

Chapter 14

❖ UDP কী?

UDP (User Datagram Protocol) হলো ইন্টারনেট প্রোটোকলের (IP) ওপর কাজ করা একটি দ্রুত কিন্তু অ-নির্ভরযোগ্য ডাটা ট্রান্সমিশন পদ্ধতি। এটি **connectionless**, অর্থাৎ ডাটা পাঠানোর আগে কোনো স্থায়ী সংযোগ তৈরি করতে হয় না।

❖ UDP কীভাবে কাজ করে?

UDP ডাটাকে ছোট ছোট অংশে (packet বা datagram) ভাগ করে পাঠায়।
প্রতিটি datagram-এ থাকে:

- উৎস (source) ও গন্তব্য (destination) পোর্ট নম্বর,
- ডাটার দৈর্ঘ্য,
- ও একটি সাধারণ error-check (checksum)।

UDP প্যাকেট পাঠিয়ে দেয়, কিন্তু সেটি গন্তব্যে পৌঁছেছে কিনা তা যাচাই করে না।

❖ UDP-এর বৈশিষ্ট্য

- **Connectionless** — কোনো handshake বা সংযোগ তৈরি হয় না।
- **Fast transmission** — TCP-এর চেয়ে দ্রুত।
- **No delivery guarantee** — প্যাকেট হারিয়ে যেতে পারে।
- **Lightweight** — কম overhead থাকে।

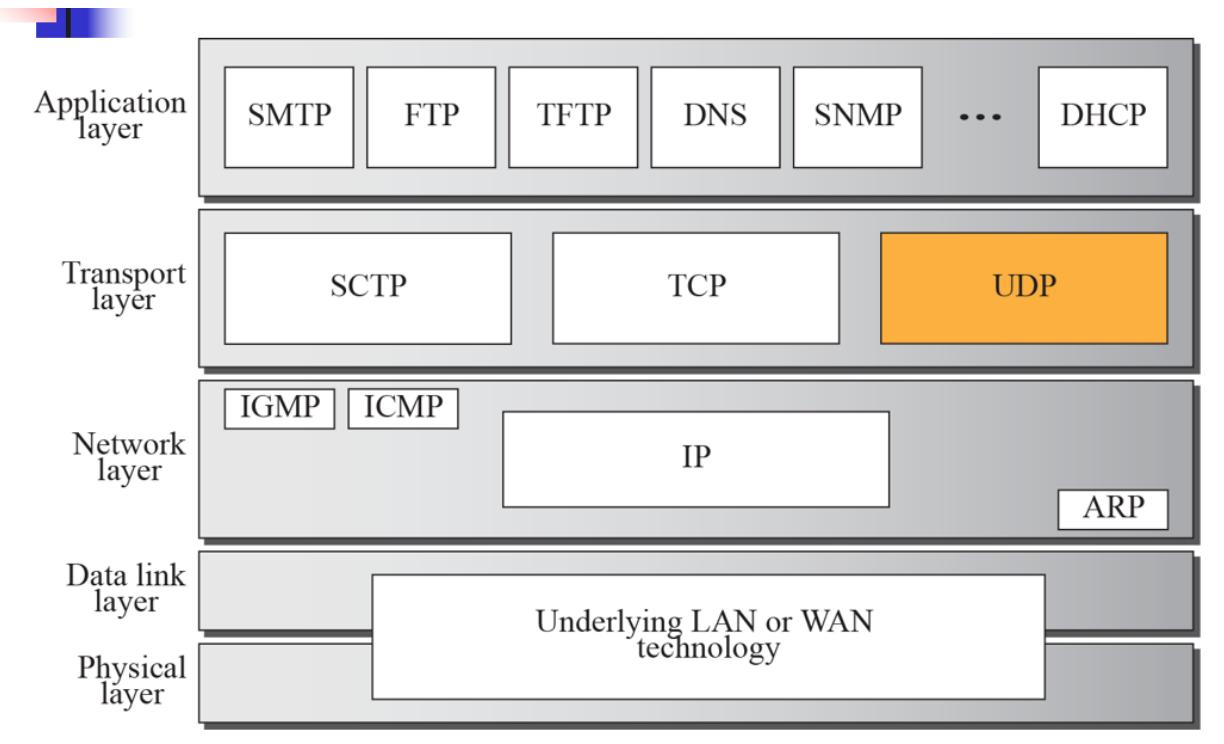
❖ UDP ব্যবহারের ক্ষেত্র

UDP সাধারণত সেইসব ক্ষেত্রে ব্যবহৃত হয় যেখানে গতি (speed) নির্ভরযোগ্যতার (reliability) চেয়ে বেশি গুরুত্বপূর্ণ, যেমনঃ

- লাইভ ভিডিও / অডিও স্ট্রিমিং
- অনলাইন গেমিং
- DNS (Domain Name System) কোয়েরি
- VoIP (Voice over IP)

❖ উদাহরণ

যেমন, আপনি যদি অনলাইন ভিডিও দেখেন (যেমন YouTube Live), তখন UDP ব্যবহার হতে পারে কারণ এতে গতি দরকার, কিন্তু মাঝে মাঝে একটি প্যাকেট হারালে বড় সমস্যা হয় না।



🌐 UDP-এর অবস্থান ও ভূমিকা (from the figure)

UDP বা User Datagram Protocol

ট্রেনে **Transport layer**-এ অবস্থান করে।

এটি **Application layer** (যেমন: DNS, DHCP, TFTP ইত্যাদি প্রোগ্রাম) এবং **Network layer** (যেখানে IP কাজ করে) — এই দুই স্তরের **মধ্যবর্তী সেতু** হিসেবে কাজ করে।

❖ চিত্র ব্যাখ্যা (Layer অনুযায়ী)

স্তর (Layer) উদাহরণ প্রোটোকল কাজ

Application Layer	SMTP, FTP, TFTP, DNS, SNMP, DHCP	ব্যবহারকারীর অ্যাপ্লিকেশনগুলোর কাজ যেমন ইমেইল, ফাইল ট্রান্সফার, নাম রেজোলিউশন ইত্যাদি সম্পন্ন করে।
Transport Layer	TCP, UDP, SCTP	ডেটা পরিবহনের দায়িত্বে থাকে। UDP এখানে connectionless communication সরবরাহ করে।
Network Layer	IP, ICMP, IGMP, ARP	ডেটা প্যাকেটকে এক ডিভাইস থেকে অন্য ডিভাইসে রাউট করে।
Data Link Layer	LAN বা WAN প্রযুক্তি	ফিজিক্যাল মিডিয়ামের মাধ্যমে ডেটা ট্রান্সফার নিশ্চিত করে।

স্তর (Layer)	উদাহরণ প্রোটোকল	কাজ
Physical Layer	ক্যাবল, সিগন্যাল, হার্ডওয়্যার	ডেটাকে আসল বৈদ্যুতিক বা অপটিক্যাল সিগন্যাল আকারে প্রেরণ করে।

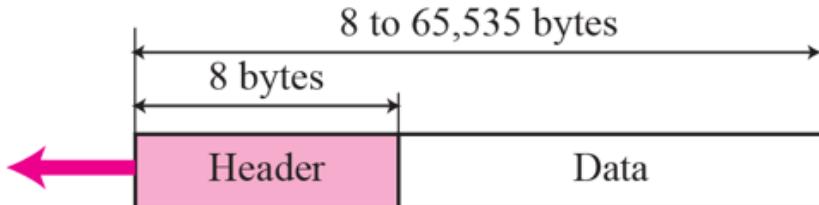
◊ UDP-এর ভূমিকা বিস্তারিতভাবে

UDP Transport Layer-এ কাজ করে এবং Application Layer-এর প্রোগ্রাম (যেমন: DNS, DHCP, SNMP ইত্যাদি) কে IP Network Layer-এর সাথে যুক্ত করে। এটি ডেটা পাঠানোর আগে কোনো সংযোগ তৈরি করে না, বরং **ডেটা প্যাকেট (Datagram)** আকারে পাঠিয়ে দেয়।

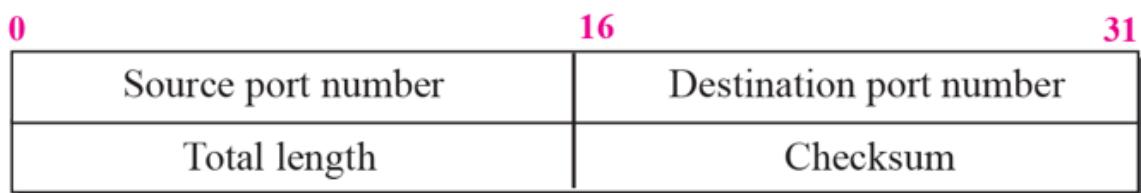
◊ সহজ ভাষায় সারাংশ

- UDP অবস্থান করে Transport Layer-এ।
- এটি Application Layer ও IP Layer-এর মধ্যে মধ্যস্থতাকারী।
- UDP-এর মূল উদ্দেশ্য হলো: দ্রুত, কম-ওভারহেডে ডেটা পাঠানো — যদিও এটি 100% নির্ভরযোগ্য নয়।

Figure 14.2 User datagram format



a. UDP user datagram



b. Header format

☞ UDP User Datagram Structure (চিত্র অনুযায়ী)

UDP প্যাকেট দুটি অংশে ভাগ করা থাকে:

1. **Header (হেডার)** — ৮ বাইট (Fixed size)
2. **Data (ডেটা অংশ)** — ০ থেকে ৬৫,৫৩৫ বাইট পর্যন্ত হতে পারে

☞ অর্থাৎ একটি UDP ডাটাগ্রামের মোট দৈর্ঘ্য ৮ থেকে ৬৫,৫৩৫ বাইট পর্যন্ত হতে পারে।

☞ UDP Header Format (চিত্র অনুযায়ী)

UDP Header সর্বদা ৮ বাইট হয় এবং এটি চারটি ক্ষেত্র (field) নিয়ে গঠিত, প্রতিটি ২ বাইট (১৬-বিট) আকারের।

ক্ষেত্র	আকার	বর্ণনা
① Source Port Number (2 bytes)	১৬-বিট	প্রেরকের পোর্ট নম্বর (Sender's port)। এটি ঐচ্ছিক, অর্থাৎ কিছু ক্ষেত্রে ০ হতে পারে।

ক্ষেত্র	আকার	বর্ণনা
② Destination Port Number (2 bytes)	১৬-বিট	প্রাপকের পোর্ট নম্বর (Receiver's port), যা নির্ধারণ করে ডেটা কোন অ্যাপ্লিকেশনে যাবে।
③ Total Length (2 bytes)	১৬-বিট	পুরো UDP প্যাকেটের দৈর্ঘ্য (Header + Data)।
④ Checksum (2 bytes)	১৬-বিট	ক্রটি শনাক্তকরণের জন্য ব্যবহৃত একটি যাচাইকরণ সংখ্যা (Error detection)।

② Header-এর মোট দৈর্ঘ্য:

- $4 \text{ fields} \times 2 \text{ bytes} = 8 \text{ bytes}$

③ UDP Datagram Example (সহজভাবে)

ধরা যাক, তুমি DNS রিকোয়েস্ট পাঠাচ্ছো UDP দিয়ে:

- **Source port:** 56789
- **Destination port:** 53
- **Total length:** 36 bytes
- **Checksum:** একটি স্বয়ংক্রিয়ভাবে গণিত মান

UDP Header (8 bytes) + DNS Data (28 bytes) = 36 bytes total datagram।

◇ মূল বৈশিষ্ট্য সারাংশ

- UDP header খুব ছোট \Rightarrow তাই একে **lightweight** বলা হয়।
- এতে কোনো sequence number বা acknowledgment নেই \Rightarrow তাই **fast** কিন্তু **unreliable**।

Example 14.1

The following is a dump of a UDP header in hexadecimal format.

CB84000D001C001C

- a. What is the source port number?
- b. What is the destination port number?
- c. What is the total length of the user datagram?
- d. What is the length of the data?
- e. Is the packet directed from a client to a server or vice versa?
- f. What is the client process?

◇ দেওয়া UDP হেডার (Hexadecimal):

CB84 00D0 01C0 001C

UDP হেডারের চারটি ১৬-বিট (2-byte) ফিল্ড আছে:

ফিল্ড হেক্সাডেসিমাল মান বাইট সংখ্যা

Source Port	CB84	2 bytes
Destination Port	00D0	2 bytes
Total Length	01C0	2 bytes
Checksum	001C	2 bytes

(a) Source port number

Hex CB84

→ Decimal = 0xCB84 =

(C = 12, B = 11, 8 = 8, 4 = 4)

= (12×4096) + (11×256) + (8×16) + 4

= 49152 + 2816 + 128 + 4 = **52000**

Source Port = 52000

(b) Destination port number

Hex 00D0

$$\rightarrow \text{Decimal} = 0x00D0 = (0 \times 4096) + (0 \times 256) + (13 \times 16) + 0 = 208$$

Destination Port = 208

(c) Total length of the UDP datagram

Hex 01C0

$$\rightarrow \text{Decimal} = 0x01C0 = (1 \times 256) + (12 \times 16) + 0 = 256 + 192 = \mathbf{448 \text{ bytes}}$$

Total Length = 448 bytes

(d) Length of the data

UDP header = 8 bytes

$$\begin{aligned} \text{So, Data length} &= \text{Total length} - \text{Header length} \\ &= 448 - 8 = \mathbf{440 \text{ bytes}} \end{aligned}$$

Data length = 440 bytes

(e) Packet direction

Usually, **client → server** communication uses **well-known destination ports** (small numbers like 53, 80, 208, etc.) and **random high-numbered source ports** (like 52000).

