# Networked systems – Sockets

Pr Sara Bouchenak

Sara.Bouchenak@insa-lyon.fr
http://liris.cnrs.fr/~sbouchen/

## Examples of network-based applications

- Email services
  - e.g. Gmail
- Document sharing services
  - e.g. Dropbox
- Social media
  - e.g. Facebook, Twitter
- Video-on-demand services



## OSI layers

| Layer | Function | Examples of protocols |
|---|---|---|
| 7. Application | Services the actual applications used by the end user | SMTP, HTTP |
| 6. Presentation | Formats data to be presented to the application layer | JPEG, GIF, ASCII |
| 5. Session | Session establishment between processes running on different machines | RPC |
| 4. Transport | Ensures that messages are delivered error-free, in sequence, and with no losses or duplication | TCP, UDP |
| 3. Network | Controls the operantions of the subnet, deciding which physical path the data takes | IP |
| 2. Data link | Error-free transfer of data frames from one node to another over the Physicla layer | PPP |
| 1. Physical | Transmission and reception of unstructured raw bit stream over the physical medium | |



## OSI layers (cont.)

| Layer | Function | Examples of protocols |
|---|---|---|
| **7. Application** | **Services the actual applications used by the end user** | **SMTP, HTTP** |
| **6. Presentation** | **Formats data to be presented to the application layer** | **JPEG, GIF, ASCII** |
| **5. Session** | **Session establishment between processes running on different machines** | **RPC** |
| **4. Transport** | **Ensures that messages are delivered error-free, in sequence, and with no losses or duplication** | **TCP, UDP** |
| 3. Network | Controls the operantions of the subnet, deciding which physical path the data takes | IP |
| 2. Data link | Error-free transfer of data frames from one node to another over the Physicla layer | PPP |
| 1. Physical | Transmission and reception of unstructured raw bit stream over the physical medium | |

# Objectives

- Introduce paradigms and mechanisms to build networked-based applications
  - Sockets
  - Remote Procedure Call (RPC / Java RMI)
  - World Wide Web
- Practical work
  - Application implementations

| Layer | Function | Examples of protocols |
|---|---|---|
| 7. Application | Services the actual applications used by the end user | SMTP, HTTP |
| 6. Presentation | Formats data to be presented to the application layer | JPEG, GIF, ASCII |
| 5. Session | Session establishment between processes running on different machines | RPC |
| 4. Transport | Ensures that messages are delivered error-free, in sequence, and with no losses or duplication | TCP, UDP |
| 3. Network | Controls the operations of the subnet, deciding which physical path the data takes | IP |
| 2. Data link | Error-free transfer of data frames from one node to another over the Physicla layer | PPP |
| 1. Physical | Transmission and reception of unstructured raw bit stream over the physical medium | |



# Professors

- Sara Bouchenak
  - Professor INSA Lyion – LIRIS Laboratory
  - CM, TP
- Frédérique Biennier
  - Professor INSA Lyion – LIRIS Laboratory
  - CM



# Agenda

| Lectures | Lab |
|---|---|
| Introduction to Sockets<br>Introduction to RPC/RMI | |
| Introduction to WWW<br>Building multi-tier web applications | Building Socket-based applications (Part I) |
| | Building Socket-based applications (Part II) |
| Network infrastructures | Building HTTP servers (Part I) |
| Network infrastructures | Building HTTP servers (Part II) |



# Outline

1. Introduction to sockets
2. **Addressing**
3. Point-to-point communication with TCP sockets
4. Point-to-point communication with UDP sockets
5. Group communication with multicast
6. Locating network resources

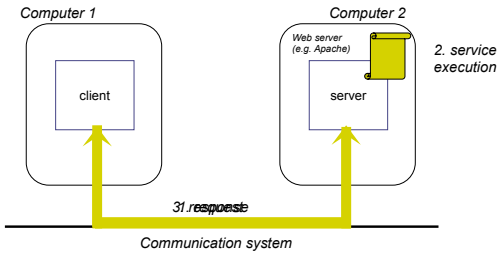
## Client/server applications

- The server provides some service
  - Examples
    - processing database queries
    - sending out current stock prices
- The client uses the service provided by the server
  - Examples
    - displaying database query results to the user
    - making stock purchase recommendations to an investor



