Data Communication
(DC)

Lecture 9a

Overview of the contents

- Simple protocol
- Stop-and-Wait protocol
- Go-Back-N protocol
- Selective-Repeat protocol
- Bidirectional protocol
- User datagram protocol

Simple protocol

Transport Layer

Transport layer protocols

To better understand the behavior of general transport layer protocols, we start with the simplest one and gradually add more complexity.

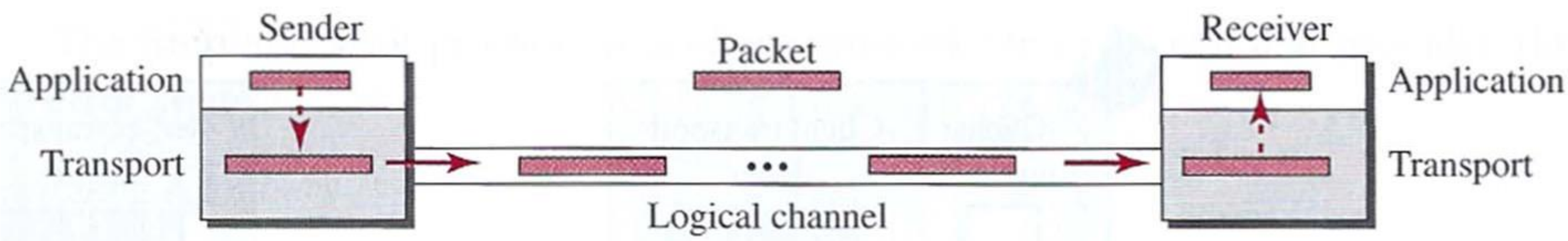
We first discuss all of these protocols as a unidirectional protocol (i.e., simplex) in which the data packets move in one direction, then discuss how they can be changed to bidirectional protocols where data can be moved in two directions (i.e., full duplex).

The TCP/IP protocol uses a transport-layer protocol that is either a modification or a combination of some of these general protocols.

Transport Layer
Simple protocol

This protocol has neither Flow nor Error control, where the packets move in one direction.

- assume the receiver can handle all packets it receive immediately.
- and it immediately removes packet header and passes data on to the application layer without any delay.
- In other words, we imagine that the receiver will never be overwhelmed with too much incoming data.

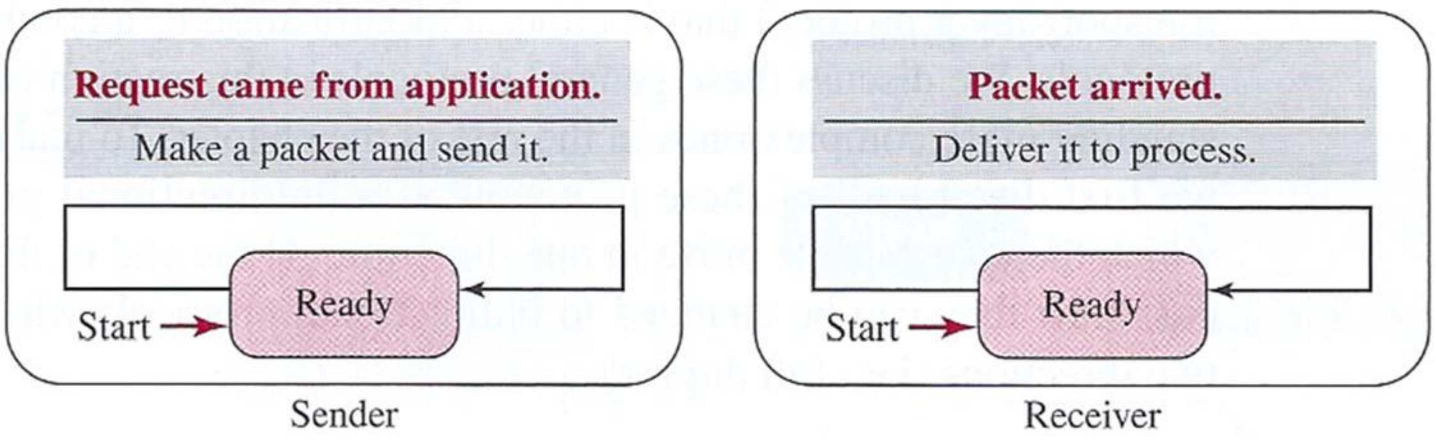


Transport Layer

Simple protocol

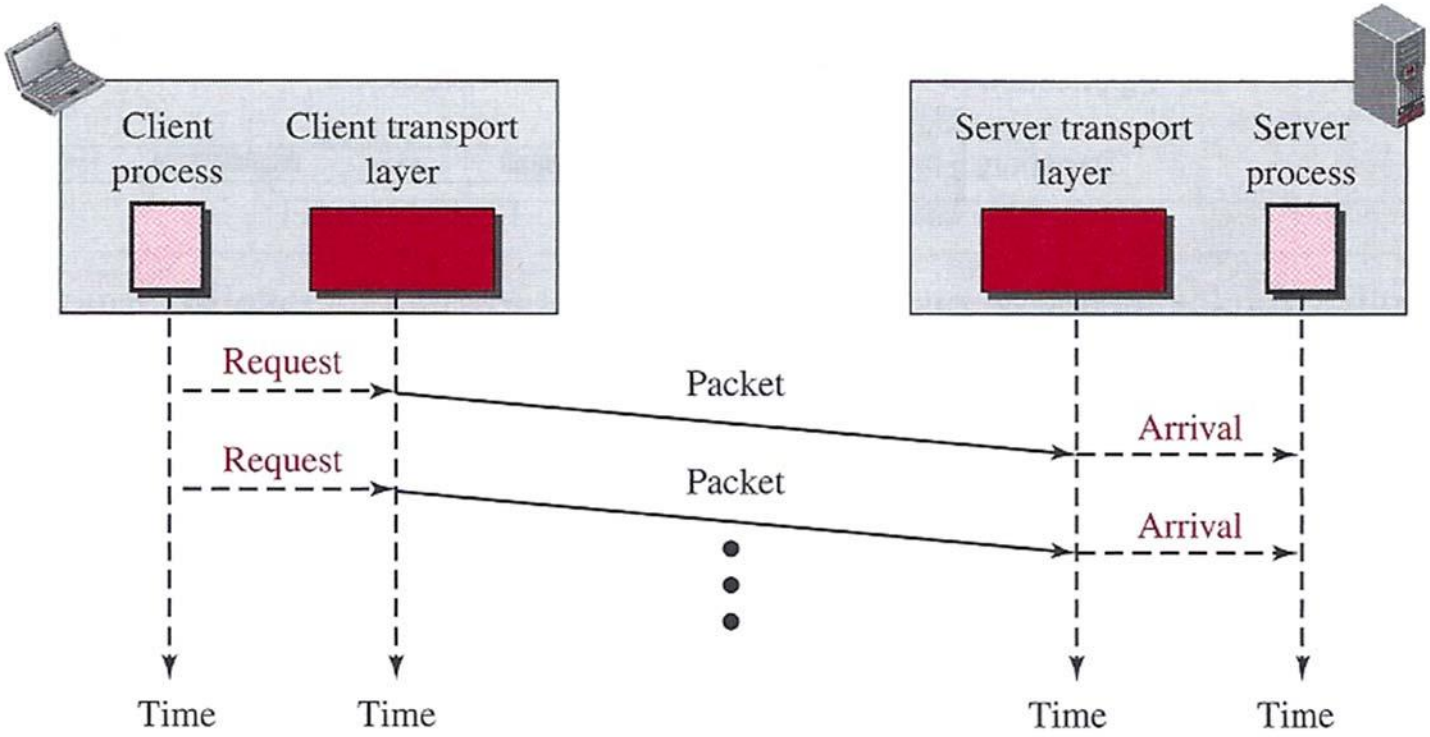
Here there is no need for flow control.

- The transport layer, on the sender side, gets its data from its application layer, encapsulates data and sends them (via its network layer).
- The transport layer, on the receiver side, receives packets (from its network layer), extracts data from it and passes it on to its application layer (a process).



Transport Layer

Simple protocol: flow diagram



The sender sends packets one after another without even thinking about the receiver. You can see the transmission time (the packet lines are a little inclined).

Stop-and-Wait protocol

Transport Layer

Stop-and-Wait Protocol

Here, a simple error control mechanism is added to the protocol, which makes it possible to find and correct errors.

In order to find and correct errors in the sent packets, we need to add redundant bits to our packets. (Checksum or CRC)

When a package is received, it is examined for errors

- If it is defective, it is simply discarded

When an error is found, cf. this protocol, this is manifested by a receiver remaining <u>silent</u>. (an acknowledgement will be sent back if no error)

Transport Layer

Stop-and-Wait Protocol

Packets lost during transmission are more difficult to handle

The received packet could be:

- Correct
- Duplicate
- Out of order (and we could not know the order)

The solution is to number the packets with sequence numbers,
then the recipient can determine if the received packet is in the right order or if it is out of order.

- If so, then other packets are either lost or duplicated.

Transport Layer

Stop-and-Wait Protocol

Corrupted or lost packages must be retransmitted, cf. this protocol.

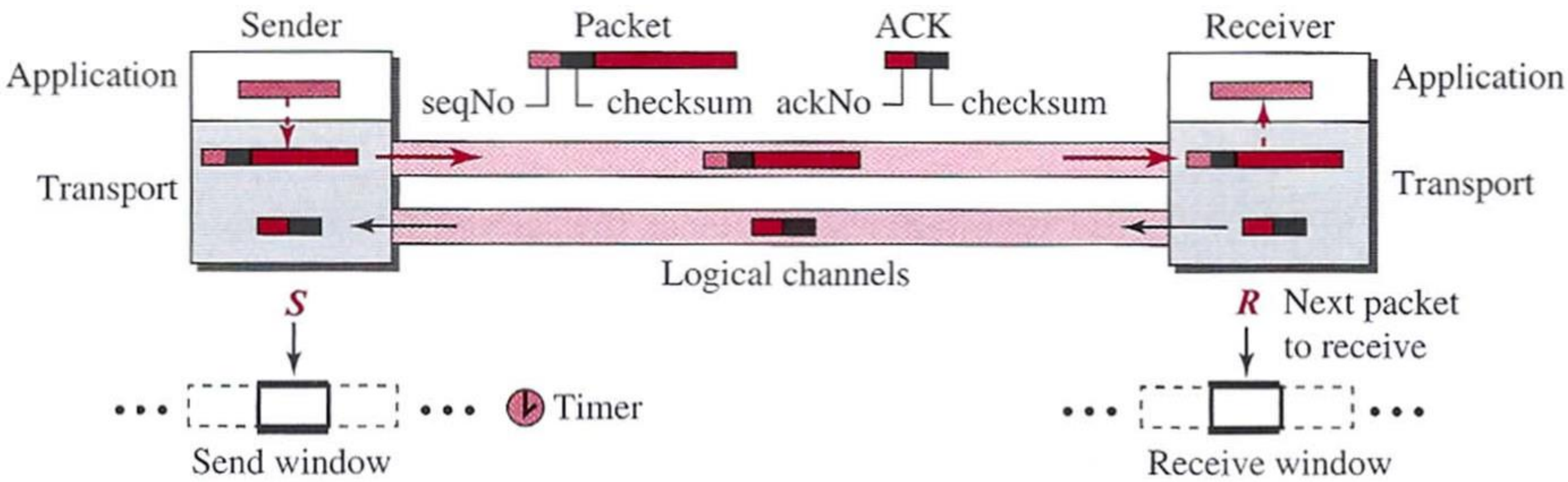
- If the receiver does not respond, then an error has occurred.
- The sender who saves a copy of the sent packet starts a timer at the time of dispatch.

