



# Introduction to Computer Networking

Dr. Mahmoud M. Elkhoully

[www.elkhoully.net](http://www.elkhoully.net)

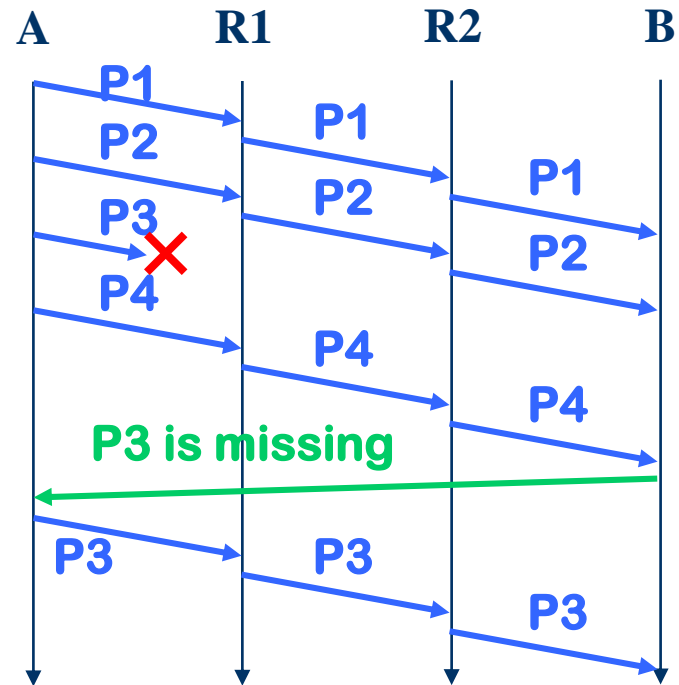
# Contents

---

1. Where should packet losses be repaired ?
2. Mechanisms for error recovery
3. Flow Control

# 1. The Layered Model Transforms Errors into Packet Losses

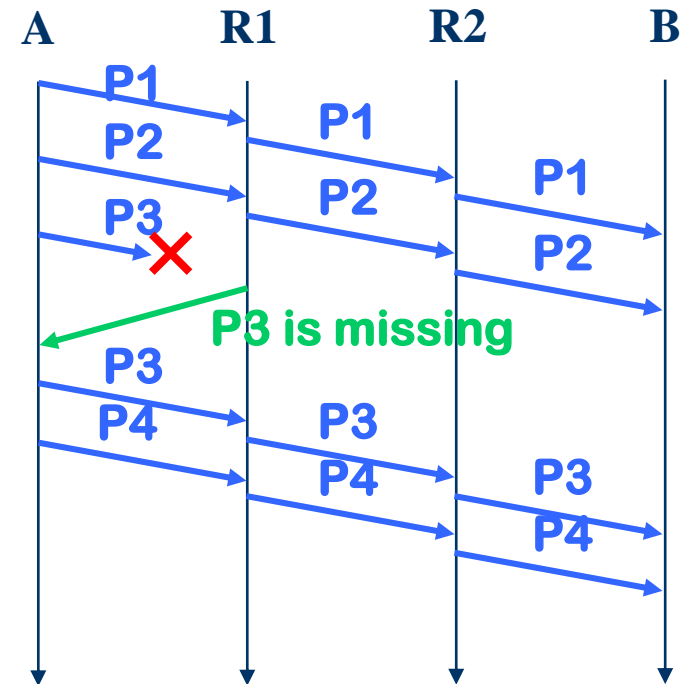
- Packet losses occur due to
    - error detection by MAC
    - *buffer* overflow in bridges and routers
    - Other exceptional errors may occur too
- Q.** give some examples



# 1. The Layered Model Transforms Errors into Packet Losses

- Therefore, packet losses must be repaired.
- This can be done using either of the following *strategies*:
  - **end to end**: host A sends 10 packets to host B. B verifies if all packets are received and asks for A to send again the missing ones.
  - or **hop by hop**: every router would do this job.

Which one is better ? We will discuss this in the next slides.



# The Case for End-to-end Error Recovery

- There are arguments in favour of the end-to-end strategy.  
The keyword here is *complexity*.
  - The TCP/IP architecture tries to keep intermediate systems as simple as possible. Hop by hop error recovery makes the job of routers too complicated.
    - Needs to remember some information about every packet flow -> too much processing per packet
    - Needs to store packets in case they have to be retransmitted -> too much memory required

## 2. Mechanisms for Error Recovery

- In this section we discuss the methods for repairing packet losses that are used in the Internet.
- We have seen one such method already:  
Q. which one ?

### solution

Stop and Go is an example of *packet retransmission protocol*. Packet retransmission is the general method used in the Internet for repairing packet losses. It is also called *Automatic Repeat Request (ARQ)*.

- TCP is an ARQ protocol

# ARQ Protocols

- *Why* invented ?
  - Repair packet losses
- *What* does an ARQ protocol do ?
  1. Recover lost packets
  2. Deliver packets at destination *in order*, i.e. in the same order as submitted by source
- *How* does an ARQ protocol work ?

Similar to *Stop and Go* but:

  - It may differ in many details such as
    - How packet loss is detected
    - The format and semantics of acknowledgements
    - Which action is taken when one loss is detected
  - Practically all protocols use the concept of *sliding window*, which we review now.