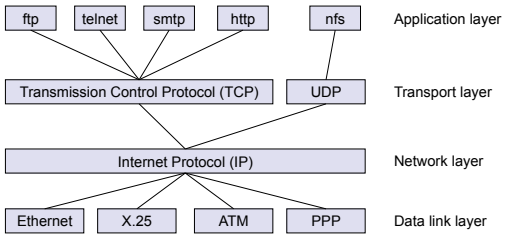
## Client/server applications (2)

- Synchronous communication






## Protocol layers






## TCP, UDP, IP

- IP (Internet Protocol)
  - Corresponds to the network layer of the OSI model
  - Addressing, routing and transport of data packets
- TCP (Transmission Control Protocol)
  - A reliable protocol
  - Guarantees the delivery of data packets (i.e. data can not be dropped)
  - Puts data packets together in the right order (i.e. data must arrive on the client side in the same order in which the server sent it)
- UDP (User Datagram Protocol)
  - Is not reliable
  - More efficient
  - Used with applications which put high demands on transmission power, but can accept reductions in transmission quality (e.g. transmission of image/ sound in real time)

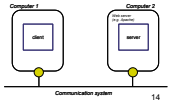
## Ports

- A computer connected to a TCP/IP network is identified by its IP address (unique identifier of a computer)
- When a process needs to be addressed via the network, it is identified by
  - the underlying computer's IP address + a port number associated with the process (unique identifier of a process)
- Port numbers are managed by the operating system
- Port numbers are allocated to processes when they want to use the network
- Many important services have a standardized port
  - Example: port 25 for telnet service



## Ports (2)

- Server side
  - When a server process is started on a computer, the OS links it to a fixed port
  - The server waits as a background process (daemon) for incoming connections addressed to this port
- Client side
  - When a connection to a service is to be opened between a client process on computer and a server process on another computer, the client porcess is assigned a port number
  - The server process can then uniquely identify the client process in connection in the entire network
- Ports as communication points
  - Ports represent communication points






## Sockets

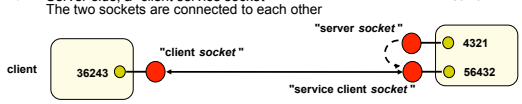
- A socket may be seen as a communication channel between two points (i.e. two ports)
- Stricly speaking, a socket is an endpoint of a communication connection between two computers
- A socket is identified by an IP address and a port number
- From the programmer's point of view, a socket represents the mechanism to transfer data from one computer to another



## How sockets work

- Example with TCP/IP, in 3 steps:

1. The server creates a "server socket" (associated with a port), and waits for incoming connections
2. The client connects to the "server socket"; two sockets are created
   - Client-side, a "client socket"
   - Server-side, a "client service socket"
     The two sockets are connected to each other




3. The client and server communicate via the two sockets (writing/ reading bytes); the "server socket" can accept new connections

## Sockets – A bit of history

- Originally
  - Sockets were developed for BSD Unix, in the 1980s
  - Sockets used to be part of the operating system; they had to be invoked via system-specific libraries for C/C++
  - Programming distributed applications was hard (access was different from one OS to another, programs were not portable)
- Today
  - Sockets are available on all platforms and represent the most fundamental communication mechanism
  - Example: the Java programming interface for sockets abstracts them from the underlying OS, making them easier to use



## Outline

1. Introduction to sockets
2. **Addressing**
3. Point-to-point communication with TCP sockets
4. Point-to-point communication with UDP sockets
5. Group communication with multicast
6. Locating network resources



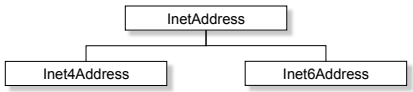
## Addressing in Java

- The *java.net* package provides the following addressing-related classes:
  - InetAddress
  - Inet4Address
  - Inet6Address
  - SocketAddress
  - InetSocketAddress



