LIÈGE université

Faculté des Sciences Appliquées

INFO0010 - Introduction to Computer Networking

THE MONSTER HUNTING GAME

Guidelines & Complement

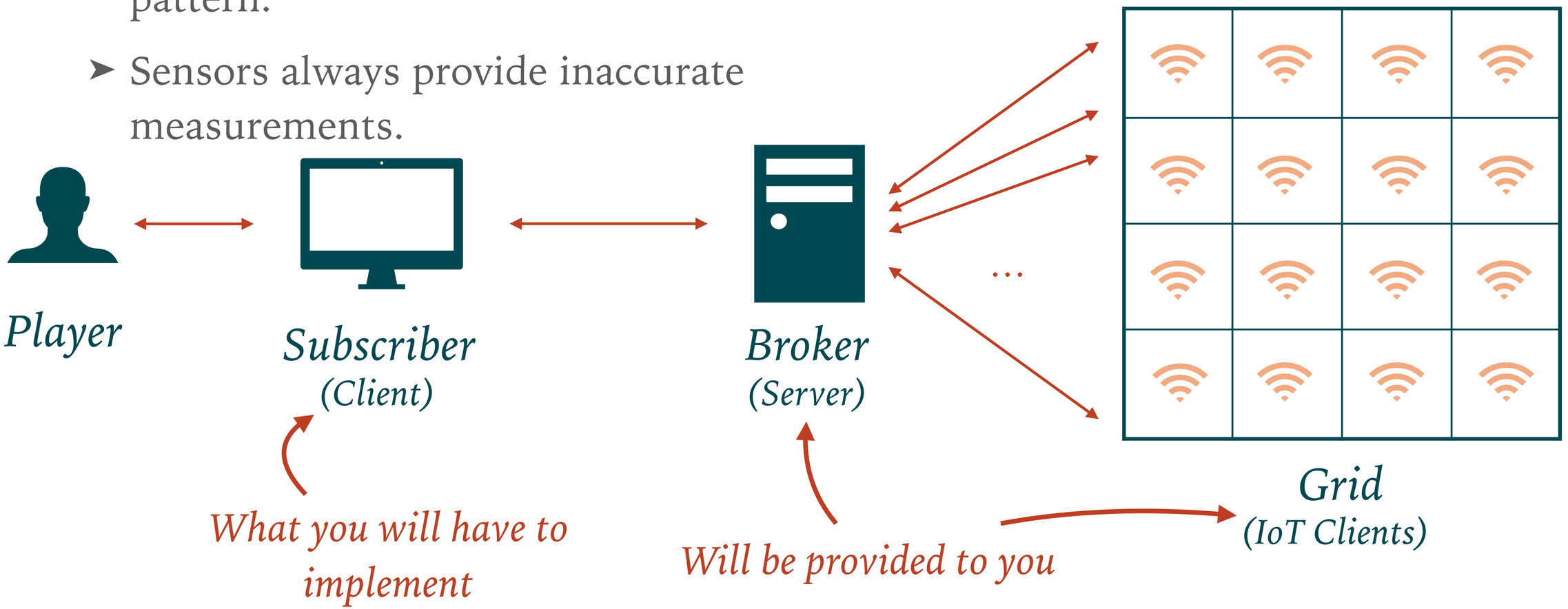
Emeline Marechal *Guy Leduc*

Year 2021-2022

THE ASSIGNMENT

PROJECT IN A NUTSHELL (I)

- *You will develop a Client/Server Application to **hunt a monster** with IoT devices.*
  - The Broker implements the publish/subscribe pattern.
  - Sensors always provide inaccurate measurements.



PROJECT IN A NUTSHELL (II)

- *System Operations:*
  - The game is played in turn between you and the monster.
  - One iteration:
    1. The monster moves on the Grid (up, down, left, right, or diagonally),
    2. The IoT sensors measure their environment and publish the information to the Broker,
    3. This information is sent to the Subscriber (if it has correctly subscribed),
    4. The Subscriber displays the information to the player,
    5. The Subscriber publishes the player's guess to the Broker to know if they caught the monster or not. If yes, the game is over. If not, get back to 1.

PROJECT IN A NUTSHELL (III)

- *(Un)intentional malevolence:*
  - What happens if the Broker sends a position outside of the Grid?
  - What happens if the Broker sends malformed packet?
    - **Good behavior:** ignore message and send back error.
    - **Bad behavior:** trigger an exception that will crash the program.

