INSTITUTE OF VISUAL COMPUTING

TU Graz
SCIENCE
PASSION
TECHNOLOGY

Object-Oriented Programming 2: Lecture 4
Networking & Socket Programming

Tobias Schreck, Benedikt Kantz, Paul Raith

Intro

- Goals of today:
  - Networking Basics
  - The Client-Server Model
  - TCP vs UDP
  - Examples
  - Chat Server Flipped Classroom example
- Low-Level details will be presented in the Computer Organization and Networks lecture!

What is Networking?

What is Networking?

A collection of connected devices that can communicate with one another and share resources and information.

- **Nodes** are any connected devices which send/receive or forward data (e.g. computer, server, printer, ...).
- **Networking Devices** manage and support networking functions (e.g. routers, switches, access points, ...).
- **Transmission Media:** The physical or wireless medium through which data travels.

What is Networking?

A collection of connected devices that can communicate with one another and share resources and information.

- **Nodes** are any connected devices which send/receive or forward data (e.g. computer, server, printer, ...).
- **Networking Devices** manage and support networking functions (e.g. routers, switches, access points, ...).
- **Transmission Media:** The physical or wireless medium through which data travels.

What is Networking?

A collection of connected devices that can communicate with one another and share resources and information.

- **Nodes** are any connected devices which send/receive or forward data (e.g. computer, server, printer, ...).
- **Networking Devices** manage and support networking functions (e.g. routers, switches, access points, ...).
- **Transmission Media:** The physical or wireless medium through which data travels.

What is Networking?

A collection of connected devices that can communicate with one another and share resources and information.

- **Nodes** are any connected devices which send/receive or forward data (e.g. computer, server, printer, ...).
- **Networking Devices** manage and support networking functions (e.g. routers, switches, access points, ...).
- **Transmission Media:** The physical or wireless medium through which data travels.

Networking Layers

OSI Model vs TCP/IP Model

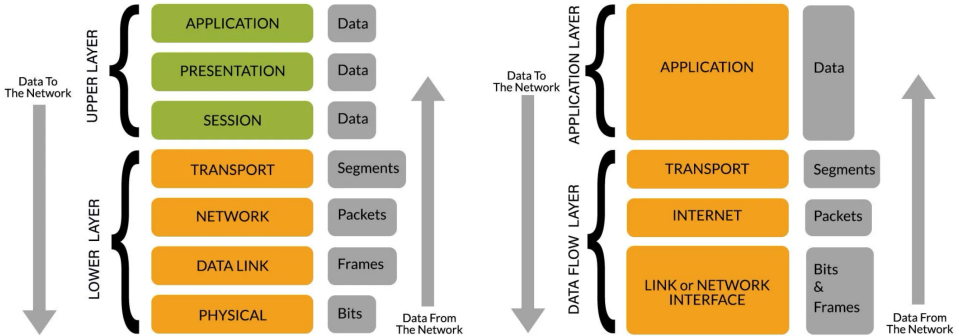


Figure: https://www.rtautomation.com/rtas-blog/a-refresher-course-on-osi-tcp-ip/

Networking Layers

- **Application Layer:**
  - What the user directly interacts with (e.g., HTTP for web, FTP for file transfer, our Chat Protocol).
  - Handles data formatting, encryption (if implemented).
- **Transport Layer:**
  - Provides end-to-end communication services.
  - **TCP vs. UDP:** *We use TCP via Java Sockets.*

Networking Layers

- **Application Layer:**
  - What the user directly interacts with (e.g., HTTP for web, FTP for file transfer, our Chat Protocol).
  - Handles data formatting, encryption (if implemented).
- **Transport Layer:**
  - Provides end-to-end communication services.
  - **TCP vs. UDP:** *We use TCP via Java Sockets.*

Networking Layers

- **Internet Layer:**
  - Handles addressing and routing data across different networks (e.g., IP addresses).
  - Finds the best path for data packets.
- **Network Interface Layer:**
  - Deals with the physical transmission of data on a specific network medium (Ethernet, Wi-Fi, fiber).
  - Translates IP packets into actual electrical/light signals.