## IP addressing in Java

- For IP addressing, three classes are provided:
  - ***InetAddress*** represents an IP address, which is either a 32- or 128-bit unsigned number used by IP
  - ***Inet4Address*** represents a 32-bit IPv4 address
    It has the familiar form *n.n.n.n*, where *n* is a byte; e.g., 129.250.35.250
  - ***Inet6Addresss*** represents a 128-bit IPv6 address



## Java classes related to IP addressing

- java.net.InetAddress class
  - Represents an IP address, which is either a 32- or 128-bit unsigned number used by IP
  - Textual representation of an IP address: it is family specific (see Inet4Address for example)
  - Host name associated with an IP address, e.g. *hoff.e.ejf-grenoble.fr*
  - Host name to IP address resolution: using a network naming service such as DNS (Domain Name System), NIS (Network Information Service)
  - Some API elements
    - String **getHostAddress**() Returns the IP address string in textual presentation.
    - String **getHostName**() Gets the host name for this IP address.
    - static InetAddress **getLocalHost**() Returns the local host.



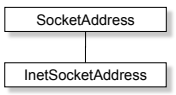
## Java classes related to IP addressing (2)

- java.net.Inet4Address class
  - Textual representation of IPv4 address has the familiar form *n.n.n.n*, where *n* is a byte; e.g., 129.250.35.250
  - Some API elements
    - byte[] **getAddress**() Returns the raw IP address of this InetAddress object.
    - String **getHostAddress**() Returns the IP address string in textual presentation form.



## Socket addressing in Java

- Two classes are provided for socket addressing:
  - SocketAddress
    - It is an abstract socket address, independent of a specific protocol
    - It is intended for subclassing for a specific protocol (e.g. InetSocketAddress)
  - InetSocketAddress
    - It represents an IP socket address and can include:
      - an IP address (e.g., 129.250.35.250) and port (e.g., 80)
      - a hostname (e.g., coastnews.com) and port (e.g., 1000)
      - port only (e.g., 1010), in which case, a wildcard IP address is assumed






## Outline

1. Introduction to sockets
2. Addressing
3. **Point-to-point communication with TCP sockets**
4. Point-to-point communication with UDP sockets
5. Group communication with multicast
6. Locating network resources

## Java sockets over TCP

- Network communication in the connected mode (using TCP)
  - General schema:
    - a connection is opened between a client and a server,
    - a series of requests (i.e. messages) are exchanged between the client and the server,
    - the connection is closed
  - The server maintains a client session in which the state between different requests is maintained
  - TCP guaranties: reliability, delivery and order of messages
  - Adequate to long communication where several messages are exchanged (e.g. HTTP client sessions)



## Java classes related to TCP sockets

- The *java.net* package provides the following classes related to TCP communication:
  - ServerSocket
  - Socket
- *ServerSocket*
  - It represents the socket on a server that waits and listens for requests for service from a client
- *Socket*
  - It represents the endpoints for communication between a server and a client.



## Example of Java sockets/TCP

**Server side**

```
import java.net.*;

int port = 1234;
// Create a server socket associated with
// the server port
ServerSocket serverSocket = new
    ServerSocket(port);
// End-less loop
while (true) {
    System.out.println("Waiting for
    client…");
    // Server waits for a connection
    Socket serverToClient =
    serverSocket.accept();

    // A client connected
    System.out.println("Client " +
    client.getInetAddress() + "
    connected.");
    // Server receives a message from
    // client
    ...
}
```

**Client side**

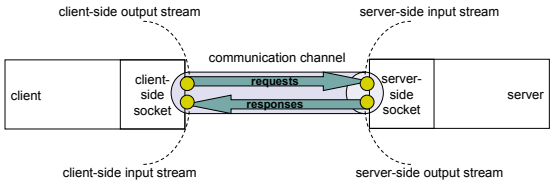
```
import java.net.*;

String serverHost = "sun";
int serverPort = 1234;

// Client connects to server
Socket clientToServer = new
    Socket(serverHost, serverPort);

// Client connected
System.out.println("Connected to " +
    server.getInetAddress());

// Client sends a message to server
...
```



