Reliable Transport

COMPUTER NETWORKS A.A. 24/25

Leonardo Maccari, DAIS: Ca' Foscari University of Venice,
leonardo.maccari@unive.it

Venice, fall 2024

License

- These slides contain material whose copyright is of Olivier Bonaventure, Universite catholique de Louvain, Belgium https://inl.info.ucl.ac.be
- The slides are licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License.

Sect. 1 Reliable Transport

Error Detection Vs Error Correction

- We have seen how we can use a checksum to detect errors.
- But how do we correct the errors?
- There is another technique that is called error correction, which provides enough redundancy not only to detect the error, but even to correct it
- The easiest way is to use a redundant encoding.
  - Map every 1 to 111
  - Map every 0 to 000
- If 010 is received, this is most likely an error in the transmission of a 0 bit.

Error Correction

However, notice that:

- Errors can still be present: `010` could be the transmission of a `1` with two bit errors!
- We are constantly wasting 2/3 of the bit-rate, however, we expect errors to happen rarely.
- Is it convenient? The designer of the Internet made the following assumption:

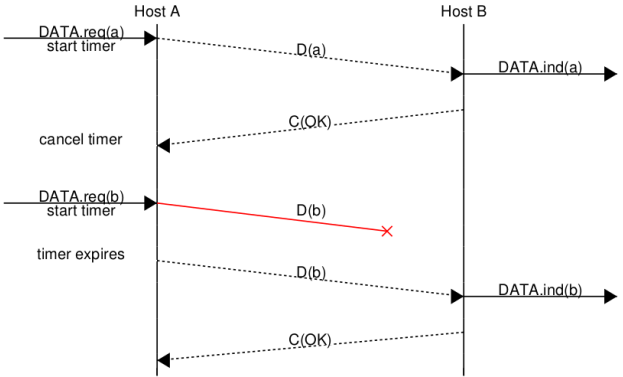
Packet Drop

As a general principle, it is assumed that the network works *reasonably well*: the probability of error is assumed to be low enough, so errors are infrequent. Given this, it is not convenient to waste too many resources for correcting/avoiding errors, it is more convenient to throw away frames with errors, and ask for a retransmission.

In practice, frames with errors are just dropped, the receiver pretends he never received them.

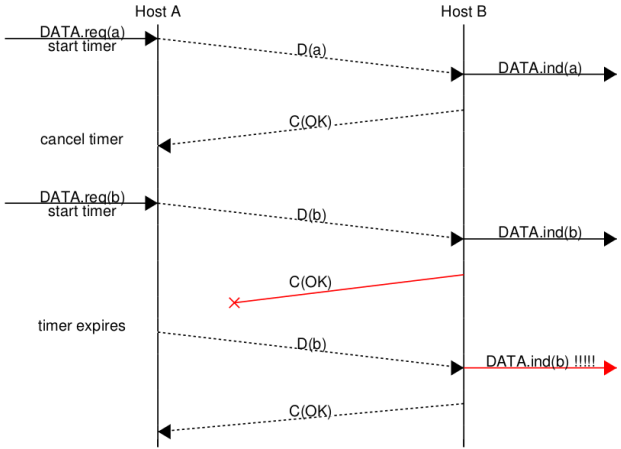
Errors in data frames

- Since we know how to detect errors, the sender of a frame can use a very easy scheme:
  1. Send the frame
  2. Start a timer
  3. If the ACK does not arrive, go to 1
- It does not matter if the frame never arrived, or it arrived with errors, the result is the same.



Errors in ACK frames

- However, this situation can not be handled correctly
- The frame is delivered twice, but the receiver will not know that the second frame is a copy of the first one.



