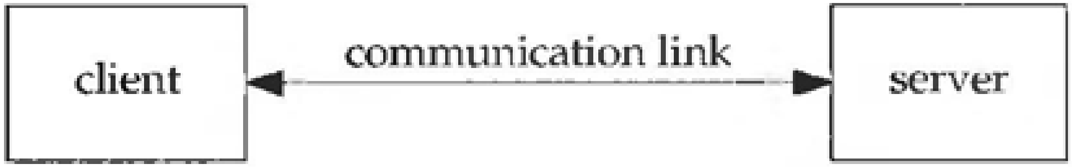
Unix Network Programming

Remote Communication

Dr Hamed Vahdat-Nejad

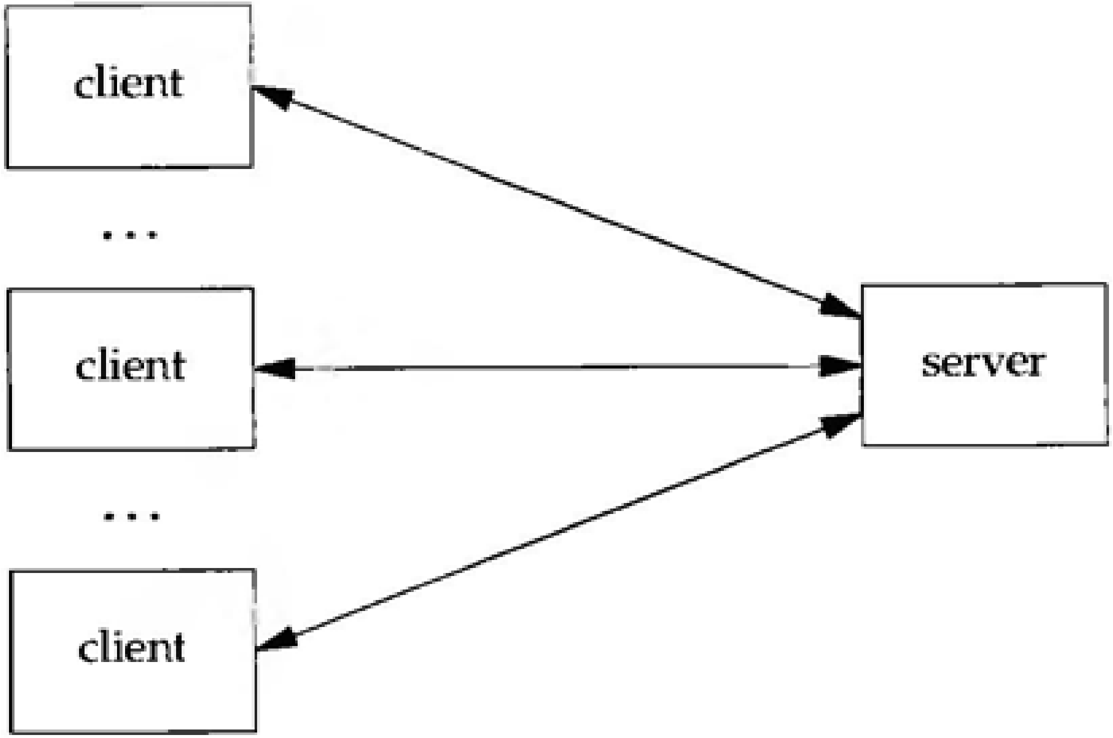
Network Applications

- Types:
  - Client
  - Server

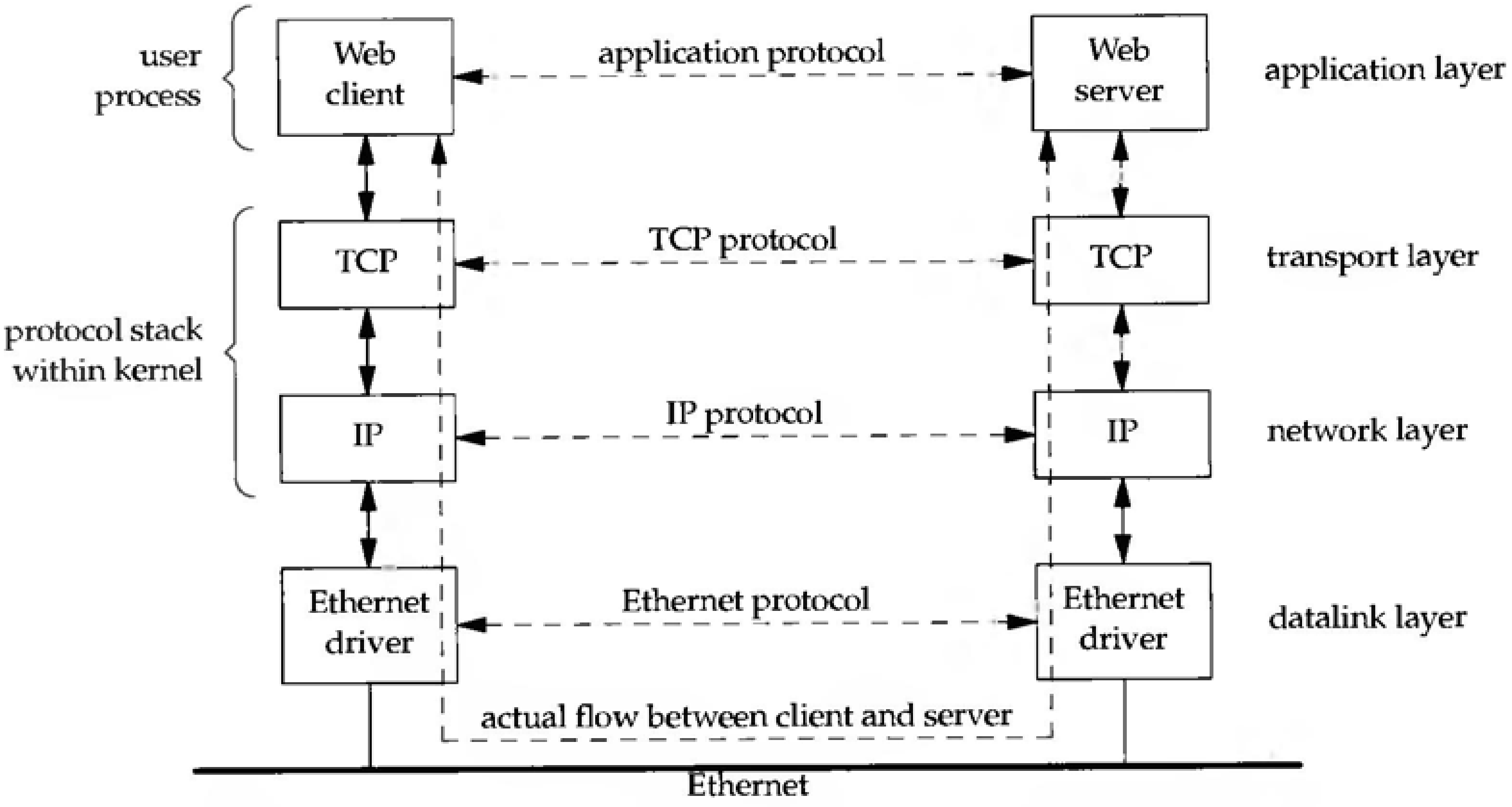


- Exampels:
  - A web browser (client) Ap communicating with a Web server
  - An FTP client Fetching a file from an FTP server
- Protocols:
  - TCP
    - IP