So, since:

- Source port = 52000 (high number → client)
- Destination port = 208 (low number → server)

Packet is from client → server

(f) Client process

Since the source port (52000) is a **temporary / ephemeral port**,
এটি ক্লায়েন্ট অ্যাপ্লিকেশন দ্বারা ডাইনামিকভাবে তৈরি করা হয়।

Client process: an ephemeral client (random high port)

১. সারসংক্ষেপ টেবিল

প্রশ্ন

উত্তর

- | | |
|----------------------|-----------------------------------|
| (a) Source Port | 52000 |
| (b) Destination Port | 208 |
| (c) Total Length | 448 bytes |
| (d) Data Length | 440 bytes |
| (e) Direction | Client → Server |
| (f) Client Process | Ephemeral (temporary) client port |

UDP Services (UDP-এর প্রধান সেবা)

❖ 1. Process-to-Process Communication

UDP সরাসরি একটি প্রক্রিয়া (process) থেকে অন্য প্রক্রিয়ায় ডেটা প্রেরণ করে।

অর্থাৎ — Application layer process → Transport layer (UDP) → Network layer (IP)

প্রতিটি প্রক্রিয়াকে port number দ্বারা চিহ্নিত করা হয়।

- **Source port number:** প্রেরক প্রক্রিয়া নির্দেশ করে।
- **Destination port number:** গ্রাহক প্রক্রিয়া নির্দেশ করে।

→ এর ফলে, একই ডিভাইসে একাধিক অ্যাপ্লিকেশন একই সময়ে ডেটা পাঠাতে বা পেতে পারে।

❖ 2. Connectionless Service

UDP হলো connectionless protocol, অর্থাৎ কোনো স্থায়ী সংযোগ (connection establishment) তৈরি করে না।

প্রত্যেকটি প্যাকেট (datagram) আলাদাভাবে পাঠানো হয় —
প্রেরক ও গ্রাহক একে অপরকে আগেভাগে চিনতে হয় না।

→ এতে communication খুব দ্রুত হয়, তবে reliability কম।

❖ 3. Unreliable Delivery

UDP ডেটা পৌঁছেছে কিনা ঘাটাই করে না, acknowledgment পাঠায় না।

যদি কোনো প্যাকেট হারিয়ে যায় বা নষ্ট হয়, UDP সেটি পুনরায় পাঠায় না।

→ এটি speed-এর বিনিময়ে reliability ত্যাগ করে।

◊ 4. Low Overhead (Lightweight)

UDP হেডার ছোট (মাত্র 8 bytes)।

এতে কোনো sequence number, acknowledgment, congestion control ইত্যাদি নেই।

তাই এটি দ্রুত এবং কম রিসোর্স ব্যবহার করে।

◊ 5. Multiplexing and Demultiplexing

UDP একই সময়ে অনেক process থেকে প্যাকেট পাঠানো ও গ্রহণ করা সক্ষম।

- **Multiplexing:** একাধিক অ্যাপ্লিকেশন থেকে ডেটা পাঠানো।
- **Demultiplexing:** প্রাপ্তি ডেটা নির্দিষ্ট অ্যাপ্লিকেশনে পৌঁছে দেওয়া (port number দেখে)।

→ উদাহরণ: একই কম্পিউটারে DNS, TFTP, SNMP — সব UDP ব্যবহার করতে পারে।

◊ 6. Checksum for Error Detection

UDP ডেটার integrity বজায় রাখার জন্য একটি checksum ফিল্ড ব্যবহার করে।

এটি ডেটা বিকৃত হয়েছে কিনা তা শনাক্ত করতে সাহায্য করে (যদিও পুনরুদ্ধার করে না)।

◊ 7. Support for Simple Query-Response Protocols

UDP খুব উপযোগী simple request-response system-এর জন্য, যেমনঃ

- DNS (Domain Name System)
- DHCP (Dynamic Host Configuration Protocol)
- SNMP (Simple Network Management Protocol)

→ এগুলোতে গতি বেশি গুরুত্বপূর্ণ, তাই UDP ব্যবহার হয়।

⇒ UDP Services Summary Table

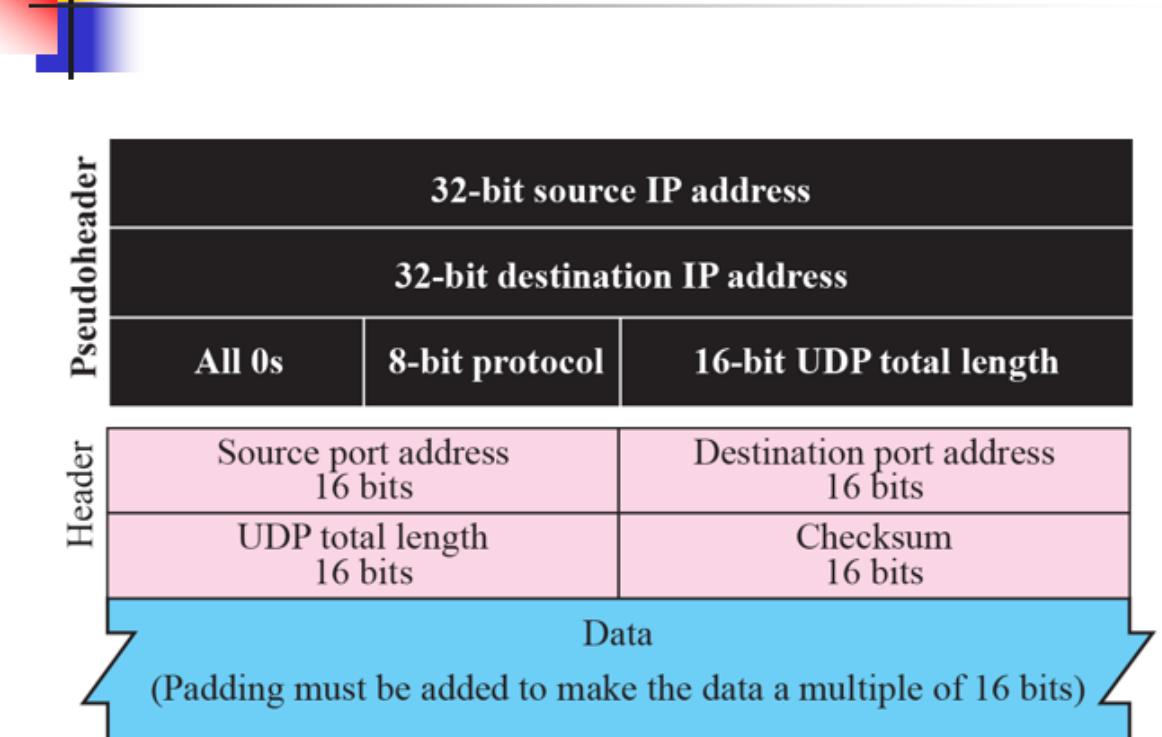
UDP Service	Description
Process-to-Process Communication	Port number ব্যবহার করে এক প্রক্রিয়া থেকে অন্য প্রক্রিয়ায় ডেটা পাঠানো।
Connectionless	কোনো সংযোগ তৈরি না করেই ডেটা পাঠানো।
Unreliable	প্যাকেট হারালে পুনরায় পাঠানো হয় না।

UDP Service	Description
Low Overhead	ছোট হেডার ও কম রিসোর্স ব্যবহার।
Multiplexing/Demultiplexing	একাধিক অ্যাপ্লিকেশনের ডেটা আলাদা করা ও পাঠানো।
Error Detection	Checksum দ্বারা ত্রুটি শনাক্ত।
Support for Simple Protocols	DNS, DHCP, SNMP ইত্যাদির মতো দ্রুত সেবা।

💡 UDP ব্যবহৃত প্রধান প্রোটোকলগুলো

- **DNS (Domain Name System)**
- **DHCP (Dynamic Host Configuration Protocol)**
- **TFTP (Trivial File Transfer Protocol)**
- **SNMP (Simple Network Management Protocol)**
- **RIP (Routing Information Protocol)**
- **VoIP (Voice over IP)**
- **Online Gaming & Live Streaming**

Figure 14.3 Pseudoheader for checksum calculation



Pseudoheader for UDP Checksum Calculation

UDP-র **Checksum** কেবল UDP হেডার ও ডেটা-এর উপর ভিত্তি করে হিসাব করা হয় না, বরং এর সাথে যুক্ত হয় একটি অতিরিক্ত অংশ — যাকে বলা হয় **Pseudoheader**।

Pseudoheader আসলে IP layer-এর কিছু তথ্যের কপি, যা UDP প্যাকেট পাঠানোর সময় Checksum হিসাবের জন্য অস্থায়ীভাবে যোগ করা হয়।
এই Pseudoheader ট্রান্সমিশনে পাঠানো হয় না — শুধু গণনার সময় ব্যবহৃত হয়।

Pseudoheader-এর গঠন (Structure)

Pseudoheader নিচের ফিল্ডগুলো নিয়ে গঠিত ছুঁ

ফিল্ড	আকার (বিট)	ব্যাখ্যা
Source IP address	32 bits	প্রেরকের IP ঠিকানা
Destination IP address	32 bits	প্রাপকের IP ঠিকানা
All Os (Reserved)	8 bits	সবগুলো শূন্য; ভবিষ্যতে ব্যবহারের জন্য সংরক্ষিত
Protocol	8 bits	প্রোটোকল নম্বর (UDP = 17)
UDP total length	16 bits	UDP হেডার + ডেটার মোট দৈর্ঘ্য

Checksum Calculation Includes

Checksum গণনা করার সময় UDP নিম্নলিখিত অংশগুলিকে একত্রে যোগ করে:

1. **Pseudoheader**
2. **UDP Header**
3. **UDP Data (Payload)**

তারপর:

- সবগুলো 16-bit word যোগ করা হয়
- যদি overflow হয়, তা আবার নিচে carry করে যোগ করা হয়
- শেষে **one's complement** নেওয়া হয়

ফলাফলটি হয় **Checksum Value (16-bit)**

→ যা UDP হেডারের “Checksum” ফিল্ডে রাখা হয়।

¶ Receiver Side-এ যাচাই (Verification)

Receiver একই প্রক্রিয়ায় পুনরায় Checksum গণনা করে।

যদি প্রাপ্ত Checksum + সব data word যোগ করলে **সব বিট 1** (1111111111111111) হয়,

তাহলে ডেটা সঠিক

অন্যথায় ডেটা ত্রুটিপূর্ণ

💡 Padding সম্পর্কে (চিত্রের নিচে)

UDP ডেটা অংশের দৈর্ঘ্য যদি 16-bit-এর গুণিতক না হয়,

তাহলে শেষে কিছু padding bits (0s) যোগ করা হয়,

যাতে Checksum গণনার জন্য data একটি পূর্ণ 16-bit সেটে পরিণত হয়।

▣ সংক্ষেপে সারাংশ

UDP Checksum = One's complement sum of (Pseudoheader + UDP Header + Data)

এটি ডেটা সঠিকভাবে ট্রান্সমিট হয়েছে কিনা তা যাচাই করতে ব্যবহৃত হয়।

💡 Checksum ব্যবহারের কারণ

বৈশিষ্ট্য

ব্যাখ্যা

ডেটা অখণ্ডতা (Data Integrity) প্রেরক ও গ্রাহক একইভাবে ডেটা যাচাই করতে পারে।

ত্রুটি শনাক্ত (Error Detection) প্যাকেট বিকৃত বা হারিয়ে গেলে তা শনাক্ত করা যায়।

IPv6-এ বাধ্যতামূলক

IPv6 প্রোটোকলে UDP Checksum অবশ্যই থাকতে হয়।

Figure 14.4 Checksum calculation for a simple UDP user datagram



153.18.8.105				10011001 00010010	→ 153.18
171.2.14.10				00001000 01101001	→ 8.105
All 0s	17	15		10101011 00000010	→ 171.2
1087		13		00001110 00001010	→ 14.10
15		All 0s		00000000 00010001	→ 0 and 17
T	E	S	T	00000000 00001111	→ 15
I	N	G	Pad	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

Figure 14.4 — Checksum calculation for a simple UDP user datagram

এটি একটি ছেটি UDP ডাটাগ্রামের উদাহরণ, যাতে ৭ বাইট ডেটা (T, E, S, T, I, N, G) রয়েছে। ডেটা ৭ বাইট হওয়ায় শেষে একটি padding byte (0) যোগ করা হয়েছে, যাতে checksum গণনা ১৬-বিট ওয়ার্ডে করা যায়।

Checksum গণনায় ব্যবহৃত অংশগুলো

UDP checksum গণনার সময় তিনটি অংশ ব্যবহার করা হয়:

① Pseudoheader (কালো অংশ)

এটি IP লেয়ার থেকে নেওয়া তথ্য — UDP প্রোটোকলের সাথে যুক্ত হয় কেবল checksum গণনার জন্য।

ফিল্ড মান

Source IP 153.18.8.105

Destination IP 171.2.14.10

Reserved (All 0s) 0

Protocol 17 (UDP)

UDP Length 15 bytes

② UDP Header (গোলাপি অংশ)