If no ACK is received before the timer expires, the packet is retransmitted and the timer is restarted.

The copy is saved until an ACK is received before the timer expires.
Since an ACK packet can also be damaged and lost, they must also have redundant bits and a sequence number.

In this protocol, corrupted ACK packets are discarded and out-of-order ACK packets are ignored.

Transport Layer
Stop-and-Wait Protocol



The Stop-and-Wait protocol is a connection-oriented protocol that provides flow and error control. Note that only one packet and one acknowledgment can be in the channels at any time.

Transport Layer
Stop-and-Wait Protocol

Sequence numbering

An important consideration here is how many different sequence numbers we need to have in order to make a unique communication.

Sequence numbers can turn around.

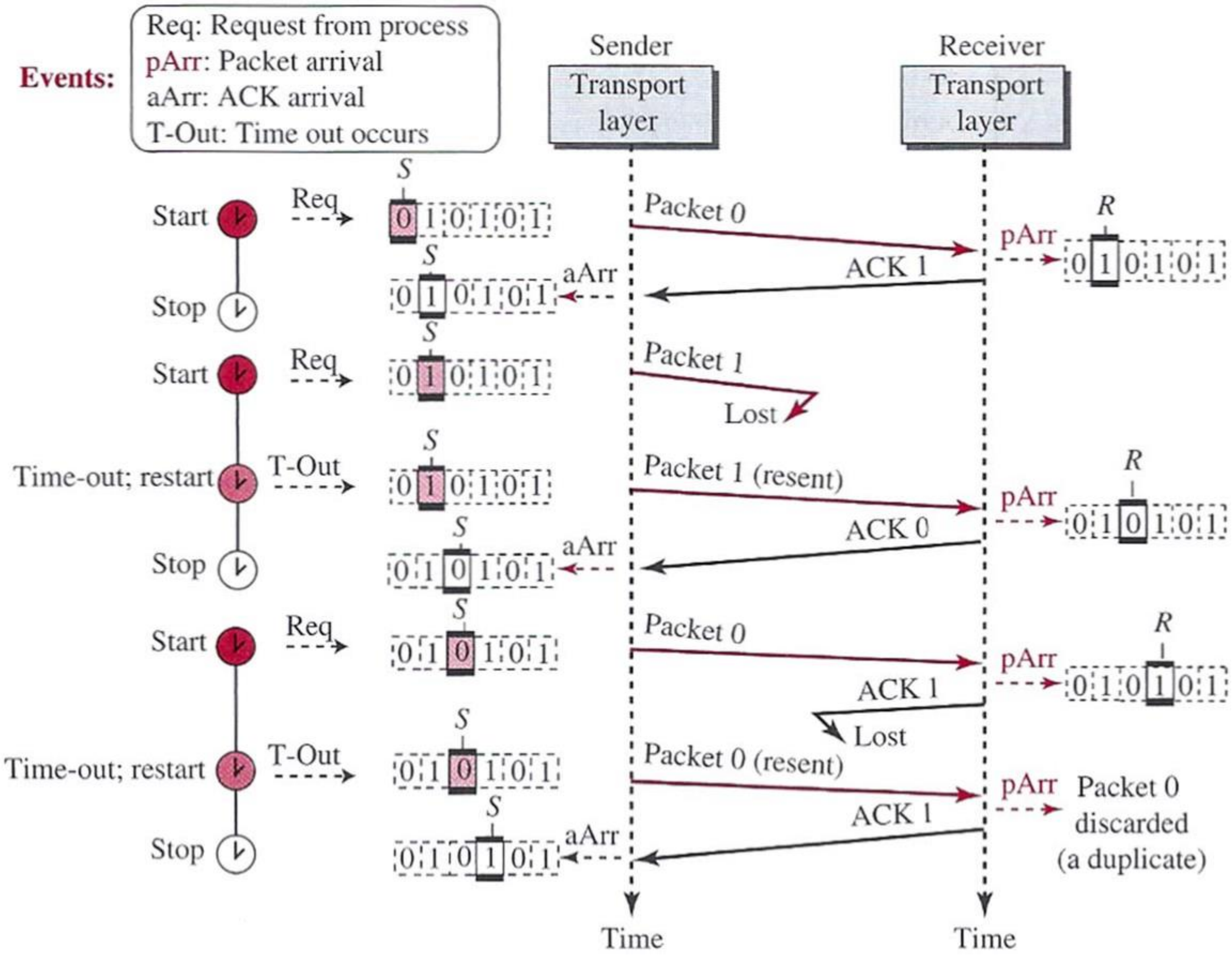
- This means that if you have a sequence number field of m bits, then sequence numbers can go from ***0*** to $\boldsymbol{2^m-1}$, then they are repeated.

In this protocol only 2 different sequence numbers (**1 bit**) are needed since receipt (ACK) for each packet is arrived before the next is sent.

An ACK packet, sent as a receipt for a correctly received packet, contains the sequence number of the next packet that the receiver expects to receive.

Transport Layer

Stop-and-Wait Protocol: Example



Note: the time-out value should be at least the time it takes to send the packet and receive an ACK

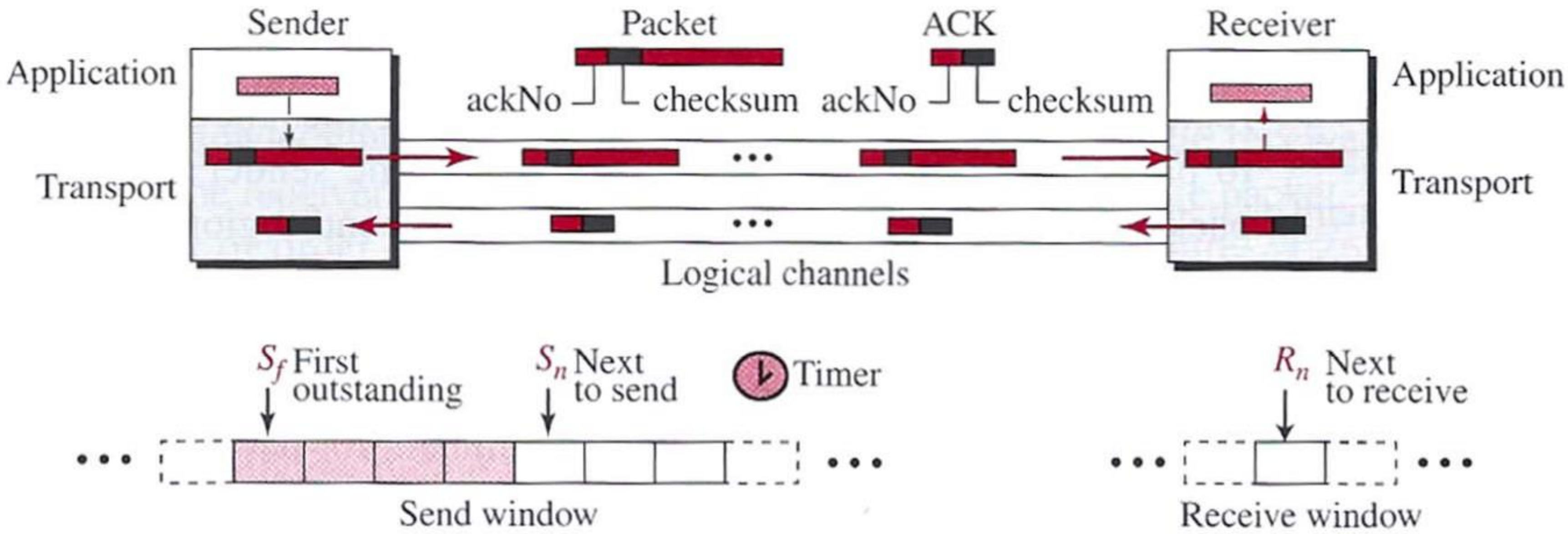
Go-Back-N protocol

Transport Layer

Go-Back-N Protocol

The problem with the previous protocol is that we only send one packet when we have received a receipt (ACK) that shows the previous packet has been received correctly.

Therefore, we are now expanding this protocol so that we can send more than one packets while the sender is waiting for acknowledgement. **In other words, multiple packets must be in transition to keep the channel busy.**



Transport Layer

Go-Back-N Protocol

Sequence numbering

Also, in this protocol we have a field in the header for numbering the packets, and here it also applies that we use modulus 2^m. Thus, the numbering for m bit, goes from 0 to 2^m-1.

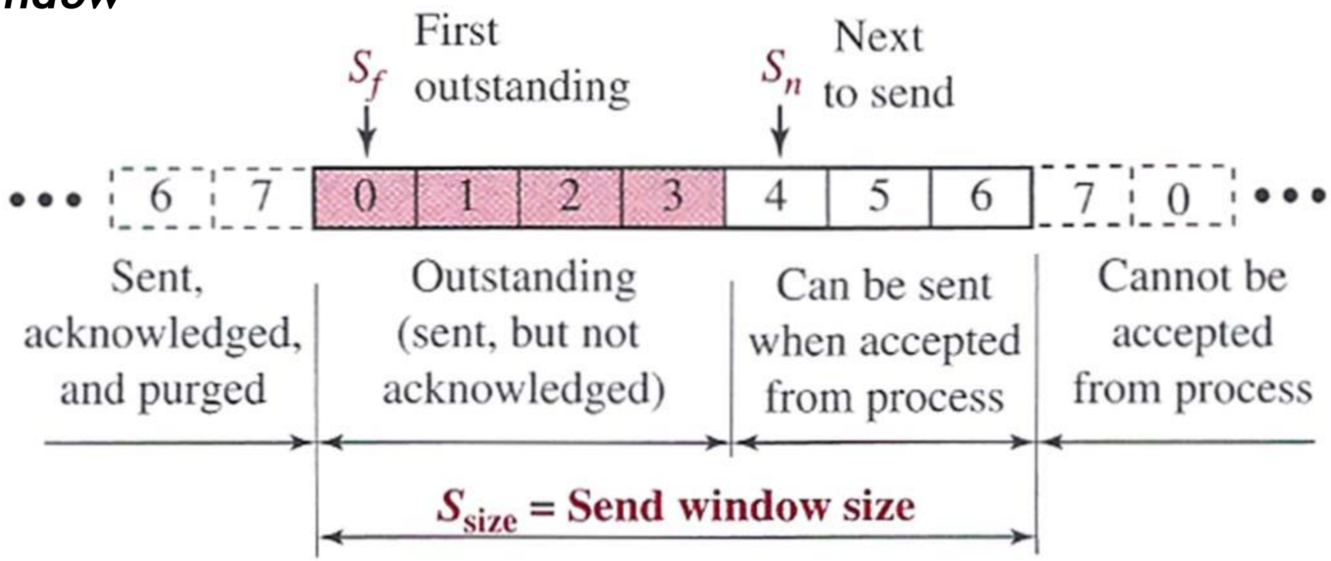
Sliding Window

Once we have determined a number of sequence numbers we will apply, then it is important to understand that we can only use less than 2^m numbers in a sender sliding window, which means that the sender can only have a limited number of ***outstanding packets***, i.e., sent packets that have not yet been acknowledged.