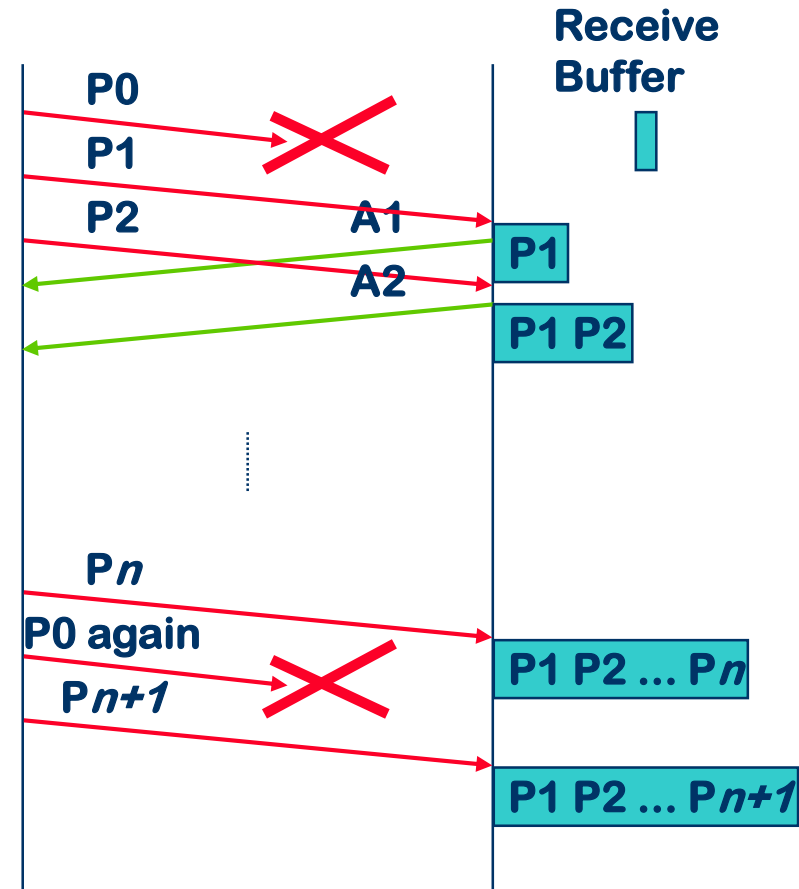
# Windows in TCP

- TCP uses two Windows
  - Send window and receive window
- This means four windows for a bidirectional communication
  - To make simple, we make an assumption that communication is only unidirectional
  - The bidirectional communication can be inferred using two unidirectional communications with piggybacking



# Why Sliding Window ?

- *Why* invented ?
  - Overcome limitations of Stop and Go
  - Q. what is the limitation of Stop and Go ?
  - solution
- *What* does it do ?
  1. Allow multiple transmissions  
But this has a problem: the required buffer at destination may be very large
  2. This problem is solved by the sliding window. The sliding window protocol puts a limit on the number of packets that may have to be stored at receive buffer.



# How Sliding Window Works.

## Legend



Maximum  
Send Window  
=  
Offered Window  
( = 4 here)



Usable Window

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

0 1 2 3 4 5 6 7 8 9 10 11 12

P = 0

A = 0

P = 1

P = 2

A = 1

P = 3

P = 4

P = 5

A = 2

A = 3

P = 6

A = 4

P = 7

A = 5

P = 8

A = 6

P = 9

A = 7

P = 10

# How Sliding Window Works

On the example, packets are numbered 0, 1, 2, ..

The sliding window principle works as follows:

- a window size  $W$  is defined. In this example it is fixed. In general, it may vary based on messages sent by the receiver. The sliding window principle requires that, at any time: number of unacknowledged packets at the receiver  $\leq W$
- the *maximum send window*, also called *offered window* is the set of packet numbers for packets that either have been sent but are not (yet) acknowledged or have not been sent but may be sent.
- the *usable window* is the set of packet numbers for packets that may be sent for the first time. The usable window is always contained in the maximum send window.

