Computer Network
Socket Programming Project

2025.10.07

Outline

- Project Description
- Tasks & Grading
- Phase 1
- Phase 2
- Requirements & Submission
- Introduction to Socket Programming

Project Description

- The project is a **real-time online chatroom** application that allows users to communicate through various media types, including **text, files, and live video streaming**. The system is designed to support seamless and **secure communication** in both private and public chat modes.

Tasks & Grading

- Phase 1 (40pts)
  - Basic Server-Client Communication (20pts)
  - Authentication Features (20pts)
- Phase 2 (60pts + 20pts bonus)
  - Multithread server (15pts)
  - Sending Chat Messages (15pts)
  - Message Encryption with OpenSSL (10pts + **5pts bonus**)
  - Group chatting (10pts + **5pts bonus**)
  - Transfer files (10pts)
  - Audio/Video Streaming (**10pts bonus**)

Phase 1

Basic Server-Client Communication (20pts)

- Server (10pts)
  - Server can receive messages from the client.
  - Server can respond to client messages.
- Client (10pts)
  - Client can send messages to the server.
  - Client can receive responses from the server.
- In Phase 1, implementing a thread pool / multithreaded server is **not required**.

Authentication Features (20pts)

- User Registration (5pts)
- User Login (5pts)
  - During login, the client must provide its listening port to the server.
- User Logout (5pts)
- List Online Users (5pts)
- Full credit requires basic exception handling (e.g., unknown commands, duplicate registration/login, invalid port numbers).

Phase 2

Multithread server (15pts)

- Create a high-performance multithreaded server that can handle up to 10 concurrent connections using a worker pool approach.
- This server should be implemented using **POSIX threads (pthread) and worker pool design pattern**.
- **Do not** use fork() for process creation.

Sending Chat Messages (15pts)

- Sending Chat Messages (15pts)
  - The system must implement **P2P chatting**.
  - Messages are sent directly from client to client; the server is used **only for discovery** (online list, IP/port lookup).
- Hints
  - Client should create a listening socket to receive incoming messages.
  - Clients should inform the server about their listening port upon connection.
  - Clients should obtain other clients' IP addresses and ports from the server.
- Requirements
  - While in the chatroom, both sides can exchange messages.
  - Full credit requires basic exception handling

Message Encryption with OpenSSL (15pts)

- Base requirement — 10 pts
  - Implement a secure communication system for both client-to-client and client-to-server interactions.
  - Symmetric and asymmetric encryption are both accepted.
- Bonus — 5 pts (session key freshness)
  - Each connection **establishes a fresh symmetric session key** that's **unknown to anyone except the two peers**.
  - You may take inspiration from TLS key establishment.

Group chatting (15pts)

- Basic — 10 pts
  - Implement a **group chatroom (relay mode)**.
  - Clients send messages to the **server**, which **relays** them to all members of the room.
  - Messages should appear **in order.**
  - **Encryption is required.**
- Bonus — 5 pts (privacy + rekey on membership change)
  - The **server must not be able to decrypt message content**

    (metadata such as room ID, sender ID, timestamps may remain in cleartext).
  - **Change the group key whenever membership changes** (join or leave).

Transfer files (10pts)

- Implement a file transfer feature based on existing communication system.
- Requirements
  - Send and receive files in chunks.
  - Receiver must be able to download the file after transfer.
  - **Need encryption with OpenSSL**
  - You are only allowed to use standard library and POSIX library.
  - Transmission does not have to occur inside the chatroom.

Audio/Video Streaming (10pts)(Bonus)

- Develop a frame-based streaming feature for audio or video files.
- Requirements
  - Sender: Implement file selection and frame-based transmission.
  - Receiver: Implement frame reception, reassembly, and **real-time** playback/display.
  - You are allowed to use any library.
  - **Encryption is optional.**
  - Implement either video **or** audio streaming.
  - Transmission does not have to occur inside the chatroom.