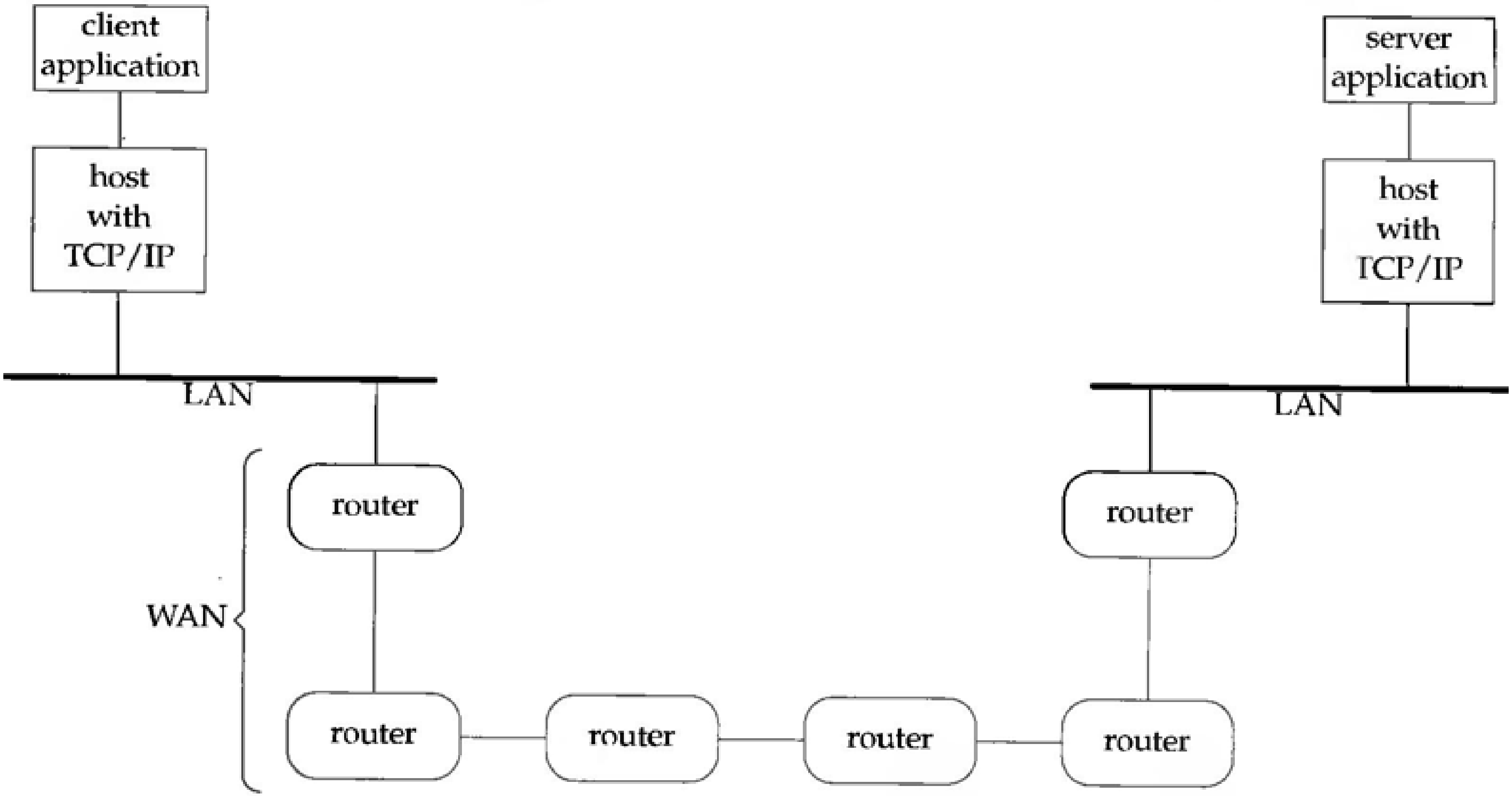
Server handling multiple clients



Client and server on the same Ethernet

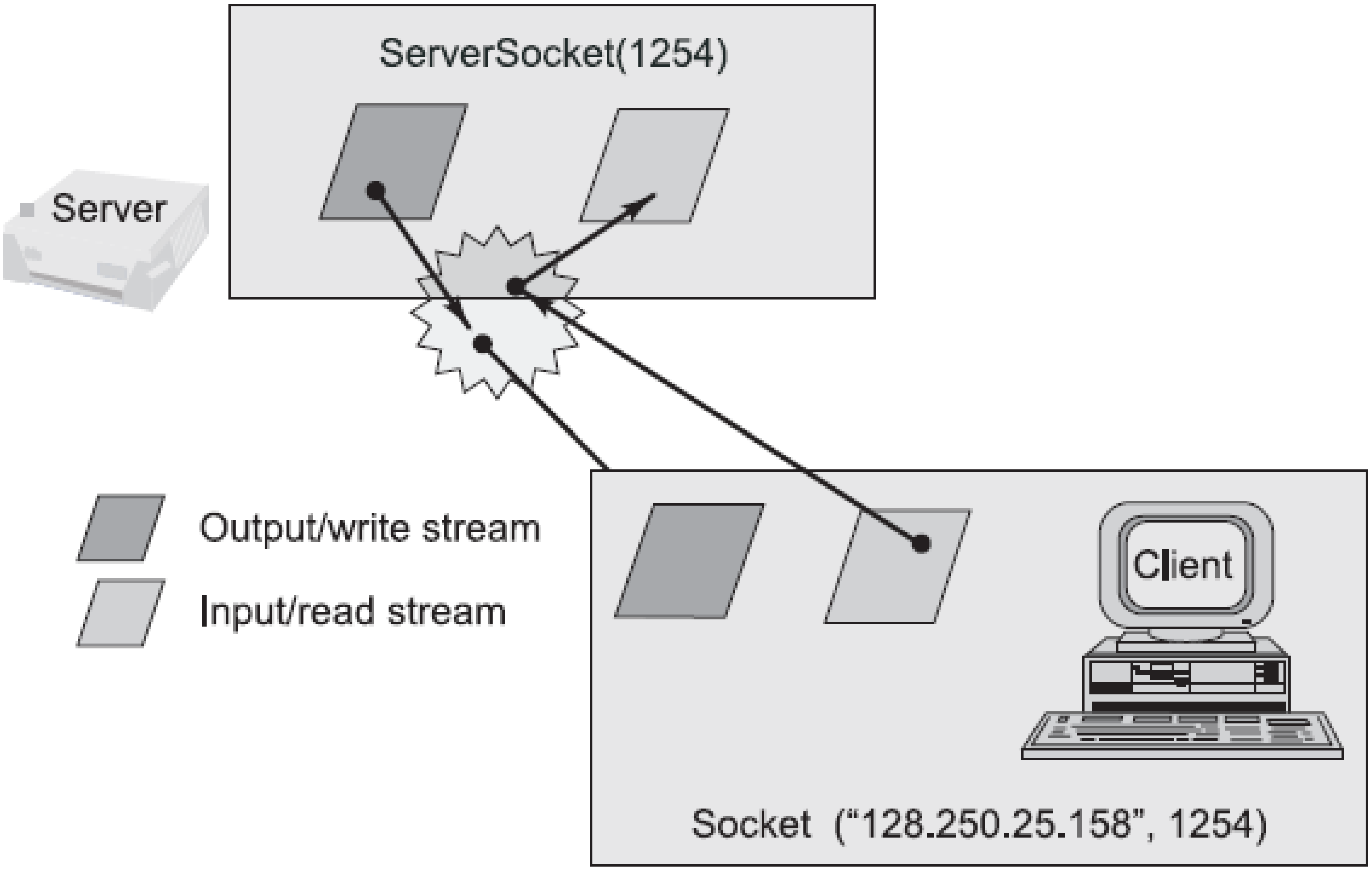


Client and server on different LANs



Java Sockets

- The two key classes from the java.net package used in creation of server and client programs are:
  - ServerSocket
  - Socket
- A server program creates a specific type of socket that is used to listen for client requests (server socket),
- In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams.



ServerSocket(1254)

Server

Output/write stream

Input/read stream

Client

Socket ("128.250.25.158", 1254)

Creating a simple server

1. Open the Server Socket:

```
ServerSocket server = new ServerSocket( PORT );
```

2. Wait for the Client Request:

```
Socket client = server.accept();
```

3. Create I/O streams for communicating to the client

```
DataInputStream is = new DataInputStream(client.getInputStream());
DataOutputStream os = new DataOutputStream(client.getOutputStream());
```

