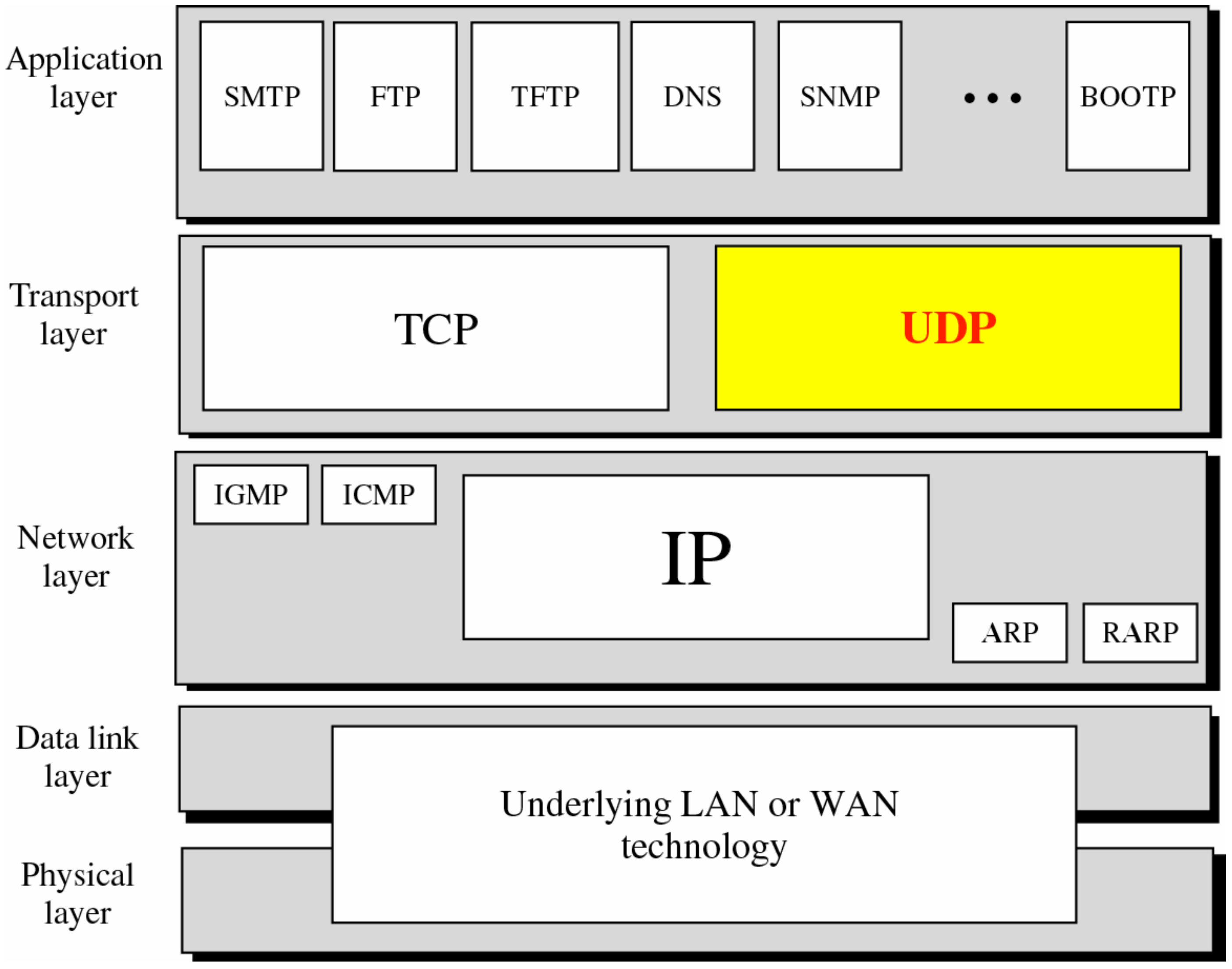
Chapter 3
User Datagram Protocol (UDP)

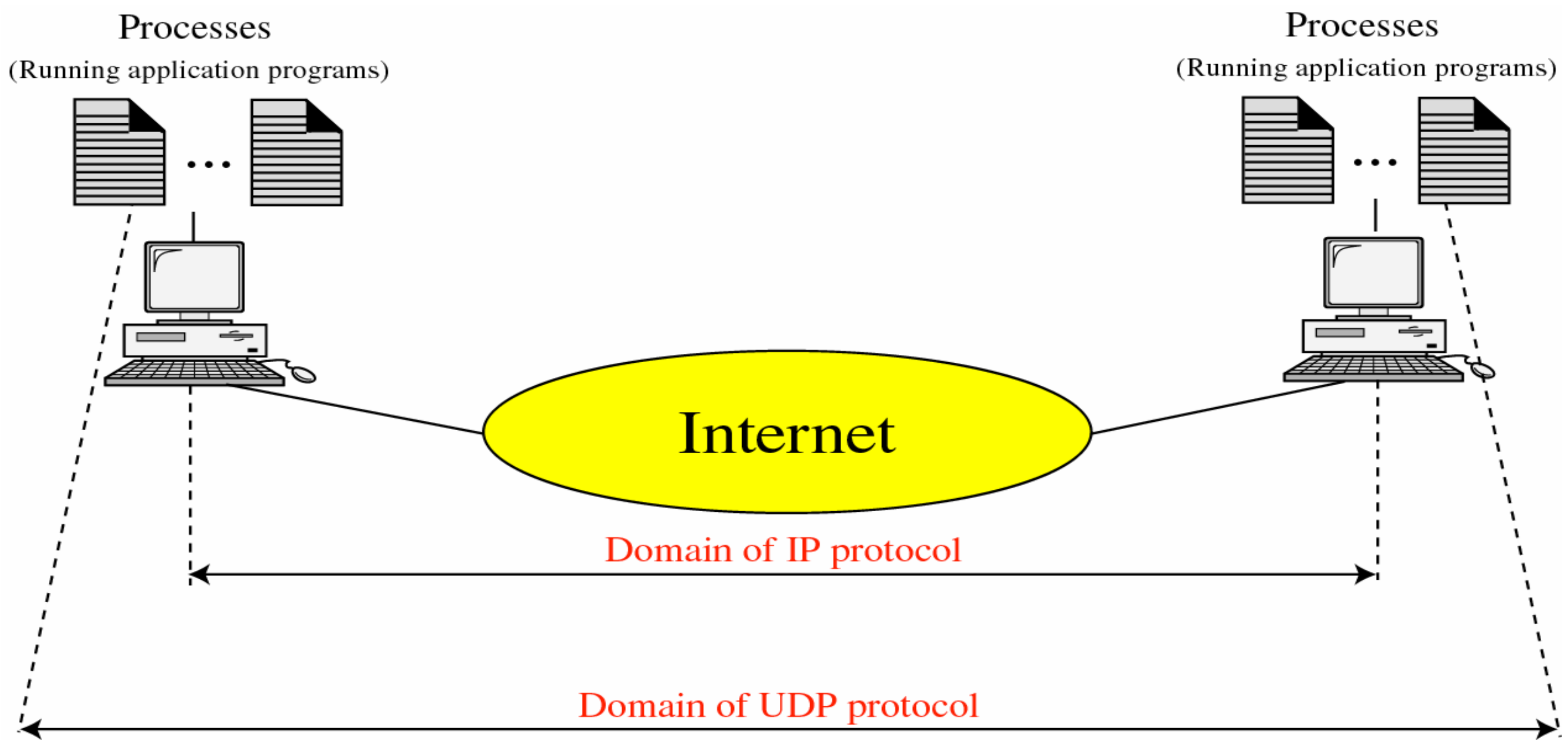
Position of UDP in the TCP/IP protocol suite