Transport Layer

Go-Back-N Protocol

Sender window

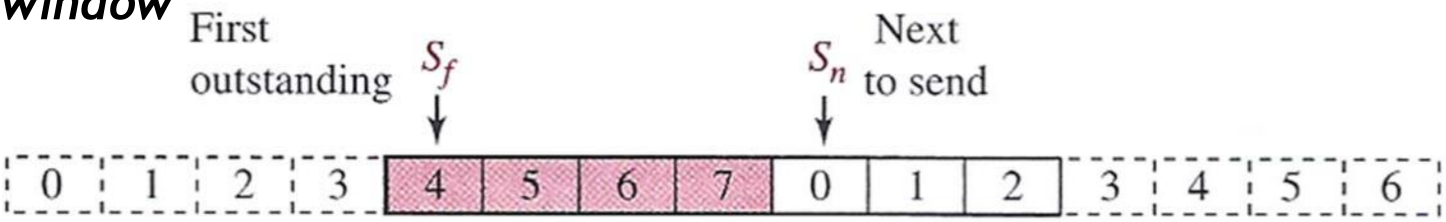


- The packets acknowledged (far left), the sender no longer has copies of them.
- The packets in the dark field, the sender needs to have copies of them, as their fate is not known yet, we do not know if they should be resent later yet.
- To the right of the dark block, the range of sequence numbers for packets that can be sent. However, the corresponding data have not yet been received from the application layer.

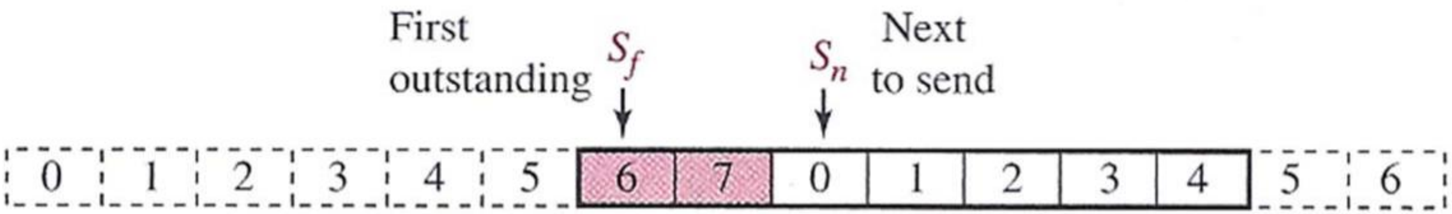
Transport Layer

Go-Back-N Protocol

Sender window



a. Window before sliding



b. Window after sliding (an ACK with ackNo = 6 has arrived)

variable **Sf** defines the sequence number of the first (oldest) outstanding packet.
variable **Sn** holds the sequence number that will be assigned to the next packet to be sent.

The size of the window thus depends on how many packets the sender can keep copies of.

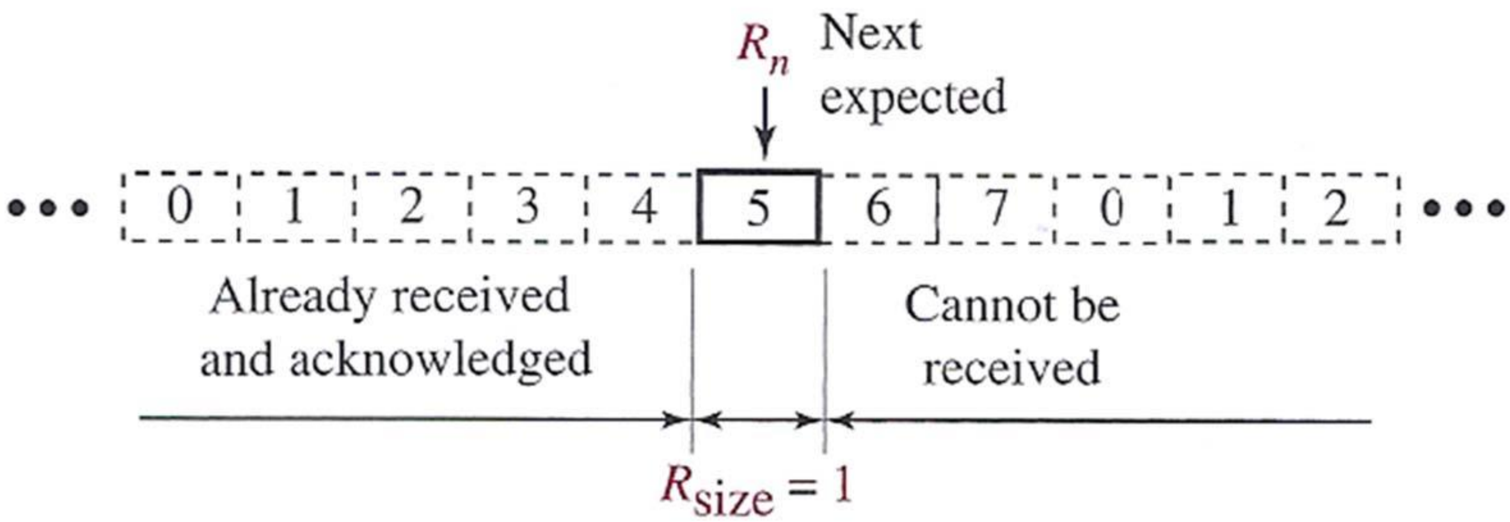
Transport Layer

Go-Back-N Protocol

Receiver window

The receiver's window is of **only size 1**.

The receiver only waits for a specific packet (here 5), all other received packets are discarded and must be retransmitted.



Transport Layer
Go-Back-N Protocol

Timers

Only one timer is used here.

The timer is set all the time so that it reflects the time of dispatch of the oldest packet (the first outstanding packet) that has not yet been acknowledged.

If the timer expires, all outstanding packets are retransmitted.

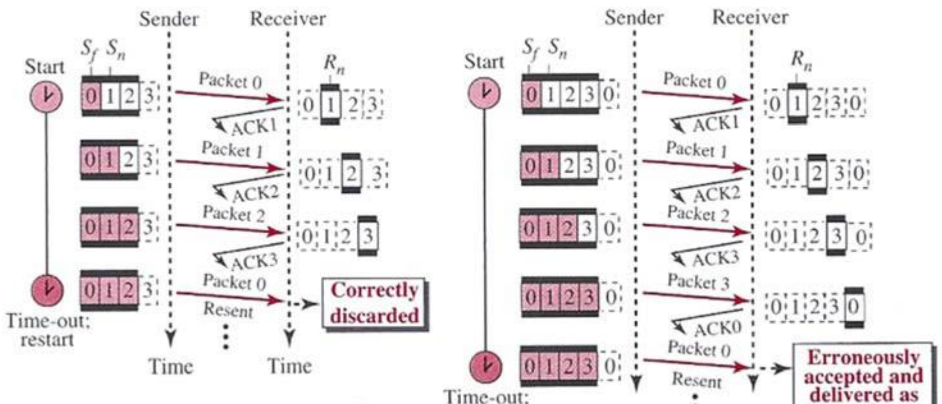
That is why the protocol is called Go-Back-N.
On a time-out, the machine goes back N locations and resends all packets.

Transport Layer

Go-Back-N Protocol

Sender window size

Now we need to see why the window should be less than ***2^m***.



a. Send window of size < 2^m

b. Send window of size = 2^m

Example

We choose **m=2**.

The size of the window is selected: 2^m-1 = **3**.

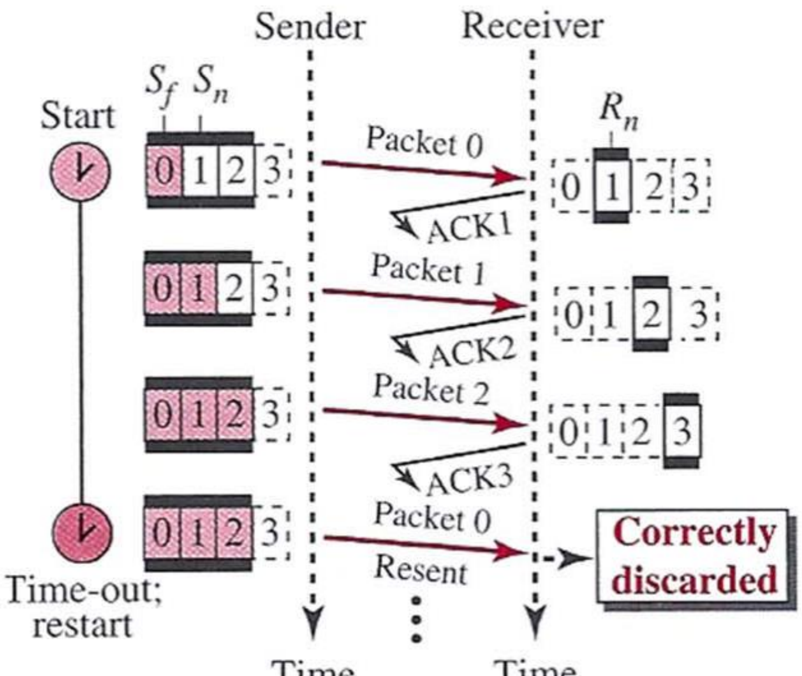
In the figure (full size next slide), a window size of 3 (***a***) compared against with one of 4 (***b***).

- If the window is 3 (i.e., less than 2^2) and all receipts are lost, then the timer for packet 0 will expire and all three packets will be retransmitted. Nothing special would happen.
- If the window is 4 (i.e., equal to 2^2) and all receipts are lost, then the timer for packet 0 will expire and all four packets will be retransmitted.
  But at that point, the receiver expects packet 0 and mistakenly accepts the resent packet 0 as a new packet 0.