## Streams

- Definition
  - Streams are an abstraction for arbitrary data streams
- Examples
  - Streams from/to a socket
  - Streams from/to a file
  - Streams from/to the console
- Input and output streams
  - Input streams used to receive (i.e. read) bytes
  - Output streams used to send (i.e. write) bytes

# Client/server streams






# Streams in Java

- The *java.io* package contains the following classes related to streams:
  - *InputStream* / *OutputStream*
    - abstract classes that respectively represent input streams of bytes and output streams of bytes
  - *ObjectInputStream* / *ObjectOutputStream*
    - classes respectively representing input streams of Java objects and output streams of Java objects
  - *FileInputStream* / *FileOutputStream*
    - classes representing input streams for respectively reading data from a file ou writing data to a file
  - *FilterInputStream* / *FilterOutputStream*
    - classes that respectively contain other input streams or output streams, which it uses to possibly transfrom data along the way



# Java sockets and streams

**Server side**

```
import java.io.*;

...
// End-less loop
while (true) {
    System.out.println("Waiting for
    client…");
    // Server waits for a connection
    Socket serverToClient =
    serverSocket.accept();
    // A client connected
    System.out.println("Client " +
    serverToClient.getInetAddress() + "
    connected.");

    // Get the server's output stream
    OutputStream os =
            serverToClient.getOutputStream();
    // Build data and transform it
    // to bytes
    Date date = new Date();
    byte[] b = date.toString().getBytes();
    // Write in output stream
    os.write(b);
}
```

**Client side**

```
import java.io.*;

...
// Client connects to server
Socket clientToServer = new
    Socket(serverHost, serverPort);
// Client connected
System.out.println("Connected to " +
    clientToServer.getInetAddress());

// Get the client's input stream
InputStream is =
    clientToServer.getInputStream();
// Read data from input stream
byte[] b = new byte[100];
int num = is.read(b);
// Transform data from bytes to String
String date = new String(b);
System.out.println("Server said: " +
    date);
```



# Other Java stream types

**Server side**

```
import java.io.*;
...
// End-less loop
while (true) {
    System.out.println("Waiting for
    client…");
    // Server waits for a connection
    Socket serevrToClient =
    serverSocket.accept();
    // A client connected
    System.out.println("Client " +
    serevrToClient.getInetAddress() + "
    connected.");
    // Get the server's output stream
    OutputStream os =
            serevrToClient.getOutputStream();
    // Get the associated object output //
    stream
    ObjectOutputStream oos = new
            ObjectOutputStream(os);
    // Write object in output stream
    Date date = new Date();
    oos.writeObject(date);
    // Close output stream
    oos.close();
}
```

**Client side**

```
import java.io.*;
...
// Client connects to server
Socket clientToServer = new
    Socket(serverHost, serverPort);
// Client connected
System.out.println("Connected to " +
    clientToServer.getInetAddress());

// Get the client's input stream
InputStream is =
    clientToServer.getInputStream();
// Get the associated object input stream
ObjectInputStream ois = new
    ObjectInputStream(is);

// Read object from input stream
Date date = (Date) ois.readObject();

System.out.println("Server said: " +
    date);
// Close input stream
ois.close();
```

# Outline

1. Introduction to sockets
2. Addressing
3. Point-to-point communication with TCP sockets
4. **Point-to-point communication with UDP sockets**
5. Group communication with multicast
6. Locating network resources



# Java sockets over UDP

- Networking in the unconnected mode (using UDP)
  - Some applications that communicate over the network do not require reliable, point-to-point channel provided by TCP
  - Applications might benefit from a mode of communication that delivers independent packages of information whose arrival and order of arrival are not guaranteed
  - UDP protocol provides a mode of network communication whereby applications send packets of data, called datagrams, to one another.
  - A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.



