

TCP/IP-(18MCA45E)

UNIT-III

‘Group Management – IGMP Message ’

FACULTY:

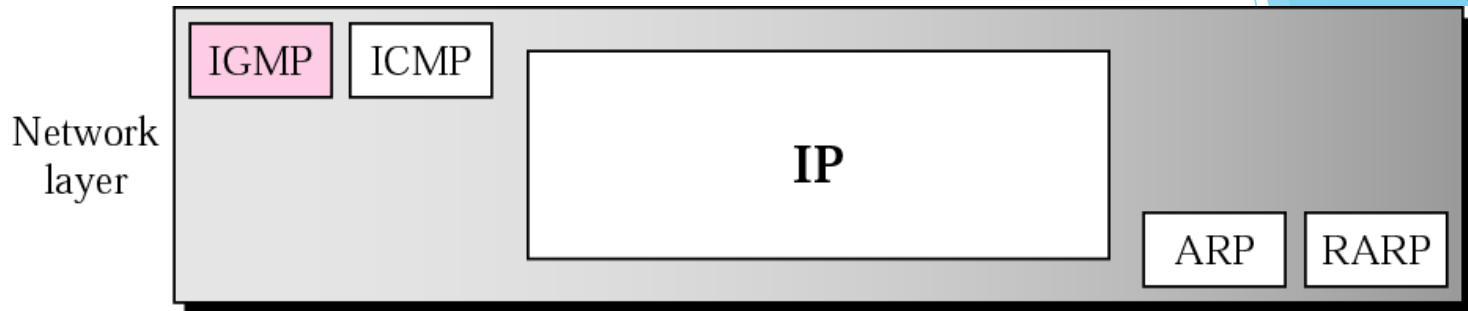
Dr. R. A. Roseline, M.Sc., M.Phil., Ph.D.,

Associate Professor and Head,

Post Graduate and Research Department of Computer
Applications,

Government Arts College (Autonomous), Coimbatore – 641
018.

Position of IGMP in the network layer



Unicast – one-to-one relationship

Multicast – one-to-many relationship – IGMP helps facilitate that one-to-many relationship

Like ICMP, IGMP is a companion to IP

IGMP is NOT a multicast routing protocol – but rather a protocol that manages the **group membership**

IGMP gives the multicast routers info about the membership status of hosts (routers) connected to the network. .

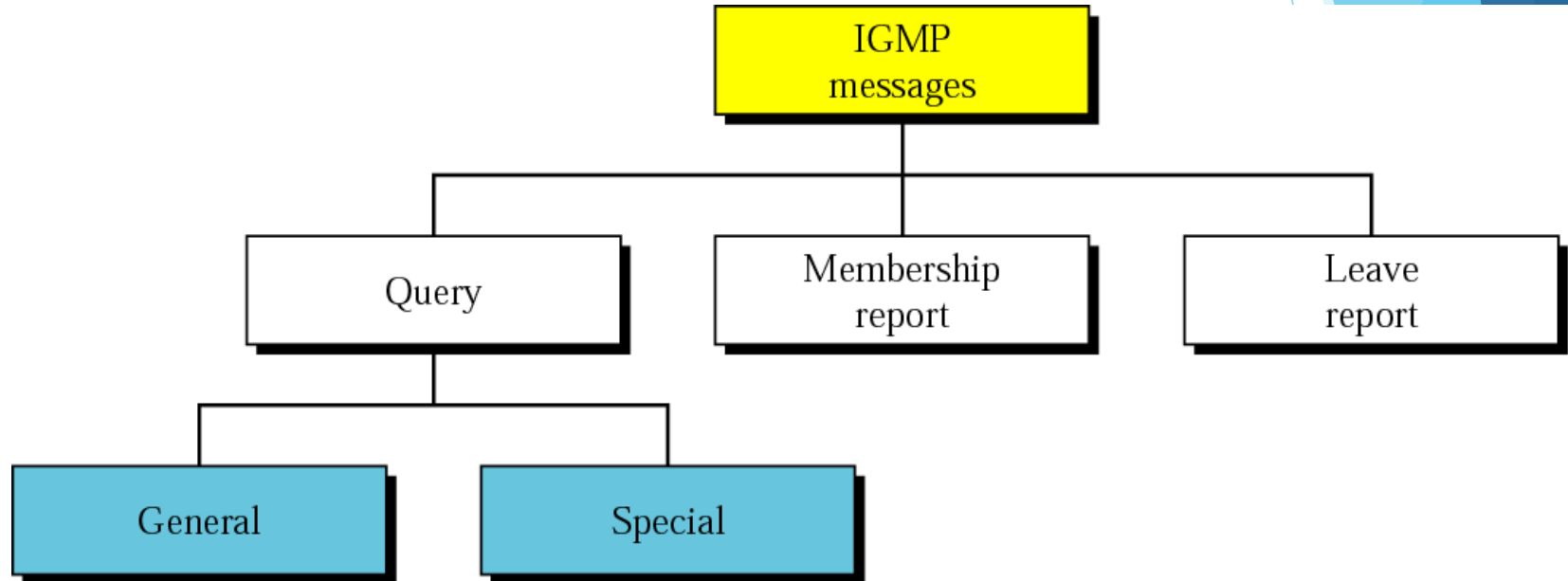


IGMP is a group management protocol. It helps a multicast router create and update a list of loyal members related to each router interface.

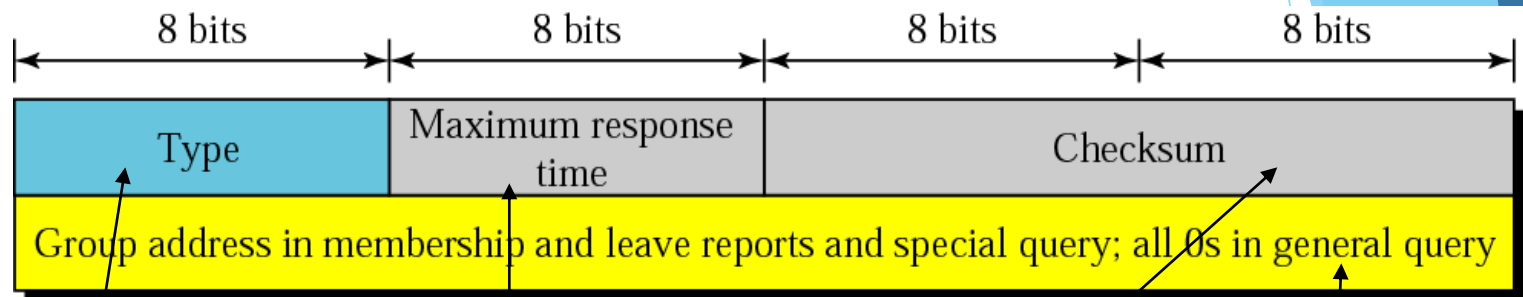
(Visualize a set of “multicast” routers amongst a set of “unicast” routers – and IGMP’s job is to facilitate this communication and info amongst the “multicast” routers”)

IGMP MESSAGES

IGMP has three types of messages: the query, the membership report, and the leave report. There are two types of query messages, general and special.



IGMP message format



Shows the type of message

Amount of time a query must be answered in – 10ths of a second units

Checksum over the entire 8-byte message

0 for general query: contains group id for special query, membership report and leave report messages

Type	Value
General or Special Query	0x11 or 00010001
Membership Report	0x16 or 00010110
Leave Report	0x17 or 00010111

IGMP OPERATION

A multicast router connected to a network has a list of multicast addresses of the groups with at least one loyal member in that network. For each group, there is one router that has the duty of distributing the multicast packets destined for that group.

The topics discussed in this section include:

Joining a Group

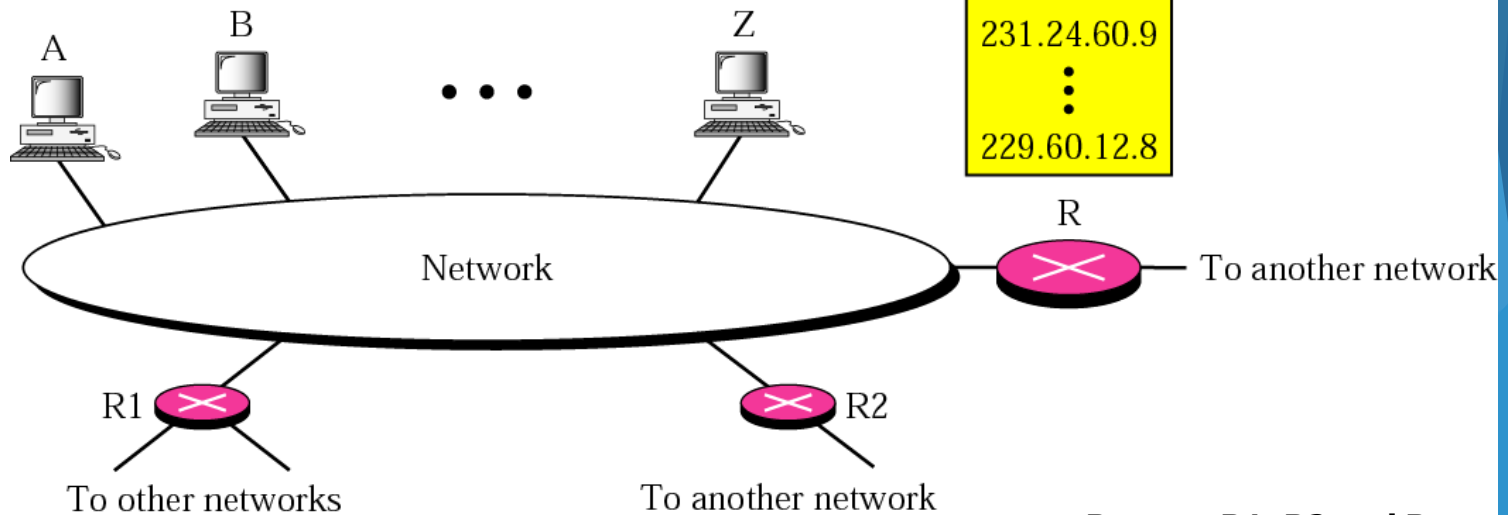
Leaving a Group

Monitoring Membership

IGMP operation

A multicast router connected to a network has a list of multicast addresses of the groups with at least one loyal member in that network. For each group, there is one router that has the duty of distributing the multicast packets destined for that group.

A host can have a membership in a group – this means one of that host's processes receives a multicast packet



A multicast router can have a membership in a group – this means one of that router's interfaces receives a multicast packet

Routers R1, R2 and R list of groupids are mutually exclusive

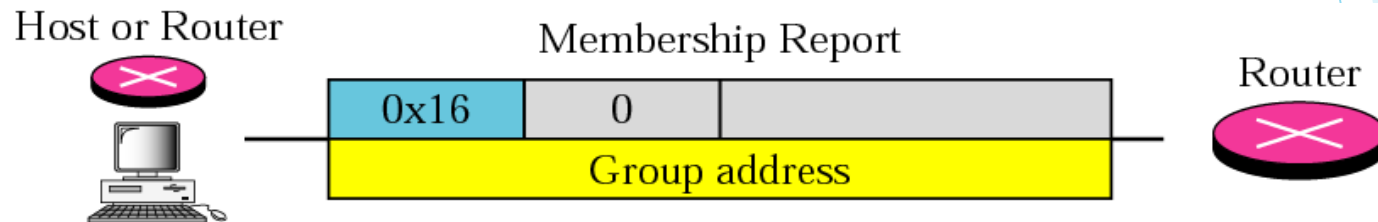
Membership report – Joining A Group

A host or router can join a group

A host maintains a list of processes that have group membership

If a process wants to join a group, the host adds process and the desired group to its list

If it is the first time entry, the host sends a “membership report” message to the distributing router (in order to receive multicast packets fro that desired group)



A router can join a group

A router maintains a list of interfaces that have group membership

If an interface wants to join a group, the router adds the interface and the desired group to its list

If it is the first time entry, the router sends a “membership report” message. The message is sent out of all interfaces other than one from which the new interest comes



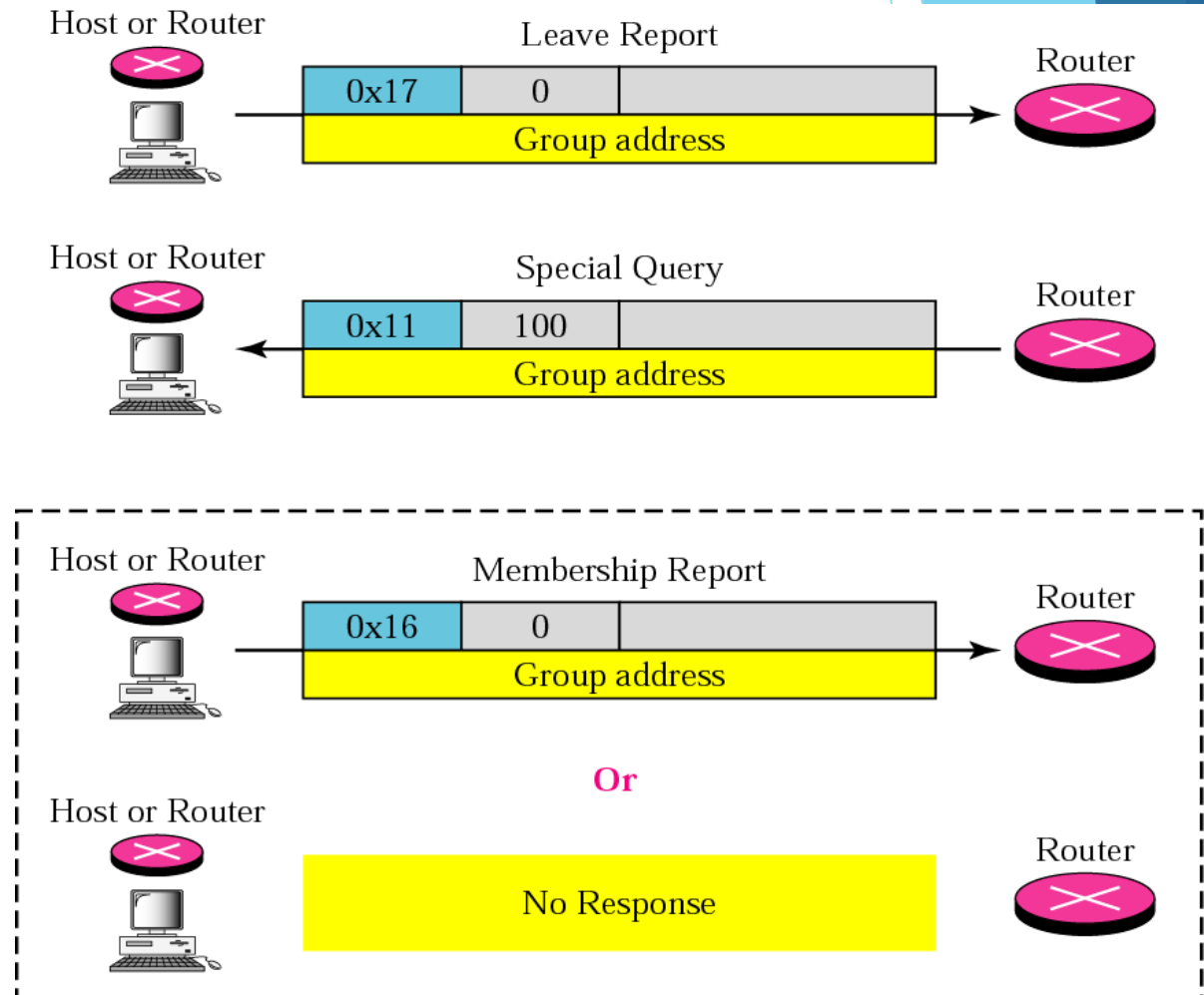
In IGMP, a membership report is sent twice, one after the other.

(if the first is lost or damaged, the second one should make it.)

Leave report

When a host (or router) sees that no process is interested in a specific group, it sends a leave report

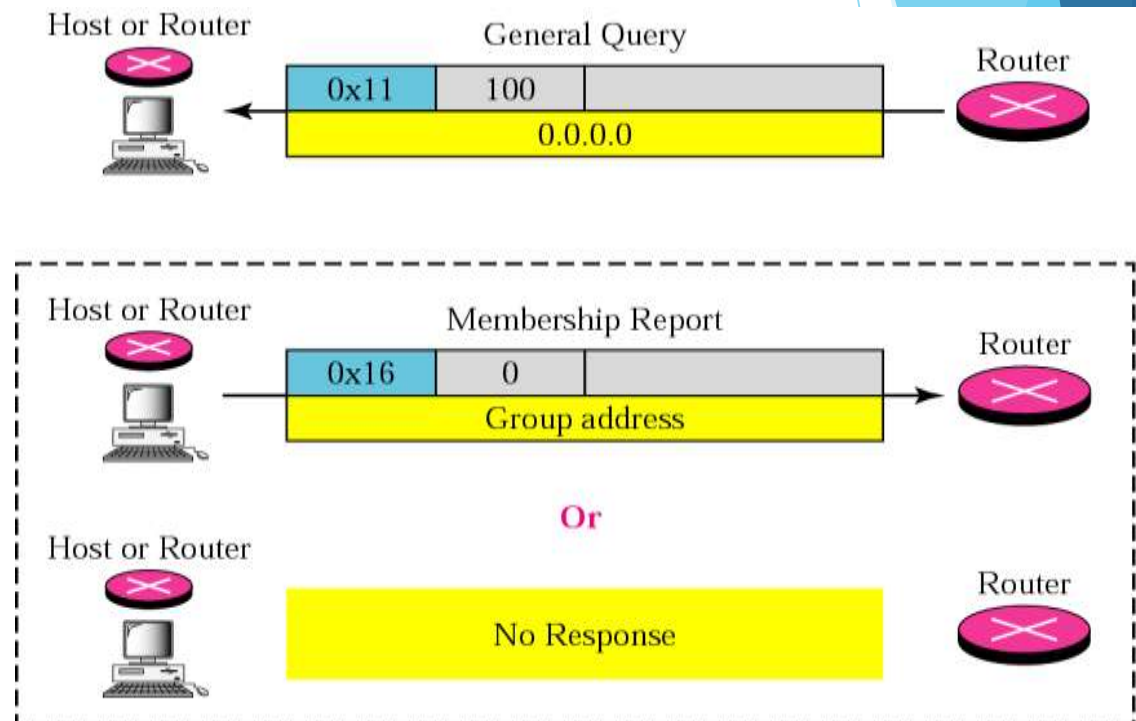
After receiving a leave report, the router doesn't automatic remove the groupid – there could be other interested hosts or interfaces – therefore the router sends a special query message – if no feedback is received in a specified amount of time – it then purges the groupid from the list



General query message

What about the case when there is only 1 host interested in a particular groupid and that host goes down ? Does the router maintain that groupid or what ?

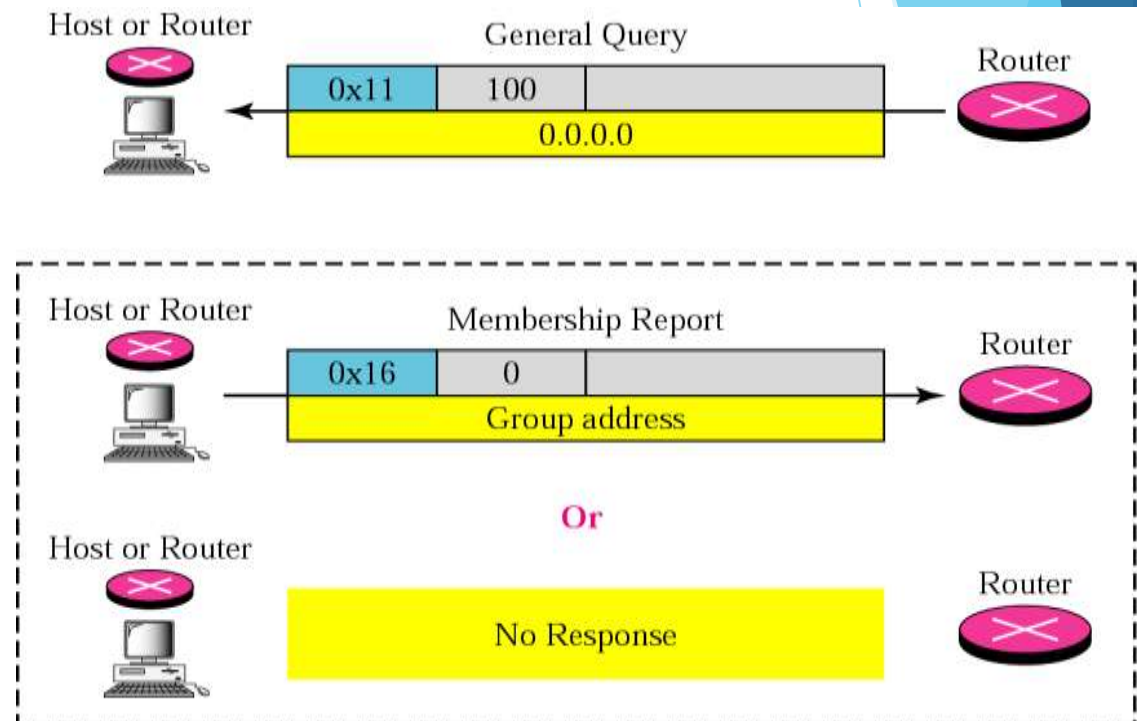
The router periodically sends "general query" messages – the general query message queries for membership continuation for all groups (not just one) – if no response is received for a particular groupid (it is removed) – if more than one host/router are interested in the same group – only one host/router responds – cuts down on traffic



Delayed Response

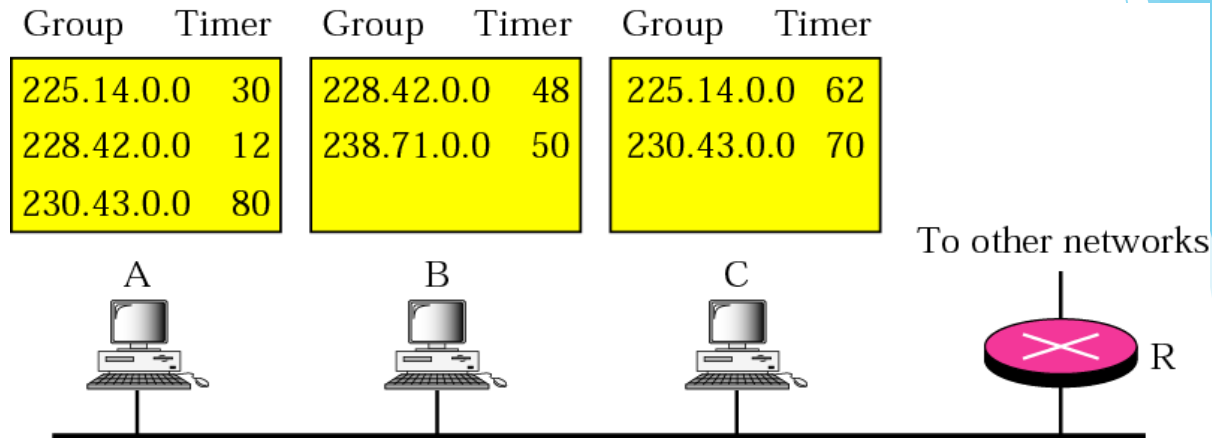
If more than one host/router are interested in the same group – only one host/router responds – cuts down on traffic – how is this implemented ? **Delayed Response**

Each router needing to send a response has randomly generated wait times before sending a report FOR EACH group – because the reports are broadcasted – the router will know if some other router has already sent a report regarding the groupid (therefore relinquishing it from having to send a report)



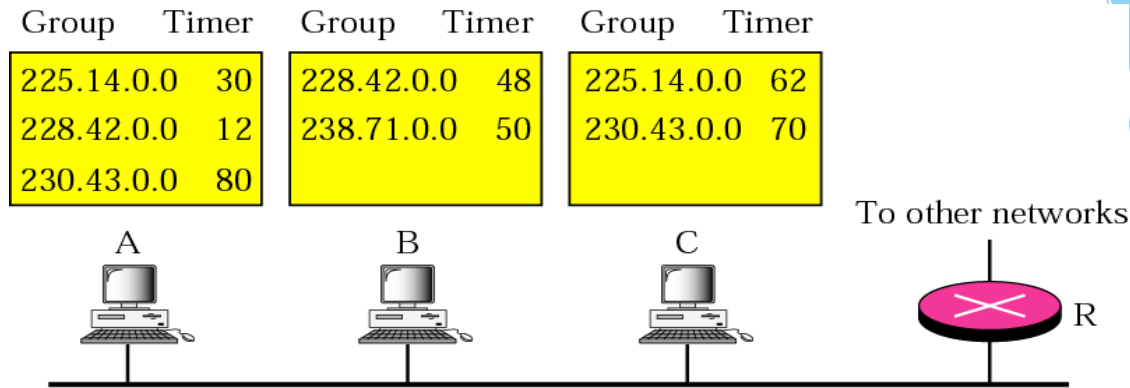
EXAMPLE 1

Imagine there are three hosts in a network as shown below.



A query message was received at time 0; the random delay time (in tenths of seconds) for each group is shown next to the group address. Show the sequence of report messages.

EXAMPLE 1 (CONTINUED)



Solution

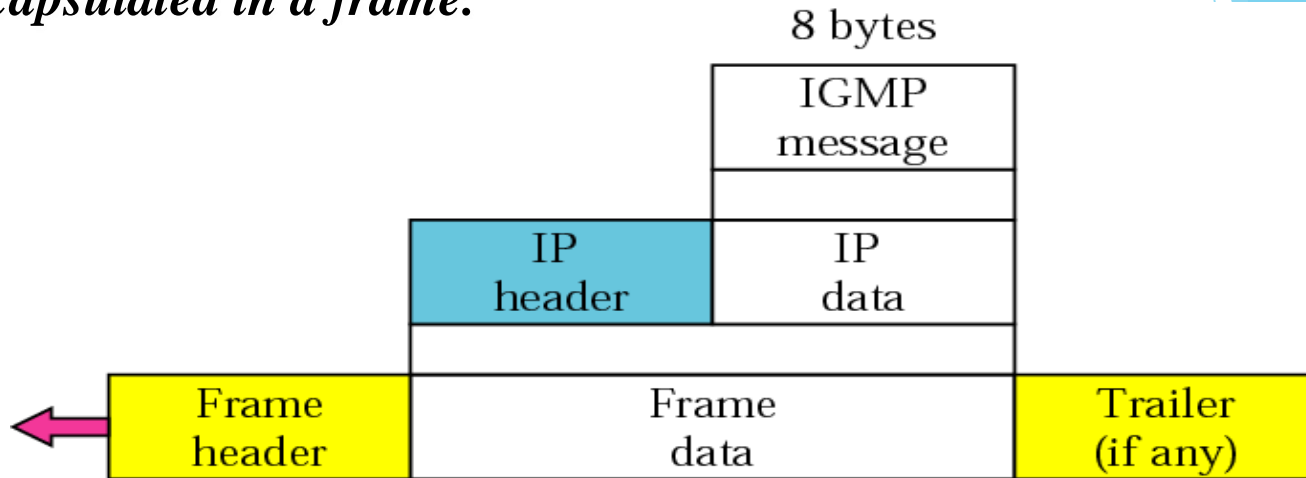
The events occur in this sequence:

- Time 12:** The timer for 228.42.0.0 in host A expires and a membership report is sent, which is received by the router and every host including host B which cancels its timer for 228.42.0.0.
- Time 30:** The timer for 225.14.0.0 in host A expires and a membership report is sent, which is received by the router and every host including host C which cancels its timer for 225.14.0.0.
- Time 50:** The timer for 238.71.0.0 in host B expires and a membership report is sent, which is received by the router and every host.
- Time 70:** The timer for 230.43.0.0 in host C expires and a membership report is sent, which is received by the router and every host including host A which cancels its timer for 230.43.0.0.

Note that if each host had sent a report for every group in its list, there would have been seven reports; with this strategy only four reports are sent.

Encapsulation of IGMP packet

The IGMP message is encapsulated in an IP datagram, which is itself encapsulated in a frame.



Because the IGMP occurs within the physical LAN, the TTL of the IP is set to 1 – guarantees the message doesn't leave the LAN

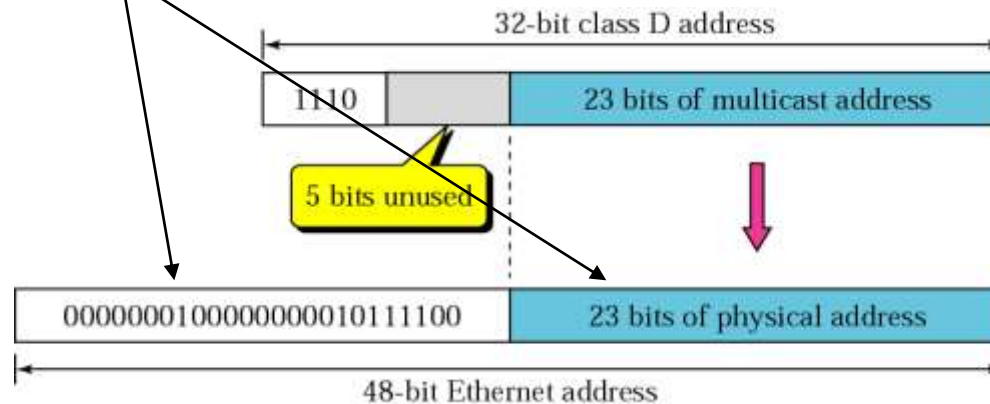
Regarding the data link layer:

Because the IP packet has a MULTICAST address, ARP can't be used in finding the physical address and forwarding – therefore, the data link layer (or underlying technology) must support multicast addressing

Mapping class D to Ethernet physical address

Ethernet supports physical multicast addressing

If the first 25 bits indicate this pattern, then the remaining 23 bits can take on a group



The router extracts the least significant 23 bits of the class D – however, the class D is 28 bits – therefore, 2^5 (32) multicast addresses are mapped to a single multicast address at the IP level

Therefore, the host must check the IP and discard any packets that do not belong to it.



Note:

An Ethernet multicast physical address is in the range

01:00:5E:00:00:00

to

01:00:5E:7F:FF:FF.

EXAMPLE 2

Change the multicast IP address 230.43.14.7 to an Ethernet multicast physical

Solution

We can do this in two steps:

a. We write the rightmost 23 bits of the IP address in hexadecimal. This can be done by changing the rightmost 3 bytes to hexadecimal and then subtracting 8 from the leftmost digit if it is greater than or equal to 8. In our example, the result is 2B:0E:07.

b. We add the result of part a to the starting Ethernet multicast address, which is (01:00:5E:00:00:00). The result is

01:00:5E:2B:0E:07

EXAMPLE 3

Change the multicast IP address 238.212.24.9 to an Ethernet multicast address.

