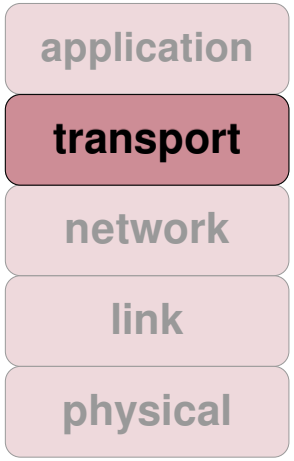
UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Computer Networks II



UDP Chat

Step 1: Hello world

- Write a client and a server that, using UDP, provide the following features:
  - Client: it sends a message containing “hello”
  - Server: it prints the received message on screen

Step 1: Client

```
#!/usr/bin/python3

from socket import *

sock = socket(AF_INET, SOCK_DGRAM)
sock.sendto("hello".encode(),
            ('localhost', 12345))
sock.close()
```

shebang: special comment
to instruct the shell that this program must be executed using the python3 interpreter

The socket class is
at the **socket** module
(we use '*' only in slides)

Socket family IPv4 (AF_INET)
Socket type UDP (SOCK_DGRAM)

Destination is indicated by the
tuple (address, port).

To make the file executable, you should change permisions properly:

```
$ chmod u+x client.py
$ ./client.py
```

Step 1: Server

```
#!/usr/bin/python3

from socket import *

sock = socket(AF_INET, SOCK_DGRAM)
sock.bind(('', 12345))
msg, client = sock.recvfrom(1024)
print(msg.decode(), client)
sock.close()
```

method **bind( )** attaches the server socket to a specific IP address and port

Recibing buffer has a limit of 1024 bytes

The result is a tuple with the format (data, tuple(address, port)).

Execute the server in a terminal and the client in another one.
How can you verify that the server is listening in the port 12345?

Step 2: “**Hello**” **with reply**

- Modify the previous exercise to allow to the server send back a reply to the client. The client should print the server reply in its screen.

Step 2: Solution

client

```
#!/usr/bin/python3
from socket import *

sock = socket(AF_INET, SOCK_DGRAM)
sock.sendto("hello".encode(), ('localhost', 12345))
msg, server = sock.recvfrom(1024)
print(msg.decode(), 'from', server)
sock.close()
```

Client receives a message from server

server

```
#!/usr/bin/python3
from socket import *

sock = socket(AF_INET, SOCK_DGRAM)
sock.bind(('', 12345))
msg, client = sock.recvfrom(1024)
print(msg.decode(), 'from', client)
sock.sendto('hi'.encode(), client)
sock.close()
```

Server sends a message to client

Step 3: User input

- Write a simple client-server chat application
  - Bidirectional communication, but not simultaneous
    - The client initiates conversation
  - Application ends if any of client or server sends the string “bye”.
- Hints:
  - You can get keyboard input using funtion **`input()`**
  - To go out of a loop use the **`break`** keyword.

Step 4: Simultaneous

- Modify the previous exercise so any of the peers to be able to send and receive at any time.
- Hints: use threads

```
import _thread

def receive(<args>):
    # reciving task
...
_thread.start_new_thread(receive, (<tuple args>))
...
```