Concurrent Computing (Computer Networks)

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨Daniel.Page@bristol.ac.uk⟩

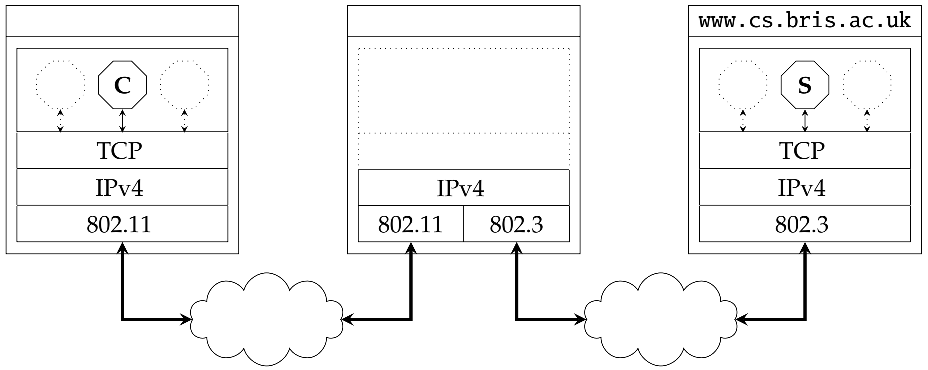
March 14, 2016

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

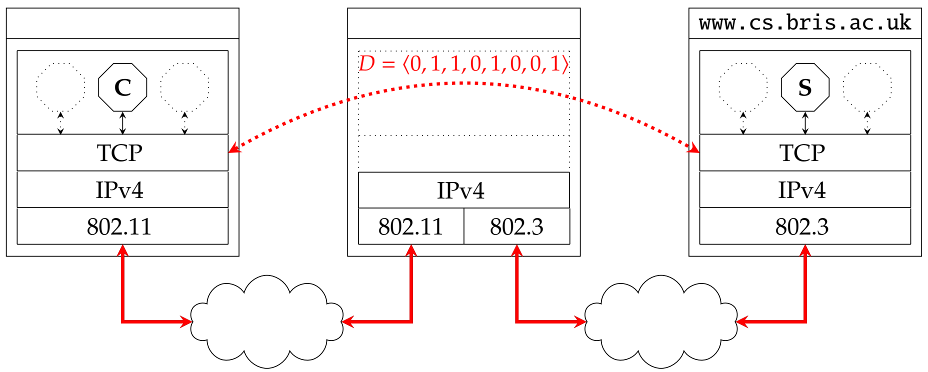


- Goal: investigate the **transport layer** e.g.,
  1. connection management,
  2. error, flow and congestion control

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

Notes:

COMS20001 lecture: week #22

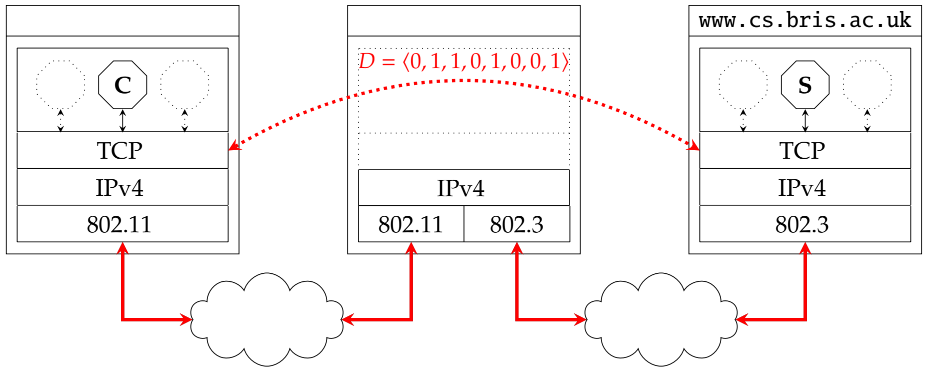


$F = H_{802.11} \parallel H_{IPv4} \parallel H_{TCP} \parallel \langle 0,1,1,0,1,0,0,1 \rangle \parallel T_{802.11}$

- 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).

Notes:



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

- 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) ...

Notes:

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

Definition (datagram model)

For instance, **User Datagram Protocol (UDP)** [7] 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)** [8] 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. $\text{TCP} \gg \text{UDP} \simeq \text{IP} + \epsilon$.

Notes:

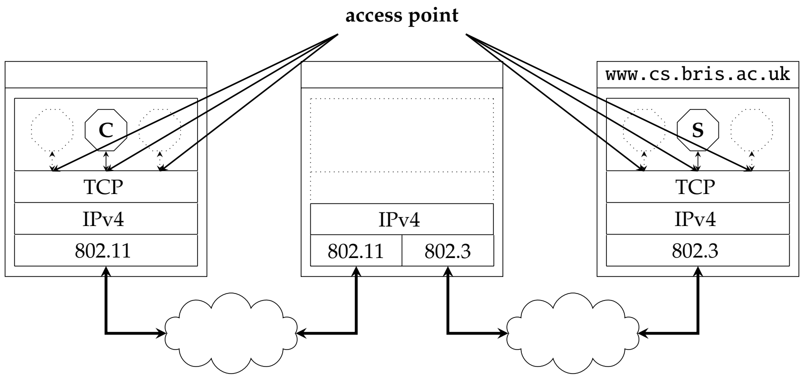
- The terse relationship at the bottom is meant to capture the idea that TCP provides a *lot* more functionality than UDP, which is basically just a thin layer over IP (providing ports, and not a lot else). The one thing it *does* provide is multiplexing, namely the ability for more than one application to utilise the single network stack via ports.
  Put another way, applications have a choice: they get *everything* by opting for TCP, or more or less *nothing* by selecting UDP!
- Strictly speaking, we should note UDP and TCP are *implementations* of the datagram and stream models (rather than the models *themselves*). They are not the *only* possible implementations, for example, just the ones we happen use within TCP/IP.
- The connection-based stream model means TCP is essentially using a virtual circuit. As already discussed, it's also useful to consider analogies st.
  - the datagram (UDP) model is like the postal service, and
  - the stream (TCP) model is like the telephone network.
- The term reliability can be misunderstood: in *both* models, the issue of errors (e.g., corrupted or lost packets) *still* exists: using TCP does not magically produce an error-free network!
  A better way to distinguish UDP and TCP is to say the former exposes errors to the application (meaning *it* has to resolve them somehow), whereas the latter hides them (automatically resolving them, modulo any conditions where it simply cannot do so). Whereas segments transmitted using UDP can be lost, duplicated or received out-of-order, TCP automatically copes: it automatically deals with detection and retransmission of any lost segments, detecting and discarding duplicate segments, and buffering segments to ensure they are delivered in-order.
- The terms might not convey the difference, but it's important to view flow and congestion control as distinct: specifically,
  - flow control deals with controlling source vs. destination, e.g., so the destination can cope with rate source is transmitting at, whereas
  - congestion control deals with controlling source vs. network, e.g., so the network can cope with rate source is transmitting at.

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).



Notes:

- The use of ports basically allows multiple applications to interface with one network stack (via the transport layer): the network layer will multiplex communication via the correct transport protocol, and the transport layer then passes data to and from the application layer.
  A reasonable analogy is that a port is like a mailbox that any given application can lease (via the OS): data is passed to and from the application and network via this mailbox in some sense.
- The fact that a socket pair identifies one connection means we can only have one (active) connection between two given hosts via two given ports. Put another way, the connection described by

  ((137.222.102.12, TCP, 49152), (137.222.10.80, TCP, 80))

  is between a web-browser on a host whose IP address is 137.222.102.12 and a web-server whose IP address is 137.222.10.80. The former is using the ephemeral port 49152, and the latter the well-known (for HTTP) port 80. It is not possible for a second connection to exist with the same description. On the other hand, the web-browser *could* form another connection described by

  ((137.222.102.12, TCP, 49153), (137.222.10.80, TCP, 80))

  which is distinct: even though the web-server port is the same, this is allowable since the connection is still identifiable.
- IANA maintain a definitive set of assigned field values, e.g., for well-known port numbers, at

  `http://www.iana.org/assignments/port-numbers`

  They make a coarser-grained distinction between so-called system ports (0 to 1023), user ports (1024 to 49151) and dynamic or private ports (49152 to 65535).

Concepts (1)

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).