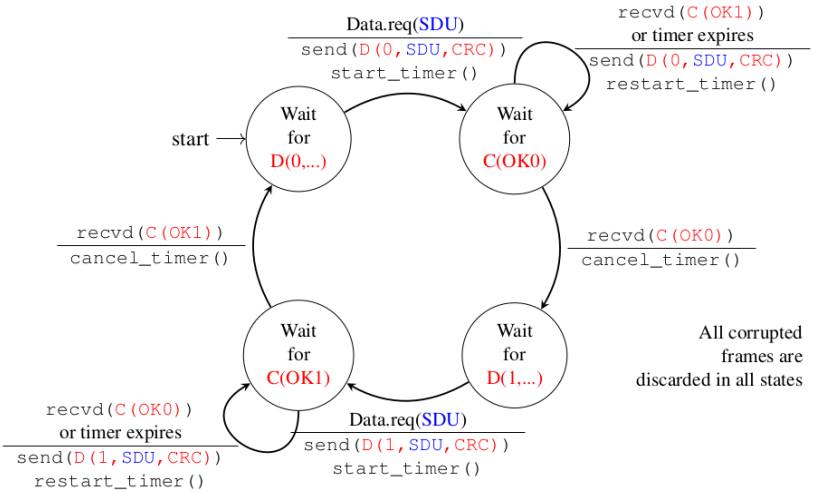
Sequence Numbers

- The receiver needs to be able to distinguish copies of frames.
- We add to the header another field: the *sequence number*.
- In the simplest case, the sequence number can be a single bit, that is flipped at every frame.
- The `ACK` frame will report which sequence number is referring to
- This simple scheme is called Alternating Bit Protocol

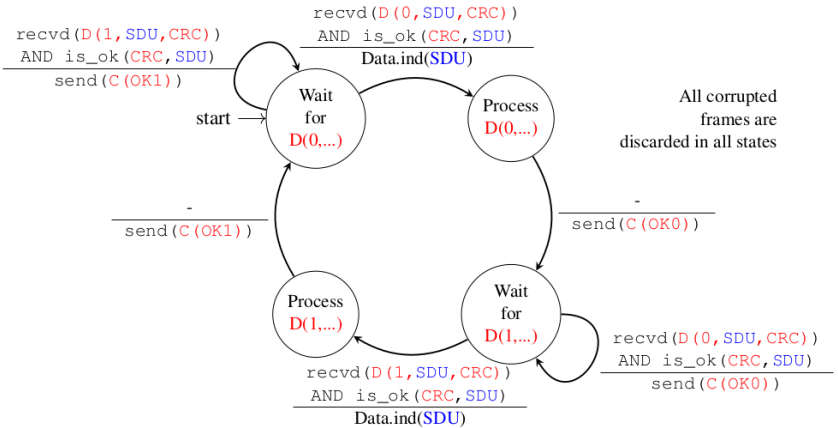
Terminology of the ABP SM:

- in ISO/OSI terminology, an SDU is a Service Data Unit, that is any data that a layer passes to the underlying one.
- D(seq, SDU, CRC): a data frame containing a seq number, an SDU, and a checksum
- C(OK1): a control frame that says that frame with sequence number 1

ABP: Sender State Machine



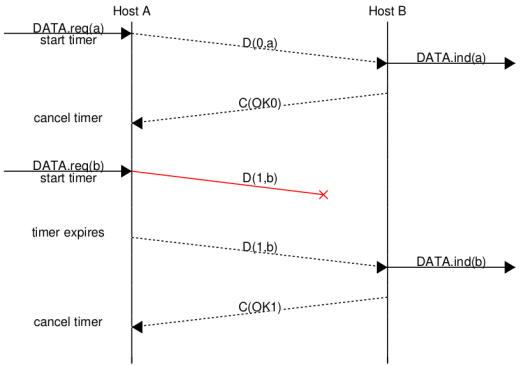
ABP: Receiver State Machine



Note Well

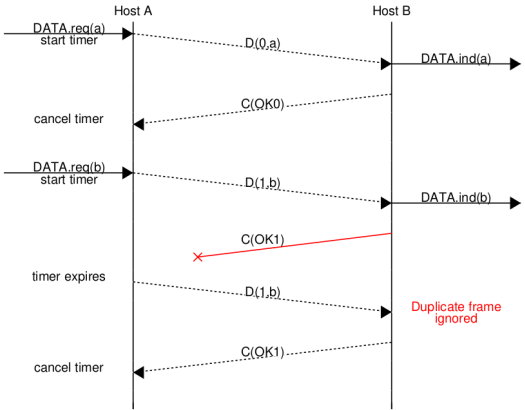
- The ACK is sent only after the processing of the frame is completed, so the sender can decrease the sending rate if the receiver is not quick enough to process frames
- In the self loops the receiver receives a copy of an old frame. It is ACKed, but discarded, it is not processed.
- The SMs keep working up to when they are synchronized. If the receiver is waiting for seq 1, and the sender reboots, the sender goes in the initial state and there is no way to resync.
- We will see that real protocols need a set-up phase

