User Datagram Protocol (UDP)

Position in TCP/IP Protocol Suite

- UDP is a **transport layer protocol**.
- It sits **between the Application Layer and IP Layer**.
- **Role**: Acts as a bridge between application programs (like DNS, DHCP, etc.) and the network operations.
  UDP is a simple, connectionless, and unreliable transport protocol that provides process-to-process communication using port numbers.

UDP Responsibilities

- **Process-to-Process Communication** → Uses **port numbers**.
- **Minimal Control** → Very limited features:
  - No flow control (doesn't slow down or manage speed).
  - No acknowledgment (receiver doesn't confirm receipt).
  - Some error control → If error is found, UDP just **drops the packet silently**.
- It does not improve IP services, except it changes **host-to-host → process-to-process** communication.

UDP = Connectionless + Unreliable
(no guarantee that data will reach, but very fast and simple).

Why Use UDP?

Even though UDP looks weak, it has **advantages**:

- Very **simple** (minimum overhead, small header).
- **Faster** than TCP.
- Best for **small messages** where reliability is not important (like DNS queries, online games, voice/video calls).

Example: Sending "Hello" through UDP is faster because there is no need for acknowledgment or connection setup like TCP.

UDP User Datagram

- A UDP packet is called a **user datagram**.
- Each datagram has:

- **Header** → 8 bytes fixed size.
- **Data** → 0 to 65,527 bytes (because total max size = 65,535).

UDP Header Format (8 bytes)

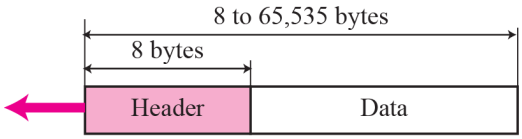
1. **Source Port Number (16 bits)**
   - Port number of the sender process.
   - Range: 0 – 65,535.
   - Client → usually uses **ephemeral port** (temporary, chosen randomly).
   - Server → usually uses **well-known port** (fixed, like 53 for DNS).
2. **Destination Port Number (16 bits)**
   - Port number of the receiving process.
   - If destination is a server → usually a **well-known port**.
   - If destination is a client → usually an **ephemeral port** (copied from request).
3. **Length (16 bits)**
   - Total size of UDP packet (header + data).
   - Formula:
     `UDP Length = IP Total Length – IP Header Length`
   - Still included in UDP header for **efficiency**.
4. **Checksum (16 bits)**
   - Used to detect **errors** in header + data.
   - If error is found, packet is dropped.

Example 14.1 Explained

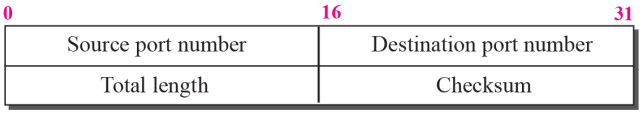
UDP header dump: `CB84 000D 001C 001C`

- **Source Port** = `CB84 (hex)` = 52100 (decimal) → client's ephemeral port.
- **Destination Port** = `000D (hex)` = 13 → well-known port (Daytime service).
- **Total Length** = `001C (hex)` = 28 bytes.
- **Data Length** = 28 – 8 = 20 bytes, as header is fixed 8 byte.
- Since destination port is **well-known (13)** → packet is **client → server**.

- Client process = **Daytime service**.



a. UDP user datagram



b. Header format

UDP Services

In Chapter 13, you learned that transport layer protocols (like TCP/UDP) usually provide several services.
But UDP provides only a **minimal set** of them. Let's see which services are supported.

Process-to-Process Communication

- UDP provides **process-to-process communication** using **sockets**.
- A socket = **IP address + Port number**.
- Each process on a host is identified by a **port number**.
- Example: DNS (Port 53), SNMP (Port 161), etc.

A socket is the combination of an IP address and a port number that uniquely identifies a process in a network.

Connectionless Service

- UDP is **connectionless**.
- Each user datagram is **independent**.
- No relationship between packets, even if from the same source and going to the same destination.
- Packets are **not numbered**, and there is: No connection setup, No connection termination
- Each packet may take **different paths** in the network.
- Because of this, **UDP cannot handle continuous data streams**.
    - Data must fit into a single datagram (max ~65,507 bytes = 65,535 – 8 (UDP header) – 20 (IP header)).

Only short messages (small packets) are suitable for UDP.

Flow Control

- UDP does **not provide flow control**.