Requirements & Submission

Requirements

- Only C / C++ (Unix/Linux Socket Programming)
  - For windows user, you can use Win Socket or install WSL.
- Accepted Library for socket programming
  - #include <sys/socket.h>
  - #include <netinet/in.h>
  - #include <arpa/inet.h>
  - #include <winsock2.h>
- **No plagiarism**. Both the plagiarist and the original author will receive a score of 0 if plagiarism is detected.
- Team
  - Phase 1: Only individual submissions are allowed.
  - Phase 2: 1~2 people each team.

Submission

- Phase 1 (Deadline: **2025/11/4 23:59**)
  - **< 5 minutes video (including demo and code explanation)**
  - Your code
  - README file
- Phase 2 (Deadline: **2025/12/16 23:59**)
  - **< 10 minutes video (including demo and code explanation)**
  - Your code
  - README file
- Your README file must include
  - Compilation instructions
  - Usage guide
  - Any additional information necessary to run your project

Submission

- Submit your code and YouTube Link on NTU COOL
- For both Phase 1 and Phase 2, submit a zip file named {student_number}.zip (e.g. b12902666.zip) containing:
  - b12902666/
    - README.md
    - code/
      - xxx.cpp
      - xxx.cpp
      - (other files or folders)
- If there is an incorrect file name or format, **10 points will be deducted.**
- Late Submission is **NOT ALLOWED**.

FAQ

- 繳交格式一定要按照規定嗎？
  - 是，請將所有程式碼放到 code 資料夾底下，code 資料夾底下沒有規定排列方式
- 超過 100 分會以實拿分數計算嗎
  - 超過 100 分會以 100 分計算，但加分題可以提供你不同的拿分策略，或是挑戰更高的完成度。
- Demo 影片要搭配解說嗎
  - Phase 1 請搭配文字說明或是講解，Phase 2 請完整講解功能及實作方式
- Phase 1 當server process結束之後需要保留註冊者資訊嗎？
  - 沒有特別規定，可以依照同學的實作方式自行決定

