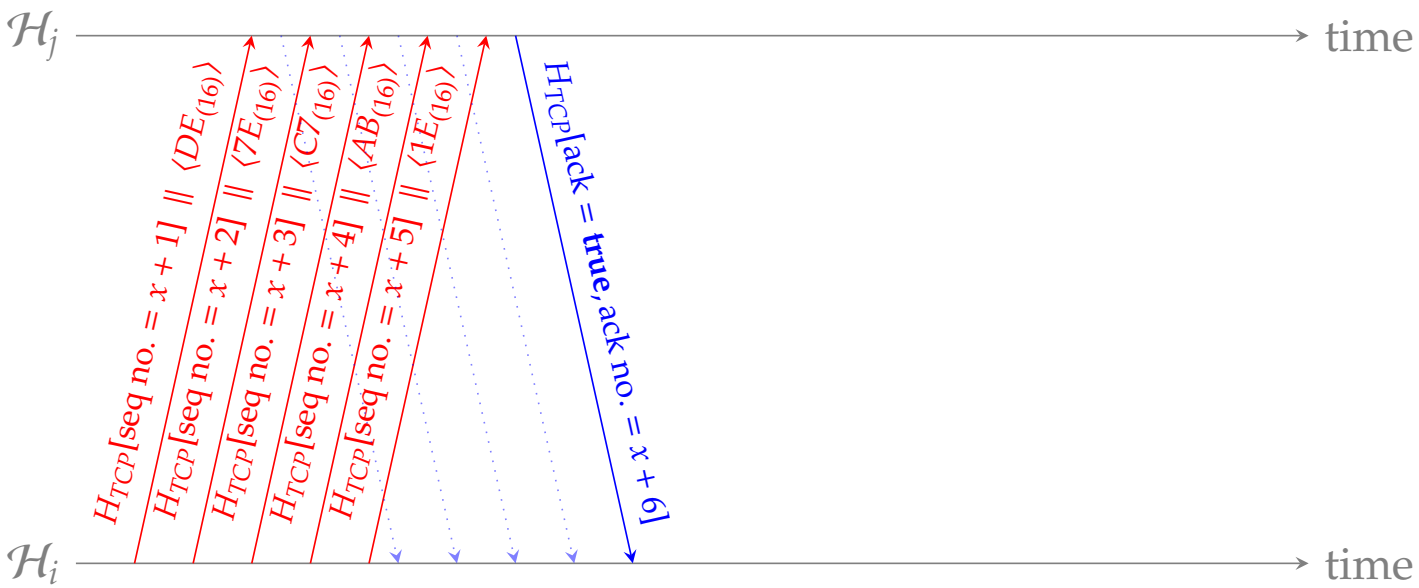
- Problem: naively, we send one ACK per segment (whether piggybacked or not).

TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



- Problem: naively, we send one ACK per segment (whether piggybacked or not).

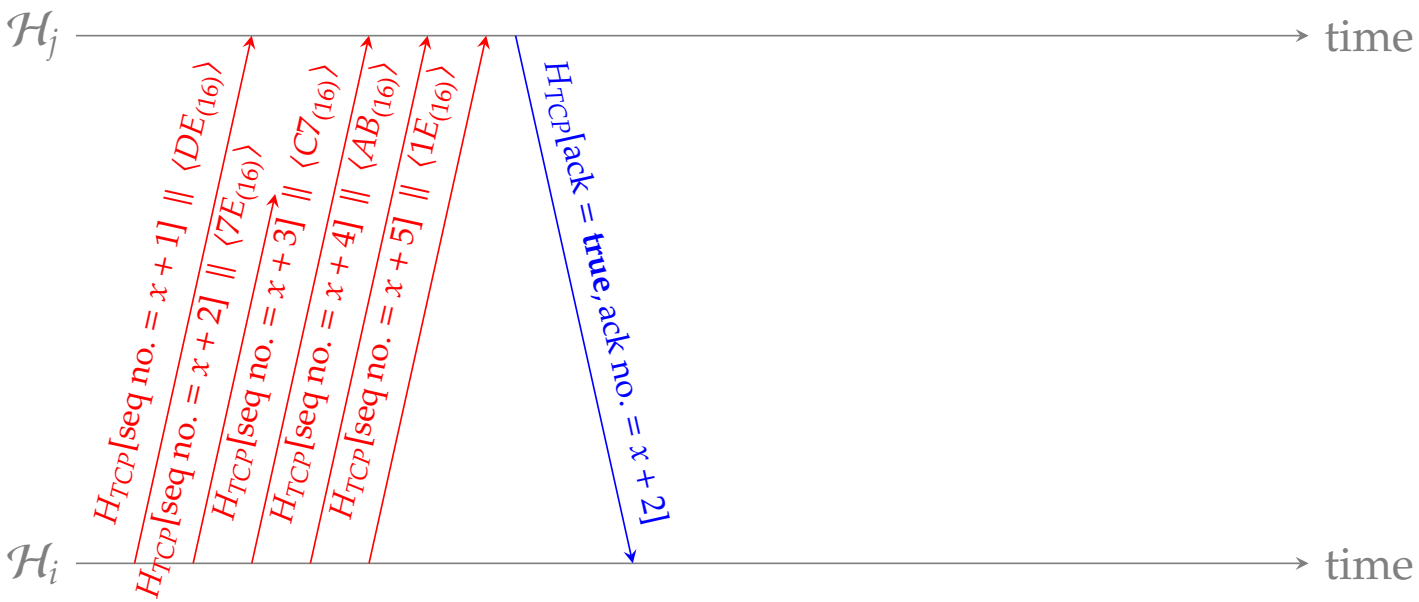
TCP Improvements (6)
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