# How Sliding Window Works

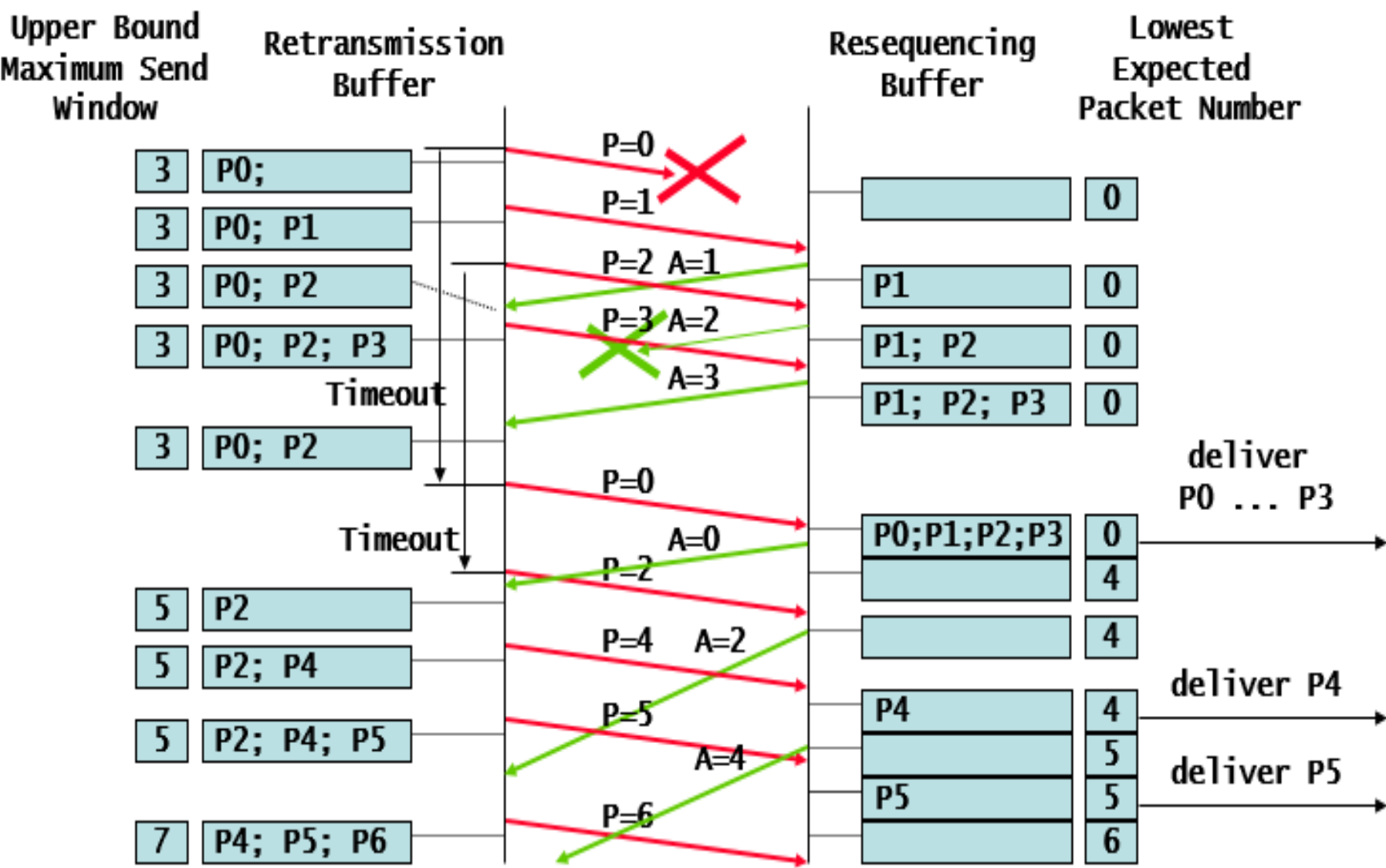
- the lower bound of the maximum send window is the smallest packet number that has been sent and not acknowledged
- the maximum window *slides* (moves to the right) if the acknowledgement for the packet with the lowest number in the window is received

A sliding window protocol is a protocol that uses the sliding window principle.

With a sliding window protocol,  $W$  is the maximum number of packets that the receiver needs to buffer in the re-sequencing (= receive) buffer.

If there are no losses, a sliding window protocol can have a throughput of 100% of link rate (overhead is not accounted for) if the window size satisfies:  $W \geq b / L$ , where  $b$  is the bandwidth delay product, and  $L$  the packet size. Counted in bytes, this means that **the minimum window size for 100% utilization is the bandwidth-delay product.**

# An Example of ARQ Protocol with Selective Repeat



The previous slide shows an example of ARQ protocol, which uses the following details:

1. packets are numbered by source, starting from 0.
2. window size = 4 packets;
3. Acknowledgements are positive and indicate exactly which packet is being acknowledged
4. Loss detection is by timeout at sender when no acknowledgement has arrived
5. When a loss is detected, only the packet that is detected as lost is re-transmitted (this is called *Selective Repeat*).