Solution

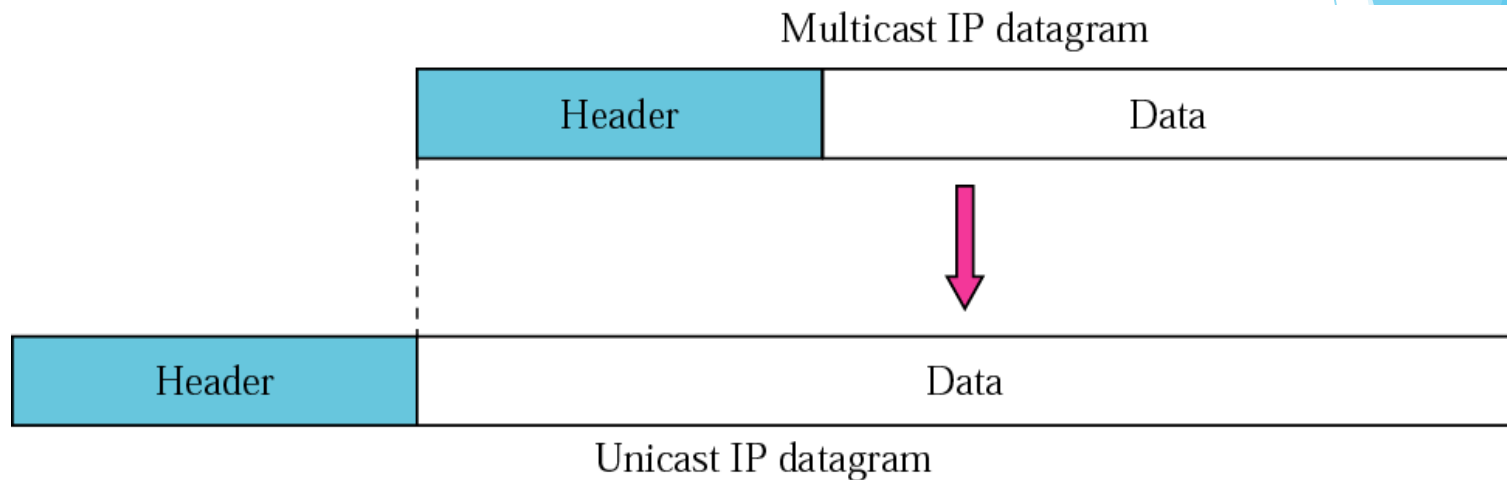
a. The right-most three bytes in hexadecimal are D4:18:09. We need to subtract 8 from the leftmost digit, resulting in 54:18:09..

b. We add the result of part a to the Ethernet multicast starting address. The result is

01:00:5E:54:18:09

Tunneling

Most WANs do not support physical multicast addressing – therefore tunneling is used – the multicast packet is encapsulated in the unicast packet and sent through the network



EXAMPLE 4

We use netstat with three options, -n, -r, and -a. The -n option gives the numeric versions of IP addresses, the -r option gives the routing table, and the -a option gives all addresses (unicast and multicast). Note that we show only the fields relative to our discussion.

```
$ netstat -nra
```

```
Kernel IP routing table
```

<i>Destination</i>	<i>Gateway</i>	<i>Mask</i>	<i>Flags</i>	<i>Iface</i>
<i>153.18.16.0</i>	<i>0.0.0.0</i>	<i>255.255.240.0</i>	<i>U</i>	<i>eth0</i>
<i>169.254.0.0</i>	<i>0.0.0.0</i>	<i>255.255.0.0</i>	<i>U</i>	<i>eth0</i>
<i>127.0.0.0</i>	<i>0.0.0.0</i>	<i>255.0.0.0</i>	<i>U</i>	<i>lo</i>
<i>224.0.0.0</i>	<i>0.0.0.0</i>	<i>224.0.0.0</i>	<i>U</i>	<i>eth0</i>
<i>0.0.0.0</i>	<i>153.18.31.254</i>	<i>0.0.0.0</i>	<i>UG</i>	<i>eth0</i>

Any packet with a multicast address from 224.0.0.0 to 239.255.255.255 is masked and delivered to the Ethernet interface.

The background features abstract, overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the right side of the frame, creating a modern, layered effect. The rest of the background is plain white.

UDP

Introduction

- Responsibilities of Transport Layer
 - to create a process-to-process communication
 - using port numbers in case of UDP
 - to provide a flow-and-error control mechanism at the transport level
 - But, no flow control mechanism and no acknowledgment for received packets in UDP
 - If UDP detects an error in the received packets, it silently drops it.
 - to provide a connection mechanism for the processes
 - sending streams of data to the transport layer by process
 - making the connection, chopping the stream into transportable units, numbering them and sending them one by one

Introduction (cont'd)

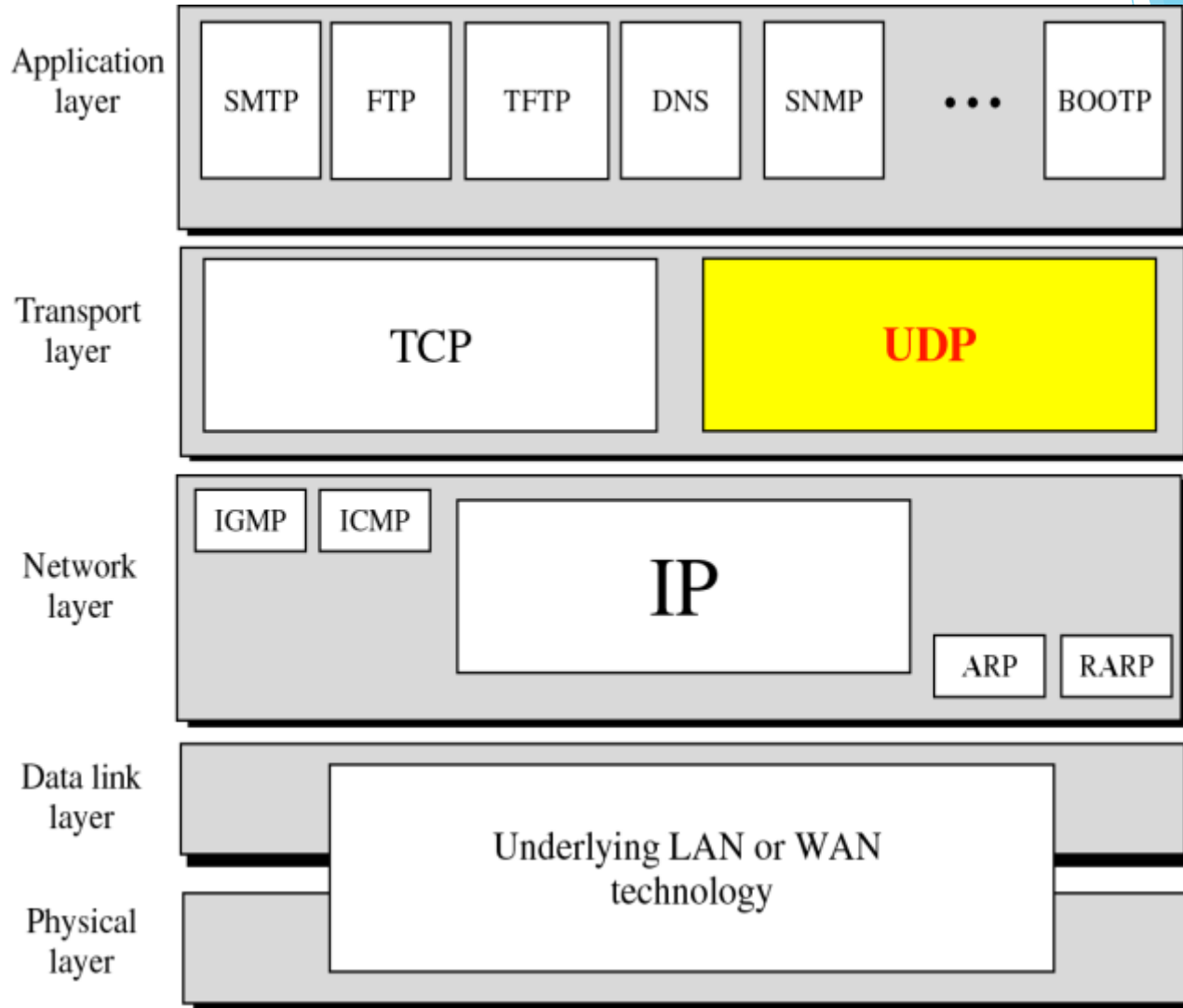
- ▶ Normally, at the receiving end, waiting until all the different units belonging to the transport layer have received, checking, passing those that are error free and delivering them to the receiving process as a stream
- ▶ But, UDP
 - ▶ does not do any of the above
 - ▶ can only receive a data unit from the process and deliver it, unreliably, to the receiver
 - ▶ data unit must be small enough to fit in a UDP packet
- ▶ UDP is called a connectionless, unreliable transport protocol
 - ▶ providing process-to-process communication instead of host-to-host communication
 - ▶ performing very limited error checking

Introduction (cont'd)

- ▶ If UDP is so powerless, why would a process want to use it?
 - ▶ very simple protocol using a minimum of overhead
 - ▶ if a process wants to send a small message and does not care much about reliability, it can use UDP
 - ▶ if it sends a small message, taking much less interaction between the sender and receiver than it does using TCP

Introduction (cont'd)

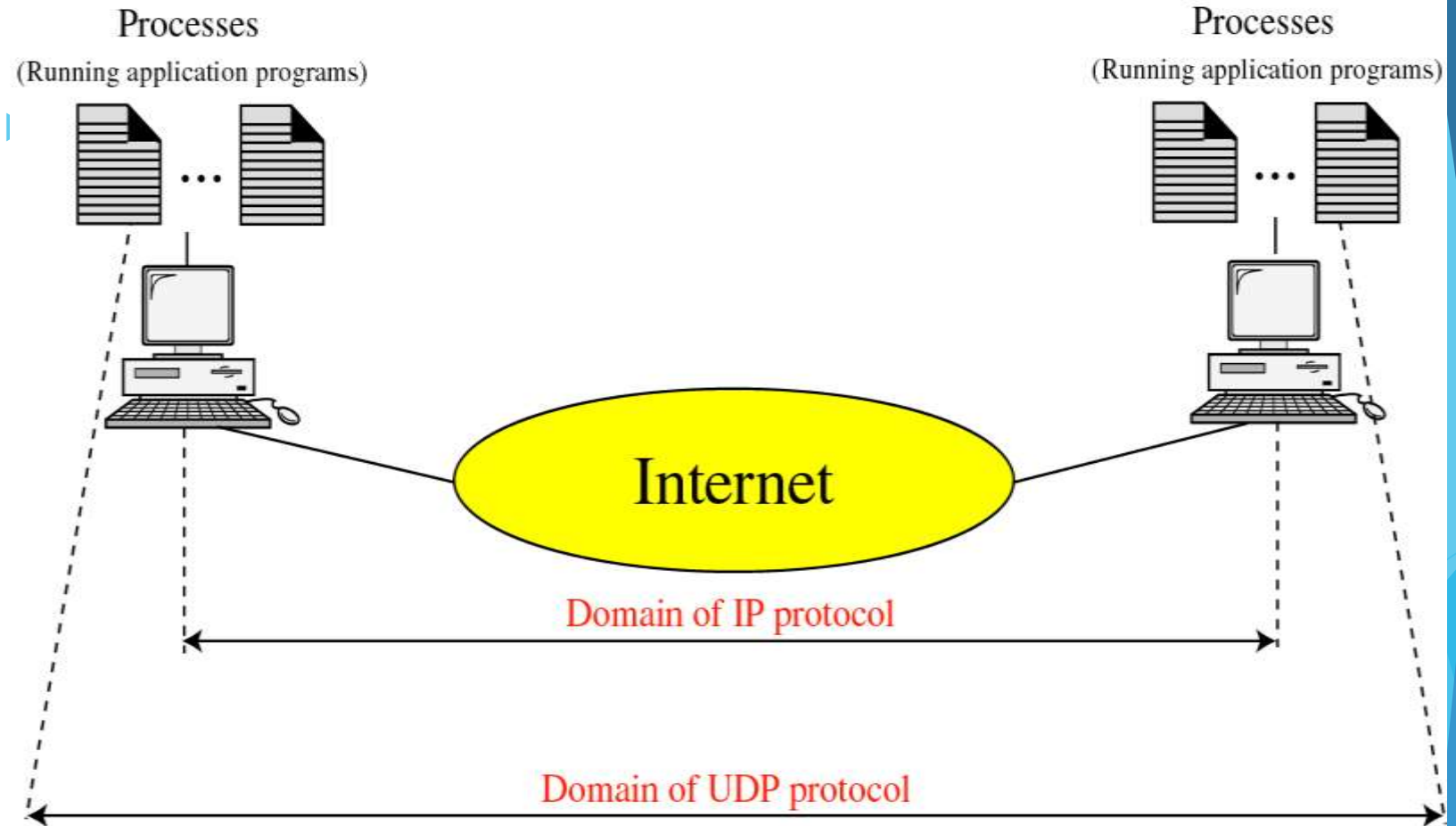
Position of UDP in the TCP/IP protocol suite



11.1 Process-to-process communication

- ▶ IP is responsible for communication at the computer level (host-to-host communication)
- ▶ UDP is responsible for delivery of the message to the appropriate process

Process-to-process communication (cont'd)

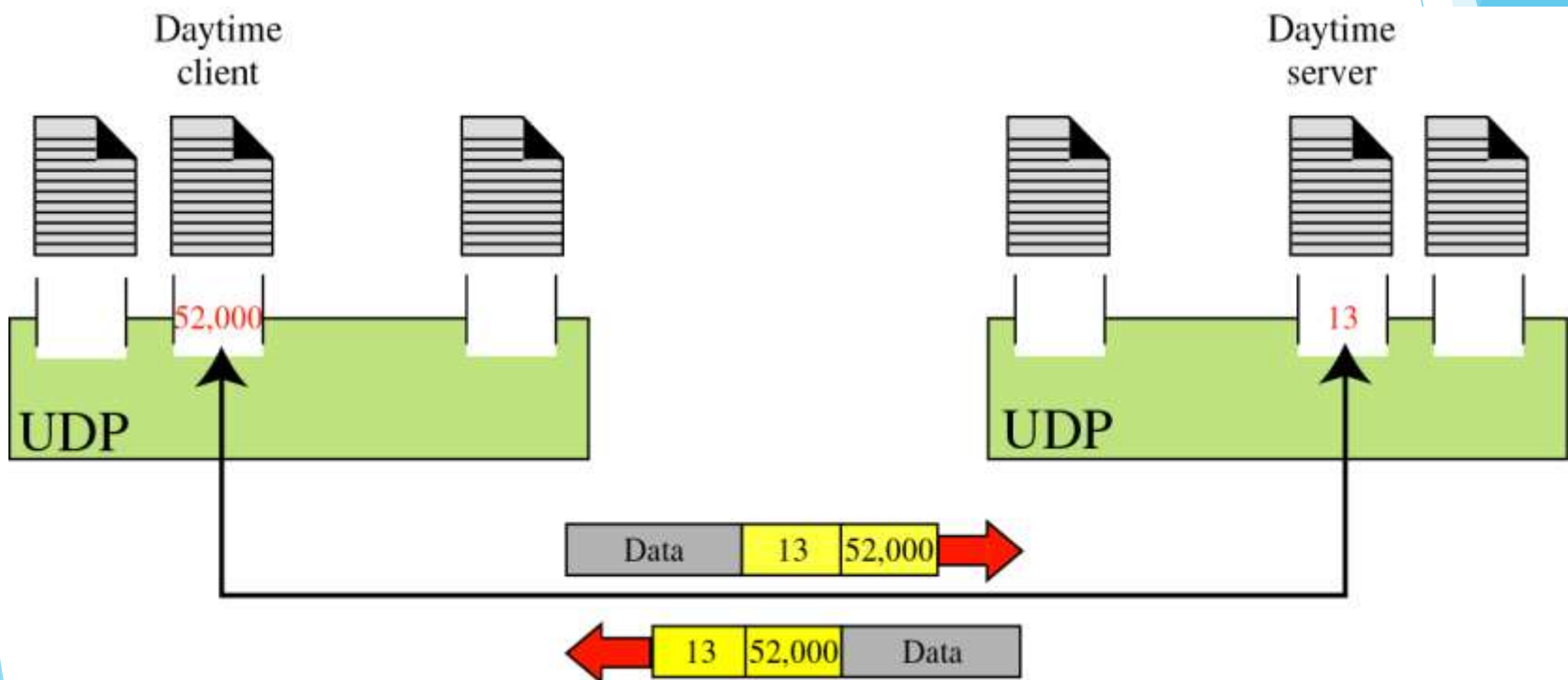


Process-to-process communication (cont'd)

- ▶ Port Numbers
 - ▶ used in client-server paradigm
 - ▶ used for defining processes
 - ▶ integers between 0 and 65,535
 - ▶ The client program defines itself with a port number, chosen randomly by the UDP software running on the client host
 - ▶ the ephemeral port number
 - ▶ But, the server process must also define itself with a port number that is not randomly chosen
 - ▶ using well-known port number

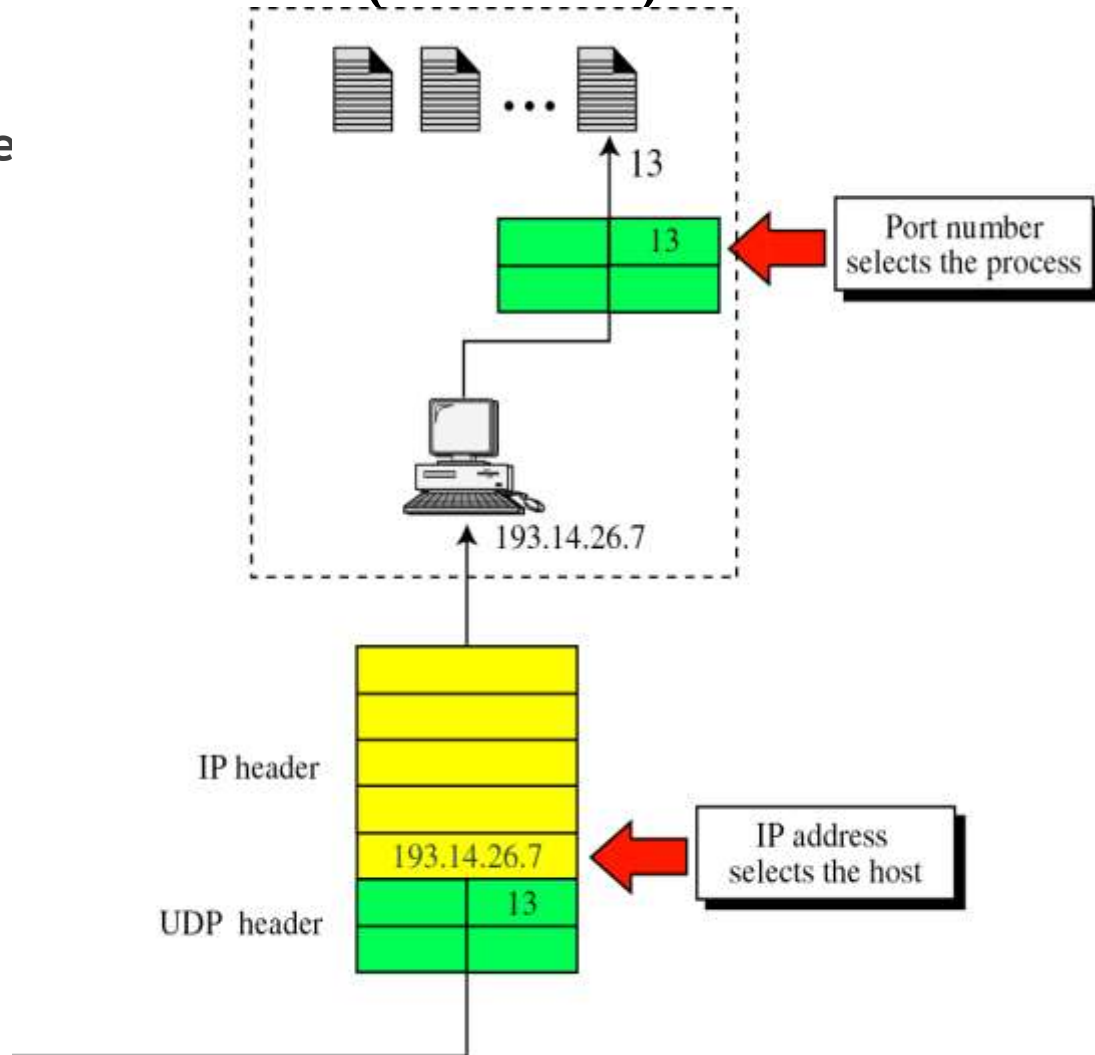
Process-to-process communication (cont'd)

- ▶ Example : Daytime process
 - ▶ for client, an ephemeral (temporary) port number 52,000 and for server, the well-known (permanent) port number 13.



Process-to-process communication (cont'd)

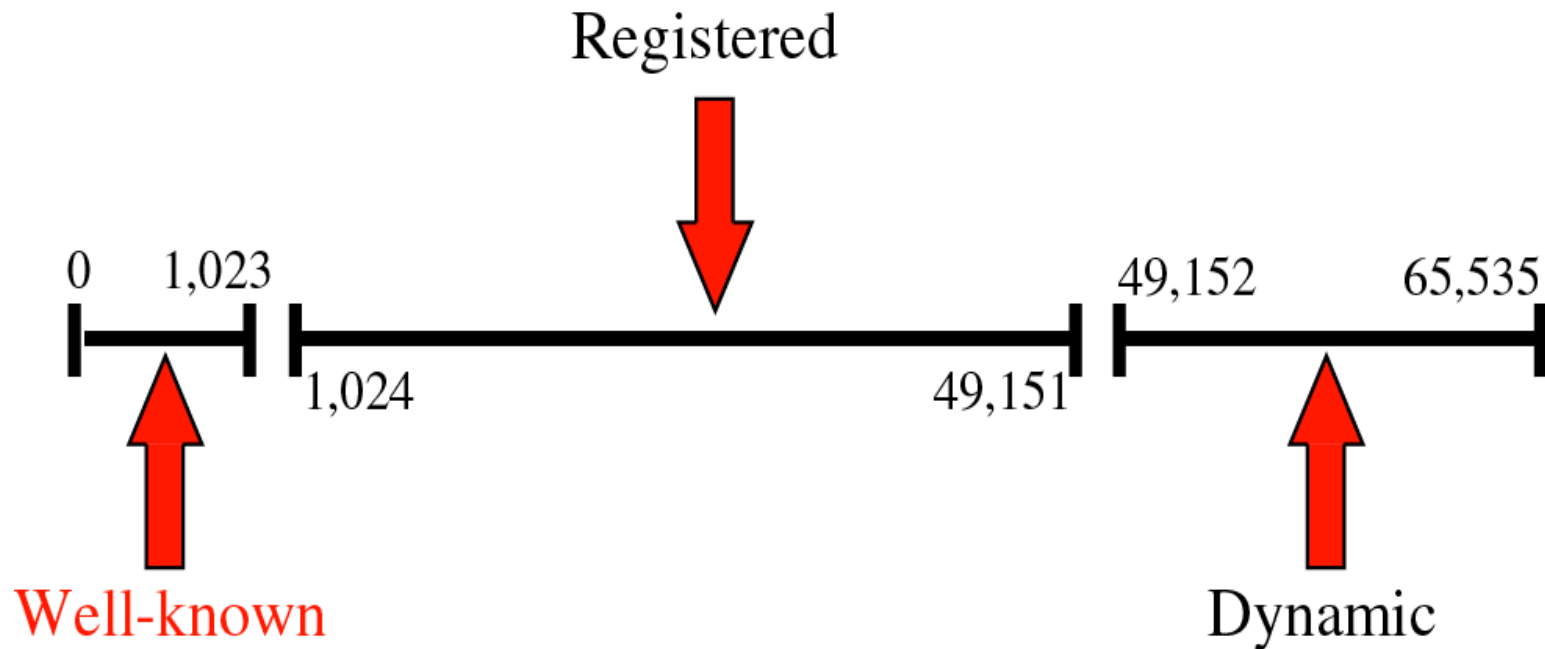
▶ IP address



Process-to-process communication (cont'd)

- ▶ IANA (Internet Assigned Numbers Authority)
 - ▶ port numbers divided into 3 ranges
 - ▶ well-known ports : ranging from 0 to 1,023
 - ▶ registered ports : ranging from 1,024 to 49,151
 - ▶ They are not controlled by IANA. But they can only be registered with IANA to prevent duplication.
 - ▶ dynamic ports : ranging from 49,152 to 65,535
 - ▶ neither controlled nor registered
 - ▶ can be used any process.
 - ▶ are ephemeral ports
- ▶ Ranges Used by Other Systems
 - ▶ Other operating system may use ranges other than IANA's for the well-known and ephemeral ports
 - ▶ BSD Unix has 3 ranges: reserved, ephemeral, and non-privileged

Process-to-process communication (cont'd)



Process-to-process communication (cont'd)

- ▶ Well-known Ports for UDP
 - ▶ Some port numbers can be used by both UDP and TCP

<i>Port</i>	<i>Protocol</i>	<i>Description</i>
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users

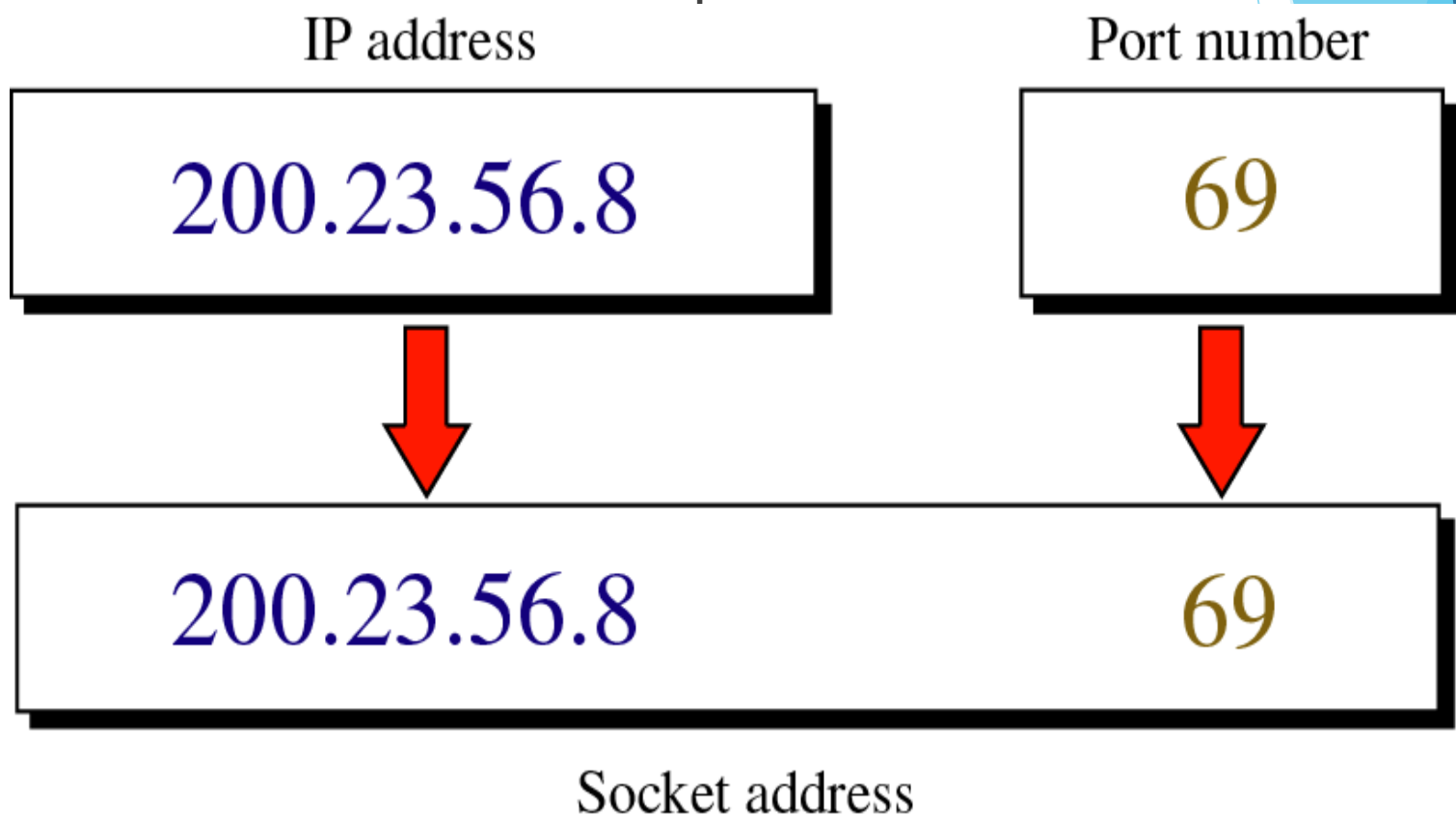
Process-to-process communication (cont'd)

<i>Port</i>	<i>Protocol</i>	<i>Description</i>
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	Bootps	Server port to download bootstrap information
68	Bootpc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

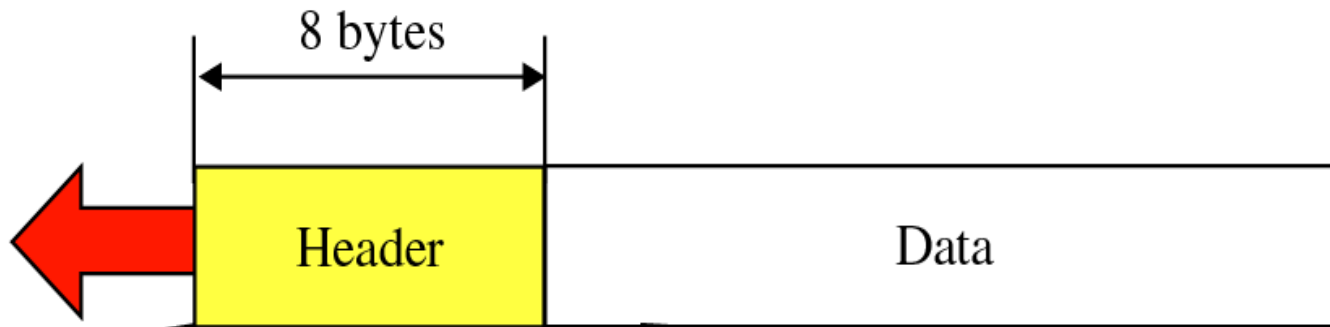
Process-to-process communication (cont'd)

▶ Socket Address

- ▶ the combination of an IP address and a port number



User Datagram



Source port number 16 bits	Destination port number 16 bits
Total length 16 bits	Checksum 16 bits

User Datagram

▶ Source port number

- ▶ In case of the client (a client sending a request), having ephemeral port number requested by the process
- ▶ In case of the server (a sever sending response), having a well-known port number

▶ Destination port number

- ▶ Used by the process running on the destination host

▶ Length

- ▶ Defining the total length of the user diagram, header + data
- ▶ Minimum 8 bytes. (header + no data)
- ▶ the length of data : 0 to 65,507 (65,535 - 20 - 8) bytes
- ▶ $\text{UDP length} = \text{IP Length} - \text{IP header's length}$

▶ Checksum

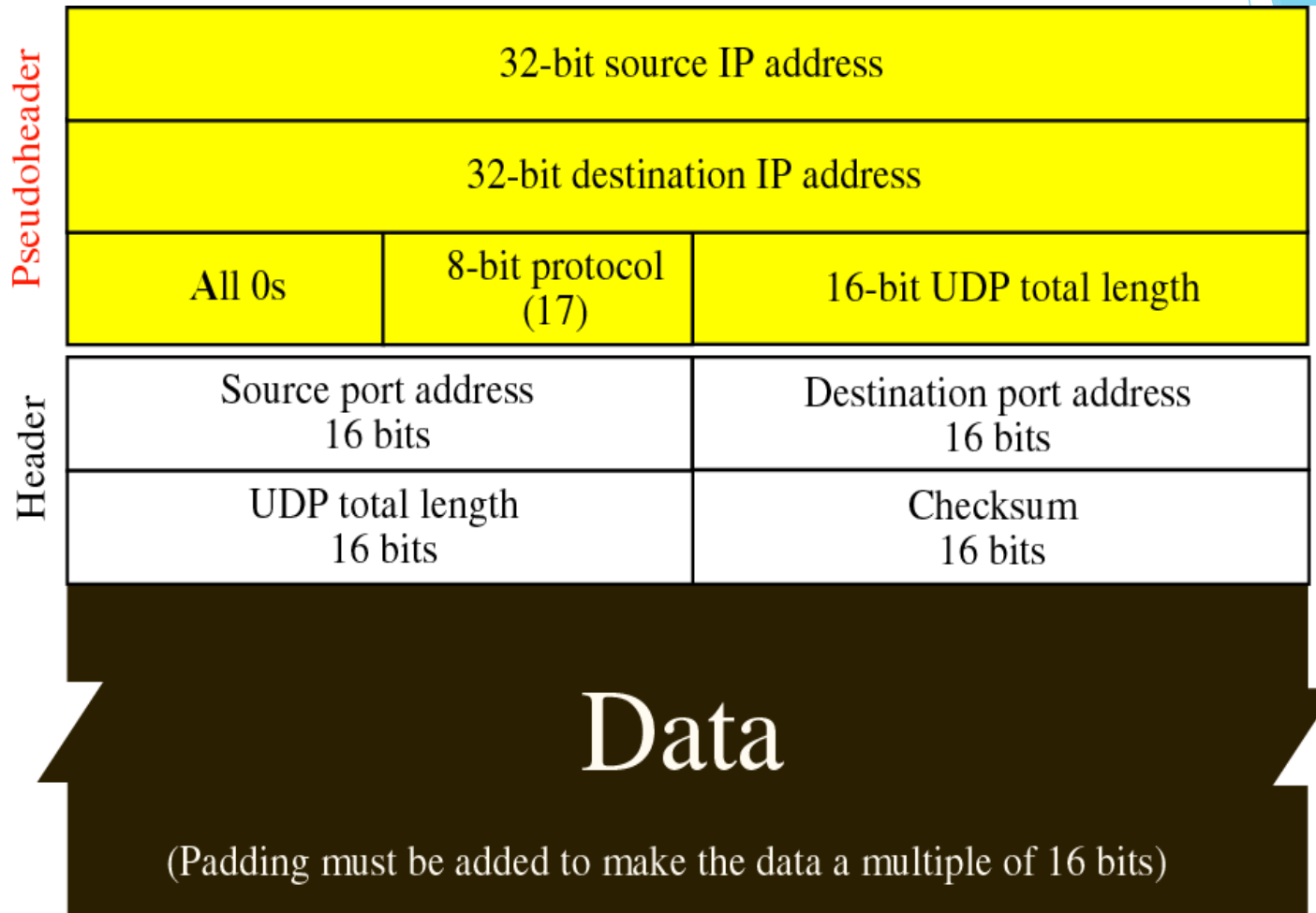
- ▶ used to detect errors over the entire user datagram

Checksum

- ▶ UDP checksum calculation is different from the checksum for IP and ICMP
- ▶ It includes as follows.
 - ▶ pseudoheader : part of the header of the IP packet
 - ▶ UDP header
 - ▶ data from the application layer

Checksum (cont'd)

- ▶ Pseudoheader added to the UDP datagram



Checksum (cont'd)

- ▶ Checksum Calculation at Sender
 1. Add the pseudoheader to the UDP datagram
 2. Fill the checksum field with zeros
 3. Divide the total number of bytes into 16-bit words
 4. If the total number of bytes is not even, add one byte of padding (all 0s)
 5. Add all 16-bit sections using one's complement arithmetic.
 6. Complement the result, and insert it in the checksum field
 7. Drop the pseudoheader and added padding
 8. Deliver the UDP user datagram to the IP software for encapsulation

Checksum (cont'd)

- ▶ Checksum Calculation at Receiver
 1. Add the pseudoheader to the UDP user datagram
 2. Add padding if needed
 3. Divide the total bits into 16-bit sections
 4. Add all 16-bit sections using one's complement arithmetic
 5. Complement the result
 6. If the result is all 0s, drop the pseudoheader and any added padding and accept the user datagram.