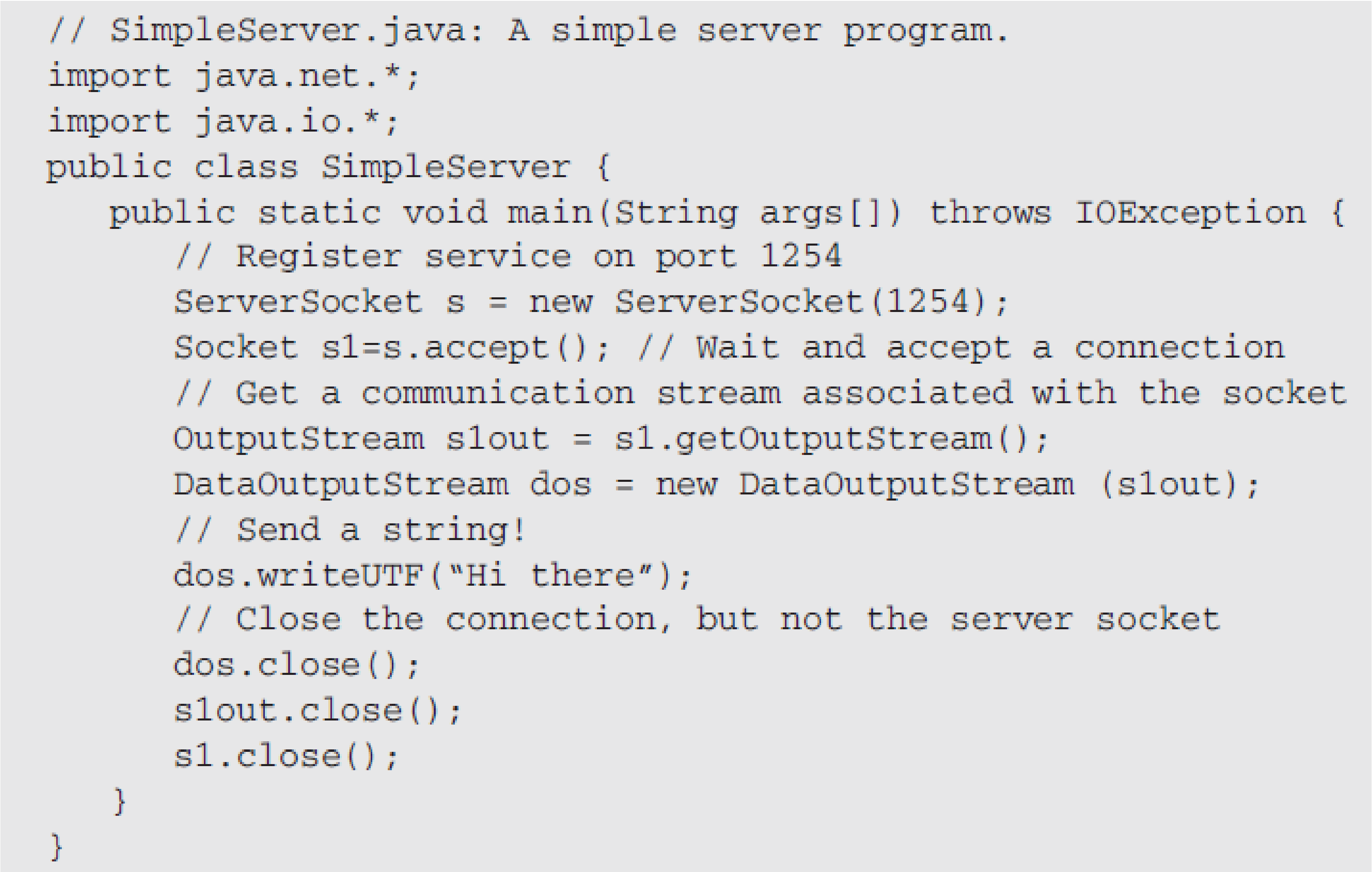
4. Perform communication with client

   Receive from client: String line = is.readLine();

   Send to client: os.writeBytes(“Hello\n”);

5. Close socket:

```
client.close();
```



```
// SimpleServer.java: A simple server program.
import java.net.*;
import java.io.*;
public class SimpleServer {
   public static void main(String args[]) throws IOException {
      // Register service on port 1254
      ServerSocket s = new ServerSocket(1254);
      Socket s1=s.accept(); // Wait and accept a connection
      // Get a communication stream associated with the socket
      OutputStream s1out = s1.getOutputStream();
      DataOutputStream dos = new DataOutputStream (s1out);
      // Send a string!
      dos.writeUTF(“Hi there”);
      // Close the connection, but not the server socket
      dos.close();
      s1out.close();
      s1.close();
   }
}
```

Creating a simple Client

1. Create a Socket Object:

   Socket client = new Socket(server, port_id);

2. Create I/O streams for communicating with the server.

   is = new DataInputStream(client.getInputStream());

   os = new DataOutputStream(client.getOutputStream());