# Java classes related to UDP

- The *java.net* package provides the following classes related to UDP communication :
  - DatagramPacket
  - DatagramSocket
- *DatagramPacket*
  - It represents a datagram packet used for connectionless delivery and normally includes destination address and port information
- *DatagramSocket*
  - It is a socket used for sending and receiving datagram packets over a network via UDP
  - A *DatagramPacket* is sent from a *DatagramSocket* by calling the *send* method
  - The *receive* method is use for receiving a *DatagramPacket* on a *DatagramSocket*



# Example of Java sockets/UDP

**Server side**

```
import java.net.*;

int port = 1234;
// Create a datagram socket associated
// with the server port
DatagramSocket serverSock = new
    DatagramSocket(port);
// End-less loop
while (true) {
    System.out.println("Waiting for client
    packet…");
    byte[] buf = new byte[256];
    // Create a datagram packet
    DatagramPacket packet = new
            DatagramPacket(buf, buf.length);
    // Wait for a packet
     serverSock.receive(packet);

    // Get client IP address and port number
    InetAddress clientAddr = packet.getAddress();
    int clientPort = packet.getPort();
    // Build a response
    initialize buf ...
    // Build a datagram packet for response
    packet = new DatagramPacket(buf, buf.length,
            clientAddr, clientPort);
    // Send a response
    serverSock.send(packet);
}
```

**Client side**

```
import java.net.*;

int serverPort = 1234;
String serverHost = ...;

// Create a datagram socket
DatagramSocket clientSock = new DatagramSocket();

byte[] buf = new byte[256];
// Get server's IP address
InetAddress serverAddr =
    InetAddress.getByName(serverHost);
// Build a request
initialize buf ...
// Create a datagram packet destined for the
// server
DatagramPacket packet = new DatagramPacket(buf,
    buf.length, serverAddr, serverPort);
// Send datagram packet to server
clientSock.send(packet);

// Build a datagram packet for response
packet = new DatagramPacket(buf, buf.length);
// Receive response
clientSock.receive(packet);
String received = new String(packet.getData(), 0,
    packet.getLength());
System.out.println("Response: " + received);
```

## Outline

1. Introduction to sockets
2. Addressing
3. Point-to-point communication with TCP sockets
4. Point-to-point communication with UDP sockets
5. **Group communication with multicast**
6. Locating network resources



## Multicast

- Besides point-to-point communication, Java provides multicast communication which can be used to communicate with a group of communication points
- Multicat is already available at the Internet protocol level
- Multicast is based on the UDP transport protocol
- Multicast communication uses IP addresses of a particular class (class D) which are not linked to host computers, but are exclusively reserved for multicast communication



## Multicast – How it works

- First, all participants in a communication group must register with the group to ba able to join the communication
- Messages can be sent to all group members or received from others
- To end the membership in a group, the participant must sign out
- Applications: multimedia conferences on the Internet are examples of applications for the multicast



## Java multicast

- The *java.net* package provides the following classes related to multicast communication :
  - DatagramPacket
  - MulticastSocket
- *MulticastSocket*
  - A *MulticastSocket* is a (UDP) *DatagramSocket*, with additional capabilities for joining "groups" of other multicast hosts on the internet
  - One would join a multicast group by first creating a *MulticastSocket* with the desired port, then invoking the *joinGroup(InetAddress groupAddr)* method
  - When one sends a message to a multicast group, all subscribing recipients to that host and port receive the message
  - The socket does not need be a member of the multicast group to send messages to it
  - When a socket subscribes to a multicast group/port, it receives datagrams sent by other hosts to the group/port, as do all other members of the group and port.
  - A socket relinquishes membership in a group by the *leaveGroup(InetAddress addr)* method

## Java multicast example

```
import java.net.*;
...
// Group IP address
InetAddress groupAddr =
    InetAddress.getByName("228.5.6.7");
Int groupPort = 6789;

// Create a multicast socket
MulticastSocket s = new
    MulticastSocket(groupPort);
// Join the group
s.joinGroup(groupAddr);

// Build a datagram packet for a message
// to send to the group
String msg = "Hello";
DatagramPacket hi = new
    DatagramPacket(msg.getBytes(),
    msg.length(), groupAddr, groupPort);

// Send a multicast message to the group
s.send(hi);
```

```
// Build a datagram packet for response
byte[] buf = new byte[1000];
DatagramPacket recv = new
    DatagramPacket(buf, buf.length);

// Receive a datagram packet response
s.receive(recv);
...
// OK, I'm done talking - leave the group
s.leaveGroup(groupAddr);
```