- Problem: naively, we send one ACK per segment (whether piggybacked or not).
- Solution: allow a **cumulative ACK**, that
  - explicitly ACKs a segment, *and*
  - implicitly ACKs previous segments.

TCP Improvements (6)

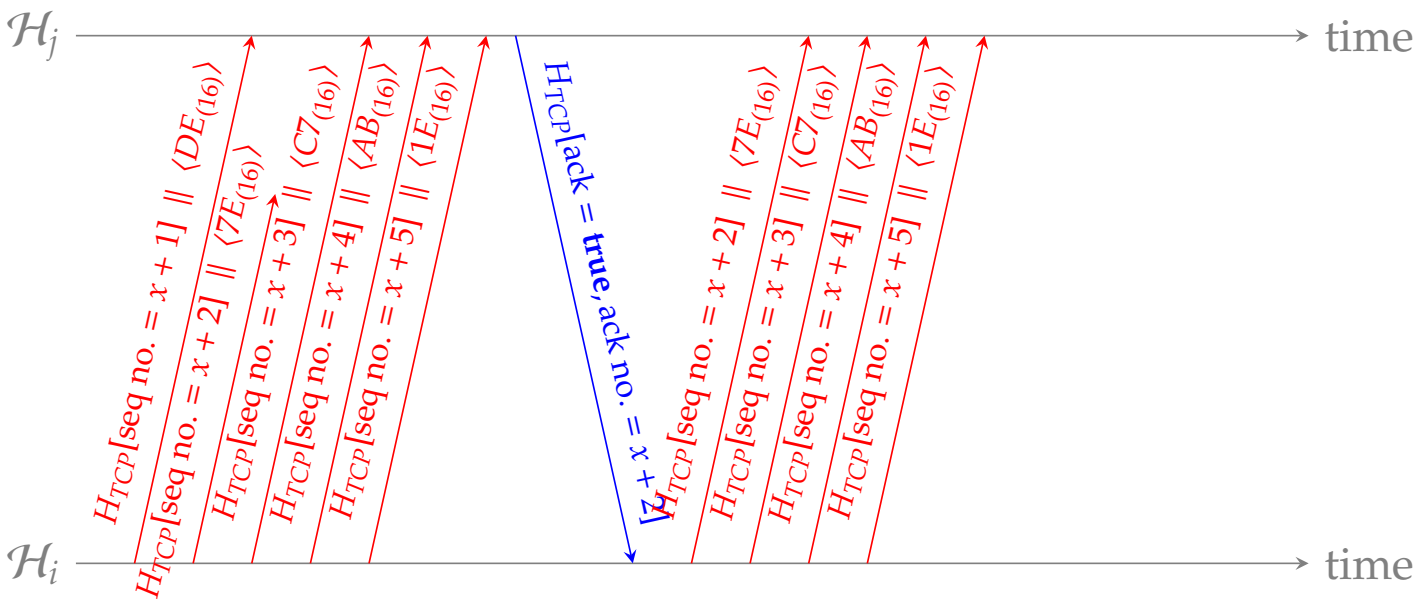
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



- Problem: can't cumulatively ACK *later* segments even if valid.