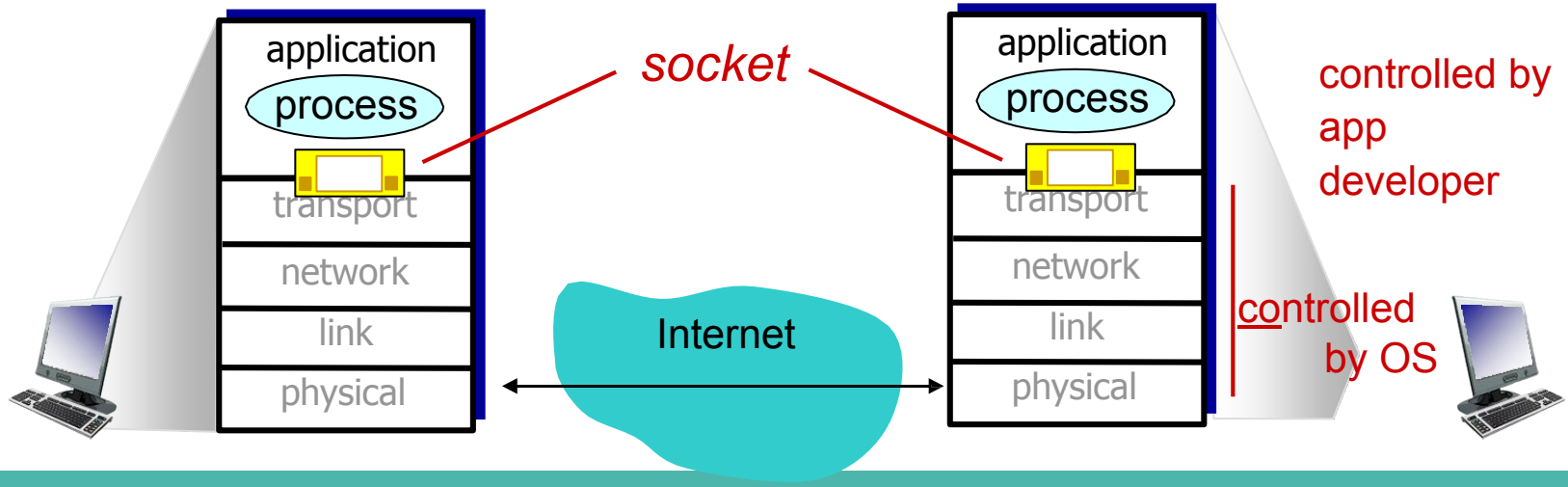
FAQ

- 可以假設 client 都有 public IP 嗎？
  - 可以假設 client 在相同的 Internet 底下，都可以透過 IP Address 互相通訊。
- 如果為兩人一組有需要在哪邊註明嗎？
  - 請在 README 以及影片中註明同組的所有同學們，作業繳交只需要由其中一位上傳即可。
- Phase 2 影片要有 code explanation，code explanation 要針對那些功能講解呢？