Q. Is it possible with this protocol that a packet is retransmitted whereas it was correctly received?

**solution**

# \*An Example of ARQ Protocol with Go Back N

Next Sequence

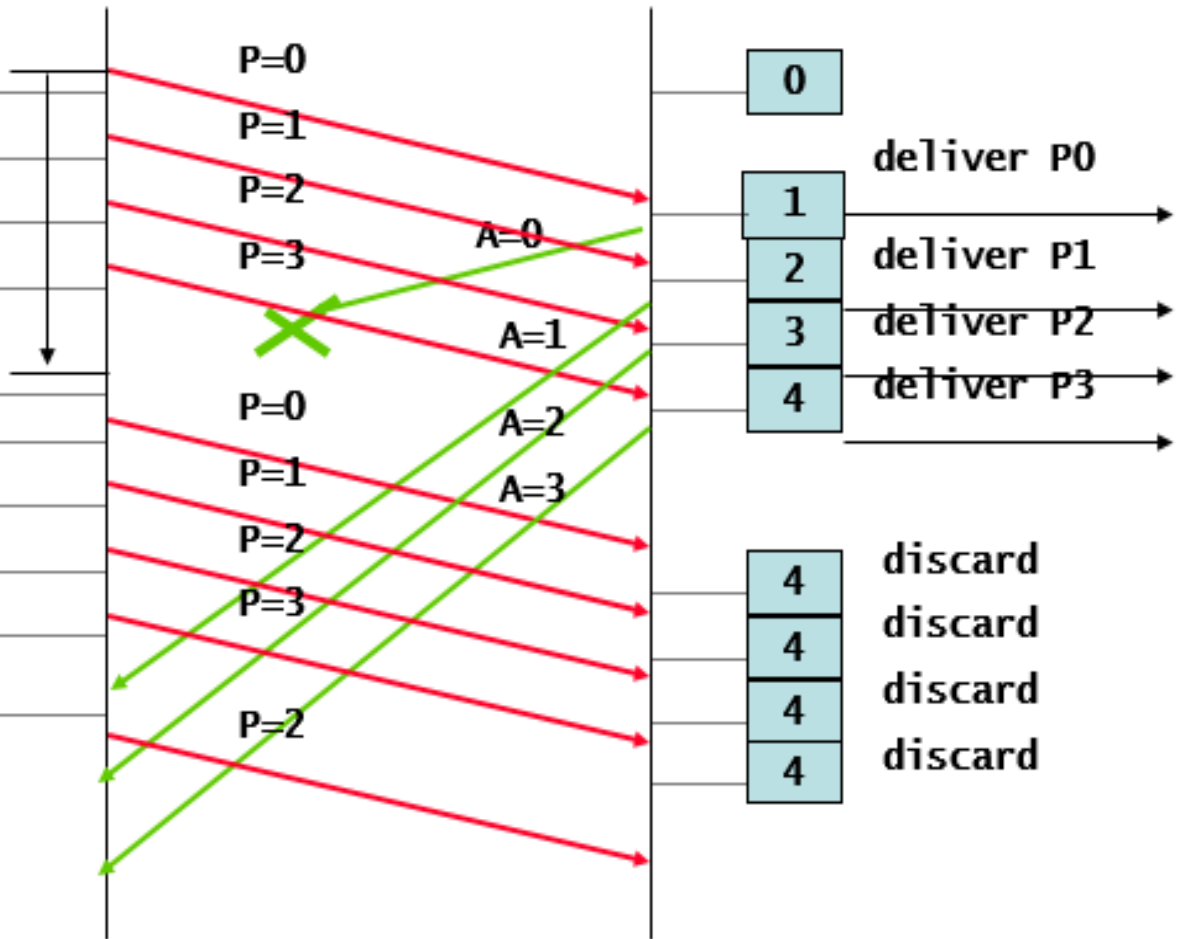
Number for  
Sending  
 $V(S)$

Retransmission  
Buffer

Next Expected  
Packet Number  
 $V(R)$

0	1	P0;
0	2	P0; P1
0	3	P0; P1; P2
0	4	P0; P1; P2; P3
0	0	P0; P1; P2; P3
0	1	P0; P1; P2; P3
0	2	P0; P1; P2; P3
0	3	P0; P1; P2; P3
0	4	P0; P1; P2; P3
2	4	P2; P3

Lowest  
unacknowledged  
packet number  
 $V(A)$



The previous slide shows an example of ARQ protocol, which uses the following details:

1. window size = 4 packets;
2. Acknowledgements are positive and are *cumulative*, i.e. indicate the highest packet number up to which all packets were correctly received
3. Loss detection is by timeout at sender
4. When a loss is detected, the source starts retransmitting packets from the last acknowledged packet (this is called *Go Back n*).

Q. Is it possible with this protocol that a packet is retransmitted whereas it was correctly received?

### Solution

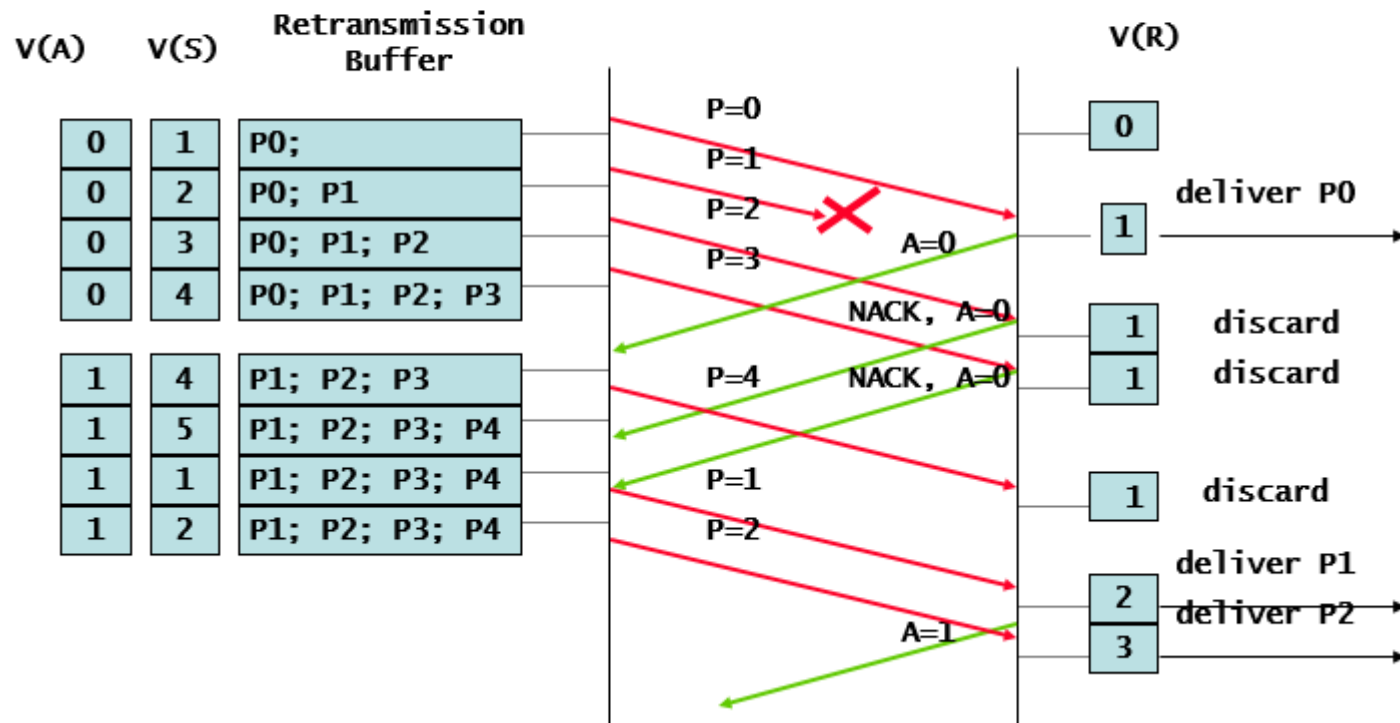
Go Back  $n$  is less efficient than selective repeat, since we may unnecessarily retransmit a packet that was correctly transmitted. Its advantage is its extreme simplicity:

- ▶ (less memory at destination) It is possible for the destination to reject all packets other than the expected one. If so, the required buffer at destination is just one packet
- ▶ (less processing) The actions taken by source and destination are simpler

Go Back  $n$  is thus suited for very simple implementations, for example on sensors.