Notes:

- The use of ports basically allows multiple applications to interface with one network stack (via the transport layer): the network layer will multiplex communication via the correct transport protocol, and the transport layer then passes data to and from the application layer.
  A reasonable analogy is that a port is like a mailbox that any given application can lease (via the OS): data is passed to and from the application and network via this mailbox in some sense.
- The fact that a socket pair identifies one connection means we can only have one (active) connection between two given hosts via two given ports. Put another way, the connection described by

  ((137.222.102.12, TCP, 49152), (137.222.10.80, TCP, 80))

  is between a web-browser on a host whose IP address is 137.222.102.12 and a web-server whose IP address is 137.222.10.80. The former is using the ephemeral port 49152, and the latter the well-known (for HTTP) port 80. It is not possible for a second connection to exist with the same description. On the other hand, the web-browser *could* form another connection described by

  ((137.222.102.12, TCP, 49153), (137.222.10.80, TCP, 80))

  which is distinct: even though the web-server port is the same, this is allowable since the connection is still identifiable.
- IANA maintain a definitive set of assigned field values, e.g., for well-known port numbers, at

  `http://www.iana.org/assignments/port-numbers`

  They make a coarser-grained distinction between so-called system ports (0 to 1023), user ports (1024 to 49151) and dynamic or private ports (49152 to 65535).

Concepts (2)

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** .

Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

Concepts (2)

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** .

Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

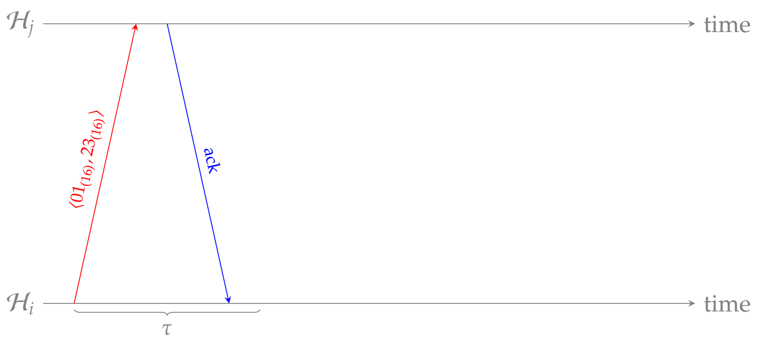
Concepts (2)

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

    iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

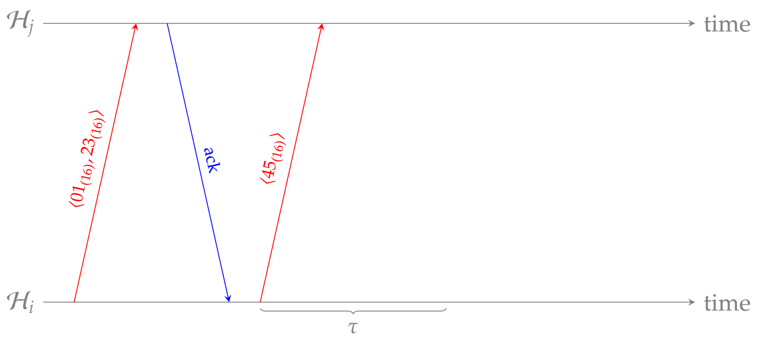
Concepts (2)

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

    iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

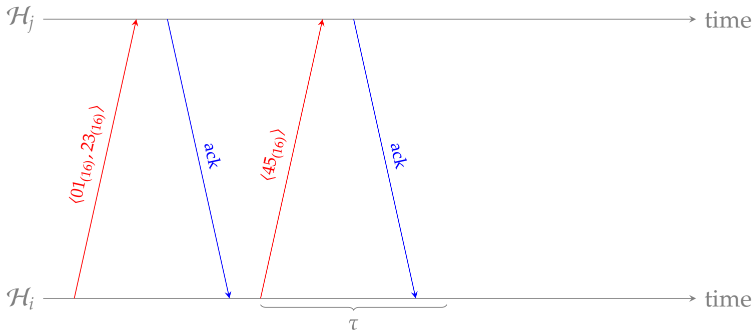
  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$"

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

Concepts (2)

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** .

Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$"

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

Daniel Page (Daniel.Page@bristol.ac.uk)
Concurrent Computing (Computer Networks) git # 3627080 @ 2016-03-11 University of BRISTOL

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** .

Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

    iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$"

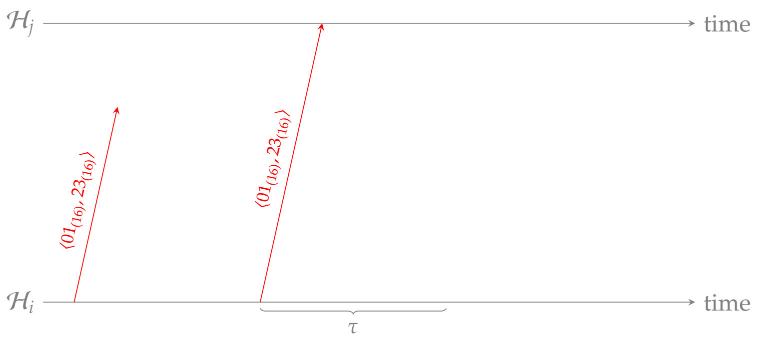
  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

    iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$"

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

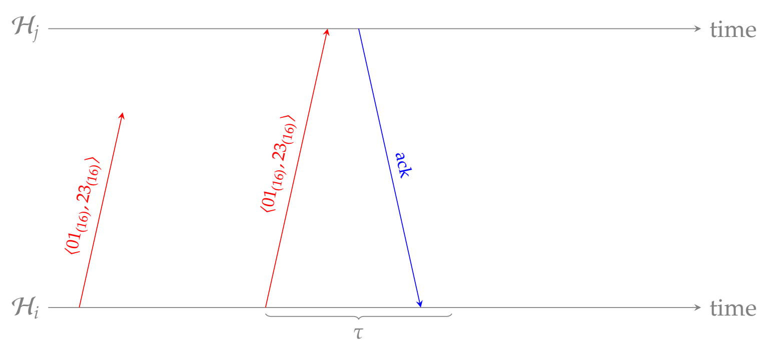
Concepts (2)

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

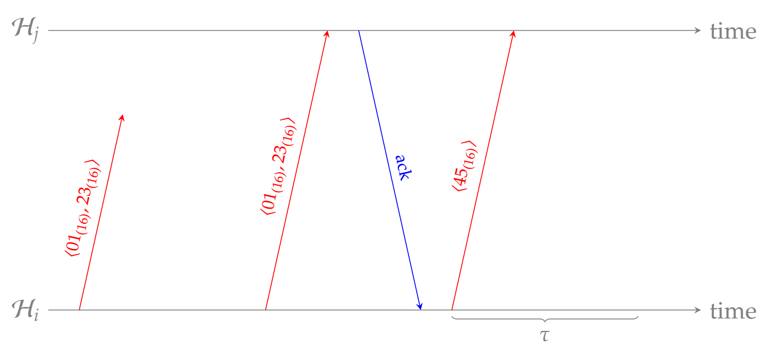
Concepts (2)

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

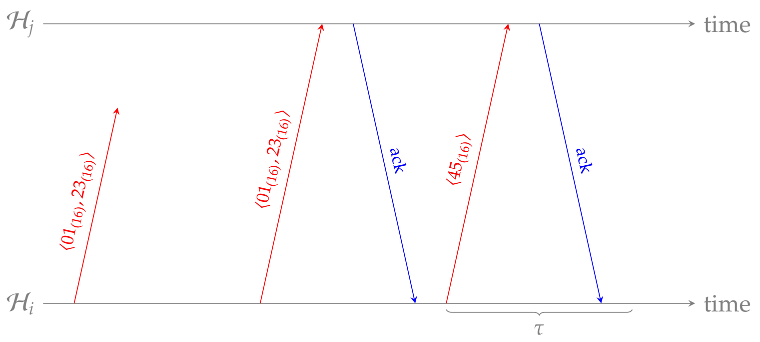
  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

    iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

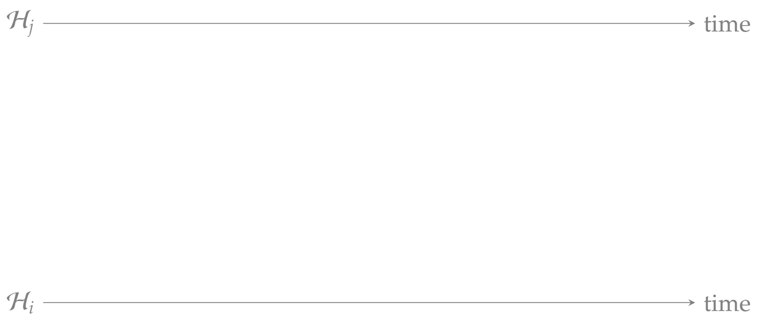
Concepts (2)

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

    iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

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** .

Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

    iff. we add a numbering scheme to disambiguate each step

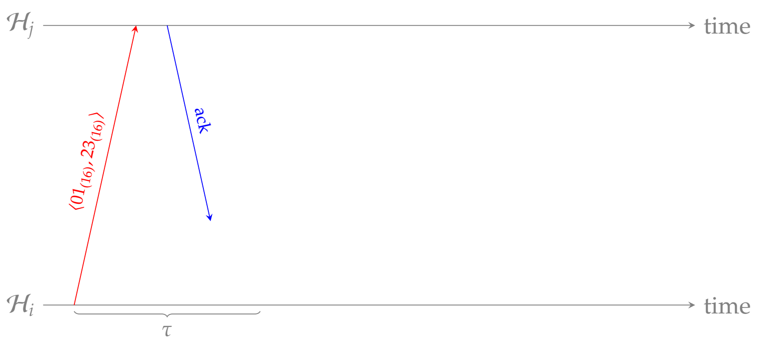
  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

Concepts (2)

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 .



Notes:

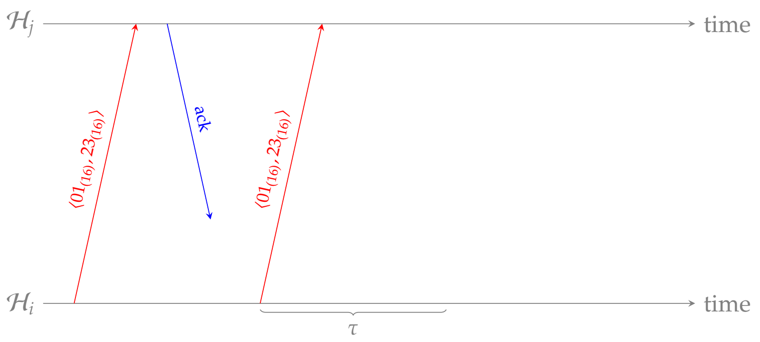
In order, the diagrams show we can detect when
a segment is successfully delivered, per associated ACK,
a segment is unsuccessfully delivered, due to the lack of associated ACK,
the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
when
a segment is duplicated, or
the ACK associated with a segment is delayed
iff. we add a numbering scheme to disambiguate each step
where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term ACK clocking to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
The sequence and ACK numbers are, in some sense, both examples of serial number: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the next one to have sequence number $x + 1$")
are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the sliding-window improvement.
Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

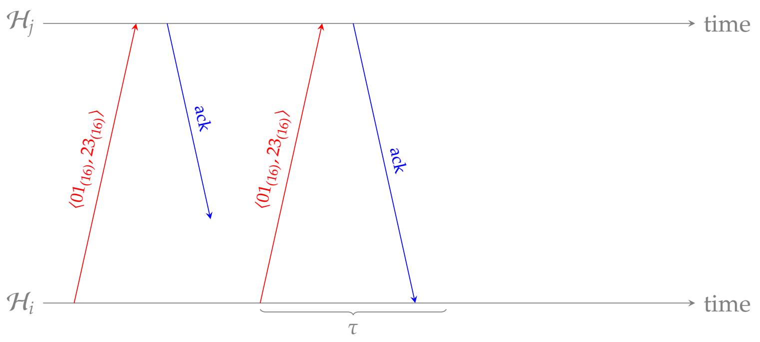
Concepts (2)

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

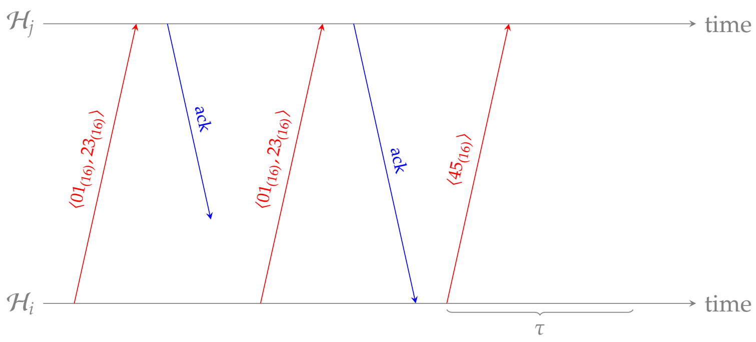
  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

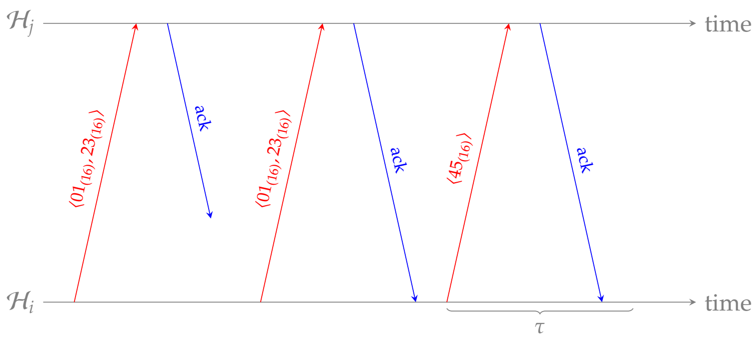
Concepts (2)

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

Concepts (2)

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** .

Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

    iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

Concepts (2)

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** .

Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

    iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

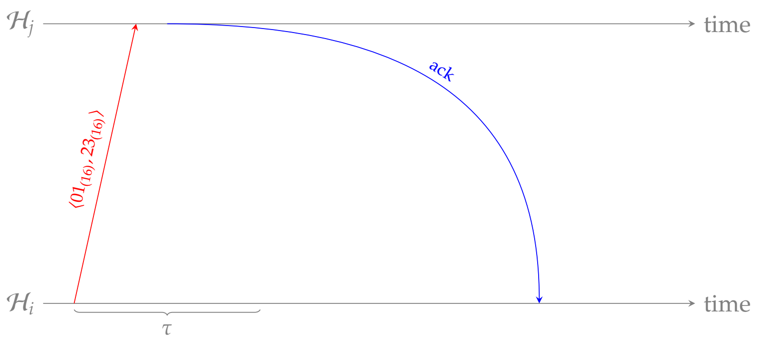
Concepts (2)

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

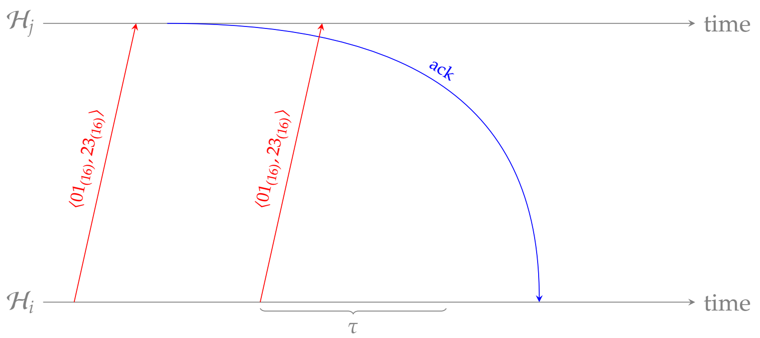
Concepts (2)

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

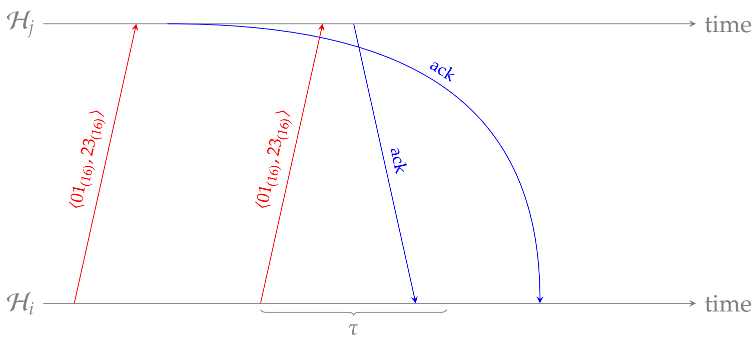
  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

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** .



Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

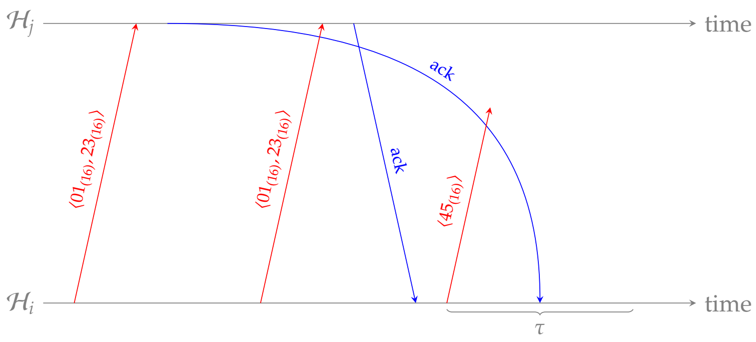
Concepts (2)

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** .