## Outline

1. Introduction to sockets
2. Addressing
3. Point-to-point communication with TCP sockets
4. Point-to-point communication with UDP sockets
5. Group communication with multicast
6. **Locating network resources**



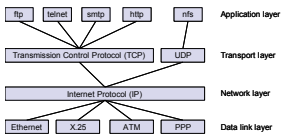
## Identifying and locating network resources

- Examples of network resources
  - An image, audio file available on the network
  - A program file, such as a Servlet, available on the network
- Classes related to locating or identifying network resources (c.f. package *java.net*)
  - URI
  - URL
  - URLClassLoader
  - URLConnection
  - URLStreamHandler
  - HttpURLConnection
  - JarURLConnection



## Identifying and locating network resources (2)

- ***URI***
  - It represents a Uniform Resource Identifier for a resource.
  - It is an identifier for a resource but not necessarily a locator for that resource.
- ***URL***
  - It represents a Uniform Resource Locator for a resource.
  - A *URL* tells how to access the resource, while a *URI* may or may not.
  - The protocol used to locate the resource is known from the *URL*.



## URL – Uniform Resource Location

- A URL is a pointer to a "resource" on the World Wide Web
- A resource can be something as simple as a file or a directory, or it can be a reference to a more complicated object, such as a query to a database or to a search engine
- Example of a URL:
  - http : //liris.cnrs.fr
- A URL has two main components:
  - Protocol identifier
  - Resource name



## URL

- URL's protocol
  - The protocol identifier indicates the name of the protocol to be used to fetch the resource.
  - The example uses the Hypertext Transfer Protocol (HTTP), which is typically used to serve up hypertext documents.
  - HTTP is just one of many different protocols used to access different types of resources on the net.
  - Other protocols include File Transfer Protocol (FTP), File, and News.
- URL's resource
  - The format of the resource name depends entirely on the protocol used.
  - For many protocols, including HTTP, the resource name contains one or more of the components listed below:
    - **Host Name** The name of the machine on which the resource lives.
    - **Port Number** The port number to which to connect (typically optional).
    - **Filename** The pathname to the file on the machine.
    - **Reference** A reference to a named anchor within a resource that usually identifies a specific location within a file (typically optional).



## Identifying and locating network resources in Java

- ***URLConnection***
  - It is the abstract superclass of all classes that represent a connection between an application and a network resource identified by a URL.
  - Given a URL and hence a protocol, *URL.openConnection*() returns an instance of the appropriate implementation of *URLConnection* for the protocol.
  - The instance provides the means (*URLConnection.connect*()) to actually open the connection and access the URL.
- ***HttpURLConnection***
  - It is the most commonly used implementation of *URLConnection*.
  - It is for http protocol, the protocol used for accessing content on web servers.



## Outline

1. Introduction to sockets
2. Addressing
3. Point-to-point communication with TCP sockets
4. Point-to-point communication with UDP sockets
5. Group communication with multicast
6. Locating network resources

# Agenda

| Lectures | Lab |
|---|---|
| Introduction to Sockets<br>Introduction to RPC/RMI | |
| Introduction to WWW<br>Building multi-tier web applications | Building Socket-based applications (Part I) |
| | Building Socket-based applications (Part II) |
| Network infrastructures | Building HTTP servers (Part I) |
| Network infrastructures | Building HTTP servers (Part II) |



# References

This lecture is built from:


- Sun Microsystems. *Java Tutorial on Networking*. http://java.sun.com/docs/books/tutorial/networking/
- http://www.reddit.com/r/networking/
- M. Boger. *Java in Distributed Systems: Concurrency, Distribution and Persistence*. Wiley, 2001.
- This lecture is partly based on lectures given by Sacha Krakowiak, http://sardes.inrialpes.fr/people/krakowia/