- Receiver may **overflow** if too many messages arrive quickly.
- If needed, the **application itself** must handle flow control.

No sliding window, no rate control in UDP.

Error Control

- UDP does not provide full error control.
- Only **checksum** is available.
- If checksum detects error → datagram is **silently discarded**.
- Sender does not know if data was lost or duplicated.
- If needed, the **application** must handle error correction.

UDP is unreliable, no retransmission.

Well-known UDP Ports (Table 14.1)

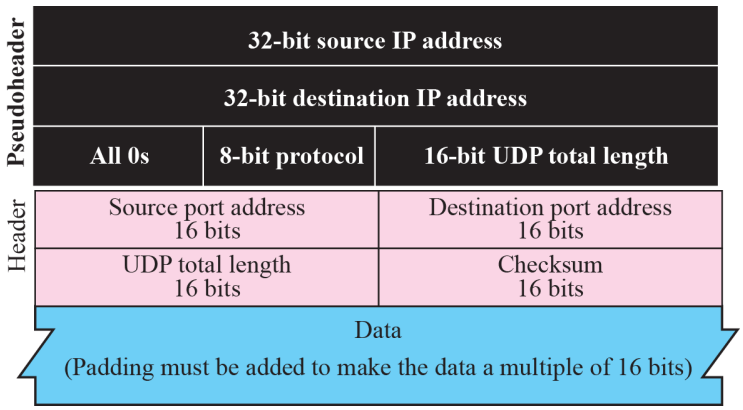
- **7** → Echo (returns the same message back)
- **9** → Discard (throws away received message)
- **11** → Users (lists active users)
- **13** → Daytime (returns date and time)
- **17** → Quote of the Day
- **19** → Chargen (returns random characters)
- **53** → DNS (Domain Name Service)
- **67/68** → BOOTP (bootstrap for clients/servers)
- **69** → TFTP (Trivial File Transfer Protocol)
- **111** → RPC (Remote Procedure Call)
- **123** → NTP (Network Time Protocol)
- **161/162** → SNMP (Simple Network Management Protocol)
- **53 (DNS), 67/68 (BOOTP), 69 (TFTP), 123 (NTP), 161 (SNMP)**

Checksum in UDP

- Checksum ensures **data integrity** (detects errors in header + data).
- But UDP checksum calculation uses **pseudoheader + UDP header + data**.

Pseudoheader includes:

- Source IP address
- Destination IP address
- Protocol (UDP = 17)
- UDP total length



Why pseudoheader?

- To make sure the packet is delivered to the **correct host** and **correct protocol** (UDP, not TCP).
- Sender may **skip checksum** → put all **0s** in checksum field.
- If checksum is calculated and result is **0**, it is changed to **all 1s** (to avoid confusion with "checksum not used").

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

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

FOr the first sum , we will do addition of one complement

তুমি সব 16-bit word গুলোকে **binary তে যোগ করবে**,
কিন্তু যদি **carry বের হয় (মানে 17th bit)** → সেটা আবার **ফলাফলের সাথে যোগ করতে হবে**।

ধরা যাক আমরা দুটি 16-bit word যোগ করছি:Word 1 = 10011001 00010010 ,Word 2 = 00001000 01101001

```
10011001 00010010
+00001000 01101001
------------------
10010001 011111011  ← (এখানে এক্সট্রা 1-bit carry আছে সামনে)
```

যেহেতু এটা 17-bit হয়ে গেছে, তাই **ওই carry (1)** টা আবার নিচের 16-bit ফলাফলের সাথে যোগ করতে হবে।

```
Result = 00100101 11111011
Carry = 1
------------------
Final sum = 00100110 11111011
```

কেন এই "carry যোগ" করার নিয়ম?

এটা হলো **one's complement addition** এর নিয়ম।
এতে "overflow" (অতিরিক্ত bit) কে বাদ না দিয়ে **ফলাফলের সাথে ঘুরিয়ে (wrap-around)** যোগ করা হয়।
এর ফলে checksum সঠিকভাবে কাজ করে।

শেষে One's Complement নেওয়া

যখন সব word যোগ শেষ হবে → পুরো sum এর **1's complement** (মানে bit উল্টে) নিও।

```
Sum:          10010110 11101011
1's complement: 01101001 00010100
```

এইটা final **checksum** হবে

ধাপ	কাজ	ব্যাখ্যা
1️⃣	UDP header + data কে 16-bit word এ ভাগ করো	যেমন 8-bit + 8-bit = 16-bit
2️⃣	সব word যোগ করো **binary তে**	সাধারণ binary addition
3️⃣	Carry থাকলে ফলাফলের সাথে যোগ করো	one's complement addition
4️⃣	শেষে সব bit উল্টাও	একে বলে **one's complement**
5️⃣	এটাই checksum	UDP header এ বসাও

a. The sender decides not to include the checksum?