## \*An Example of ARQ Protocol with Go Back N and Negative Acks



The previous slide shows an example of ARQ protocol, which uses the following details:

1. window size = 4 packets;
2. Acknowledgements are positive or *negative* and are cumulative. A positive ack indicates that packet n was received as well as all packets before it. A negative ack indicates that all packets up to n were received but a packet after it was lost
3. Loss detection is either by timeout at sender or by reception of negative ack.
4. When a loss is detected, the source starts retransmitting packets from the last acknowledged packet (*Go Back n*).

**Q.** What is the benefit of this protocol compared to the previous ?

[solution](#)

# Where are ARQ Protocols Used ?

- Hop-by-hop
  - MAC layer
    - Modems: Selective Repeat
    - WiFi: Stop and Go
- End-to-end
  - Transport Layer:
    - TCP: variant of selective repeat with some features of go back n
  - Application layer
    - DNS: Stop and Go

# Are There Alternatives to ARQ ?

*Coding* is an alternative to ARQ.

- **Forward Error Correction (FEC):**
  - Principle:
    - Make a data block out of  $n$  packets
    - Add redundancy (ex Reed Solomon codes) to block and generate  $k+n$  packets
    - If  $n$  out of  $k+n$  packets are received, the block can be reconstructed
  - **Q.** What are the pros and cons ?  
solution
  - Is used for data distribution over satellite links
  - Other FEC methods are used for voice or video (exploit the fact that some distortion may be allowed – for example: interpolate a lost packet by two adjacent packets)

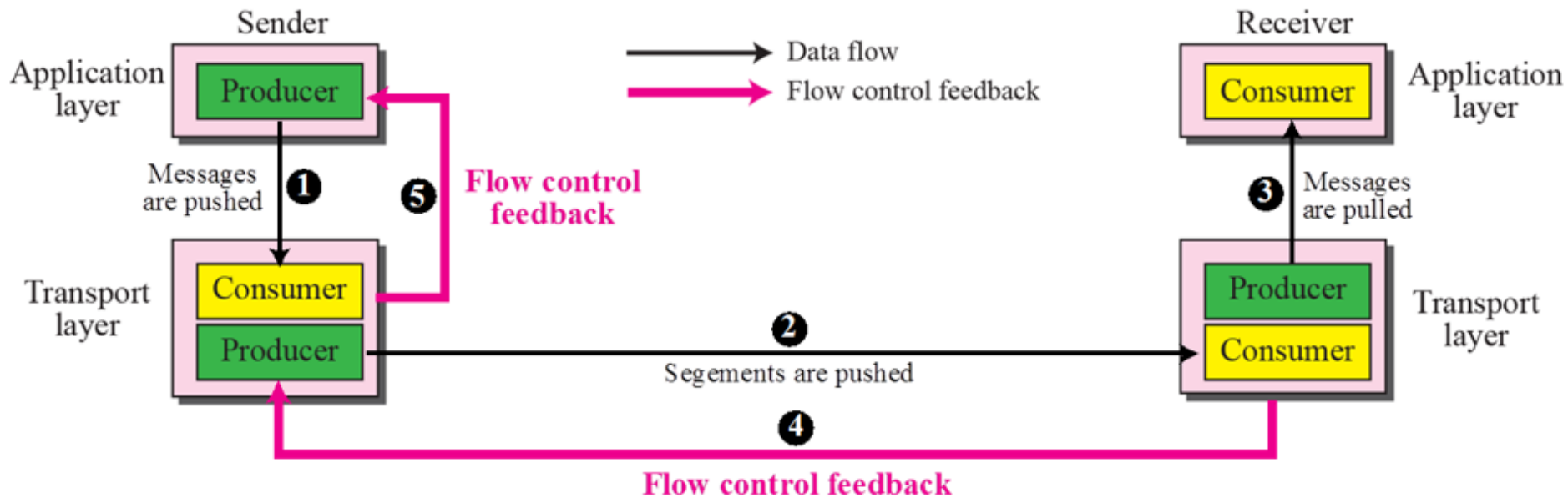
# FEC may be combined with ARQ

- Example with multicast, using *digital fountain* codes
  - Source has a file to transmit; it sends  $n$  packets
  - A destination that misses one packet sends a request for retransmission; source uses a fountain code and sends packet  $n+1$
  - If this or another destination still does not has enough, sources codes and sends packets  $n+2$ ,  $n+3$ , ... as necessary
  - All packets are different
  - Any  $n$  packets received by any destination allows to reconstruct the entire file
  - Used for data distribution over the Internet.