ABP: Case of SDU Drop



- Host B does not send the `C(OK1)`
- The timer on Host A expires, then Host A remains in the waiting state and re-sends the frame
- Once received the `C(OK1)` is sent and both SMs can move to the next state

ABP: Case of ACK Drop



- Host B is waiting for sequence number 1
- The timer expires on Host A, the frame `D(1,d)` is re-sent
- Host B can detect it is a copy, sends the ACK anyway

Change of the Frame Format

- In the previous images the data frames have an added field, that is the sequence number: `D(seq_n, data)`
- Also the `ACK` frames are different, `OK0` for data frames with `seq_n = 0` and `OK1` for data frames with `seq_n = 1`

Note again: does the upper layer primitive change? NO. Layering allows to modify one layer without modifying the others. However now the datalink layer offers a new service: reliable delivery.

ABP: Performance

- Assume that it takes 10ms for a frame to go from the sender to the receiver
- Then the timer that the sender uses must be at least 20ms, this is called a *round trip time* (RTT)
- Assume the frame is 1500B, which is the standard size of an Internet packet
- Assume the link is a gigabit link: 10^9 b/s
- It takes $\frac{1500*8}{10^9} = 0.012$ms to send a data frame.
- An `ACK` frame is made of two bits (one for the header, one for the `seq_n`): $\frac{2}{10^9}$ms. It is small enough to be ignored.

Time Evolution

$t = 0$: A start sending

$t = 0.012$: A finished sending

$t = 10.0$: Starts to receive the frame

$t = 10.012$: B finished receiving the frame, sends ACK

$t = 20.012$: A finished receiving the ACK (ACK is small, so time to transmit is negligible) and can start another transmission

It takes 20.012 ms to send a frame made of 1500B, which corresponds to a bit-rate of: $\frac{1500*8}{0.020024} \simeq 0.6Mb/s$

Protocol Overhead

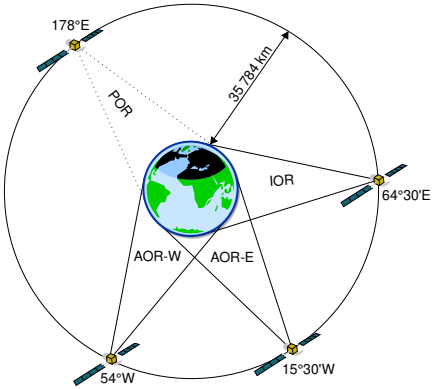
If we ack every single frame, no matter how fast the link is, the RTT makes the transmission slower. This is another form of protocol overhead, that is introduced to offer a better service at the Datalink layer: reliable delivery.

Keep in mind that we can increase the capacity of a link, but there are physical limits to reducing the RTT: information can not travel faster than light.

Realistic Example

- The Inmarsat satellite system uses 4 satellites to cover the whole globe
- The satellites use a geostationary orbit (36000 km from the ground)
- Each of them covers a fixed area of $1/3$ of the globe

Inmarsat[1]

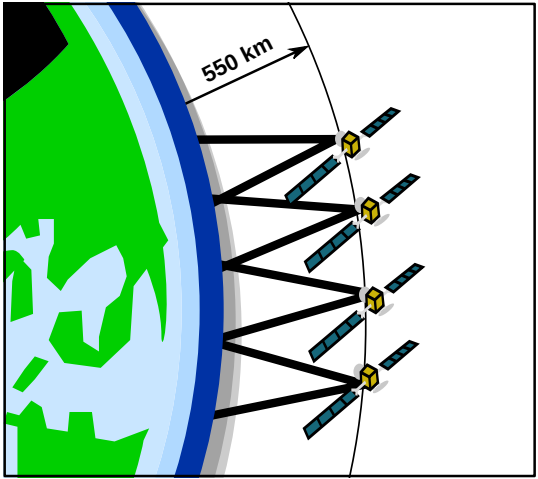


[1]The image by wikipedia, released with Creative Commons Attribution-Share Alike 3.0 license: https://en.wikipedia.org/wiki/File:Couverture_satellite_inmarsat.svg

Delay

- electromagnetic waves travel at the speed of light: 3×10^8m/s
- The RTT due to the wave propagation alone is $2 \times \frac{35,784,000}{3 \times 10^8} \simeq 200ms$
- There is no way you can reduce this RTT, as long as you use geostationary satellites
- What can you do to reduce delay?