ফিল্ড মান

Source Port 1087

Destination Port 13

UDP Length 15

Checksum 0 (calculated later)

③ UDP Data (নীল অংশ)

Data = “T E S T I N G”

→ 7 bytes

Padding যোগ করার পর হয়:

T E S T I N G 0 (মোট 8 bytes বা 4 word)

Checksum গণনার ধাপ

Step 1: সব 16-bit word একত্রে লেখা

Pseudoheader, Header, এবং Data মিলে নিচের 16-bit word গঠন হয়:

10011001	00010010	→ 153.18
00001000	01101001	→ 8.105
10101011	00000010	→ 171.2

00001110	00001010	→ 14.10
00000000	00000000	→ All 0s
00010001	00001111	→ Protocol (17) + Length (15)
00000100	00111111	→ Source port (1087)
00000000	00001101	→ Destination port (13)
00000000	00001111	→ UDP length (15)
00000000	00000000	→ Checksum (initially 0)
01010100	01000101	→ T, E
01010011	01010100	→ S, T
01001001	01001110	→ I, N
01000111	00000000	→ G, Padding 0

Step 2: সবগুলো 16-bit word যোগ করা (binary addition)

এই যোগফলের ফলাফল চিত্রে দেখা যায়:

10010110 11101111

(যদি carry হয়, সেটি আবার যোগ করা হয়)

Step 3: One's complement নেওয়া

এখন সব বিট উল্টিয়ে ($1 \rightarrow 0, 0 \rightarrow 1$) নিলে চূড়ান্ত Checksum পাওয়া যায় ↗

01101001 00010100

→ Checksum = 0x6914 (Hexadecimal)

Checksum যাচাই (Receiver side)

Receiver একইভাবে checksum গণনা করে এবং প্রাপ্ত checksum-এর সাথে যোগ করে:

- যদি যোগফল সব 1 হয় (1111111111111111) → ডেটা সঠিক
 - না হলে → ডেটা ক্রটিপূর্ণ
-

💡 গুরুত্বপূর্ণ বিষয়

বিষয়

ব্যাখ্যা

Padding ডেটা যদি odd সংখ্যা হয়, তাহলে এক বাইট 0 যোগ করা হয়।

বিষয়

ব্যাখ্যা

Pseudoheader কেবল checksum গণনার জন্য ব্যবহৃত হয়, পাঠানো হয় না।

Checksum ডেটা অখণ্ডতা যাচাই করে, কিন্তু ত্রুটি সংশোধন করে না।

সারসংক্ষেপ

এই Figure 14.4 UDP checksum প্রক্রিয়াটি প্রদর্শন করছে—

যেখানে Pseudoheader + UDP header + Data (+ padding) একত্রে যোগ করে One's complement নেওয়া হয়।

এই মান UDP হেডারের Checksum ফিল্ডে বসে,

এবং এটি ডেটা অক্ষত আছে কিনা যাচাই করতে ব্যবহৃত হয়।

Example 14.3

What value is sent for the checksum in one of the following hypothetical situations?

- a. The sender decides not to include the checksum.**
- b. The sender decides to include the checksum, but the value of the sum is all 1s.**
- c. The sender decides to include the checksum, but the value of the sum is all 0s.**

Example 14.3 — UDP Checksum Field Values (Special Cases)

প্রশ্নটি তিনটি কানুনিক (hypothetical) পরিস্থিতি দেখাচ্ছে যেখানে checksum ফিল্ডে কী মান পাঠানো হবে তা নির্ধারণ করতে হবে।

- ◊ (a) The sender decides not to include the checksum.

ব্যাখ্যা:

UDP-তে checksum ট্রিচিক (optional) — বিশেষত IPv4-এর ক্ষেত্রে।

যদি প্রেরক (sender) checksum ব্যবহার না করার সিদ্ধান্ত নেয়,

তাহলে এটি নির্দেশ করার জন্য checksum ফিল্ডে সরগুলো bit = 0 সেট করা হয়।

Answer:

Checksum field = **all 0s (0000 0000 0000 0000)**

- অর্থাৎ — “checksum গণনা করা হয়নি” বোঝাতে 0 মান পাঠানো হয়।
- (b) The sender decides to include the checksum, but the value of the sum is all 1s.

ব্যাখ্যা:

ধরা যাক checksum গণনার সময় সব 16-bit word যোগ করলে সব **বিট = 1 (1111 1111 1111 1111)** হয়।

তাহলে one's complement নিলে ফলাফল হবে সব 0s (0000 0000 0000 0000)।

কিন্তু checksum ফিল্ডে সব 0 থাকলে এটি case (a)-এর সাথে বিভ্রান্তি সৃষ্টি করবে —
কারণ “0” মান মানে হতে পারে “checksum ব্যবহার করা হয়নি”।

তাই, এমন বিভ্রান্তি এড়াতে sender আবার complement নেয়।

অর্থাৎ:

দ্বিতীয়বার complement → ফলে সব 1s হয়ে যায়।

Answer:

Checksum field = **all 1s (1111 1111 1111 1111)**

- (c) The sender decides to include the checksum, but the value of the sum is all 0s.

ব্যাখ্যা:

এই অবস্থা বাস্তবে ঘটে না।

কারণ checksum গণনার সময় pseudoheader, header, এবং data মিলিয়ে কিছু না কিছু nonzero
মান থাকে।

সব মান 0 হওয়া মানে প্রতিটি অংশ 0 — যা অসম্ভব।

Answer:

This situation **never occurs** in reality.

১৩. সারসংক্ষেপ টেবিল

পরিস্থিতি	ব্যাখ্যা	পাঠানো মান (checksum field)
(a) Checksum অন্তর্ভুক্ত নয়	checksum বাদ দিলে	সব 0s (0000)
(b) Checksum অন্তর্ভুক্ত, sum = all 1s	বিপ্রাণ্তি এড়াতে complement নেওয়া হয়	সব 1s (FFFF)
(c) Checksum অন্তর্ভুক্ত, sum = all 0s	বাস্তবে অসম্ভব	ঘটে না

১৪. মনে রাখার বিষয়

- UDP Checksum **optional in IPv4, mandatory in IPv6**
- “All 0s” মানে “no checksum used”
- “All 1s” মানে checksum ব্যবহার হয়েছে, এবং মানটি double complement থেকে এসেছে

Figure 14.5 Encapsulation and decapsulation

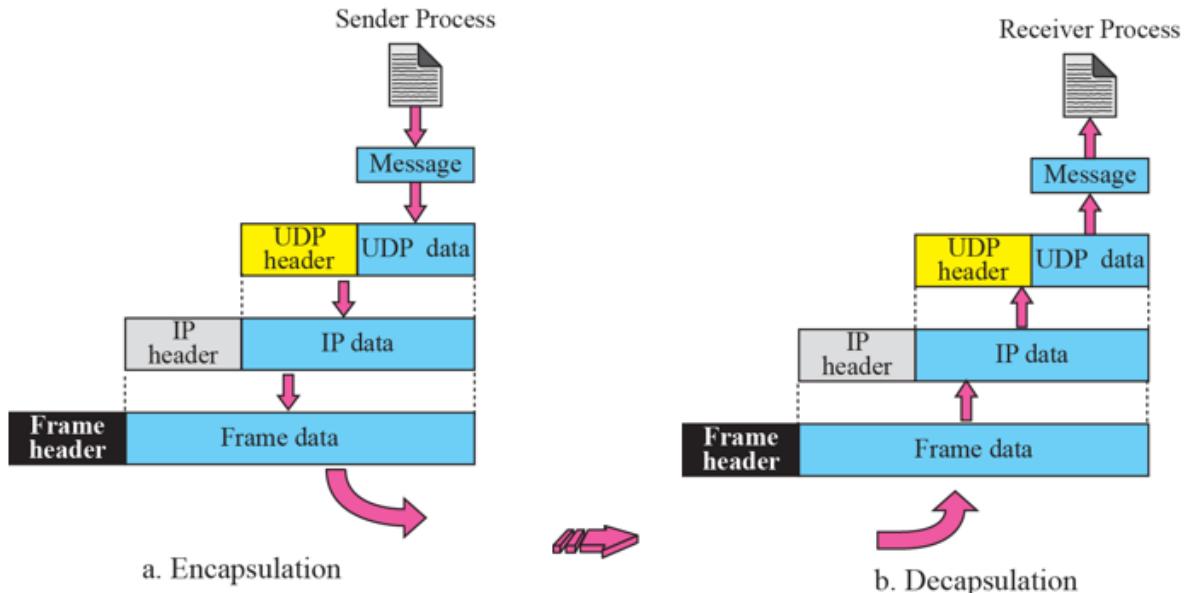


Figure 14.5 – Encapsulation and Decapsulation (UDP)

UDP হল Transport Layer-এর একটি প্রোটোকল।

এটি Application layer message কে গ্রহণ করে, তার সাথে নিজের UDP header যোগ করে IP এবং Data Link layer-এর মাধ্যমে প্রেরণ করে।

এই পুরো প্রক্রিয়াটি দুই দিকেই ঘটে:

- ① **Encapsulation (sender side)**
 - ② **Decapsulation (receiver side)**
-

❖ (a) Encapsulation (প্রেরকের দিক)

Encapsulation মানে হলো —

উপরের স্তরের (Application layer) ডেটাকে নিচের স্তরে পাঠানোর আগে প্রয়োজনীয় header যোগ করা।

Step-by-step ব্যাখ্যা:

ধাপ

কাজ

- ① Application layer থেকে একটি **message (data)** UDP-তে আসে।
- ② UDP প্রোটোকল message-এর আগে নিজের **UDP header** যোগ করে → এখন এটি হয় **UDP segment (datagram)**।
- ③ IP layer UDP datagram-এর আগে **IP header** যোগ করে → এখন এটি হয় **IP packet**।
- ④ Data link layer IP packet-এর আগে **Frame header** যোগ করে → এটি হয় **Frame**, যা ফিজিক্যাল নেটওয়ার্কে পাঠানো হয়।

→ সর্বশেষ গঠন:

[Frame Header] + [IP Header] + [UDP Header] + [Data]

এটাই ফাইনাল প্যাকেট, যা **Physical Layer** দিয়ে গন্তব্যে পাঠানো হয়।

◊ (b) Decapsulation (গ্রাহকের দিক)

Decapsulation মানে হলো —

নিচের স্তর থেকে আসা প্যাকেটকে ধাপে ধাপে খুলে আসল message পুনরুদ্ধার করা।

Step-by-step ব্যাখ্যা:

ধাপ

কাজ

- ① Frame নেটওয়ার্কের মাধ্যমে গ্রাহকের কাছে পৌঁছায়।
 - ② Data link layer Frame header সরিয়ে IP packet রেখে দেয়।
 - ③ IP layer IP header সরিয়ে UDP datagram রেখে দেয়।
 - ④ UDP layer UDP header সরিয়ে আসল **Application message** প্রক্রিয়াজাত করে অ্যাপ্লিকেশনে পাঠায়।
- শেষে প্রাপ্ত message = প্রেরকের পাঠানো আসল data।

◻ UDP Data Flow Summary

প্রক্রিয়া দিক

কাজ

Encapsulation Sender প্রতিটি স্তর তার নিজস্ব header যোগ করে।

Transmission Network প্যাকেট ফিজিক্যাল মিডিয়ামের মাধ্যমে প্রেরিত হয়।

Decapsulation Receiver প্রতিটি স্তর তার header সরিয়ে উপরের স্তরে data পাঠায়।

❑ সহজভাবে বোঝার মতো উদাহরণ

ধরা যাক, তুমি DNS query পাঠাও।

Sender side (Encapsulation):

DNS message → UDP header যোগ → IP header যোগ → Frame header যোগ → পাঠানো হয়।

Receiver side (Decapsulation):

Frame header সরানো → IP header সরানো → UDP header সরানো → DNS message পাওয়া।

୧ ସାରସଂକ୍ଷେପ

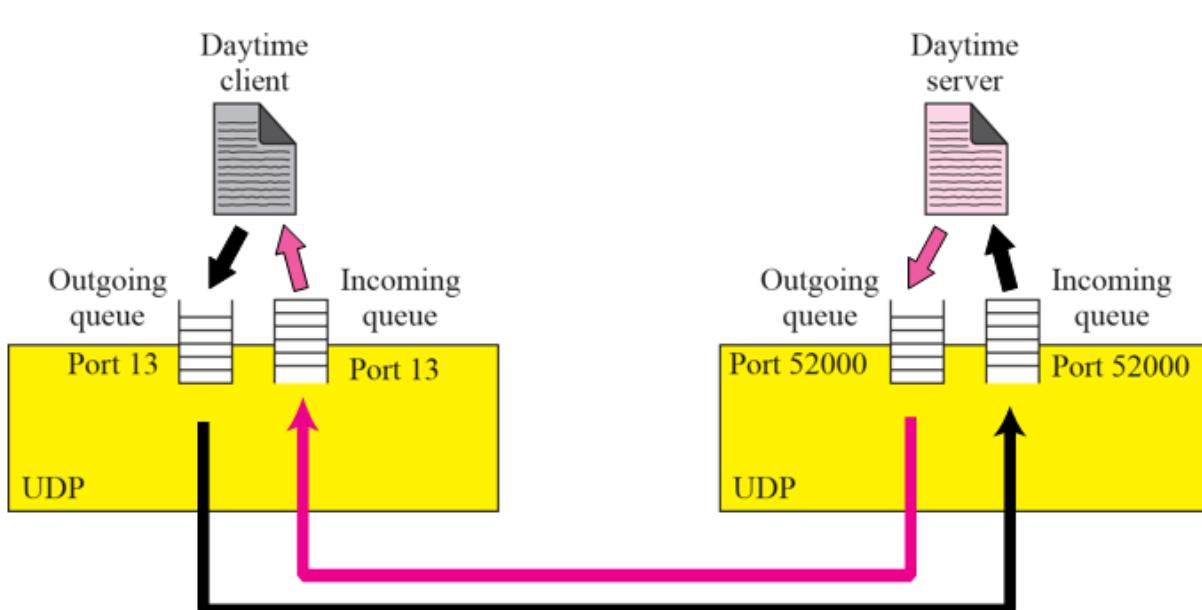
ধাপ	কাজের ধরন	প্রোটোকল
Application Layer	Message	TCP, UDP
Transport Layer	Header	IP
Network Layer	Header	ICMP, ARP
Data Link Layer	Header	Ethernet, Wi-Fi
Physical Layer	Transmission	SiGeNyanal, ParthaNo