Checksum (cont'd)

153.18.8.105			
171.2.14.10			
All 0s	17	15	
1087		13	
15		All 0s	
T	E	S	T
I	N	G	All 0s

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)
<hr/>			
10010110	11101011	→	Sum
01101001	00010100	→	Checksum

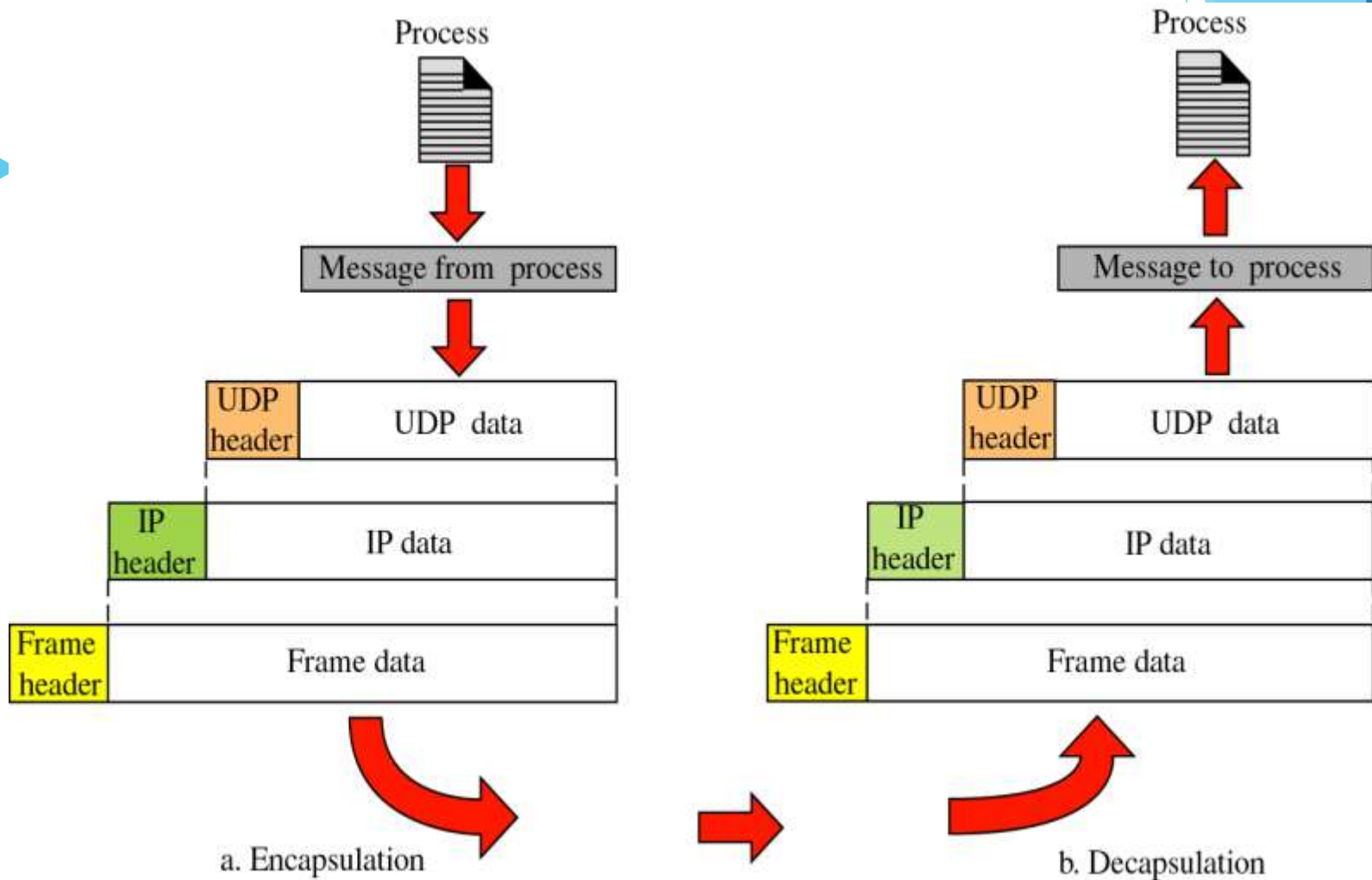
UDP Operation

- ▶ Connectionless Services
 - ▶ each user datagram sent by UDP is an independent datagram
 - ▶ each user datagram can travel a different path

UDP Operation (cont'd)

- ▶ Flow and error control
 - ▶ no flow control, hence no windowing mechanism
 - ▶ The receiver may overflow with incoming messages
 - ▶ no error control mechanism except for the checksum
 - ▶ the sender does not know if a message has been lost or duplicated
 - ▶ So, the process using UDP provides these mechanism

UDP Operation (cont'd)

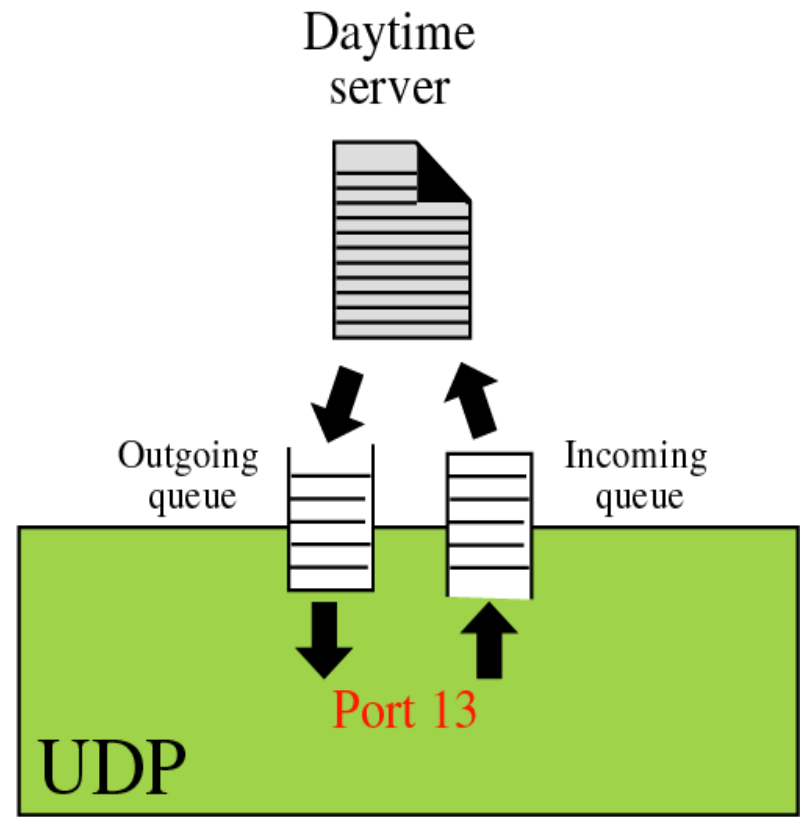
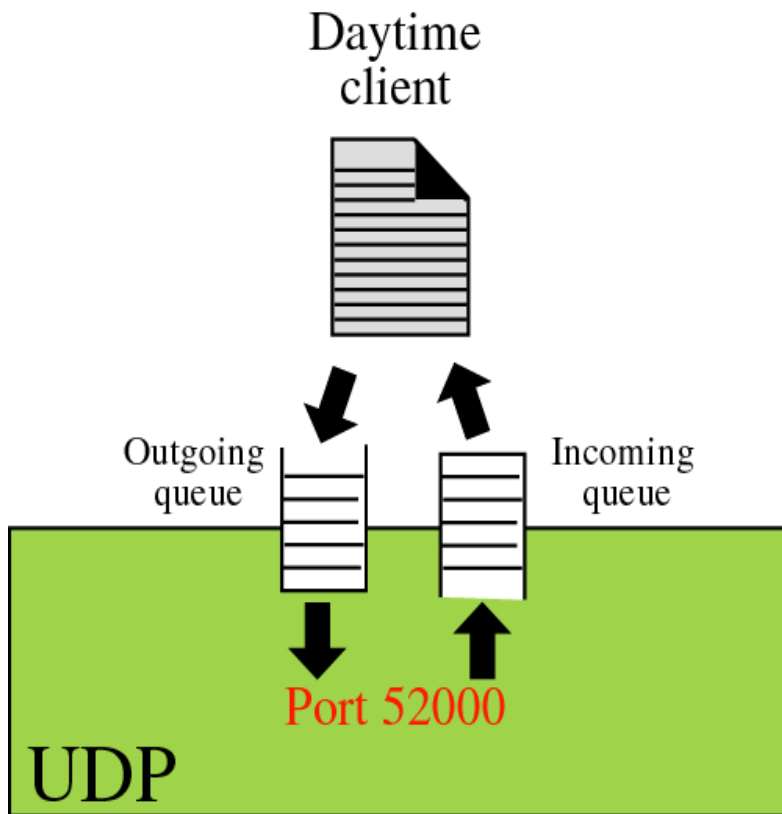


UDP Operation (cont'd)

▶ Queuing

- ▶ The queues function as long as the process is running.
- ▶ If an outgoing queue is happened overflow, the operating system can ask the client process to wait before sending any more messages.
- ▶ When a message arrives for a client, check an incoming queue. If there is no such incoming queue, UDP discard the user datagram and ask the ICMP protocols to send a port unreachable message to the server.
- ▶ At the server, the queues remain open as long as the server is running
 - ▶ An outgoing queue can overflow. If this happens, the operating system asks the server to wait before sending any more messages

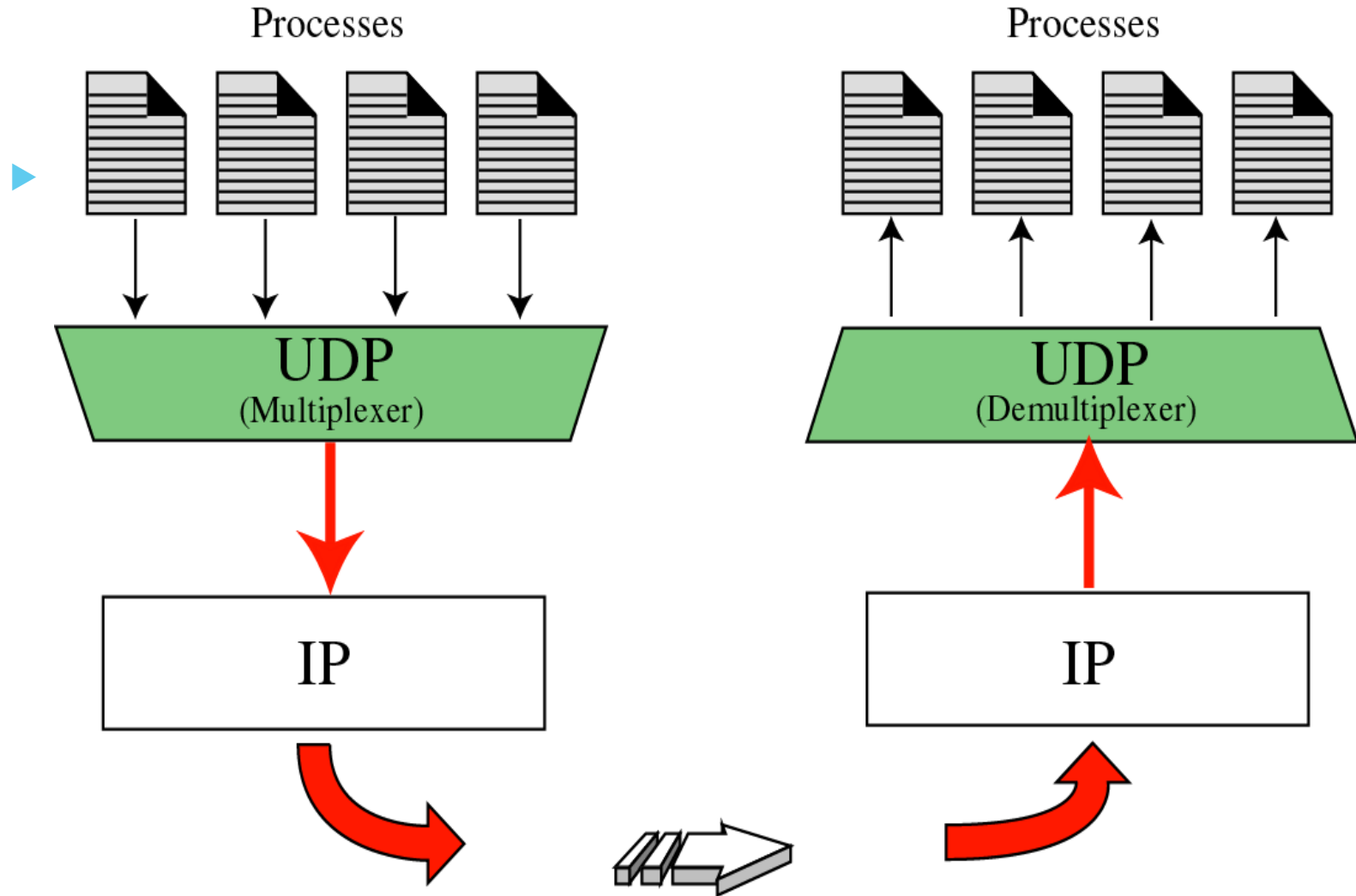
UDP Operation (cont'd)



UDP Operation (cont'd)

- ▶ Multiplexing
 - ▶ At the sender site, there may be several processes that need to send user datagrams
 - ▶ differentiating by their assigned port numbers
- ▶ Demultiplexing
 - ▶ At the receiver site, there is only one UDP
 - ▶ UDP receives user datagrams from IP.
 - ▶ After error checking and dropping of the header, UDP delivers each message to the appropriate port based on the port numbers

UDP Operation (cont'd)



11.5 Use of UDP

- ▶ The following lists some uses of the UDP protocols
 - ▶ suitable for a process that requires simple request-response communication and with little concern for flow and error control
 - ▶ not used for a process that needs to send bulk data, such as FTP
 - ▶ suitable for a process with internal flow and error control mechanisms
 - ▶ TFTP process including flow and error control
 - ▶ suitable transport protocol for multicasting and broadcasting
 - ▶ multicasting and broadcasting capabilities are embedded in the UDP software, but not in the TCP software

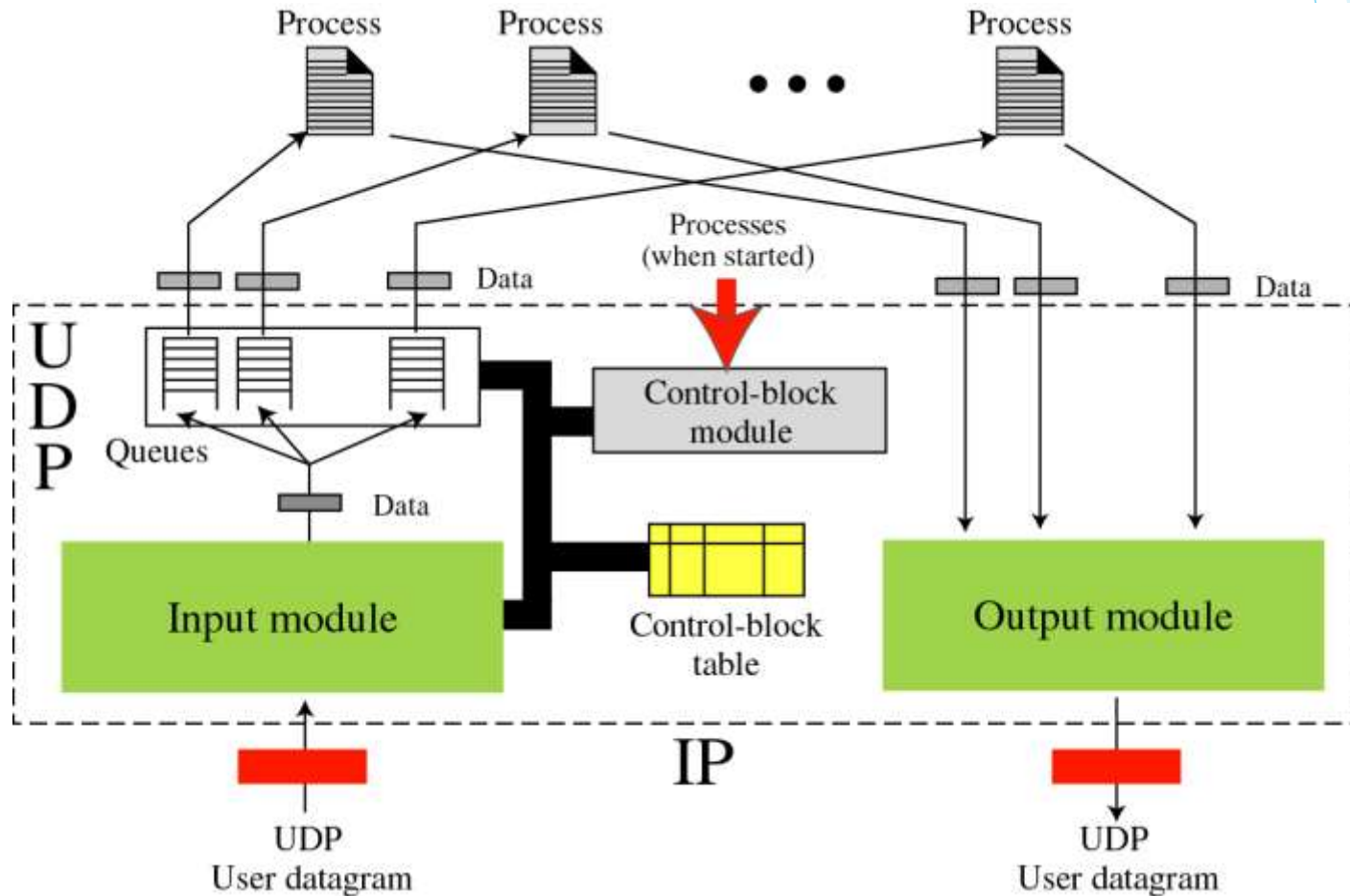
Use of UDP (cont'd)

- ▶ used for management protocol such as SNMP
- ▶ used for some route updating protocol such as RIP

UDP Design

Involving 5 components

a control-block table, input queues, a control-block module, an input module, and an output module



UDP Design (cont'd)

- ▶ Control-Block Table
 - ▶ Keeping track of the open ports
 - ▶ Each entry having 4 field: the state, which can be FREE or IN-USE, the process ID, the port number and the corresponding queue number
- ▶ Input Queues
 - ▶ One for each process
- ▶ Control-block Module
 - ▶ Responsible for management of the control-block table
 - ▶ When a process starts, it asks for a port number from the OS
 - ▶ OS assigns well-known port numbers to servers and ephemeral port number to clients

UDP Design (cont'd)

Control-Block Module

Receive: a process ID and a port number

1. Search the control block table for a FREE entry.
 1. If (not found)
 1. Delete an entry using a predefined strategy.
 2. Create a new entry with the state IN-USE.
 3. Enter the process ID and the port number.
2. Return.

UDP Design (cont'd)

Input Module

Receive: a user datagram from IP

1. Look for the corresponding entry in the control-block table.

1. If (found)

1. Check the queue field to see if a queue is allocated.

1. If (no)

1. Allocate a queue.

2. Enqueue the data in the corresponding queue.

2. If (not found)

1. Ask the ICMP module to send an “unreachable port” message.

2. Discard the user datagram.

2. Return.

UDP Design (cont'd)

▶ Output Module

Output Module

Receive: data and information from a process

1. Create a UDP user datagram.
2. Send the user datagram.
3. Return.

UDP Design (cont'd)

- ▶ Examples
 - ▶ Control block table at the beginning of examples

<i>State</i>	<i>Process ID</i>	<i>Port Number</i>	<i>Queue Number</i>
IN-USE	2,345	52,010	34
IN-USE	3,422	52,011	
FREE			
IN-USE	4,652	52,012	38
FREE			

UDP Design (cont'd)

- ▶ **Example 1**
 - ▶ Arrival of a user datagram with destination port number 52,012
 - ▶ The input module searches for this port number and finds it.
 - ▶ The input module sends the data to queue 38
 - ▶ The control block table does not change.

UDP Design (cont'd)

- ▶ **Example 2**
 - ▶ A process starts
 - ▶ Asking the OS for a port number and is granted port number 52,014
 - ▶ Sending process ID (4,978) and the port number to the control-block module to create an entry in the table
 - ▶ Taking the first FREE entry
 - ▶ No queue number assigned
 - ▶ No user diagrams have arrived for this destination

UDP Design (cont'd)

<i>State</i>	<i>Process ID</i>	<i>Port Number</i>	<i>Queue Number</i>
IN-USE	2,345	52,010	34
IN-USE	3,422	52,011	
IN-USE	4,978	52,014	
IN-USE	4,652	52,012	38
FREE			

UDP Design (cont'd)

- ▶ **Example 3,**
 - ▶ A user datagram now arrives 52,011
 - ▶ Input module checks the table and finds that no queue has been allocated for this destination
 - ▶ since this is the first time a user datagram has arrived for this destination
 - ▶ the module create a queue and gives it a number (43)

UDP Design (cont'd)

<i>State</i>	<i>Process ID</i>	<i>Port Number</i>	<i>Queue Number</i>
IN-USE	2,345	52,010	34
IN-USE	3,422	52,011	43
IN-USE	4,978	52,014	
IN-USE	4,652	52,012	38
FREE			

UDP Design (cont'd)

▶ Example 4

- ▶ After a few seconds, a user datagram arrives for port 52,222.
- ▶ Cannot find the entry for the this destination
- ▶ Dropping datagram and sending to the source ICMP message such as “unreachable port”

▶ Example 5

- ▶ After few seconds, process needs to send a user datagram. It delivers the data to the output module which adds the UDP header

TCP

1. Error Recovery

- ▶ In section 1, we first discuss *where* packet losses should be dealt with.
- ▶ In sections 2 and following we will discuss *how* this is implemented in the Internet in detail

The Philosophy of Errors in a Layered Model

- ▶ The physical layer is not completely error-free - there is always some bit error rate (BER).

Information theory tells us that for every channel there is a *capacity* C such that

- ▶ At any rate $R < C$, arbitrarily small BER can be achieved
- ▶ At rates $R \geq C$, any BER such that $H_2(\text{BER}) > 1 - C/R$ is achievable, with $H_2(p) = \text{entropy} = -p \log_2(p) - (1 - p) \log_2(1 - p)$

- ▶ The TCP/IP architecture *decided*

- ▶ Every layer i offers an **error free** service to the upper layer:

SDUs are either delivered without error or discarded

- ▶ Example: MAC layer

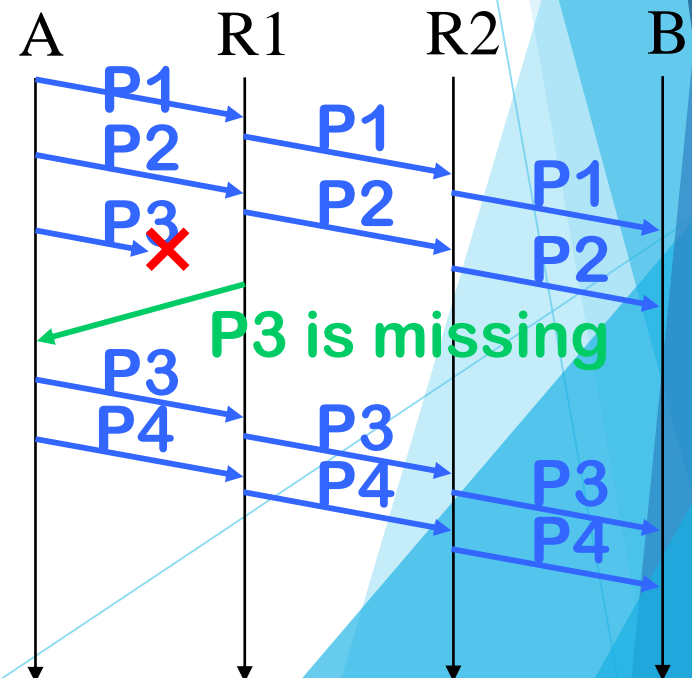
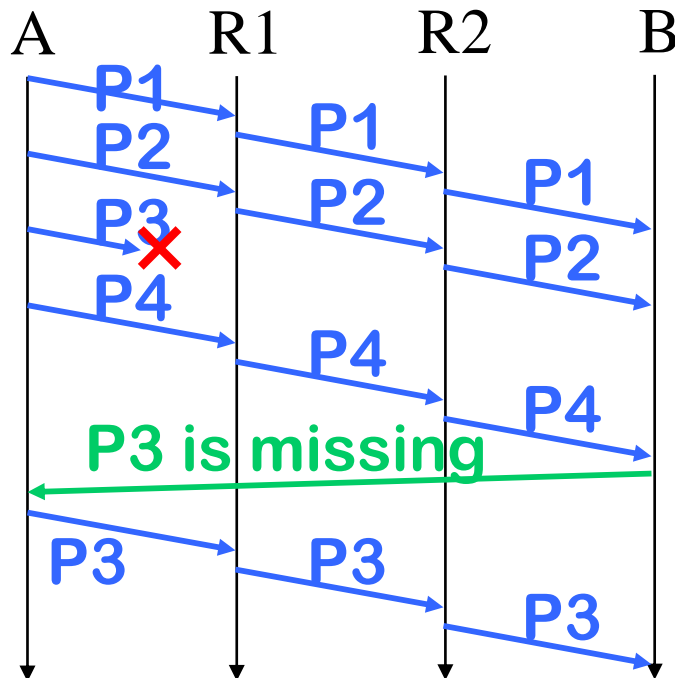
- ▶ Q1. How does an Ethernet adapter know whether a received Ethernet frames has some bit errors? What does it do with the frame?
- ▶ WiFi detects errors with CRC and does *retransmissions* if needed
- ▶ Q2. Why does not Ethernet do the same?

■ solution

The Layered Model Transforms Errors into Packet Losses

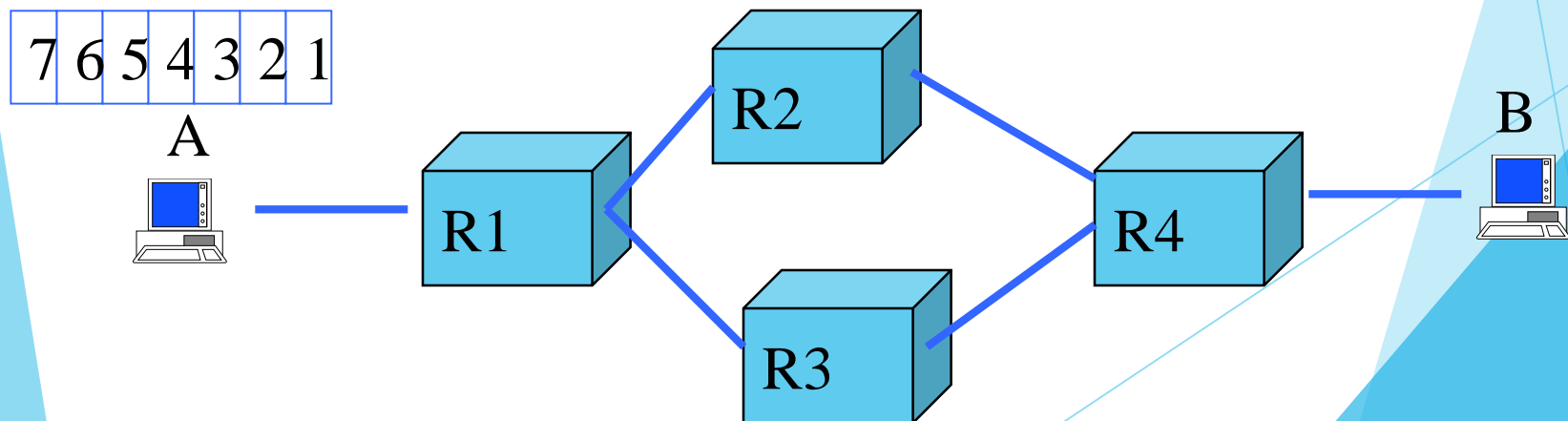
- ▶ Packet losses occur due to
 - ▶ error detection by MAC
 - ▶ *buffer* overflow in bridges and routers
 - ▶ Other exceptional errors may occur too
- Q. give some examples

- ▶ 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
 - ▶ IP packets may follow parallel paths, this is incompatible with hop-by-hop recovery. **R2 sees only 3 out of 7 packets but should not ask R1 for retransmission**



* The Case for Hop-By-Hop Error Recovery

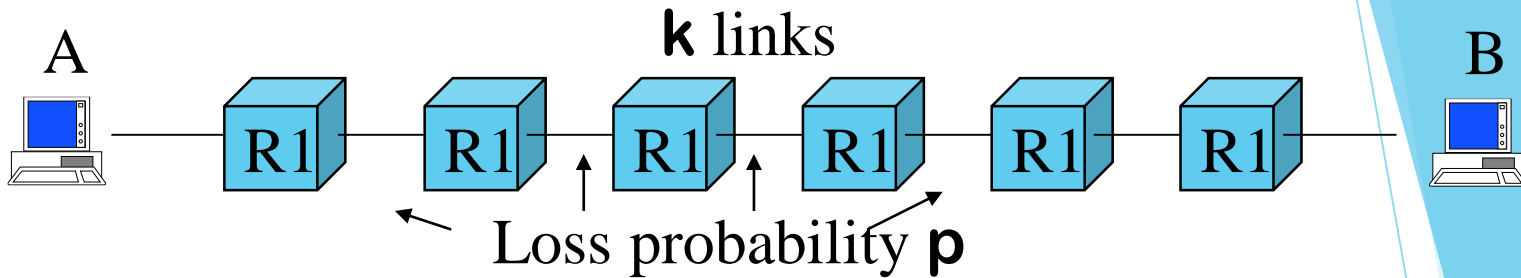
- ▶ There are also arguments in favour of hop-by-hop strategy. To understand them, we will use the following result.
 - ▶ *Capacity of erasure channel*: consider a channel with bit rate R that either delivers correct packets or loses them. Assume the loss process is stationary, such that the packet loss rate is $p \in [0, 1]$. The capacity is $R(1-p)$ packets/sec.