Process to Process Communication

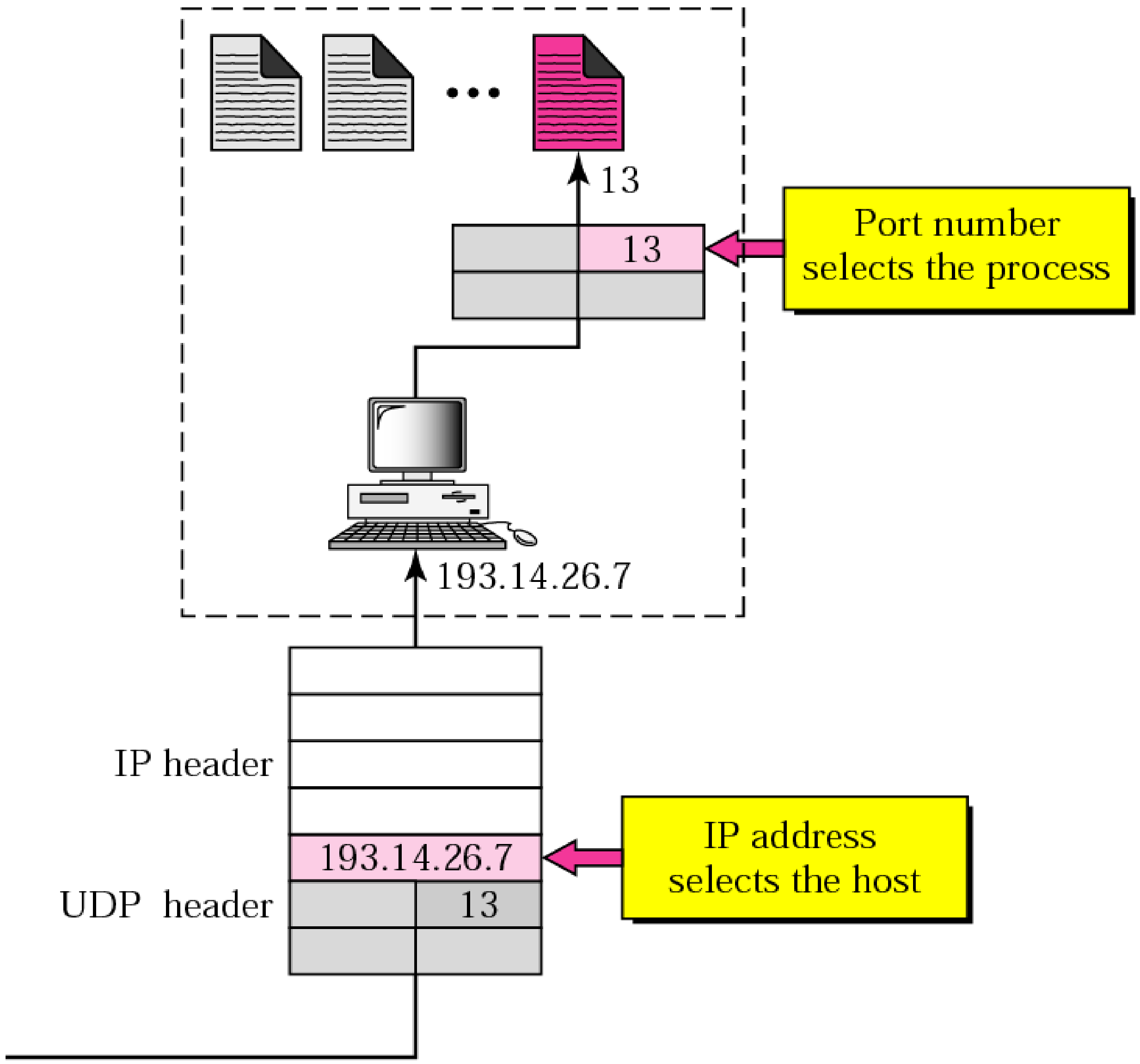
- *Before we examine UDP, we must first understand host-to-host communication and process-to-process communication and the difference between them.*
- At the IP layer, a destination address identifies a host computer; no further distinction is made regarding which user or which application program will receive the datagram.
- Transport layer adds a mechanism that distinguishes among destinations within a given host, allowing multiple application programs (processes) executing on a given computer to send and receive datagrams independently

UDP versus IP



UDP is <u>connectionless</u>, <u>unreliable</u> transport protocol.
It is however very simple protocol that uses minimal overhead.

IP addresses versus port numbers



Ultimate Destination

- The operating systems in most computers support multiprogramming, which means they permit multiple processes to execute simultaneously.
  - The systems are called multitasking systems.
- A process creates and recieves messages. However, concluding a process as ultimate destination for a datagram is misleading because:
  - Processes are created and destroyed dynamically, senders seldom know enough to identify a process on another machine.
  - We would like to be able to replace processes that receive datagrams without informing all senders (e.g., rebooting a machine can change all the processes, but senders should not be required to know about the new processes).

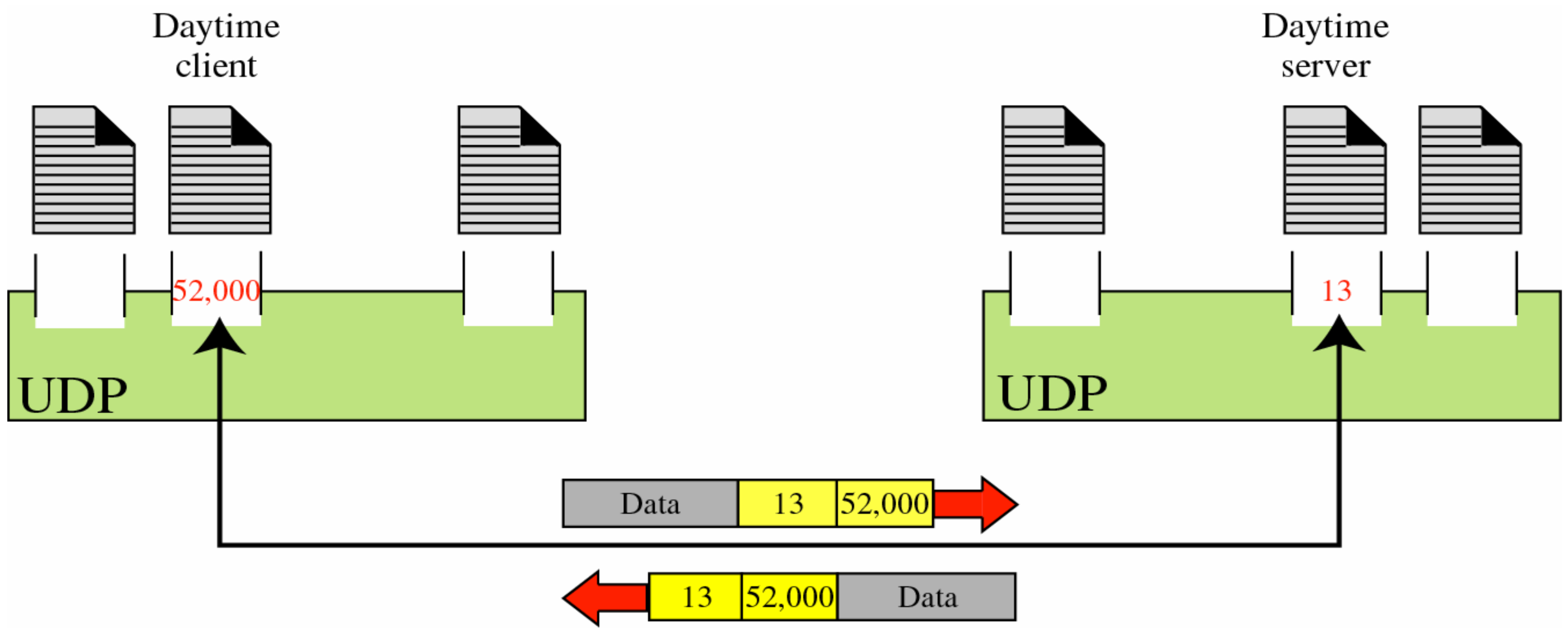
Ultimate Destination Cont’d

- Third, we need to identify destinations from the functions they implement without knowing the process that implements the function (e.g., allow a sender to contact a file server without knowing which process on the destination machine implements the file server function).
- More important, in systems that allow a single process to handle two or more functions, it is essential that we arrange a way for a process to decide exactly which function the sender desires.