# 3. Flow Control

- In TCP, the sender window size is totally controlled by the receiver window value. However, the actual window size can be smaller if there is congestion in the network.
- *Some Points about TCP's Sliding Windows:*
  - ◆ The size of the window is the lesser of rwnd and cwnd
  - ◆ The source does not have to send a full window's worth of data.
  - ◆ The window can be opened or closed by the receiver, but should not be shrunk.
  - ◆ The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.
  - ◆ The receiver can temporarily shut down the window; the sender, however, can always send a segment of one byte after the window is shut down.
    - To prevent deadlock by proving

# Data Flow and Flow Control Feedbacks in TCP



# 3. Flow Control

- *Why* invented ?
  - Differences in machine performance: A may send data to B much faster than B can use. Or B may be shared by many processes and cannot consume data received at the rate that A sends.
  - Data may be lost at B due to lack of buffer space – waste of resources !
- *What* does it do ?
  - Flow control prevents prevent buffer overflow at receiver
- *How* does it work ?
  - Backpressure, or
  - Credits

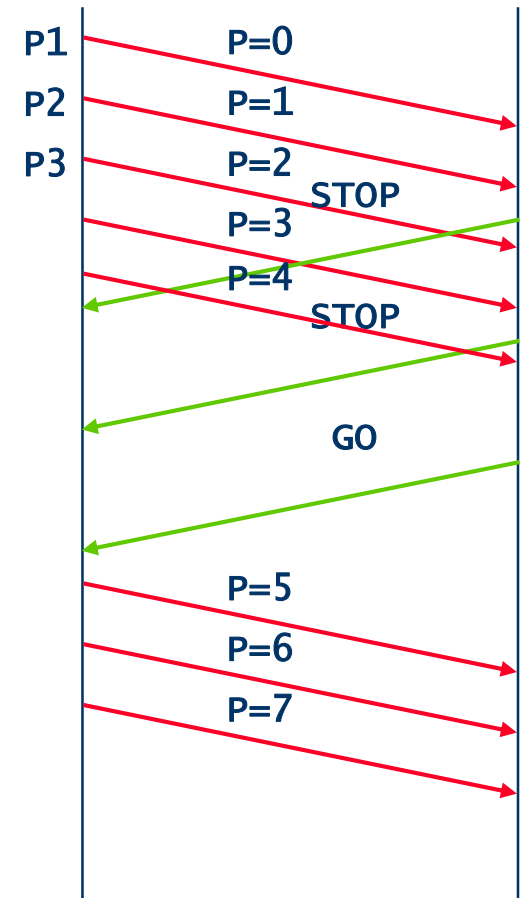


# Backpressure Flow Control

- Destination sends STOP (= PAUSE) or GO messages
- Source stops sending for  $x$  msec after receiving a STOP message
- Simple to implement
- Q. When does it work well ?

## solution

- Where implemented ?
  - X-ON / X-OFF protocols inside a computer
  - Between Bridges in a LAN
- Issues
  - Loops in feedback must be avoided (otherwise deadlock)