TCP Improvements (6)

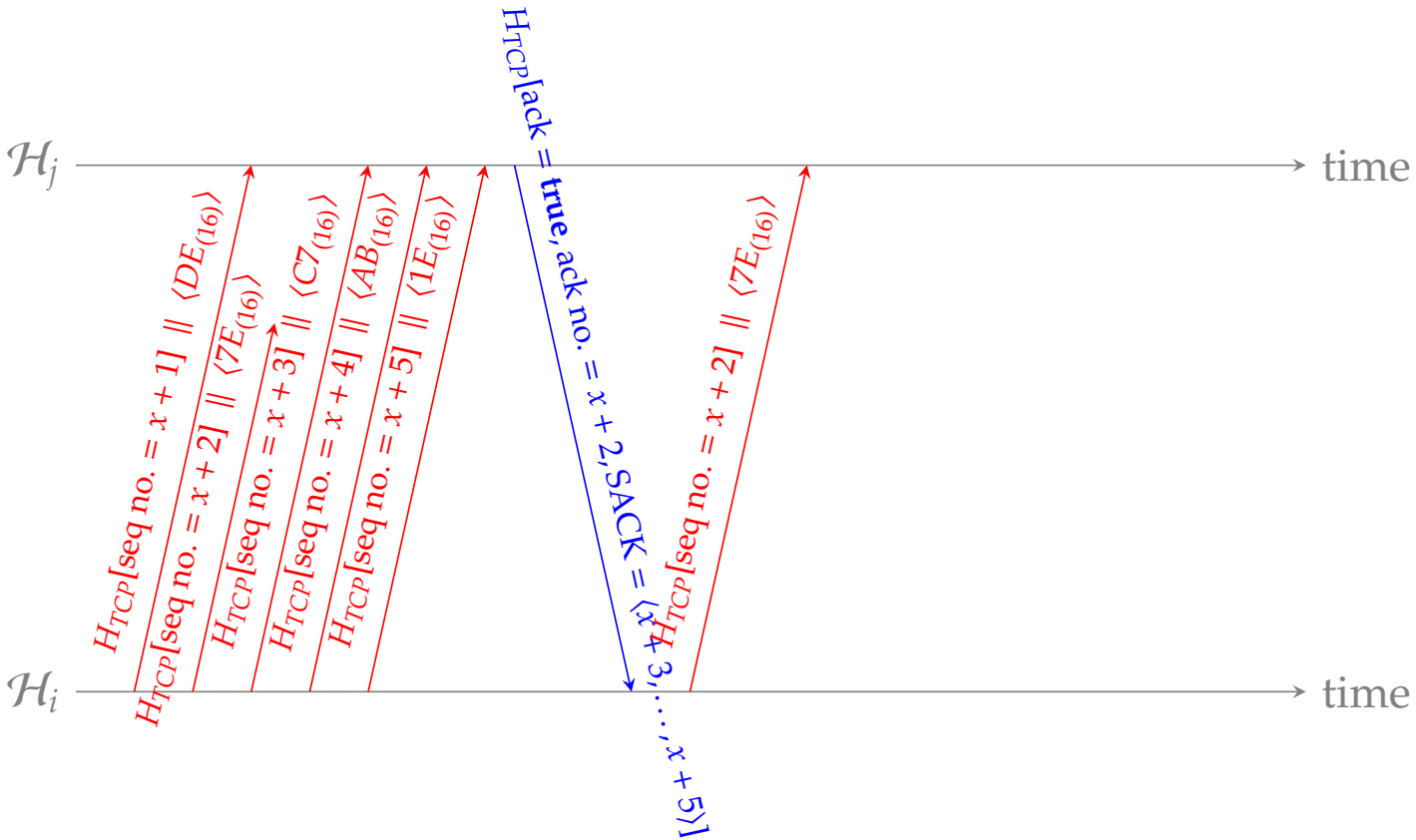
Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



- Problem: can't cumulatively ACK *later* segments even if valid.

TCP Improvements (6)

Sliding-window ARQ $\rightsquigarrow$ cumulative and selective ACKs



- Problem: can't cumulatively ACK *later* segments even if valid.
- Solution: allow a **selective ACK** [7], that
  - acts as (cumulative) ACK for contiguous segment(s), *and*
  - "hint" at other discontiguous segment(s).