Networking Layers

- **Internet Layer:**
  - Handles addressing and routing data across different networks (e.g., IP addresses).
  - Finds the best path for data packets.
- **Network Interface Layer:**
  - Deals with the physical transmission of data on a specific network medium (Ethernet, Wi-Fi, fiber).
  - Translates IP packets into actual electrical/light signals.

Networking Layers

What we care about in this lecture:

- **Application Layer:** Our Chat Protocol (the messages we define).
- **Transport Layer:** We *use* TCP via Java's `Socket` class.
- **The Rest:** Handled by the OS & network hardware. We don't touch it! :D

Networking Layers

What we care about in this lecture:

- **Application Layer:** Our Chat Protocol (the messages we define).
- **Transport Layer:** We *use* TCP via Java's `Socket` class.
- **The Rest:** Handled by the OS & network hardware. We don't touch it! :D

Networking Layers

What we care about in this lecture:

- **Application Layer:** Our Chat Protocol (the messages we define).
- **Transport Layer:** We *use* TCP via Java's `Socket` class.
- **The Rest:** Handled by the OS & network hardware. We don't touch it! :D


WHAT IS YOUR ADDRESS?
173.168.16.11
NO, YOUR LOCAL ADDRESS
127.0.0.1
I MEAN YOUR PHYSICAL ADDRESS !
28:05:FF:58:31:05

Client-Server Model Architecture

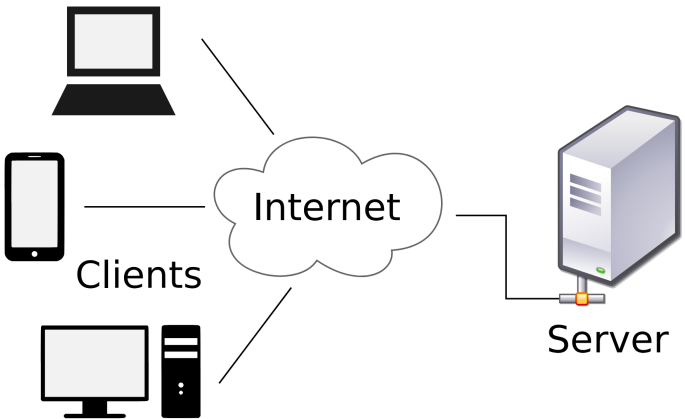


Figure: https://commons.wikimedia.org/wiki/File:Client-server-model.svg

Client-Server Model Architecture

- A central, "always-on" server provides data or services to multiple clients that initiate requests.
- Efficiently centralizes resources, data, and management, allowing many simple clients to share a single, powerful, and (hopefully) secure application.
- **Server**: Listens for and responds to requests (e.g., a web server, our chat server).
- **Client**: Initiates requests (e.g., your web browser, your chat client).

Client-Server Model Architecture

- A central, "always-on" server provides data or services to multiple clients that initiate requests.
- Efficiently centralizes resources, data, and management, allowing many simple clients to share a single, powerful, and (hopefully) secure application.
- **Server**: Listens for and responds to requests (e.g., a web server, our chat server).
- **Client**: Initiates requests (e.g., your web browser, your chat client).

Client-Server Model Architecture

- A central, "always-on" server provides data or services to multiple clients that initiate requests.
- Efficiently centralizes resources, data, and management, allowing many simple clients to share a single, powerful, and (hopefully) secure application.
- **Server**: Listens for and responds to requests (e.g., a web server, our chat server).
- **Client**: Initiates requests (e.g., your web browser, your chat client).

Client-Server Model Architecture

- A central, "always-on" server provides data or services to multiple clients that initiate requests.
- Efficiently centralizes resources, data, and management, allowing many simple clients to share a single, powerful, and (hopefully) secure application.
- **Server**: Listens for and responds to requests (e.g., a web server, our chat server).
- **Client**: Initiates requests (e.g., your web browser, your chat client).

Client-Server Model Architecture

- A central, "always-on" server provides data or services to multiple clients that initiate requests.
- Efficiently centralizes resources, data, and management, allowing many simple clients to share a single, powerful, and (hopefully) secure application.
- **Server**: Listens for and responds to requests (e.g., a web server, our chat server).
- **Client**: Initiates requests (e.g., your web browser, your chat client).