Daniel Page (Daniel.Page@bristol.ac.uk)
Concurrent Computing (Computer Networks) git # 3627080 @ 2016-03-11 University of BRISTOL

Notes:

- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

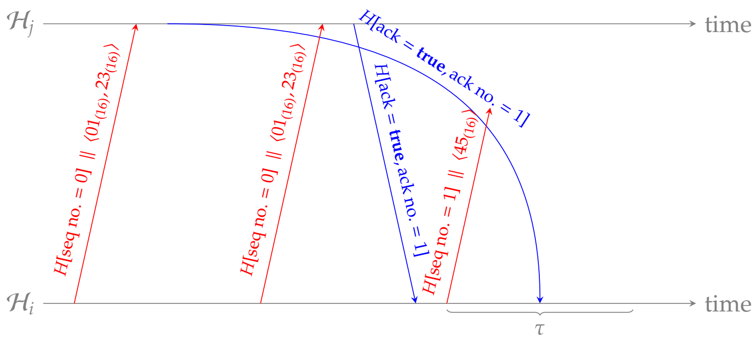
  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

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**).



Notes:

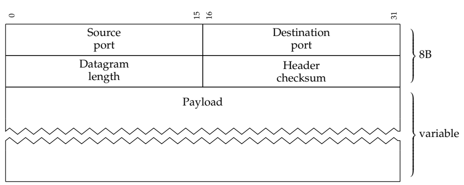
- In order, the diagrams show we can detect when
  - a segment is successfully delivered, per associated ACK,
  - a segment is unsuccessfully delivered, due to the lack of associated ACK,
  - the ACK associated with a segment is unsuccessfully delivered, due to the lack of said ACK, and
  - when
    - a segment is duplicated, or
    - the ACK associated with a segment is delayed

  iff. we add a numbering scheme to disambiguate each step

  where unsuccessful delivery might mean loss or corruption, and the delayed case also captures the idea of reordering (i.e., out-of-order delivery).
- In a sense, the ACKs are controlling when segments can be transmitted. It's common to use the term **ACK clocking** to describe this, because ACKs from the destination act as a clock (or synchronisation) signal wrt. transmission by the source.
- The sequence and ACK numbers are, in some sense, both examples of **serial number**: [6] includes a generalised explanation of how they must be dealt with. Why? It's more tricky than you'd think, because of issues such as wrap-around (e.g., from $2^n - 1$ to 0; ideally, we need to cope gracefully with such conditions.
- In a general setting, we have various options about how the sequence and ACK numbers are generated. For example,
  1. given a segment with sequence number x, we could have the associated ACK number be x as well, i.e., say "I've recieved a segment with sequence number x",
  2. given a segment with sequence number x, we could have the associated ACK number be $x + 1$, i.e., say "I've recieved a segment, and I expect the *next* one to have sequence number $x + 1$")

  are both valid options. The example opts for the latter, motivated by the fact that this is (more or less) what TCP does.
- The astute may already see some limitations. For example, the stop-and-wait protocol is not very efficient (since hosts and network are idle for lengthy periods imposed by the "wait" aspect). This issue is examined (a lot) more detail later, when we cover the **sliding-window** improvement.
  Intuitively, it is enough to recognise stop-and-wait as a special case where $w = 1$ (in the sense there is at most 1 outstanding, or un-ACK'ed segment): sliding-window generalises this to $w > 1$. A second, arguably more important issue is that we need to select the time-out threshold τ. Again, this is an aspect we (re)examine in more detail later.

Data Structure (**UDP datagram**)



Notes:

- You might prefer the original ASCII art from [7, Section 3.1]:

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

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

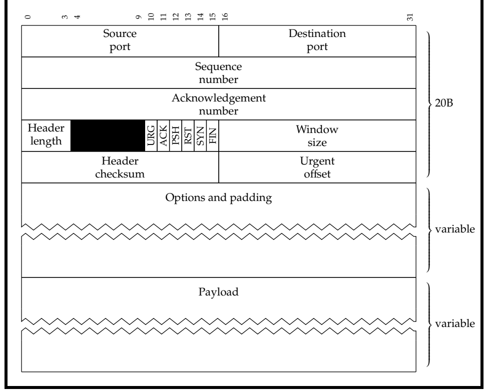
  plus various auxiliary actions, e.g., **connection reset**.

Notes:

- A connection reset basically means abort, or forcibly disconnect; this contrasts with connection termination, which could be viewed as meaning orderly disconnection.
  There are a range of situations where the former is required, and they are explained in detail by [8, Section 3.4]. Many relate to corner cases that could occur in the connection establishment and termination protocols. More generally, however, the idea is that if a host becomes confused or desyncronised wrt. what it receives, then it will abort the connection. Some example scenarios include:
  – the segment relates to a connection that does not exist or has been closed, or
  – the segment is not synchronised, e.g., it is an ACK to something never actually transmitted.

TCP (2)

Data Structure (TCP segment [8, 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.

Notes:

- You might prefer the original ASCII art from [8, Section 3.1]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

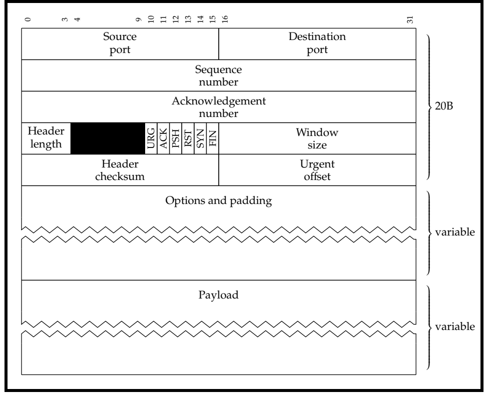
  and a before

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |                       | U | A | P | R | S | F |
| Header Length |        Reserved       | R | C | S | S | Y | I |
|               |                       | G | K | H | T | N | N |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

  and after

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

Data Structure (**TCP segment** [8, 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.

Notes:

- You might prefer the original ASCII art from [8, Section 3.1]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

and a before

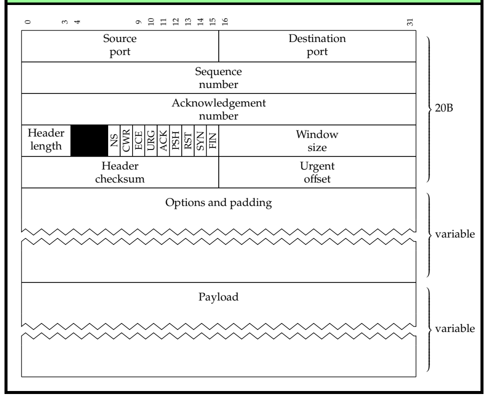
```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |                       | U | A | P | R | S | F |
| Header Length |        Reserved       | R | C | S | S | Y | I |
|               |                       | G | K | H | T | N | N |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

and after

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```


TCP (2)

Data Structure (**TCP segment** [8, Section 3.1]+[10, Section 6]+[11, 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)**.

Notes:

- You might prefer the original ASCII art from [8, Section 3.1]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

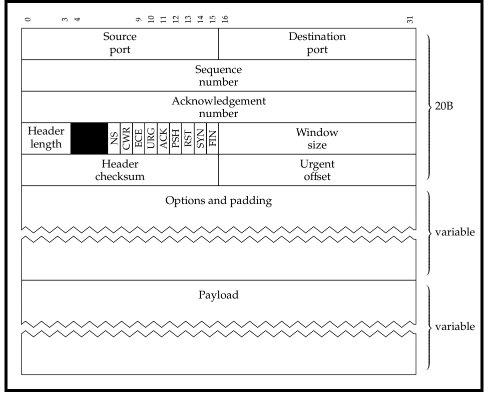
and a before

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |                       | U | A | P | R | S | F |
| Header Length |        Reserved       | R | C | S | S | Y | I |
|               |                       | G | K | H | T | N | N |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

and after

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

Data Structure (TCP segment [8, Section 3.1]+[10, Section 6]+[11, 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.

Notes:

- You might prefer the original ASCII art from [8, Section 3.1]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

  and a before

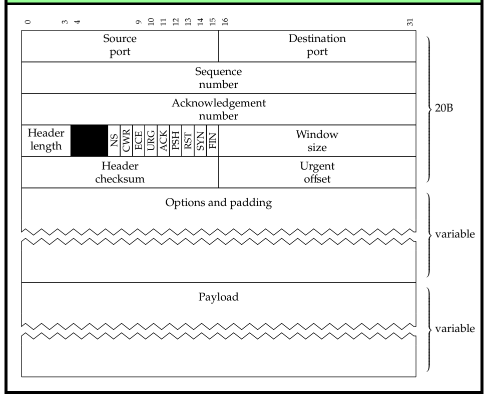
```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |                       | U | A | P | R | S | F |
| Header Length |        Reserved       | R | C | S | S | Y | I |
|               |                       | G | K | H | T | N | N |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

  and after

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```


TCP (2)

Data Structure (TCP segment [8, Section 3.1]+[10, Section 6]+[11, 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.

