

UNIT 1

Introduction

(3 Hours) — Network Programming (CAC355)

Disclaimer: These notes were generated automatically from the CACS355 syllabus and may contain errors or omissions. Please verify important details (syntax, method names, default values) against your textbook (Java Network Programming, E. R. Harold) and class notes before the exam.

1.1 Network Programming Features and Scope

Network programming is the writing of programs that communicate with other programs across a computer network. The programs may run on different machines (or on the same machine) and exchange data using agreed rules called **protocols**. Examples include web browsers, chat applications, email clients, online games and file-transfer tools.

Main features

- **Inter-process communication over a network** — processes on different hosts exchange messages.
- **Protocol based** — communication follows standards such as TCP, UDP, IP, HTTP, FTP, SMTP.
- **Client-server or peer-to-peer** — roles of the communicating programs are defined by the design.
- **Concurrency** — servers must handle many clients at once (threads, non-blocking I/O).
- **Reliability and error handling** — networks can lose, delay or reorder data, so programs must handle timeouts and failures.
- **Security** — data in transit must be protected (authentication, encryption, SSL/TLS).

Scope

- Web applications and web services (HTTP clients and servers).
- Email, messaging, chat, VoIP and video streaming.
- Distributed systems, cloud services and remote method invocation.
- Network utilities: port scanners, monitors, file transfer, remote administration.
- Networked games and Internet of Things (IoT) devices.

1.2 Network Programming Language, Tools and Platforms

Almost any language can be used for network programming, but each has its own strengths. This course uses **Java** because the `java.net`, `java.io` and `java.nio` packages provide a rich, platform-independent networking API.

Language	Networking support	Remark
Java	<code>java.net</code> , <code>java.nio</code> , RMI, <code>javax.net.ssl</code>	Platform independent, built-in threads, used in this course
C / C++	BSD sockets, Winsock	Low level, fast, more manual work
Python	<code>socket</code> , <code>asyncio</code> , <code>requests</code>	Short code, good for scripts and prototypes
C# / .NET	<code>System.Net.Sockets</code>	Strong on Windows platforms
JavaScript (Node.js)	<code>net</code> , <code>http</code> , <code>WebSocket</code>	Event driven, web-oriented

Common tools

- **JDK** (compiler `javac`, runtime `java`) and IDEs such as NetBeans, Eclipse, IntelliJ IDEA.
- **telnet / netcat (nc)** — connect manually to a port to test a protocol.
- **Wireshark / tcpdump** — packet capture and analysis.
- **ping, traceroute, netstat, nslookup** — diagnostic commands.
- **Postman / curl** — send HTTP requests for testing.

Platforms

Java programs run on the Java Virtual Machine (JVM), so the same class files work on Windows, Linux and macOS (“write once, run anywhere”). Network programs can also target servers, desktops, mobile devices and embedded systems.

1.3 Client and Server Applications

Client

A **client** is a program that **initiates** communication and requests a service. It is usually short-lived, interacts with the user, and knows the address of the server. Examples: web browser, email client, FTP client.

Server

A **server** is a program that **waits** for incoming requests, performs the requested work and returns a response. It normally runs continuously, listens on a well-known port and may serve many clients at the same time. Examples: web server (port 80), SMTP server (port 25), DNS server (port 53).

Aspect	Client	Server
Role	Requests a service	Provides a service
Start of communication	Initiates the connection	Waits (listens) for connections
Lifetime	Short, runs while user needs it	Long, usually runs continuously
Address knowledge	Must know the server address and port	Learns the client address from the request
Concurrency	Usually one server at a time	Many clients at once

Example — Common well-known ports

HTTP 80, HTTPS 443, FTP 21, SSH 22, Telnet 23, SMTP 25, DNS 53, POP3 110, IMAP 143.

1.4 Client–Server Model and Software Design

In the **client–server model**, the work of an application is divided between service requesters (clients) and service providers (servers). Clients send requests over the network, and the server sends back replies.

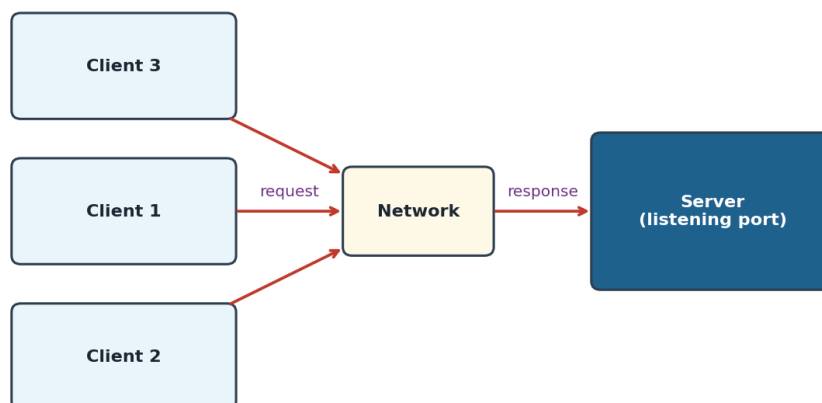


Figure 1.1 — Many clients communicating with a server over a network

Types of servers

- **Iterative server** — handles one client at a time; the next client waits. Simple but slow under load.
- **Concurrent server** — creates a new thread or process for each client (or uses a thread pool), so many clients are served together.
- **Connection-oriented server** — uses TCP; a connection is set up before data transfer.
- **Connectionless server** — uses UDP; each datagram is independent.
- **Stateful vs stateless** — a stateful server remembers client information between requests; a stateless server (e.g., plain HTTP) does not.

Architectures

Architecture	Description	Example
Two-tier	Client talks directly to the server that holds data and logic	Desktop app with a database
Three-tier	Presentation, application logic and data are separate tiers	Web app: browser, app server, database
N-tier	Logic split across several layers or services	Enterprise systems
Peer-to-peer	Every node is both client and server	BitTorrent

Basic design steps of a network application

- Decide the **protocol**: message format, order of messages, error handling.
- Choose the **transport**: TCP for reliability, UDP for speed and low overhead.
- Decide the **server type**: iterative or concurrent.
- Handle **exceptions and timeouts**, and close resources properly.
- Add **logging and security** (authentication, SSL/TLS).

Example — Skeleton of a TCP client and server in Java

Both sides use the Socket classes in java.net; the server accepts, the client connects.

```
// Server
ServerSocket server = new ServerSocket(5000);
Socket s = server.accept();           // waits for a client
// ... read/write using s.getInputStream() / s.getOutputStream()
s.close();

// Client
Socket c = new Socket("localhost", 5000); // connects to server
// ... read/write using c.getInputStream() / c.getOutputStream()
c.close();
```

Listing 1.1 — Minimal socket skeleton (details in Units 6 and 7)

Key Takeaways

- ✓ Network programming lets programs on different hosts exchange data using protocols.
- ✓ Java is used here because `java.net` and `java.nio` provide portable networking APIs.
- ✓ A client initiates requests; a server listens and responds, often to many clients at once.
- ✓ Servers can be iterative or concurrent, connection-oriented (TCP) or connectionless (UDP).
- ✓ Good design defines the protocol, transport, concurrency model, error handling and security.

— *End of Unit 1* —