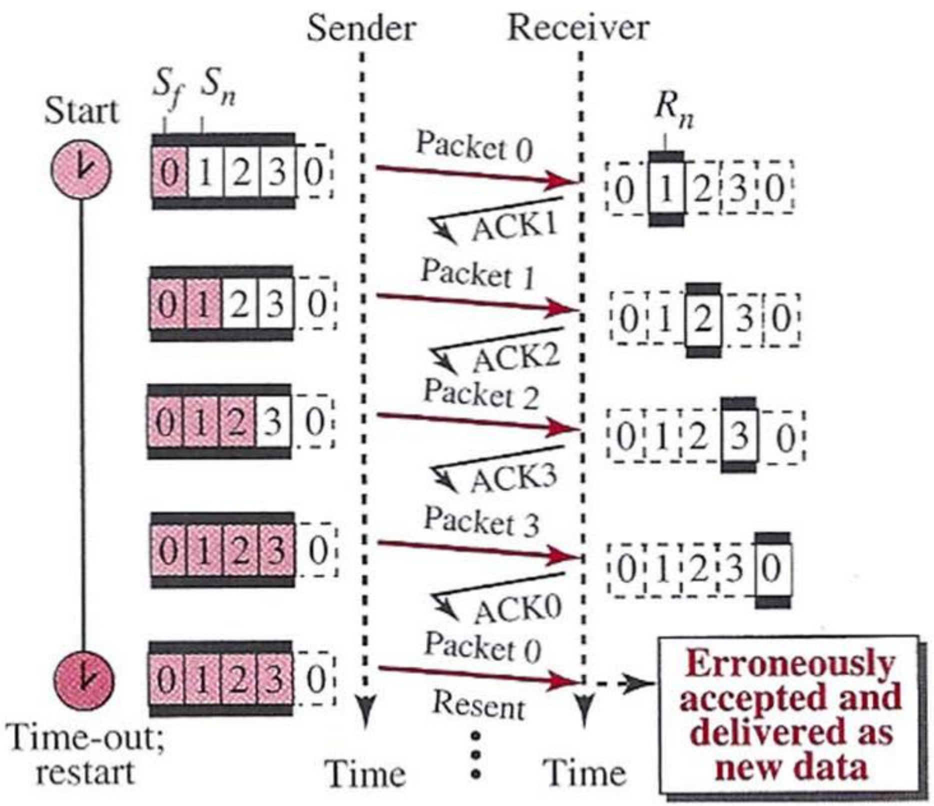
Transport Layer

Go-Back-N Protocol

Sender window size



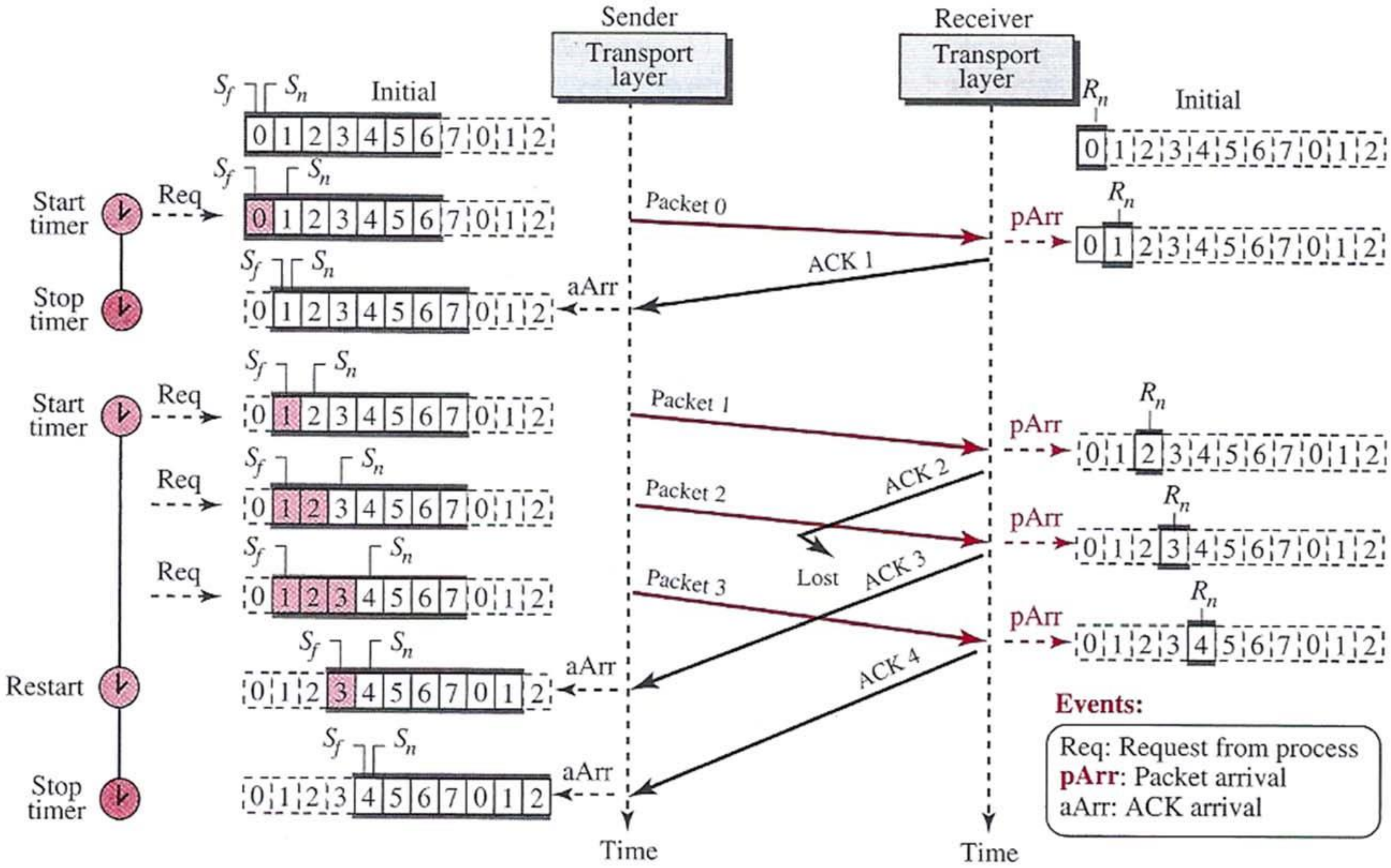
a. Send window of size < 2^m



b. Send window of size = 2^m

Transport Layer

Go-Back-N Protocol: example



Notice: cumulative ACK can help here.

Communication may continue even if acknowledgments are delayed or lost.

That is, although ACK 2 is lost, ACK 3 is cumulative and serves as both ACK 2 and ACK 3.

Transport Layer

Go-Back-N Protocol: Example

The sender sends packets 0, 1, 2 and 3 is sent, but packet 1 is lost.

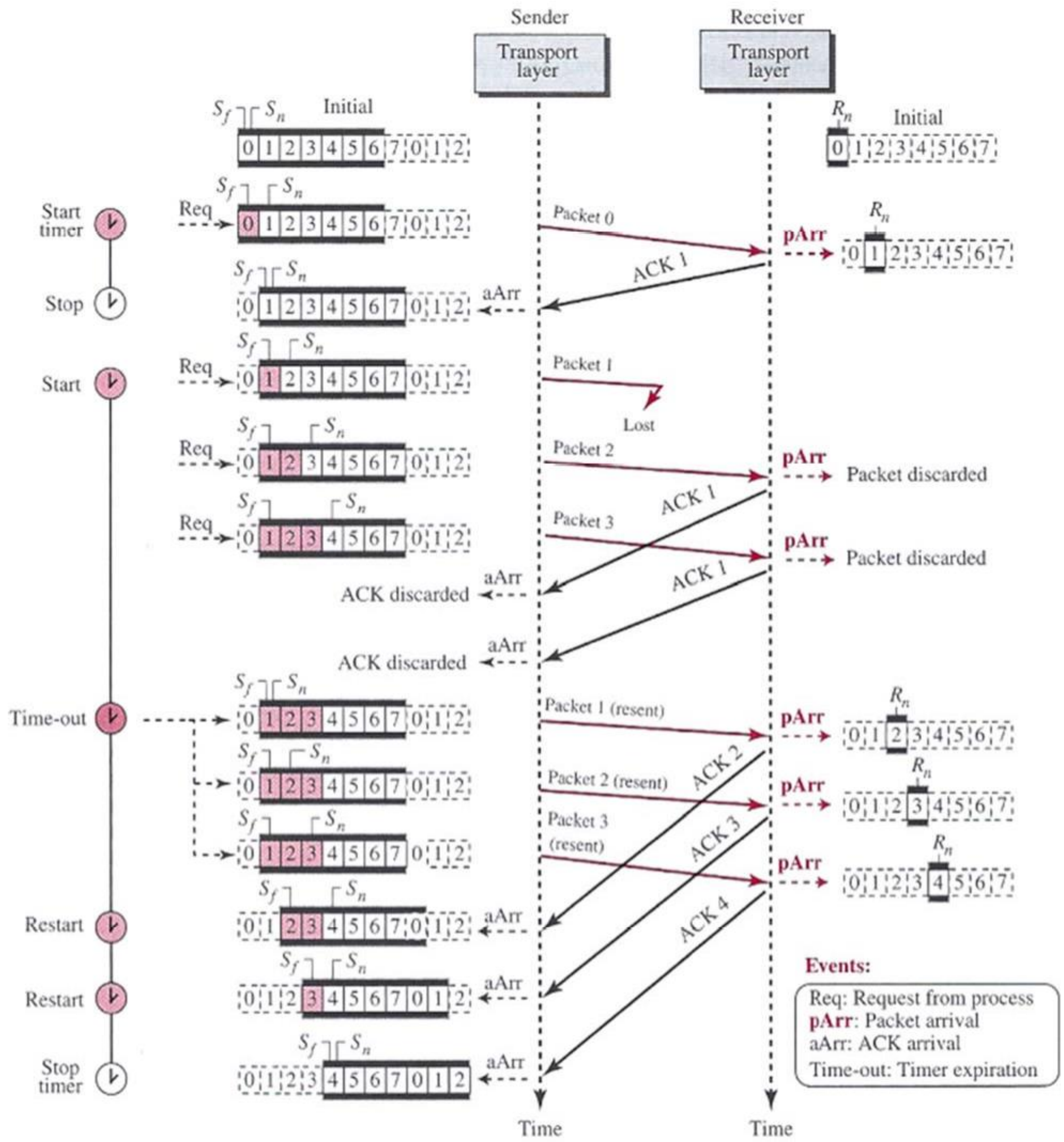
The receiver receives packets 2 and 3, but they are discarded as they are received out-of-order.

The receiver sends an ACK on packet 1 both when packets 2 and 3 are received, as it is packet 1 it expects.

The sender discards these ACKs as the ACK number is equal to S_f and not greater than it.

Time-out on packet 1 occurs and packets 1, 2 and 3 are retransmitted.

The received packets are acknowledged (and the timer eventually stops)



Transport Layer

Go-Back-N Protocol

Note that: ***Stop-and-Wait* is a special case of *Go-Back-N*.**

Stop-and-Wait has only two sequence numbers,
while ***Go-Back-N*** has several.

In addition, the sender window size in ***Stop-and-Wait*** is only 1.
in ***Go-Back-N***, it is usually more than 1.

Selective-Repeat protocol

Transport Layer
Selective-Repeat Protocol

The Go-Back-N protocol simplifies the process on the receiver side.

Here you only need to keep track of one variable, and you do not need any buffers to keep track of out-of-order packets, they are simply discarded.

All these nice things have their "price": *This type of protocol is very inefficient when the network layer loses many packets*.

The Selective Repeat protocol resend only selective packets that are actually lost, not all the outstanding packets.

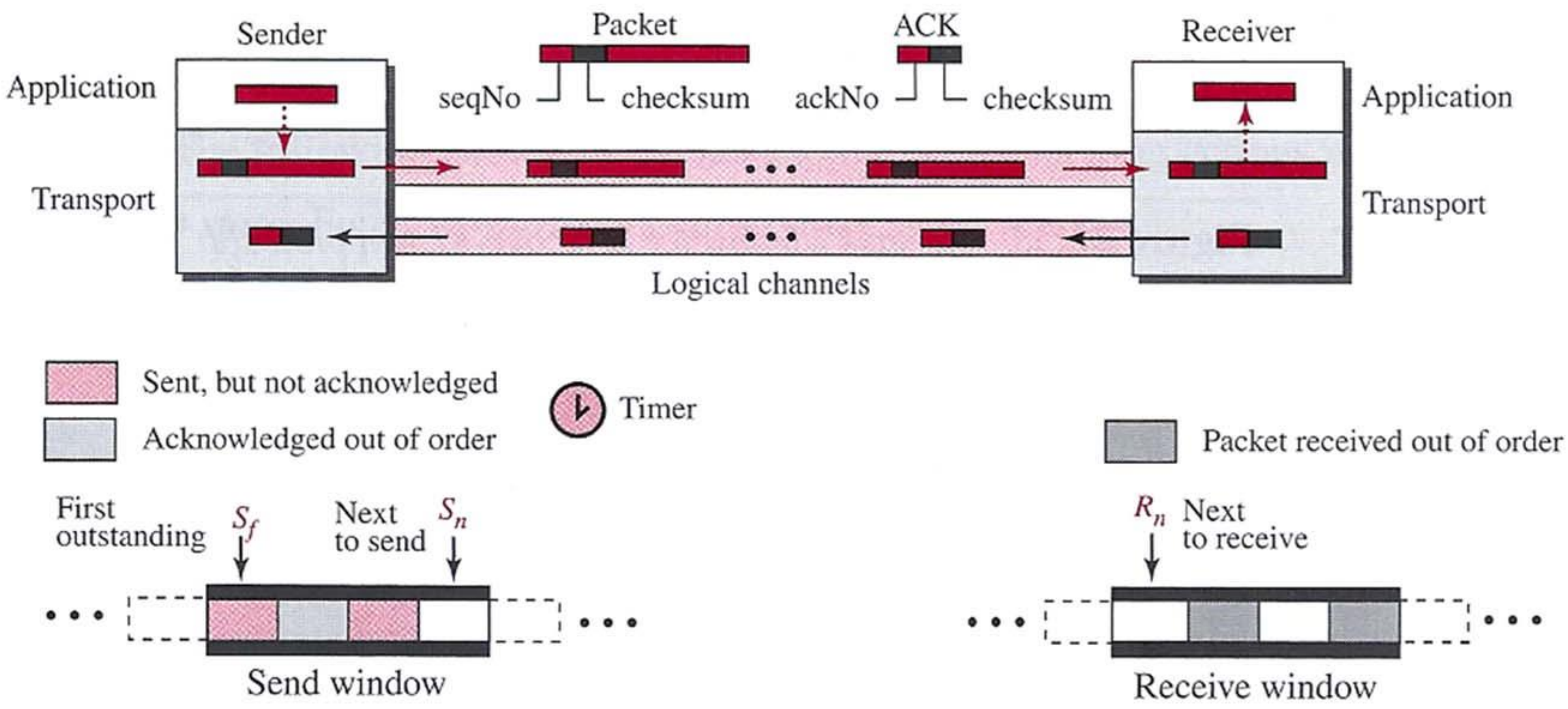
This is more efficient, but it also means that the receiver becomes more complex.