  - 請針對 Phase 2 新增的所有功能進行講解，需要投影程式碼的部分搭配講解，讓助教能 夠知道實作方式，不需要逐行講解，只要針對流程做大概的講解。

Introduction to Socket Programming

Socket

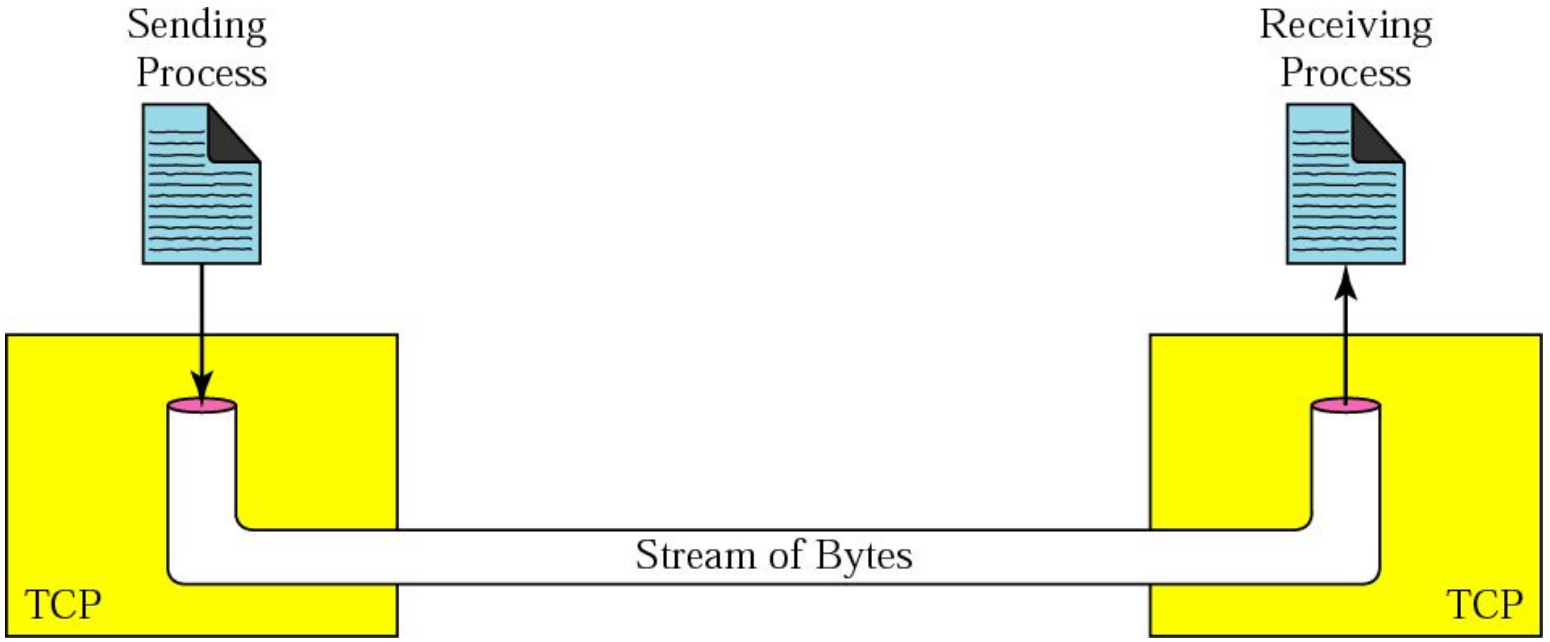
- Process sends/receives messages to/from its socket
- Socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



