

Network Programming



Benoît Garbinato
distributed object programming lab

Monday, May 8, 2006

1

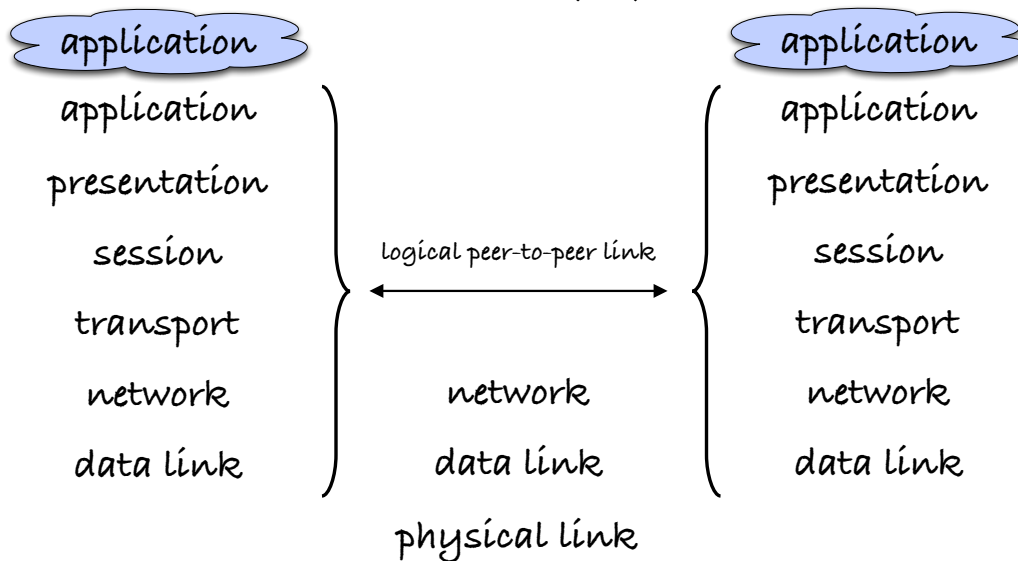
Network programming

- 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)

Monday, May 8, 2006

2

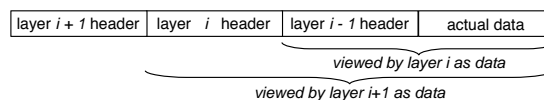
The OSI model (1)



The OSI model (2)

- Physical link physical medium, electrical/optical signal processing.
- Data link grouping of bits into blocks, error detection/correction, local address format, medium access layer.
- Network global address format, routing of data packets (no flow control).
- Transport end-to-end connection, flow control, retransmission, order.
- Session failure detection & reconnection in case of crashes.
- Presentation standard data representation (e.g., marshaling convention).
- Application basic application-level functionality (http, ftp, smtp, etc.).

Data encapsulation:



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

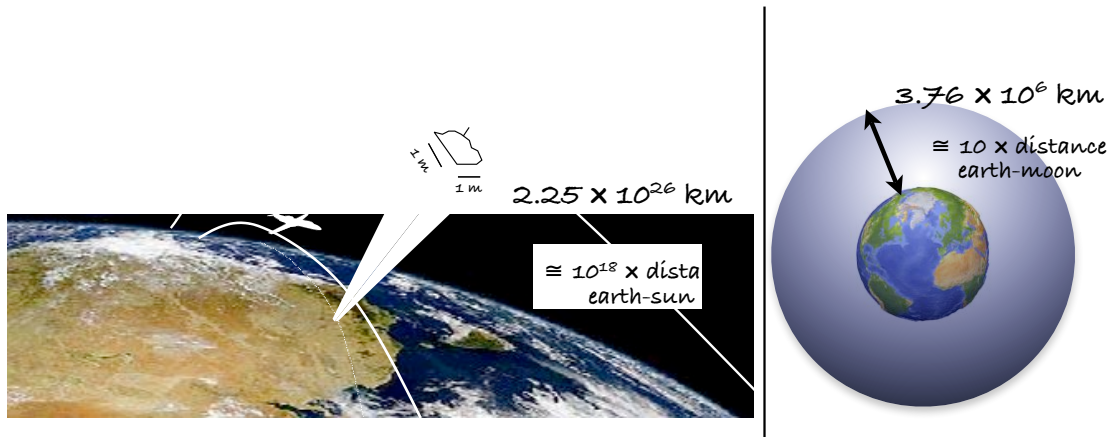
- An IP address is used by the IP protocol (Network Layer) to name hosts (computers) and routers.
- An IP address consists of 32-bits (4 bytes) and is usually written in dotted decimal format, e.g., 130.223.171.8

Class	First	Networks	Hosts	Address format
A	1-126	$2^7 - 2 = 126$	$2^{24} - 2 = 16'777'214$	net id host id
B	128-191	$2^{14} - 2 = 16'384$	$2^{16} - 2 = 65'534$	net id host id
C	192-223	$2^{21} - 2 = 2'097'152$	$2^8 - 2 = 254$	net id host id
D	224-239	-	-	multicast
E	240-247	-	-	reserved