Therefore, instead of thinking of a process as the ultimate destination, we will imagine that each machine contains a set of abstract destination points called ***protocol ports.***

- Each protocol port is identified by a positive integer. The local operating system provides an interface mechanism that processes use to specify a port or access it.

Port numbers



Normally servers must have a known port number, while clients can have ephemeral (temporary) port numbers assigned by UDP. (Ephemeros = lasting a day, Greek).

Port numbers $0..2^{16}-1 = 0..65{,}535$

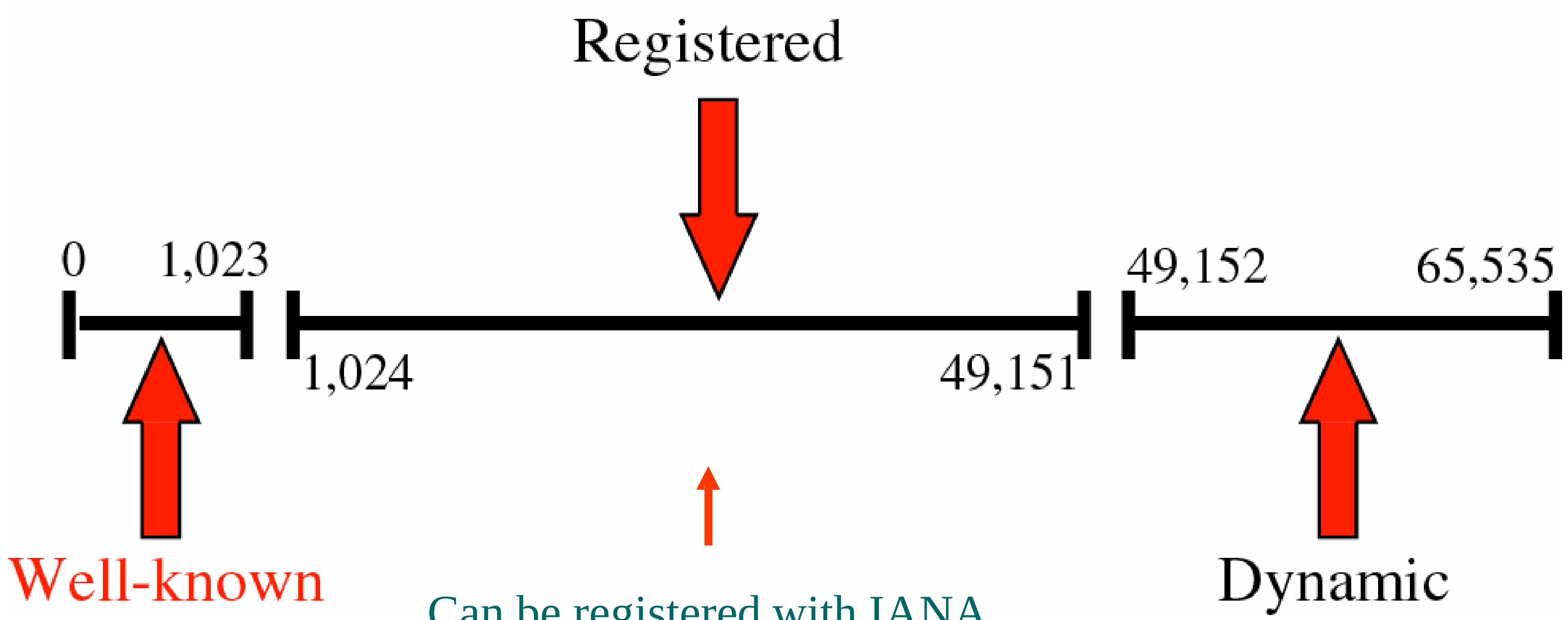
Port numbers Cont’d

To communicate with a foreign port, a sender needs to know both the IP address of the destination machine and the protocol port number of the destination within that machine.

- Each message must carry the number of the ***destination port*** on the machine to which the message is sent, as well as the ***source port*** number on the source machine to which replies should be addressed.
- Thus, it is possible for any process that receives a message to reply to the sender.
- UDP provides protocol ports used to distinguish among multiple programs executing on a single machine.
- UDP software at the destination deliver the message to the correct recipient and enables the recipient to send a reply using these port numbers.

IANA ranges

(IANA = Interned Assigned Numbers Authority)