This means in practice that, for example, over a link at 10Mb/s that has a packet loss rate of 10% we can transmit 9 Mb/s of useful data.

The packet loss rate (PLR) can be derived from the bit error rate (BER) as follows, if bit errors are independent events, as a function of the packet length in bits L :

$$PLR = 1 - (1 - BER)^L$$

* The Capacity of the End-to-End Path



- ▶ We can now compute the capacity of an end-to-end path with both error recovery strategies.
 - ▶ Assumptions: same packet loss rate p on k links; same nominal rate R . Losses are independent.
 - ▶ Q. compute the capacity with end-to-end and with hop by hop error recovery.

solution

* End-to-end Error Recovery is Inefficient when Packet Error Rate is high

k	Packet loss rate	C_1 (end-to-end)	C_2 (hop-by-hop)
10	0.05	0.6 £ R	0.95 £ R
10	0.0001	0.9990 £ R	0.9999 £ R

- ▶ The table shows the capacity of an end-to-end path as a function of the packet loss rate p
- ▶ Conclusion: end-to-end error recovery is not acceptable when packet loss rate is high
- ▶ Q. How can one reconcile the conflicting arguments for and against hop-by-hop error recovery ?

[solution](#)

Conclusion: Where is Error Recovery located in the TCP/IP architecture ?

- ▶ The TCP/IP architecture assumes that
 1. The MAC layer provides error-free packets to the network layer
 2. The packet loss rate at the MAC layer (between two routers, or between a router and a host) must be made very small. It is the job of the MAC layer to achieve this.
 3. Error recovery must also be implemented end-to-end.

- ▶ Thus, packet losses are repaired
 - ▶ At the MAC layer on lossy channels (wireless)
 - ▶ In the end systems (transport layer or application layer).

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.

Why Sliding Window ?

► Why invented ?

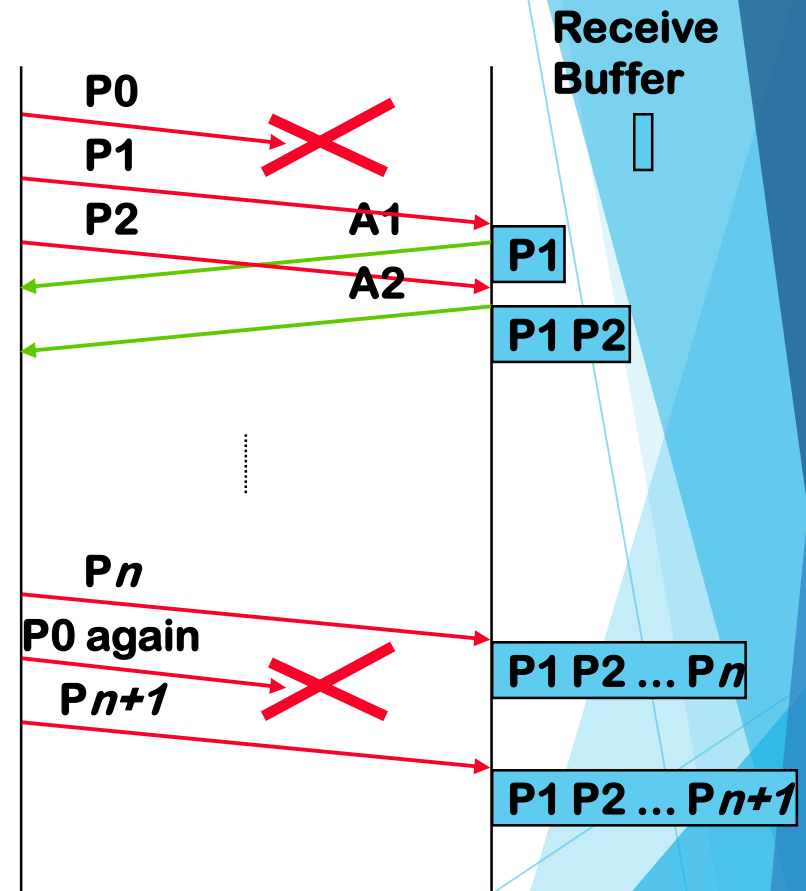
- Overcome limitations of Stop and Go

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

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

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

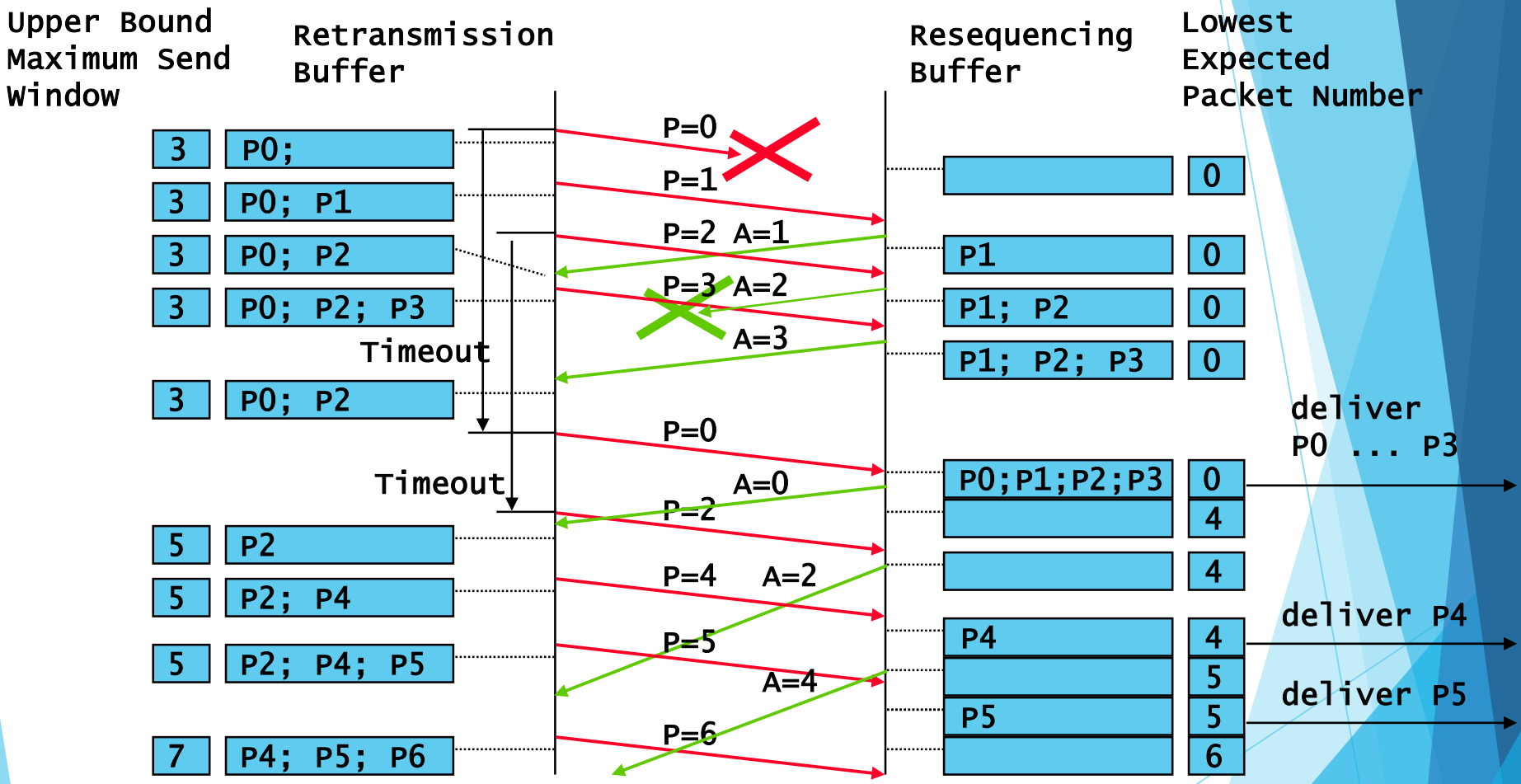
The sliding window principle works as follows:

- 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.
- 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 retransmitted (this is called *Selective Repeat*).

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

[solution](#)

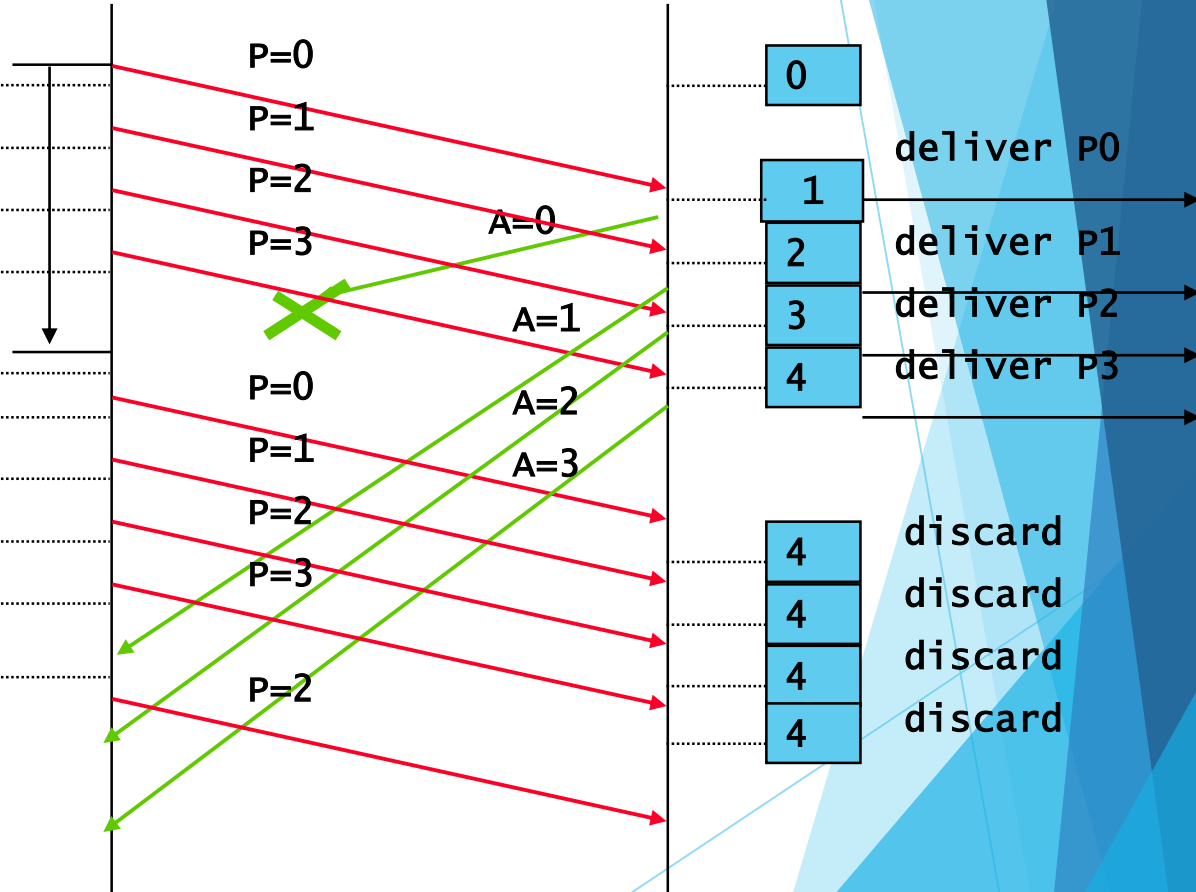
*An Example of ARQ Protocol with Go Back N

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?

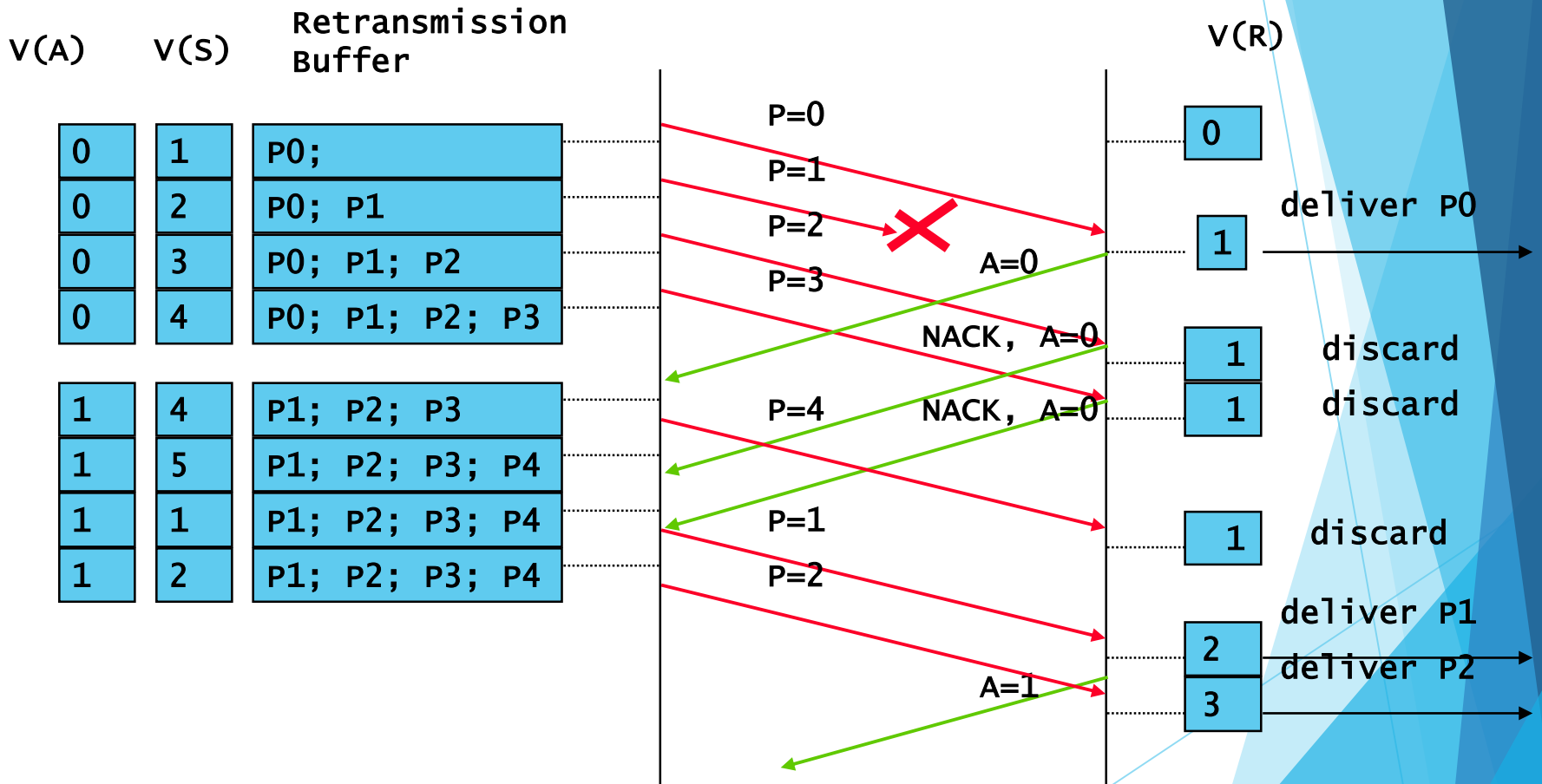
Solution

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

- ▶ *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



edits

Flow Control* ≠ *Congestion control

congestion control is about preventing too many losses inside the network

Backpressure Flow Control

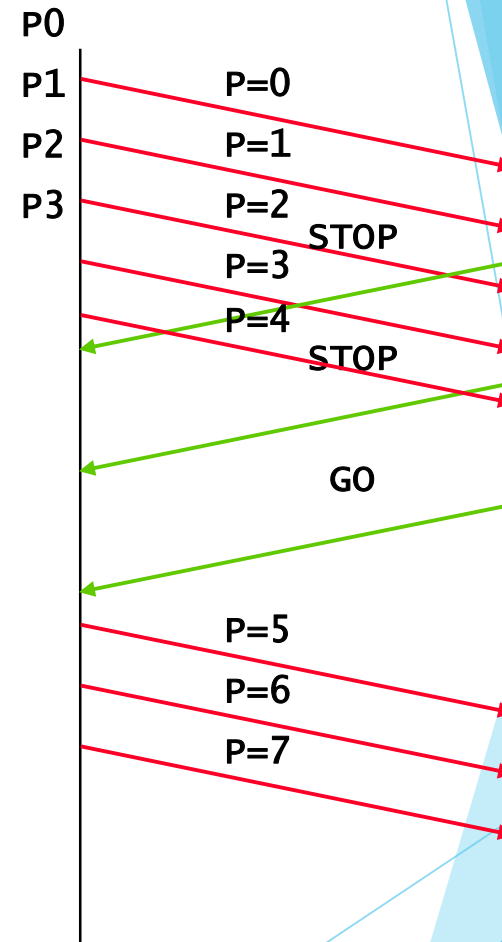
- ▶ Destination sends STOP (= PAUSE) or GO messages
- ▶ Source stops sending for x msec after receiving a STOP message
- ▶ Simple to implement
- ▶ Q. When does it work well ?

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)



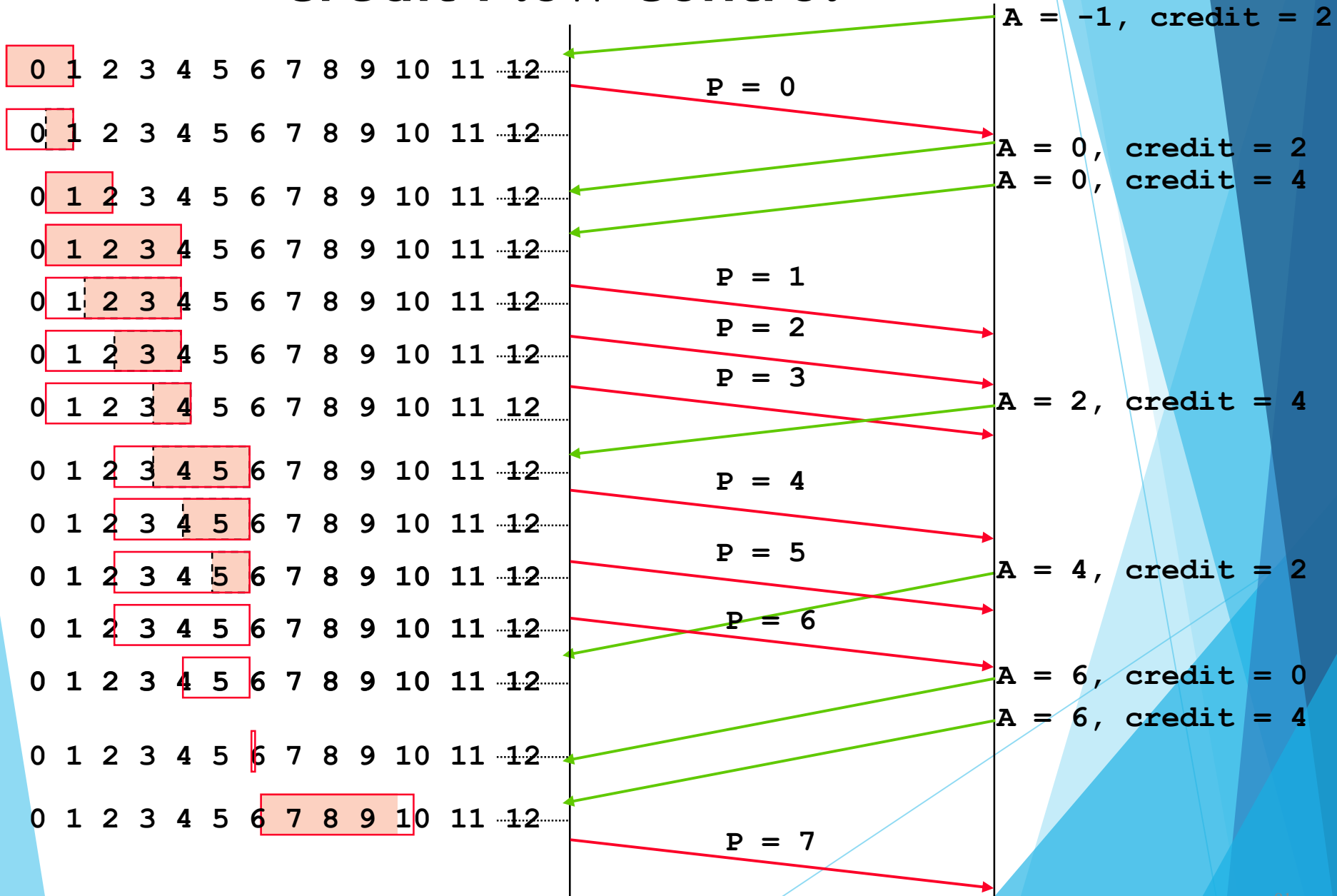
Can we use Sliding Window for Flow Control ?

- ▶ 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 ?

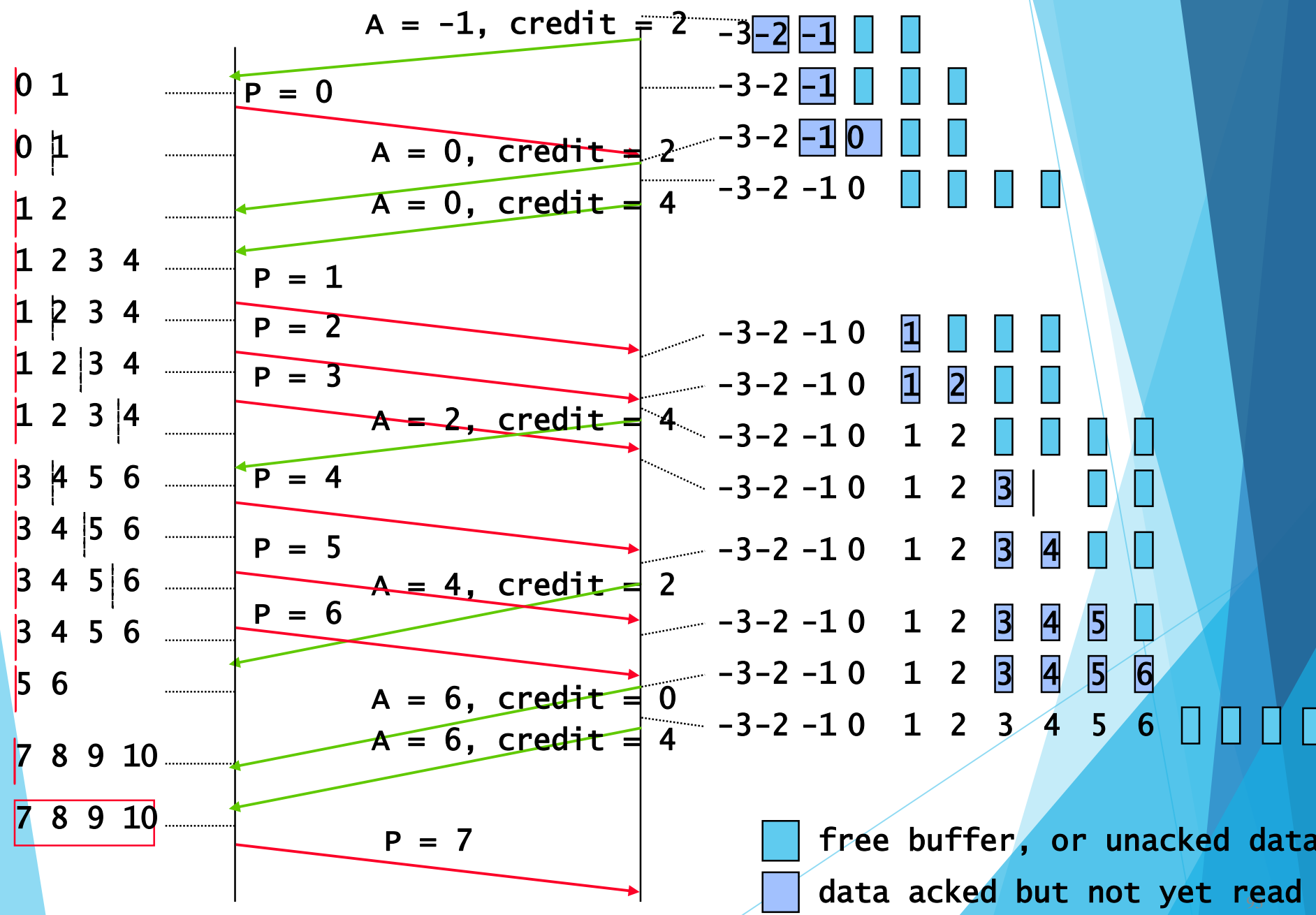
[solution](#)

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 advertised 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.

4. The Transport Layer

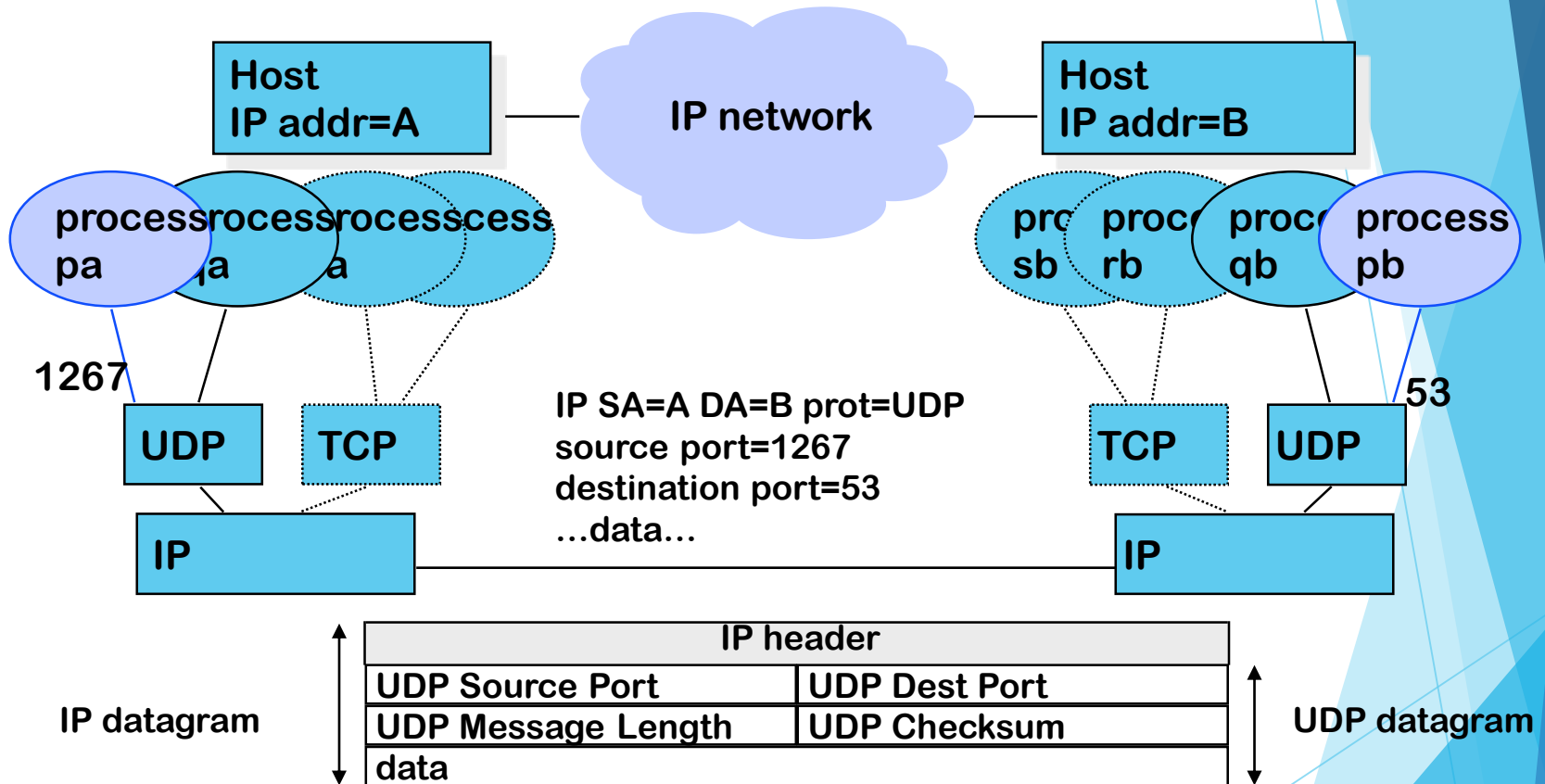
Reminder:

- ▶ network + link + phy carry packets end-to-end
- ▶ **transport layer** makes network services available to programs
- ▶ is in end systems only, not in routers
- ▶ In TCP/IP there are two transport layers
 - ▶ UDP (User Datagram Protocol): offers only a programming interface, no real function
 - ▶ TCP (Transmission Control Protocol): implements error recovery + flow control

Why both TCP and UDP ?

- ▶ Most applications use TCP rather than UDP, as this avoids re-inventing error recovery in every application
- ▶ But some applications do not need error recovery in the way TCP does it (i.e. by packet retransmission)
 - ▶ For example: Voice applications
Q. why ?
[solution](#)
 - ▶ For example: an application that sends just one message, like name resolution (DNS). TCP sends several packets of overhead before one single useful data message. Such an application is better served by a Stop and Go protocol at the application layer.
 - ▶ For example: multicast (TCP does not support multicast IP addresses)

UDP Uses Port Numbers



- ▶ The picture shows two processes (= application programs) pa , and pb , are communicating. Each of them is associated locally with a port, as shown in the figure.
- ▶ In addition, every machine (in reality: every communication adapter) has an IP address.
- ▶ The example shows a packet sent by the name resolver process at host A, to the name server process at host B. The UDP header contains the source and destination ports. The destination port number is used to contact the name server process at B; the source port is not used directly; it will be used in the response from B to A.
- ▶ The UDP header also contains a checksum to protect the UDP data plus the IP addresses and packet length. Checksum computation is not performed by all systems. Ports are 16 bits unsigned integers. They are defined statically or dynamically. Typically, a server uses a port number defined statically.
- ▶ Standard services use well-known ports; for example, all DNS servers use port 53 (look at `/etc/services`). Ports that are allocated dynamically are called ephemeral. They are usually above 1024. If you write your own client server application on a multiprogramming machine, you need to define your own server port number and code it into your application.

The UDP service

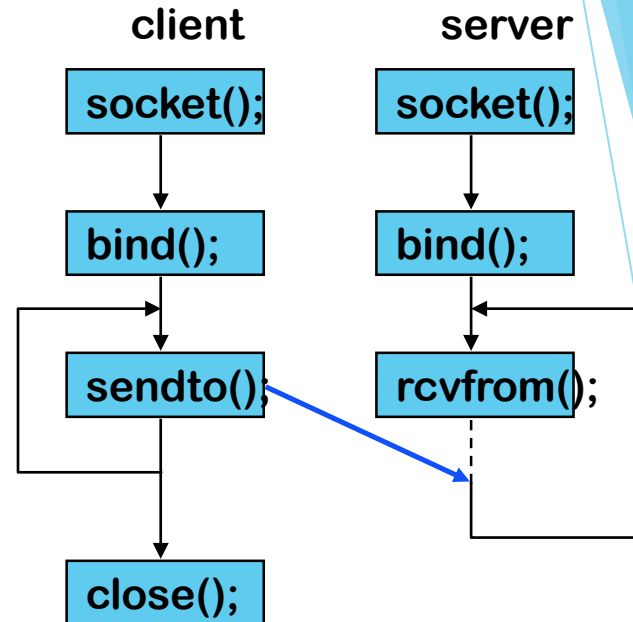
- ▶ UDP service interface
 - ▶ one message, up to 8K
 - ▶ destination address, destination port, source address, source port
- ▶ UDP service is message oriented
 - ▶ delivers exactly the message or nothing
 - ▶ several messages may be delivered in disorder
 - ▶ Message may be lost, application must implement loss recovery.
- ▶ If a UDP message is larger than MTU, then fragmentation occurs at the IP layer

UDP is used via a Socket Library

▶ The socket library provides a programming interface to TCP and UDP

▶ The figure shows toy client and server UDP programs. The client sends one string of chars to the server, which simply receives (and displays) it.