The value sent for the checksum field is all 0s to show that the checksum is not calculated.

UDP–তে checksum optional।

1. Checksum বাদ দিতে চাইলে
    - Sender simply checksum ফিল্ডে all 0s লিখে দেয়।
    - Receiver বুঝে যায় → "এখানে checksum ব্যবহার করা হয়নি"।

b. The sender decides to include the checksum, but the value of the sum is all 1s. When the sender complements the sum, the result is all 0s; the sender complements the result again before sending. The value sent for the checksum is all 1s. The second complement operation is needed to avoid confusion with the case in part a.

2. Checksum যদি হিসাব করে 0 আসে?

   - ধরো সবগুলো 16-bit word যোগ করে one's complement নিলে ফলাফল 0000 0000 0000 0000 আসলো।
   - কিন্তু 0 মানে তো "checksum ব্যবহার হয়নি" (case-1 এর মতো)।
   - তাই এই confusion এড়াতে → result 0 হলে সেটাকে 1111 1111 1111 1111 (all 1s) করে রাখা হয়।

c. The sender decides to include the checksum, but the value of the sum is all 0s. This situation never happens because it implies that the value of every term included in the calculation of the sum is all 0s, which is impossible; some fields in the pseudoheader have nonzero valuesChecksum result আসলে কখনও 0 হবে না

   - কারণ checksum হিসাব করার সময় pseudo-header এর মধ্যে সবসময় nonzero fields থাকে (যেমন source IP, destination IP, protocol number 17, UDP length ইত্যাদি)।
   - এগুলোর জন্য result mathematically একেবারে 0 হওয়া সম্ভব না।
   - তাই practically "all 0s" মানে শুধু checksum not used।

Congestion Control

- UDP has **no congestion control** because it is **connectionless**.
- It assumes that packets are **small and occasional**, so they won't flood the network.
- But today, when UDP is used for **real-time audio/video streaming**, this assumption may not always be true.

No congestion control in UDP → risk of network congestion in heavy use.

Congestion Control = নেটওয়ার্কে অতিরিক্ত ভিড় এড়ানো।

- একটা রাস্তায় অনেক গাড়ি একসাথে ঢুকে গেলে জ্যাম হয়।
- তেমনি, নেটওয়ার্কে যদি অনেক ডেটা একসাথে পাঠানো হয় → router/switch buffer ভরে যায় → packet drop হয়।
- Congestion control এর কাজ হলো: **sender কে নিয়ন্ত্রণ করা**, যেন সে নেটওয়ার্কের অবস্থা বুঝে ধীরে/সতর্কভাবে ডেটা পাঠায়।

কেন UDP তে congestion control নেই?

UDP ডিজাইন করা হয়েছিল **ছোট ও occasional packet** পাঠানোর জন্য (যেমন DNS query, SNMP ইত্যাদি)। তখন ধারণা ছিল, এগুলো network flood করবে না।

- এই ক্ষেত্রে **হাজার হাজার প্যাকেট একসাথে যায়**।
- যেহেতু congestion control নাই → নেটওয়ার্কে **flooding বা congestion** হতে পারে।
- **TCP** → congestion control আছে → নেটওয়ার্কে ভিড় হলে ধীর হয়ে যায়।
- **UDP** → congestion control নেই → network overload এর ঝুঁকি থাকে।