Can be registered with IANA
to avoid duplications,
but not controlled by IANA

Assigned
and
controlled
by IANA

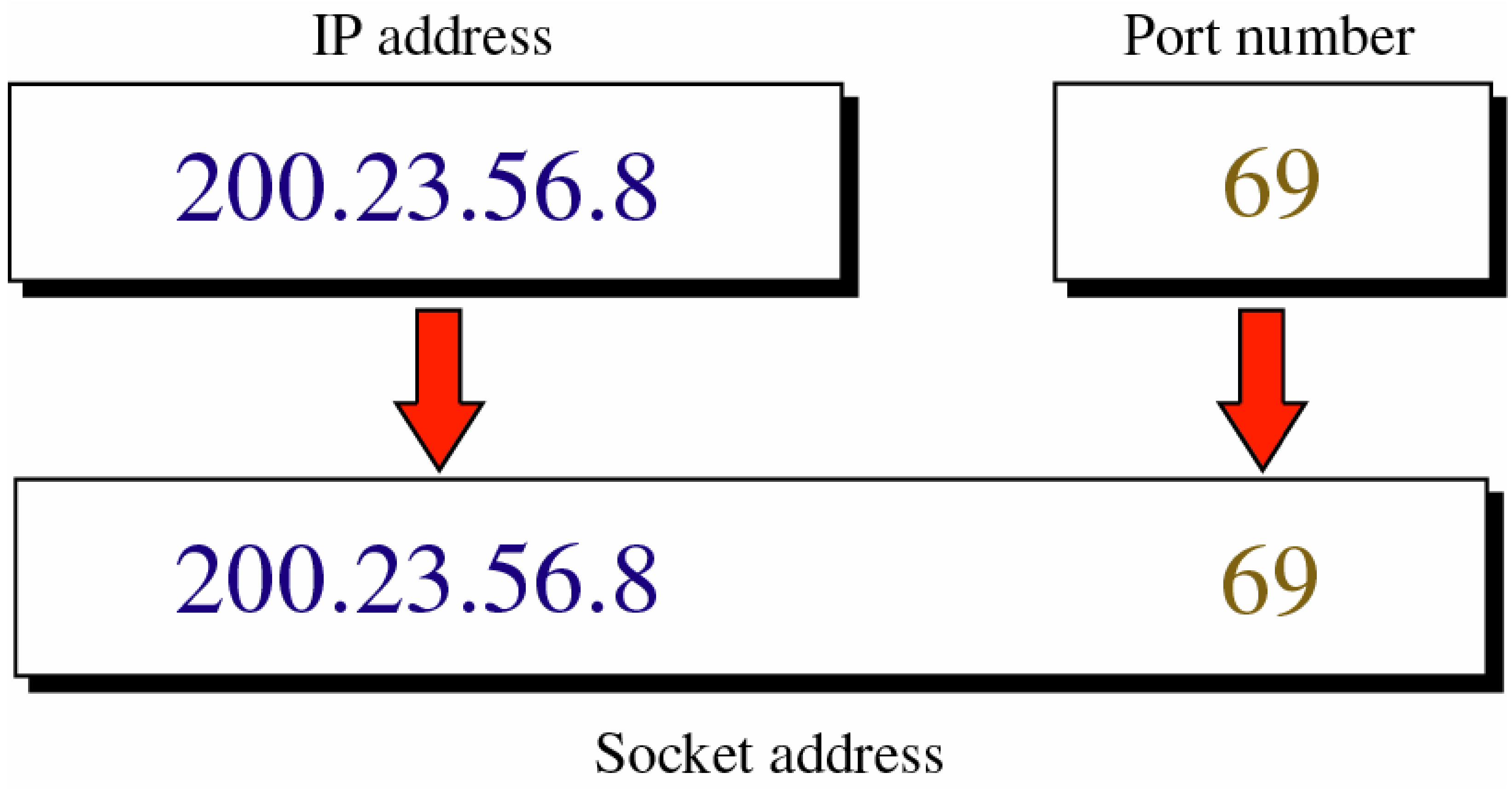
Ephemeral ports

Examples of some IANA assigned well known port numbers (used in UDP and TCP)

Port Number	Description	Port Number	Description
1	TCP Port Service Multiplexer	111	RPC (Remote Procedure Call)
7	Echo (echoes received datagram to sender)	115	Simple File Transfer Protocol (SFTP)
20	FTP -- Data	118	SQL Services
21	FTP -- Control	119	Newsgroup (NNTP)
22	SSH Remote Login Protocol	137	NetBIOS Name Service
23	Telnet	143	Interim Mail Access Protocol (IMAP)
25	Simple Mail Transfer Protocol (SMTP)	150	NetBIOS Session Service
42	Host Name Server (Nameserv)	156	SQL Server
53	Domain Name System (DNS)	161	SNMP
67	Bootps	162	SNMP (trap)
69	Trivial File Transfer Protocol (TFTP)	179	Border Gateway Protocol (BGP)
70	Gopher Services	194	Internet Relay Chat (IRC)
79	Finger	389	Lightweight Directory Access Protocol
80	HTTP	443	HTTPS
103	X.400 Standard	458	Apple QuickTime
108	SNA Gateway Access Server	546	DHCP Client
109	POP2 (Post Office Protocol)	547	DHCP Server

Echo service is a TCP application service that listens on port 7
for echo connection requests (this is not ping)

Socket addresses



UDP Datagram

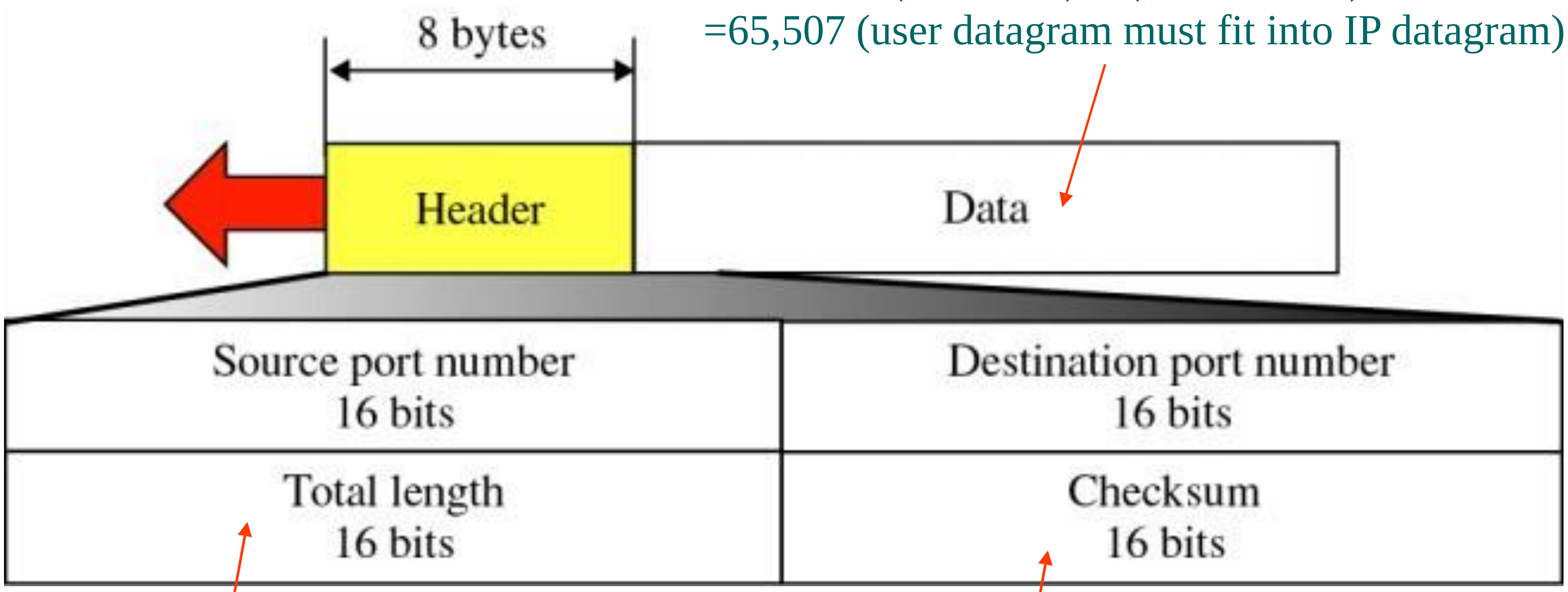
Each UDP message is called a ***user datagram.*** Conceptually, a user datagram consists of two parts: a UDP header and a UDP data area.

> ***The User Datagram Protocol (UDP) provides an unreliable connectionless delivery service using IP to transport messages between machines. It uses IP to carry messages, but adds the ability to distinguish among multiple destinations within a given host computer.***

An application program that uses UDP accepts full responsibility for handling the problem of reliability, including message loss, duplication, delay, out-of-order delivery, and loss of connectivity.

User datagram

Max length of data:
65,535-20 (IP header)-8 (UDP header)
=65,507 (user datagram must fit into IP datagram)



Total length of UDP datagram in bytes (header and data)

Header and data!
Can be disabled by filling the field with all 0s

UDPlength = IP length – IP header length

UDP Datagram Cont’d

- ***SOURCE PORT*** and ***DESTINATION PORT*** fields contain the 16-bit UDP protocol port numbers used to demultiplex datagram among the processes waiting to receive them.
  - ***SOURCE PORT*** is optional. When used, it specifies the port to which replies should be sent; if not used, it should be zero.
- ***LENGTH*** field contains a count of octets in the UDP datagram, including the UDP header and the user data.
  - Min value for ***LENGTH*** is eight, the length of the header.
- UDP checksum is optional; a value of zero in the ***CHECKSUM*** field means that the checksum has not been computed.
- Designers chose to make the checksum optional to allow implementations to operate with little computational overhead when using UDP across a highly reliable local area network.