Reducing Delay



- Starlink uses Low-Earth orbit satellites: 550km from the ground[a].
- RTT is reduced by a factor 65
- However, we need 10.000 more
- And they move so they are very hard to manage...

[a] See `https://www.starlink.com/technology`

Take Away

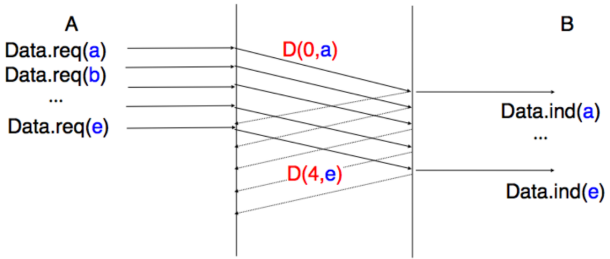
- The alternating bit protocol is simple to implement and reliable
- However, it introduces an intolerable delay when links are long
- We can not use it when links are really long, we need to introduce *pipelining*.

Reliable Transport

↳1.1 Go-back-n and Selective Repeat

Pipelining

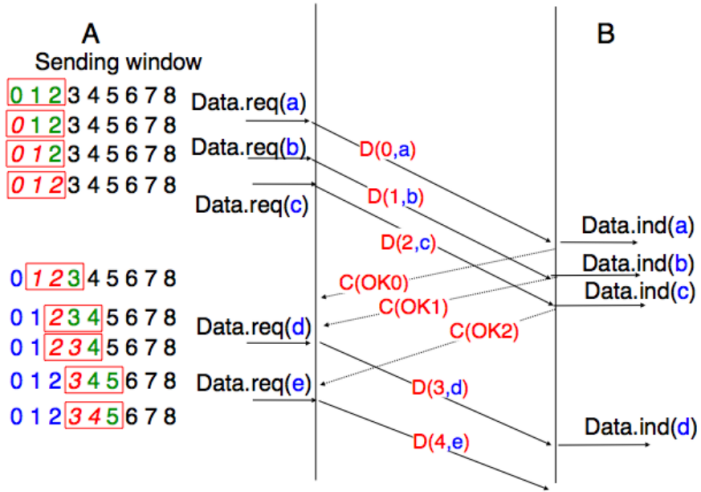
- The best way to exploit a fast channel is to *pipeline* the packets
- This means sending a batch of frames, and while A sends them, B can start acknowledging
- However, we still need to avoid overflow and guarantee the reliability, that is achieved with a *sliding window* mechanism.



Sliding Window

- sequence number are integer numbers, A and B agree on a sliding window size.
- In the figure you have a window size set to 5
- The sender (A) will send 5 frames and stop.
- As data frames arrive B will send `ACK`s for specific sequence numbers.
- When the frame with the lowest sequence number is acked, A moves the window right and sends another, an so on.

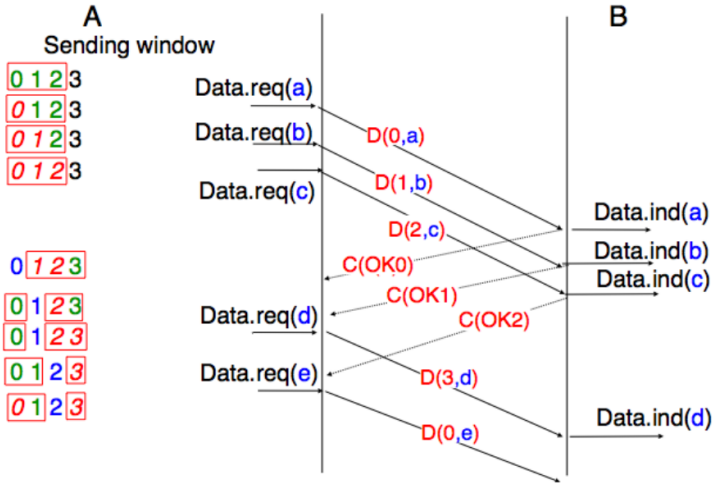
Ex with Window Size 3



Finite Seq. Numbers

- Consider that a frame header is finite
- The bits dedicated to the Seq. number are limited to let's say n
- Then the header field can range from 0 to $2^n - 1$
- The sequence number will at some point wrap up, but this is not a problem
- We call `maxseq`, the size of the space of the sequence numbers (i.e. 2^n).