3. Perform I/O or communication with the server:

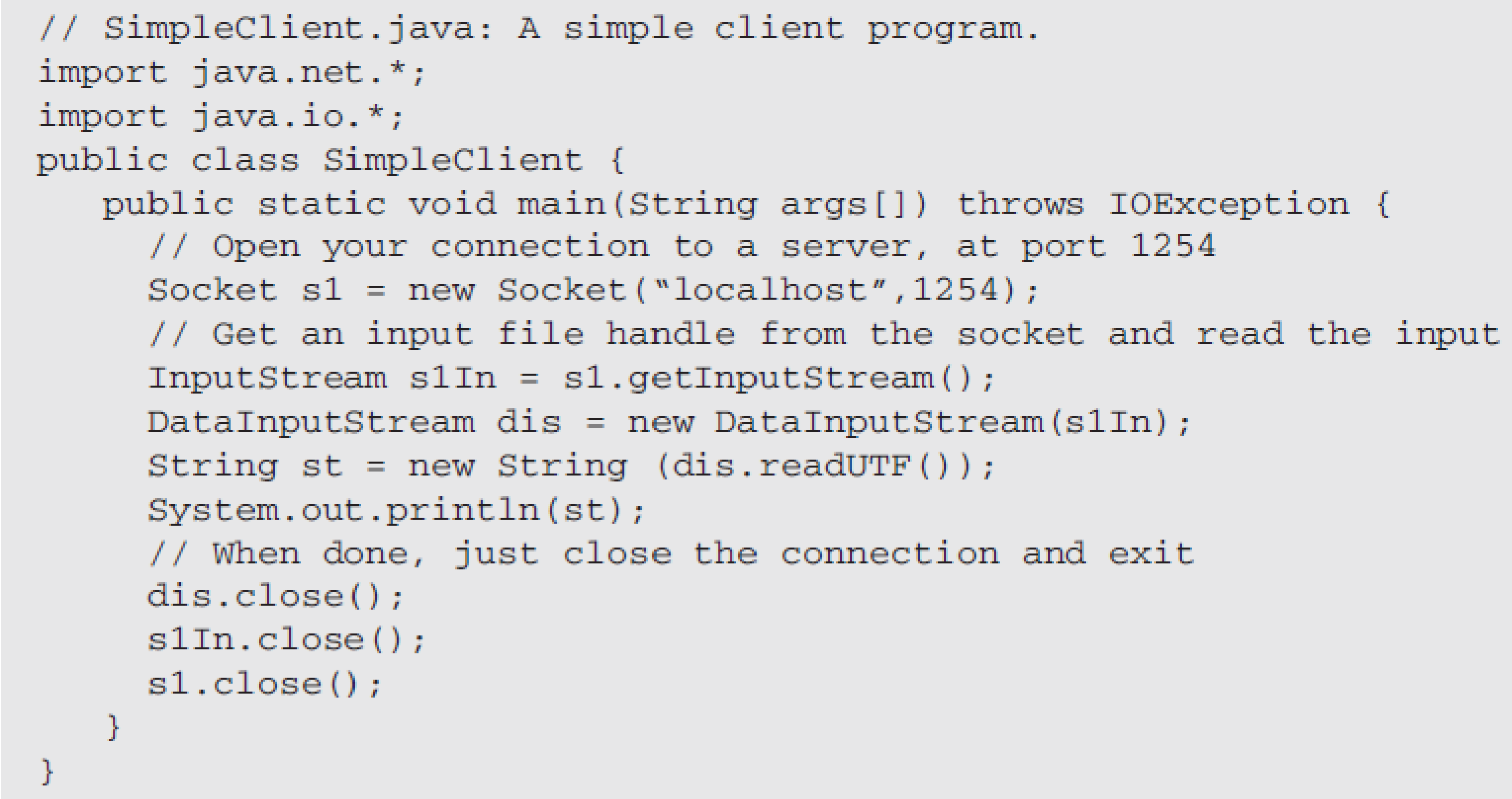
   Receive data from the server: String line = is.readLine();

   Send data to the server: os.writeBytes(“Hello\n”);

4. Close the socket when done:

   client.close();

establishment of connection to a server, reading a message sent by the server and displaying it on the console



```
// SimpleClient.java: A simple client program.
import java.net.*;
import java.io.*;
public class SimpleClient {
   public static void main(String args[]) throws IOException {
     // Open your connection to a server, at port 1254
     Socket s1 = new Socket(“localhost”,1254);
     // Get an input file handle from the socket and read the input
     InputStream s1In = s1.getInputStream();
     DataInputStream dis = new DataInputStream(s1In);
     String st = new String (dis.readUTF());
     System.out.println(st);
     // When done, just close the connection and exit
     dis.close();
     s1In.close();
     s1.close();
   }
}
```

Comparison of Transport protocols

- UDP
  - Simple
  - Unreliable
- TCP
  - Sophisticated
  - Reliable

- TCP & UDP capabilities are provided as APIs or Sockets to the users.
- The TCP and UDP protocols use *ports* to map incoming data to a particular *process* running on a computer.
- Some ports have been reserved to support common/well known services:
  - ftp 21/tcp
  - telnet 23/tcp
  - smtp 25/tcp
  - login 513/tcp
- User-level processes/services generally use port number value >= 1024

2.3 UDP : User Datagram Protocol

- The application writes a ***datagram*** using a UDP socket, which is sent to the destination using IPv4 or IPv6.
- UDP provides a ***connectionless*** service.
- Each UDP datagram has a length and we can consider a datagram as a ***record***.
- No Ack, Sequence#, RTT, Timeout, or Retransmission
- RFC 768

2.4 TCP: Transmission Control Protocol

- Provides ***connections*** between clients and servers.
- Provides ***reliability***.
  - Acknowledgement
  - RTT (*round-trip-time*)
- TCP provides ***flow control*** to avoid overflow at the receiver side
  - ***Window:*** The amount of room currently available in the receiver buffer