UDP Checksum

Checksum is calculated for UDP header and data (recall: IP checksum doesn’t cover data).

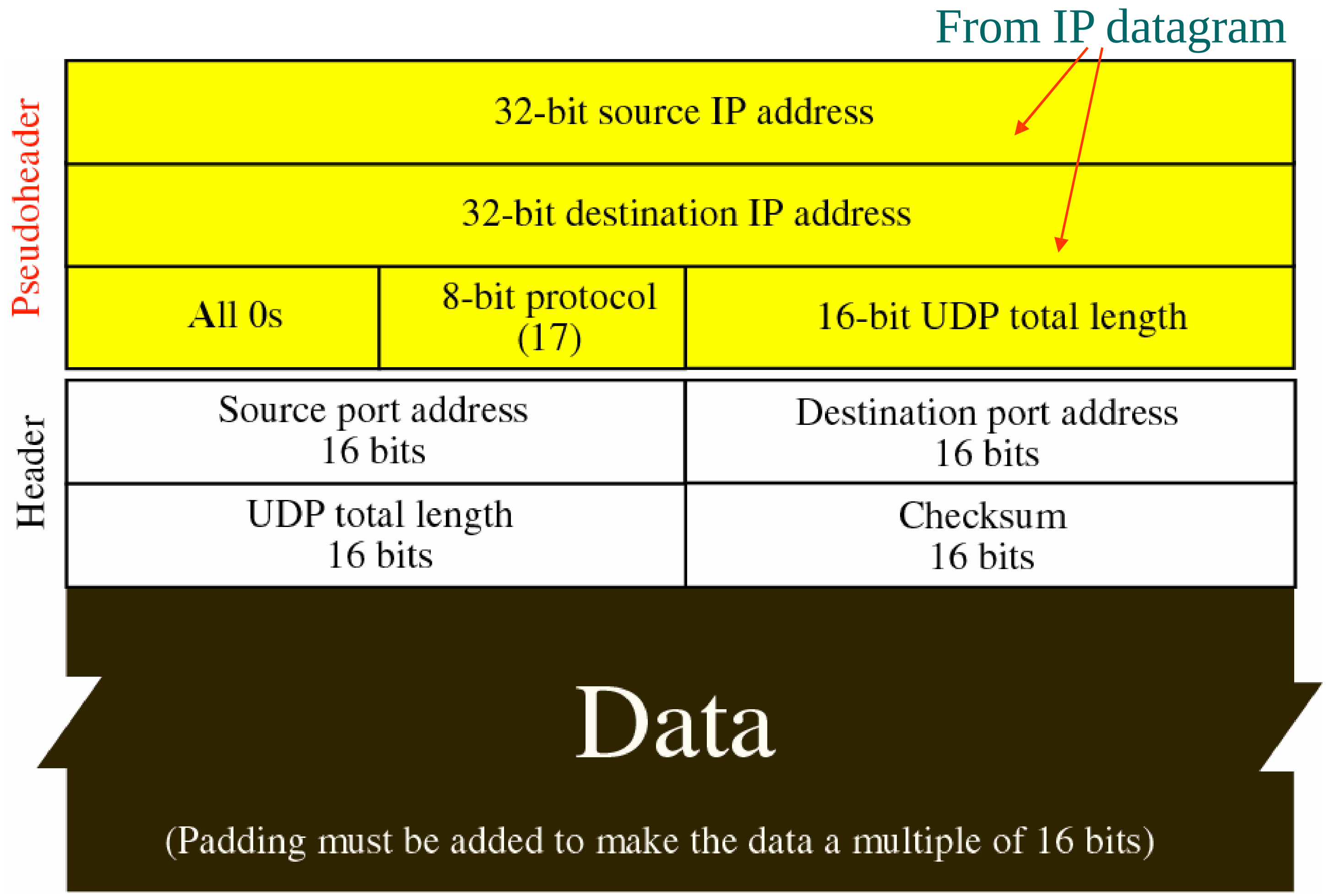
UDP checksum test is end-to-end test, i.e. test performed only at the sender and receiver end stations, while the IP checksum test is performed in every intermediate node (router).

In addition to UDP header and data, the UDP checksum includes the source and the destination IP address in order to prevent misrouting. Suppose that the destination IP address in IP header was corrupted, i.e. changed to some other IP address, and that this change wasn’t discovered by the IP checksum test. Consequently, the UDP datagram would arrive to the wrong IP address. UDP can detect this and silently drop the datagram.

Inclusion of the IP destination address into UDP checksum test is achieved through the usage of the UDP pseudoheader which is added by the UDP layer at both sides, sender and receiver. The UDP layer gets the entire IP datagram from the IP layer, which it uses to form the pseudoheader. (The IP header is saved in UDP for possible ICMP error messages which include the IP header of the offending datagram)