Ex with Window Size 3



Flow Control and Frame Loss

- The moving window implements flow control and reduces the impact of the RTT.
- Now the frames arrive in batches, the ACKS are generated in batches too so the performance of the Datalink layer improve
- However, We need a policy to decide how to treat a loss of some frame

Note one key element: a server may handle hundreds of thousands of connections at the same time. Besides being correct, a policy must be fast. It does not matter if it is not perfect.

- Go-back-n is one policy

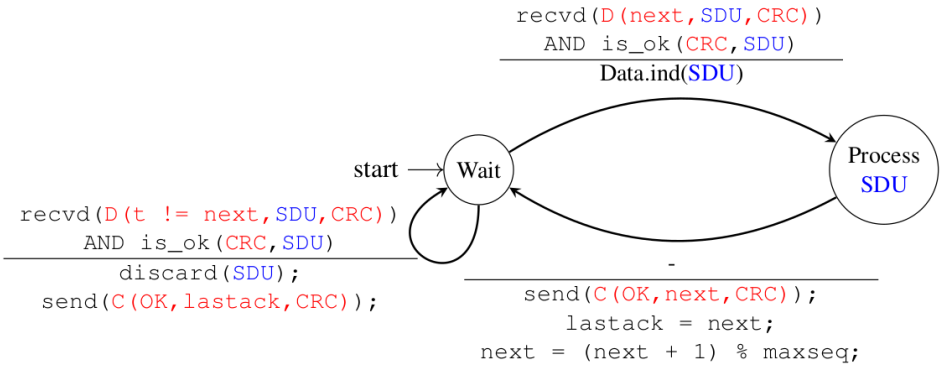
Go-back-n: the Receiver B

- B only accepts the frames that arrive in-sequence.
- When B receives a data frame, it always returns an acknowledgment containing the sequence number of the last in-sequence frame that it has received.
- B discards any out-of-sequence frame that it receives.
- This acknowledgment is said to be *cumulative*, meaning that it acknowledges the last frame and all the previous ones with a lower seq. number

Go-back-n: the Receiver software

- Frames are processed one by one, there is no buffering at the receiver (less memory).
- The receiver maintains just two variables:
  - `lastack`: the last sequence numebr that was ACKed
  - `next`: the next sequence number that is expected

Receiver SM



B needs two state variables: `lastack` and `next`.

Go-back-n: The Sender A

- A Maintains a sending buffer that can store an entire sliding window of frames, maximum size iw `W`, the window size.
- The frames are sent with increasing sequence numbers, up to when the sending buffer is full.
- A then stops and waits for an acknowledgment.
- A uses a single retransmission timer that is started when the first frame is sent.