Notes:

- You might prefer the original ASCII art from [8, Section 3.1]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

  and a before

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |                       | U | A | P | R | S | F |
| Header Length |        Reserved       | R | C | S | S | Y | I |
|               |                       | G | K | H | T | N | N |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

  and after

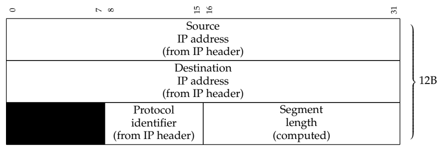
```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

- The TCP checksum is

$$\text{checksum} = H(\text{ pseudo-header } \parallel \text{ TCP header } \parallel \text{ TCP payload })$$

i.e., covers an extra **pseudo-header**, namely

Data Structure (TCP pseudo-header [8, Section 3.1])



- Why?!
  - Reliability is *normally* viewed wrt. errors in data ...
  - ... but delivery errors can also occur (e.g., the segment is somehow delivered to the wrong address or interpreted using the wrong protocol).
  - The pseudo-header deals with some such cases, *without* adding (communication) overhead).

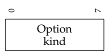
Notes:

- You might prefer the original ASCII art from [8, Section 3.1]:

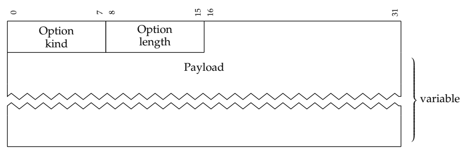
```
+--------+--------+--------+--------+
|           Source Address          |
+--------+--------+--------+--------+
|         Destination Address       |
+--------+--------+--------+--------+
|  zero  |  PTCL  |    TCP Length   |
+--------+--------+--------+--------+
```

- Note that the pseudo-header is not communicated in the same way as the real header: it is virtual, in the sense the content only really exists when being incorporated into the checksum, which is possible because it replicates fields from elsewhere (e.g., from the network layer).
- One implication of this mechanism is that it violates the layered model. Whereas we *want* to separate the network and transport layers from each other using a "clean" interface, this is an example where such an interface breaks down (albeit intentionally, motivated by practical concerns of the time: now you might say it is unlikely a segment will be delivered to the wrong address, but then it was a legitimate issue).

Data Structure (TCP segment option: case #1 [8, Section 3.1])



Data Structure (TCP segment option: case #2 [8, Section 3.1])



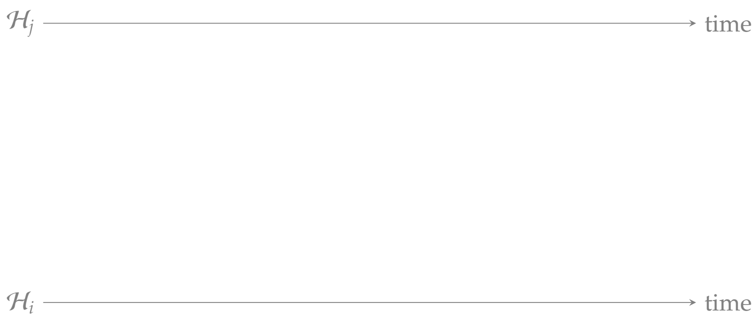
Notes:

- IANA maintain a definitive set of assigned TCP options at

  http://www.iana.org/assignments/tcp-parameters

- The options basically form a list, which is terminated by an option with a special type (i.e., whose kind field is zero).
- Each option can have an associated payload (case #2) or not (case #1). If there *is* a payload, the *entire* option length is specified using the associated 8-bit field.

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.

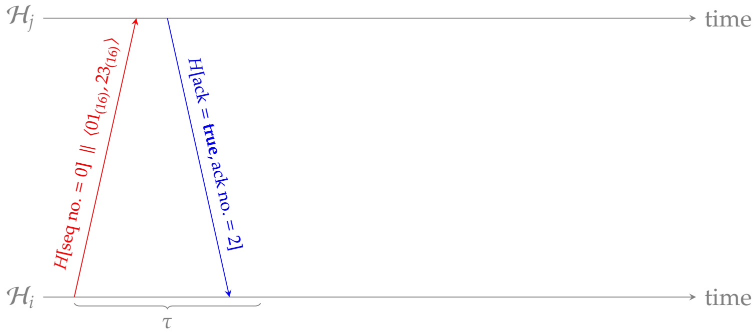
Notes:

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.

Notes:

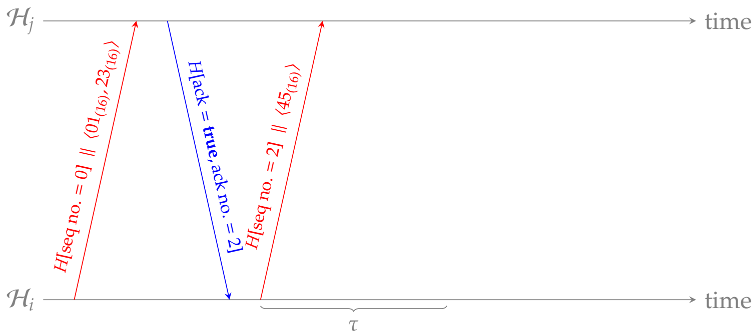
TCP (5)



- 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.

Notes:

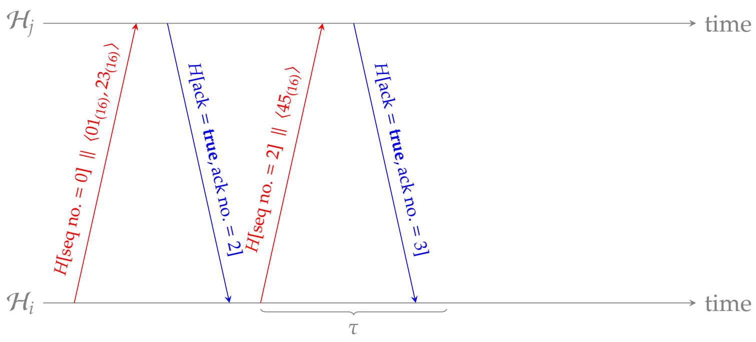
TCP (5)



- 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.

Daniel Page ⟨Daniel.Page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 3627080 @ 2016-03-11
University of BRISTOL

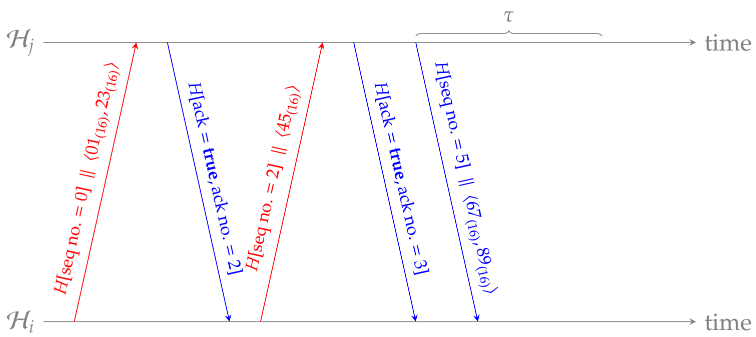
Notes:



- 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.

Notes:

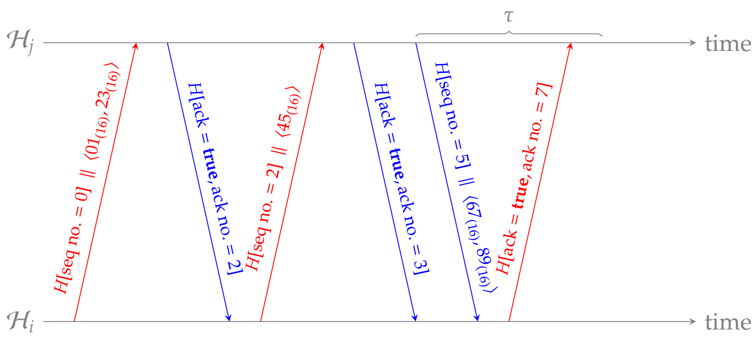
TCP (5)



- 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.

Notes:

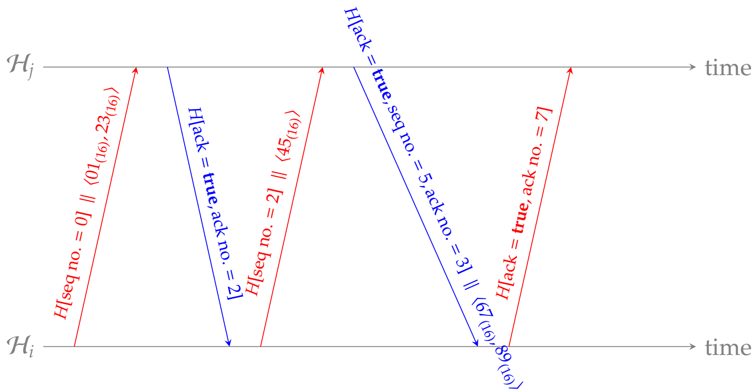
TCP (5)



- 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.

Notes:

TCP (5)

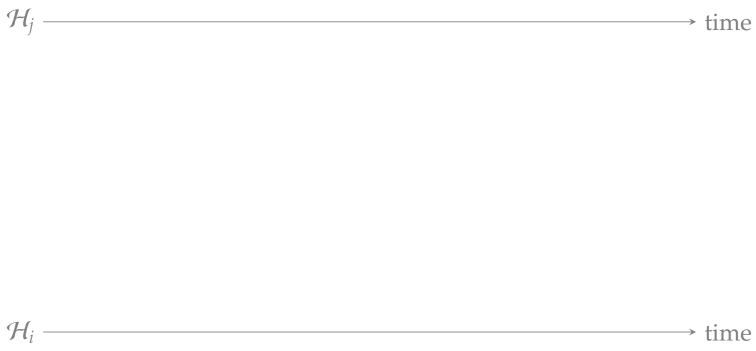


- Question: why have a sequence *and* ACK numbers?
- Answer: it allows optimisation of stop-and-wait via **ACK piggy-backing** st.