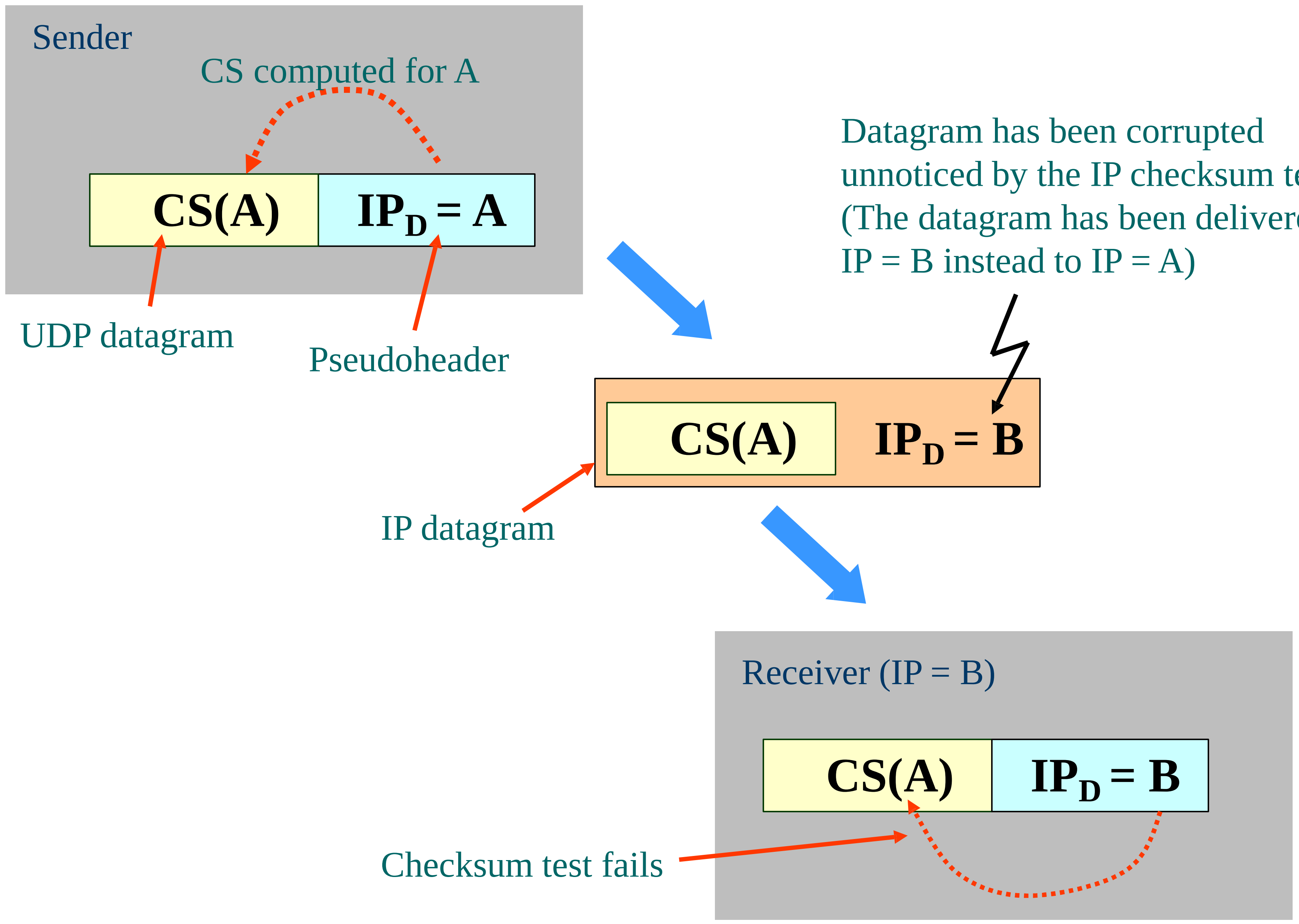
Pseudo Header

Pseudo header is added to the UDP datagram at the time of checksum calculation (at transmitter and at receiver). The pseudo header in not transmitted. Used to verify the correctness of the IP address (see next slide).

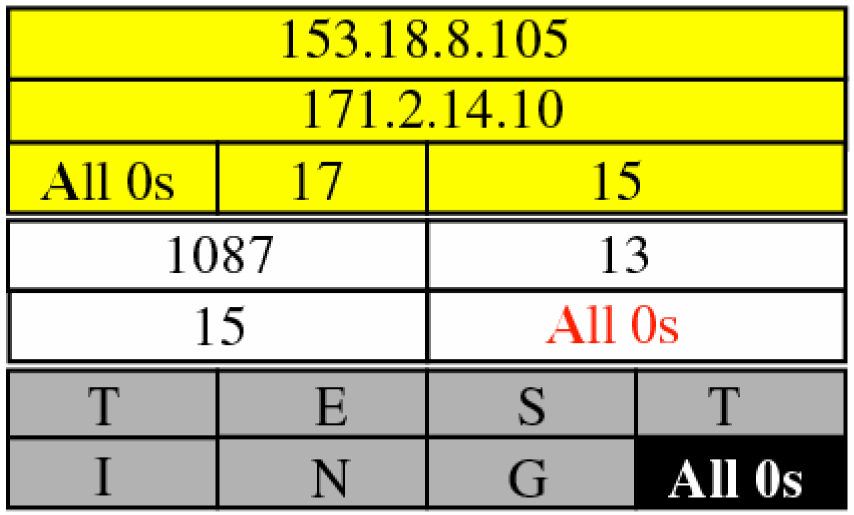

From IP datagram
Pseudoheader
32-bit source IP address
32-bit destination IP address
All 0s
8-bit protocol (17)
16-bit UDP total length
Header
Source port address 16 bits
Destination port address 16 bits
UDP total length 16 bits
Checksum 16 bits
Data
(Padding must be added to make the data a multiple of 16 bits)

Neither the pseudo header nor the padding bits are transmitted with the UDP nor are they included with the length.

UDP Checksum



Checksum calculation of a simple UDP user datagram



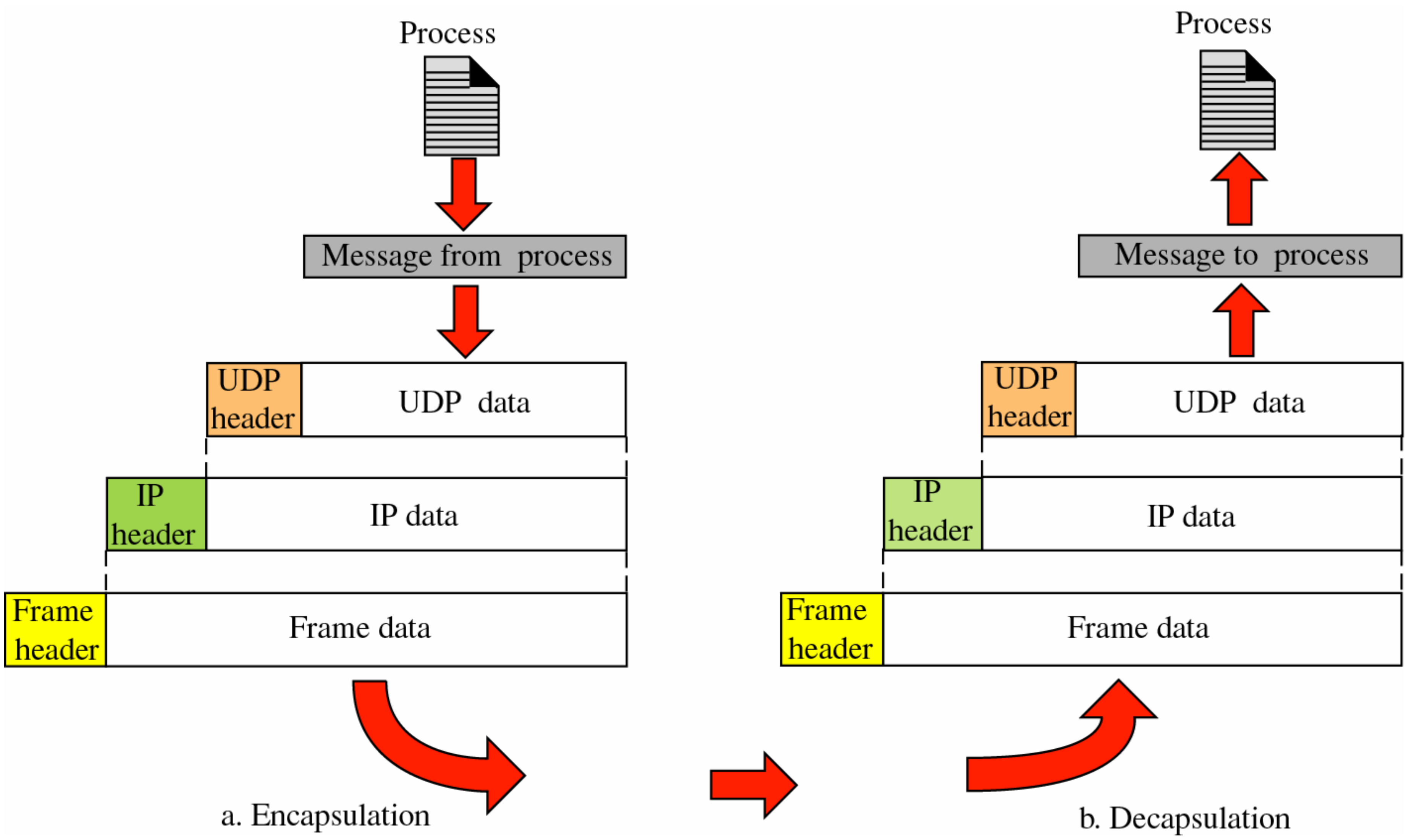
10011001	00010010	→ 153.18
00001000	01101001	→ 8.105
10101011	00000010	→ 171.2
00001110	00001010	→ 14.10
00000000	00010001	→ 0 and 17
00000000	00001111	→ 15
00000100	00111111	→ 1087
00000000	00001101	→ 13
00000000	00001111	→ 15
00000000	00000000	→ 0 (checksum)
01010100	01000101	→ T and E
01010011	01010100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

NOTICE: The UDP checksum is optional (some OS vendors disable CS by default).
If the CS is not calculated at the sender side, the CS field is filled with all 0s which is one's complement of all 1s (negative zero).
The receiver recognizes that and doesn't apply the CS test.