2.5 TCP Connection Establishment and Termination

- Three-Way Handshake

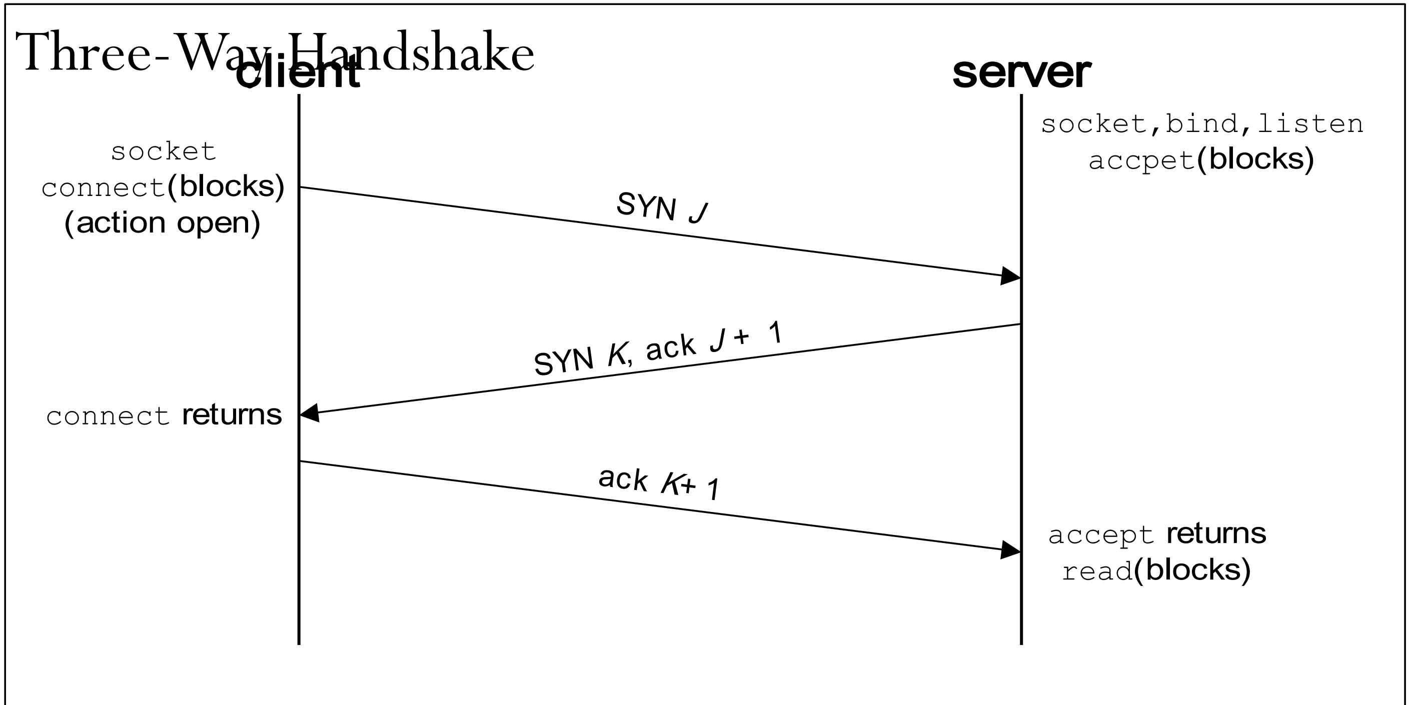
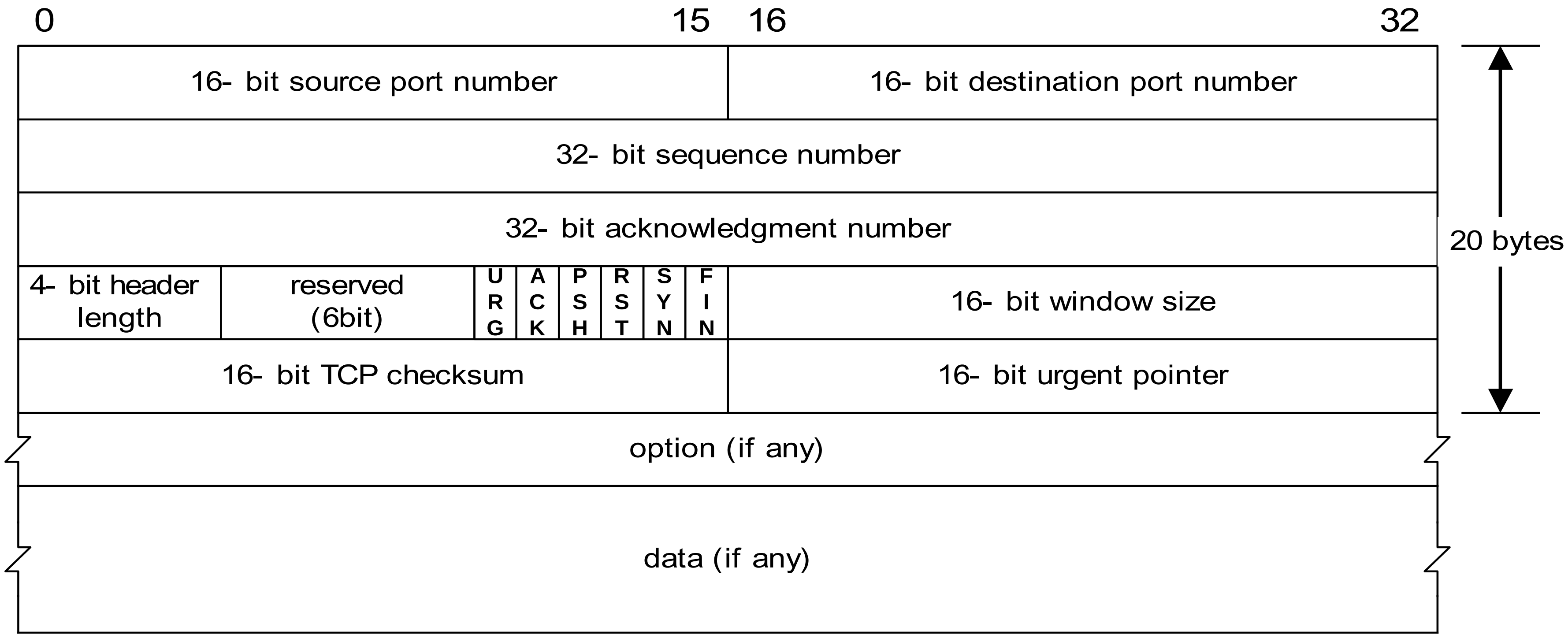


Figure 2.2 TCP three- way handshake

- SYN segment
- ACK

TCP Header



0 15 16 32

16- bit source port number

16- bit destination port number

32- bit sequence number

32- bit acknowledgment number

4- bit header length

reserved (6bit)