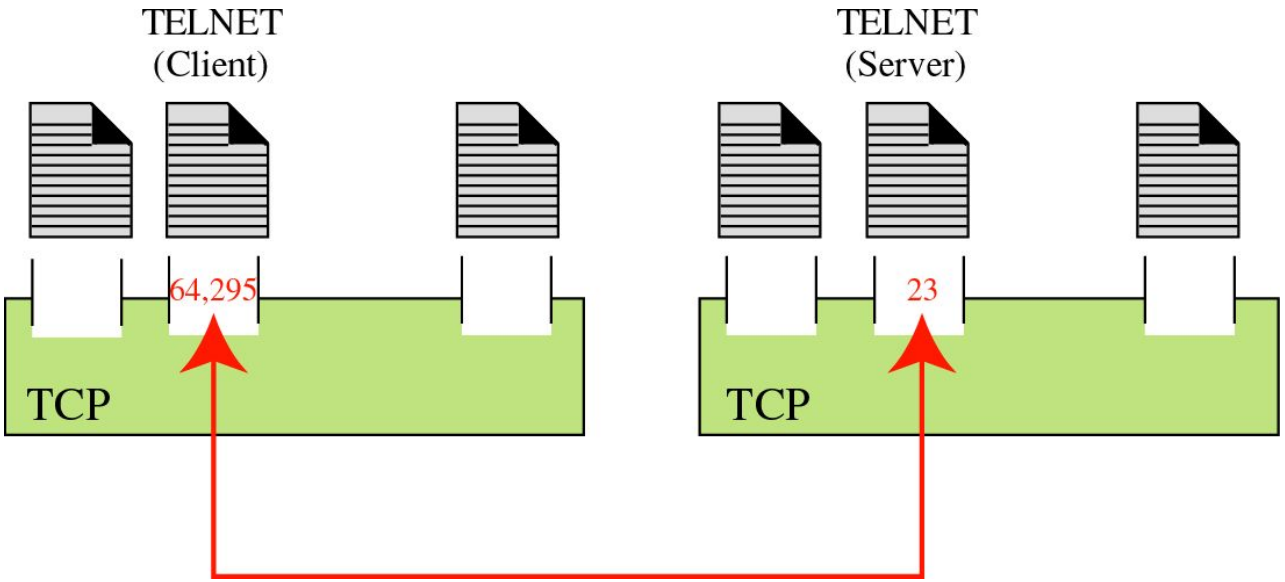
Socket Programming

- **Goal:** learn how to build client/server applications that communicate using sockets
- **Socket:** **door** between application process and end-end-transport protocol
- Two socket types for two transport services:
  - **UDP:** unreliable datagram
  - **TCP:** reliable, byte stream-oriented