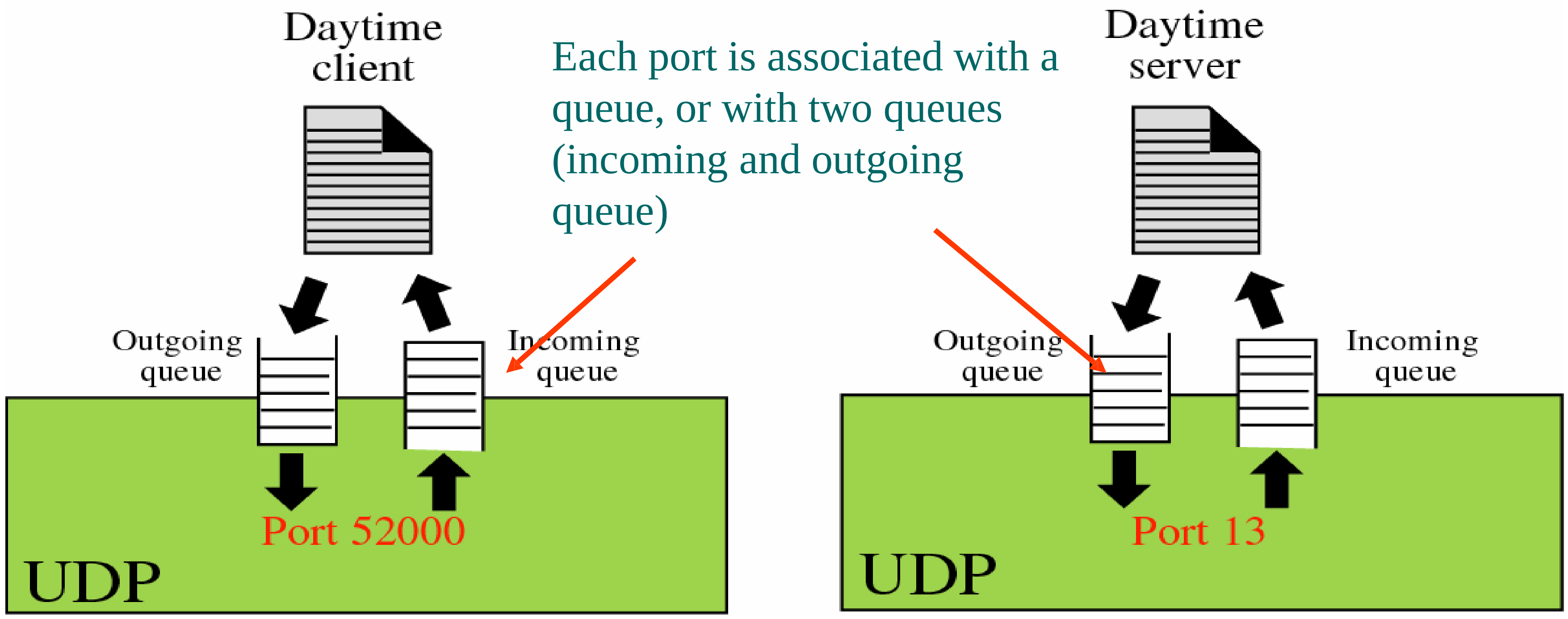
Checksum calculation cases

- What happens to UDP messages for which the computed checksum is zero?
- A computed value of zero is possible because UDP uses the same checksum algorithm as IP: it divides the data into 16-bit quantities and computes the one's complement of their one's complement sum.
- Surprisingly, zero is not a problem because one's complement arithmetic has two representations for zero: all bits set to zero or all bits set to one. When the computed checksum is zero, UDP uses the representation with all bits set to one.

Encapsulation and decapsulation



Queues in UDP

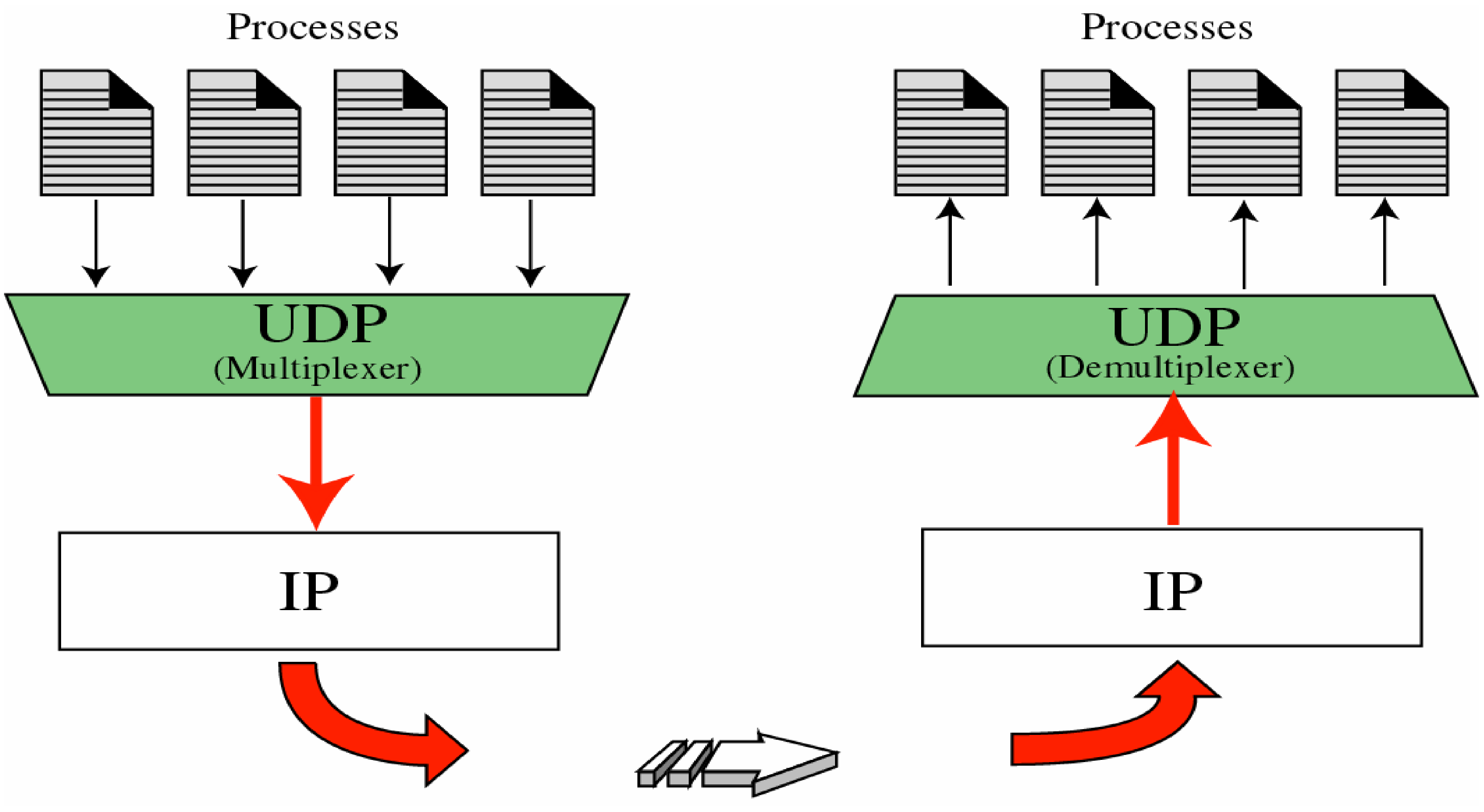


Each port is associated with a queue, or with two queues (incoming and outgoing queue)

If the port number doesn't exists at the time a UDP datagram arrives it will be dropped and an ICMP message (port unreachable) will be sent to the source. Same happens if an existing queue gets overflown.