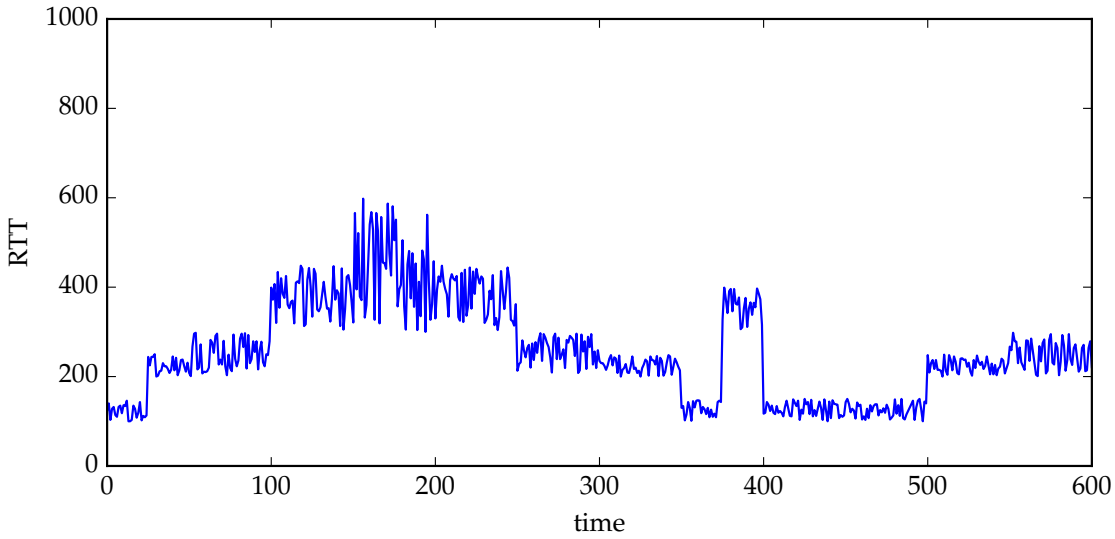
- Problem: to implement any ARQ-based scheme we need to select τ, but
  - too *small* a τ means we provoke many spurious retransmissions, and
  - too *large* a τ means we under-utilise the available bandwidth,

  i.e., τ relates to the **Goldilocks principle** [1]: we need it to be "just right" ...

1. for a LAN this is easy
   - RTT is typically small,
   - RTT has low variation,

   but
2. for an inter-network this is not easy
   - RTT is typically large,
   - RTT has high variation.

TCP Improvements (11)
Adaptive retransmission



- Note that:
  - the baseline RTT of ~ 100μs relates to the transmission and propagation delay, and
  - the noise in RTT samples comes from sources such as variation in routing and the load on hosts and the network.

- Solution: use **adaptive retransmission** [9, Section 2].
  1. Let

     R_t denote the measured RTT at time t
     V_t denote the predicted RTT variance at time t
     S_t denote the predicted RTT at time t

  2. Update the prediction via a moving average, i.e.,

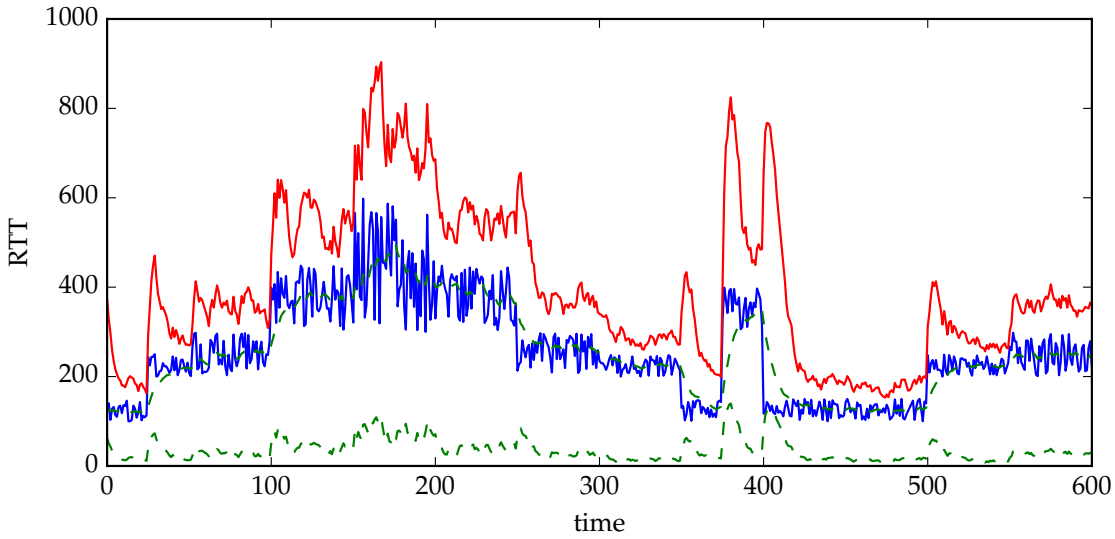
$$\begin{aligned} V_{t+1} &= (1-\beta)\cdot V_t + \beta \cdot |R_t - S_t| \\ S_{t+1} &= (1-\alpha)\cdot S_t + \alpha \cdot R_t \end{aligned}$$