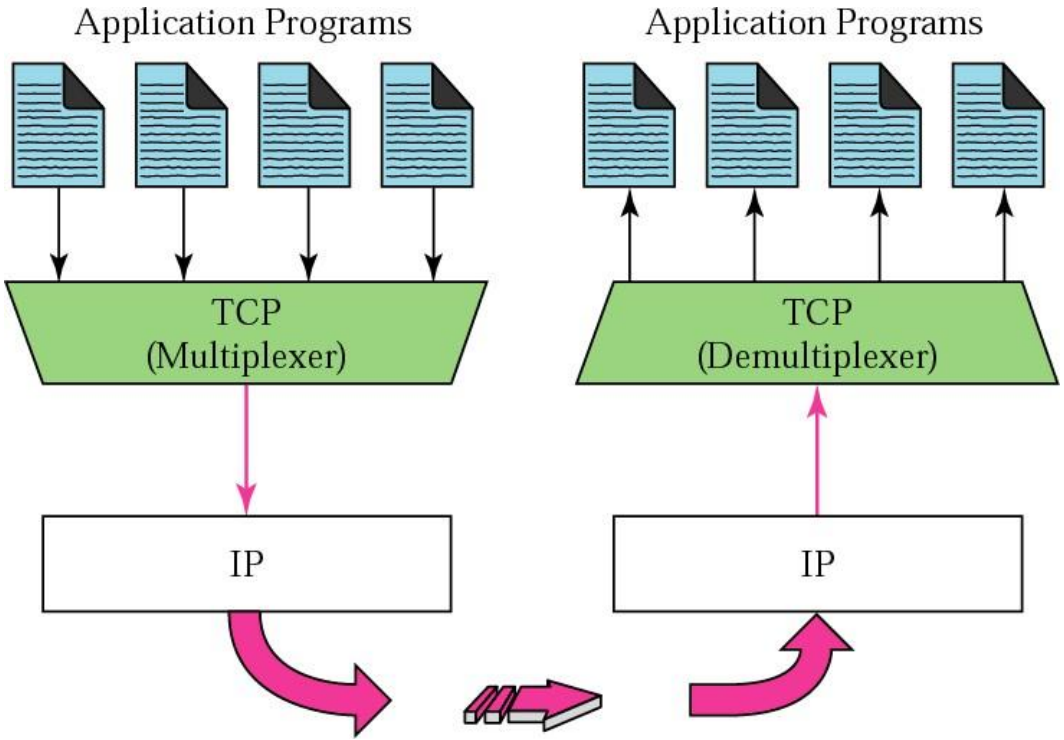
Stream Delivery



Port Numbers



Multiplexing and Demultiplexing



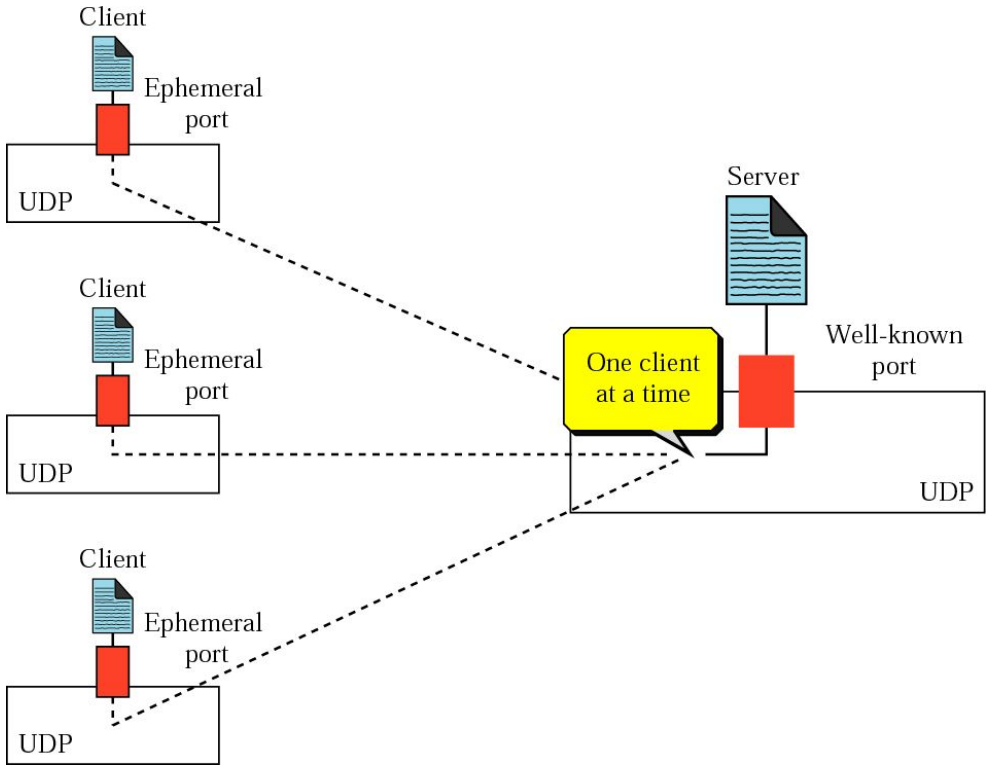
Socket Programming with UDP

- UDP: no connection between client & server
  - **no** handshaking before sending data
  - **sender** explicitly attaches **IP dest**ination address and **port** number to each packet
  - **receiver** extracts sender **IP addr**ess and **port** number from received packet
- UDP: transmitted data may be **lost** or received **out-of-order**
- Application viewpoint:
  - UDP provides **unreliable data transfer** of groups of bytes ("datagrams") between client and server