Multiplexing and demultiplexing

There is only one UDP in sender and receiver, and possibly several processes using it. Therefore multiplexing is needed.
(Multiplexing is achieved through port numbers.)



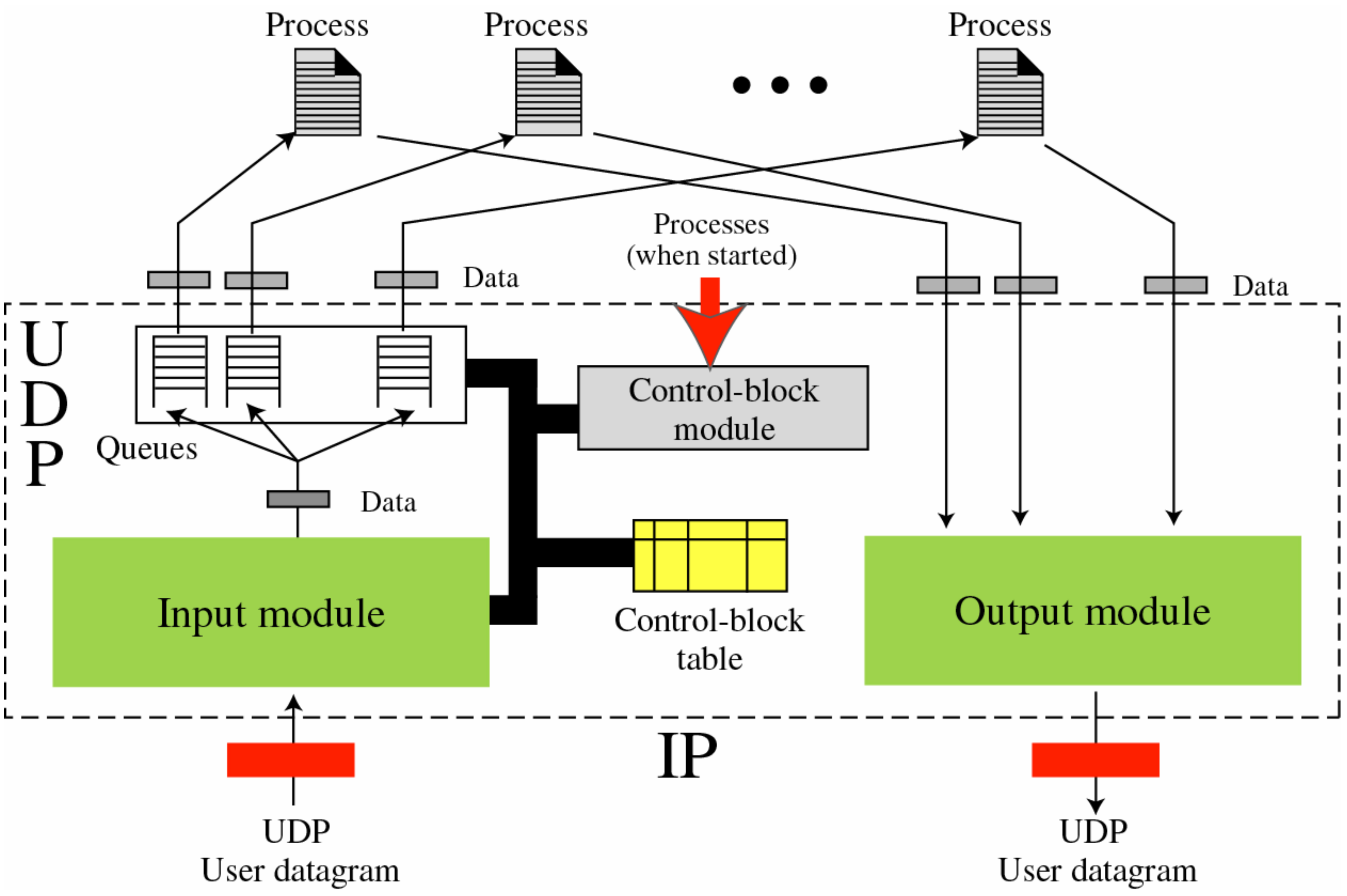
Use of UDP

- UDP is suitable for a process that requires simple request-response communication where flow and error control are not crucial.
- UDP is not suitable for transmitting a large amount of data (like FTP) because they are encapsulated into IP datagrams that are independently routed.
- UDP is suitable for processes that have internal flow/error control (like TFTP)
- UDP is suitable for multicasting and broadcasting.
- UDP is used in network management (like Simple Network Management Protocol (SNMP)
- UDP is used for some route updating protocols (like RIP)
- UDP is preferred in some real-time applications such as voice and telemetry involving certain degree of redundancy (temperature measurement, streaming video,…)
- UDP is also preferred in applications that have short transactions like NSF, DNS, RIP.

UDP Package

- *To show how UDP handles the sending and receiving of UDP packets, we present a simple version of the UDP package. The UDP package involves five components: a control-block table, input queues, a control-block module, an input module, and an output module*.

UDP package


Process
Process
Process
Processes
(when started)
Data
Data
UDP
Queues
Data
Control-block
module
Control-block
table
Input module
Output module
IP
UDP
User datagram
UDP
User datagram

Control-Block Table

Keeps track of the open ports

State	Process ID	Port Number	Queue Number

Input Module

```
Receive UDP datagram from IP;
Construct the pseudoheader;
Compute checksum;
if (CS test fails)
      Silently drop the UDP datagram;

Search the Control Block Table (CBT);
if (there is a corresponding entry in CBT)
      if (queue not yet allocated exists)
            Allocate a queue;
      Insert data into queue;
else{
      Ask ICMP to send "Port unreachable" message;
      Discard the UDP datagram;
}
return;
```

Control Block Module

```
// Invoked when a socket is created
Get port ID (input parameter);
Get process ID;
Search the CBT for a free entry;
if (table full)
        Delete an entry using a predefined strategy;
Create new entry in CBT;
Enter the process ID and port ID;
return;
```

Output Module

```
Receive data and destination address structure from
process;
Create UDP header;
Create pseudoheader (with zero CS field);
Compute check sum and fill the CS field;
Encapsulate data into UDP datagram (without pseudo
header);
Send the detagram to IP layer;
return;
```

Control-block table at the beginning

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

Example 1

The first activity is the arrival of a user datagram with destination port number 52,012. The input module searches for this port number and finds it. Queue number 38 has been assigned to this port, which means that the port has been previously used. The input module sends the data to queue 38. The control-block table does not change.

Example 2

After a few seconds, a process starts. It asks the operating system for a port number and is granted port number 52,014. Now the process sends its ID (4,978) and the port number to the control-block module to create an entry in table. The module does not allocate a queue at this moment because no user datagrams have arrived for this destination

State	***Process ID***	***Port Number***	***Queue Number***
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			

Example 3

A user datagram now arrives for port 52,011. The 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 creates a queue and gives it a number (43).

State	*Process ID*	*Port Number*	*Queue Number*
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			

Example 4

After a few seconds, a user datagram arrives for port 52,222. The input module checks the table and cannot find the entry for this destination. The user datagram is dropped and a request is made to ICMP to send an "unreachable port" message to the source.