URG ACK PSH RST SYN FIN

16- bit window size

16- bit TCP checksum

16- bit urgent pointer

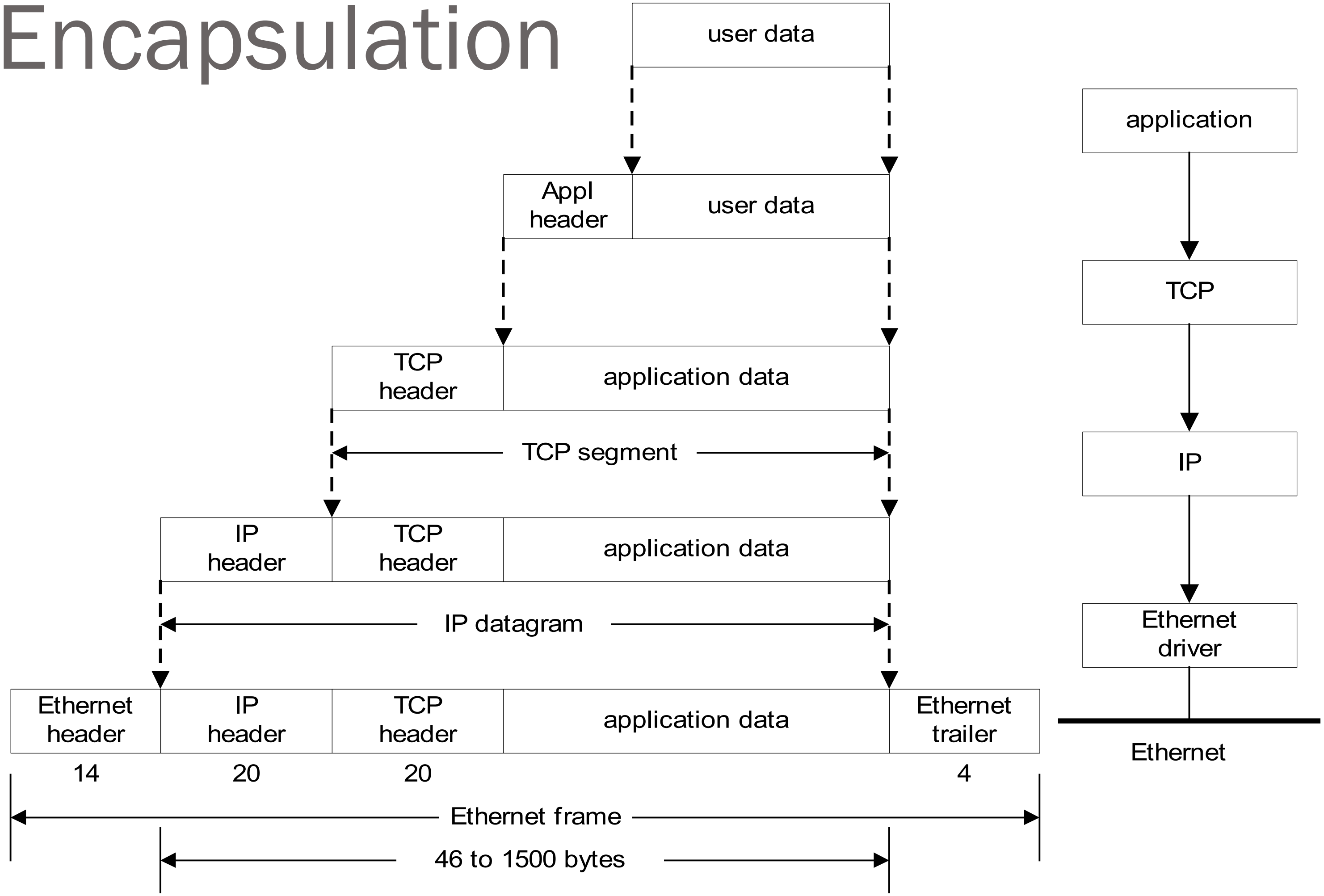
option (if any)

data (if any)

20 bytes

TCP Header

Encapsulation



user data

Appl header

user data

TCP header

application data

TCP segment

IP header

TCP header

application data

IP datagram

Ethernet header

IP header

TCP header

application data

Ethernet trailer

14

20

20

4

Ethernet frame

46 to 1500 bytes

application

TCP

IP

Ethernet driver

Ethernet

Encapsulation of data as it gose down the protocol stacks

2.5 TCP Connection Establishment and Termination (cont)

- TCP Options
  - MSS option
    - With this option the TCP sending the SYN announces its ***maximum segment size***, the maximum amount of data that it is willing to accept in each TCP segment, on this connection.
  - Window Scale option
  - Timestamp option

