Network Programming

Network programming is not distributed programming (somewhat lower-level).

They both rely on:
- computers as processing & storage resources
- a network and a common protocol stack

But network programming lacks:
- naming and location transparency
- an integrated programming & operating model
  (usually achieved thanks to a middleware)
The OSI model (1)

- **Application**
- **Presentation**
- **Session**
- **Transport**
- **Network**
- **Data Link**
- **Physical Link**

Logical peer-to-peer link

The OSI model (2)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical</td>
<td>Physical medium, electrical/optical signal processing.</td>
</tr>
<tr>
<td>Data Link</td>
<td>Grouping of bits into blocks, error detection/correction, local address format, medium access layer.</td>
</tr>
<tr>
<td>Network</td>
<td>Global address format, routing of data packets (no flow control).</td>
</tr>
<tr>
<td>Transport</td>
<td>End-to-end connection, flow control, retransmission, order.</td>
</tr>
<tr>
<td>Session</td>
<td>Failure detection &amp; reconnection in case of crashes.</td>
</tr>
<tr>
<td>Presentation</td>
<td>Standard data representation (e.g., marshaling convention).</td>
</tr>
<tr>
<td>Application</td>
<td>Basic application-level functionality (http, ftp, smtp, etc.).</td>
</tr>
</tbody>
</table>

Data encapsulation:

- Layer i + 1 header
- Layer i header
- Layer i - 1 header
- Actual data
  - Viewed by layer i as data
  - Viewed by layer i+1 as data
The Internet: some history

- In the middle of the cold war, early 1970s, the Department of Defense (DOD) decides to build a set of tools for interconnecting computer networks.
- The responsibility of this task falls on the Advance Research Project Agency (ARPA), which develops the ARPAnet protocol suite. A key design issue of ARPAnet was to resist to the massive destruction resulting from a nuclear attack.
  - Fully distributed architecture (no single point of failure)
- In the 1980s, the ARPAnet technology, also named TCP/IP (after its two main building blocks), spreads into the academic community (also very distributed), which had developed it.

TCP/IP

TCP = Transmission Control Protocol
IP = Internet Protocol

- It is cornerstone of the Internet
- It is de facto standard
- It is an often misunderstood technology
- It is an old yet alive protocol suite
The OSI model & TCP/IP

Internet Protocol (IP) ⇛ Network Layer (OSI n° 3)
- Packet oriented
- Routing with best-effort guarantee
- Error detection
- Datagram fragmentation

Transmission Control Protocol (TCP) ⇛ Transport Layer (OSI n° 4)
- Stream oriented
- Reliability guarantee
- FIFO order guarantee

Naming hosts with TCP/IP

- An IP address is used by the IP protocol (Network Layer) to name hosts (computers) and routers.

- An IP address consists of 32 bits (4 bytes) and is usually written in dotted decimal format, e.g., 130.223.171.8
Towards IPv6

Addresses encoded on 128 bits

\[ 2^{128} > 3.4 \times 10^{38} \text{ addresses are available} \]

Naming applications

Within a single host, applications are named (addressed) using ports. At the operating system level, this is known as port multiplexing.
TCP versus UDP (1)

TCP = *Transmission Control Protocol*

"...bla bla ... bla bla bla ..."

stream

UDP = *User Datagram Protocol*

TCP versus UDP (2)

TCP and UDP exhibit dual features:

<table>
<thead>
<tr>
<th></th>
<th>connection oriented</th>
<th>reliable channels</th>
<th>fifo ordering</th>
<th>message boundaries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TCP</strong></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td><strong>UDP</strong></td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

Network Programming © Benoit Garbinato
The Socket abstraction

- Sockets are programming abstractions that represent bidirectional communication endpoints between two or more processes.
- There exists two types of sockets: TCP sockets and UDP sockets.
- In Java, sockets are instances of various classes found in the `java.net` package.

TCP Sockets

- Because TCP is connection-oriented, we have two classes for TCP sockets in Java:

  ```java
  public class Socket {
    public Socket(String host, int port) {...}
    public OutputStream getOutputStream() {...}
    public InputStream getInputStream() {...}
    public void close() {...}
  }
  
  public class ServerSocket {
    public ServerSocket(int port) {...}
    public Socket accept() {...}
  }
  
  This captures the asymmetry when establishing a communication channel.
TCP sockets: server side

```java
public class DictionaryServer {
    private static Dictionary dico = new Hashtable();
    public static void main(String[] args) {
        ServerSocket connectionServer = null;
        Socket clientSession = null;
        PrintWriter out = null;
        BufferedReader in = null;
        dico.put("inheritance", "héritage");
        dico.put("distributed", "réparti");
        // Etc...
        try {
            connectionServer = new ServerSocket(4444);
            clientSession = connectionServer.accept();
            out = new PrintWriter(clientSession.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(clientSession.getInputStream()));
            String word, mot;
            while ((word = in.readLine()) != null) {
                mot = (String) dico.get(word);
                if (mot == null) mot = "sorry, no translation available for " + word + "!";
                out.println(mot);
            }
        }
        catch (IOException e) {
            System.out.println(e);
            System.exit(1);
        }
    }
}
```

TCP sockets: client side