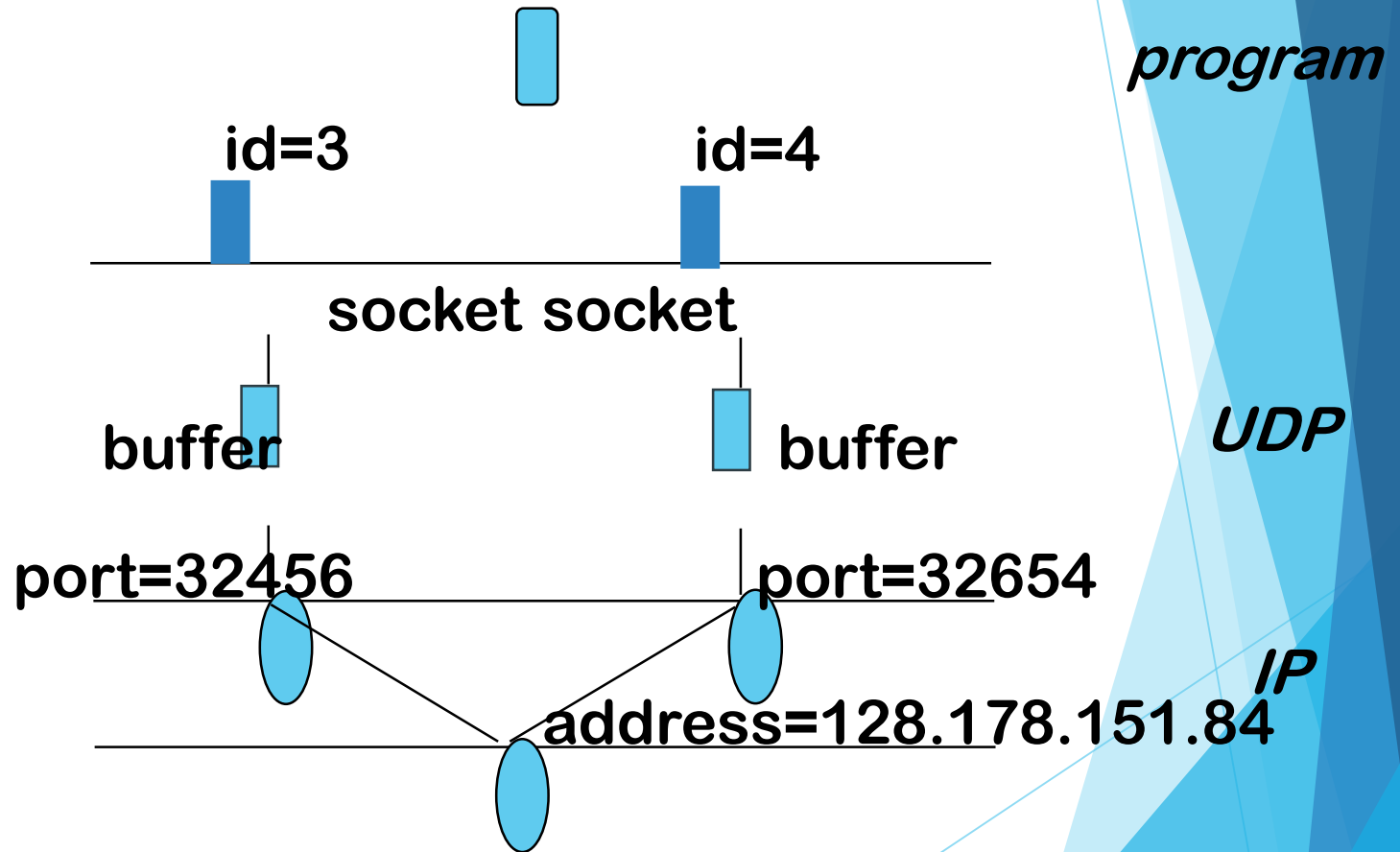
- ▶ `socket()` creates a socket and returns a number (=file descriptor) if successful
- ▶ `bind()` associates the local port number with the socket
- ▶ `sendto()` gives the destination IP address, port number and the message to send
- ▶ `recvFrom()` blocks until one message is received for this port number. It returns the source IP address and port number and the message.



```
% ./udpClient <destAddr> bonjour les amis
%
```

```
% ./udpServ &
%
```

How the Operating System views UDP



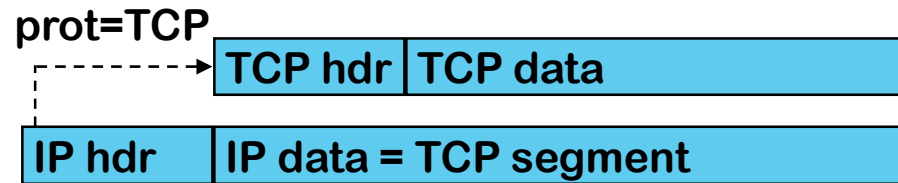
5. TCP basics

- ▶ *Why* invented ?
 - ▶ Repair packet losses
 - ▶ Save application from doing it.
- ▶ *What* does TCP do ?
 - ▶ TCP guarantees that all data is delivered in sequence and without loss, unless the connection is broken;
 - ▶ TCP should work for all applications that transfer data, either in small or large quantities
 - ▶ TCP does not work with multicast IP addresses, UDP does.
 - ▶ TCP also does flow control
 - ▶ TCP also does congestion control (not seen in this module)
- ▶ *How* does TCP work ?
 - ▶ first, a connection (=synchronization of sequence numbers) is opened between two processes
 - ▶ then TCP implements ARQ (for error recovery) and credits (for flow control)
 - ▶ in the end, the connection is closed

The TCP Service

- ▶ TCP offers a *stream* service
 - ▶ A stream of bytes is accepted for transmission and delivered at destination
 - ▶ TCP uses port numbers like UDP eg. TCP port 80 is used for web server.
 - ▶ TCP requires that a connection is opened before data can be transferred. A TCP connection is identified by: **srce IP addr, srce port, dest IP addr, dest port**

TCP views data as a stream of bytes

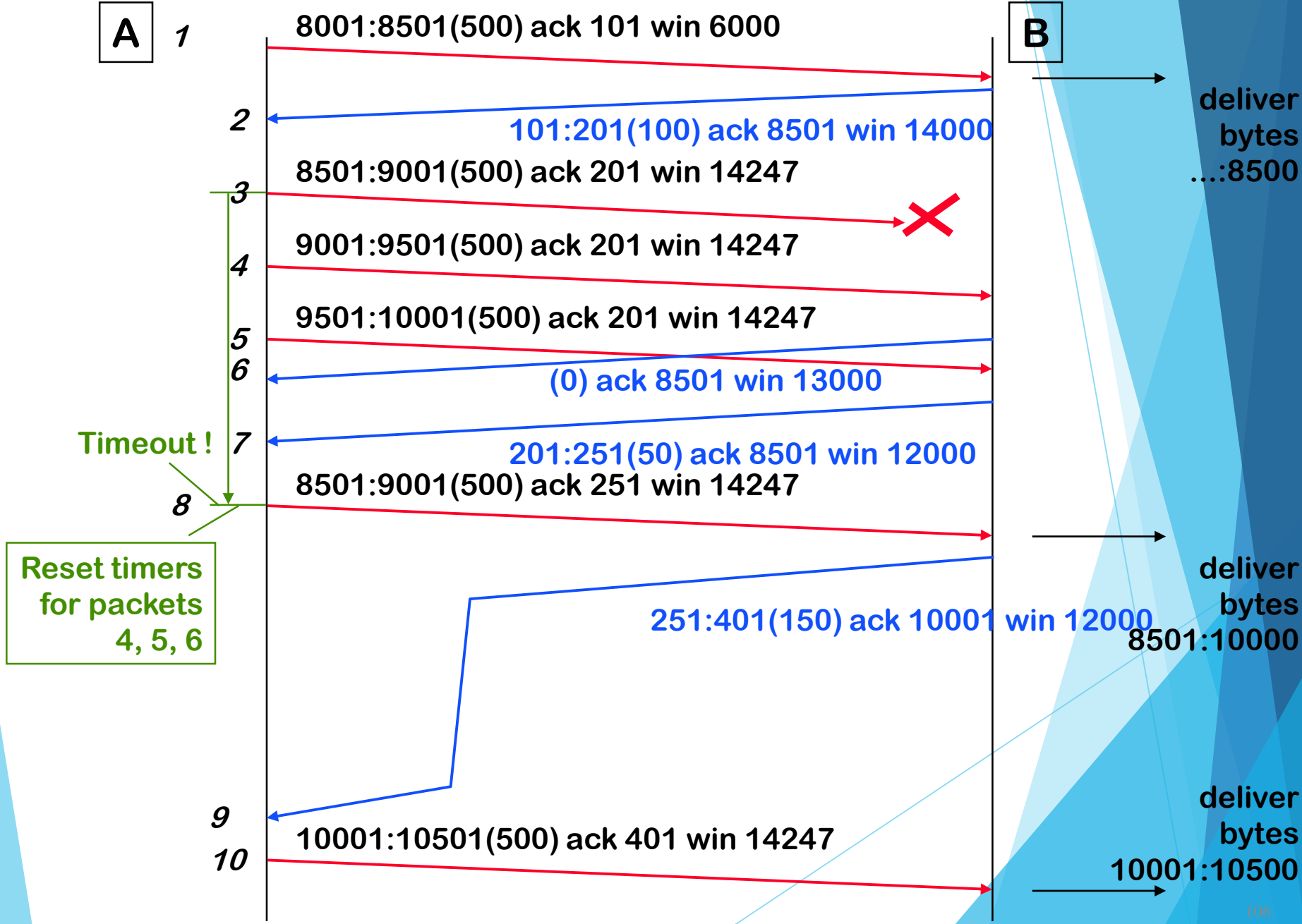


- ▶ TCP-PDUs are called TCP segments
 - ▶ bytes accumulated in buffer until sending TCP decides to create a segment
 - ▶ MSS = maximum “segment“ size (maximum data part size)
 - ▶ “B sends MSS = 236” means that segments, without header, sent to B should not exceed 236 bytes
 - ▶ 536 bytes by default (576 bytes IP packet)
- ▶ Sequence numbers based on byte counts, not packet counts
- ▶ TCP builds segments independent of how application data is broken
 - ▶ unlike UDP
- ▶ TCP segments never fragmented at source
 - ▶ possibly at intermediate points with IPv4
 - ▶ where are fragments re-assembled ?

TCP is an ARQ protocol

- ▶ **Basic operation:**
 - ▶ sliding window
 - ▶ loss detection by timeout at sender
 - ▶ retransmission is a hybrid of go back and selective repeat
 - ▶ Cumulative acks
- ▶ **Supplementary elements**
 - ▶ fast retransmit
 - ▶ selective acknowledgements
- ▶ **Flow control is by credit**
- ▶ **Congestion control**
 - ▶ adapt to network conditions

TCP Basic Operation

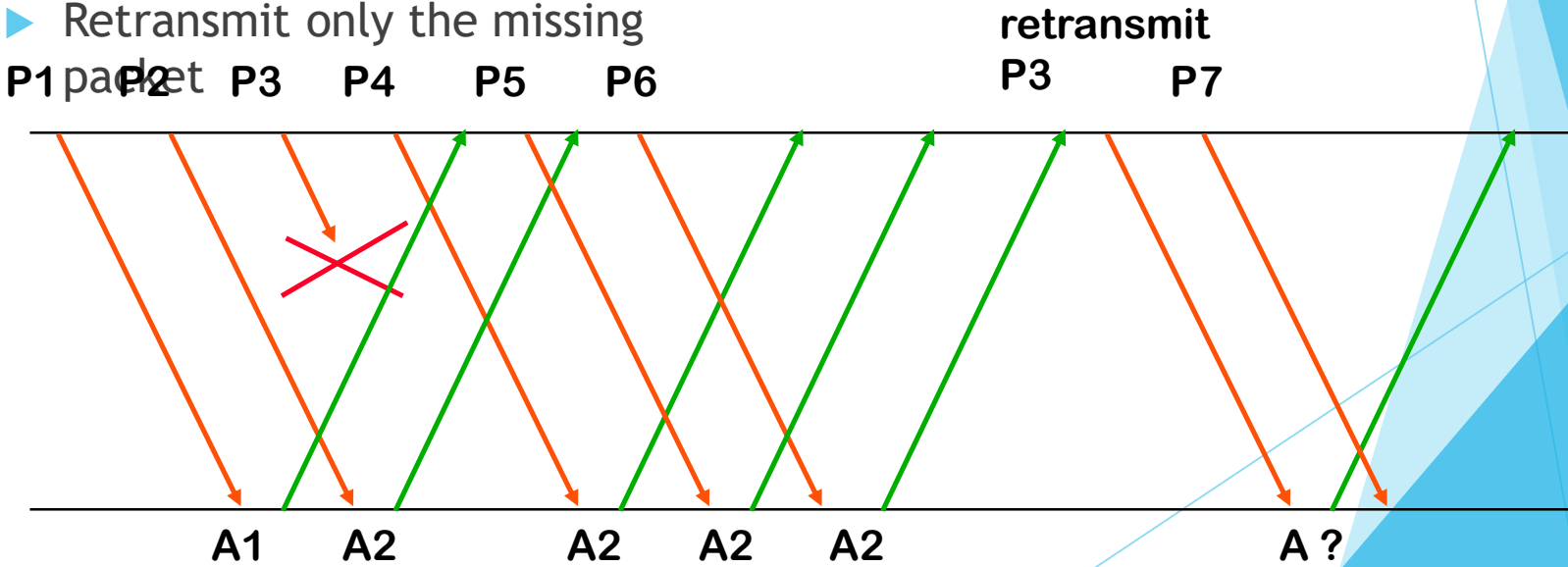


- ▶ The picture shows a sample exchange of messages. Every packet carries the sequence number for the bytes in the packet; in the reverse direction, packets contain the acknowledgements for the bytes already received in sequence. The connection is bidirectional, with acknowledgements and sequence numbers for each direction. Acknowledgements are not sent in separate packets (“*piggybacking*”), but are in the TCP header. Every segment thus contains a sequence number (for itself), plus an ack number (for the reverse direction). The following notation is used:
 - ▶ “firstByte” : “lastByte+1” (“segmentDataLength”) ack” ackNumber+1 “win” offeredWindowSize. Note the +1 with ack and lastByte numbers.
- ▶ At line 8, a retransmission timer expires, causing the retransmission of data starting with byte number 8501 (Go Back n principle). Note however that after segment 9 is received, transmission continues with byte number 10001. This is because the receiver stores segments received out of order.
- ▶ The window field (win) gives to the sender the size of the window. Only byte numbers that are in the window may be sent. This makes sure the destination is not flooded with data it cannot handle.
- ▶ Note that numbers on the figure are rounded for simplicity. Real examples use non-round numbers between 0 and $2^{32} - 1$. The initial sequence number is not 0, but is chosen at random using a 4 μ sec clock.
- ▶ The figure shows the implementation of TCP known as “TCP SACK”, which is the basis for current implementations. An earlier implementation (“TCP Tahoe”) did not reset the pending timers after a timeout; thus, this was implementing a true Go Back n protocol; the drawback was that packets were retransmitted unnecessarily, because packet losses are usually simple.

Losses are Also Detected by “Fast Retransmit”

- ▶ Why invented: retransmission timeout in practice often very approximate thus timeout is often too large. Go back n is less efficient than SRP
- ▶ What it does
 - ▶ Detect losses earlier

- ▶ How it works
 - ▶ if 3 duplicate acks for the same bytes are received before retransmission timeout, then retransmit
- Q. which ack is sent last on the figure ?
- [solution](#)



Selective Acknowledgements

- ▶ *Why invented ?*
 - ▶ Fast retransmit works well if there is one isolated loss, not if there are a few isolated losses
- ▶ *What does it do ?*
 - ▶ Acknowledge exactly which bytes are received and allow their selective retransmission
- ▶ *How does it do it ?*
 - ▶ up to 3 SACK blocks are in TCP option, on the return path; a SACK block is a positive ack for an interval of bytes; first block is most recently received
 - ▶ Sent by destination when : new data is received that does not increase ack field
 - ▶ source to detect a loss by gap in received acknowledgement
 - ▶ If gap detected, missing bytes are retransmitted

TCP uses Connections

- ▶ TCP requires that a connection (= synchronization) is opened before transmitting data
 - ▶ Used to agree on sequence numbers
- ▶ The next slide shows the states of a TCP connection:
 - ▶ Before data transfer takes place, the TCP connection is opened using SYN packets. The effect is to synchronize the counters on both sides.
 - ▶ The initial sequence number is a random number.
 - ▶ The connection can be closed in a number of ways. The picture shows a graceful release where both sides of the connection are closed in turn.
 - ▶ Remember that TCP connections involve only two hosts; routers in between are not involved.

TCP Connection Phases

*application
active open*

listen *passive open*

Connection
Setup

Data Transfer

Connection
Release

syn_sent

SYN, seq=x

snc_rcvd

SYN seq=y, ack=x+1

established

ack=y+1

established

active close

fin_wait_1

FIN, seq=u

close_wait

ack=u+1

fin_wait_2

FIN seq=v

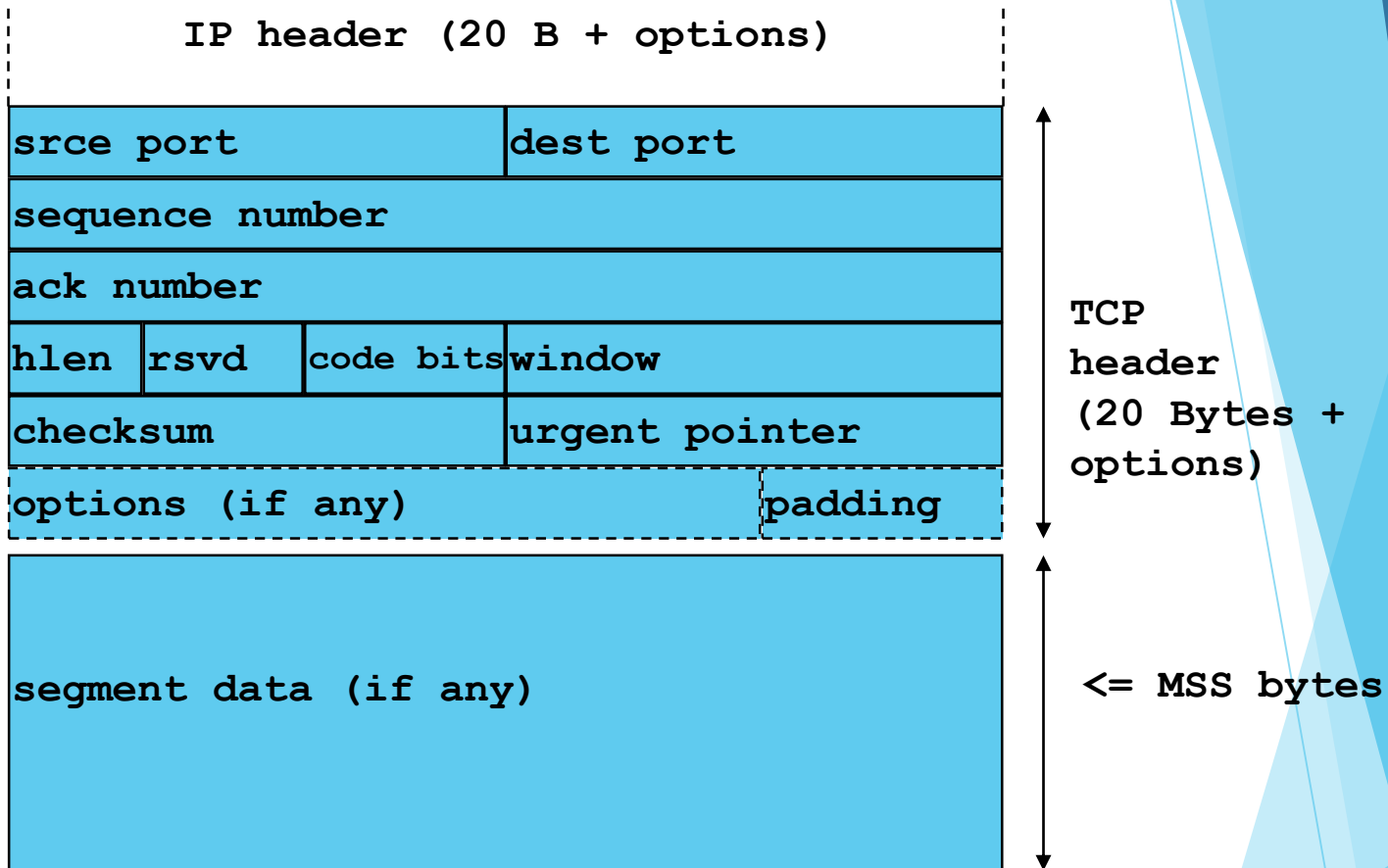
application close:

time_wait

ack=v+1

last_ack

closed



<u>code bit</u>	<u>meaning</u>
urg	urgent ptr is valid
ack	ack field is valid
psh	this seg requests a push
rst	reset the connection
syn	connection setup
fin	sender has reached end of byte stream

*TCP Segment Format

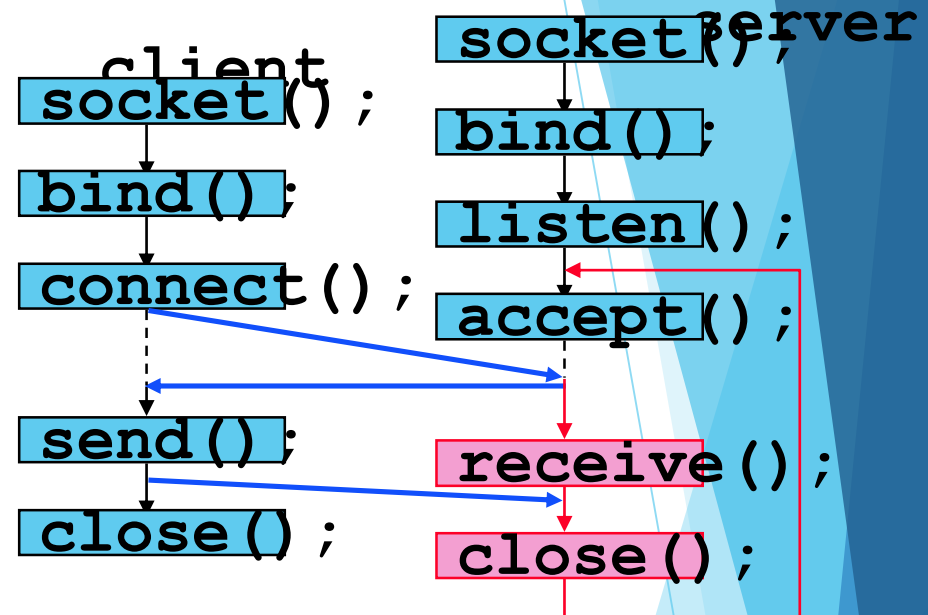
The next slide shows the TCP segment format.

- | the push bit can be used by the upper layer using TCP; it forces TCP on the sending side to create a segment immediately. If it is not set, TCP may pack together several SDUs (=data passed to TCP by the upper layer) into one PDU (= segment). On the receiving side, the push bit forces TCP to deliver the data immediately. If it is not set, TCP may pack together several PDUs into one SDU. This is because of the stream orientation of TCP. TCP accepts and delivers contiguous sets of bytes, without any structure visible to TCP. The push bit used by Telnet after every end of line.
- | the urgent bit indicates that there is urgent data, pointed to by the urgent pointer (the urgent data need not be in the segment). The receiving TCP must inform the application that there is urgent data. Otherwise, the segments do not receive any special treatment. This is used by Telnet to send interrupt type commands.
- | RST is used to indicate a RESET command. Its reception causes the connection to be aborted.
- | SYN and FIN are used to indicate connection setup and close. They each consume one sequence number.
- | The sequence number is that of the first byte in the data. The ack number is the next expected sequence number.
- | Options contain for example the Maximum Segment Size (MSS) normally in SYN segments (negotiation of the maximum size for the connection results in the smallest value to be selected).
- | The checksum is mandatory.

TCP is used via a Socket Library

▶ The figure shows toy client and servers. The client sends a string of chars to the server which reads and displays it.

- ▶ `socket()` creates a socket and returns a number (=file descriptor) if successful
- ▶ `bind()` associates the local port number with the socket
- ▶ `connect()` associates the remote IP address and port number with the socket and sends a SYN packet
- ▶ `send()` sends a block of data to the remote destination
- ▶ `listen()` can be omitted at first reading; `accept()` blocks until a SYN packet is received for this local port number. It creates a new socket (in pink) and returns the file descriptor to be used to interact with this new socket
- ▶ `receive()` blocks until one block of data is ready to be consumed on this port number. You must tell in the argument of `receive` how many bytes at most you want to read. It returns the number of bytes that is effectively returned and the block of data.

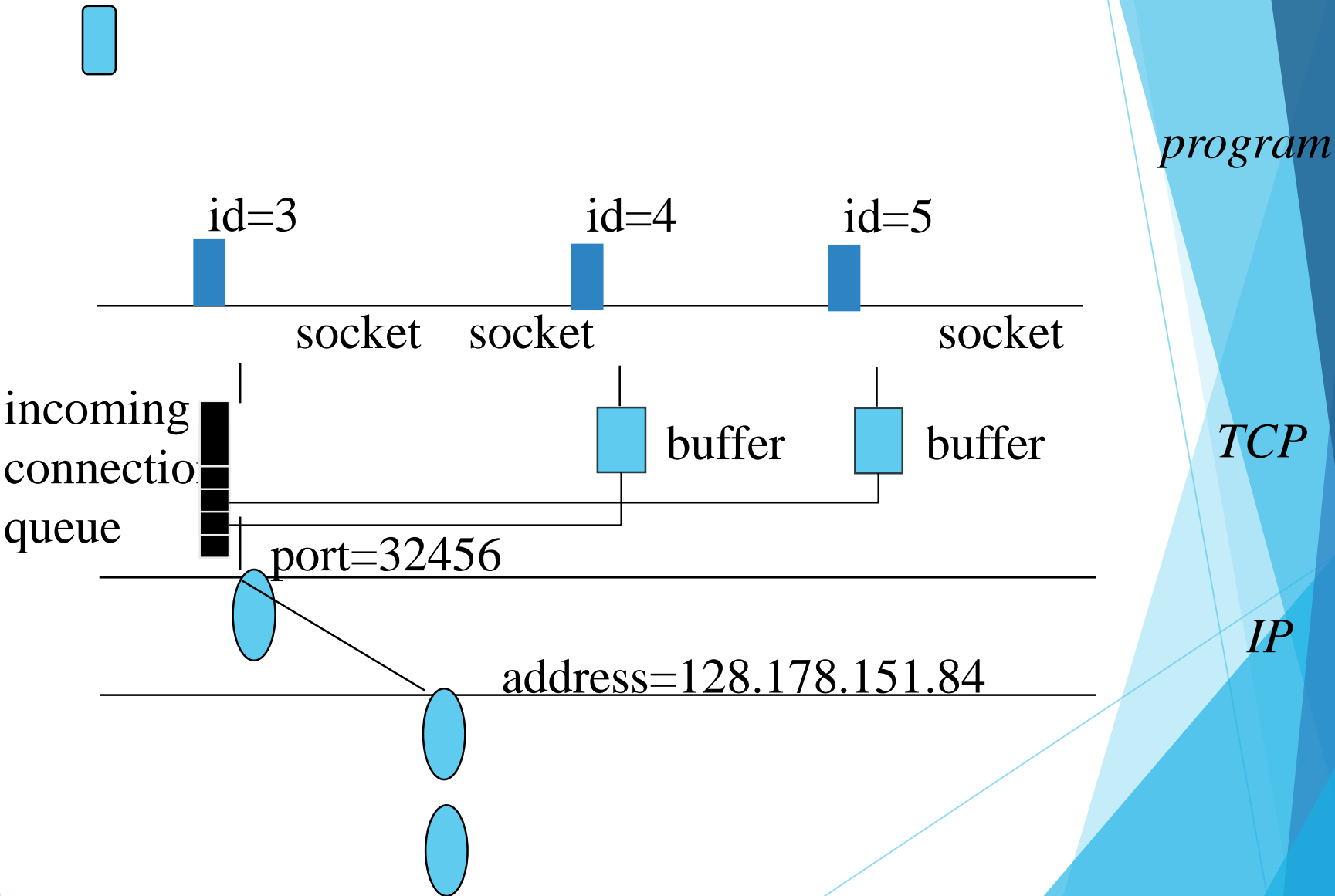


```
% ./tcpClient
<destAddr> bonjour
les amis
```

```
% ./tcpServ &
```

```
%
```

How the Operating System views TCP Sockets



Test Your Understanding

- ▶ Consider the UDP and TCP services

Q1. what does service mean here ?

Q2. does UDP transfer the blocks of data delivered by the calling process as they were submitted ? Analyze: delineation, order, missing blocks.

Q3. does TCP transfer the messages delivered by the calling process as they were submitted ? Analyze: delineation, order, missing blocks.

- ▶ One more question

Q4. Is Stop and Go a sliding window protocol ?

[solution](#)

6. TCP, advanced

- ▶ TCP implements a large number of additional mechanisms. Why ?

1. *The devils' in the detail*

Doing ARQ and flow control the right way poses a number of small problems that need to be solved. We give some examples in the next slides.

This will give you a feeling for the complexity of the real TCP code. Note that there are many other details in TCP, not shown in this lecture.

2. *Congestion control is done in TCP*

Congestion control is a network layer function (avoid congestion in the network) that the IETF decided to implement in TCP - we discuss why in the module on congestion control cc.pdf. We do not consider congestion control in this module.

When to send an ACK

- ▶ *Why* is there an issue ?
 - ▶ When receiving a data segment, a TCP receiver may send an acknowledgement immediately, or may wait until there is data to send (“piggybacking”), or until other segments are received (cumulative ack). Delaying ACKs reduces processing at both sender and receiver, and may reduce the amount of IP packets in the network. However, if ACKs are delayed too long, then receivers do not get early feedback and the performance of the ARQ scheme decreases. Also, delaying ACKs also delays new information about the window size.
- ▶ *What* is this algorithm doing ?
 - ▶ Decide when to send an ACK and when not.
- ▶ *How* does it do its job ?
 - ▶ Sending an ACK is delayed by at most 0.5 s. In addition, in a stream of full size segments, there should be at least one ACK for every other segment.
 - ▶ Note that a receiving TCP should send ACKs (possibly delayed ACKs) even if the received segment is out of order. In that case, the ACK number points to the last byte received in sequence + 1.

Nagle's Algorithm

▶ **Why** is there an issue ?

- ▶ A TCP source can group several blocks of data -- passed to it by `sendto()` -- into one single segment. This occurs when the application receives very small blocks to transmit (ex: Telnet: 1 char at a time). Grouping saves processing and capacity when there are many small blocks to transmit, but adds a delay.

▶ **What** is this algorithm doing ?

- ▶ Decide when to create a segment and pass it the IP layer for transmission.

▶ **How** does it do its job ?