     where $\alpha = \frac{1}{8}$ and $\beta = \frac{1}{4}$.
  3. Set τ_t (i.e., the value of τ at at time t) to

$$\tau_t = S_t + K \cdot V_t$$

     where $K = 4$.

TCP Improvements (13)
Adaptive retransmission



Conclusions

- Take away points:
  - The transport layer interfaces applications with the network ...
  - ... *it* must (and does) cope with numerous demands, e.g.,
    - clean interface vs. diverse, complex network, and
    - efficiency vs. unreliable, unpredictable network.
  - This means TCP is complicated (UDP less so), in that
    - it supports a *lot* of functionality,
    - it evolves over time, sometimes retaining mechanisms based on assumptions or problems that no longer hold, meaning
    - interaction between mechanisms is sometimes hard to predict (and so sometimes undesirable, cf. Nagle's Algorithm vs. delayed ACKs [14]).

Additional Reading

- *Wikipedia: Transport layer*. URL: http://en.wikipedia.org/wiki/Transport_layer.
- *Wikipedia: Transmission Control Protocol*. URL: http://en.wikipedia.org/wiki/Transmission_Control_Protocol.
- *Wikipedia: User Datagram Protocol*. URL: http://en.wikipedia.org/wiki/User_Datagram_Protocol.
- W. Stallings. "Chapter 23: Transport protocols". In: *Data and Computer Communications*. 9th ed. Pearson, 2010.

References

[1] *Wikipedia: Goldilocks principle*. URL: http://en.wikipedia.org/wiki/Goldilocks_principle (see p. 144).

[2] *Wikipedia: Transmission Control Protocol*. URL: http://en.wikipedia.org/wiki/Transmission_Control_Protocol (see p. 149).

[3] *Wikipedia: Transport layer*. URL: http://en.wikipedia.org/wiki/Transport_layer (see p. 149).

[4] *Wikipedia: User Datagram Protocol*. URL: http://en.wikipedia.org/wiki/User_Datagram_Protocol (see p. 149).

[5] W. Stallings. "Chapter 23: Transport protocols". In: *Data and Computer Communications*. 9th ed. Pearson, 2010 (see p. 149).

[6] V. Jacobson, R. Braden, and D. Borman. *TCP Extensions for High Performance*. Internet Engineering Task Force (IETF) Request for Comments (RFC) 1323. 1992. URL: http://tools.ietf.org/html/rfc1323.

[7] M. Mathis et al. *TCP Selective Acknowledgment Options*. Internet Engineering Task Force (IETF) Request for Comments (RFC) 2018. 1996. URL: http://tools.ietf.org/html/rfc2018 (see pp. 141–143).

[8] J. Nagle. *Congestion Control in IP/TCP Internetwork*. Internet Engineering Task Force (IETF) Request for Comments (RFC) 896. 1984. URL: http://tools.ietf.org/html/rfc896.

[9] V. Paxson and M. Allman. *Computing TCP's Retransmission Time*. Internet Engineering Task Force (IETF) Request for Comments (RFC) 2988. 2000. URL: http://tools.ietf.org/html/rfc2988 (see p. 146).

[10] J. Postel. *Transmision Control Protocol*. Internet Engineering Task Force (IETF) Request for Comments (RFC) 793. 1981. URL: http://tools.ietf.org/html/rfc793 (see pp. 4, 33–37).

[11] J. Postel. *User Datagram Protocol*. Internet Engineering Task Force (IETF) Request for Comments (RFC) 768. 1980. URL: http://tools.ietf.org/html/rfc768 (see p. 4).

[12] 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. URL: http://tools.ietf.org/html/rfc3168 (see pp. 33–37).

[13] 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. URL: http://tools.ietf.org/html/rfc3540 (see pp. 33–37).

[14] G. Minshall et al. "Application performance pitfalls and TCP's Nagle algorithm". In: *ACM SIGMETRICS Performance Evaluation Review* 27.4 (2000), pp. 36–44 (see p. 148).