Transport Layer

Selective-Repeat Protocol

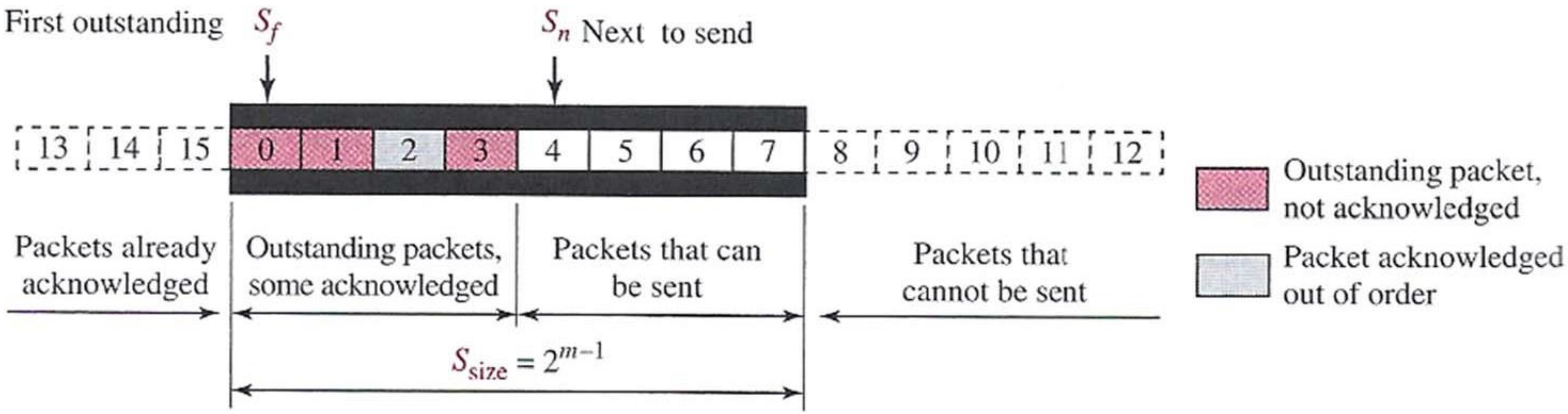
On the sender side, the window is similar to the one from the Go-Back-N, it's just smaller. Later we will see why the size is smaller.

The receiver window in Selective-Repeat is totally different from the one in Go-Back-N. The size of the receiver window is the same as the size of the sender window.



Transport Layer
Selective-Repeat Protocol

Sender window



On the sender side, the window has become a little more complex

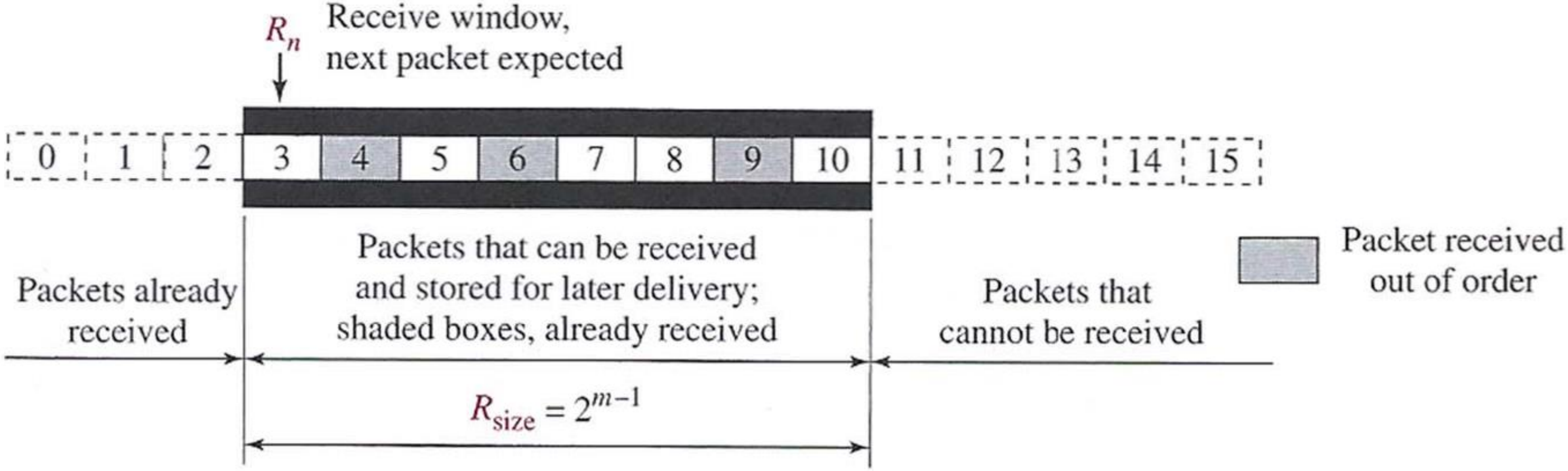
- **The red boxes** are packets for which the receipts (ACK) have not yet been received.
- **The gray box** is a packet for which the receipt (ACK) has been received (but out of order).

When packets 0 and 1 are acknowledged, the window will slide by three boxes to the right.

Transport Layer

Selective-Repeat Protocol

Receiver window



On the receiver side, the window has become more complex.

- The gray boxes are packets received out-of-order.
- The white boxes are the expected packets.

In this case, the window will slide by two boxes to the right when the packet 3 is received, because then packets 3 and 4 will be in order, and they can thus be delivered on up to the application layer.

Transport Layer

Selective-Repeat Protocol: Example

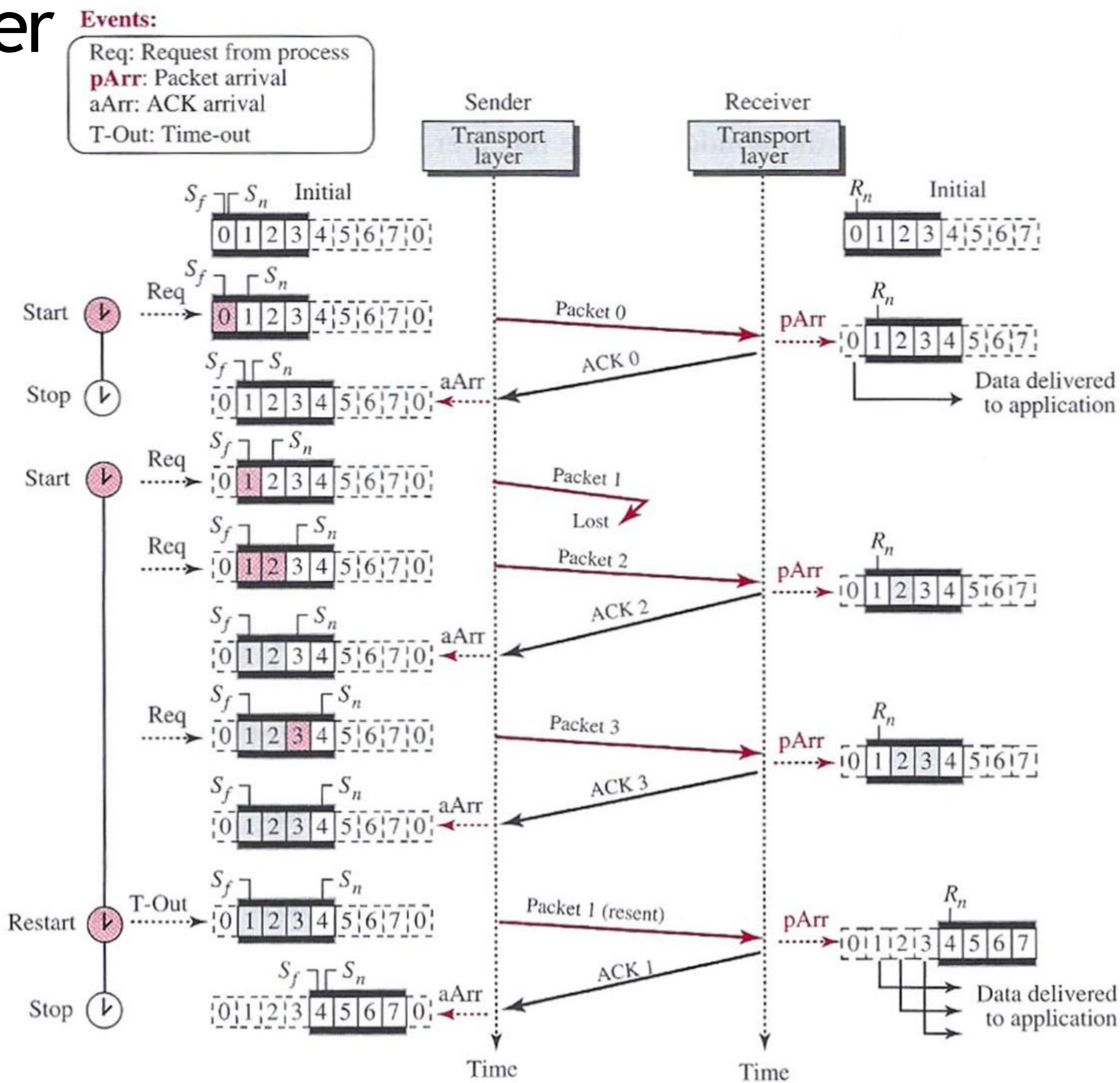
Packet 0 is sent and acknowledged

Packet 1 is lost, packets 2 and 3 are received out-of-order (and acknowledged).

Note: the timer runs for packet 1, as it is the earliest packet that has not been acknowledged.

When the timer expires, packet 1 is resent, and the receiver acknowledges this.

Packets 1, 2 and 3 can be delivered to the application layer in order.



Transport Layer

Selective-Repeat Protocol

The size of the window

We will now show why the size of the window may only be up to half of $\boldsymbol{2^m}$
($\boldsymbol{m}$ is the number of bits in the sequence numbering)

E.g., **m = 2** (numbers from 0 to 3), which means that the maximum window size is $2^m/2 = \mathbf{2}$.

The figure in the next slide compares a window of size 2 and another one of size 3.