Never expect! Always check!

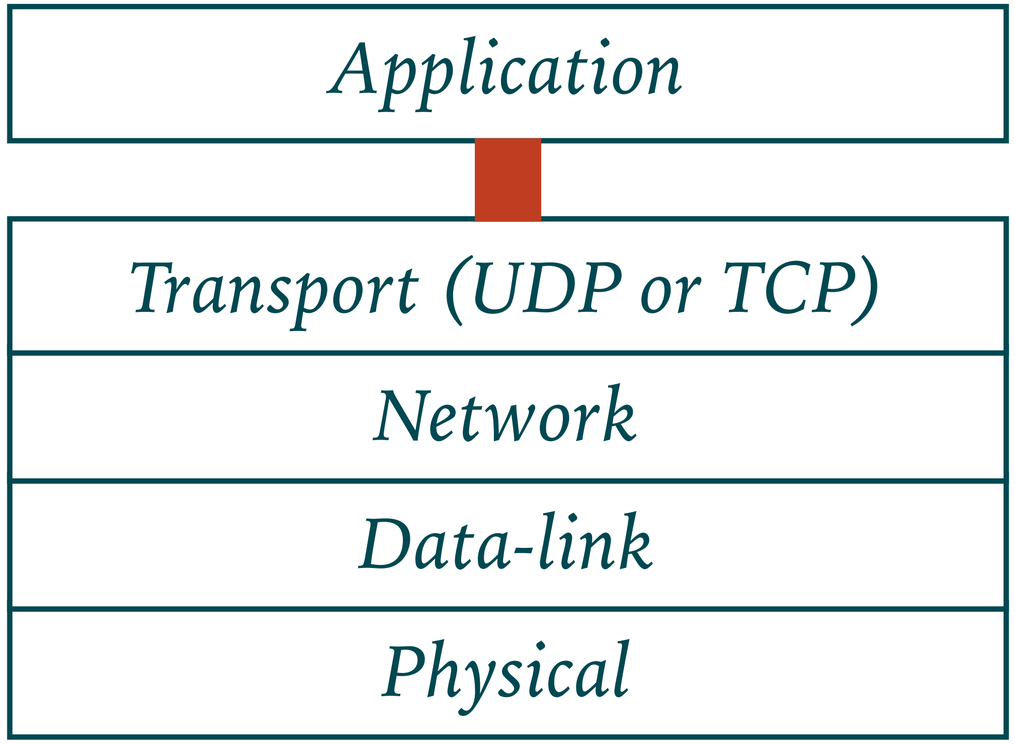
PROJECT IN A NUTSHELL (IV)

- *Guidelines:*
  - Java (1.8) Sockets,
  - Console input and output (no fancy GUI),
  - Imposed binary protocol to follow,
  - To be realized alone,
  - Must be **fully operational** on the `ms8xx.montefiore.ulg.ac.be` machines. See http://www.student.montefiore.ulg.ac.be/accounts.php to create an account if not already done,
  - **Hard deadline: 2nd of November 2021.**

SOCKET PROGRAMMING

WHAT IS A SOCKET? (I)

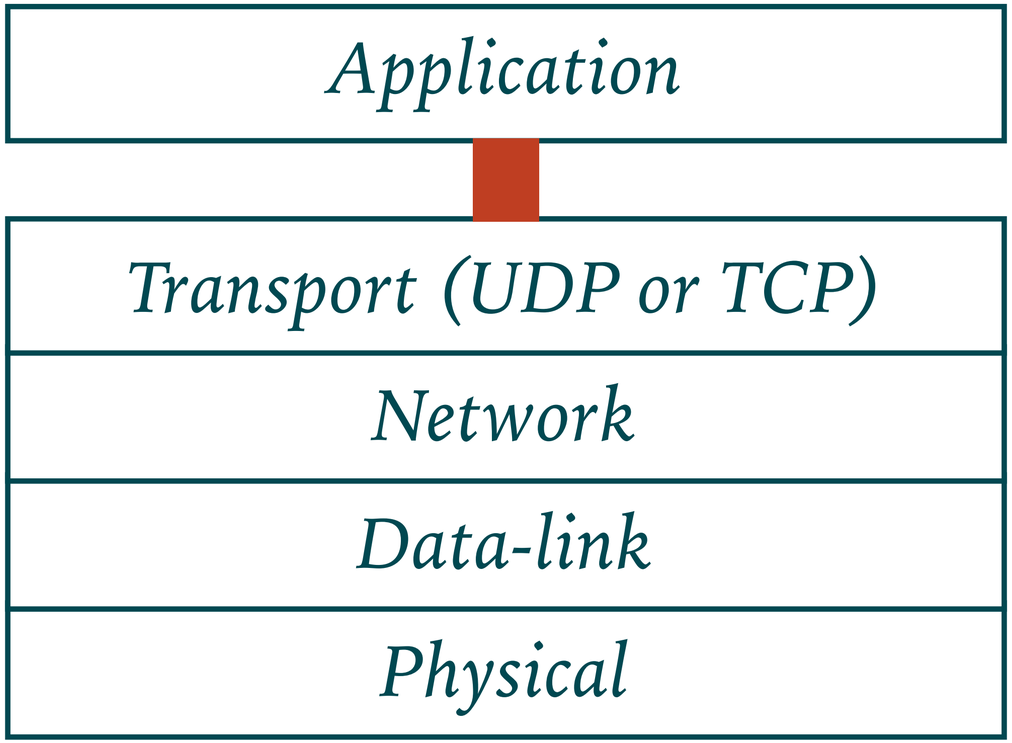
- *Interface to the network protocol stack (typically, the transport layer)*
  - *Allows communication between two **processes** (same or remote machines)*