- ▶ accept only one unacknowledged tinygram (= segment smaller than MSS):
- ▶ Nagle's algorithm can be disabled by application
 - ▶ example: X window system (TCP_NODELAY socket option)
 - ▶ if Nagle enabled, then applies also to pushed data

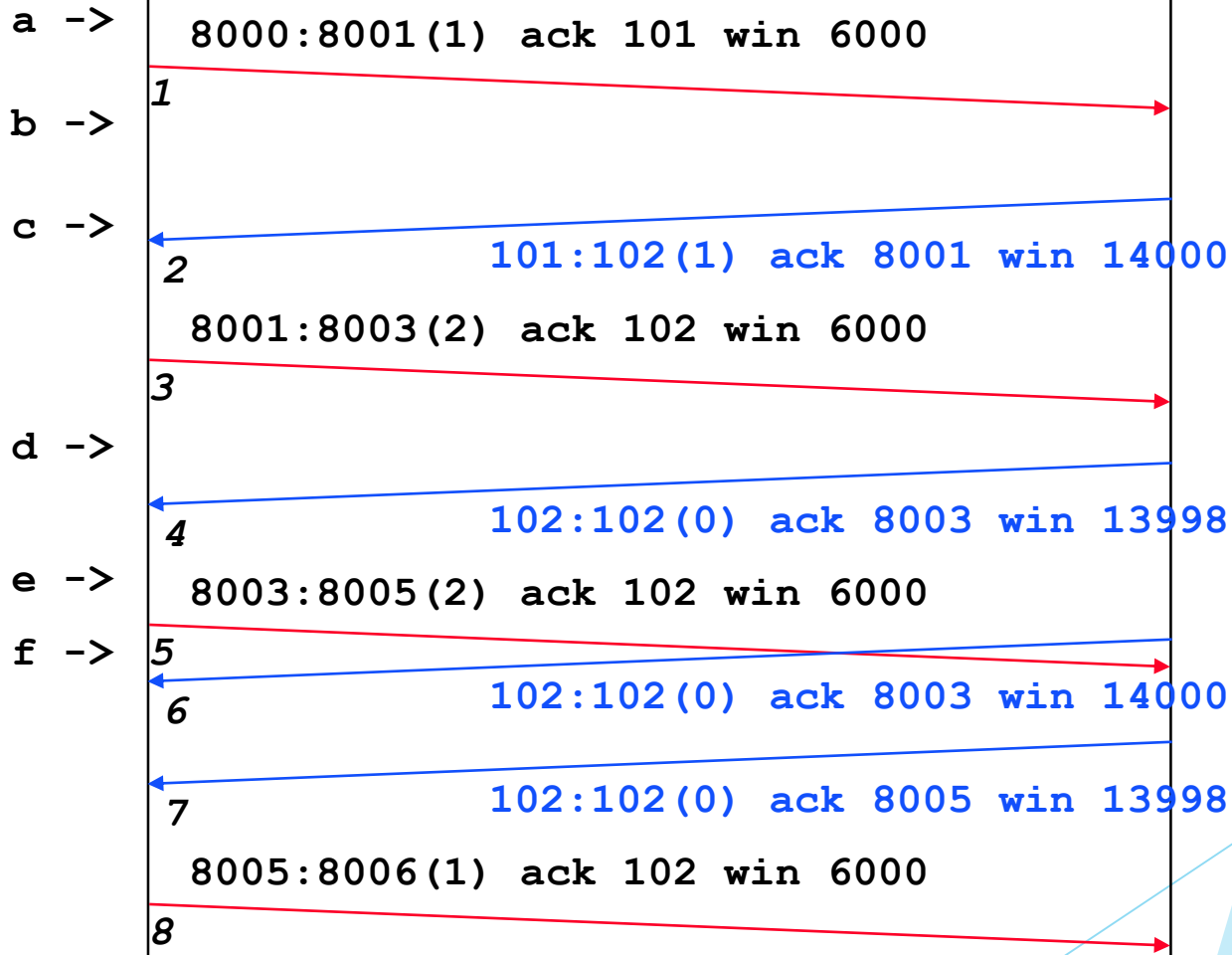
(data written by upper layer) or (new ack received) ->

```
if full segment ready  
then send segment  
else if there is no unacknowledged data  
then send segment  
else start override timer; leave  
override timer expires -> create segment and send
```

Example: Nagle's algorithm

A

B



Silly Window Syndrome Avoidance: Why ?

- ▶ Silly Window Syndrome occurs when
 - ▶ Receiver is slow or busy
 - ▶ sender has large amount of data to send
 - ▶ but small window forces sender to send many small packets -> waste of resources

ack 0 win 2000 <-----

0:1000 -----> bufferSize= 2000B, freebuf= 1000B

1000:2000 -----> freebuf= 0B

ack 2000, win 0 <-----

application reads 1 Byte: freeBuf = 1

ack 2000, win 1 <-----

2000:2001 -----> freeBuf = 0

application reads 1 Byte: freeBuf = 1

ack 2001, win 1 <-----

2001:2002 -----> freeBuf = 0

application reads 1 Byte: freeBuf = 1

ack 2002, win 1 <-----

2002:2003 -----> freeBuf = 0

Silly Window Syndrome Avoidance

▶ *What* does SWS avoidance do ?

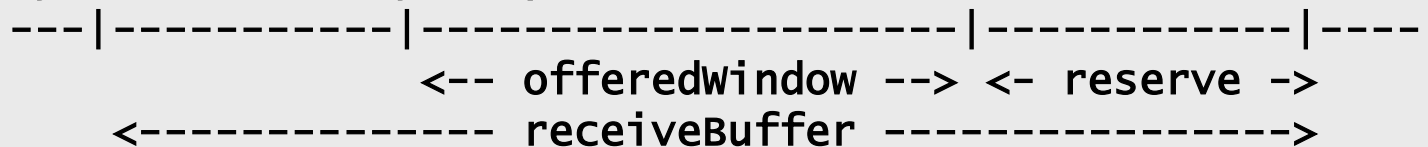
▶ Prevent receiver from sending small incremental window updates

▶ *How* does SWS avoidance work ?

receiver moves the window by increments that are as large as one MSS or 1/2 receiveBuffer:

**keep nextByteExpected + offeredWindow fixed until:
reserve · min (MSS, 1/2 receiveBuffer)**

highestByteRead nextByteExpected



SWS Avoidance Example

```
ack 0 win 2000 <-----  
    0:1000 -----> bufferSize= 2000B, freebuf = 1000B, reserve = 0B  
    1000:2000 -----> freebuf= 0B, reserve = 0B  
ack 2000, win 0 <-----  
    application reads 1 Byte: freeBuf=reserve=1B,  
    ....  
    application has read 500 B: reserve = 500  
persistTimer expires  
window probe packet sent  
  
    2000:2001 ----->  
    data is not accepted (out of window)  
ack 2000, win 0 <-----  
    ....  
    application has read 1000 B: reserve = 1000  
ack 2000, win 1000 <-----  
    2000:3000 ----->
```

- ▶ There is also a SWS avoidance function at sender
 - ▶ *Why?* Cope with destinations that do not implement SWS avoidance at receiver - see the RFCs for what and how
- ▶ **Q.** What is the difference in objective between Nagle's algorithm and SWS avoidance ?

[solution](#)

Round Trip Estimation

- ▶ **Why** ? The retransmission timer must be set at a value slightly larger than the round trip time, but too much larger
- ▶ **What** ? RTT estimation computes an upper bound RTO on the round trip time

▶ **How** ?

sampleRTT = last measured round trip time
estimatedRTT = last estimated average round trip time
deviation = last estimated round trip *deviation*

initialization (first sample):

estimatedRTT = **sampleRTT** + 0.5s; **deviation** = **estimatedRTT**/2

new value of sampleRTT available ->

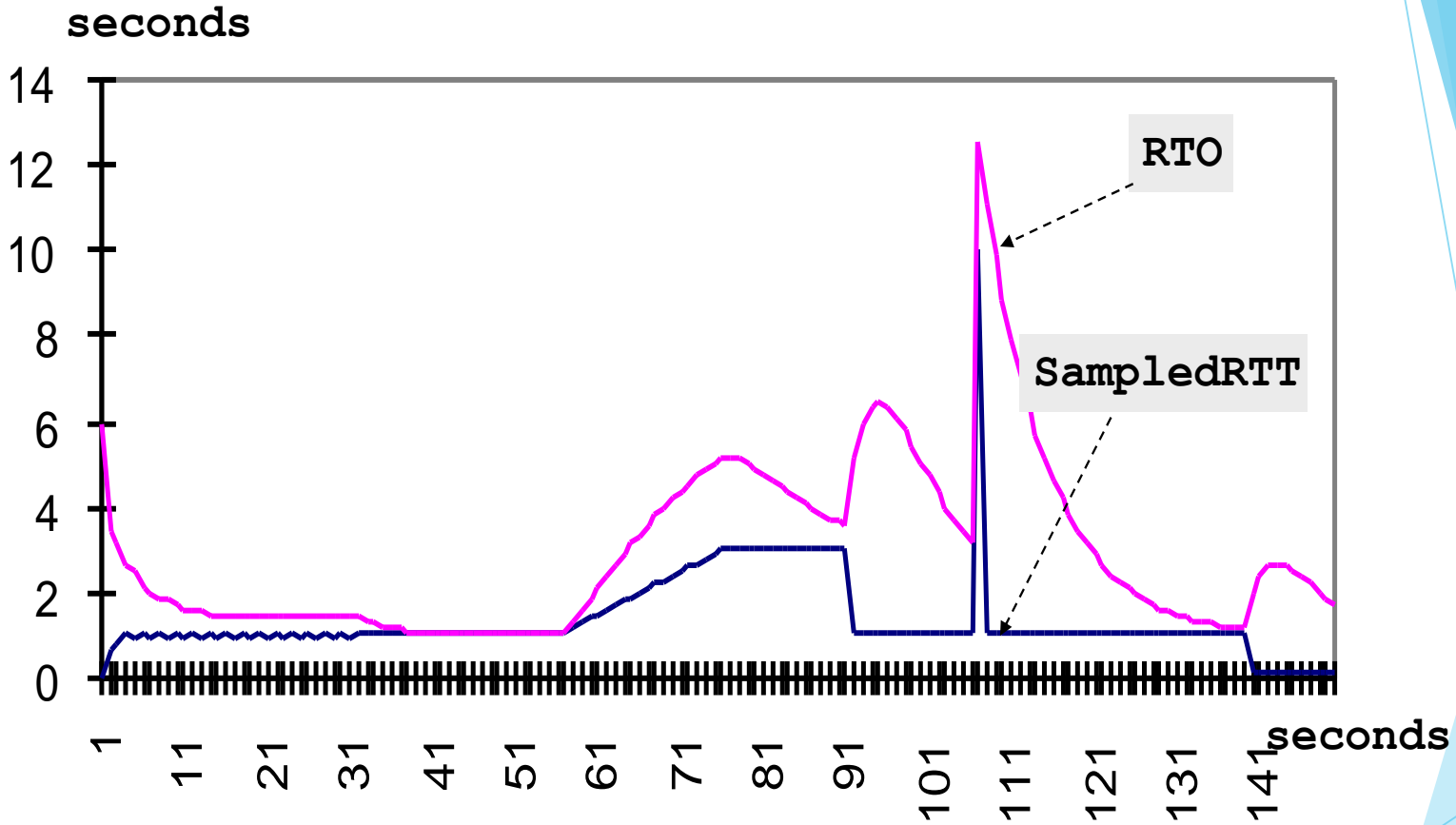
Err = **sampleRTT** - **estimatedRTT**

estimatedRTT = **estimatedRTT** + 0.125 * **Err**

deviation = **deviation** + 0.250 * (|**Err**| - **deviation**)

RTO = **estimatedRTT** + 4***deviation**

Sample RTO



Conclusions

- ▶ TCP provides a reliable service to the application programmer.
- ▶ TCP is complex and is complex to use, but is powerful. It works well with various applications such as short interactive messages or large bulk transfer.
- ▶ TCP is even more complex than we have seen as it also implements congestion control, a topic that we will study in a follow-up lecture.

Solutions

The Philosophy of Errors in a Layered Model

- ▶ The physical layer is not completely error-free - there is always some bit error rate (BER).

Information theory tells us that for every channel there is a *capacity* C such that

- ▶ At any rate $R < C$, arbitrarily small BER can be achieved
 - ▶ At rates $R \geq C$, any BER such that $H_2(\text{BER}) > 1 - C/R$ is achievable
- ▶ The TCP/IP architecture *decided*
 - ▶ Every layer $i, 2$ offers an error free service to the upper layer:

SDUs are either delivered without error or discarded

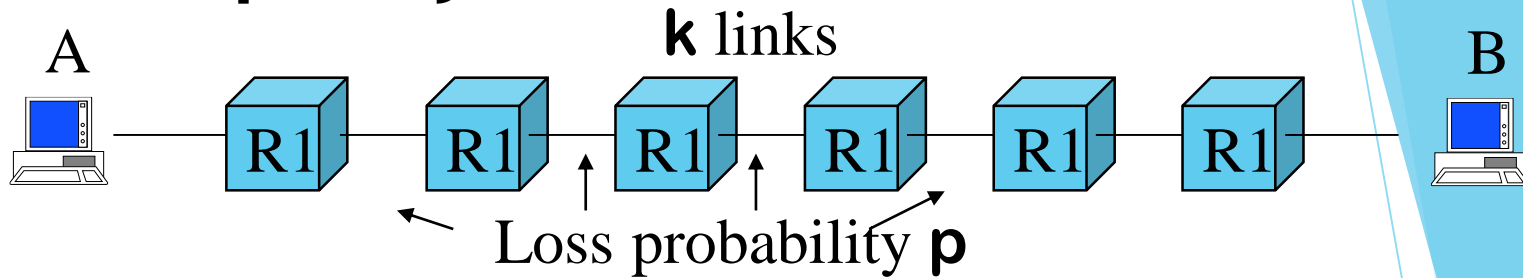
- ▶ Example: MAC layer
 - ▶ Q1. How does an Ethernet adapter know whether a received Ethernet frames has some bit errors? What does it do with the frame?
A1. It checks the CRC. If there is an error, the frame is discarded
 - ▶ WiFi detects errors with CRC and does *retransmissions* if needed
Q2. Why does not Ethernet do the same?
A2. BER is very small on cabled systems, not on wireless

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
 - A. changes in routes may cause some packets to be lost by TTL exhaustion during the transients

[back](#)

The Capacity of the End-to-End Path



► Q. compute the capacity with end-to-end and with hop by hop error recovery

A.

► Case 1: end-to-end error recovery

End to end Packet Error Rate = $1 - (1 - p)^k$

Capacity $C_1 = R \times (1-p)^k$

► Case 2: hop-by-hop error recovery

Capacity one hop = $R \times (1-p)$

End-to-end capacity $C_2 = R \times (1-p)$

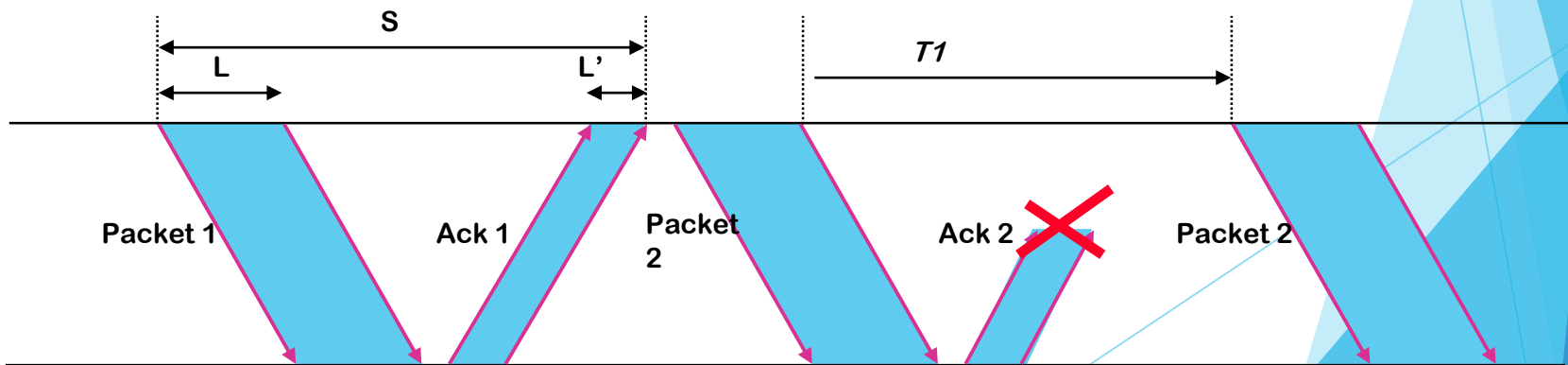
End-to-end Error Recovery is Inefficient when Packet Error Rate is high

k	Packet loss rate	C_1 (end-to-end)	C_2 (hop-by-hop)
10	0.05	$0.6 \text{ } \epsilon \text{ } R$	$0.95 \text{ } \epsilon \text{ } R$
10	0.0001	$0.9990 \text{ } \epsilon \text{ } R$	$0.9999 \text{ } \epsilon \text{ } R$

- ▶ The table shows the capacity of an end-to-end path as a function of the packet loss rate p
- ▶ Conclusion: end-to-end error recovery is not acceptable when packet loss rate is high
- ▶ Q. How can one reconcile the conflicting arguments for and against hop-by-hop error recovery ?
A.
 1. Do hop-by-hop error recovery only on links that have high bit error rate: ex on WiFi, not on Ethernet.
 2. Do hop-by-hop error recovery at the MAC layer (in the adapter), not in the router
 3. In addition, do end-to-end error recovery in hosts

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 ?
A. the stop and go protocol.
 - ▶ Packets are numbered at source
 - ▶ Destination sends one acknowledgement for every packet received
 - ▶ Source waits for ack; if after T_1 seconds the ack did not arrive, packet is retransmitted



Why Sliding Window ?

Why invented ?

- ▶ Overcome limitations of Stop and Go

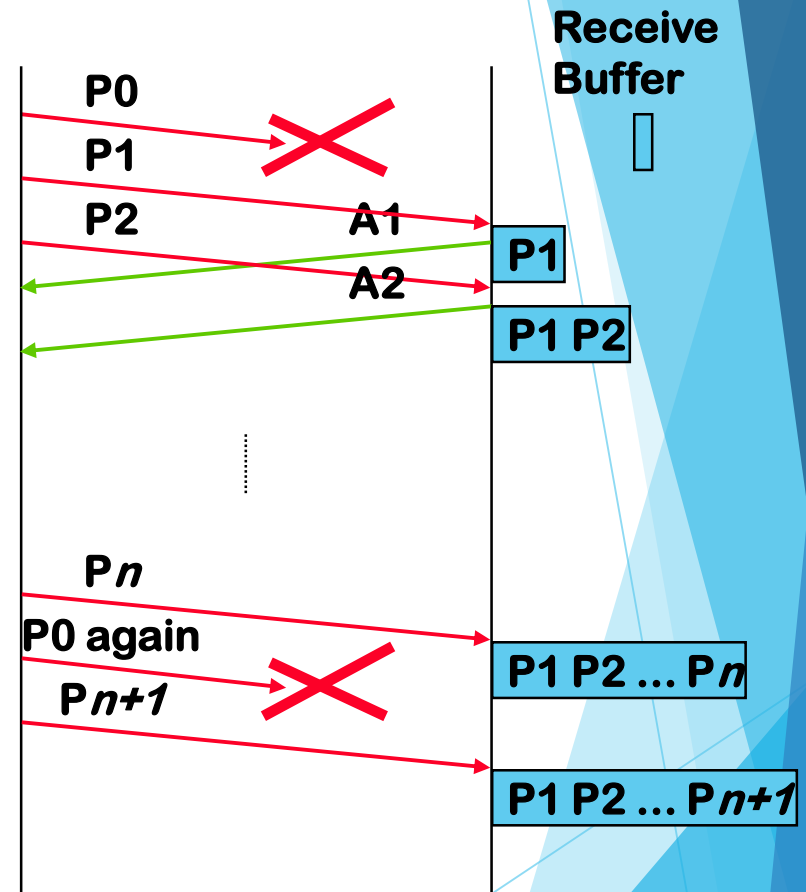
Q. what is the limitation of Stop and Go ?

A. when the bandwidth-delay product is not very small, the throughput is small. The protocol wastes time while waiting for acks.

[back](#)

What does it do ?

1. Allow mutple 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.



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
4. acknowledged
5. Loss detection is by timeout at sender when no acknowledgement has arrived
6. When a loss is detected, only the packet that is detected as lost is re-transmitted (this is
7. called *Selective Repeat*).

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

8. A. Yes, if an ack is lost.

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?

A. Yes, for several reasons

1. If an ack is lost
2. If packet n is lost and packet $n+1$ is not

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 ?

A. If the timer T_1 cannot be set very accurately, the previous protocol may wait for a long time before detecting a loss. This protocol reacts more rapidly.

[back](#)

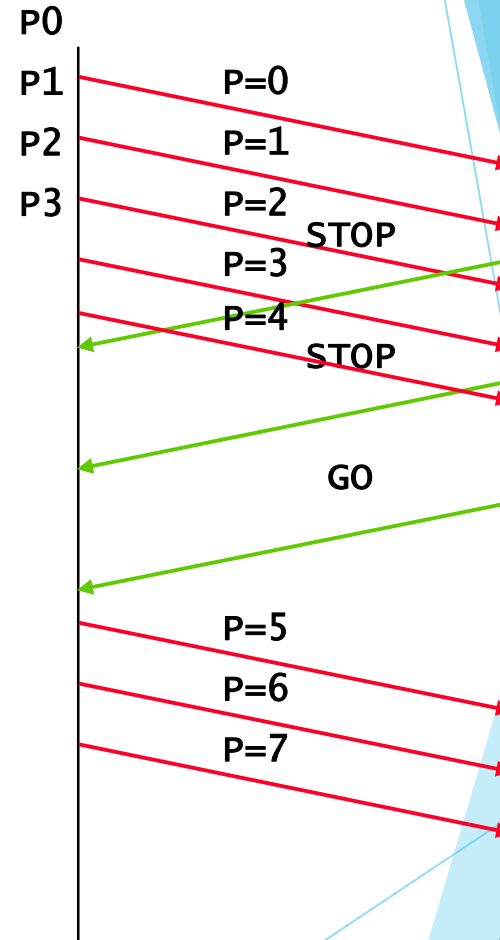
Are There Alternatives to ARQ ?

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 ?
 - A. Pro: does not require retransmission. On network with very large delay, this is a benefit.
 - Pro: works better for multicast, since different destinations may have lost different packets.
 - Con: less throughput: redundancy is used even if not needed, ARQ transmits fewer packets
 - ▶ [back](#)
- ▶ 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)

Backpressure Flow Control

- ▶ Destination sends STOP (= PAUSE) or GO messages
- ▶ Destination stops sending for x msec after receiving a STOP message
- ▶ Simple to implement
- ▶ Q. When does it work well ?
A. If bandwidth delay product is small
[back](#)



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 ?

A. Yes, since with a sliding window of size W , the number of packets sent but unacknowledged is $\leq W$. However, this poses a problem at the source: non acknowledged packets may be retransmitted, whereas they were correctly received.

[back](#)

Why both TCP and UDP ?

- ▶ Most applications use TCP rather than UDP, as this avoids re-inventing error recovery in every application
- ▶ But some applications do not need error recovery in the way TCP does it (i.e. by packet retransmission)
 - ▶ For example: Voice applications
 - Q. why ?
 - A. delay is important for voice. Packet retransmission introduces too much delay in most cases.
 - [back](#)
- ▶ For example: an application that sends just one message, like name resolution (DNS). TCP sends several packets of overhead before one single useful data message. Such an application is better served by a Stop and Go protocol at the application layer.

Multicast Routing

- ▶ Multicast Routing is one of the routing protocols in TCP/IP communication. In computer networking, there are several multicast group communication protocols where data transmission is addressed to a group of destination computers simultaneously. (Multicast Source Discovery Protocol, Multicast BGP, Protocol Independent Multicast)

Multicast

- ▶ In computer networking, multicast is group communication where data transmission is addressed to a group of destination computers simultaneously. Multicast can be one-to-many or many-to-many distribution. Multicast should not be confused with physical layer point-to-multipoint communication.

Multicast Routing Protocol

- ▶ A Multicast Routing Protocol is used to communicate between multicast routers and enables them to calculate the multicast distribution tree of receiving hosts. Protocol Independent Multicast (PIM) is the most important Multicast Routing Protocol.
- ▶ A multicast routing protocol is a mechanism for constructing a loop-free shortest path from a source host that sends data to the multiple destinations that receives the data.

Dynamic Host Configuration Protocol (DHCP)

- ▶ BOOTP is not dynamic configuration protocol.
 - ▶ When a client requests its IP address, the BOOTP sever looks up a table that matches the physical address of the client with its IP address.
 - ▶ This means that the binding between the physical address and the IP address of the client should already exist.
 - ▶ **What if a host moves from one physical network to another ?**
- ▶ DHCP is extension to BOOTP and has backward compatible with BOOTP
 - ▶ meaning that a host running the BOOTP client can request a static configuration to a DHCP server

DHCP (Cont'd)

- ▶ DHCP provides temporary IP addresses for a limited period of time
- ▶ DHCP has two DBs
 - ▶ one for statically binding between physical address and IP address
 - ▶ the other one with a pool of available IP addresses
 - ▶ When a DHCP client requests a temporary IP addresses, the DHCP sever assigns an IP address from a pool for a negotiable period of time
 - ▶ When a DHCP client sends a request to a DHCP server
 - ▶ At first, checking its static database
 - ▶ If not , selecting an IP address from the available pool

DHCP (Cont'd)

- ▶ Leasing
 - ▶ The DHCP server issues a lease for a specific period of time
 - ▶ When the lease expires, the client must either stop using the IP address or renew the lease
- ▶ DHCP Operation
 1. A client broadcasts a DHCPDISCOVER message using destination port 67
 2. Servers respond with a DHCPOFFER message including an IP address
 - ◆ Offering the duration of the lease - default : one hour
 - ◆ The server that sends a DHCPOFFER locks the offered IP address so that it is not available to any other clients

DHCP (Cont'd)

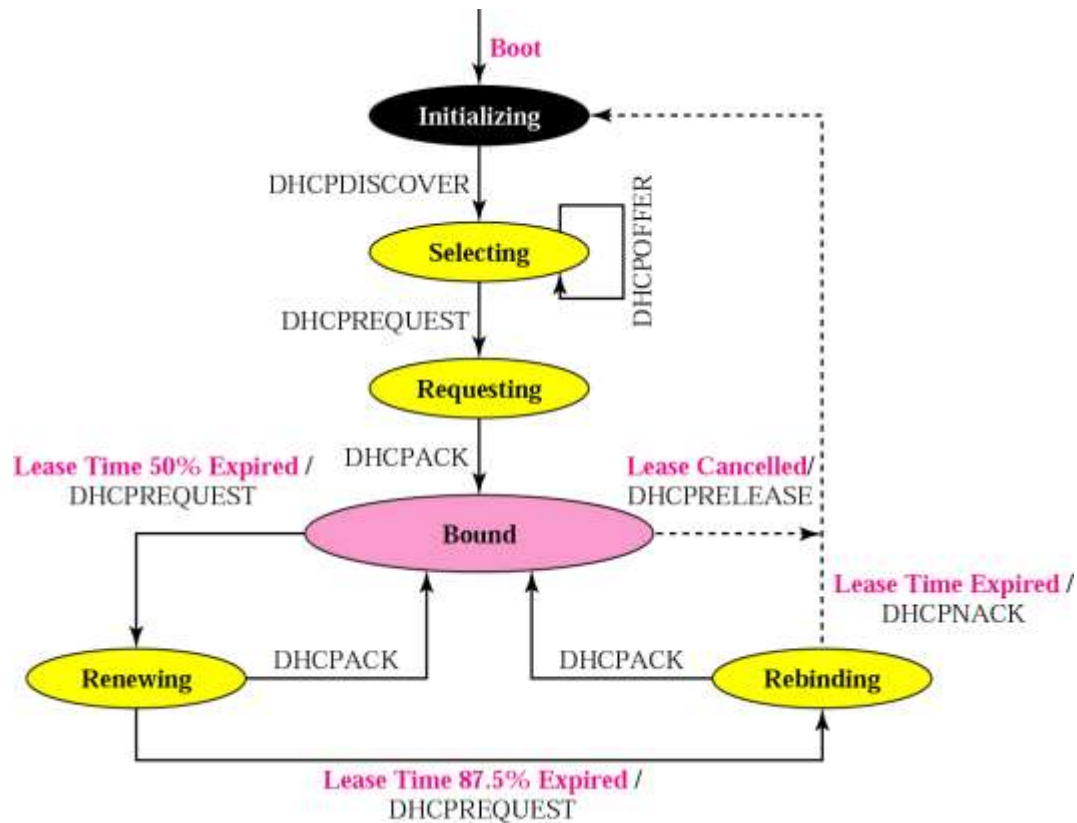
- ▶ If the client receives no DHCPOFFER message, it will try four more times, each with a SPAN of two seconds.
- ▶ If there is no reply to any of these DHCPDISCOVERs, the client sleeps for five minutes before trying again
- ▶ The client chooses one of the offers and sends a DHCPREQUEST message to the selected server
- ▶ The server responds with a DHCPACK message and creates the binding between the client physical address and its IP address
- ▶ Before 50 percent of the lease period is reached, the client sends another DHCPREQUEST and asks for renewal

DHCP (Cont'd)

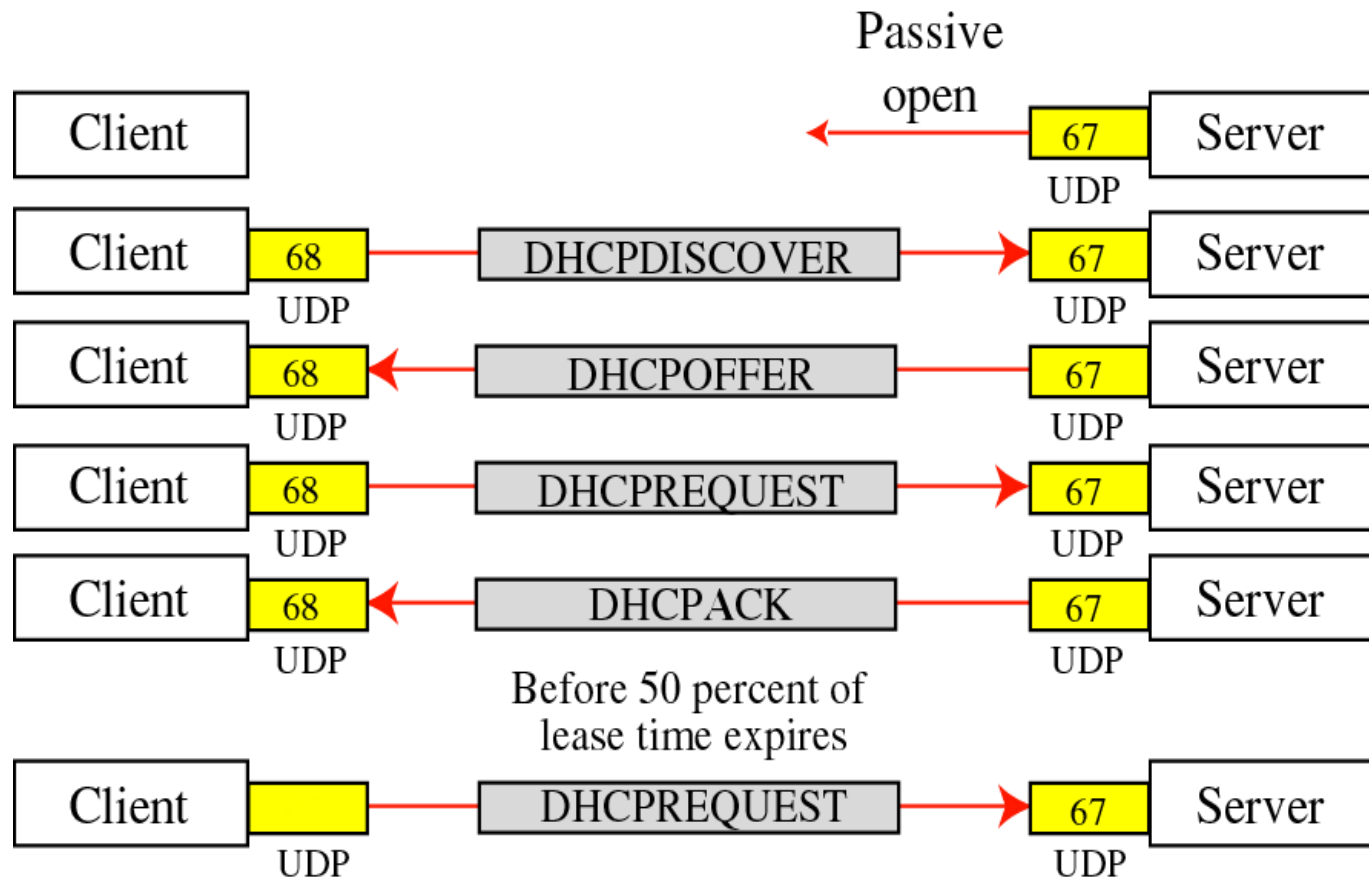
- ▶ If the server responds with a DHCPACK, the client has a new lease agreement and can reset its timer. If the server responds with a DHCPNACK, the client must immediately stop using the IP address and find another server (step 1)
- ▶ If the sever does not respond, the client sends another DHCPREQUEST when the lease time reaches 87.5 percent. If the client terminates the lease prematurely, the client sends a DHCPRELEASE message to the server.