2.5 TCP Connection Establishment and Termination (cont)

- TCP Connection Termination

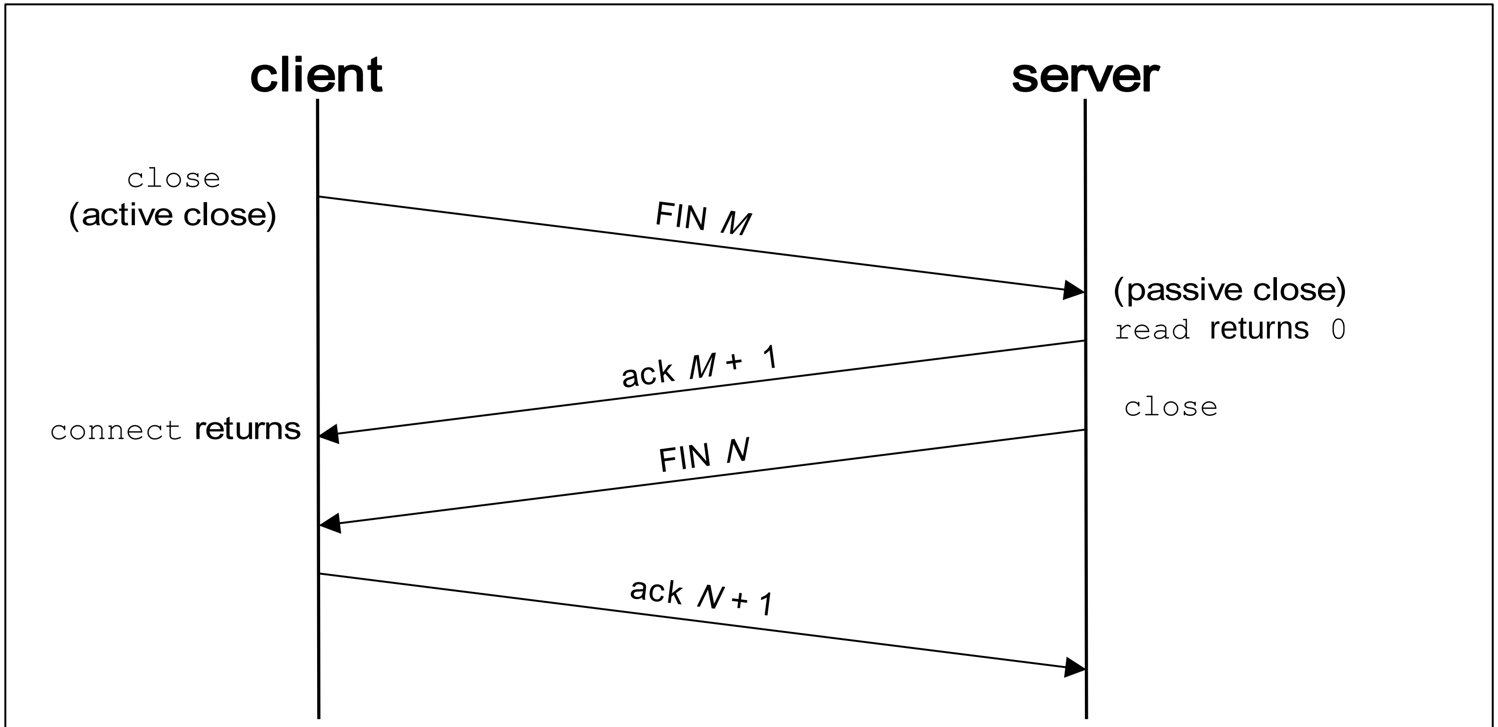


Figure 2.3 Packets exchanged when a TCP connection is closed.

- ***half-close*** : Between steps 2 and 3 it is possible for data to flow from the end doing the passive close to the end doing active close.

2.7 Port Numbers

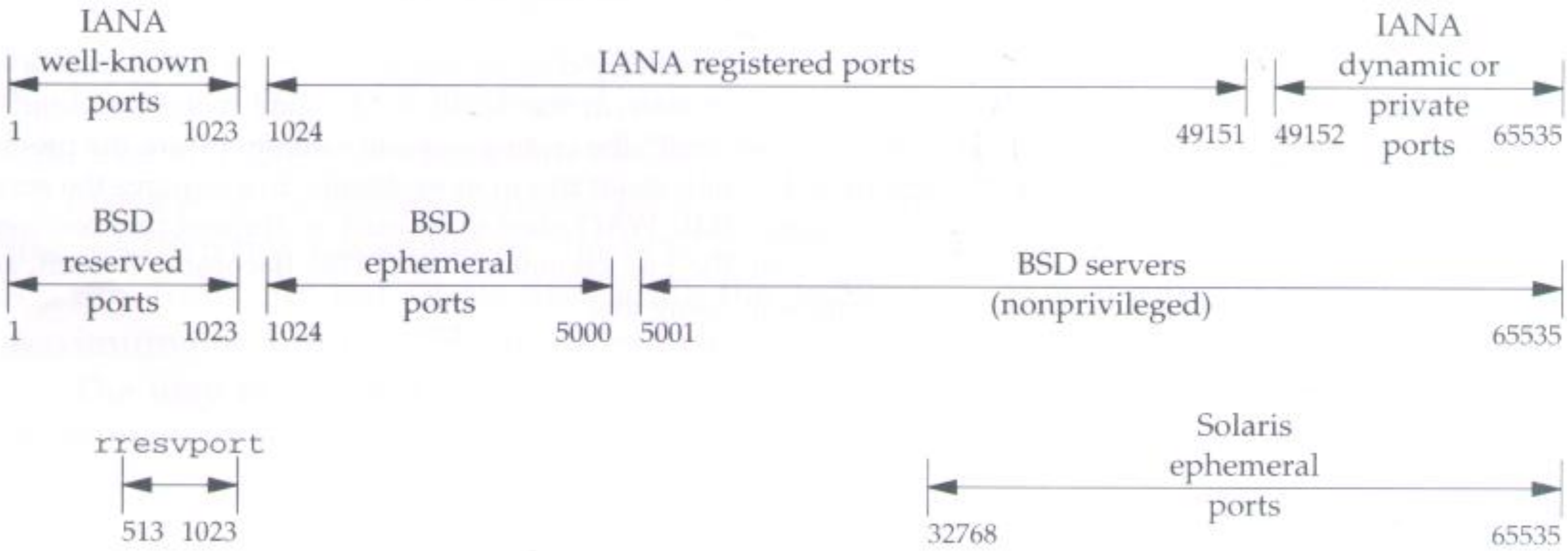


Figure 2.6 Allocation of port numbers.