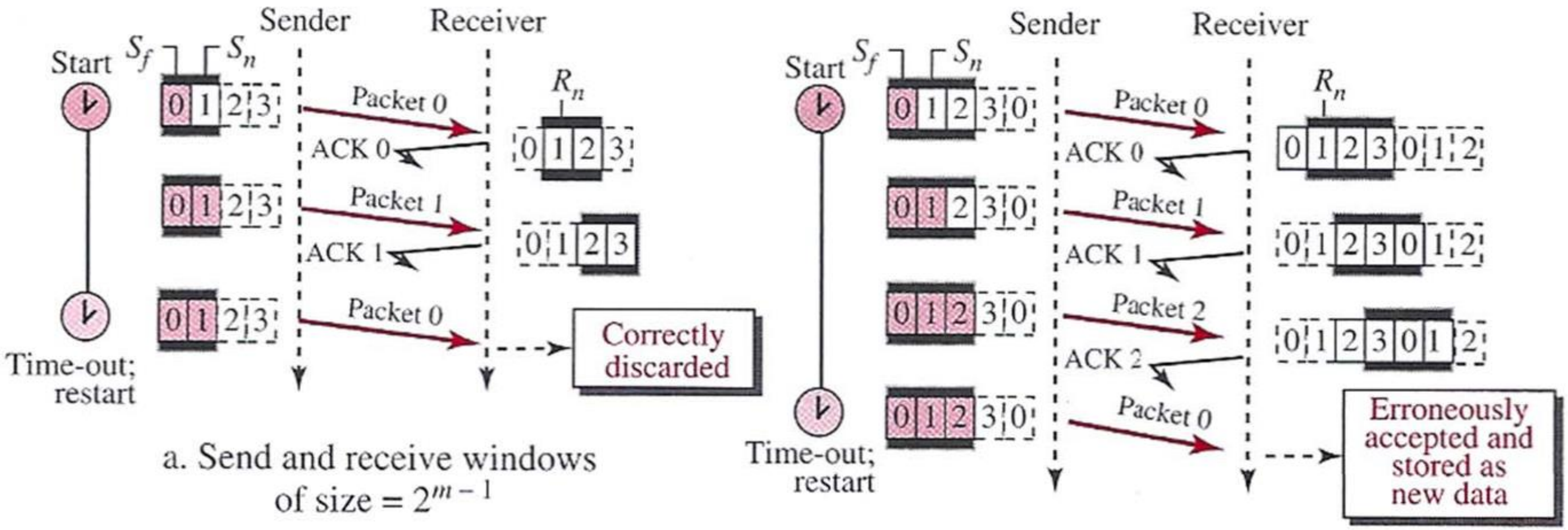
- If the window size is 2 and all ACKs are lost, then timer 0 will expire and packet 0 will be resent. But the receiver is waiting for packet 2, not packet 0, so this duplicate is discarded. ***Which is correct.***

- If the window size is 3 and all ACKs are lost, then timer 0 will expire and packet 0 will be resent. The receiver is waiting for a new packet 0 rather than the old packet 0 which is resent. The receiver mistakenly accepts this duplicate. ***Which is wrong.***

Transport Layer

Selective-Repeat Protocol

The size of the window



a. Send and receive windows of size = 2^{m-1}

b. Send and receive windows of size > 2^{m-1}

The sequence number 0 is not in the window

The sequence number 0 is part of the window

In Selective Repeat ARQ, window sizes can only be half of the number of sequence numbers (2^m)

Bidirectional protocol

Transport Layer

Bidirectional protocols: Piggybacking

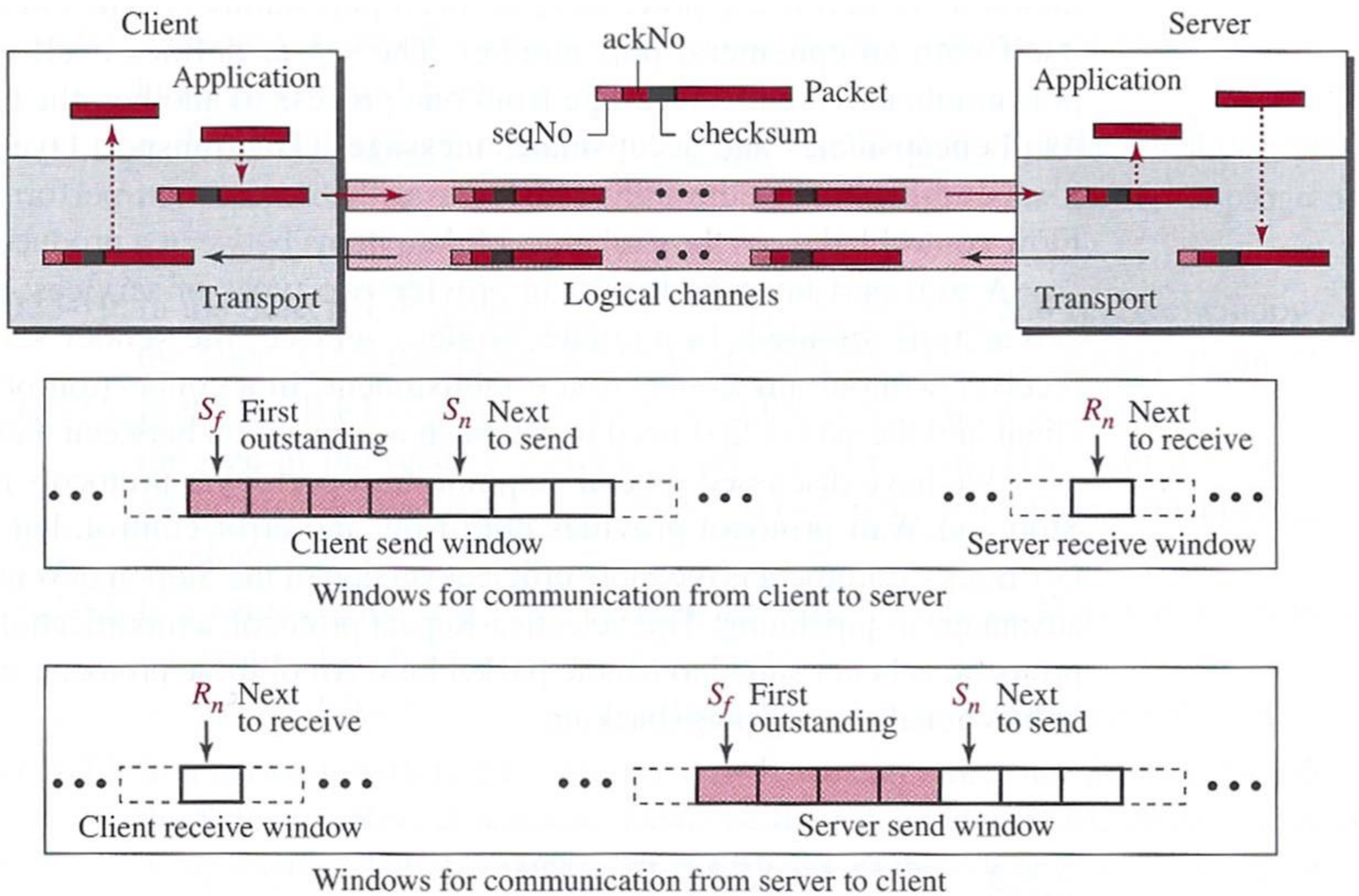
All the protocols we have viewed here are **unidirectional** protocols.

But in the real world, **bidirectional** communication is usually used.

- Therefore, receipts (ACK) to be returned to a sender are embedded in the packets sent the other way around, and vice versa.
- Thus, a packet header will contain both a sequence number for what is sent and an ACK number for what is received.

Transport Layer

Bidirectional protocols: Piggybacking



Piggybacking is shown here with **Go-Back-N**.

The client and server each use two independent sender and receiver windows.

Transport Layer

Protocols

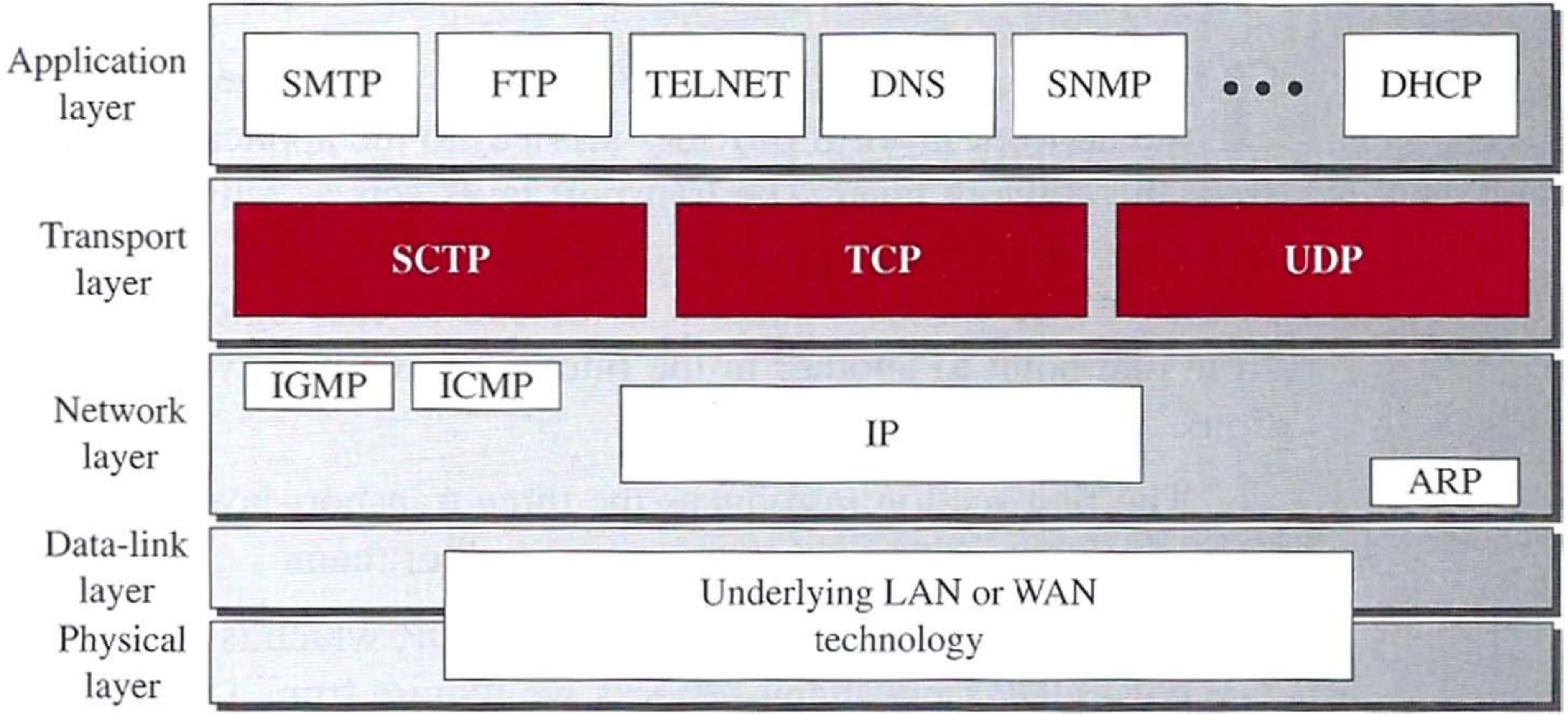
One of the questions you might ask yourself is:
If the Data Link layer is reliable and offers flow and error control (as we have seen that before), then do we also need to have flow and error control on the Transport layer?

The answer is YES !
Reliability on the Data Link layer is between two nodes!
We also need reliability between two ends (process to process)!

Transport Layer

Protocols

Now we have viewed some of the general principles that underlie the Transport Layer. We will now take a closer look at the specific protocols available in the TCP/IP Protocol Suite.



Transport Layer

Protocols

In the TCP/IP protocol suite there are three common transport layer protocols:

- User Datagram Protocol (**UDP**) which is:
  - **Unreliable**
  - **Connectionless**

  This protocol is simple and efficient.
  It is used by applications that can add error control
  (at the application layer level).