DHCP (Cont'd)

□ DHCP Transition Diagram

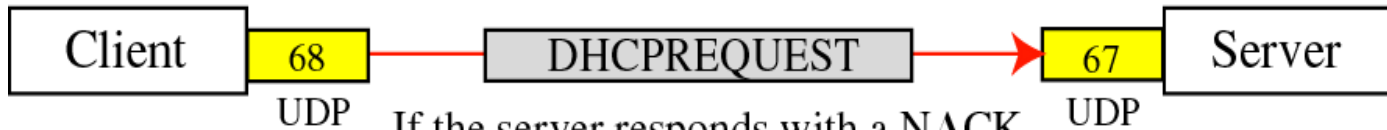


DHCP (Cont'd)

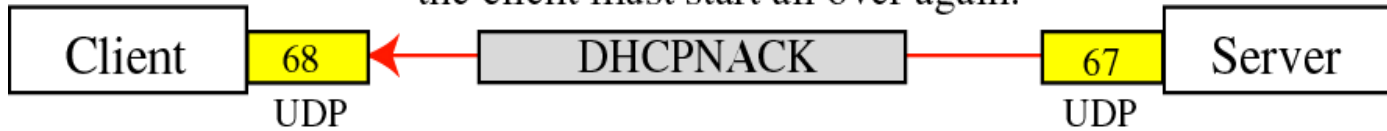


DHCP (Cont'd)

If the server does not respond,
the request is repeated.



If the server responds with a NACK,
the client must start all over again.



If the server responds with an ACK,
the client has a new lease.



•
•
•



DHCP (Cont'd)

▶ Packet Format

- ▶ To make DHCP backward compatible with BOOTP, it is only added a one-bit flag to the packet.
- ▶ extra options have been added to the option field
- ▶ Flag :
 - ▶ Let client specify a forced broadcast reply from the server
- ▶ Option :
 - ▶ several options are added
 - ▶ Ex) the value 53 for the tag subfield is used to define the type of interaction between the client and server
 - ▶ MAX : 312 bytes

DHCP (Cont'd)

Operation code	Hardware type	Hardware length	Hop count
Transaction ID			
Number of seconds	F	Unused	
Client IP address			
Your IP address			
Server IP address			
Gateway IP address			
Client hardware address (16 bytes)			
Server name (64 bytes)			
Boot file name (128 bytes)			
Options (Variable length)			

DHCP (Cont'd)

▶ DHCP Options

<i>Value</i>	<i>Value</i>
1 DHCPDISCOVER	5 DHCPACK
2 DHCPOFFER	6 DHCPNACK
3 DHCPREQUEST	7 DHCPRELEASE
4 DHCPDECLINE	

The background features a white space with blue geometric shapes on the right side, including overlapping triangles and a vertical bar, creating a modern, abstract design.

Domain Name System (DNS)

Domain Name System (DNS)

Need System to map name to an IP address and vice versa

We have used a host file in our Linux laboratory.

Not feasible for the entire Internet.

Thus, divide huge amount of info and store in parts on many different computers.

Host needing info contacts the closest server containing the needed info.

This is DNS.

Hierarchical Name Space is used. Names are made up of several parts:

acme.gatech.edu

Domain Name Space: names are defined in an inverted tree structure. Read

names from node

up to root of tree.

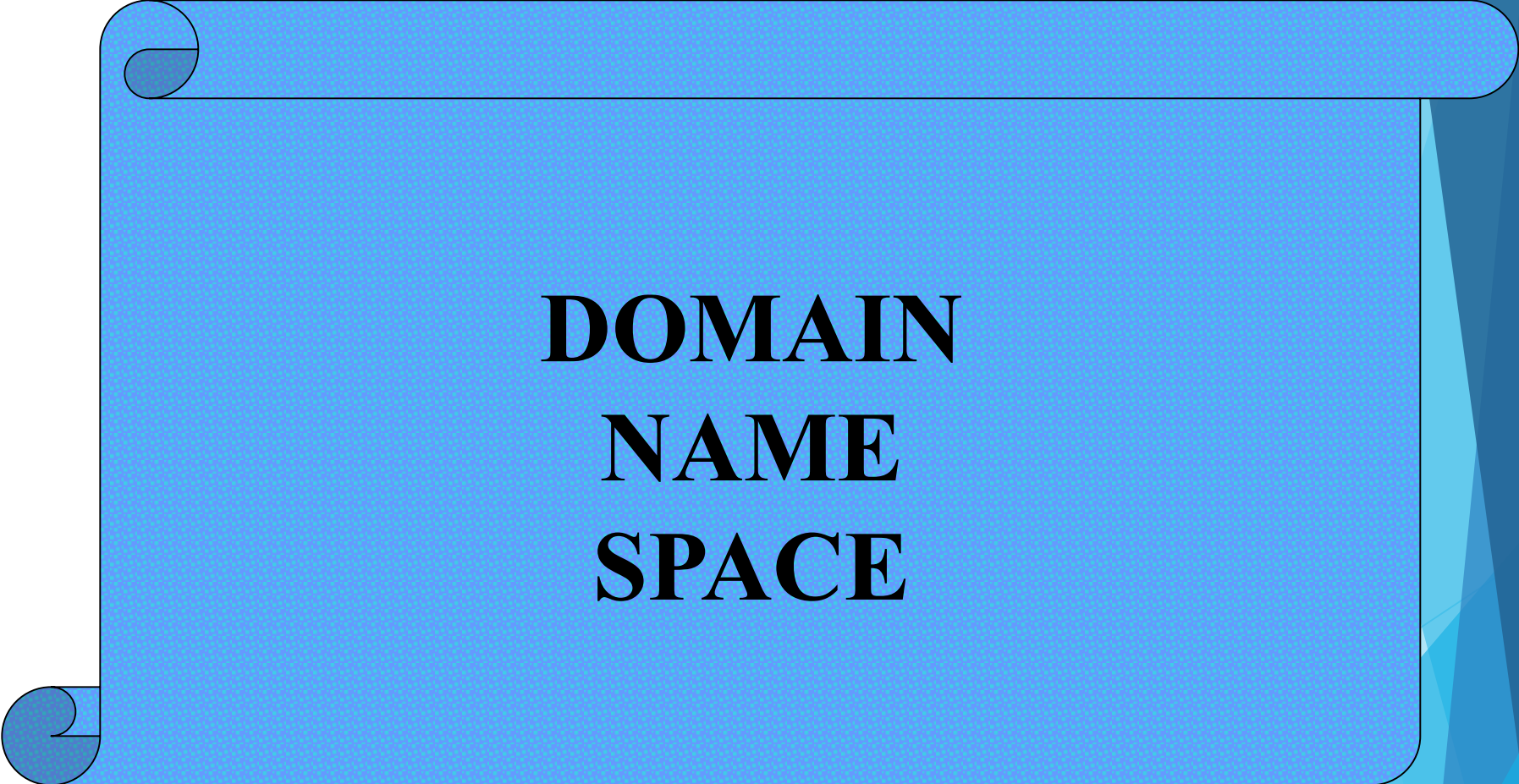
Source: TCP/IP Protocol Suite by Forouzan

CONTENTS

- **NAME SPACE**
- **DOMAIN NAME SPACE**
- **DISTRIBUTION OF NAME SPACE**
- **DNS IN THE INTERNET**
- **RESOLUTION**
- **DNS MESSAGES**
- **TYPES OF RECORDS**
- **COMPRESSION**
- **EXAMPLES**
- **DDNS**
- **ENCAPSULATION**

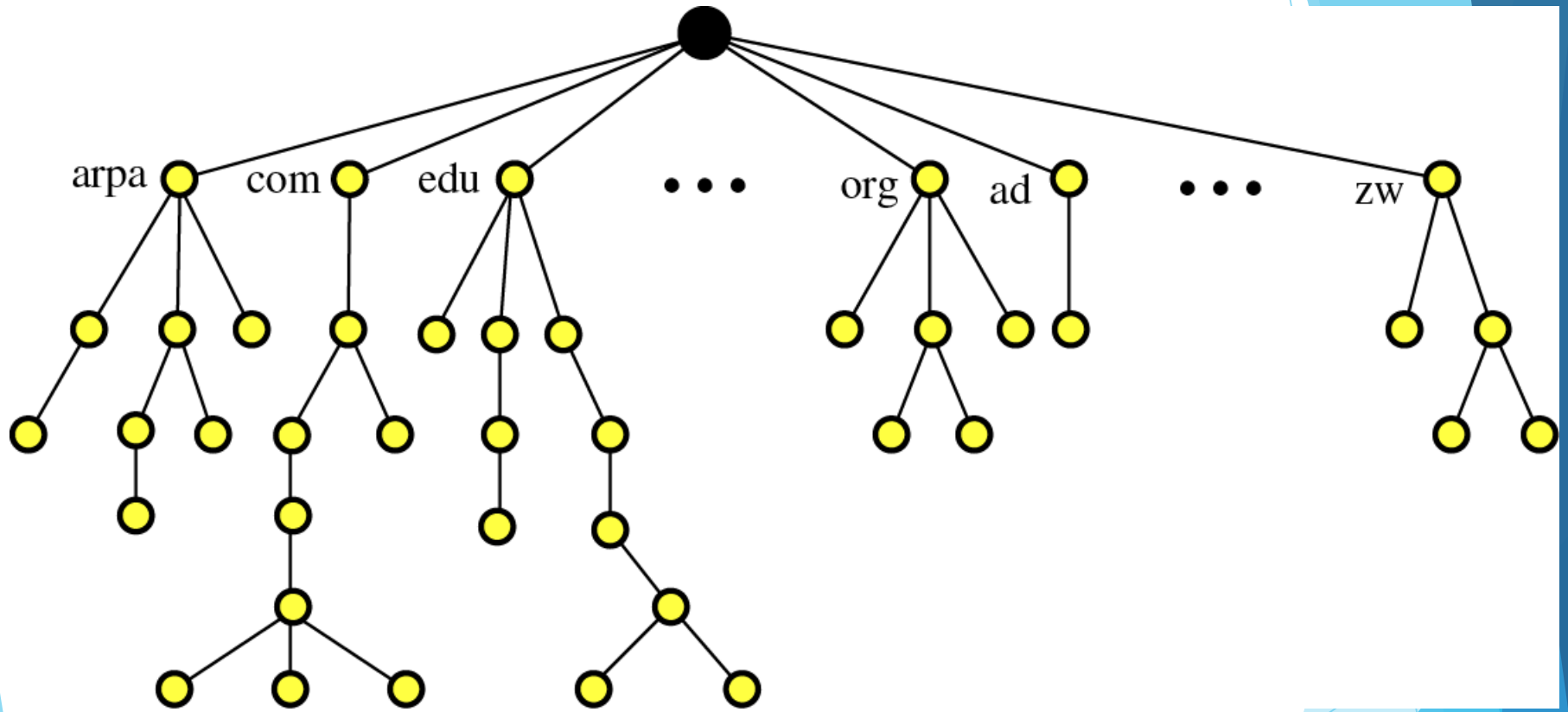
A blue scroll-like graphic with a black outline and rounded corners. The top and bottom edges are slightly curved, and there are small circular details on the left side that suggest a scroll. The text "NAME SPACE" is centered in a bold, black, serif font.

NAME SPACE

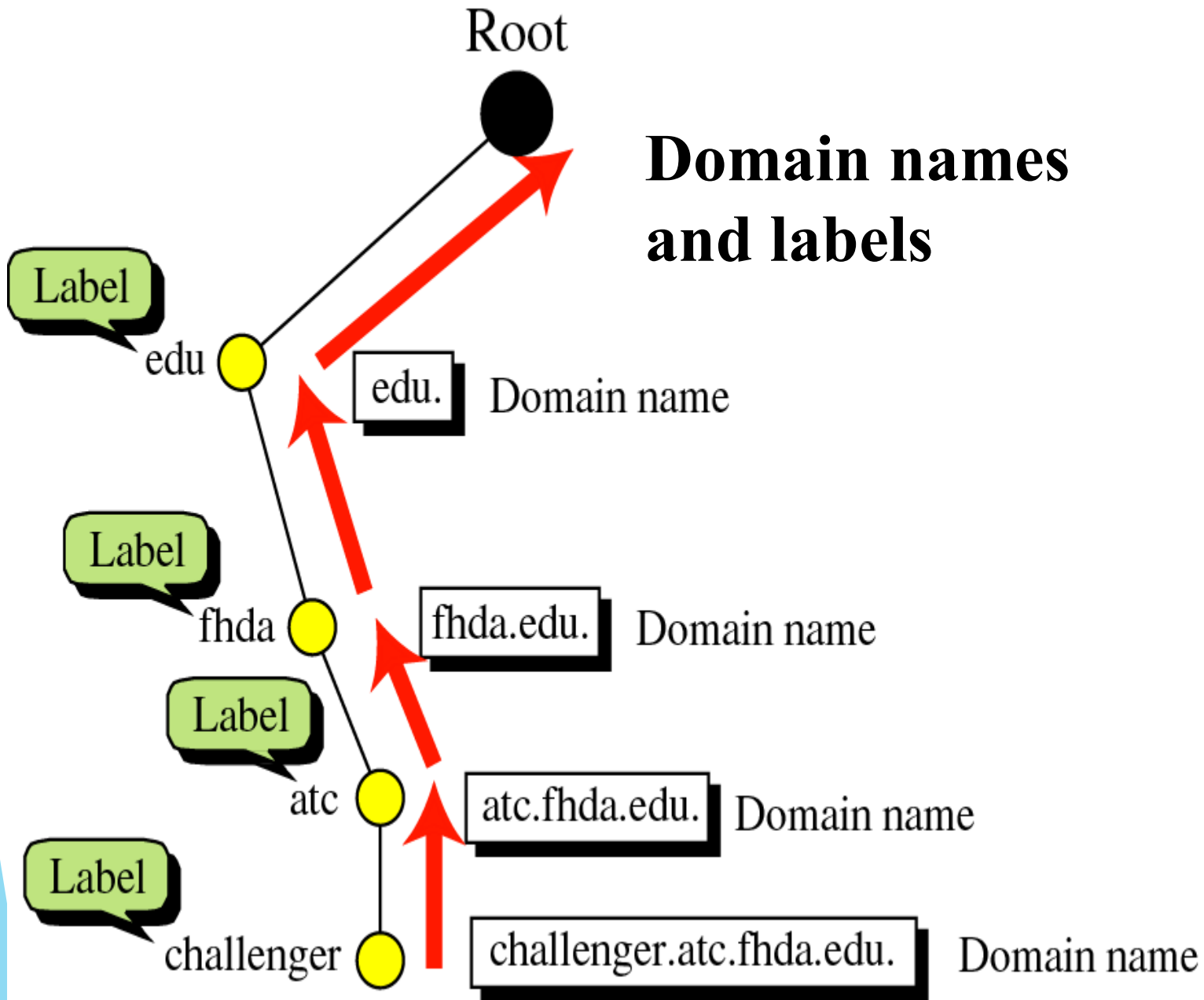
A blue scroll graphic with a white border and two rolled-up corners. The text is centered on the scroll.

**DOMAIN
NAME
SPACE**

Domain name space



Domain names and labels



FQDN and PQDN

FQDN

challenger.atc.fhda.edu.

cs.hmme.com.

www.funny.int.

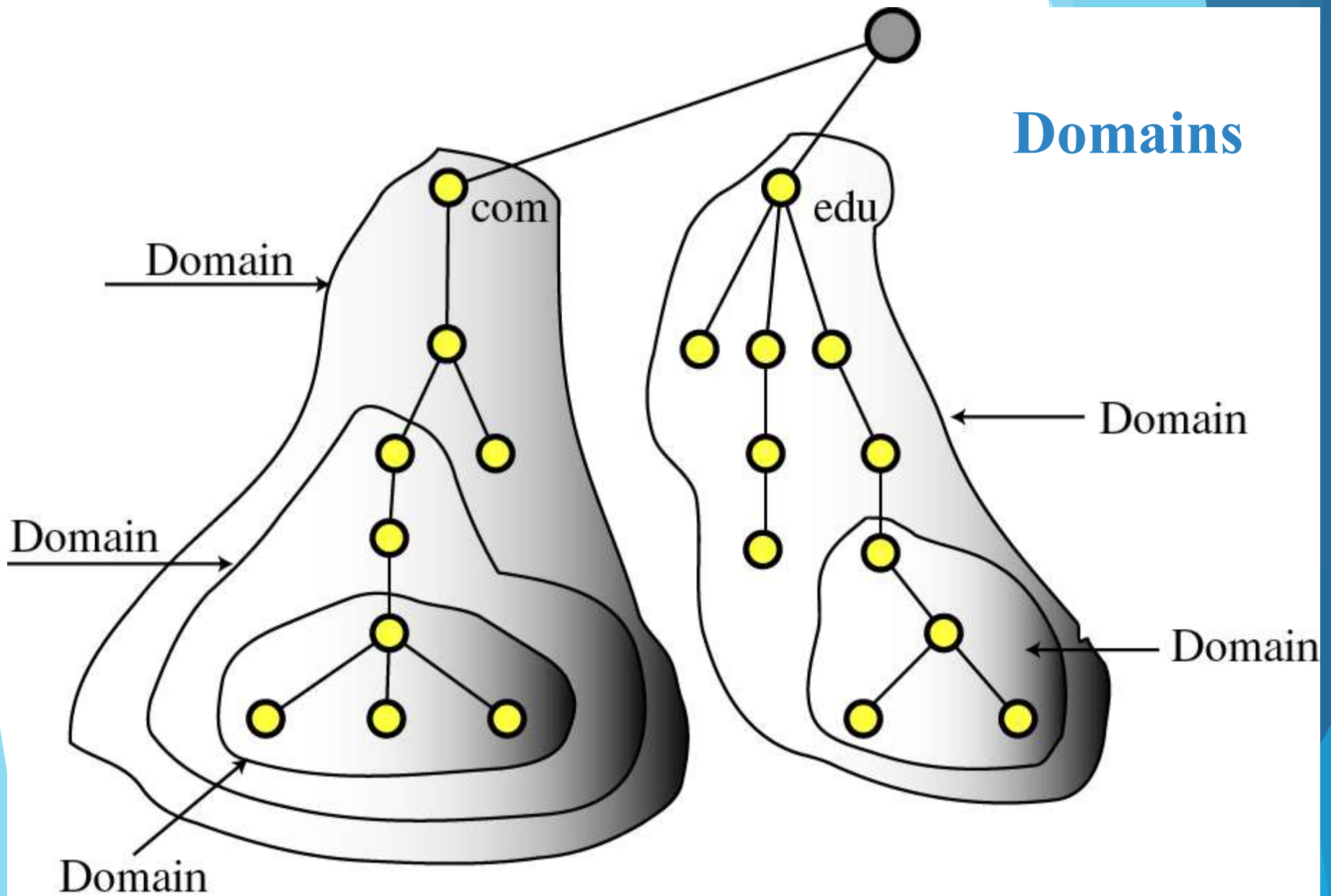
PQDN

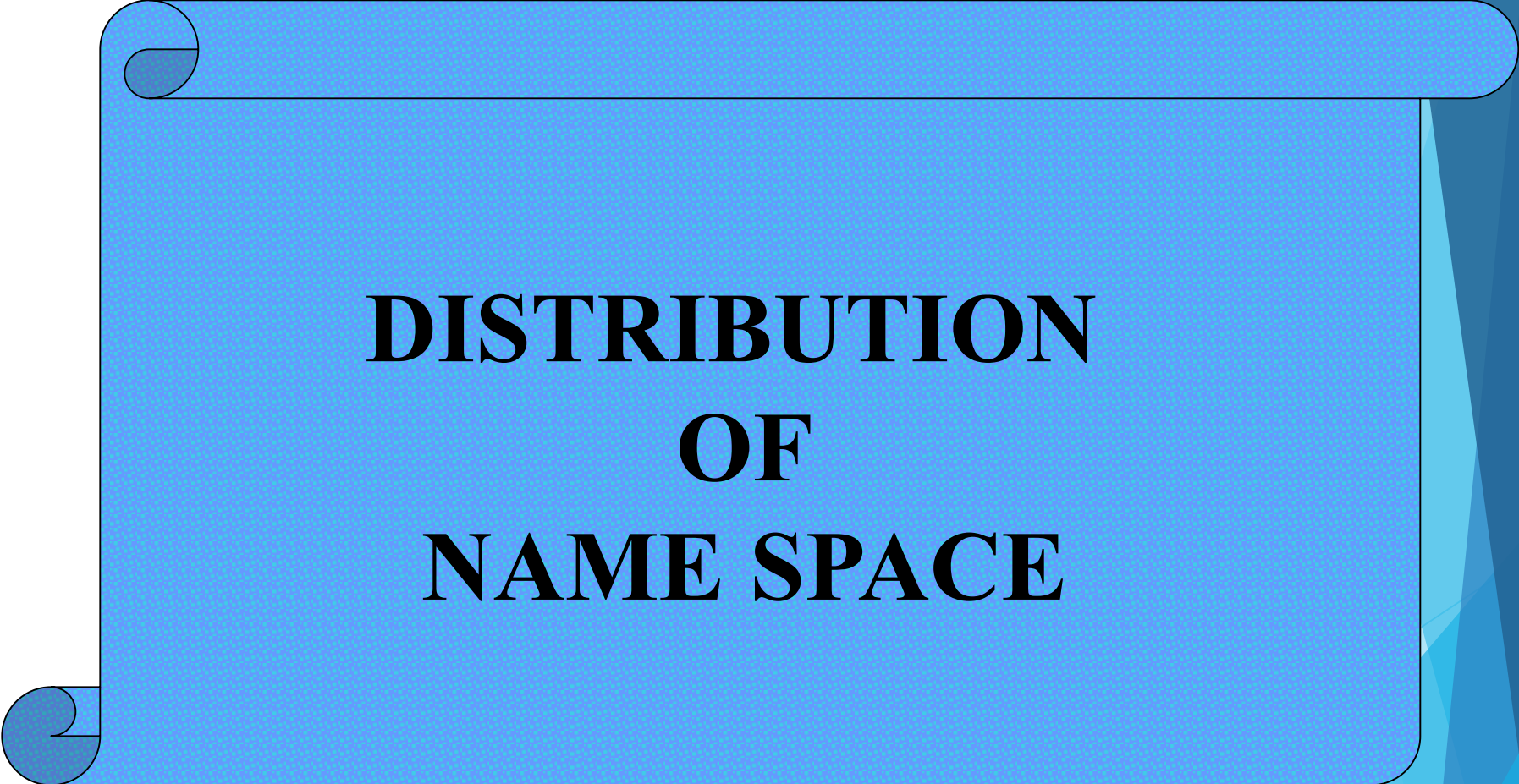
challenger.atc.fhda.edu

cs.hmme

www

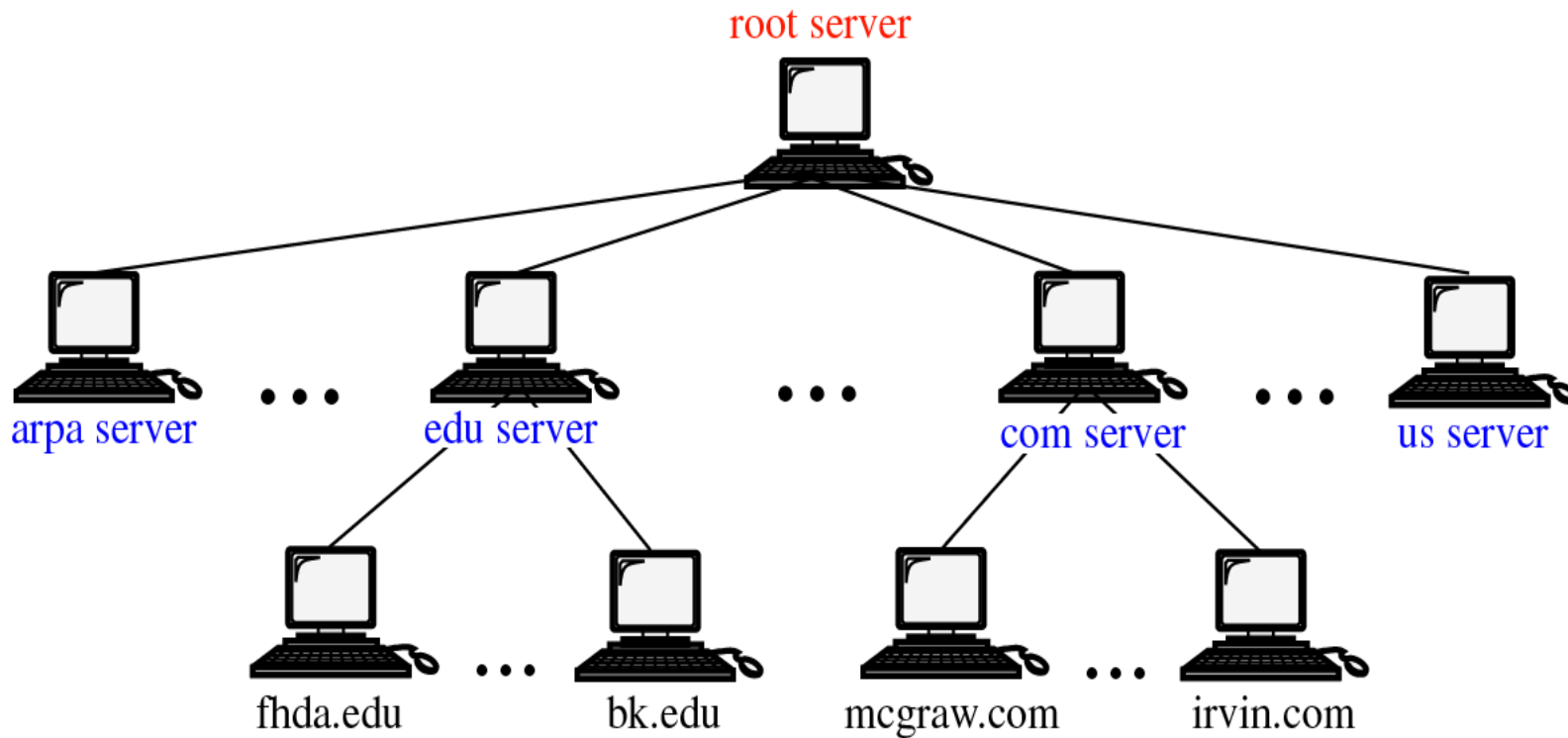
Domains



A blue scroll graphic with a black outline and two circular tabs on the left side. The text is centered on the scroll.

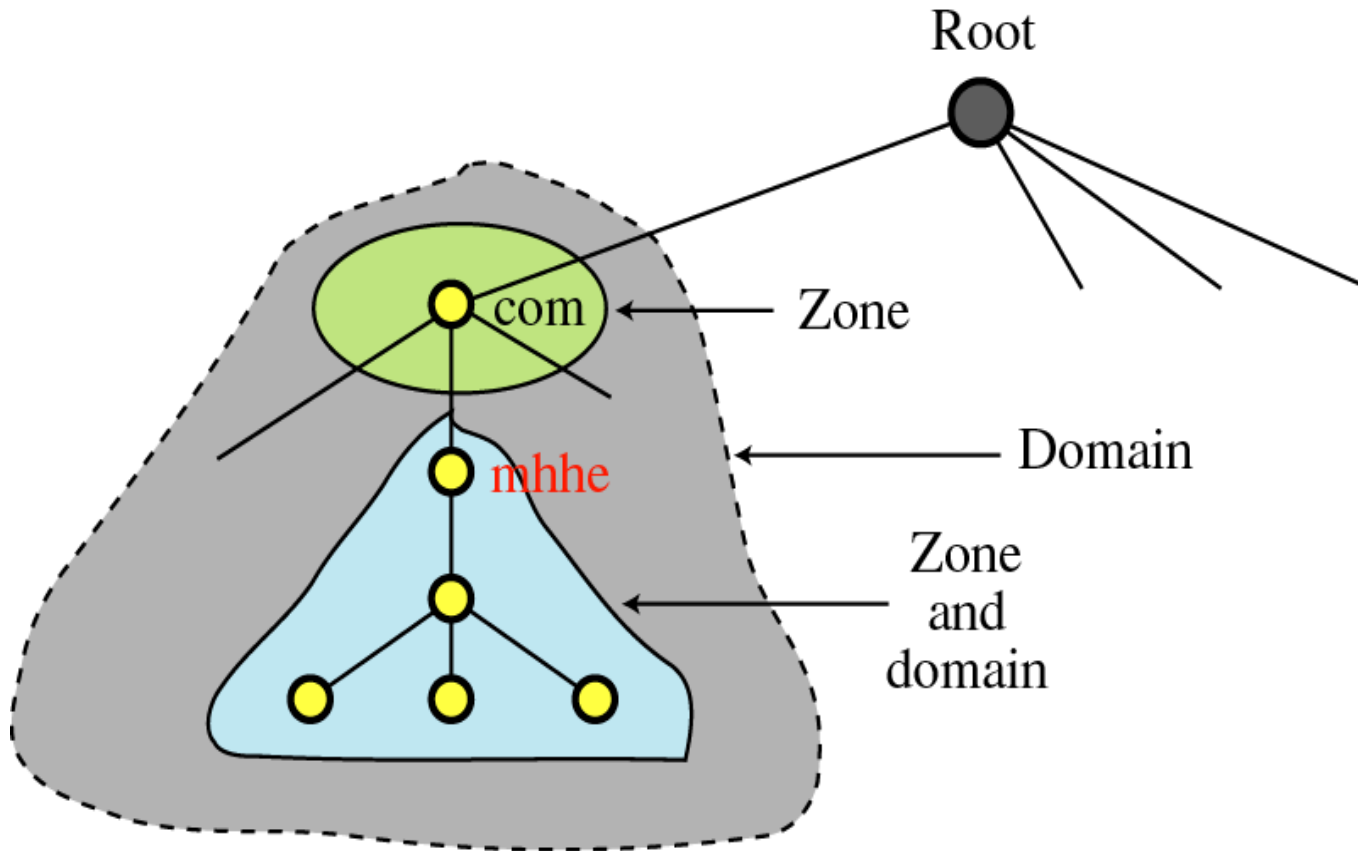
**DISTRIBUTION
OF
NAME SPACE**

DNS servers are used to distribute the info among many servers. We use a hierarchy of servers just like the hierarchy of names.



What a server has authority for is called a zone. A root server's zone is the whole tree.

We use primary and redundant servers.