Key Networking Terminology

- **IP Address:** Identifies the **machine** (e.g., `127.0.0.1` is `localhost`).
- **Port Number:** Identifies the specific **application/process** on that machine.
  - Common "well-known" ports: 80 (HTTP), 443 (HTTPS), 22 (SSH).
- **Socket:** The communication **endpoint** that combines an IP and a port.
  - In Java, this is the **object** we create and use to send and receive data (e.g., `new Socket("10.0.0.5", 6767)`).

Key Networking Terminology

- **IP Address:** Identifies the **machine** (e.g., `127.0.0.1` is `localhost`).
- **Port Number:** Identifies the specific **application/process** on that machine.
  - Common "well-known" ports: 80 (HTTP), 443 (HTTPS), 22 (SSH).
- **Socket:** The communication **endpoint** that combines an IP and a port.
  - In Java, this is the **object** we create and use to send and receive data (e.g., `new Socket("10.0.0.5", 6767)`).

Key Networking Terminology

- **IP Address:** Identifies the **machine** (e.g., `127.0.0.1` is `localhost`).
- **Port Number:** Identifies the specific **application/process** on that machine.
  - Common "well-known" ports: 80 (HTTP), 443 (HTTPS), 22 (SSH).
- **Socket:** The communication **endpoint** that combines an IP and a port.
  - In Java, this is the **object** we create and use to send and receive data (e.g., `new Socket("10.0.0.5", 6767)`).

TCP vs. UDP

- TCP (Transmission Control Protocol)
  - **Connection-Oriented:** A connection is established first, TCP-Handshake.
  - **Reliable & Ordered:** Guarantees that messages arrive and in right order, can be slower.
  - **Use Case:** Web Browsing, file transfer, E-Mail, ...

TCP vs. UDP

- TCP (Transmission Control Protocol)
  - **Connection-Oriented:** A connection is established first, TCP-Handshake.
  - **Reliable & Ordered:** Guarantees that messages arrive and in right order, can be slower.
  - **Use Case:** Web Browsing, file transfer, E-Mail, ...
- UDP (User Datagram Protocol)
  - **Connectionless:** "Fire and forget."
  - **Unreliable & Unordered:** Fast, but no guarantees for message arrival.
  - **Use Case:** Video Streaming, Gaming, ...

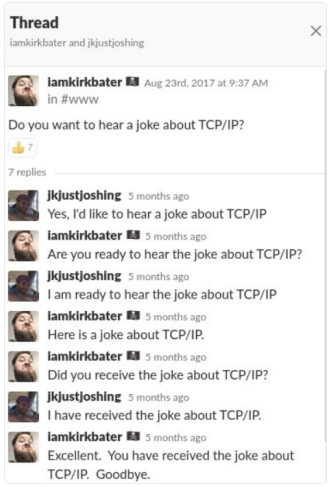
TCP

UDP

Kirk Bater
@KirkBater

Follow

This image is a TCP/IP Joke. This tweet is a UDP joke. I don't care if you get it.



Networking in Java

- Java's `java.net` package provides a "high-level" API for networking.
- It abstracts away the complex details of the OS and network stack.
- **The Big Idea:** We treat the network connection just like any other **I/O Stream**.
- We get an `InputStream` to **read from** the network.
- We get an `OutputStream` to **write to** the network.
- This is very similar to reading/writing files, just over the internet!

Networking in Java

- Java's `java.net` package provides a "high-level" API for networking.
- It abstracts away the complex details of the OS and network stack.
- **The Big Idea:** We treat the network connection just like any other **I/O Stream**.
- We get an `InputStream` to **read from** the network.
- We get an `OutputStream` to **write to** the network.
- This is very similar to reading/writing files, just over the internet!

Networking in Java

- Java's `java.net` package provides a "high-level" API for networking.
- It abstracts away the complex details of the OS and network stack.
- **The Big Idea:** We treat the network connection just like any other **I/O Stream**.
- We get an `InputStream` to **read from** the network.
- We get an `OutputStream` to **write to** the network.
- This is very similar to reading/writing files, just over the internet!

Networking in Java

- Java's `java.net` package provides a "high-level" API for networking.
- It abstracts away the complex details of the OS and network stack.
- **The Big Idea:** We treat the network connection just like any other **I/O Stream**.
- We get an `InputStream` to **read from** the network.
- We get an `OutputStream` to **write to** the network.
- This is very similar to reading/writing files, just over the internet!