  - communication is reduced, i.e., it optimises use of bandwidth, *but*
  - if applied rigidly, means the host needing to transmit an ACK *might* block.

Notes:

TCP (6)

- Example:



Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP 3-**way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (6)

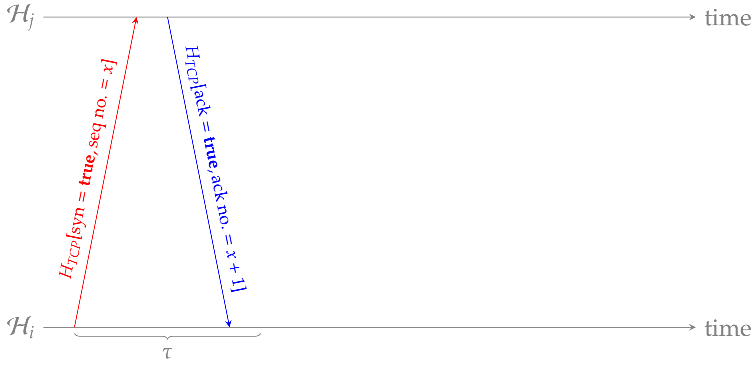
- Example:
  1. connection establishment,

Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP 3-**way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

- Example:
  1. connection establishment,

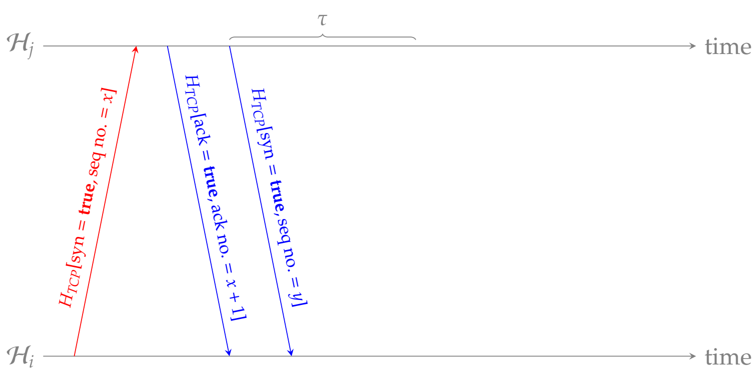


Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

- Example:
  1. connection establishment,

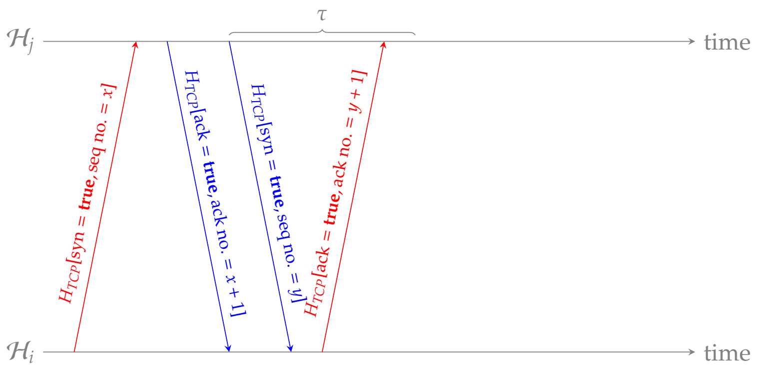


Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

- **Example**:
  1. connection establishment,



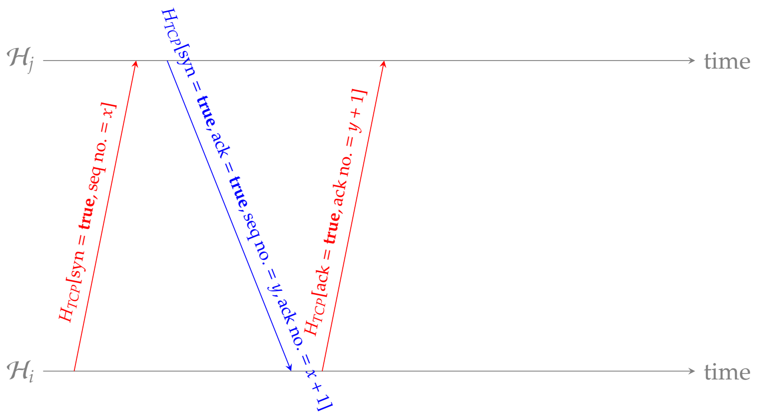
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (6)

- **Example**:
  1. connection establishment,



Daniel Page (Daniel.Page@bristol.ac.uk)
Concurrent Computing (Computer Networks)
git # 3627080 @ 2016-03-11
University of BRISTOL

Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (6)

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

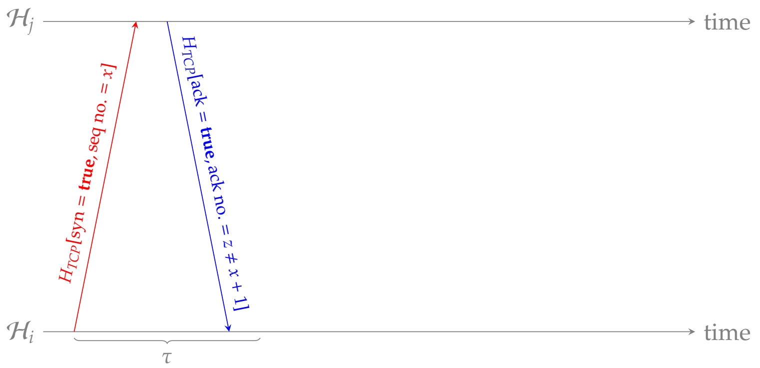
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (6)

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



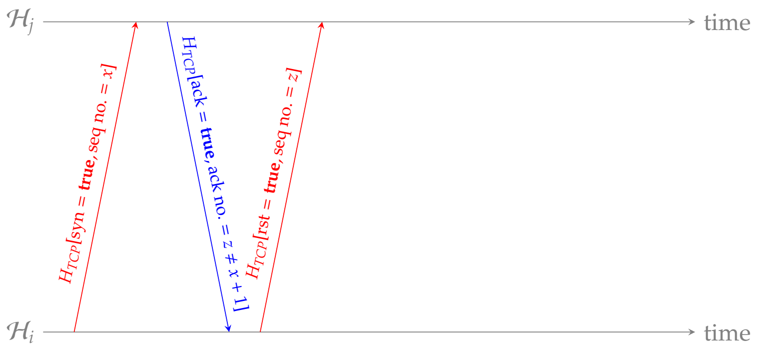
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (6)

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



Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP 3**-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (6)

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

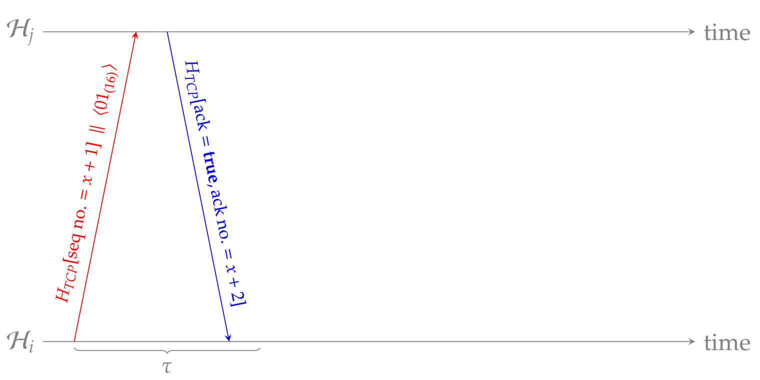
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP 3**-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (6)

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



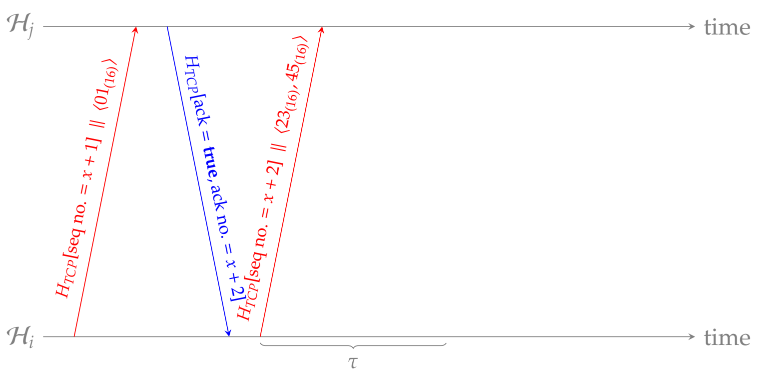
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x+1$ and $y+1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x+1$ and $y+1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x+1$) correctly otherwise the protocol aborts.

TCP (6)

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



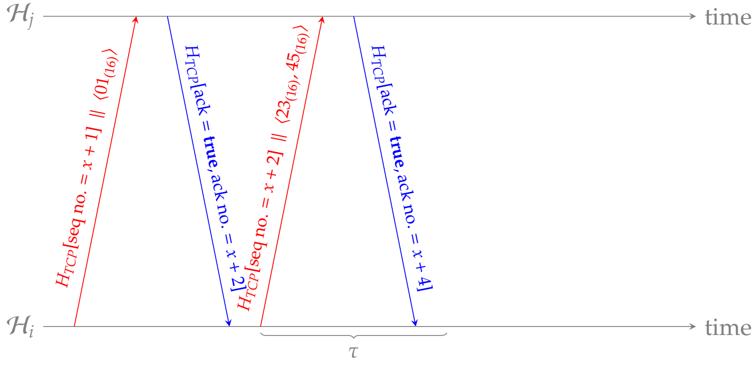
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x+1$ and $y+1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x+1$ and $y+1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x+1$) correctly otherwise the protocol aborts.

TCP (6)

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



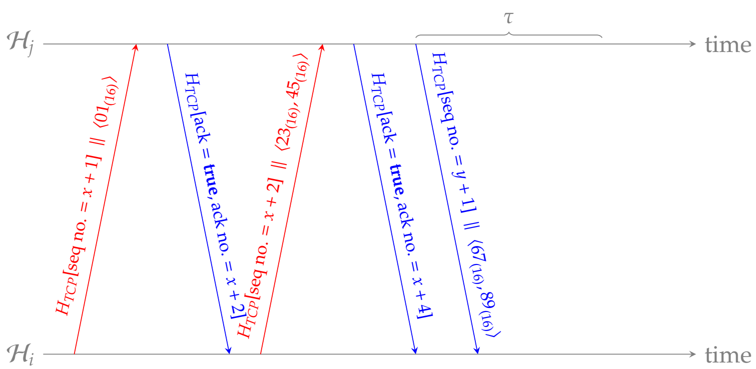
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (6)

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

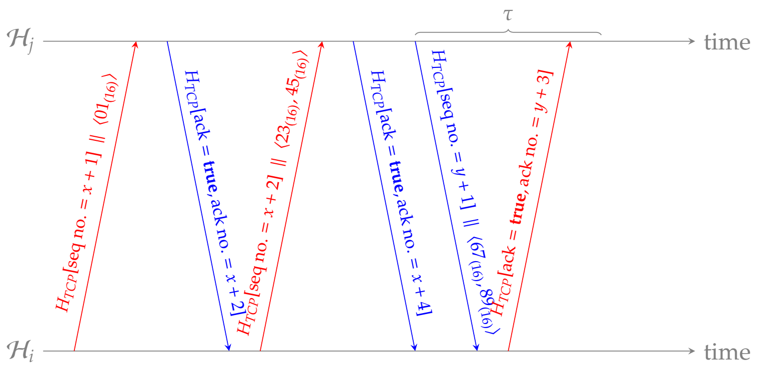


Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

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

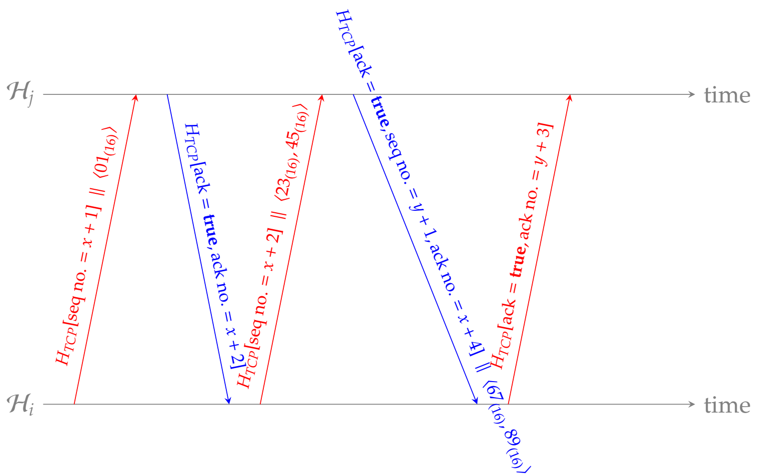


Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x+1$ and $y+1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x+1$ and $y+1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x+1$) correctly otherwise the protocol aborts.

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



Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x+1$ and $y+1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x+1$ and $y+1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x+1$) correctly otherwise the protocol aborts.

TCP (6)

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

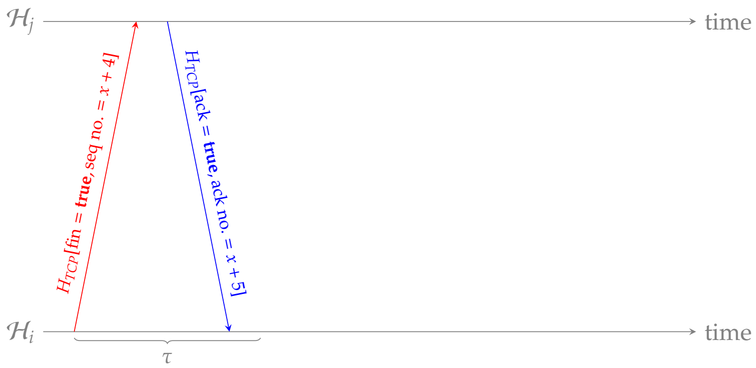
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (6)

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



Notes:

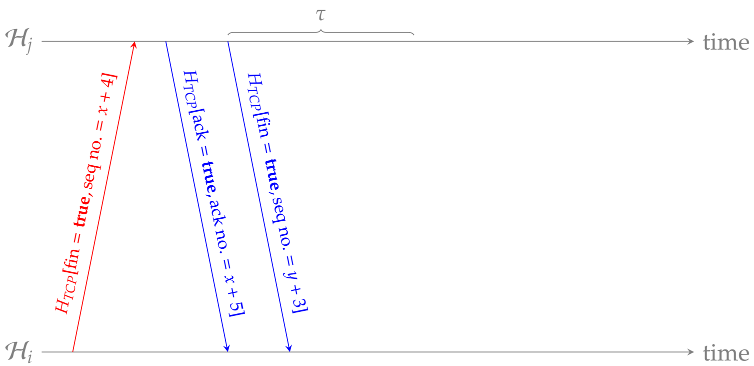
- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