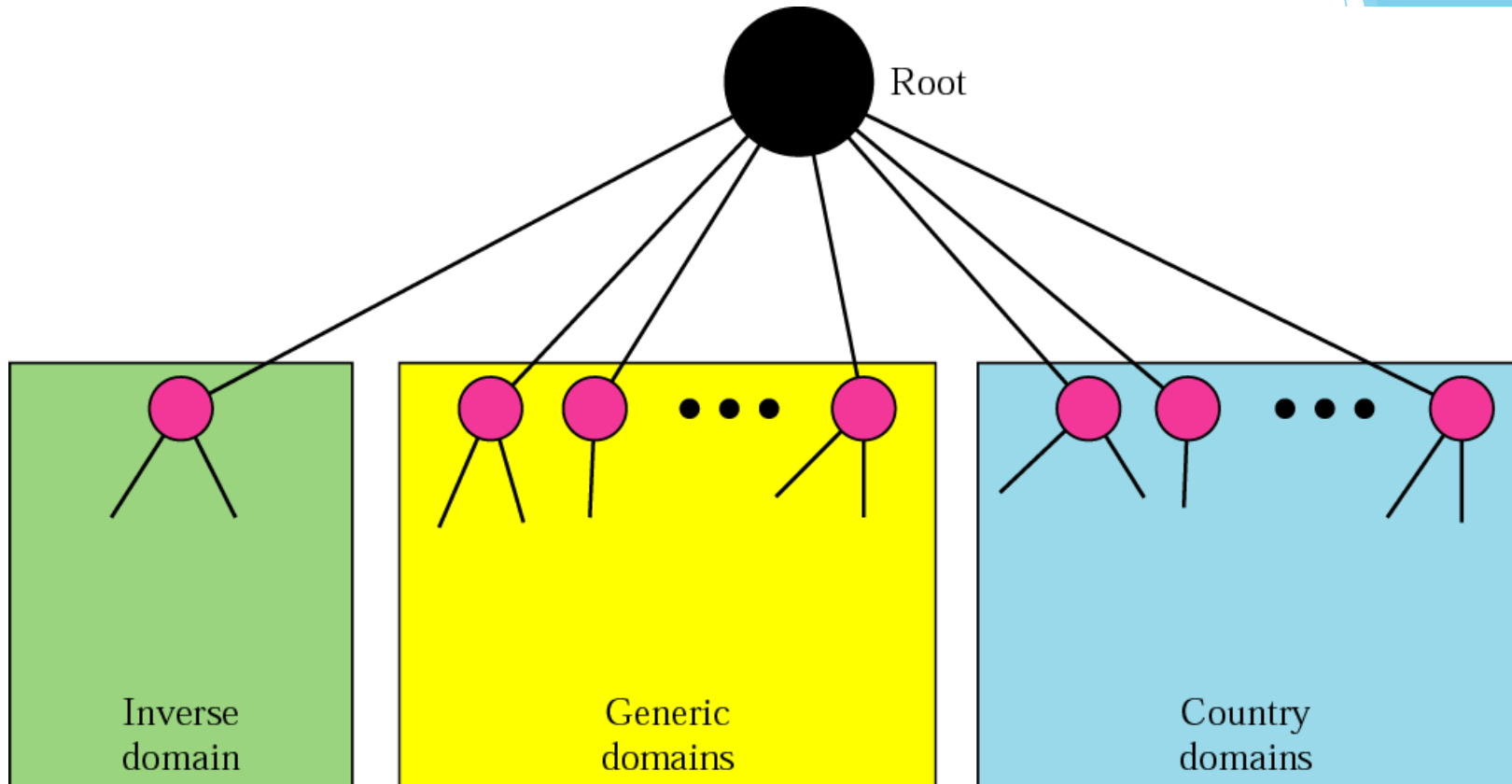
Note

A primary server loads all information from the disk file; the secondary server loads all information from the the primary server. When the primary downloads information from the secondary, it is called zone transfer.

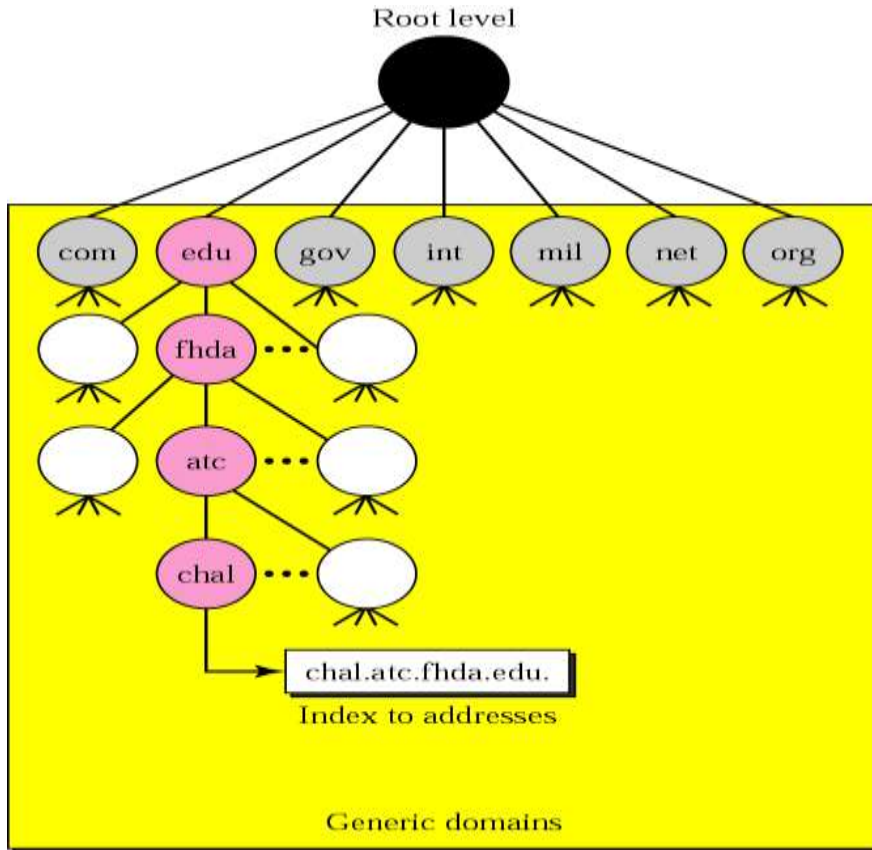
A blue scroll graphic with a black outline and two circular tabs on the left side, one at the top and one at the bottom. The scroll is set against a background of overlapping blue and white geometric shapes.

**DNS
IN THE
INTERNET**

DNS in the Internet

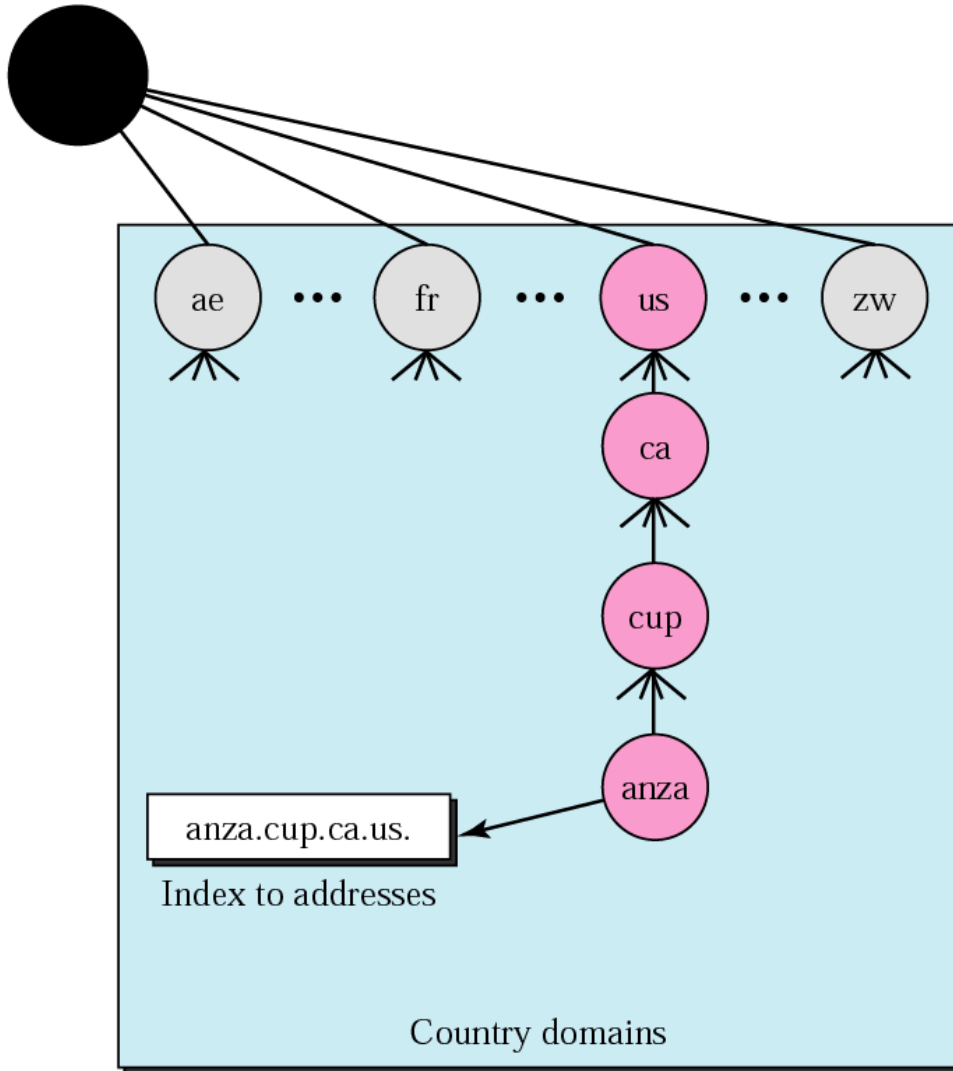


Generic domains

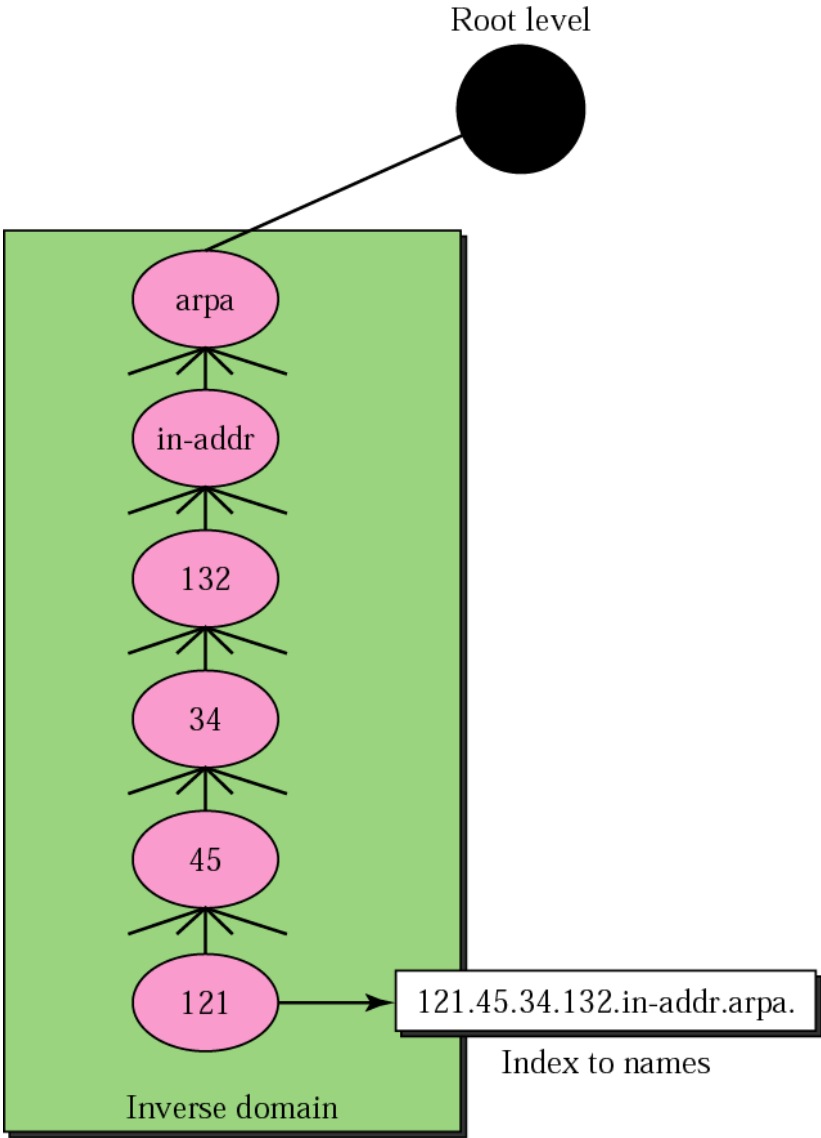


Country domains

Root level



Inverse domain



A blue scroll graphic with a white background. The scroll is unrolled, showing the word "RESOLUTION" in the center. The scroll has a dark blue border and a lighter blue fill. The word "RESOLUTION" is written in a bold, black, serif font. The scroll is set against a background of various shades of blue geometric shapes.

RESOLUTION

Resolution

DNS uses a client server architecture. A host needing info contacts a client named a resolver.

The resolver client contacts a DNS server.

Recursive Resolution:

The resolver asks for a recursive answer from a DNS server.

The server must respond with the complete answer.

If it does not know the answer the server itself asks a parent server in the hierarchy.

If the parent does not know, the parent asks a higher level server in the hierarchy.

Eventually the resolver will be told the answer by the first DNS server the resolver contacted.

Iterative Resolution:

If client does not specify a recursive answer, client will get an iterative answer.

This means if the first server contacted does not know the answer, the server returns

the IP address of what the server thinks is a smarter server.

This continues until the answer is found.

Protocol that transports DNS messages

DNS uses either TCP or UDP. Always port 53. UDP is used when messages are less than 512 bytes because many UDP implementations have a 512 byte maximum size limit.

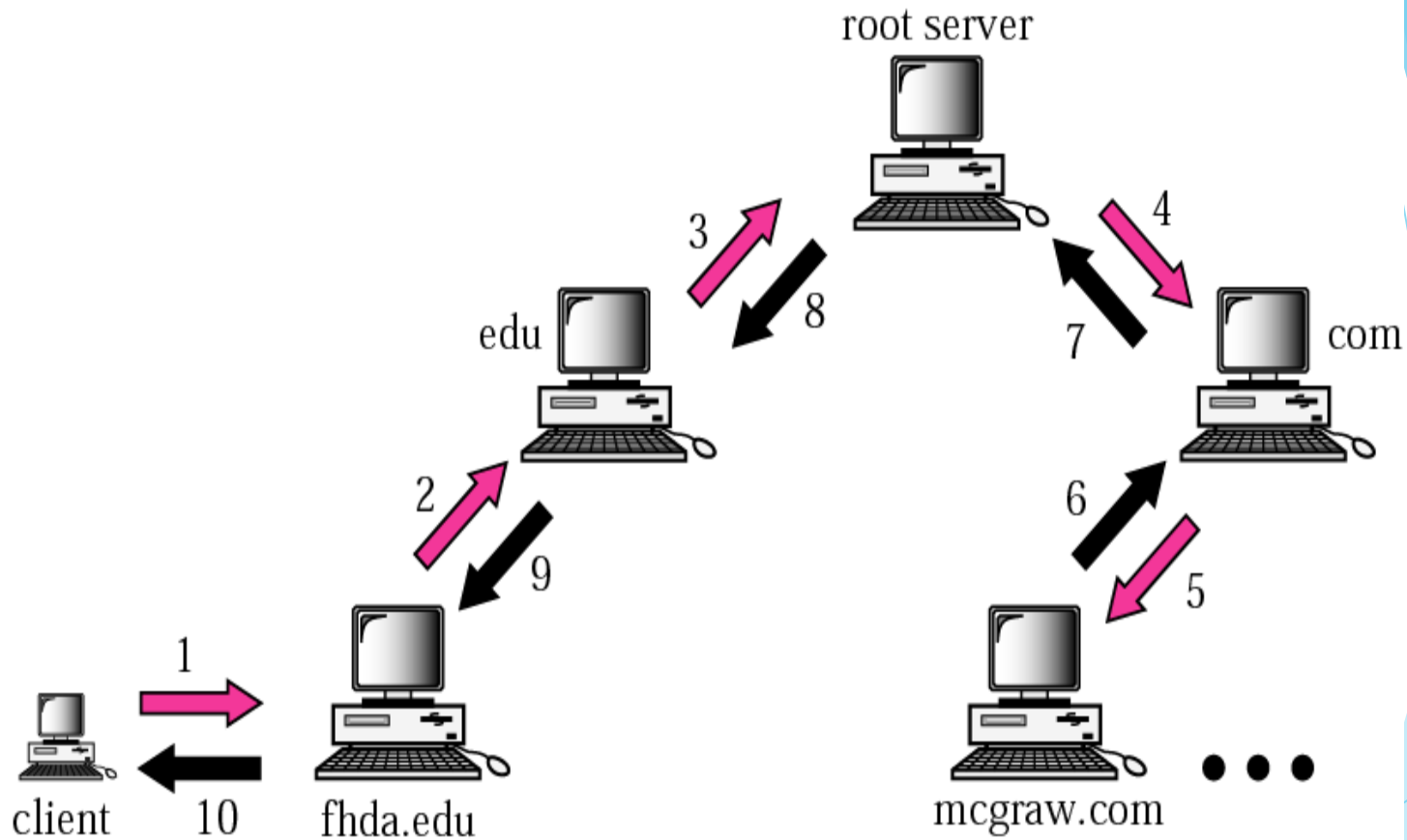
If message larger than 512 bytes:

If client knows message is larger than 512 it will use a TCP connection

If client does not know size of message opens a UDP port to server, but if the response is larger than 512, server truncates response and sets

the TC bit as a sign to the client to try again using a TCP connection instead.

Recursive resolution



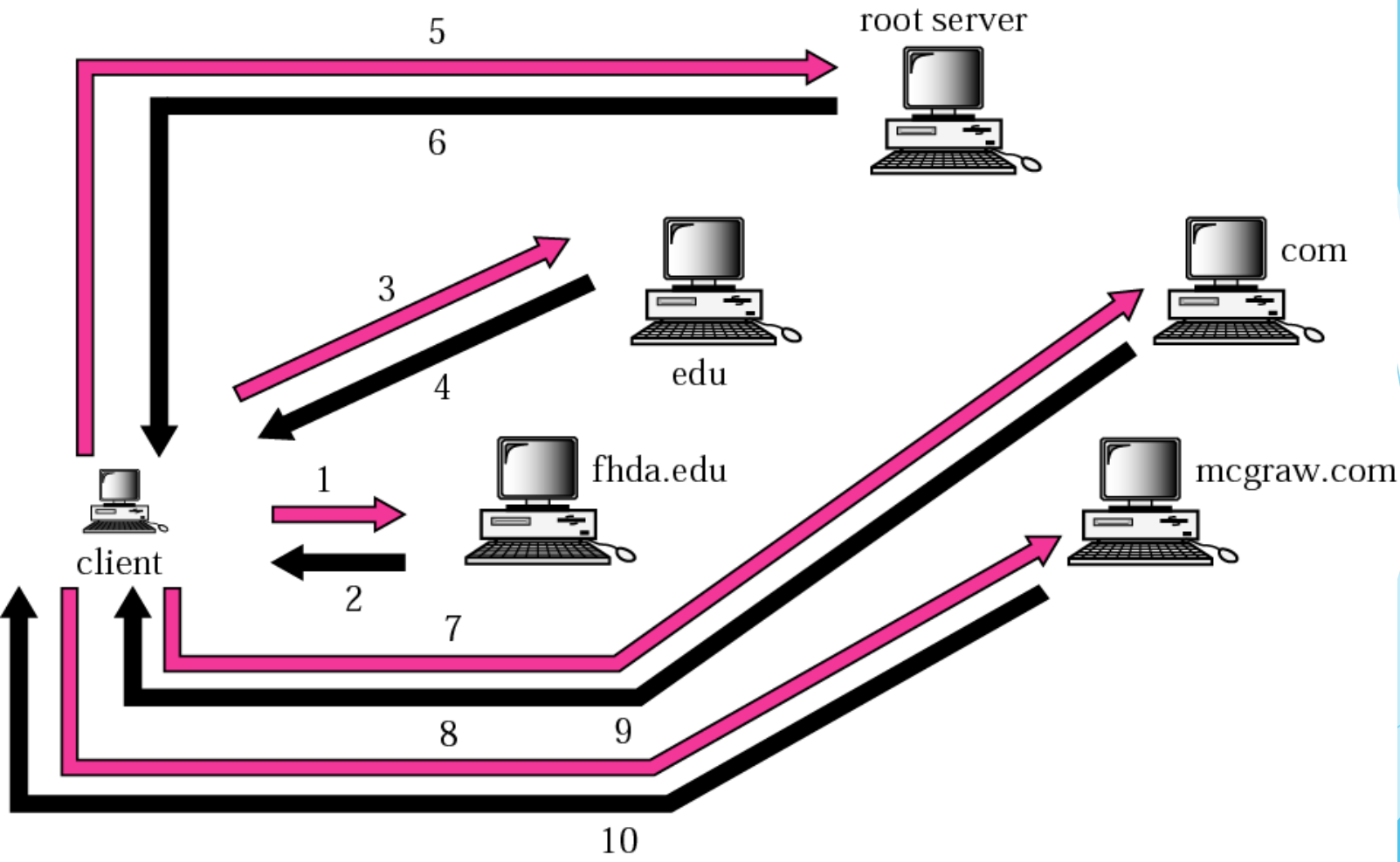
Here is a typical list of root servers held by a typical name server:

```
; This file holds the information on root name servers
; needed to initialize cache of Internet domain name
; servers (e.g. reference this file in the
; "cache . <file>" configuration file of BIND domain
; name servers).
;
; This file is made available by InterNIC registration
; services under anonymous FTP as
; file      /domain/named.root
; on server  FTP.RS.INTERNIC.NET
; last update:  Aug 22, 1997
; related version of root zone:  1997082200
;
;
; formerly NS.INTERNIC.NET
;
.           3600000 IN NS A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET. 3600000 A 198.41.0.4
;
; formerly NS1.ISLEDU
;
.           3600000 NS B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET. 3600000 A 128.9.0.107
;
; formerly C.PSINET
;
.           3600000 NS C.ROOT-SERVERS.NET.

ETC.....
```

Source:<http://computer.howstuffworks.com/dns5.htm>

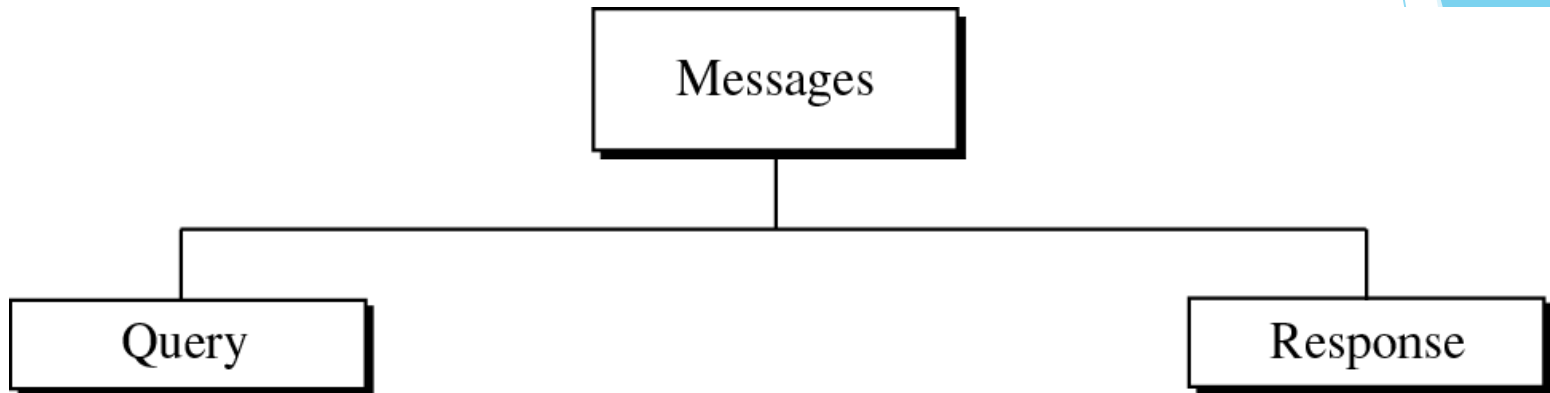
Iterative resolution



A blue scroll graphic with a black outline and two unrolled corners on the left side. The text is centered on the scroll.

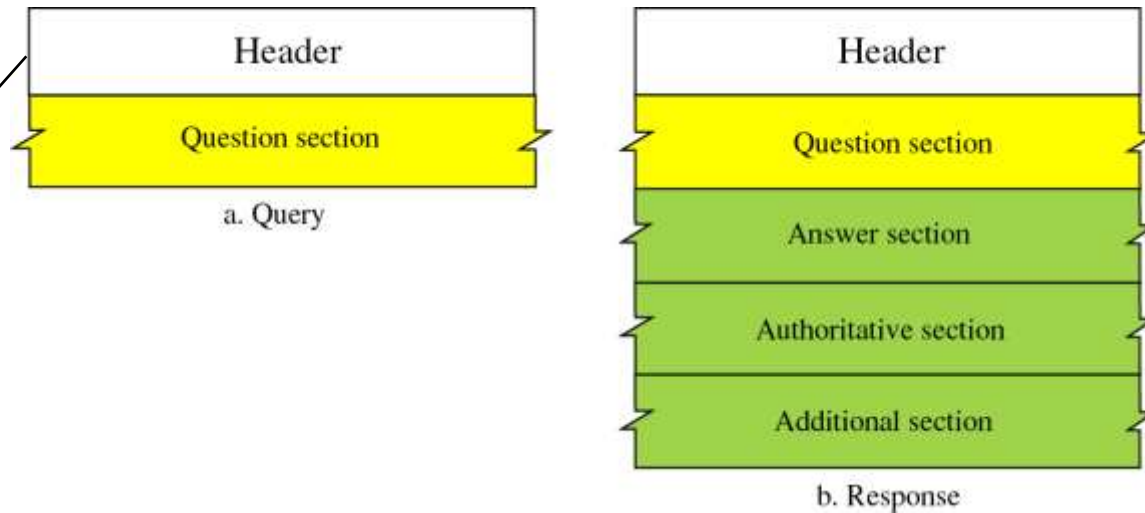
DNS MESSAGES

DNS messages



DNS Message Formats

Two basic types: Query and Response



Identification	Flags
Number of question records	Number of answer records (All 0s in query message)
Number of authoritative records (All 0s in query message)	Number of additional records (All 0s in query message)



Header:

Identification: 2 byte field so client may match response to the question. Client creates number, Server just repeats the number in the request

Flags:

- QR Query/Response: One bit 0=query 1=response
- Opcode: four bits define type of query or response 0=normal 1=inverse, 2=server status is requested
- AA authoritative answer: One bit value of 1 means server responding is authoritative server
- TC truncated: One bit if it equals 1 means answer was larger than than 512 bytes and was truncated
- RD recursion desired: one bit if set to 1 means we want a recursive answer
- RA recursion available: One bit when set to 1 means a recursive response is available. This is set only in the response message
- Reserved: three bit field set to 000
- rCode: Four bit field contains error status

Number of Question Records: two byte field with number of queries in the question section of the message

Number of Answer Records: two byte field with number of answers contained in answer section of the message

Number of Authoritative Records: Two byte field containing the number of authoritative records in the authoritative records section of a response message

Number of Additional Records: Two byte field containing the number additional records in the additional section of a response message.


Remainder of DNS Message Format

Question Section: Section consisting of one or more question records. Exists in both query and response

Answer Section: Section consisting of one or more answer records. Exists in response only.

Authoritative Section: Section consisting of one or more resource records. Exists in response only. This contains the domain name about one or more of the authoritative servers for the query.

Additional Info Section: Contains one or more resource records. Exists in response only.

A blue scroll graphic with a white border and two rolled-up corners, containing the title text. The background features abstract blue geometric shapes.

TYPES OF RECORDS

Types of Records

Two Types of Records in DNS

- Question Records are found in Query section and response section of DNS messages. We echo the question record in the response in case you forgot your question before you get your answer :>)
- Resource Records are used the answer section, authoritative section, and additional section of a response message

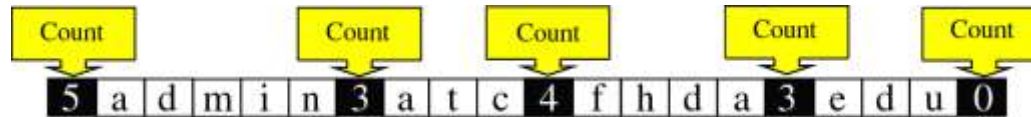
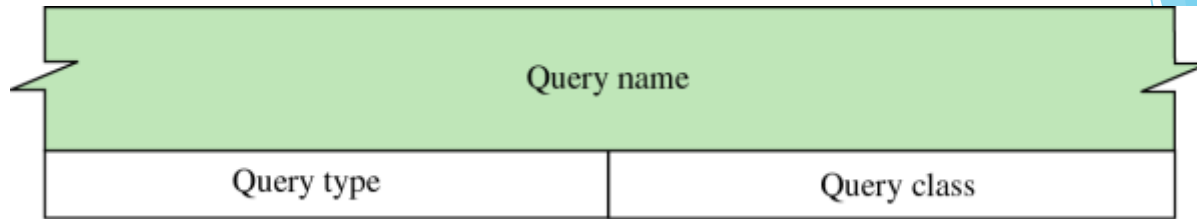
Question Record

Question Record used to get info from server.

Resource Record

Resource records are returned from server to client

Question Record Format



(Each count byte is a binary value between 0 and 63, count bytes are not ASCII)

Query Name: Variable length field containing a domain name

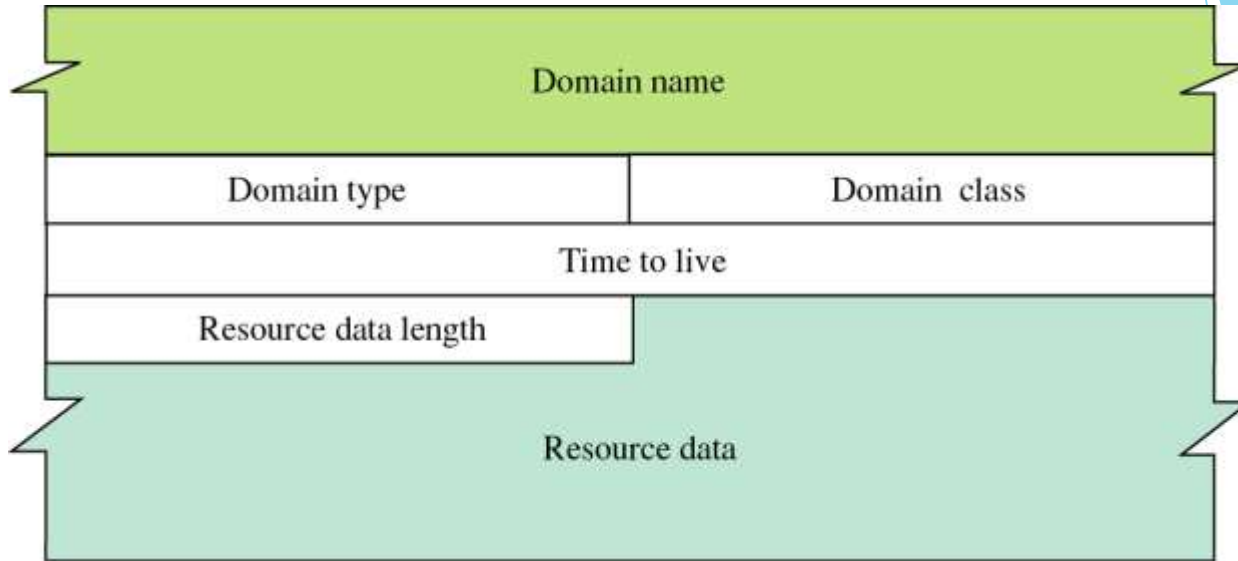
Query Type: 2 byte field containing the type of query:

Type	Mnemonic	Details
1	A	IP Address. Convert a domain name to IP address
2	NS	Name Server. IDs authoritative server for a zone
5	CNAME	Canonical Name. Defines an alias for official name of a host
12	PTR	Convert an IP address to a domain name
etc		

Query Class: 2 Byte field specifying the protocol using DNS. Internet has a value of 1.

Resource Record Format

Resource records are returned from server to client



Domain Name: Variable length field containing domain name

Domain Type: Same as query type field from before but a reduced “Query type” list

Domain Class: 2 Byte field specifying the protocol using DNS. Internet has a value of 1.

Time to Live: 4 byte field with number of seconds answer is valid. Receiver can cache this answer for this period of time

Resource Data Field Length: 2 bytes representing the length of the resource data field

Resource Data: Variable length field containing answer to query



Thank you

The Content in this Material are from the Textbooks and Reference books given in the Syllabus