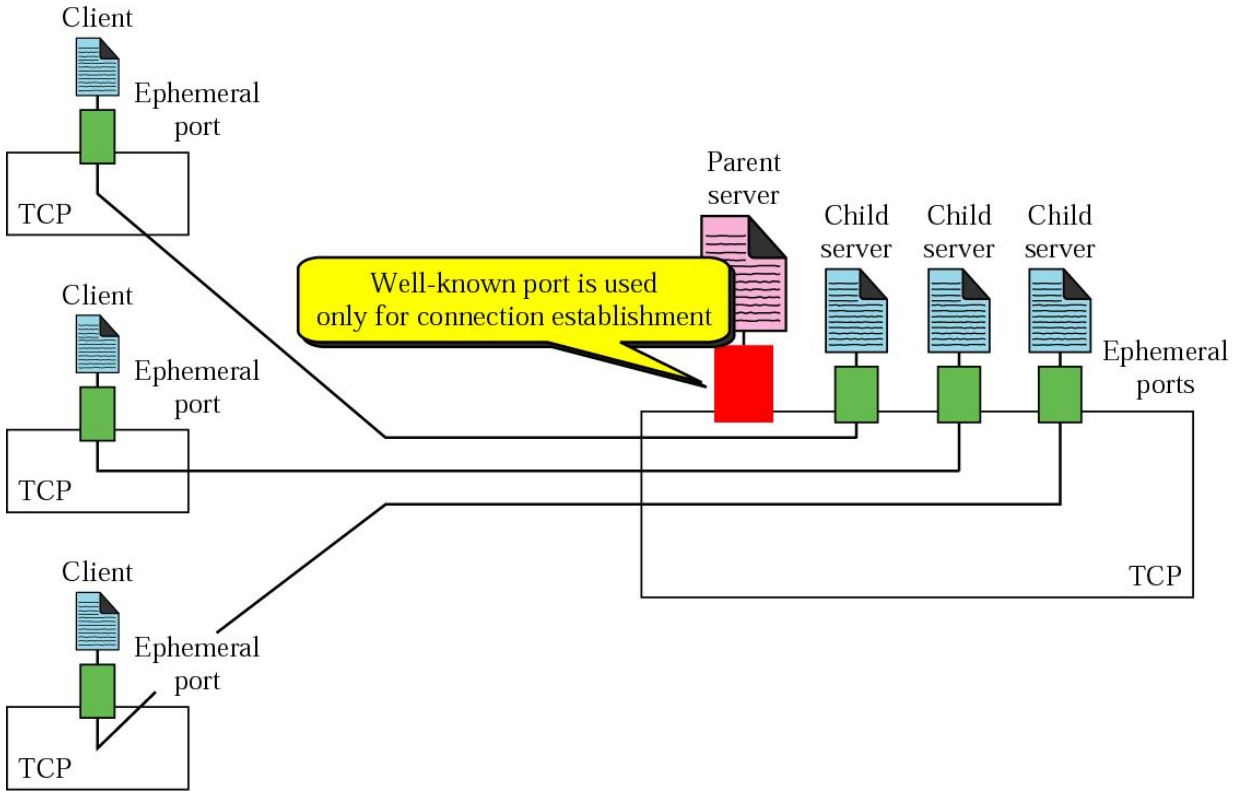
Socket programming with TCP

- client must contact server
  - server process **must first be running**
  - server must have created **socket** (door) that **welcomes client's** contact
- client contacts server by:
  - Creating **TCP socket**, specifying **IP addr**ess, **port** number of server process
  - When client creates a socket, a **TCP connection is established** between client and server
- when contacted by client, server TCP creates new socket for server process to communicate with that particular client
  - allows server to talk **with multiple clients**
  - **source port** number & **source IP** used to distinguish clients (more in Chap 3)
- Application viewpoint:
  - TCP provides **reliable**, **in-order byte-stream** transfer ("pipe") between client and server

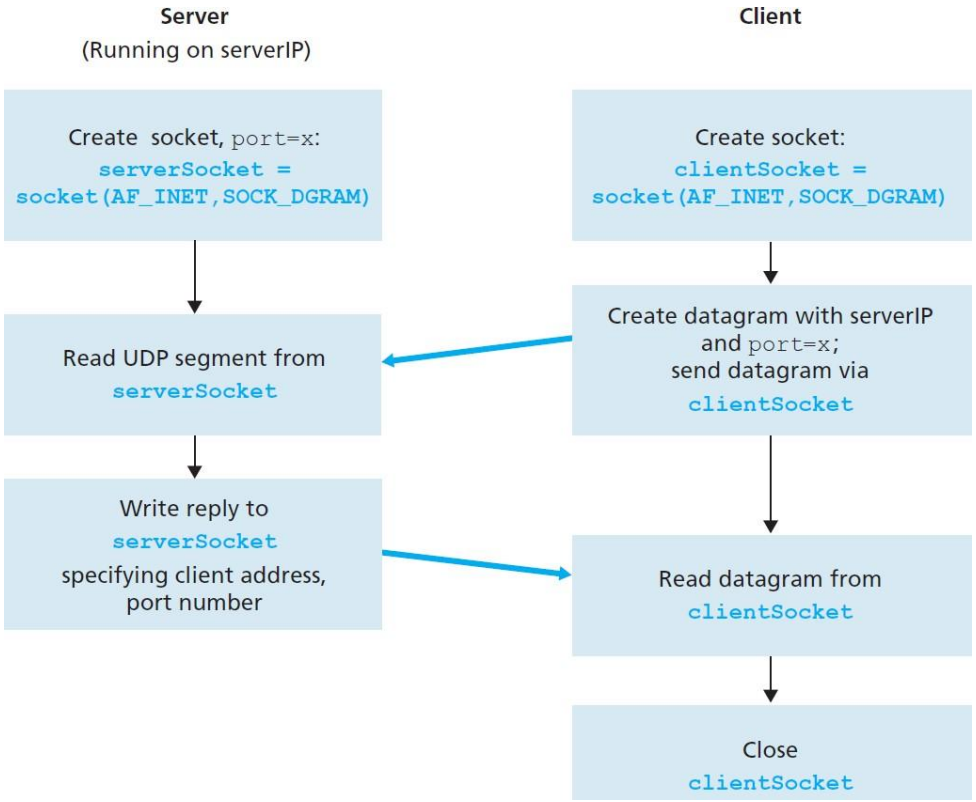
Connectionless iterative server (UDP)