WHAT IS A SOCKET? (I)

- *Interface to the network protocol stack (typically, the transport layer)*
  - *Allows communication between two **processes** (same or remote machines)*

Like a door through which the Application pushes bytes to the Transport Layer



COMMUNICATION MODELS (I)

- *TCP vs. UDP*

	UDP	*TCP*
Connection		✗
Reliable (no loss, no reordering)		✗
Stream-oriented		✗
Congestion control		✗

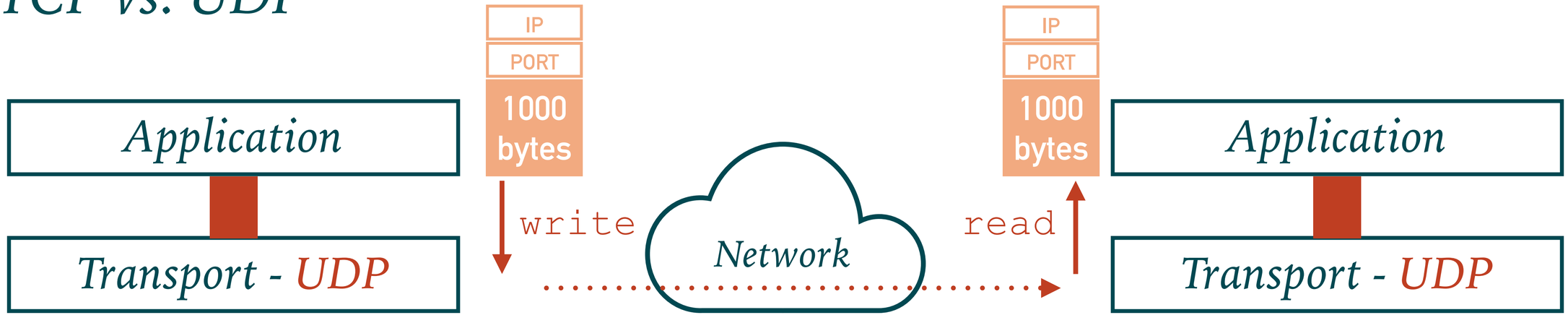
COMMUNICATION MODELS (II)