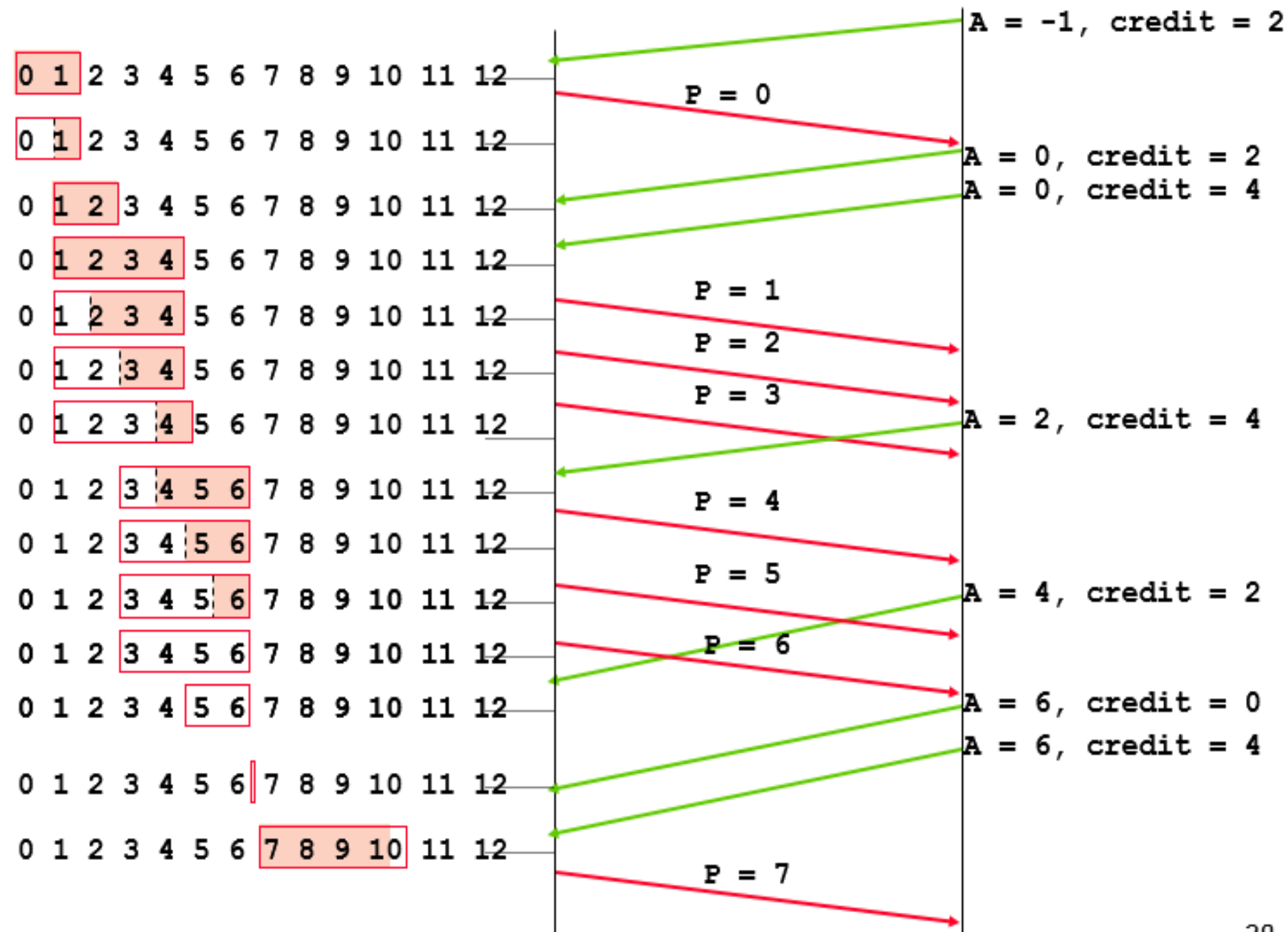


# Can we use Sliding Window for Flow Control ?

- One could use a sliding window for flow control, as follows
  - Assume a source sends packets to a destination using an ARQ protocol with sliding window. The window size is 4 packets and the destination has buffer space for 4 packets.
  - Assume the destination delays sending acks until it has enough free buffer space. For example, destination has just received (but not acked) 4 packets. Destination will send an ack for the 4 packets only when destination application has consumed them.