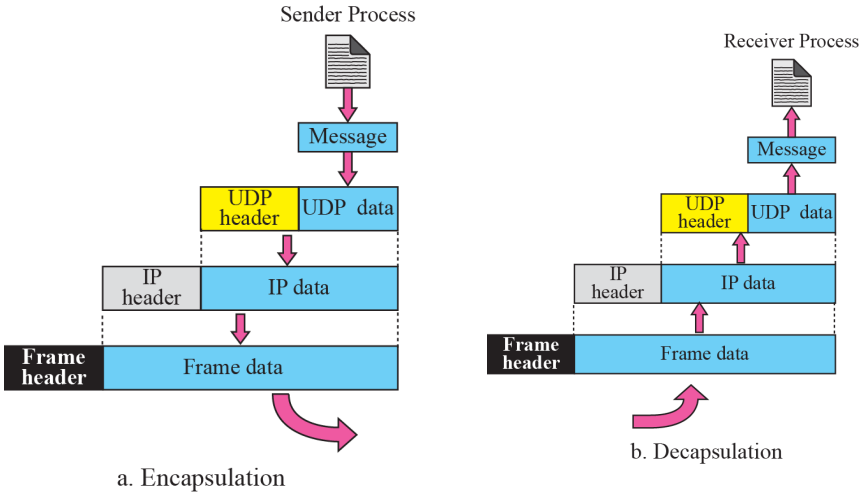
Encapsulation & Decapsulation

Encapsulation (Sender side):

1. Application process creates a **message**.
2. UDP adds its **header** → makes a **user datagram**.
3. Passes it to IP → IP adds its header (with protocol = 17 for UDP).
4. Then Data Link layer adds frame header (and trailer).
5. Physical layer sends as electrical/optical signals.

Decapsulation (Receiver side):

1. Physical layer → converts signals back to bits.
2. Data Link layer → removes frame header/trailer.
3. IP layer → removes IP header.
4. UDP → checks checksum, removes UDP header.
5. Application receives original message + sender's socket address.



Encapsulation = wrapping message with headers. Decapsulation = removing headers to deliver message.

Queuing

- In UDP, each **port number** is linked with a **queue** (for incoming/outgoing messages).
- **Client side**:
  - Gets an **ephemeral port number** (temporary).
  - Messages are placed in outgoing queue → UDP sends them.
  - Incoming queue stores received messages.
  - If a queue is full → messages are dropped, and ICMP sends "port unreachable."
- **Server side**:
  - Uses **well-known port numbers** (e.g., DNS uses port 53).
  - Has permanent incoming & outgoing queues while running.
  - Same rule: if queues overflow → messages dropped, ICMP reply sent.

Queues store UDP datagrams before sending/after receiving; if overflow → message lost.

Multiplexing & Demultiplexing

- **Multiplexing (Sender side):**
  - Many processes (apps) → one UDP.
  - UDP adds port numbers to separate them.
  - Passes all datagrams to IP.
- **Demultiplexing (Receiver side):**
  - One UDP → many processes.
  - Uses **destination port number** to deliver each datagram to the correct process.
- Multiplexing = many processes share UDP.
- Demultiplexing = UDP delivers datagram to correct process.

UDP vs Generic Simple Protocol

- UDP is almost the same as the **connectionless simple protocol** (Chapter 13).
- The only difference: UDP has an **optional checksum** for error detection.
- If checksum shows error → datagram is discarded (no feedback to sender).

UDP = Simple connectionless protocol + optional checksum.

UDP Applications

UDP is not reliable (no error control, no congestion control, connectionless), but it is still used in many applications because sometimes **speed and low delay are more important than reliability**. Think of it like **fast delivery vs safe delivery**. TCP = safe but slow, UDP = fast but may lose packets.

1️⃣ Connectionless Service

- UDP is **connectionless**: every packet is independent.
- **Advantage**: Good for short request-response communications.
- **Disadvantage**: Bad for long messages (packets may arrive out of order).

Example 14.4 (Good):

- DNS (Domain Name System)
- Client sends a short query → Server sends short reply → only **2 packets** needed (fast).

Example 14.5 (Bad):

- SMTP (Email)
- Email is long (with text, images, audio, etc.).
- If sent over UDP → split into many packets → may arrive **out of order** → problem.
- SMTP prefers TCP because delay is okay but order must be correct.
- **KEY POINT-**
- Short messages → UDP better.
- Long messages → TCP better.

Lack of Error Control

- UDP = **unreliable** (no retransmission of lost packets).
- TCP resends lost/corrupted packets → causes uneven delays.
- Sometimes reliability is important, sometimes it isn't.

Example 14.6 (Needs reliability):

- Downloading a large text file.
- Must be 100% correct (no missing part).

- Use TCP, not UDP.