মূল ধারণা

Encapsulation — ডেটাকে নেটওয়ার্কে পাঠানোর জন্য হেডার যুক্ত করা।

Decapsulation — ডেটা গ্রহণ করে হেডারগুলো সরিয়ে আসল বার্তা বের করা।

Figure 14.6 *Queues in UDP*



🌐 Figure 14.6 — Queues in UDP

UDP হলো একটি **connectionless transport layer protocol**,
তাই এটি কোনো স্থায়ী সংযোগ (connection) তৈরি না করে
শুধুমাত্র port number ব্যবহার করে যোগাযোগ স্থাপন করে।

UDP-তে প্রতিটি port-এর সাথে দুটি queue যুক্ত থাকে:

- ① **Outgoing Queue** — ডেটা পাঠানোর জন্য
 - ② **Incoming Queue** — ডেটা গ্রহণের জন্য
-

ଡাটা প্রক্রিয়া ব্যাখ্যা

❖ Left Side — Daytime Client

- এখানে client একটি UDP অ্যাপ্লিকেশন, যা server-এর কাছ থেকে সময় জানতে চায় (Daytime protocol)।
- ক্লায়েন্ট UDP-এর **Port 52000** ব্যবহার করে।
- UDP ডাটাগ্রাম তৈরি করে এবং সেটিকে server-এর well-known port 13-এ পাঠায়।

কাজের ধাপ:

1. Client অ্যাপ্লিকেশন message তৈরি করে → UDP layer-এ দেয়।
 2. UDP তা **Outgoing Queue (Port 52000)**-এ রাখে।
 3. এরপর প্যাকেটটি নেটওয়ার্কের মাধ্যমে server-এর দিকে যায়।
-

❖ Right Side — Daytime Server

- Server একটি প্রোগ্রাম যা **UDP port 13**-এ শুনছে (listening)।
 - UDP প্যাকেট পাওয়ার পর সেটি server-এর **Incoming Queue (Port 13)**-এ যায়।
 - Server সেই ডেটা পড়ে এবং উত্তর (response) তৈরি করে।
 - উত্তর পাঠানোর সময় UDP তা **Outgoing Queue (Port 13)** থেকে প্রেরণ করে
এবং সেটি যায় client-এর **Incoming Queue (Port 52000)**-এ।
-

▣ পুরো প্রক্রিয়ার সারাংশ

দিক	প্রক্রিয়া	Port Number	Queue
Client → Server	অনুরোধ পাঠানো	Source: 52000 → Destination: 13 Outgoing (52000) → Incoming (13)	
Server → Client	উত্তর পাঠানো	Source: 13 → Destination: 52000 Outgoing (13) → Incoming (52000)	

💡 UDP Queue-এর বৈশিষ্ট্য

বৈশিষ্ট্য	ব্যাখ্যা
Connectionless	কোনো স্থায়ী সংযোগ নেই, কেবল প্যাকেট পাঠানো হয়।
Queue-based Communication	প্রতিটি পোর্টের নিজস্ব incoming ও outgoing queue থাকে।
Port Number দ্বারা শনাক্তকরণ	প্রতিটি message সঠিক প্রক্রিয়াতে পাঠানোর জন্য port ব্যবহার হয়।
Temporary Client Port	ক্লায়েন্ট সাধারণত high-numbered ephemeral port (যেমন 52000) ব্যবহার করে।
Well-known Server Port	সার্ভার সাধারণত নির্দিষ্ট low-numbered port (যেমন 13) ব্যবহার করে।

⌚ সহজভাবে বোঝার মতো উদাহরণ

ধরা যাক তুমি একটি ক্লায়েন্ট প্রোগ্রাম চালাও যা UDP দিয়ে “Daytime server”-এর সময় জানতে চায়।

- ক্লায়েন্ট: UDP Port **52000** থেকে রিকোয়েস্ট পাঠায়।
- সার্ভার: UDP Port **13**-এ রিকোয়েস্ট পায় এবং সময় পাঠিয়ে দেয়।
- উত্তর ফিরে আসে ক্লায়েন্টের **Incoming Queue (Port 52000)**-এ।

এভাবে কোনো স্থায়ী সংযোগ ছাড়াই ডেটা আদান-প্রদান সম্পন্ন হয়।

▣ সারসংক্ষেপ

UDP প্রত্যেকটি পোর্টে দুইটি queue ব্যবহার করে —
Outgoing queue (পাঠানোর জন্য) এবং **Incoming queue** (গ্রহণের জন্য)।

Client সাধারণত একটি **অস্থায়ী** (ephemeral) পোর্ট ব্যবহার করে, আর Server ব্যবহার করে একটি **well-known fixed port**।

 Multiplexing এবং Demultiplexing কী?

- Multiplexing (একাধিক উৎস → এক গন্তব্যে পাঠানো)

Multiplexing হলো এমন একটি প্রক্রিয়া যেখানে একাধিক অ্যাপ্লিকেশন বা প্রক্রিয়া (process) একই সময়ে নেটওয়ার্কের মাধ্যমে ডেটা পাঠাতে পারে — এবং সেগুলো UDP বা TCP-এর মাধ্যমে IP layer-এ প্রেরিত হয়।

→ সহজভাবে বললে:

একাধিক অ্যাপ্লিকেশন → এক transport layer (UDP/TCP)

UDP প্রতিটি অ্যাপ্লিকেশনকে একটি **Source Port Number** দেয়,
যাতে IP layer বুর্বতে পারে কোন ডেটা কোন অ্যাপ্লিকেশন থেকে এসেছে।

- Demultiplexing (এক গন্তব্য → সঠিক অ্যাপ্লিকেশনে পৌঁছানো)

Demultiplexing হলো বিপরীত প্রক্রিয়া —

যেখানে প্রাপ্ত প্যাকেট (datagram) সঠিক গন্তব্য অ্যাপ্লিকেশনে পৌঁছে দেওয়া হয়।

→ সহজভাবে বললে:

এক transport layer → একাধিক অ্যাপ্লিকেশন (port অনুযায়ী ডেলিভারি)

❑ UDP Multiplexing এবং Demultiplexing-এর কাজের ধাপ

ধাপ	দিক	বর্ণনা
①	Multiplexing (Sender side)	বিভিন্ন Application Layer process (যেমন DNS, DHCP, SNMP) UDP ব্যবহার করে message তৈরি করে। UDP প্রতিটি message-এর জন্য একটি Source Port Number নির্ধারণ করে। এরপর UDP সব message IP layer-এ পাঠায়।
②	Demultiplexing (Receiver side)	UDP প্যাকেট পাওয়ার পর Destination Port Number দেখে বোর্বে কোন Application process-এ তা পাঠাতে হবে (যেমন Port 53 → DNS, Port 67 → DHCP)।

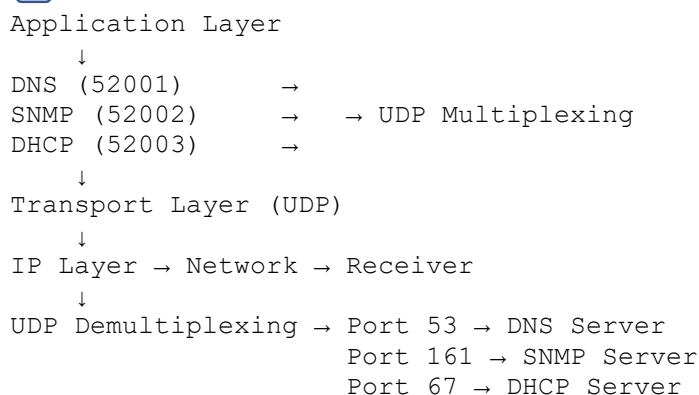
ଉদাহরণ দিয়ে বোঝানো

ধরা যাক, তোমার কম্পিউটার থেকে তিনটি অ্যাপ্লিকেশন একসাথে UDP ব্যবহার করছে কি

Application	Source Port	Destination Port	কাজ
DNS client	52001	53	ডোমেইন রেজল্যুশন
SNMP client	52002	161	নেটওয়ার্ক ম্যানেজমেন্ট
DHCP client	52003	67	IP কনফিগারেশন

- এখানে UDP তিনটি অ্যাপ্লিকেশনের ডেটা একত্রে IP layer-এ পাঠাচ্ছে (**Multiplexing**)।
 - সার্ভার পাশে যখন ডেটা পৌঁছায়, UDP Destination Port দেখে সেগুলোকে আলাদা করে সঠিক প্রোগ্রামে পাঠাচ্ছে (**Demultiplexing**)।
-

চিত্রে সংক্ষেপে



💡 UDP-তে Multiplexing/Demultiplexing-এর বৈশিষ্ট্য

বৈশিষ্ট্য

Connectionless UDP কোনো সংযোগ তৈরি না করেই ডেটা পাঠায়।

Port-based Identification প্রতিটি প্যাকেটের Source ও Destination Port নম্বর দিয়ে সনাক্ত হয়।

Parallel Communication একাধিক প্রক্রিয়া একসাথে কাজ করতে পারে।

Efficiency প্রক্রিয়াটি সহজ, দ্রুত এবং কম ওভারহেডযুক্ত।

৩ সংক্ষিপ্ত সারাংশ

ধারণা	সংজ্ঞা	দিক
Multiplexing	একাধিক উৎস অ্যাপ্লিকেশন থেকে ডেটা সংগ্রহ করে IP layer-এ পাঠানো Sender side	
Demultiplexing	IP layer থেকে প্রাপ্ত ডেটা সঠিক গন্তব্য অ্যাপ্লিকেশনে পৌঁছে দেওয়া	Receiver side

সহজভাবে মনে রাখো

Multiplexing → Combine → পাঠানো

Demultiplexing → Separate → পৌঁছে দেওয়া

“Although UDP meets almost none of the criteria we mentioned in Chapter 13 for a reliable transport-layer protocol, UDP is preferable for some applications...”

এই বাক্যটি আসলে ব্যাখ্যা করছে কেন UDP, TCP-এর মতো নির্ভরযোগ্য না হয়েও অনেক অ্যাপ্লিকেশনে বেশি উপযোগী।

UDP নির্ভরযোগ্য নয়, তবুও কেন এটি ব্যবহারযোগ্য

UDP (User Datagram Protocol) হলো একটি **connectionless** ও **unreliable** transport-layer protocol।

এর মানে UDP:

- কোনো সংযোগ স্থাপন করে না,
- acknowledgment পাঠায় না,
- packet loss বা retransmission হ্যান্ডেল করে না।

তবুও কিছু ক্ষেত্রে UDP TCP-এর চেয়ে ভালো বিকল্প।

কারণ — কিছু অ্যাপ্লিকেশন speed, simplicity, এবং low overhead-কে reliability-এর চেয়ে বেশি গুরুত্ব দেয়।

-
- ◊ Chapter 13 অনুযায়ী নির্ভরযোগ্যতার মানদণ্ড (reliability criteria)

TCP-র মতো নির্ভরযোগ্য প্রোটোকলের জন্য প্রয়োজন:

1. Connection establishment (সংযোগ তৈরি)
2. Flow control
3. Error control
4. Acknowledgment
5. Sequencing

UDP এদের কোনোটিই দেয় না।

তবুও এটি জনপ্রিয়, কারণ এতে এমন কিছু সুবিধা আছে যা নির্দিষ্ট অ্যাপ্লিকেশনের জন্য উপযুক্ত।

- ◇ কেন UDP কখনো TCP-এর চেয়ে ভালো

1 Speed is more important than reliability

UDP কোনো delay তৈরি করে না — কারণ connection setup, retransmission, বা acknowledgment নেই।

তাই এটি খুব দ্রুত কাজ করে।

→ উদাহরণ: লাইভ ভিডিও স্ট্রিমিং, অনলাইন গেমিং, VoIP।

2 Simplicity and low overhead

UDP-এর হেডার ছোট (8 bytes), এবং কোনো অতিরিক্ত কন্ট্রোল মেকানিজম নেই।

এর ফলে নেটওয়ার্কে কম ট্রাফিক ও কম বিলম্ব ঘটে।

→ এটি real-time data transmission-এর জন্য আদর্শ।

3 Some applications handle reliability themselves

কিছু অ্যাপ্লিকেশন নিজস্ব পর্যায়ে (application layer) error handling করে নেয়, UDP-র সাহায্য ছাড়াই।