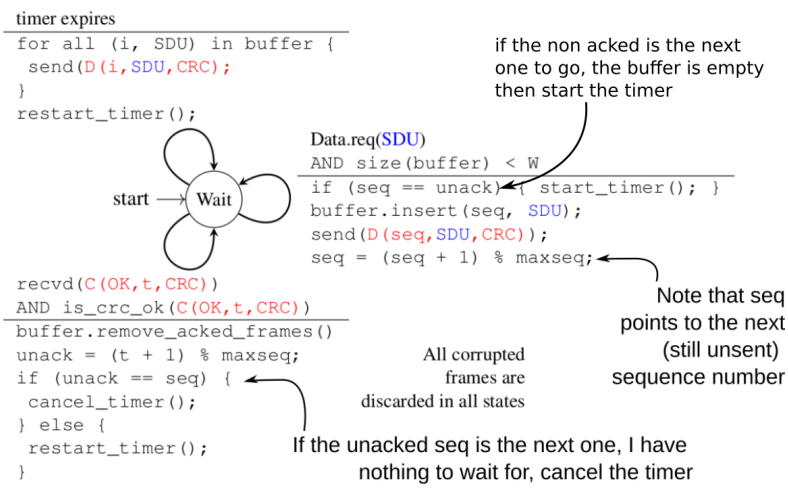
Go-back-n: The Sender A

- When A receives an acknowledgment:
  - it removes from the sending buffer *all the acknowledged frames* (remember, it is a *cumulative ack*)
  - it restarts the retransmission timer only if there are still unacknowledged frames in its sending buffer.
- If the retransmission timer expires, A assumes that all the unacknowledged frames currently stored in its sending buffer have been lost.
- It thus retransmits all the frames in the buffer and restarts its retransmission timer.

Go-back-n: The Sender Software

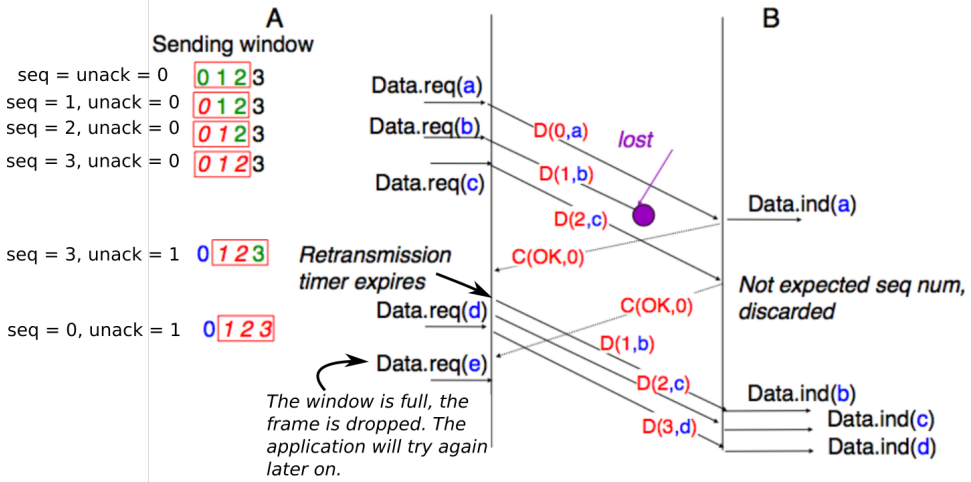
- there are three state variables:
  - `size(buffer)`: the current amount of frames in the sending buffer
  - `seq`: the last sequence number that has been sent
  - `unack`: the sequence number of the first non-acked frame
- And one single timer

Sender SM



`seq`: the last sent sqn, `unack`: first non-acked sqn, `W` = window size.

Go-back-n: Example[2]

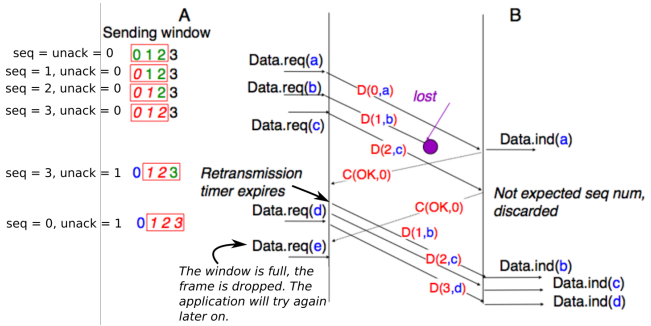


[2]The image from the book was edited, as it is not entirely clear.

Go-back-n: Example

Note:

- When A receives C(OK,0), seq 0 is removed, the window shifts, the timer restarts (`unack != seq`).
- Then the timer expires and **both b and c are** resent.
- When **Data.req(d)** arrives, the buffer has one free slot: it is sent
- When **e** arrives the buffer is full. The frame is rejected, the application will have to try again later.