- Transmission Control Protocol (**TCP**) which is:
  - **Reliable**
  - **Connection-oriented.**

  This protocol can be used by all applications where reliability is important.
  The application does not have to provide error control itself,
  (it is at Transport layer level)

- Stream Control Transmission Protocol (**SCTP**) which is a reliable and connection-oriented.
  A new protocol that combines the good features of UDP and TCP.

Transport Layer
Port numbers

Port	Protocol	UDP	TCP	SCTP	Description
7	Echo	√	√	√	Echoes back a received datagram
9	Discard	√	√	√	Discards any datagram that is received
11	Users	√	√	√	Active users
13	Daytime	√	√	√	Returns the date and the time
17	Quote	√	√	√	Returns a quote of the day
19	Chargen	√	√	√	Returns a string of characters
20	FTP-data		√	√	File Transfer Protocol
21	FTP-21		√	√	File Transfer Protocol
23	TELNET		√	√	Terminal Network
25	SMTP		√	√	Simple Mail Transfer Protocol
53	DNS	√	√	√	Domain Name Service
67	DHCP	√	√	√	Dynamic Host Configuration Protocol
69	TFTP	√	√	√	Trivial File Transfer Protocol
80	HTTP		√	√	HyperText Transfer Protocol
111	RPC	√	√	√	Remote Procedure Call
123	NTP	√	√	√	Network Time Protocol
161	SNMP-server	√			Simple Network Management Protocol
162	SNMP-client	√			Simple Network Management Protocol

User datagram protocol

Transport Layer

User Datagram Protocol (UDP) [Protocol 17]

This protocol is characterized as follows:

- Connectionless
- Unreliable communication

It does not really add any extra functionality compared to the IP protocol.

However, it offers process-to-process communication instead of host-to-host communication. There is also a very limited error control (through checksum).

Do you want to use UDP even though it is so limited? The answer is YES!

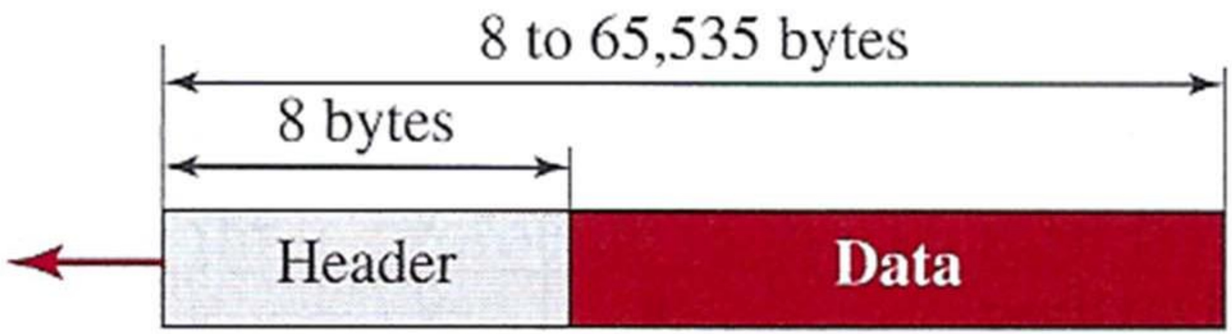
UDP is simple, it generates a <u>minimum of overhead</u>.

It requires <u>less sender-receiver interaction</u> than TCP or SCTP.

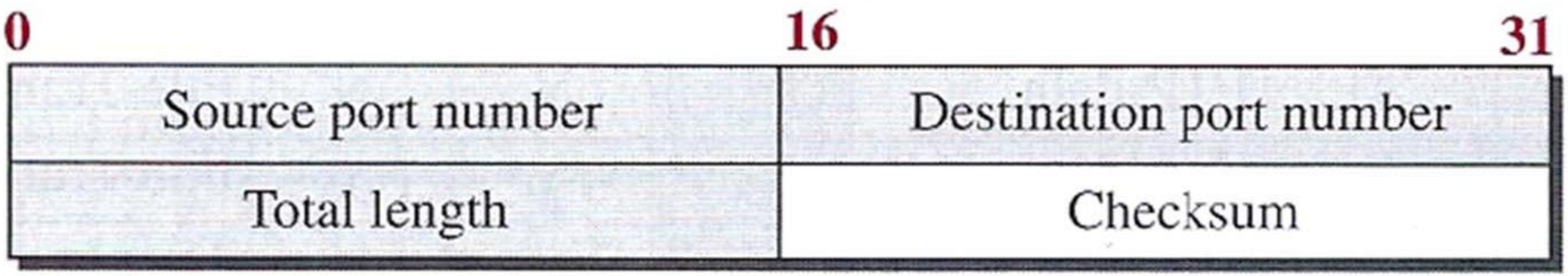
Transport Layer

User Datagram Protocol (UDP) [Protocol 17]

UDP packets (also called **User Datagrams**) have a fixed size of their header of ***8 bytes***.



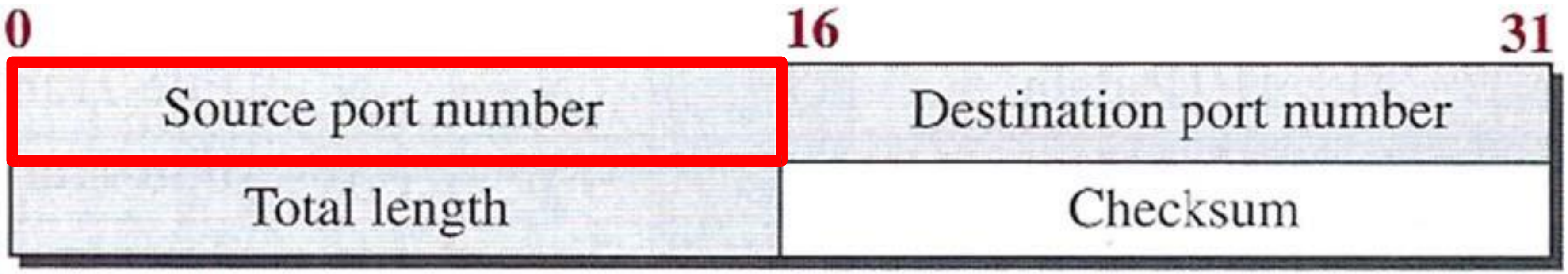
a. UDP user datagram



b. Header format

Transport Layer

User Datagram Protocol (UDP) [Protokol 17]



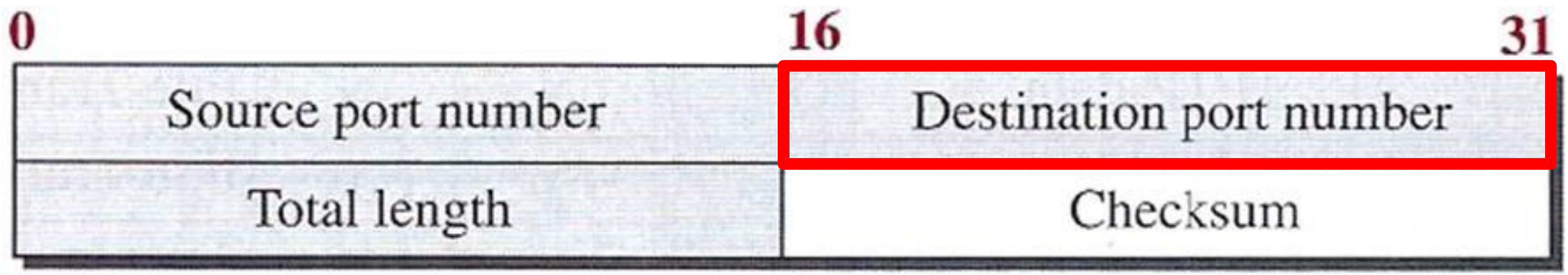
Sender port number: This port number is used by the sender process that is on the sender host.
The field has a length of *16 bit* (i.e., 65,536 different ports).

If the sender host is a client (a client usually sends requests), then the port number is usually a temporary (dynamic) port number (ranging from **49,152 - 65,535**).

If the sender host is the server (a server usually sends responses), then the port number is usually a well-known port number in the range **0 - 1023**.

Transport Layer

User Datagram Protocol (UDP) [Protokol 17]



Receiver port number: This port number is used by the receiver process that is on the receiver host.

The field has a length of *16 bit* (i.e., 65,536 different ports).

Transport Layer

User Datagram Protocol (UDP) [Protokol 17]

0	16 31
Source port number	Destination port number
Total length	Checksum

Length: This field is 16 bits and indicates the total length of the datagram (header + data).

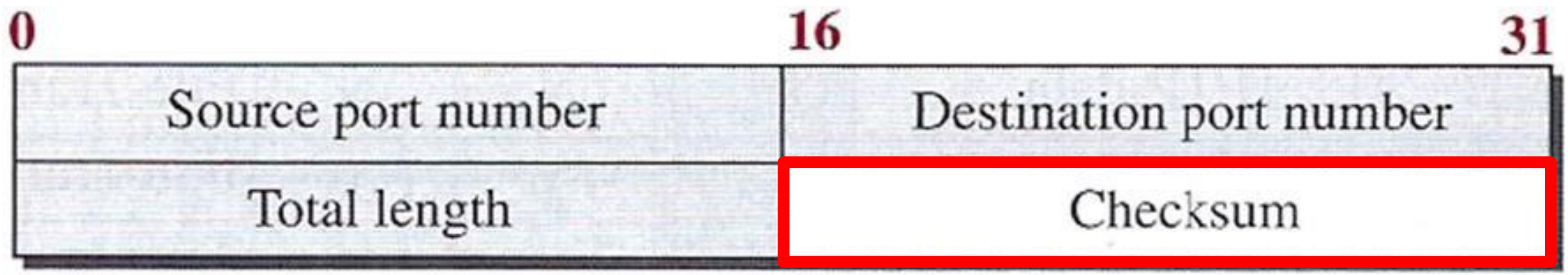
The total length can be 65,535 byte.

But it will never be that long, as the UDP datagram must be inserted into an IP datagram

with a maximum total length of 65,535 bytes.

Transport Layer

User Datagram Protocol (UDP) [Protokol 17]



Checksum: This field is used to detect errors in the entire datagram (both header and data), we will take a closer look at it that later

Transport Laget

User Datagram Protocol (UDP): Service

- **Process-to-process communication:** UDP adds process-to-process communication. Here, the so-called ***socket address*** is used, which is a combination of IP address and port address.

- **Connectionless Service**: UDP offers a connectionless service.
  This means that each user datagram sent via UDP is independent.

  There is no connection, even if they come from the same source process.

  The datagrams are not numbered, and no connection is established between sender and receiver before datagrams are sent.

  Only processes that send short messages which are small enough to be in one user datagram can use the UDP protocol.