**Q.** Does this solve the flow control problem ?  
solution

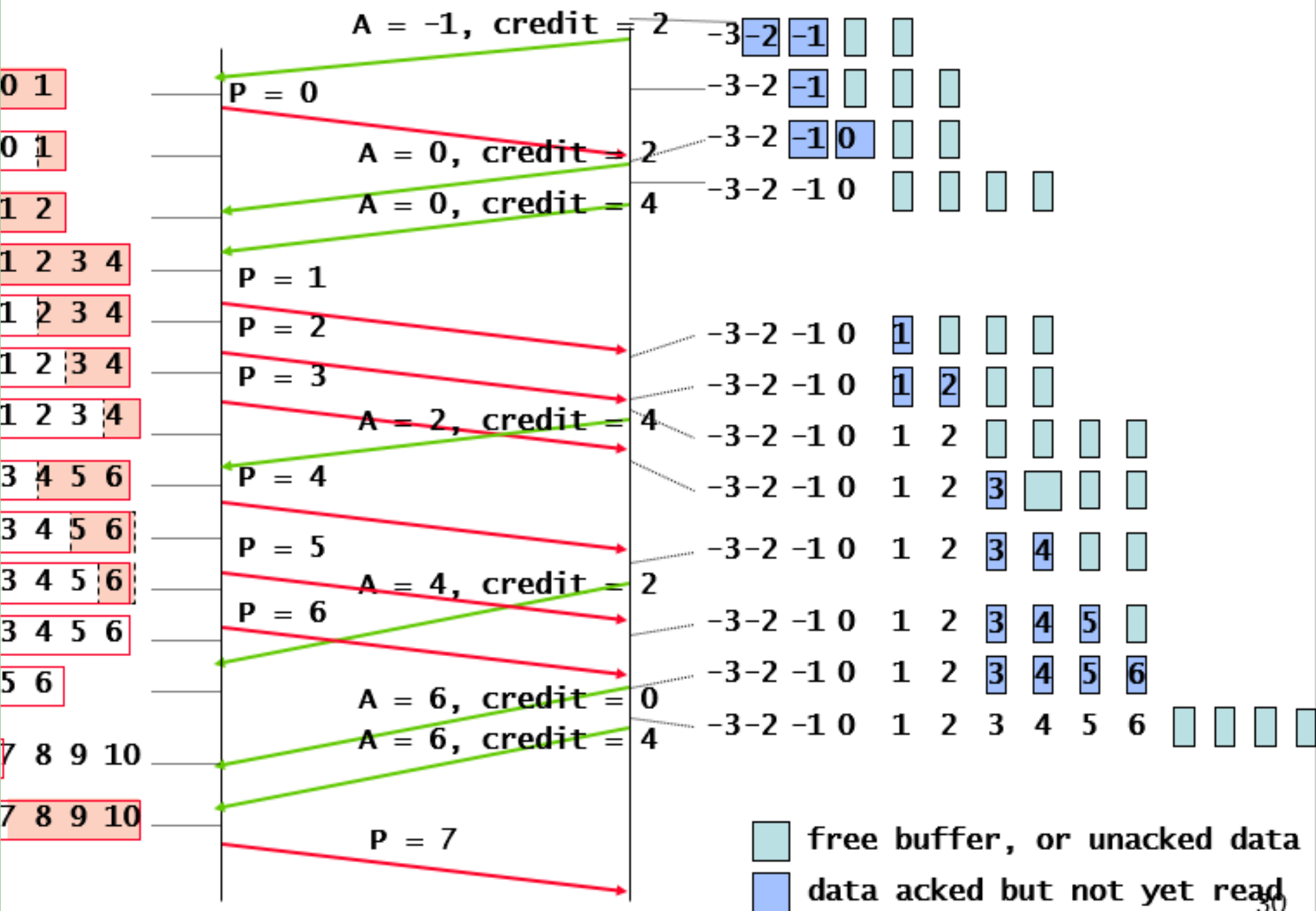
# Credit Flow Control



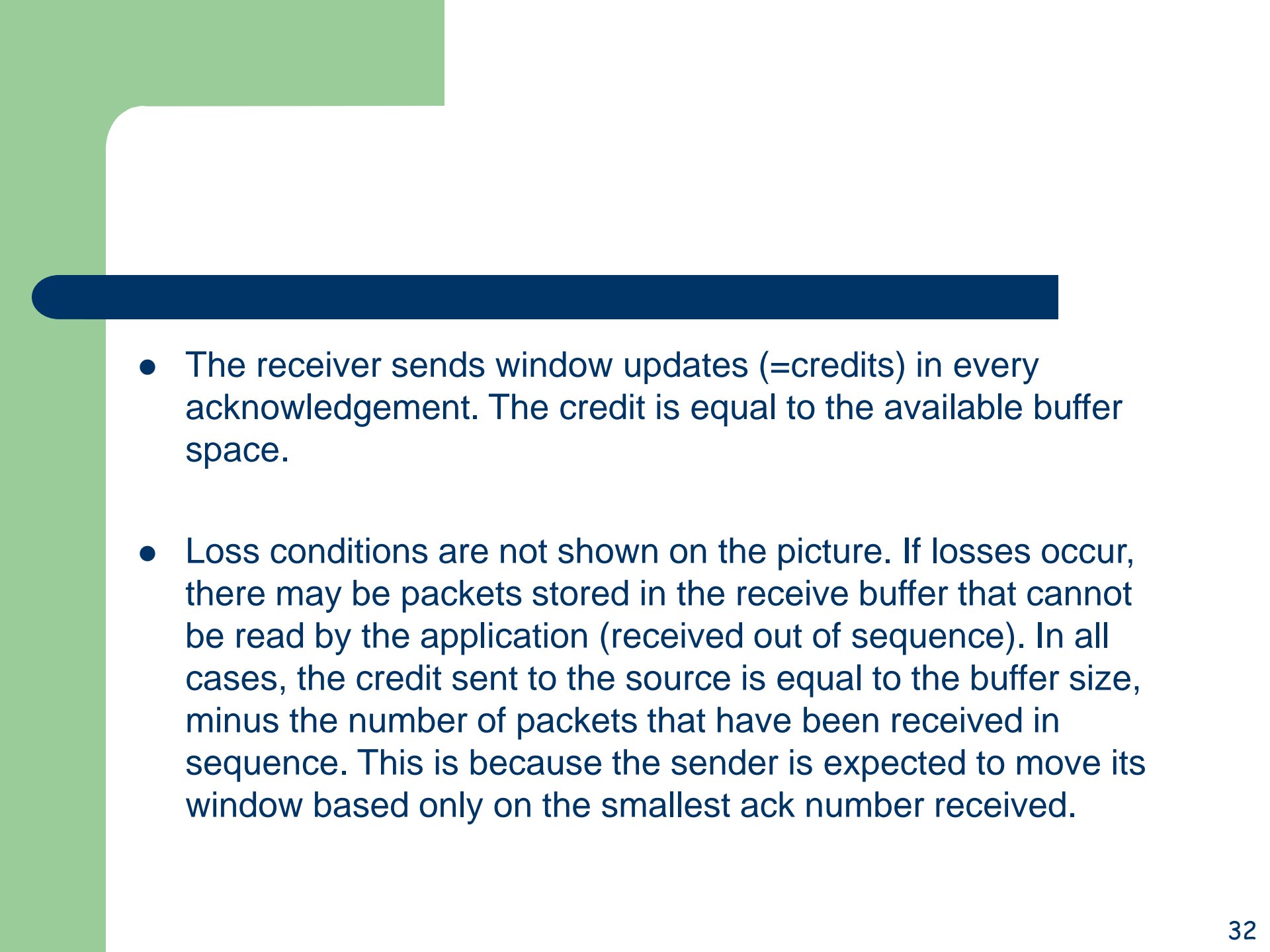
- The credit scheme solves the issue with using the sliding window alone for flow control. Credits are used by TCP, under the name of “window advertisement”.
- With a credit scheme, the receiver informs the sender about how much data it is willing to receive (and have buffer for). Credits may be the basis for a stand-alone protocol or, as shown here, be a part of an ARQ protocol. Credit schemes allow a receiver to share buffer between several connections, and also to send acknowledgements before packets are consumed by the receiving upper layer (packets received in sequence may be ready to be delivered, but the application program may take some time to actually read them).
- The picture shows the maximum send window (called “offered window” in TCP) (red border) and the usable window (pink box). On the picture, like with TCP, credits (= window advertisements) are sent together with acknowledgements. The acknowledgements on the picture are cumulative.

- Credits are used to move the right edge of the maximum send window. (Remember that acknowledgements are used to move the left edge of the maximum send window).
- By acknowledging all packets up to number  $n$  and sending a credit of  $k$ , the receiver commits to have enough buffer to receive all packets from  $n+1$  to  $n+k$ . In principle, the receiver(who sends acks and credits) should make sure that  $n+k$  is non-decreasing, namely, that the right edge of the maximum send window does not move to the left (because packets may have been sent already by the time the sdr receives the credit).
- A receiver is blocked from sending if it receives credit = 0, or more generally, if the received credit is equal to the number of unacknowledged packets. By the rule above, the received credits should never be less than the number of unacknowledged packets.
- With TCP, a sender may always send one byte of data even if there is no credit (window probe, triggered by persistTimer) and test the receiver's advertized window, in order to avoid deadlocks (lost credits).

## Credits are Modified as Receive Buffer Space Varies



- The figure shows the relation between buffer occupancy and the credits sent to the source. This is an ideal representation. TCP implementations may differ a little.
- The picture shows how credits are triggered by the status of the receive buffer. The flows are the same as on the previous picture.
- The receiver has a buffer space of 4 data packets (assumed here to be of constant size for simplicity). Data packets may be stored in the buffer either because they are received out of sequence (not shown here), or because the receiving application, or upper layer, has not yet read them.

- 
- The receiver sends window updates (=credits) in every acknowledgement. The credit is equal to the available buffer space.
  - Loss conditions are not shown on the picture. If losses occur, there may be packets stored in the receive buffer that cannot be read by the application (received out of sequence). In all cases, the credit sent to the source is equal to the buffer size, minus the number of packets that have been received in sequence. This is because the sender is expected to move its window based only on the smallest ack number received.