Maximum Window Size[3]

- Assume the sequence number is contained in header made of **n** bits
- The space of the numbers is 2^n, can we use it all? Consider the following situation:
  - A sends exactly 2^n frames, from $n = 0$ to $n = 2^n - 1$. It then waits for ACKS.
  - B receives them all, sends them to the upper layer, and sends all the ACKs. `next` in B is set to 0.
  - However, the ACKs are all lost, A will timeout and resend them all, from $n = 0$ to $n = 2^n - 1$.
  - B receives them. They match the next expected sequence number. B then does not detect they are copies, and send them up to the application.
- To solve this problem, the window must be at most of size $2^n - 1$.

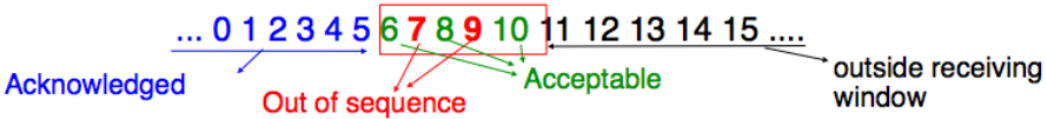
[3]Hint: this theme is explained in the book, but the text is not clear. Clarify it and make a pull request.

Limits of Go-back-n

- Go-back-n is easy to implement (the SMs are very simple, there are only two state variables and one timer), it works well when only a few frames are lost and reduces the effect on the RTT
- However, when there are many losses, the performance of go-back-n quickly drops for two reasons:
  - the receiver does not accept out-of-sequence frames
  - the sender retransmits **all unacknowledged frames** once it has detected a loss
- In the example, **c** was correctly received, but it was retransmitted anyway.

Selective Repeat

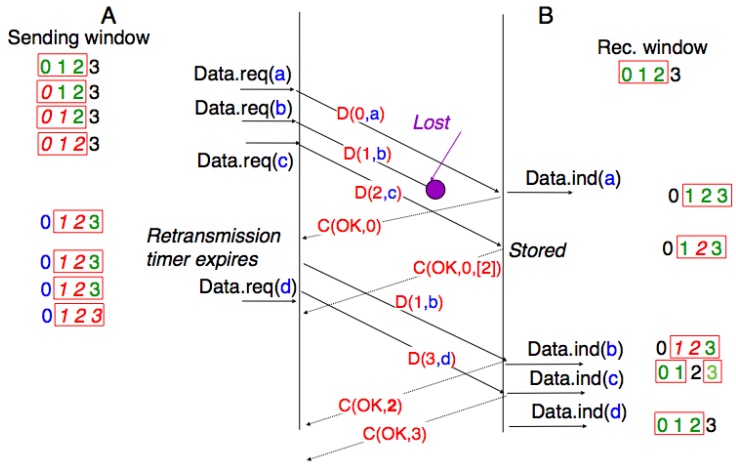
- Go-back-n can be improved by a selective repeat scheme:
  - B now has a buffer too, and it accepts frames as long as they are in the window
  - In each ACK, B reports the last sequence number before the beginning of the window, **and a list of frames correctly arrived out of order**



Selective Repeat

- Every time a sends a frame it starts a timer (one timer per frame in the buffer)
- When A receives an ACK it removes all the times that correspond to the ACKed frames
- If possible, it shifts the window right
- When a timer fires, only the corresponding frame is resent.

Selective Repeat



Real Protocols: Piggybacking

- In the real world, an exchange of data is never in one single direction
- Protocols at the higher layers include data in both ways
- It is then convenient to include the ACK number in the header of the protocol, so that an ACK can be *piggybacked* (added) to a data frame, and it does not require a dedicated frame

Real Protocols: derferring ACKS

- Piggybacking also means that ACK may be deffered: B does not send an ACK immediately, but waits a little time for a data frame to be sent
- If a data frame must be sent, the ACK is piggybacked, else, after a short timeout an ACK frame is generated
- This can help to reduce the number of ACKs, because more than one data frame can be received in the short time, and they are ACKed cumulatively.

Final Remarks

- The window size doesn't have to be the same for the receiver and the sender
- Windows should be negotiable, so that their size changes depending on the network conditions
- Most of the times, communication protocols are bi-directional, so data flows in both directions. If this happens, then ACK messages can be *piggybacked* to data frames, without requiring dedicated frames.

Live Exercises

- Let's try with the exercises from the interactive book:
  `https://beta.computer-networking.info/syllabus/default/principles/reliability.html#go-back-n-and-selective-repeat`