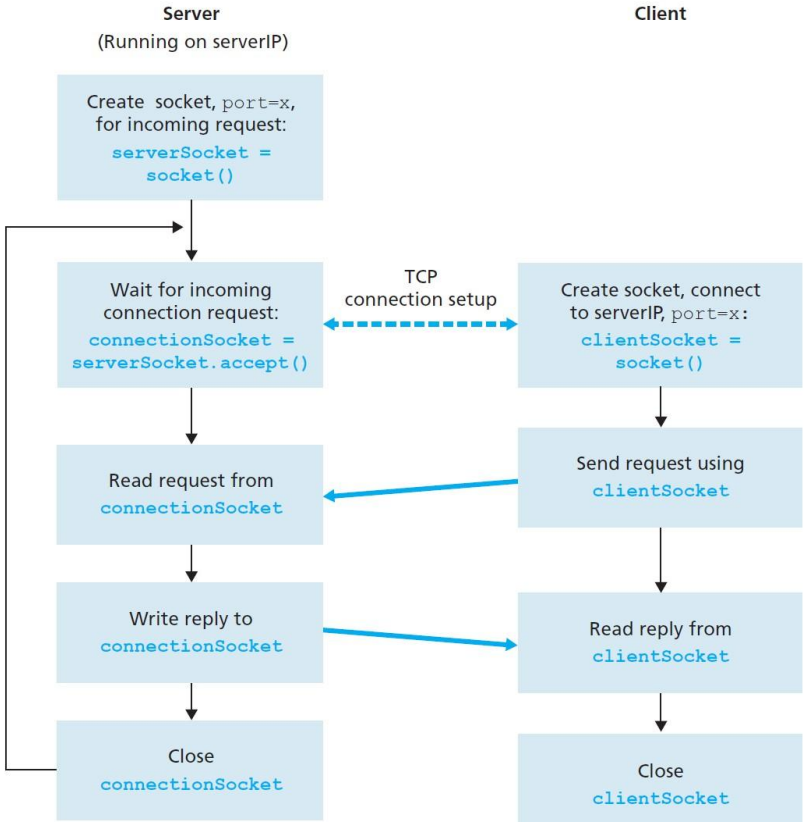
Connection-oriented concurrent server (TCP)



Client/server socket interaction: UDP

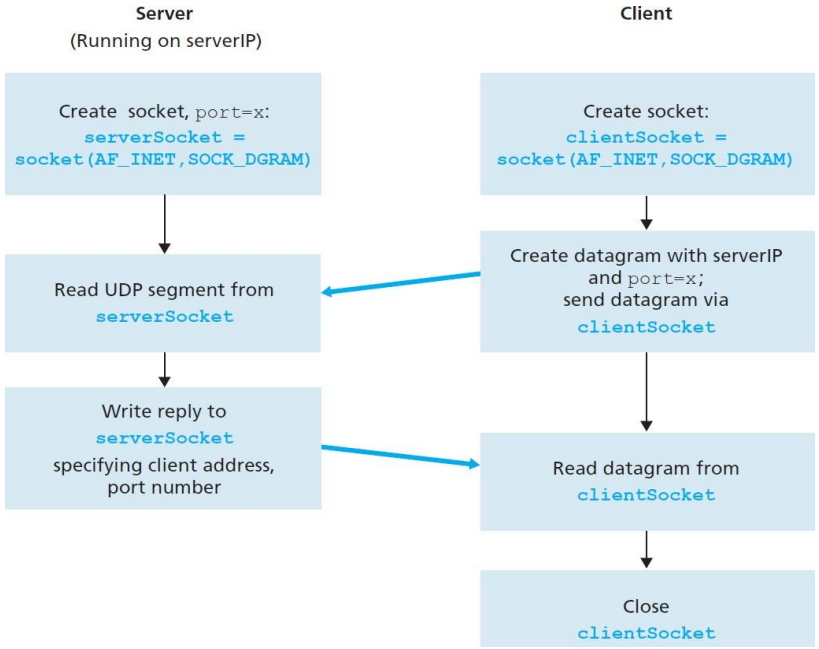


Client/server socket interaction: TCP



Client/server socket interaction

UDP



TCP