Daniel Page (Daniel.Page@bristol.ac.uk)
Concurrent Computing (Computer Networks) git # 3627080 @ 2016-03-11
University of BRISTOL

TCP (6)

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



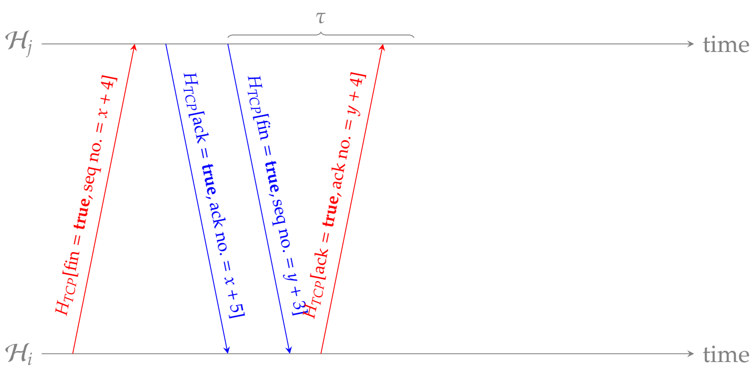
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x+1$ and $y+1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x+1$ and $y+1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x+1$) correctly otherwise the protocol aborts.

TCP (6)

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



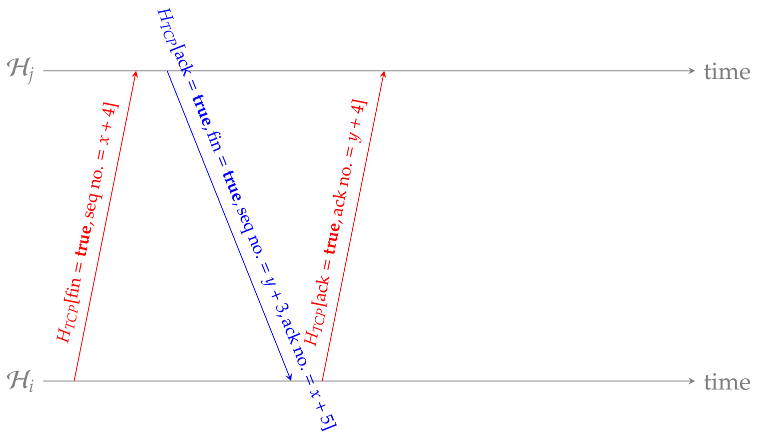
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP **3-way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x+1$ and $y+1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x+1$ and $y+1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x+1$) correctly otherwise the protocol aborts.

TCP (6)

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



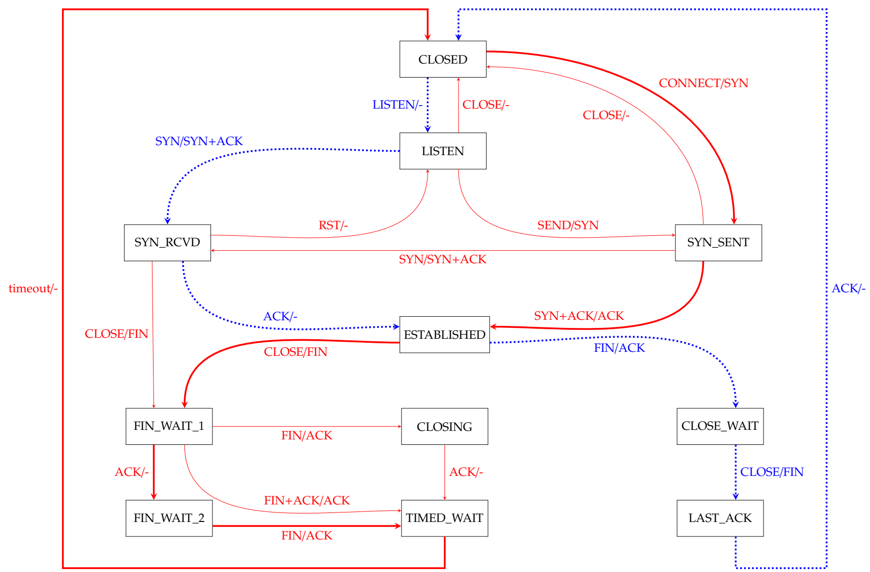
Notes:

- (At least) two quantities influence how communciation operates:
  1. The **Maximum Segment Size (MSS)** [9] is the largest segment a host is willing (or able) to accept; for TCP, this is set to 536B by default. Note that if the MSS (plus lengths of associated headers) is smaller than the underlying MTU, this avoids any fragmentation of segments by the network layer; a larger MSS is typically more efficient up to the point where fragmentation can occur, at which point there is some penalty wrt. efficiency but also reliability. The MSS is announced by a given host, using a suitable option field during the connection establishment process: this means for a given connection, the MSS for communication in one direction *could* differ from the other if/when need be.
  2. The **Maximum Segment Lifetime (MSL)** represents the longest period of time a segment is allow to remain undelivered; for TCP, this is set to 120s by default. Fixing *an* MSL (whatever it is) is important, since it allows reasoning about issues such as sequence number wrap-around (i.e., if the MSL is known, we have a bound on how much undelivered data can exist and so how "old" a segment we are likely to encounter in practice).
- For obvious reasons, the connection establishment is often referred to as the TCP 3-**way handshake protocol** (in reference to the communicated messages): in protocol design, **handshake** is a generic term for the step where parties agree on how to then communicate.
  It's common to see connection termination described as connection release or even **tear down**; the latter is a generic term for "destroying" the connection, albeit in an orderly way.
- The structure of *both* protocols can result in inconsistent state (viewed from one host or the other) in the sense the connection can be "half closed" for example.
- During connection establishment (resp. termination), if a SYN (resp. FIN) is lost then it is retransmitted per ARQ.
- Versus previous slides where we've simply used increasing sequence and ACK numbers starting from 0, now
  1. we start from x and y respectively, each of which is sometimes termed an **Initial Sequence Number (ISN)**,
  2. the sequence number in a segment is the first octet in the payload wrt. the overall octet stream,
  3. the ACK number is a segment is the next octet expected (so one can think of the ACK number as more like a request number in a sense)

  So during connection establishment and termination, the ACK numbers are $x + 1$ and $y + 1$ is because the SYN (resp. FIN) is assumed to consume 1 octet of sequence number space: $x + 1$ and $y + 1$ are what's expected next.