Networking in Java

- Java's `java.net` package provides a "high-level" API for networking.
- It abstracts away the complex details of the OS and network stack.
- **The Big Idea:** We treat the network connection just like any other **I/O Stream**.
- We get an `InputStream` to **read from** the network.
- We get an `OutputStream` to **write to** the network.
- This is very similar to reading/writing files, just over the internet!

Networking in Java

- Java's `java.net` package provides a "high-level" API for networking.
- It abstracts away the complex details of the OS and network stack.
- **The Big Idea:** We treat the network connection just like any other **I/O Stream**.
- We get an `InputStream` to **read from** the network.
- We get an `OutputStream` to **write to** the network.
- This is very similar to reading/writing files, just over the internet!

Networking in Java

Server Side:

- `ServerSocket(port)`: Listens for connections on a specific port.
- `Socket clientSocket = serverSocket.accept()`: Waits for a client to connect. This is a blocking call, use a new (Virtual) Thread or Reactive Programming approach!

Networking in Java

Client Side:

- `Socket socket = new Socket(ipAddress, port)`: Connects to the server.

Networking in Java

Client Side:

- `Socket socket = new Socket(ipAddress, port)`: Connects to the server.

How do we communicate?

Networking in Java

Client Side:

- `Socket socket = new Socket(ipAddress, port)`: Connects to the server.

How do we communicate?

- `socket.getInputStream()` → `BufferedReader`: To read text from socket.
- `socket.getOutputStream()` → `PrintWriter`: To write text to the socket.

Advanced Java Networking (For Your Interest Only!)

- **Encrypted Sockets (`SSLSocket`):**
  - `SSLSocket` and `SSLServerSocket` automatically handle encryption (SSL/TLS) → HTTPS!
- **UDP Sockets (`DatagramSocket`):**
  - For connectionless (UDP) communication.
  - You send "datagram packets" – no guarantees they arrive.
- **Non-Blocking I/O (`java.nio`):**
  - `socket.accept()` and `reader.readLine()` are **blocking**. They freeze the thread.
  - This is fine for one client, but terrible for thousands!
  - `java.nio.channels` (New I/O) allows a single thread to "multiplex" and manage thousands of connections.

Advanced Java Networking (For Your Interest Only!)

- **Encrypted Sockets (`SSLSocket`):**
  - `SSLSocket` and `SSLServerSocket` automatically handle encryption (SSL/TLS) → HTTPS!
- **UDP Sockets (`DatagramSocket`):**
  - For connectionless (UDP) communication.
  - You send "datagram packets" – no guarantees they arrive.
- **Non-Blocking I/O (`java.nio`):**
  - `socket.accept()` and `reader.readLine()` are **blocking**. They freeze the thread.
  - This is fine for one client, but terrible for thousands!
  - `java.nio.channels` (New I/O) allows a single thread to "multiplex" and manage thousands of connections.

Advanced Java Networking (For Your Interest Only!)

- **Encrypted Sockets (`SSLSocket`):**
  - `SSLSocket` and `SSLServerSocket` automatically handle encryption (SSL/TLS) → HTTPS!
- **UDP Sockets (`DatagramSocket`):**
  - For connectionless (UDP) communication.
  - You send "datagram packets" – no guarantees they arrive.
- **Non-Blocking I/O (`java.nio`):**
  - `socket.accept()` and `reader.readLine()` are **blocking**. They freeze the thread.
  - This is fine for one client, but terrible for thousands!
  - `java.nio.channels` (New I/O) allows a single thread to "multiplex" and manage thousands of connections.

Examples

- Code examples on Gitlab (subdirectory 04-Networking).
- We will now discuss them, looking into different aspects.
- Feel free to expand/go through it at your own pace at home!

Chat Server Flipped Class Room Example

- You will implement a Chat Client which communicates with a Chat Server.
- Teach Center: Download Communication Protocol & Framework: TeachCenter
- **Try to implement the protocol at home and test/debug your solution next lecture!**

STOP JAVA .COM

GARBAGE COLLECTION IS DATA CRUELTY!

THINK! What's YOUR RAM footprint?

I'D RATHER GO BAREMETAL THAN VIRTUALIZE

The only "Garbage" around here is the Java virtual machine!