Example 14.7 (Doesn't need full reliability):

- Watching a real-time video stream.
- If one packet is lost → that frame is skipped → very short blank (not noticeable).
- If TCP was used → waiting for retransmission → video freezes (worse).
- So, UDP is better for **real-time video/audio/voice**.

Key point:

- Applications needing **accuracy** (files) → TCP.
- Applications needing **speed** (real-time) → UDP.

Lack of Congestion Control

- UDP has **no congestion control**.
- Advantage: Doesn't add extra traffic by retransmitting packets like TCP does.
- So, in congested networks, UDP may actually work better.No congestion control = double-edged sword (fast but may worsen network issues if used heavily).

Typical Applications of UDP

UDP is chosen when: **speed > reliability** or application has its own error control.

- **Simple request-response**: e.g., DNS (short queries).
- **Apps with built-in error control**: e.g., TFTP (Trivial File Transfer Protocol).
- **Multicasting**: UDP supports multicasting; TCP doesn't.
- **Network management**: e.g., SNMP (Simple Network Management Protocol).
- **Routing protocols**: e.g., RIP (Routing Information Protocol).
- **Real-time applications**: Voice, Video, Streaming (where delay is worse than small data loss).

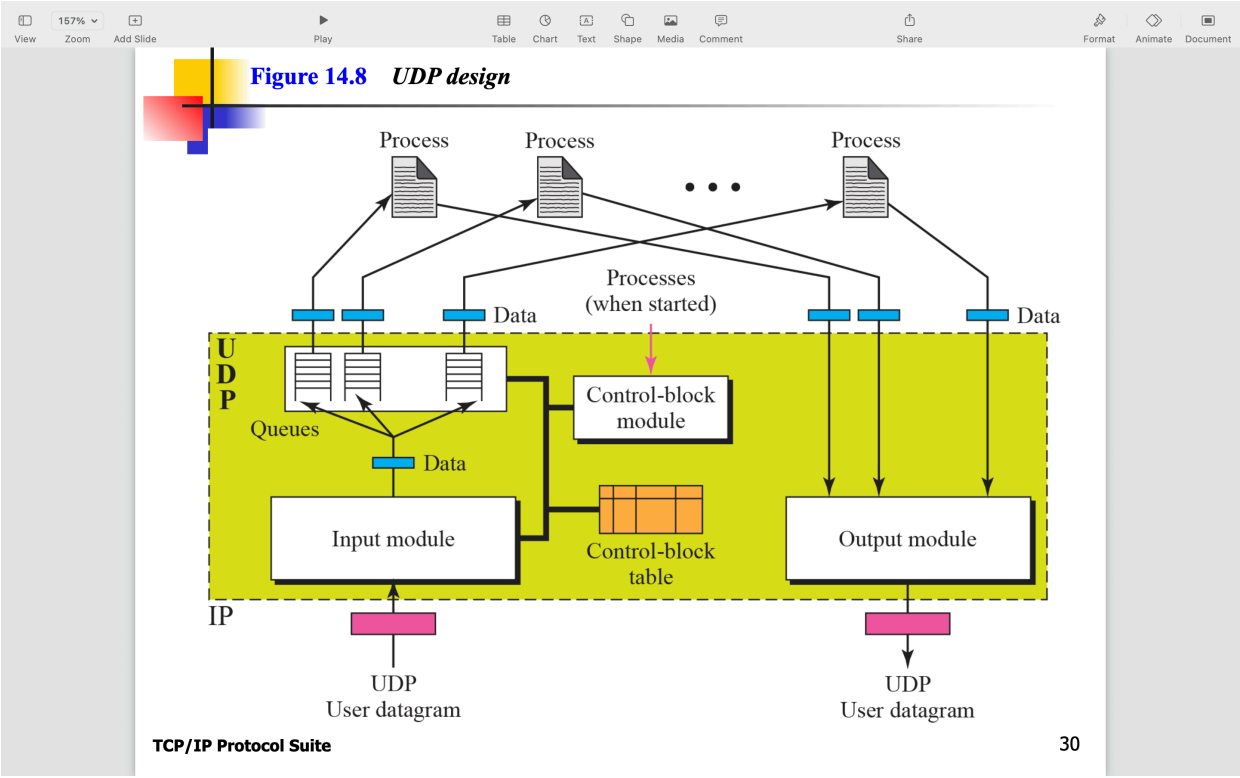
UDP Package

UDP software can be explained as a **small package** with 5 main components:

1. **Control-Block Table**
2. **Input Queues**

3. **Control-Block Module**
4. **Input Module**
5. **Output Module**

These work together to **send and receive UDP datagrams**.