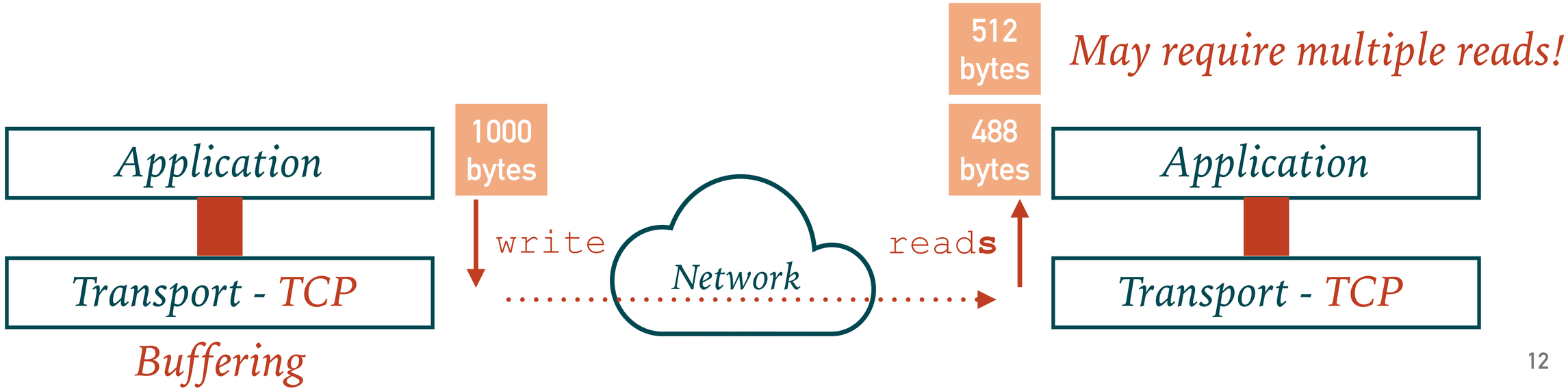
- *Stream-oriented*
  - The Application sends messages through the TCP socket, but TCP just sees a stream of bytes.
  - On the two sides of the TCP connection, the application processes may well write/read byte chunks of different sizes.
  - Streams are identical nevertheless.

Need to recover the Application Messages from the stream!

COMMUNICATION MODELS (IV)

- *TCP vs. UDP*





COMMUNICATION MODELS (III)

- *Congestion Control*
  - What has been sent through the socket may not have left the machine after calling `write()`.
  - Can use `flush()` to force the writing to the socket.
  - Can use `socket.setTcpNoDelay(true)` to disable Nagle's algorithm.

(
- Nagle's algorithm aims at improving TCP efficiency by reducing the number of small packets sent over the network.
- Not appropriate for highly interactive applications with small data transfer.
)

SIMPLE CLIENT EXAMPLE

```java
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class Client {
    public static void main (String[] argv) throws Exception {
        Socket s = new Socket( host: "address of server",  port: 8086); // Connect to server
        OutputStream out = s.getOutputStream();
        InputStream in = s.getInputStream();
        byte[] msg = new byte[64];
        while (true) {
            int len = in.read(msg);
            if (len <= 0) break; // Connection may have been closed by the other side
            out.write((len + " bytes received").getBytes());
            out.flush();
        }
        s.close();
    }
}
```

SIMPLE CLIENT EXAMPLE

```java
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class Client {
    public static void main (String[] argv) throws Exception {
        Socket s = new Socket( host: "address of server",  port: 8086); // Connect to server
        OutputStream out = s.getOutputStream();
        InputStream in = s.getInputStream();
        byte[] msg = new byte[64];
        while (true) {
            int len = in.read(msg);
            if (len <= 0) break; // Connection may have been closed by the other side
            out.write((len + " bytes received").getBytes());
            out.flush();
        }
        s.close();
    }
}
```

Very bad practice to throw Exceptions in the main. You need to properly catch Exceptions and deal with it.

SOME COMMAND LINES (I)

- To compile:

```
javac *.java
```

- To launch Java Program:

```
java MyMain
```

- To list server sockets:

```
netstat -tlp
```

- To list running connections:

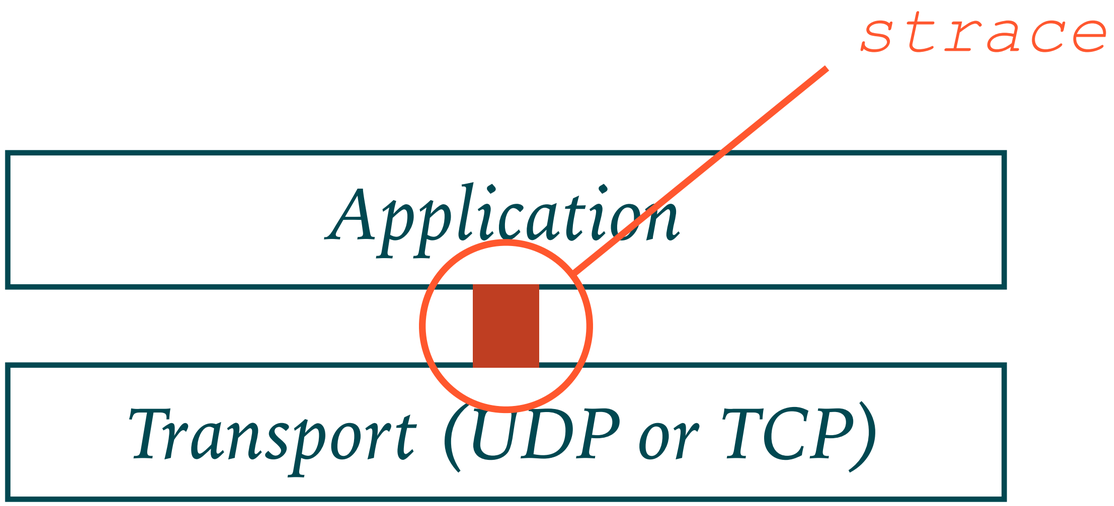
```
netstat -tcp
```

- To track system calls issued by your program:

```
strace -e trace=network -f java MyMain
```

SOME COMMAND LINES (II)

- `Strace` *live example:*



SOME COMMAND LINES (III)

- `Strace` *example output:*

Syscall and its arguments ("`man <syscall>`" to know about the args)

```
[pid 171345] sendto(5, "\1\0\r\4todo\7victory", 16, 0, NULL, 0) = 16
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, 0, NULL, NULL) = 6
[pid 171345] sendto(5, "\1\0\16\4todo\10position", 17, 0, NULL, 0) = 17
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, 0, NULL, NULL) = 6
[pid 171345] recvfrom(5, "\1\1\16\10position\4H6:3\1\1\16\10position\4I4"
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
[pid 171345] sendto(5, "\1\1\t\5guess\2I4", 12, 0, NULL, 0) = 12
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, MSG_DONTWAIT, NULL, NULL) = 6
[pid 171345] recvfrom(5, "\1\1\16\10position\4G2:2\1\1\16\10position\4G6"
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
```

SOME COMMAND LINES (II)

- `Strace` *example output:*

The bytes to send

```
[pid 171345] sendto(5, "\1\0\r\4todo\7victory", 16, 0, NULL, 0) = 16
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, 0, NULL, NULL) = 6
[pid 171345] sendto(5, "\1\0\16\4todo\10position", 17, 0, NULL, 0) = 17
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, 0, NULL, NULL) = 6
[pid 171345] recvfrom(5, "\1\1\16\10position\4H6:3\1\1\16\10position\4I4"
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
[pid 171345] sendto(5, "\1\1\t\5guess\2I4", 12, 0, NULL, 0) = 12
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, MSG_DONTWAIT, NULL, NULL) = 6
[pid 171345] recvfrom(5, "\1\1\16\10position\4G2:2\1\1\16\10position\4G6"
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
```

SOME COMMAND LINES (II)

- `Strace` *example output:*

The number of bytes to send

```
[pid 171345] sendto(5, "\1\0\r\4todo\7victory", 16, 0, NULL, 0) = 16
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, 0, NULL, NULL) = 6
[pid 171345] sendto(5, "\1\0\16\4todo\10position", 17, 0, NULL, 0) = 17
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, 0, NULL, NULL) = 6
[pid 171345] recvfrom(5, "\1\1\16\10position\4H6:3\1\1\16\10position\4I4"
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
[pid 171345] sendto(5, "\1\1\t\5guess\2I4", 12, 0, NULL, 0) = 12
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, MSG_DONTWAIT, NULL, NULL) = 6
[pid 171345] recvfrom(5, "\1\1\16\10position\4G2:2\1\1\16\10position\4G6"
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
```

SOME COMMAND LINES (II)

- `Strace` *example output:*

The number of bytes that have actually been sent (-1 on error)

```
[pid 171345] sendto(5, "\1\0\r\4todo\7victory", 16, 0, NULL, 0) = 16
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, 0, NULL, NULL) = 6
[pid 171345] sendto(5, "\1\0\16\4todo\10position", 17, 0, NULL, 0) = 17
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, 0, NULL, NULL) = 6
[pid 171345] recvfrom(5, "\1\1\16\10position\4H6:3\1\1\16\10position\4I4"
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
[pid 171345] sendto(5, "\1\1\t\5guess\2I4", 12, 0, NULL, 0) = 12
[pid 171345] recvfrom(5, "\1\2\3\2OK", 258, MSG_DONTWAIT, NULL, NULL) = 6
[pid 171345] recvfrom(5, "\1\1\16\10position\4G2:2\1\1\16\10position\4G6"
[pid 171345] sendto(5, "\1\2\3\2OK", 6, 0, NULL, 0) = 6
```