Towards IPv6

Addresses encoded on 128 bits

$\Rightarrow 2^{128} > 3.4 \times 10^{38}$ addresses are available



Network Programming © Benoît Garbinato

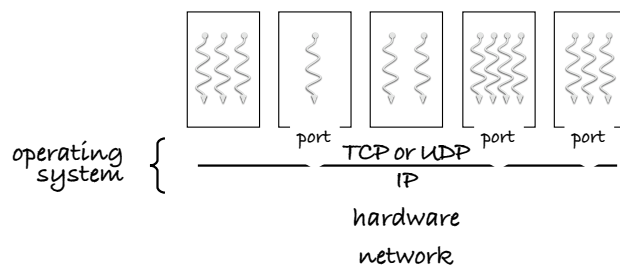


Monday, May 8, 2006

9

Naming applications

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



Network Programming © Benoît Garbinato



Monday, May 8, 2006

10

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:

	connection oriented	reliable channels	fifo ordering	message boundaries
TCP	YES	YES	YES	NO
UDP	NO	NO	NO	YES

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:

Client

```
public class Socket {  
    ...  
    public  
    Socket(String host, int port) {...}  
    public  
    OutputStream getOutputStream() {...}  
    public  
    InputStream getInputStream() {...}  
    public  
    void close() {...}  
    ...  
}
```

Server

```
public class ServerSocket {  
    ...  
    public  
    ServerSocket(int port) {...}  
    public  
    Socket accept() {...}  
    ...  
}
```

- This captures the asymmetry when establishing a communication channel

TCP sockets: server side

```
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);
            }
            out.close(); in.close(); connectionServer.close(); clientSession.close();
        } catch (IOException e) {
            System.out.println(e); System.exit(1);
        }
    }
}
```

TCP sockets: client side

```
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 = stdIn.readLine()).equals("quit") ) {
                out.println(fromUser);
                fromServer= in.readLine();
                System.out.println("-> " + fromServer);
            }
            out.close(); in.close(); stdIn.close(); mySession.close();
        } 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)

```
...
Socket clientSession= connectionServer.accept();
BufferedReader in= new BufferedReader(new InputStreamReader(clientSession.getInputStream()));
...
```

The diagram illustrates the relationship between the components in the code snippet. It shows a hierarchy of stream types:

- `data source` (under `clientSession.getInputStream()`)
- `byte stream` (under `InputStreamReader`)
- `character stream` (under `BufferedReader`)
- `buffered character stream` (under `new BufferedReader`)

- 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



1101000110101100111010011101001110001110100011110

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 :

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

```
ObjectInputStream in= new ObjectInputStream(clientSession.getInputStream());  
Collection yourCollection = (Collection) out.readObject();
```

UDP Sockets

- Because UDP is connectionless, we have only one class for UDP sockets in 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() {...}  
    ...  
}
```

```
public class DatagramPacket {  
    ...  
    public  
    DatagramPacket(...) {...}  
    public  
    byte[] getData() {...}  
    public  
    InetAddress getAddress() {...}  
    ...  
}
```

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

UDP sockets: server side

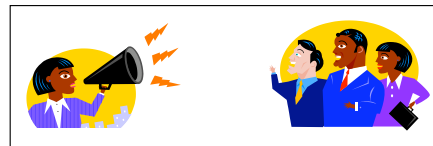
```
public class QuoteServer {  
    public static void main(String[] args) throws Exception {  
        DatagramSocket socket = null;  
        BufferedReader in = null;  
        socket = new DatagramSocket(4445);  
        in = new BufferedReader(new FileReader("one-liners.txt"));  
        String quote = null;  
        boolean moreQuotes = true;  
  
        while (moreQuotes) {  
            byte[] buf;  
            DatagramPacket packet = 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();  
    }  
}
```

```
Life is wonderful. Without it we'd all be dead.  
Daddy, why doesn't this magnet pick up this floppy disk?  
Give me ambiguity or give me something else.  
I.R.S.: We've got what it takes to take what you've got!  
We are born naked, wet and hungry. Then things get worse.  
Make it idiot proof and someone will make a better idiot.  
He who laughs last thinks slowest!  
Always remember you're unique. Just like everyone else.  
"Nice job, tripper!" "No thanks, boy. I'm stuffed!"  
A flashlight is a case for holding dead batteries!  
Lottery: A tax on people who are bad at math.  
Error: no keyboard - press F1 to continue.  
There's too much blood in my caffeine system.  
Artificial Intelligence usually beats real stupidity.  
Hard work has a future proof: business page six now.  
"Very funny, Scotty. Now beam down my clothes."  
Puritans! The heaving fear that someone, somewhere may be happy.  
...
```

UDP sockets: client side

```
public class QuoteClient {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) { System.out.println("Missing hostname"); System.exit(1); }
        DatagramSocket socket = new DatagramSocket();
        InetAddress address = InetAddress.getByName(args[0]);
        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



- ❑ A multicast allows for one-to-many communication in an anonymous way
- ❑ A multicast address is an address between 224.0.0.0 and 239.255.255.255, and defines a so-called multicast group
- ❑ In Java, multicast is available thanks to the `MulticastSocket` class:
 - ❑ Methods `joinGroup()` and `leaveGroup()` allow a receiver to respectively join and leave a multicast group
 - ❑ Method `setTimeToLive()` allows a sender to restrict the number of hops its sent messages are going through

UDP Multicast: sender

```
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.getByAddress("230.0.0.1");
            DatagramPacket packet = new DatagramPacket(buf, buf.length, group, 4446);
            socket.send(packet);
        }
        socket.close();
    }
}
```

UDP Multicast: receiver

```
public class MulticastQuoteReceiver {
    public static void main(String[] args) throws Exception {
        try {
            MulticastSocket socket = new MulticastSocket(4446);
            InetAddress group = InetAddress.getByAddress("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();
        }
    }
}
```

Questions?