→ যেমন DNS বা TFTP — তারা প্রয়োজনে নিজেরাই পুনরায় অনুরোধ পাঠায়।

4 Avoiding TCP's side effects

TCP-র কিছু বৈশিষ্ট্য (যেমন congestion control, retransmission delay) কিছু অ্যাপ্লিকেশনে অপ্রীতিকর side effect তৈরি করে।

UDP ব্যবহার করলে এই বিলম্ব এড়ানো যায়।

❖ একজন Application Designer-এর দৃষ্টিকোণ থেকে

একজন application designer সব সময় 100% reliability চায় না —
বরং performance এবং resource efficiency-এর মধ্যে একটি ভারসাম্য (compromise)
খোঁজেন।

UDP সেই ভারসাম্য প্রদান করে:

- কম বিলম্ব (low latency)
 - কম overhead
 - পর্যাপ্ত নির্ভরযোগ্যতা (যদি অ্যাপ্লিকেশন নিজে তা পরিচালনা করে)
-

💡 উদাহরণ

অ্যাপ্লিকেশন

কেন UDP উপযোগী

DNS (Port 53) খুব দ্রুত response দরকার, TCP handshake অপ্রয়োজনীয়

DHCP (Port 67/68) সংক্ষিপ্ত ও সহজ বার্তা পাঠানো

VoIP / Streaming কিছু packet হারালেও audio/video চলবে

Online Gaming গতি reliability-এর চেয়ে বেশি গুরুত্বপূর্ণ

▣ সংক্ষিপ্ত সারাংশ

বৈশিষ্ট্য

TCP

UDP

সংযোগ Connection-oriented Connectionless

নির্ভরযোগ্যতা Reliable Unreliable

গতি ধীর দ্রুত

হেডার সাইজ বড় (20 bytes) ছোট (8 bytes)

উপযুক্ত ক্ষেত্র ফাইল ট্রান্সফার, ইমেইল রিয়েল-টাইম অ্যাপ, গেমিং, DNS

মূল বক্তব্য

UDP নির্ভরযোগ্য নয়, কিন্তু কিছু অ্যাপ্লিকেশনের জন্য এটি অপ্টিমাম পছন্দ —
কারণ সেগুলিতে গতি (speed) এবং কম বিলম্ব (low latency) নির্ভরযোগ্যতার চেয়ে বেশি
গুরুত্বপূর্ণ।

“A client-server application such as DNS uses the services of UDP...”

কেন DNS-এর মতো ক্লায়েন্ট-সার্ভার অ্যাপ্লিকেশনগুলো UDP ব্যবহার করে TCP নয়।

DNS কেন UDP ব্যবহার করে

DNS (Domain Name System) হলো এমন একটি ক্লায়েন্ট-সার্ভার অ্যাপ্লিকেশন
যা কোনো ডোমেইন নামকে (যেমন www.google.com) একটি IP ঠিকানায় রূপান্তর করে।

এখানে যোগাযোগ হয় খুব ছোট আকারের query এবং response message দ্বারা —
যা একে অপরের মধ্যে দ্রুত পাঠানো দরকার।

- UDP ব্যবহারের মূল কারণ

① সংক্ষিপ্ত বার্তা (Short message)

DNS ক্লায়েন্ট সাধারণত মাত্র একটি ছোট রিকোয়েস্ট পাঠায় (যেমন www.example.com-এর IP কী?)

এবং সার্ভার একটি ছোট উত্তর পাঠায় (যেমন 192.0.2.1)।

এই দুটি বার্তা সহজেই একটি UDP datagram-এর মধ্যে ফিট হয়ে যায়।

→ তাই TCP-এর মতো বড় সংযোগ বা স্ট্রিমের প্রয়োজন নেই।

② দ্রুত প্রতিক্রিয়া (Quick response)

DNS হলো একটি “query-response” ধরনের অ্যাপ্লিকেশন —
যেখানে গতি (speed) সবচেয়ে গুরুত্বপূর্ণ।

UDP-তে কোনো connection establishment নেই (TCP handshake-এর মতো তিনটি ধাপ লাগে
না)।

ফলে এটি অনেক দ্রুত কাজ করে।

③ Connectionless প্রোটোকল কোনো সমস্যা নয়

DNS-এ প্রতিটি প্রশ্ন ও উত্তর স্বতন্ত্র (independent) —
প্রত্যেক query আলাদা datagram হিসেবে পাঠানো হয়।
তাই connectionless হওয়া কোনো সমস্যা নয়।

UDP packet out of order এলে তাতে DNS-এ কোনো প্রভাব পড়ে না,
কারণ প্রতিটি উত্তর নির্দিষ্ট query-এর জন্যই পাঠানো হয়।

৪ Overhead কম

UDP হেডার মাত্র ৮ bytes, যেখানে TCP হেডার ২০ bytes বা তার বেশি।
DNS-এর মতো ছোট মেসেজের জন্য এটি নেটওয়ার্ক ট্রাফিক কমায় এবং দ্রুততা বাড়ায়।

৫ TCP-এর অপ্রয়োজনীয় ফিচার এড়ানো

TCP-তে রয়েছে:

- সংযোগ স্থাপন ও বন্ধ করা (3-way handshake)
- acknowledgment ও retransmission
- sequencing ও congestion control

DNS-এর জন্য এগুলো অপ্রয়োজনীয় —
কারণ DNS query সাধারণত একবার পাঠানো হয়, উত্তর এলে কাজ শেষ।

১. সহজভাবে বোঝার মতো উদাহরণ

ধাপ	ক্লায়েন্ট	সার্ভার
১	DNS রিকোয়েস্ট পাঠায় (যেমন <code>www.google.com</code>) Port 53 (UDP)	
২	সার্ভার IP ঠিকানা পাঠায় (যেমন 142.250.190.4) ক্লায়েন্ট রেসপন্স গ্রহণ করে	
<input checked="" type="checkbox"/>	কাজ শেষ	কোনো connection তৈরি হয় না

এখানে প্রতিটি মেসেজ একটি UDP প্যাকেটেই ফিট হয়।

২. তবে কখন DNS TCP ব্যবহার করে

যখন DNS মেসেজ ৫১২ bytes-এর বেশি হয় (যেমন বড় zone transfer),
তখন UDP যথেষ্ট নয় — সেই ক্ষেত্রে DNS TCP ব্যবহার করে।
কিন্তু সাধারণ query-response-এর জন্য UDP-ই যথেষ্ট।

৩ সংক্ষিপ্ত সারাংশ

বৈশিষ্ট্য

DNS-এ প্রয়োগ

প্রোটোকল UDP (Port 53)

বার্তা আকার ছোট (একটি রিকোয়েস্ট, একটি রেসপন্স)

সংযোগ Connectionless

গতি দ্রুত (no handshake)

বিশ্বস্ততা প্রয়োজন নেই, কারণ response পুনরায় চাওয়া যায়

মূল ধারণা

DNS UDP ব্যবহার করে কারণ এটি দ্রুত, হালকা (lightweight) এবং প্রতিটি অনুরোধ-উত্তর একটি মাত্র datagram-এ ফিট হয়, তাই TCP-এর সংযোগ প্রক্রিয়া একদমই দরকার নেই।

“A client-server application such as SMTP ... cannot use the services of UDP...”

এটি আসলে ব্যাখ্যা করছে **কেন ইমেইল সিস্টেম (SMTP)-এর মতো অ্যাপ্লিকেশনগুলো UDP ব্যবহার না করে TCP ব্যবহার করে।**

চলো এটি সহজ ও বিস্তারিতভাবে বাংলায় ব্যাখ্যা করি ছি

🌐 কেন SMTP UDP ব্যবহার করতে পারে না

SMTP (Simple Mail Transfer Protocol) হলো ইমেইল পাঠানোর জন্য ব্যবহৃত প্রোটোকল। এটি সাধারণত TCP port 25 ব্যবহার করে — UDP নয়।

এর মূল কারণ হলো SMTP-র বার্তা (message) বা ইমেইল সাধারণত দীর্ঘ ও বড় আকারের, এবং এতে নির্ভরযোগ্য ডেলিভারি (reliable delivery) প্রয়োজন।

❖ UDP কেন উপযুক্ত নয়

১ ইমেইল বার্তা বড় (long message)

ইমেইল কেবল টেক্সট নয় — এতে ছবি, অডিও, ভিডিও বা অন্যান্য মাল্টিমিডিয়া থাকতে পারে। এমন বড় বার্তা একটি UDP datagram (maximum 65,535 bytes)-এর মধ্যে ফিট হয় না।

UDP বড় ডেটা ভাগ করতে পারে না —

তাই অ্যাপ্লিকেশনকেই নিজে বার্তাটিকে অনেকগুলো ছোট datagram-এ ভাগ করতে হবে।

→ এতে জটিলতা ও ক্রটির সম্ভাবনা বাড়ে।

② Connectionless যোগাযোগের সমস্যা

UDP হলো connectionless protocol, অর্থাৎ এটি ডেটা পাঠানোর আগে কোনো সংযোগ স্থাপন করে না।

ফলে:

- প্রতিটি datagram আলাদাভাবে পাঠানো হয়,
- এগুলো ভিন্ন পথে যেতে পারে,
- এবং **out of order** (অন্য ক্রমে) পৌঁছাতে পারে।

ইমেইলের ক্ষেত্রে, যদি বার্তার অংশগুলো অন্য ক্রমে পৌঁছায়,

তাহলে প্রাপক (receiver) সঠিকভাবে বার্তা পুনর্গঠন করতে পারবে না।

③ UDP-তে reliability নেই

UDP:

- ডেলিভারি নিশ্চয়তা দেয় না
- হারিয়ে যাওয়া প্যাকেট পুনরায় পাঠায় না
- acknowledgment বা sequence number ব্যবহার করে না

ইমেইল পাঠানোর ক্ষেত্রে এই বৈশিষ্ট্যগুলো খুব গুরুত্বপূর্ণ, কারণ:

ব্যবহারকারী নিশ্চিত হতে চায় যে বার্তাটি পৌঁছেছে এবং অক্ষত আছে।

→ তাই SMTP-এর মতো অ্যাপ্লিকেশন UDP ব্যবহার করলে বার্তা হারিয়ে যেতে পারে বা বিকৃত হতে পারে।

◊ TCP কেন SMTP-এর জন্য উপযুক্ত

বৈশিষ্ট্য

TCP-তে সুবিধা

Connection-oriented	ইমেইল পাঠানোর আগে সংযোগ তৈরি হয় (3-way handshake), ফলে নির্ভরযোগ্য যোগাযোগ সম্ভব।
Sequencing	TCP প্যাকেটগুলোর ক্রম (order) ঠিক রাখে।
Flow control	বড় বার্তা ধীরে ধীরে পাঠানো হয়, যাতে congestion না হয়।
Error control	প্যাকেট হারালে পুনরায় পাঠানো হয় (retransmission)।
Acknowledgment	প্রাপক নিশ্চিত করে যে ডেটা পেয়েছে।

→ এই বৈশিষ্ট্যগুলো ইমেইলের মতো বড় এবং গুরুত্বপূর্ণ ডেটা পাঠানোর জন্য একদম প্রয়োজনীয়।

⌚ সহজ উদাহরণ

ধরা যাক তুমি একটি 10MB ইমেইল পাঠাও:

- UDP ব্যবহার করলে বার্তাটি শত শত datagram-এ ভাগ হবে।
 - এদের কিছু হারাতে পারে, কিছু অন্য ক্রমে পৌঁছাতে পারে।
 - Receiver বার্তাটি সঠিকভাবে জোড়া লাগাতে পারবে না।
- ✗ ফলাফল: ইমেইল অসম্পূর্ণ বা নষ্ট।

TCP ব্যবহার করলে:

- TCP message-কে ছোট ছোট segment-এ ভাগ করে পাঠায়।
 - প্রতিটি segment acknowledgment সহ পৌঁছায়।
 - TCP সেগুলোকে ঠিক ক্রমে পুনরায় সাজিয়ে পুরো বার্তা তৈরি করে।
- ✓ ফলাফল: ইমেইল সম্পূর্ণ ও সঠিকভাবে পৌঁছায়।
-

💡 সারসংক্ষেপে

বৈশিষ্ট্য

UDP

TCP

সংযোগ

Connectionless

Connection-oriented

বৈশিষ্ট্য	UDP	TCP
নির্ভরযোগ্যতা	নেই	আছে
বার্তা আকার	ছোট	বড় (stream-based)
ক্রম ঠিক রাখা	না	হ্যাঁ
অ্যাপ্লিকেশন উদাহরণ DNS, DHCP, SNMP SMTP, HTTP, FTP, Telnet		

মূল ধারণা

SMTP UDP ব্যবহার করতে পারে না,
 কারণ এটি বড় ও ধারাবাহিক বার্তা পাঠায়,
 যেখানে ডেটা হারানো, ক্রমের ভুল, বা পুনর্গঠন ব্যর্থতা একেবারেই অগ্রহণযোগ্য।
 তাই এটি TCP ব্যবহার করে নির্ভরযোগ্য যোগাযোগ নিশ্চিত করে।

“Assume we are downloading a very large text file from the Internet... UDP is not a suitable transport layer.”

এই বক্তব্যটি ব্যাখ্যা করে কেন বড় ফাইল ডাউনলোডের মতো কাজে UDP ব্যবহার করা
 যায় না,
 এবং কেন এই ধরনের অ্যাপ্লিকেশনের জন্য TCP অপরিহার্য (essential)।

চলো এটি ধাপে ধাপে সহজ বাংলায় ব্যাখ্যা করি ছু

বড় ফাইল ডাউনলোডের ক্ষেত্রে UDP কেন উপযুক্ত নয়

ধরা যাক, তুমি ইন্টারনেট থেকে একটি বড় টেক্সট ফাইল (যেমন 200 MB) ডাউনলোড
 করছো।

তুমি নিশ্চয়ই চাও যে —

- ফাইলটি সম্পূর্ণ ডাউনলোড হোক,
- কোনো অংশ হারিয়ে না যাক,
- এবং কোনো ডেটা বিকৃত না হোক।

এই ধরণের নির্ভরযোগ্যতা UDP দিতে পারে না।

◦ UDP-এর সীমাবদ্ধতা

সীমাবদ্ধতা

ব্যাখ্যা

1 Connectionless communication	UDP কোনো সংযোগ তৈরি করে না। প্রতিটি ডেটাগ্রাম আলাদা ভাবে পাঠানো হয় — এতে কোনো ধারাবাহিকতা (order) বজায় থাকে না।
2 No sequencing	UDP জানে না কোন প্যাকেট আগে পাঠানো হয়েছে, কোনটা পরে এসেছে। ফলে বড় ফাইলের অংশগুলো অগোচালোভাবে (out of order) পৌঁছাতে পারে।
3 No acknowledgment	UDP প্রাপককে জিজেস করে না যে ডেটা ঠিকমতো পৌঁছেছে কিনা।
4 No retransmission	UDP প্যাকেট হারালে সেটি পুনরায় পাঠানো হয় না।
5 Possible corruption	UDP শুধুমাত্র checksum দিয়ে error detect করতে পারে, কিন্তু error correct করতে পারে না।

→ তাই বড় ফাইল পাঠাতে UDP ব্যবহার করলে ডেটার অংশ হারিয়ে যেতে পারে বা বিকৃত হতে পারে —
যা ব্যবহারকারীর জন্য অগ্রহণযোগ্য।

◦ TCP কেন উপযুক্ত (বড় ফাইলের জন্য)

TCP (Transmission Control Protocol) হলো একটি **connection-oriented, reliable, এবং byte-stream transport protocol**।

এটি নিম্নলিখিত সেবাগুলো প্রদান করে

TCP বৈশিষ্ট্য

ব্যাখ্যা

Connection-oriented	TCP ডেটা পাঠানোর আগে সংযোগ স্থাপন করে (3-way handshake)।
Reliability	প্রতিটি segment-এর জন্য acknowledgment পাঠায়।
Sequencing	সব প্যাকেটের ক্রম ঠিক রাখে।
Flow control	প্রাপক যতটা গ্রহণ করতে পারে, ততটাই পাঠায়।
Error detection & correction	ভুল শনাক্ত করে, প্রয়োজনে প্যাকেট পুনরায় পাঠায়।

TCP বৈশিষ্ট্য

No data loss

ব্যাখ্যা

সব ডেটা সঠিকভাবে পুনর্গঠিত হয়।

- তাই বড় ফাইল ডাউনলোডের ক্ষেত্রে TCP নিশ্চিত করে যে ফাইল সম্পূর্ণ, সঠিক ও অক্ষত অবস্থায় পৌঁছাবে।
-

💡 উদাহরণ

ধরা যাক তুমি HTTP বা FTP ব্যবহার করে ফাইল ডাউনলোড করছো।
এই প্রোটোকলগুলো TCP-র উপর কাজ করে কারণ তারা **reliability** চায়।

- **HTTP (Port 80)** → ওয়েব পেজ বা ফাইল ডাউনলোড
- **FTP (Port 21)** → বড় ফাইল ট্রান্সফার

TCP এখানে নিশ্চিত করে:

- প্রতিটি অংশ সঠিকভাবে পৌঁছেছে
 - কিছু হারালে পুনরায় পাঠানো হয়েছে
 - ফাইল খুললে কিছুই বিকৃত নয়
-

💡 UDP বনাম TCP (বড় ফাইল ট্রান্সফারের ক্ষেত্রে)

বৈশিষ্ট্য

UDP

TCP

সংযোগ

Connectionless

Connection-oriented

নির্ভরযোগ্যতা

নেই

আছে

সিকুয়েন্স

নেই

সঠিক ক্রমে ডেটা পৌঁছায়

হারানো ডেটা পুনরায় পাঠানো না

হ্যাঁ

উপযুক্ত ব্যবহার

রিয়েল-টাইম, ছোট ডেটা বড় ফাইল, নির্ভরযোগ্য ট্রান্সফার

☑ মূল বক্তব্য

বড় ফাইল ডাউনলোড বা ডেটা ট্রান্সফারের মতো অ্যাপ্লিকেশনের জন্য UDP উপযুক্ত নয়,
কারণ UDP কোনো **reliability guarantee** দেয় না।

এই ক্ষেত্রে TCP ব্যবহার করা প্রয়োজন, কারণ এটি নির্ভরযোগ্য ও সম্পূর্ণ ডেলিভারি নিশ্চিত করে।

“Assume we are watching a real-time stream video on our computer...”

এই ব্যাখ্যাটি বোঝায় কেন real-time video, audio, এবং voice streaming অ্যাপ্লিকেশনগুলো UDP ব্যবহার করে TCP নয়।

চলো ধাপে ধাপে সহজ বাংলায় বিষয়টি ব্যাখ্যা করি ফু

■ রিয়েল-টাইম ভিডিও স্ট্রিমিং ও UDP-এর সম্পর্ক

যখন তুমি YouTube Live, Netflix, বা Zoom-এর মতো কোনো লাইভ ভিডিও বা অডিও স্ট্রিম দেখো,
সেখানে ডেটা (ভিডিও ফ্রেম বা অডিও প্যাকেট) রিয়েল-টাইমে ছোট ছোট অংশে পাঠানো হয়।

এই ধরনের অ্যাপ্লিকেশনগুলোর জন্য মূল লক্ষ্য হলো —

✓ গতি (speed) এবং ধারাবাহিকতা (continuity)

✗ নির্ভুলতা (absolute reliability) নয়।

◦ কেন TCP এখানে উপযুক্ত নয়

TCP নির্ভরযোগ্য, কিন্তু রিয়েল-টাইম ঘোগাঘোগে খুব ধীর (slow) হয়ে যায়।

কারণ এটি:

- হারানো প্যাকেট পুনরায় পাঠায় (retransmission),
- acknowledgment-এর জন্য অপেক্ষা করে,
- সব ডেটা ক্রমানুসারে পৌঁছেছে কি না তা যাচাই করে।

→ ফলাফল: ভিডিও থেমে যায় , স্ক্রিন ব্ল্যাক হয়ে যায়, অথবা দেরি (lag) তৈরি হয়।

তুমি ভিডিও দেখছো, হঠাৎ TCP যদি কোনো প্যাকেট হারানো পায় —

তাহলে সেটি পুনরায় পাঠানোর আগে ভিডিও প্লে বন্ধ করে দেয়।

যখন নতুন প্যাকেট আসে, ততক্ষণে sync হারিয়ে যায়।

এটি দর্শকের জন্য অসহনীয় বিলম্ব (unacceptable delay)।

◊ UDP কেন উপযুক্ত

UDP হলো connectionless এবং fast প্রোটোকল,
যা corrupted বা lost প্যাকেট পুনরায় পাঠায় না।

ভিডিও স্ট্রিমিং-এ UDP-এর সুবিধাগুলো ছু

সুবিধা

No retransmission হারানো ফ্রেম পুনরায় পাঠানো হয় না → ফলে বিলম্ব হয় না।

Low latency খুব দ্রুত ডেটা পাঠানো যায় → রিয়েল-টাইম প্লেব্যাক সম্ভব।

Can ignore errors কোনো প্যাকেট নষ্ট হলে UDP সেটি বাদ দিয়ে পরের ফ্রেম ডেলিভার করে।

Smooth streaming ক্রিন সামান্য ব্ল্যাক্স হলেও পুরো ভিডিও থেমে যায় না।

ব্যাখ্যা

◊ UDP-তে বাস্তব প্রয়োগ (Real-Time Example)

ধরা যাক একটি ভিডিও ফ্রেম 30 ফ্রেম/সেকেন্ড গতিতে পাঠানো হচ্ছে —
অর্থাৎ প্রতি সেকেন্ডে 30টি ছোট UDP datagram।

যদি এর মধ্যে ২টি প্যাকেট হারায়:

- TCP হলে ভিডিও থেমে যেত (retransmit-এর জন্য অপেক্ষা করত)।
- UDP হলে এই দুই ফ্রেম বাদ দিয়ে ভিডিও চলতে থাকবে —
হয়তো চোখে পড়বেই না ৳।

◊ অবশ্যই দরকার — Application-Level Control

যদিও UDP দ্রুত,

ভিডিও দেখা যায় out of order (অন্য ক্রমে) প্যাকেট এলেও — তা ঠিক রাখতে হয়।
তাই অ্যাপ্লিকেশন নিজেই (যেমন VLC, Zoom, বা YouTube Player):

- Frames reorder করে (যদি সামান্য delay হয়), অথবা
- Late/Corrupted frames drop করে (যাতে ভিডিও থেমে না যায়)।

→ এই কাজ সাধারণত RTP (Real-time Transport Protocol) বা RTSP (Real-time Streaming Protocol) UDP-এর উপরে চালিয়ে করা হয়।

বিন্দু সহজভাবে তুলনা

বৈশিষ্ট্য	TCP	UDP
Connection	Connection-oriented	Connectionless
Reliability	100% নির্ভরযোগ্য	কিছু হারাতে পারে
Speed	ধীর (retransmission সহ) দ্রুত	
Delay	বেশি	খুব কম
Video Quality Impact	থেমে যায়, ব্ল্যাক স্ক্রিন	সামান্য ফ্রেম মিস, কিন্তু চলমান ভিডিও
উদাহরণ	File Download, Email	Live Video, VoIP, Online Games

১. সারসংক্ষেপ

রিয়েল-টাইম ভিডিও বা অডিও স্ট্রিমিংয়ের জন্য UDP উপযুক্ত,
কারণ এতে **delay কম, retransmission নেই**, এবং
ছোটখাটো packet loss বা error দর্শকের কাছে প্রায় চোখে পড়ে না।

করি ছি

🌐 UDP Package Structure (UDP-এর অভ্যন্তরীণ উপাদানসমূহ)

UDP প্যাকেট পাঠানো ও গ্রহণের জন্য কিছু নির্দিষ্ট উপাদান (components) একসাথে কাজ করে।
একটি সাধারণ UDP প্যাকেজে থাকে পাঁচটি প্রধান অংশ —

- 1 Control-block table
 - 2 Input queues
 - 3 Control-block module
 - 4 Input module
 - 5 Output module
-

১. Control-Block Table

এটি UDP-এর তথ্য সংরক্ষণের কেন্দ্র (data registry)।

- প্রতিটি সক্রিয় (active) UDP port-এর জন্য এখানে একটি entry রাখা হয়।

- প্রতিটি entry-তে থাকে:
 - Port number (যেমন 53, 67, 52000)
 - IP address (source/destination)
 - Pointer to input/output queues
 - Process ID (যে অ্যাপ্লিকেশন ওই পোর্ট ব্যবহার করছে)

□ উদ্দেশ্য:

UDP কে জানানো, কোন port কোন প্রক্রিয়ার (process) সাথে যুক্ত,
এবং প্রাপ্ত প্যাকেট কোন অ্যাপ্লিকেশনে পাঠাতে হবে।

১. Input Queues

Incoming datagrams (প্যাকেট) সাময়িকভাবে এখানে সংরক্ষিত থাকে।

- প্রতিটি UDP port-এর জন্য একটি **input queue** থাকে।
- যখন কোনো UDP packet গন্তব্যে আসে,
UDP সেটিকে queue-তে রাখে যতক্ষণ না **application process** সেটি পড়ে।

□ উদ্দেশ্য:

প্রাপ্ত প্যাকেটগুলোকে অস্থায়ীভাবে ধরে রাখা —
যাতে UDP সেগুলো সঠিক অ্যাপ্লিকেশনে পাঠাতে পারে।

২. Control-Block Module

এটি UDP-এর ম্যানেজার বা প্রশাসক অংশ।

- Control-block table তৈরি, আপডেট এবং মুছে ফেলার কাজ করে।
- যখন নতুন কোনো process UDP port খুলে (bind করে),
তখন এটি table-এ নতুন entry যোগ করে।
- কোনো process শেষ হলে entry মুছে দেয়।

□ উদ্দেশ্য:

UDP port এবং process-এর মধ্যে সম্পর্ক (binding) পরিচালনা করা।

৩. Input Module

এটি UDP-এর receiver unit — incoming প্যাকেট প্রসেস করে।

- Network layer (IP) থেকে প্যাকেট গ্রহণ করে।
- Checksum যাচাই করে (ডেটা নষ্ট হয়েছে কিনা)।
- Control-block table দেখে **destination port** শনাক্ত করে।
- সঠিক process-এর **input queue**-তে প্যাকেট রাখে।

□ **উদ্দেশ্য:**

প্রাপ্ত UDP datagram সঠিক অ্যাপ্লিকেশনে পাঠানো।

④ **Output Module**

এটি UDP-এর **sender unit** — outgoing message পাঠায়।

- Application থেকে message ও destination তথ্য গ্রহণ করে।
- UDP header তৈরি করে (source port, destination port, length, checksum)।
- Datagram IP layer-এ পাঠিয়ে দেয়।

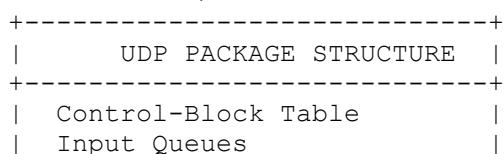
□ **উদ্দেশ্য:**

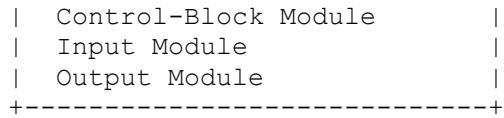
অ্যাপ্লিকেশন থেকে প্রাপ্ত ডেটাকে UDP datagram আকারে নেটওয়ার্কে পাঠানো।

⑤ **UDP Operation Flow**

ধাপ	কাজ	মডিউল
①	অ্যাপ্লিকেশন UDP-কে বার্তা পাঠায়	Output Module
②	UDP header তৈরি ও IP layer-এ পাঠানো	Output Module
③	IP থেকে UDP datagram প্রাপ্তি	Input Module
④	Checksum যাচাই ও সঠিক port শনাক্ত	Input Module + Control-Block Table
⑤	প্যাকেট সঠিক queue-তে রাখা	Input Queue
⑥	অ্যাপ্লিকেশন queue থেকে প্যাকেট গ্রহণ করে	Application Layer

💡 **চিত্রে (ধারণাগতভাবে):**





- Application → Output Module → IP layer
 - IP layer → Input Module → Input Queue → Application
-

১. সংক্ষিপ্ত সারাংশ

উপাদান

কাজ

Control-Block Table Port ও process-এর ম্যাপ সংরক্ষণ করে

Input Queues প্রাপ্ত প্যাকেট অস্থায়ীভাবে ধরে রাখে

Control-Block Module Port binding পরিচালনা করে

Input Module প্রাপ্ত প্যাকেট প্রক্রিয়াজাত করে ও queue-তে রাখে

Output Module প্যাকেট তৈরি করে ও IP layer-এ পাঠায়

২. মূল ধারণা

UDP-এর অভ্যন্তরীণ কাজ ৫টি অংশের সমন্বয়ে সম্পন্ন হয় —

Control-Block Table, Input Queues, Control-Block Module, Input Module, এবং Output Module।
এগুলো একসাথে কাজ করে UDP-কে **দ্রুত, connectionless, ও lightweight communication** প্রদান করতে সাহায্য করে।

Figure 14.8 UDP design

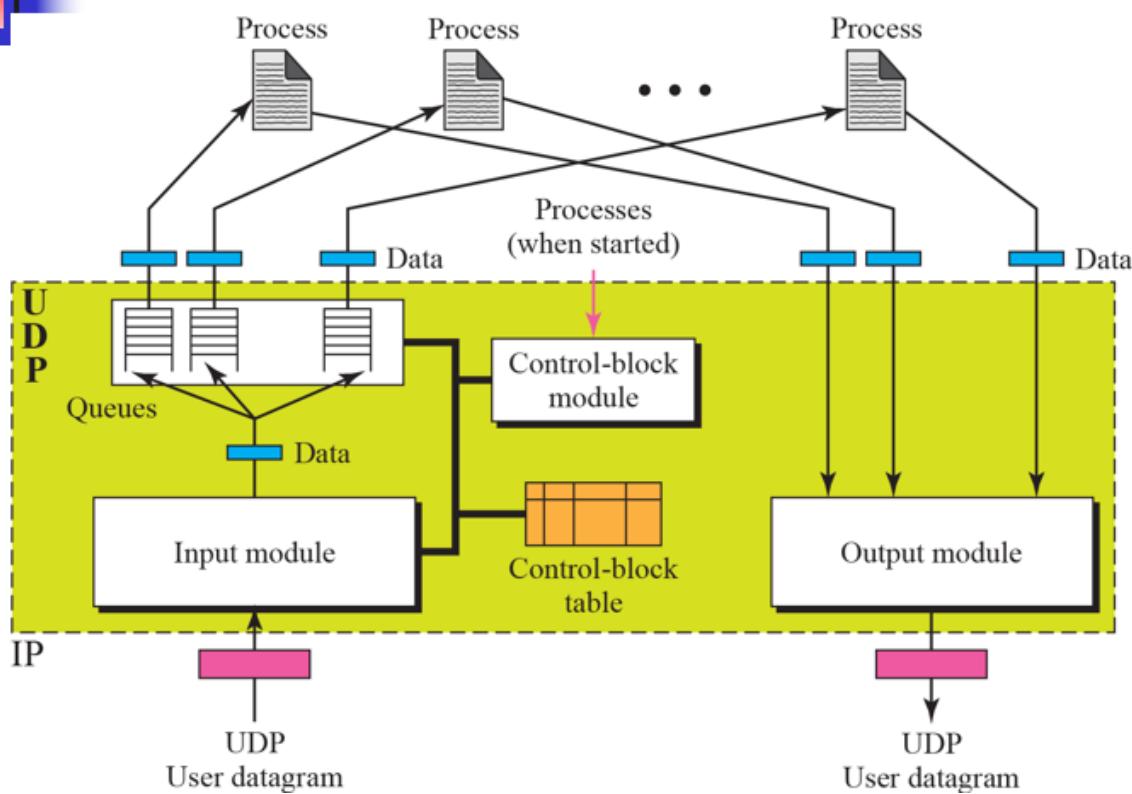


Figure 14.8 — UDP Design Overview

এই চিত্রটি দেখায় UDP কীভাবে Application layer (Processes) এবং IP layer-এর মধ্যে মধ্যস্থতাকারী হিসেবে কাজ করে।

UDP একটি মাঝের স্তর (intermediate layer) যেখানে:

- উপরের দিকে থাকে একাধিক process (applications) যেমন DNS, DHCP, SNMP ইত্যাদি।
- নিচের দিকে থাকে IP layer, যা নেটওয়ার্কে ডেটা প্রেরণ করে।

UDP-এর ভেতরে থাকে ৫টি প্রধান অংশ (যেমন তুমি আগেই বলেছিলে):

- 1 Control-block table
- 2 Queues
- 3 Control-block module
- 4 Input module
- 5 Output module

❑ UDP-এর প্রতিটি অংশের কাজ (চিত্র অনুযায়ী)

◊ 1. Processes (Application Layer)

এগুলো হলো ব্যবহারকারীর অ্যাপ্লিকেশন —
যেমন DNS client, SNMP agent, বা DHCP server।

- প্রতিটি প্রক্রিয়া UDP-এর মাধ্যমে বার্তা পাঠায় বা গ্রহণ করে।
 - প্রতিটি process একটি **port number**-এর সাথে যুক্ত থাকে।
-

◊ 2. Control-Block Table

এটি হলো UDP-এর “Address Book” বা **Port Management Database**।

- প্রতিটি সক্রিয় process-এর জন্য এখানে একটি **entry** রাখা হয়।
- এতে থাকে:
 - Process ID
 - Port number
 - IP address
 - Pointer to input/output queue

❑ **কাজ:**

UDP যেন জানে কোন incoming packet কোন process-এর জন্য,
এবং কোন process কোন port ব্যবহার করছে।

◊ 3. Control-Block Module

এটি UDP-এর **Manager Module**।

- যখন নতুন কোনো process UDP port bind করে, তখন এটি Control-Block Table-এ নতুন entry তৈরি করে।
- যখন কোনো process বন্ধ হয়, তখন সেই entry মুছে দেয়।

❑ **কাজ:**

Port binding এবং process registration পরিচালনা করা।

◊ 4. Input Module (Receiver Side)

- IP layer থেকে আসা UDP datagram এখানে আসে।
- Input Module checksum বাচাই করে এবং Control-Block Table দেখে কোন port-এ পাঠাতে হবে তা নির্ধারণ করে।
- তারপর প্যাকেটটি সেই process-এর incoming queue-তে পাঠায়।

□ কাজ:

প্রাপ্ত UDP datagram সঠিক অ্যাপ্লিকেশনে পৌঁছে দেওয়া।

➡ উদাহরণ:

IP থেকে প্যাকেট এল → UDP Input Module → Port 53 (DNS Process)-এ পাঠানো হলো।

◊ 5. Output Module (Sender Side)

- Application process থেকে যখন কোনো বার্তা পাঠাতে হয়, তা Output Module-এর মাধ্যমে যায়।
- Output Module UDP header ঘোগ করে (source port, destination port, length, checksum)।
- এরপর এটি IP layer-এ পাঠানো হয়।

□ কাজ:

অ্যাপ্লিকেশন থেকে ডেটা নিয়ে UDP datagram তৈরি করে নেটওয়ার্কে পাঠানো।

◊ 6. Queues (Input/Output Queues)

প্রতিটি process-এর জন্য UDP দুটি কিউ রাখে:

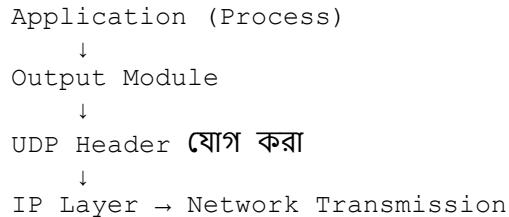
- Incoming queue: প্রাপ্ত ডেটা অস্থায়ীভাবে রাখে
- Outgoing queue: প্রেরণের জন্য অপেক্ষমান ডেটা রাখে

□ কাজ:

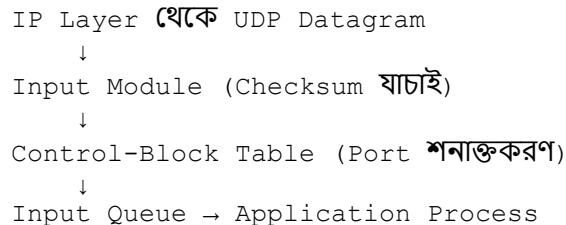
ডেটা ট্রাফিক নিয়ন্ত্রণ এবং অ্যাপ্লিকেশনকে সঠিকভাবে ডেটা সরবরাহ করা।

➡ UDP ডেটা প্রবাহ (Flow of Data)

Sender Side (Encapsulation)



Receiver Side (Decapsulation)



UDP Design-এর সারসংক্ষেপ টেবিল

অংশ	কাজ	দিক
Processes	অ্যাপ্লিকেশন প্রোগ্রাম (যেমন DNS, SNMP, DHCP) Application Layer	
Control-Block Module	Port registration ও binding পরিচালনা	Management
Control-Block Table	Active port এবং process-এর তালিকা রাখে	Internal storage
Input Module	প্রাপ্ত প্যাকেটকে সঠিক queue-তে পাঠায়	Receiving side
Output Module	বার্তা প্যাকেট তৈরি করে IP-তে পাঠায়	Sending side
Queues	প্যাকেট সাময়িকভাবে সংরক্ষণ করে	Buffering

✓ মূল ধারণা

UDP Design হলো এমন এক গঠন যেখানে **Input & Output Modules**, **Control-Block System**, এবং **Queues** একসাথে কাজ করে **দ্রুত, connectionless communication** নিশ্চিত করতে।

UDP কোনো জটিল সংযোগ তৈরি না করেই application থেকে IP পর্যন্ত ডেটা সহজে প্রবাহিত করে — এটি TCP-এর তুলনায় অনেক হালকা (lightweight)।

Table 14.5 The Control-Block Table at the Beginning of Examples

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			

🌐 Table 14.5 — The Control-Block Table (at the beginning)

এই টেবিলটি UDP-এর একটি অভ্যন্তরীণ ডেটা স্ট্রাকচার দেখায় —
যেখানে প্রতিটি UDP port এবং তার সাথে যুক্ত process ও queue number সংরক্ষিত থাকে।

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	—	—	—

ଡেটা কন্ট্রল টেবিল-এর মানে

ফিল্ড	বর্ণনা
State	পোর্টটি ব্যবহৃত হচ্ছে কি না (IN-USE বা FREE)
Process ID	কোন প্রক্রিয়া (application) পোর্টটি ব্যবহার করছে
Port Number	UDP পোর্ট নাম্বার (যেমন 52,010, 52,011, 52,012 ইত্যাদি)
Queue Number	প্রাপ্ত ডেটা সংরক্ষণের জন্য ব্যবহৃত কিউ আইডি

Figure 14.8-এর সঙ্গে সম্পর্ক

এই টেবিলটি UDP design-এর **Control-Block Table** অংশের উদাহরণ।
UDP যখন কোনো প্যাকেট পায়, তখন এটি প্রথমে Control-Block Table-এ গিয়ে **destination port number** অনুসন্ধান করে।

Example 14.8 — Explanation in Bengali

“The first activity is the arrival of a user datagram with destination port number 52,012...”

চলো লাইন ধরে বিশ্লেষণ করি ছু

1 একটি নতুন UDP datagram আসে

- গন্তব্য (destination) পোর্ট নম্বর = **52,012**
 - UDP Input Module প্যাকেটটি গ্রহণ করে।
-

2 Input Module Table-এ অনুসন্ধান করে

UDP-এর **Input Module** Control-Block Table-এ গিয়ে দেখে,
52,012 পোর্ট নম্বরটি কোন process-এর সঙ্গে যুক্ত আছে?

টেবিলে দেখা যায়:

Port Number	Process ID	Queue Number
52,012	4,652	38

3 পোর্টটি “IN-USE” অবস্থায় আছে

এটি বোঝায় —

এই পোর্ট আগে থেকেই **একটি process (ID 4652)** দ্বারা ব্যবহৃত হচ্ছে।
অর্থাৎ UDP জানে, এই ডেটা কোন queue-তে পাঠাতে হবে।

4 Queue নম্বর 38 ব্যবহার করা হয়

পোর্ট 52,012-এর সাথে যুক্ত queue হলো Queue 38। UDP Input Module প্রাপ্ত ডেটাটি সেই queue-তে পাঠায়। এই queue থেকে পরে সংশ্লিষ্ট process ডেটা পড়বে।

→ ডেটা পাঠানো হয়েছে → Queue 38

৫ Control-Block Table অপরিবর্তিত থাকে

କାରଣ ଏଇ ପୋର୍ଟ ଆଗେଇ ତୈରି କରା ହେଲିଛି,
ତାଇ କୋନୋ ନତ୍ତନ entry ଯୋଗ ହୁଏ ନା ବା state ପରିବର୍ତ୍ତନ ହୁଏ ନା।

Result:
UDP শুধু queue-তে ডেটা রাখে,
কিন্তু Control-Block Table অপরিবর্তিত থাকে।

Example 14.8 সারসংক্ষেপ টেবিল

ধাপ	কাজ	ফলাফল
1	Datagram আসে (destination port 52,012) Input Module প্যাকেট গ্রহণ করে	
2	Control-Block Table-এ অনুসন্ধান পোর্ট 52,012 পাওয়া যায়	
3	পোর্টের তথ্য যাচাই State = IN-USE, Process ID = 4652	
4	Queue শনাক্ত Queue number = 38	
5	ডেটা Queue-তে পাঠানো UDP Queue 38-এ ডেটা সংরক্ষণ করে	
6	Table পরিবর্তন Table অপরিবর্তিত থাকে	

মূল ধারণা

UDP Input Module যখন কোনো datagram পাই,
এটি Control-Block Table-এ গিয়ে destination port খুঁজে বের করে,
তারপর সেই পোর্টের নির্ধারিত queue-তে ডেটা পাঠায়।
যদি পোর্টটি আগেই ব্যবহৃত হয় (IN-USE), টেবিলে কোনো পরিবর্তন হয় না।

Table 14.6 Control-Block Table after Example 14.9

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 14.9 — New Process Starts

এই উদাহরণে দেখানো হয়েছে—

যখন একটি নতুন process শুরু হয় এবং UDP port চায়, তখন UDP কীভাবে Control-Block Table আপডেট করে।

① ঘটনার বিবরণ

① একটি নতুন process শুরু হয়েছে।

- Process ID = 4,978

② এটি OS (Operating System)-এর কাছে একটি UDP port নম্বর চায়।

- OS নতুন পোর্ট হিসেবে দেয় 52,014

③ Process UDP Control-Block Module-এ তথ্য পাঠায়:

Process ID = 4,978
Port Number = 52,014

④ Control-Block Module এই তথ্য গ্রহণ করে এবং Control-Block Table-এ প্রথম ফাঁকা (FREE) এন্ট্রি খুঁজে বের করে।

- আগের টেবিল (Table 14.5)-এ দুইটি FREE entry ছিল।
- তাই এটি প্রথম FREE entry-তে নতুন তথ্য যোগ করে।

৫ Queue Number এখনো বরাদ্দ করা হয়নি।

- কারণ এই পোর্টে এখনো কোনো UDP datagram আসেনি।
 - যখন কোনো প্যাকেট এই পোর্টে আসবে, তখনই queue তৈরি বা বরাদ্দ করা হবে।

④ Updated Control-Block Table (Table 14.6)

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	—	—	—

ଟେବିଲ ବିଶ୍ଳେଷଣ

ফিল্ড	মানে
State = IN-USE	পোর্ট 52,014 এখন সক্রিয়ভাবে ব্যবহৃত হচ্ছে।

Process ID = 4,978 এটি সেই নতুন প্রক্রিয়া যা UDP পোর্ট পেয়েছে।

Port Number = 52 014 ୦୯ ଦ୍ୱାରା ନିର୍ମିତ ପୋର୍ଟ।

Queue Number = (কাঁকা) এখন কোনো datagram আসেনি তবে

ঔষধ কা ঘটল UDP-এর ভেতরে (UDP design অনুযায়া)

চলো চিত্র (Figure 14.8)-এর প্রেক্ষিতে দোখ UDP-এর ভেতরে কী হলো ছ

- ① **Process শুরু হলো** → Control-Block Module-এ তথ্য পাঠালো।
 - ② **Control-Block Module** → Control-Block Table-এ নতুন entry তৈরি করলো।
 - ③ **Queue তৈরি হয়নি** → কারণ Input Module এখনো কোনো প্যাকেট পায়নি।

→ UDP-এর কাঠামো এখন জানে:

Port 52,014 ৰ্যবহাৰ কৰছে Process 4,978, কিন্তু এখনো কোনো প্যাকেট আসেনি।

💡 UDP-এর আচরণ বোঝা (Key Takeaway)

ধাপ	কাজ
Process UDP port চায়	OS পোর্ট বরাদ্দ দেয়
UDP Control-Block Module	টেবিলে নতুন entry তৈরি করে
Queue এখনো নেই	কারণ এখনো কোনো ডেটা পাওয়া যায়নি
ভবিষ্যতে যখন প্যাকেট আসবে UDP Queue বরাদ্দ করবে এবং Input Module সেটি queue-তে রাখবে	

☑ মূল ধারণা

যখন একটি নতুন UDP process শুরু হয়,
UDP Control-Block Module প্রথম ফাঁকা (FREE) টেবিল এন্ট্রি তার Process ID ও Port নম্বর
যোগ করে।
কিন্তু Queue তখনো বরাদ্দ করা হয় না —
Queue কেবল তখনই তৈরি হয় যখন এই পোর্টে প্রথম datagram আসে।

Table 14.7 Control-Block Table after Example 14.10

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 14.10 — Arrival of First Datagram for Port 52,011

□ পূর্বের অবস্থা (Before the event)

আগে Table 14.6-এ UDP Control-Block Table ছিল এই রকম ছিল

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	—	—	—

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	—	—	—

দেখা যাচ্ছে যে,

Port 52,011 ব্যবহৃত হচ্ছে (IN-USE) —

কিন্তু এর কোনো queue number বরাদ্দ নেই,

কারণ এখনো এই পোর্টে কোনো datagram আসেনি।

ঝটনা: একটি নতুন datagram আসে

১ একটি UDP user datagram আসে, যার destination port = 52,011।

২ UDP-এর Input Module Control-Block Table-এ দেখে এই পোর্টের তথ্য আছে কিনা।

৩ এটি পায় যে পোর্টটি IN-USE অবস্থায় আছে (Process ID = 3,422),

কিন্তু কোনো queue বরাদ্দ নেই।

Input Module-এর কাজ

Input Module বুঝে যায় যে এটি এই পোর্টে প্রথম ডেটা আগমন,
তাই এটি একটি নতুন queue তৈরি করে এই পোর্টের জন্য।

→ নতুন Queue Number = 43

তারপর Input Module Control-Block Table আপডেট করে,
Port 52,011-এর queue ফিল্ডে এই নতুন নম্বর বসিয়ে দেয়।

Updated Control-Block Table (Table 14.7)

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	—	—	—

বিশ্লেষণ

ফিল্ড	মানে
Port 52,011	আগে থেকেই IN-USE (process ID 3,422 দ্বারা)
Queue তৈরি হয়েছে	কারণ এটি এই পোর্টের জন্য প্রথম প্যাকেট আগমন
Queue Number = 43	নতুন queue ID বরাদ্দ

Table পরিবর্তিত হয়েছে শুধুমাত্র Port 52,011-এর queue field আপডেট হয়েছে

প্রক্রিয়ার ধাপ (Step-by-Step Flow)

- | ধাপ | ক্রিয়া | ফলাফল |
|-----|------------------------------------------------------------------|-------------------------------------|
| ① | Datagram আসে (dest port = 52,011) Input Module প্যাকেট গ্রহণ করে | |
| ② | Table চেক করা হয় | Entry পাওয়া যায়, কিন্তু queue নেই |
| ③ | Queue তৈরি করা হয় | Queue number = 43 |
| ④ | Table আপডেট করা হয় | Port 52,011 → Queue 43 |
| ⑤ | প্যাকেট queue-তে রাখা হয় | Process 3,422 পরে এটি পড়বে |
-

মূল ধারণা

UDP Input Module যখন কোনো নতুন পোর্টে প্রথমবার ডেটা পায়,
তখন এটি একটি নতুন queue তৈরি করে এবং Control-Block Table আপডেট করে।
এই queue ভবিষ্যতের সব incoming প্যাকেটের জন্য ব্যবহৃত হবে।—

“After a few seconds, a user datagram arrives for port 52,222...”

এটি Example 14.11-এর ব্যাখ্যা অংশ,
যেখানে দেখানো হয়েছে UDP কীভাবে অজানা বা অনুপস্থিত পোর্ট (unreachable port)
পাঠানো প্যাকেটের প্রতিক্রিয়া দেয়।

চলো এটি বাংলায় ধাপে ধাপে বিশ্লেষণ করি ॥

Example 14.11 — Datagram Arrives for a Nonexistent Port

পরিস্থিতি (Situation)

UDP-এর Control-Block Table এখন এরকম ছিল (Table 14.7 অনুযায়ী):

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	—	—	—

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	—	—	—

এই টেবিলে দেখা যাচ্ছে:

- বর্তমানে সক্রিয় পোর্টগুলো হলো: 52,010, 52,011, 52,012, এবং 52,014
 - অন্য কোনো পোর্ট (যেমন 52,222) Control-Block Table-এ নেই।
-

ঁ ঘটনা: একটি নতুন UDP datagram আসে

- ① IP layer থেকে UDP-তে একটি user datagram আসে।
 - ② Datagram-এর destination port = 52,222।
 - ③ UDP-এর Input Module Control-Block Table-এ খুঁজে দেখে:
“এই পোর্ট নম্বরের কোনো entry আছে কি?”
-

✗ ফলাফল: কোনো entry পাওয়া যায় না

UDP বুঝে যায় —

“Port 52,222 আমার কোনো active process বা port দ্বারা ব্যবহৃত নয়।”

এর মানে হলো,
এই প্যাকেটের গন্তব্য (destination process) **অস্তিত্ব নেই**।

④ UDP-এর পদক্ষেপ

UDP দুটি কাজ করে ছি

① Datagram drop করা (Discard)

UDP প্যাকেটটি ফেলে দেয় (drop করে),
কারণ এর জন্য কোনো সক্রিয় port নেই।
UDP কোনো অজানা process-এর জন্য queue তৈরি করে না।

② ICMP-এর মাধ্যমে error report করা

UDP নিজে error message পাঠায় না।
এর পরিবর্তে UDP ICMP (Internet Control Message Protocol)-কে অনুরোধ করে:

“এই source IP-কে জানাও যে port 52,222 **unreachable**।”

ICMP তখন একটি “Destination Unreachable – Port Unreachable” মেসেজ পাঠায়
মূল প্রেরক (source) IP-তে।

💡 প্রক্রিয়ার সারাংশ

ধাপ	কাজ	ফলাফল
①	UDP datagram আসে (destination port 52,222) Input Module প্যাকেট গ্রহণ করে	
②	Control-Block Table অনুসন্ধান	পোর্ট পাওয়া ঘায়নি
③	Datagram drop করা হয়	UDP discard করে
④	ICMP-কে জানানো হয়	ICMP → “Port Unreachable” বার্তা পাঠায়
☒	কোনো টেবিল পরিবর্তন হয় না	Control-Block Table অপরিবর্তিত থাকে

⚡ সহজভাবে বোঝার মতো উদাহরণ

ধরা যাক, তুমি একটি চিঠি পাঠালে **একটি ভুল ঠিকানায়** —
সেখানে কোনো মানুষ নেই (port নেই)।

ডাকঘর (UDP) চিঠিটি ফেলে দেয়,
তারপর **একটি ফেরত বার্তা (ICMP message)** পাঠায়:

“Address does not exist!”

একইভাবে UDP যখন কোনো অচেনা port পায়,
তখন সেটি discard করে এবং ICMP error report পাঠায়।

▣ UDP + ICMP সমন্বয়

প্রোটোকল	ভূমিকা
UDP	Port চেক করে; ডেটা সঠিক প্রক্রিয়ায় পৌঁছে দেয় বা drop করে
ICMP	UDP থেকে নির্দেশ পেলে source host-কে error বার্তা পাঠায়

ICMP Error Message:

Type: 3 (Destination Unreachable)
Code: 3 (Port Unreachable)

মূল ধারণা

যদি UDP কোনো অজ্ঞান বা inactive port নম্বর পায়,
তাহলে এটি প্যাকেটটি drop করে এবং
ICMP-কে নির্দেশ দেয় একটি “Port Unreachable” বার্তা পাঠাতে।

এইভাবে UDP অপ্রয়োজনীয় ডেটা গ্রহণ করা থেকে সিস্টেমকে রক্ষা করে।