  Maximum space in datagram is:
  65,507 bytes (65,535 bytes – 8 bytes UDP header – 20 bytes IP header)

Transport Layer

User Datagram Protocol (UDP): Service

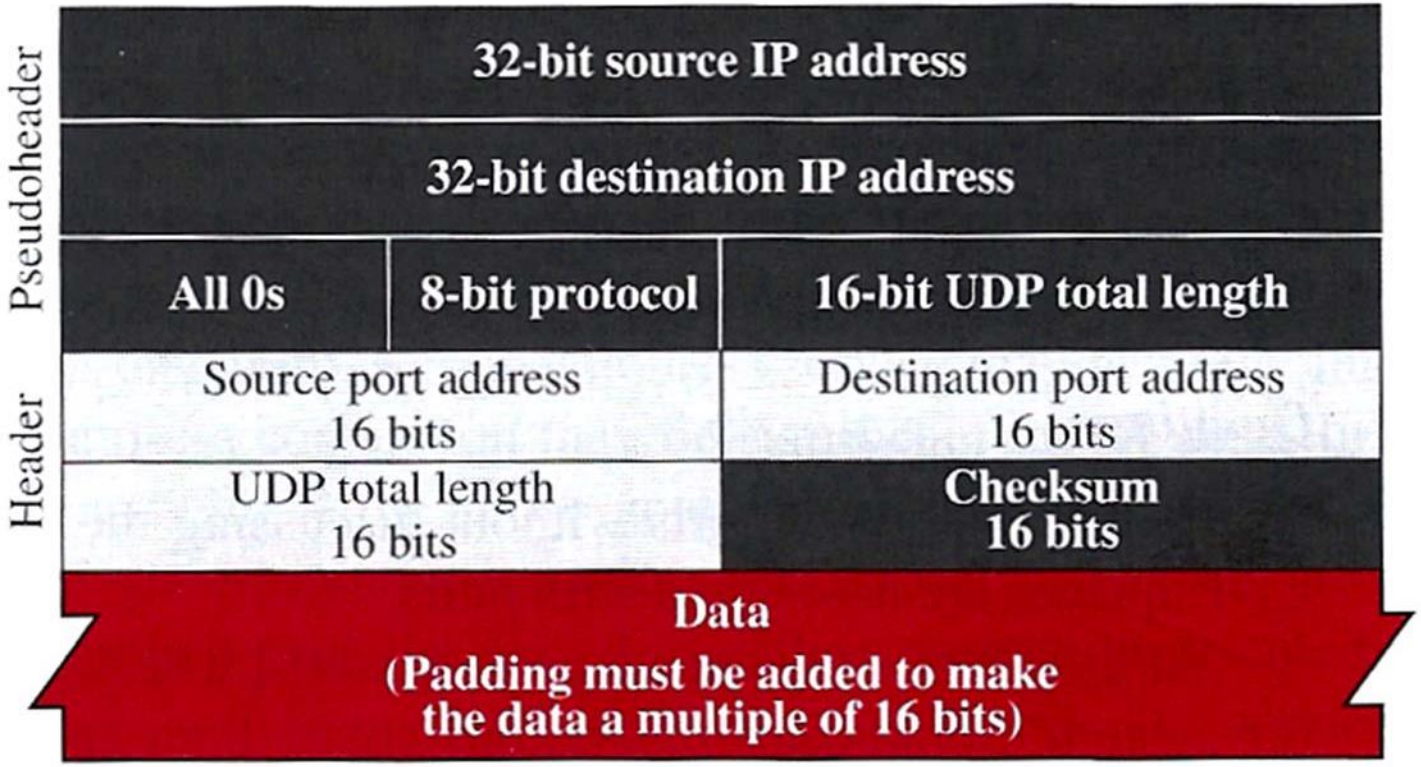
- **Flow control:** UDP is a very simple protocol that **does not offer flow control.** If the process using UDP needs flow control, *then it must take care of it itself.*

- **Error Control:** UDP offers **no error control** *other than checksum*. If an error is found via the checksum, the datagram is discarded. The sender will not be notified. If the process using UDP needs error control, *then it must take care of it itself.*

Transport Layer

User Datagram Protocol (UDP): Service

- **Checksum:** The UDP checksum is different from the one we used in IP and ICMP. Here we have three sections for the checksum calculation:
  - Pseudoheader.
  - UDP header.
  - Data from the application layer.



Transport Layer

User Datagram Protocol (UDP): Pseudoheaders

If the Pseudoheader was not included here, a UDP datagram could arrive without error.

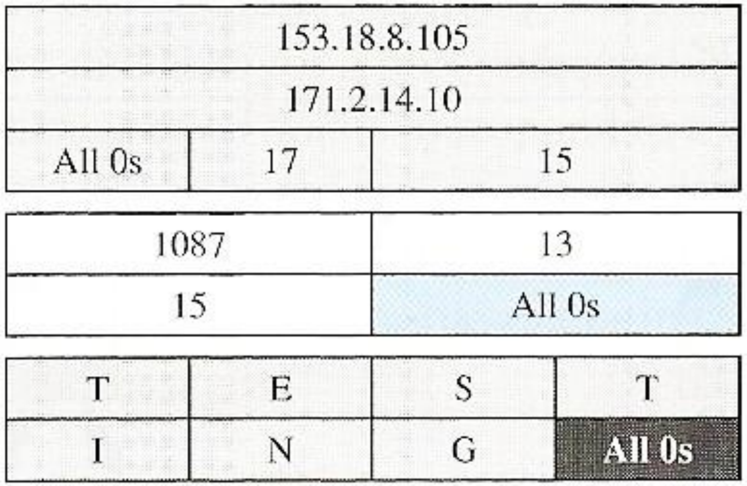
- But if the IP header was corrupted, then the UDP datagram could have ended up with a wrong host!

The 8-bit protocol field in the IP header is 17, which corresponds to UDP. This is important as both TCP and UDP can use some of the same port numbers. Therefore, a Pseudoheader is included, and a packet can also be discarded if this field is changed along the way.

Transport Layer

User Datagram Protocol (UDP): Pseudoheader Example

We can see that the protocol field in the IP header is 17, which corresponds to UDP.



10011001 00010010	→	153.18
00001000 01101001	→	8.105
10101011 00000010	→	171.2
00001110 00001010	→	14.10
00000000 00010001	→	0 and 17
00000000 00001111	→	15
00000100 00111111	→	1087
00000000 00001101	→	13
00000000 00001111	→	15
00000000 00000000	→	0 (checksum)
01010100 01000101	→	T and E
01010011 01010100	→	S and T
01001001 01001110	→	I and N
01000111 00000000	→	G and 0 (padding)
10010110 11101011	→	Sum
01101001 00010100	→	Checksum

The sender can choose not to calculate a checksum!* *It is therefore optional

If the sender chooses not to calculate checksum, ***then the checksum field is filled with 0s.***

To avoid misunderstandings in the case where the sender calculates a checksum and gets the value 0, then the checksum is complemented so that the result is **11111111 11111111.**

Transport Layer

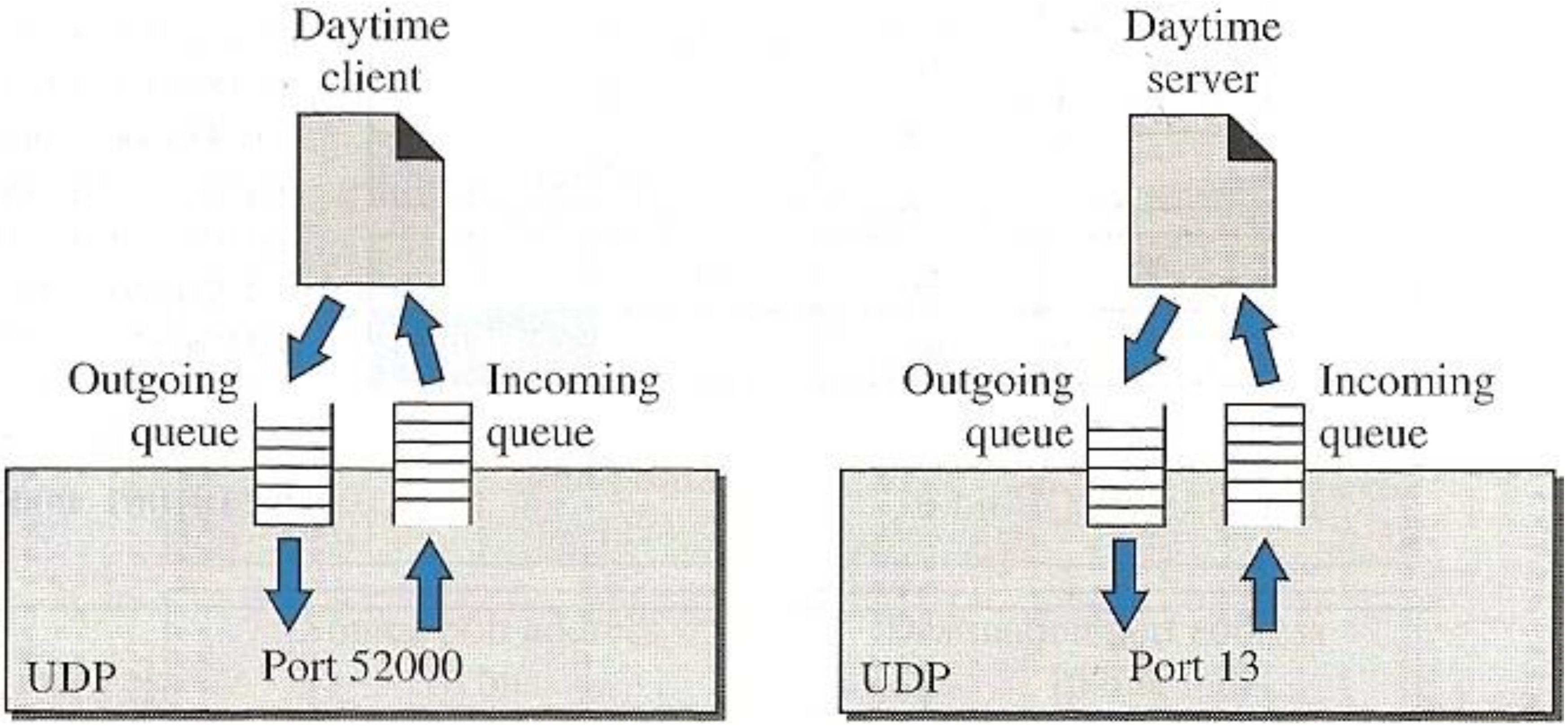
User Datagram Protocol (UDP): Encapsulation and decapsulation

When UDP communication takes place, the UDP protocol encapsulates and decapsulates the messages to and from IP datagrams.

Transport Layer

User Datagram Protocol (UDP): Queuing

- Queues are linked in UDP to each port.
- Most often, both an incoming and an outgoing queue are created, but in some cases only an incoming queue is created



Note that the user datagrams in the queues are not related to each other!

Transport Layer

User Datagram Protocol (UDP): UDP Applications

Although UDP does not meet very many criteria for reliable transport, UDP protocol is still the preferred protocol in certain applications since reliability has its costs:

- Flow- and Error-control = slower and more complex service (reliable).
- No Flow and Error-control = faster and simpler service (unreliable).

Applications that only send short queries and receive short answers can advantageously use UDP.
(with short queries and answers it is meant that they can be in one datagram)
e.g., DNS service uses UDP as transport protocol.

Lack of error control can sometimes be an advantage.

Error checking can in fact give rise to an unstable data flow, which can be inappropriate.

Sometimes it's more important to have a steady stream of data than to have complete data.
e.g. **Skype** or other real-time applications.

Transport Layer
User Datagram Protocol (UDP): UDP Applications

Typical applications of UDP

- UDP is well suited for processes that only need a <u>simple request/response communication</u>.
  UDP **is not often used for FTP service which usually deal with bulk data.**

- UDP is well suited to processes, with its own flow and error control mechanisms. For example, the <u>T</u>rivial <u>F</u>ile <u>T</u>ransfer <u>P</u>rotokollen (**TFTP**) contains procedures for flow- and error-control, it will be easy to use UDP.

- UDP is well suited as a transport protocol for multicasting.
  Multicasting capability is embedded into the UDP software, but not into the TCP software.

- UDP is used for management processes.
  e.g., ***SNMP*** (<u>S</u>imple <u>N</u>etwork <u>M</u>anagement <u>P</u>rotocol)

Transport Layer

User Datagram Protocol (UDP): UDP Applications

Typical applications of UDP

- UDP is used for some route updating protocols.
  e.g., ***RIP*** (**R**outing **I**nformation **P**rotocol)

- UDP is typically used for interactive real-time applications that cannot tolerate uneven data flow but are tolerant of losing a little data occasionally. **(e.g., real-time robot control)**