1 Control-Block Table

- Keeps track of **open ports**.
- Each entry has 4 fields:
  - **State**: FREE or IN-USE
  - **Process ID** (which program is using it)
  - **Port number**
  - **Queue number** (for storing incoming data).
- Example: If port 52,010 is in use by process ID 2,345 → entry shows IN-USE.

Input Queues

- Each process has its own **input queue**.
- Stores incoming user datagrams (messages).
- **No output queues** in this design (sending is done directly).

Control-Block Module

- Manages the **control-block table**.
- When a process starts and asks for a port number:
  - OS gives a port number (well-known for servers, temporary/ephemeral for clients).
  - Control-block module creates an entry in the table (state = IN-USE, process ID, port number).
- If table is full → (book doesn't define strategy, but normally some entry is replaced).

Input Module

- Handles incoming UDP packets.
- Steps:
  1. Check control-block table for the **destination port number**.
  2. If found →
     - If no queue yet → allocate a queue.
     - Put the data in the queue.
  3. If not found →
     - Discard the datagram.
     - Ask ICMP to send **"port unreachable"** message back to sender.

Output Module

- Handles outgoing data.
- Steps:
  1. Create a **user datagram**.
  2. Send it to IP layer.

- Very simple (no queues or checks).

Examples (Exam-Focused)

- **Example 14.8**: Datagram arrives for port 52,012 → entry exists, queue 38 already assigned → data goes to queue 38.
- **Example 14.9**: New process starts with port 52,014 → entry created in table (queue not yet allocated).
- **Example 14.10**: Datagram arrives for port 52,011 → entry exists but no queue → new queue (43) allocated.
- **Example 14.11**: Datagram arrives for port 52,222 → no entry in table → datagram discarded + ICMP unreachable message sent.

14.8) Datagram arrives for port 52,012

- Control-block table দেখে → port 52,012 এর জন্য আগে থেকেই entry আছে।
- ওই entry তে **queue number = 38** লেখা আছে।
- মানে, এই process এর incoming data রাখার জন্য queue 38 ব্যবহার হচ্ছে।
- তাই datagram টা সরাসরি queue 38 এ চলে যাবে।

বোঝায়: যদি port + queue already exist করে → datagram সেই queue তে যাবে।

14.9)**New process starts with port 52,014**

- Process টি UDP service use করতে চাইছে → OS তাকে port 52,014 assign করে।
- Control-block module control-block table এ একটা নতুন entry তৈরি করে:
    - State = IN-USE
    - Process ID = ওই নতুন process এর
    - Port number = 52,014
    - Queue number = **(empty)** → কারণ এখনো কোনো datagram আসেনি।

বোঝায়: নতুন process শুরু হলে port table এ entry তৈরি হয়, কিন্তু datagram না আসা পর্যন্ত queue assign হয় না।

14.10) Datagram arrives for port 52,011

- Table এ port 52,011 এর জন্য entry আছে।
- কিন্তু সেখানে এখনো queue assign হয়নি (queue field = empty)।

- তাই Input module নতুন একটা queue allocate করে, ধরো queue number = 43।
- এখন থেকে port 52,011 → queue 43 ব্যবহার করবে।
- এই datagram সরাসরি queue 43 এ রাখা হবে।

বোঝায়: যদি port table এ entry থাকে কিন্তু queue না থাকে → নতুন queue allocate করা হবে।

14.11) Datagram arrives for port 52,222

- Control-block table দেখা হলো → port 52,222 এর কোনো entry নেই।
- মানে কোনো process এই port ব্যবহার করছে না।
- তাই datagram **discard** করা হবে (ফেলে দেওয়া হবে)।
- সাথে সাথে ICMP কে বলা হবে → sender এর কাছে একটা **"port unreachable"** error পাঠাও।

বোঝায়: যদি port table এ entry না থাকে → datagram ফেলা হবে + sender কে error জানানো হবে।

- **14.8**: Port আছে + Queue আছে → সরাসরি queue তে।
- **14.9**: নতুন Process শুরু → entry তৈরি (queue empty)।
- **14.10**: Port আছে কিন্তু queue নাই → নতুন queue বানাও।
- **14.11**: Port নেই → discard + ICMP unreachable।