```java
public class DictionaryClient {
    public static void main(String[] args) {
        Socket mySession = null;
        PrintWriter out = null;
        BufferedReader in = null;
        BufferedReader stdIn = null;
        try {
            if (args.length < 1) { System.out.println("Hostname missing."); System.exit(1); }
            mySession = new Socket(args[0], 4444);
            out = new PrintWriter(mySession.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(mySession.getInputStream()));
            stdIn = new BufferedReader(new InputStreamReader(System.in));
            String fromServer, fromUser;
            System.out.println("Go on, ask the dictionary server!");
            while (!fromUser.equals("quit")) {
                System.out.println("-> "+ fromServer);
            }
        }
        catch (UnknownHostException e) {
            System.err.println("Host Unknown: " + args[0]); System.exit(1);
        }
        catch (IOException e) {
            System.err.println("No connection to: " + args[0]); System.exit(1);
        }
    }
}
```
Streams in Java (1)

- Streams offer a unified programming abstraction for reading and writing data
- Streams can encapsulate various types of data sources, e.g., files, byte arrays in memory, sockets, etc.
- Streams can encapsulate other streams to stack up processing of the data
- In Java, streams are instances of various classes found in the java.io package

Streams in Java (2)

- Printer and writer classes are special streams manipulating only characters
- Standard operating systems-level input and output streams are also accessed via Java streams (System.in & System.out)
Objects through the wire (1)

**Fact:** the network knows nothing about objects, only about bytes

**Problem:** how can we send a complete object graph across the network?

**Solution:** almost any Java object can be automatically transformed into a sequence of bytes and recreated from that sequence

11010001101011001110100111010011100011101000111101101000111001

Objects through the wire (2)

- The process of transforming an object graph into a byte sequence is known as serialization or marshaling
- By implementing the java.io.Serializable interface, an object becomes serializable
- Two special stream classes allow for writing and reading objects:

```java
ObjectOutputStream out = new ObjectOutputStream(clientSession.getOutputStream());
out.writeObject(myCollection);

ObjectInputStream in = new ObjectInputStream(clientSession.getInputStream());
Collection yourCollection = (Collection) in.readObject();
```
Because UDP is connectionless, we have only one class for UDP sockets in Java:

```java
public class DatagramSocket {
    public DatagramSocket() {
        // Let the system choose a port
    }
    public DatagramSocket(int port) {
    }
    public void send(DatagramPacket packet) {
    }
    public void receive(DatagramPacket packet) {
    }
    public void close() {
    }
}
```

However, the DatagramPacket is also a key class when working with UDP sockets.

UDP Sockets

UDP sockets: server side

```java
public class QuoteServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = null;
        BufferedReader in = null;
        socket = new DatagramSocket(4445);
        in = new BufferedReader(new FileReader("one-liners.txt"));
        String quote = null;
        boolean moreQuotes = true;
        while (moreQuotes) {
            byte[] buf = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            quote = in.readLine();
            if (quote == null) {
                moreQuotes = false;
                buf = ("No more, bye!").getBytes();
            } else {
                buf = quote.getBytes();
            }
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            packet = new DatagramPacket(buf, buf.length, address, port);
            socket.send(packet);
        }
        socket.close();
    }
}
```
UDP sockets: client side

```java
public class QuoteClient {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) { System.out.println("Missing hostname"); System.exit(1); }
        InetAddress address = InetAddress.getByName(args[0]);
        DatagramSocket socket = new DatagramSocket();
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Go on, ask for a quote by typing return!");
        while (!stdIn.readLine().equals("quit")) {
            byte[] buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 4445);
            socket.send(packet);
            packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String received = new String(packet.getData());
            System.out.println("-> " + received);
        }
        socket.close();
    }
}
```
UDP Multicast: sender

```java
public class MulticastQuoteSender {
    public static void main(String[] args) throws Exception {
        MulticastSocket socket = null;
        BufferedReader in = null;
        socket = new MulticastSocket();
        socket.setTimeToLive(0);
        in = new BufferedReader(new FileReader("one-liners.txt"));
        String quote = null;
        boolean moreQuotes = true;

        while (moreQuotes) {
            Thread.currentThread().sleep(500);
            byte[] buf = new byte[256];
            quote = in.readLine();
            if (quote == null) {
                moreQuotes = false;
                buf = ("No more, bye!").getBytes();
            } else {
                buf = quote.getBytes();
            }
            InetAddress group = InetAddress.getByName("230.0.0.1");
            DatagramPacket packet = new DatagramPacket(buf, buf.length, group, 4446);
            socket.send(packet);
        }
        socket.close();
    }
}
```

UDP Multicast: receiver

```java
public class MulticastQuoteReceiver {
    public static void main(String[] args) throws Exception {
        try {
            MulticastSocket socket = new MulticastSocket(4446);
            InetAddress group = InetAddress.getByName("230.0.0.1");
            socket.joinGroup(group);
            while (true) {
                byte[] buf = new byte[256];
                DatagramPacket packet = new DatagramPacket(buf, buf.length);
                System.out.print("Waiting for the next quote: ");
                socket.receive(packet);
                String received = new String(packet.getData());
                System.out.println(received);
                if (received.indexOf("bye") != -1) break;
            }
            socket.leaveGroup(group);
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```