- Typically, the ISNs are selected at random: the rational for this is that it minimises the chance of two instances of the same connection (i.e., a first connection which is then terminated, and then a second connection which is established to match the first) being confused with each other by using the same sequence numbers (too soon at least: they will still do eventually, since there's a finite number of them).
- You can think of the connection establishment and termination protocols as being (sort of) challenge-response protocols. That is, one host first transmits a challenge (e.g., a sequence number x), then the other has to respond (i.e., echo $x + 1$) correctly otherwise the protocol aborts.

TCP (7)

Algorithm (**TCP state machine**)



Notes:

- Obviously *both* end-points are using the same FSM, so you have to view it from the right perspective. Typically, there is
  - an **active host** (the one that initiates, cf. client), and
  - a **passive host** (the one that responds, cf. server).
- The FSM diagram encodes a *lot* of information:
  1. The thick (resp. thin) solid, red lines denote normal (resp. unusual, or abnormal) transitions by the client; these are normally termed **active**.
  2. The thick (resp. thin) dotted, blue lines denote normal (resp. unusual, or abnormal) transitions by the server; these are normally termed **passive**.
  3. Each transition is labelled with the event which caused it, plus the result of it (or a dash where there is no relevant result): basically the former is what the host sees, and the latter what it does.
  4. The lower-left and lower-right sub-graphs represent active (i.e., by the client) and passive (i.e., by the server) closure of a connection.
- One thing it *doesn't* offer much information about is a RST, since a host can receive one in any state (meaning the diagram becomes complex). When a RST is received, the host 1) validates it via the sequence number etc. and 2) processes it to change state accordingly. For the latter, the host proceeds as follows:
  1. in the LISTEN state, the RST is ignored (which doesn't need to be shown),
  2. in the SYN_RCVD state, if it had previously been in the LISTEN state the host transitions into the LISTEN state (which is shown), otherwise it transitions into the CLOSED state (which isn't shown),
  3. in any other state, the host transitions into the CLOSED state (which isn't shown).
- Many of the features in the FSM are there to cope with corner cases wrt. how the hosts can act. For example, it allows *simultaneous* connection establishment: if *both* hosts *simultaneous* try to connect to the other (starting in the CLOSED state), then this can *still* result in an established connection.
- The TIMED_WAIT state is important: the idea is is that the host waits for twice the maximum segment lifetime until transitioning to the CLOSED state. You can think of this as "flushing" the connection in some sense.
  The reason for this is to allow any pending communication to be "cleaned up". For example, it could be the case that the ACK transmitted by the host (when transitioning from CLOSING or FIN_WAIT_2): in this case, the FIN transmitted by the *other* host will be retransmitted, but we want to avoid this influencing some future connection.

Conclusions

Continued in next lecture ...

Notes:

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] R. Elz and R. Bush.
Serial number arithmetic.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1982, 1996.
http://tools.ietf.org/html/rfc1982.

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

Notes:

References

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

[9] J. Postel.
The TCP maximum segment size and related topics.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 879, 1982.
http://tools.ietf.org/html/rfc879.

[10] 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.

[11] 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.

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

Notes: