Advance Programming Techniques (APT)

Lecture # 44

Ehtisham Rasheed

Department of Computer Science
University of Gurjat, Gujrat

What is Network Programming?

- **Network programming** means writing applications that can **communicate over a network**—LAN, WAN, or the internet
- Network programming = sending + receiving data between two or more devices using network protocols
- C# provides powerful libraries inside **System.Net** and **System.Net.Sockets** to build network-enabled applications

When Do We Need Network Programming?

- We need network programming whenever our application must **communicate with another application or machine**, for example:

✓ Client → Server communication

- Example: A mobile app sending data to a server

✓ Real-time communication

- Example: Chat apps, multiplayer games, notifications

✓ Remote data access

- Example: A C# app retrieving data from a web API

✓ Communication between microservices

- Example: Modern distributed systems

Big Picture

- Every modern application is a network application
- WhatsApp, Google, Netflix, Banking apps, Games — all communicate over networks
- We will learn networking in **three logical layers**:

```
Sockets → HTTP/APIs → Microservices & Messaging
```

- Each layer is built **on top of the previous one**

1. What is Socket Programming?

- **Socket programming** is **low-level network programming** where applications communicate **directly using TCP or UDP**
- A socket is an endpoint for sending and receiving data over a network

Core Concepts

1 IP Address

- Identifies a machine
- Example: `192.168.1.10`, `127.0.0.1`, (localhost – same computer)

2 Port Number

- Identifies an application
- Example: `5000`

3 Protocol

- Rules of communication
- TCP or UDP

Socket = IP + Port + Protocol

```
127.0.0.1 : 5000 : TCP
```

(if **IP** is house #, **Port** is door #)

TCP vs UDP

TCP	UDP
Transmission Control Protocol	User Datagram Protocol
Reliable	Not reliable
Slow	Fast
Extensive error-checking	Basic error-checking
Acknowledgement	No Acknowledgement
Retransmit lost packets	No retransmit lost packets
Connection-based	Connectionless
Guaranteed delivery	No guarantee
Chat, banking, email	Games, video streaming, audio streaming

Client–Server Model (Sockets)

- Server opens a port
- Client connects
- Client sends message
- Server replies
- Connection closes
- Key classes:
  - Socket, TcpListner, TcpClient, UdpClient, NetworkStream

Important Classes

Class	Purpose
Socket	Low-level socket
TcpListener	Create server
TcpClient	Create client
UdpClient	UDP communication
NetworkStream	Send / receive data
Encoding	Convert string ↔ bytes

What Can We Build With Sockets?

- Chat applications
- Multiplayer games
- File transfer tools
- Remote desktop
- IoT communication
- Real-time monitoring system

2. HTTP & API Programming?

- Do we really want to manage sockets for every app?
- ❌ No — we need something **simpler and standardized**
- That's where **HTTP** comes in
- **HTTP (HyperText Transfer Protocol)** is a **high-level protocol** built on top of **TCP sockets**
- **HTTP uses sockets internally**, but hides complexity

```
Client → HTTP Request → Server
Client ← HTTP Response ← Server
```

- A **REST API** exposes services as URLs

```
GET /api/students
POST /api/login
```

HTTP Methods

Method	Purpose
GET	Fetch data
POST	Send data
PUT	Update
DELETE	Remove

Why APIs Are Better Than Sockets?

Sockets	APIs
Low-level	High-level
TCP / UDP	HTTP, gRPC
Harder	Easier
Fast	Slightly slower
Manual data handling	Automatic serialization

What Can We Build With HTTP?

- Web applications
- Mobile apps
- Cloud systems
- Microservices
- Banking & ERP systems

3. Microservices & Messaging

- A **microservice** is a **small, independent application** that performs **one specific task**
- Instead of building **one big application**, we break it into **multiple small services**
- ❌ Old Way (Monolithic App)

```
One Application
│
├── Login
├── Orders
├── Payments
├── Reports
└── Notifications
```

- ✅ Modern Way (Microservices)

```
Login Service
Order Service
Payment Service
Notification Service
Report Service
```

Each service:

- Runs independently
- Has its own database (often)
- Communicates over a network

Why Microservices Communicate?

- Because **one service cannot do everything alone**
- Example: Online Shopping System

1. **Order Service** creates an order
2. It needs to:
   - Ask **Payment Service** to charge money
   - Ask **Inventory Service** to check stock
   - Ask **Notification Service** to send email/SMS

- 👉 These services **must talk to each other over a network**

What Types of Apps Can Students Build?

1 Chat & Messaging Systems

- Console-based chat app
- Windows Forms chat messenger
- Encrypted chat application
- Group chat using sockets
- WhatsApp-style messaging demo

2 Multiplayer Online Games

- Tic-Tac-Toe online
- Chess online
- Car racing multiplayer
- Shooting game using TCP/UDP

What Types of Apps Can Students Build?

3️⃣ Client–Server Applications

- Remote calculator
- Remote file manager
- Remote desktop monitoring
- Remote shutdown/restart tool

4️⃣ IoT & Hardware Projects

- Device monitoring (sensors sending data)
- Smart home controller app
- GPS tracking visualization

What Types of Apps Can Students Build?

5 API & Web-Based Projects

- Building REST APIs using C# Web API
- Consuming APIs from a C# desktop app
- Weather, currency, stock data apps

6 Cloud & Distributed Systems

- Microservices communication
- Message queues in C#
- Push notification service

Live Demo

- Create new console app as “TcpServerDemo”
- Include following namespaces:
  - System — Basic C# features
  - System.Net — IP address handling
  - System.Net.Socket — TCP/UDP communication
  - System.Text — Convert text ↔ bytes
- Create new console app as “TcpClientDemo”

Common Questions

- ❓ Why bytes?
  - Network only understands **binary data**
- ❓ Why server runs first?
  - Server must be listening before client connects
- ❓ Why port 5000?
  - Any free port (1024–65535)
- ❓ Can multiple clients connect?
  - Yes — using **threads or async**, which comes next