Reti di Calcolatori I

Prof. Roberto Canonico
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea in Ingegneria Informatica

A.A. 2018-2019

Esempi di programmi client/server in Python 3

I lucidi presentati al corso sono uno strumento didattico che NON sostituisce i testi indicati nel programma del corso

Università degli Studi di Napoli Federico II

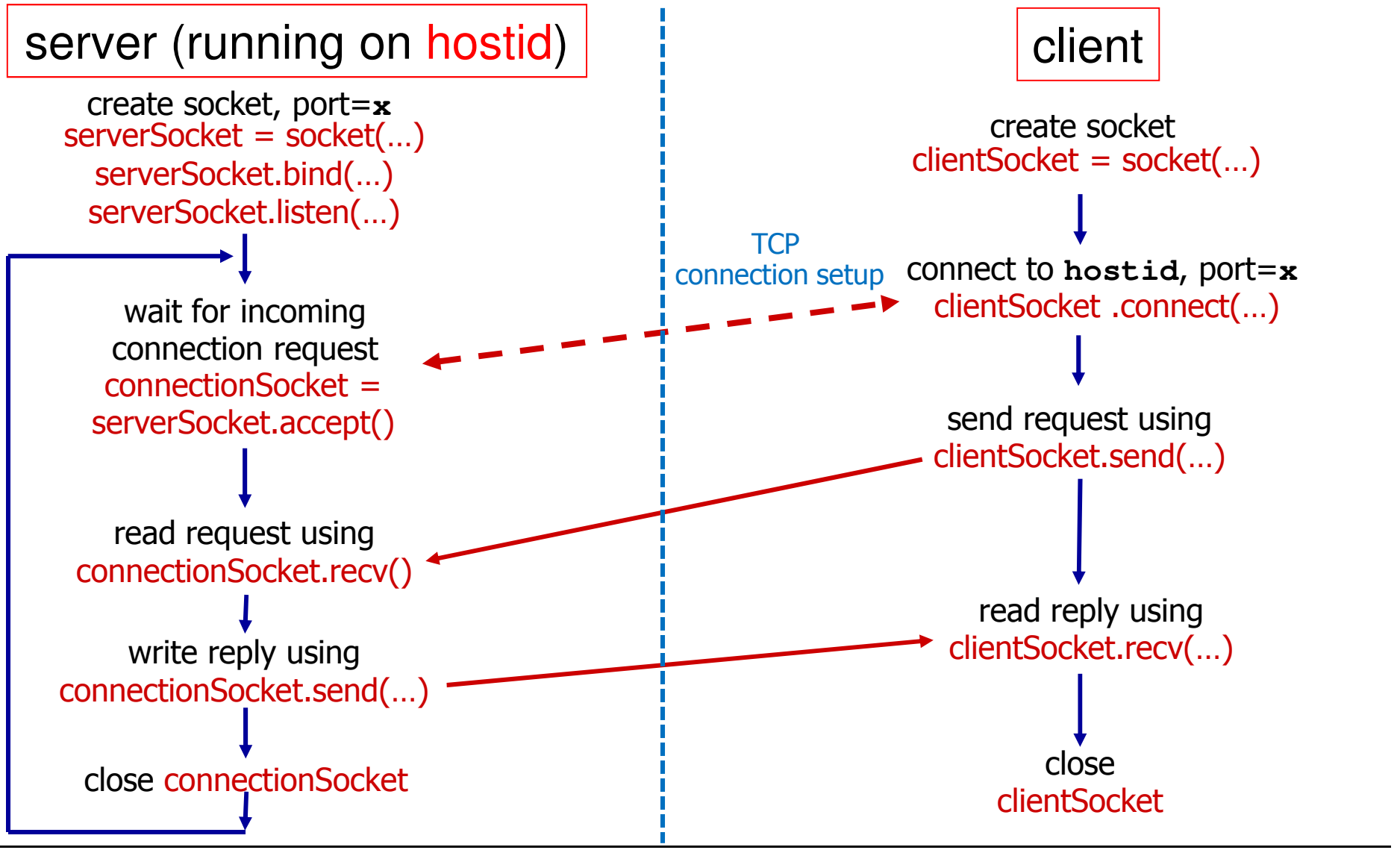
Nota di copyright per le slide COMICS

Nota di Copyright

Autori:
Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,
Marcello Esposito, Roberto Canonico, Giorgio Ventre

Università degli Studi di Napoli
Federico II

Client/server con TCP: schema



TCP client in Python 3 (1/3)

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "127.0.0.1"
SERVER_PORT = 12000
# read command line arguments
read_args()
# get user keyboard input
sentence = input("Input lowercase sentence: ")
# create TCP socket
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# connect socket to remote server
clientSocket.connect((SERVER_ADDRESS, SERVER_PORT))
print("Connected to server at (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
# Send sentence into socket, no need to specify server IP and port
request = sentence.encode()
clientSocket.send(request)
# read reply message from socket into response
response = clientSocket.recv(1024)
# print out received response
print("Received reply <%s>" % response.decode())
# close socket
clientSocket.close()
print("Connection closed with server at (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
```

TCP client in Python 3 (2/3)

```
# ...

def usage():
    print("Usage: %s [-a X.X.X.X] [-p N]" % sys.argv[0])
    print("Arguments:")
    print("-a X.X.X.X --> server listening address (default 127.0.0.1)")
    print("-p N  --> server listening port number in 1024..65535 (default 12000)")
    print("Flags:")
    print("-h    --> help")

# ...
```

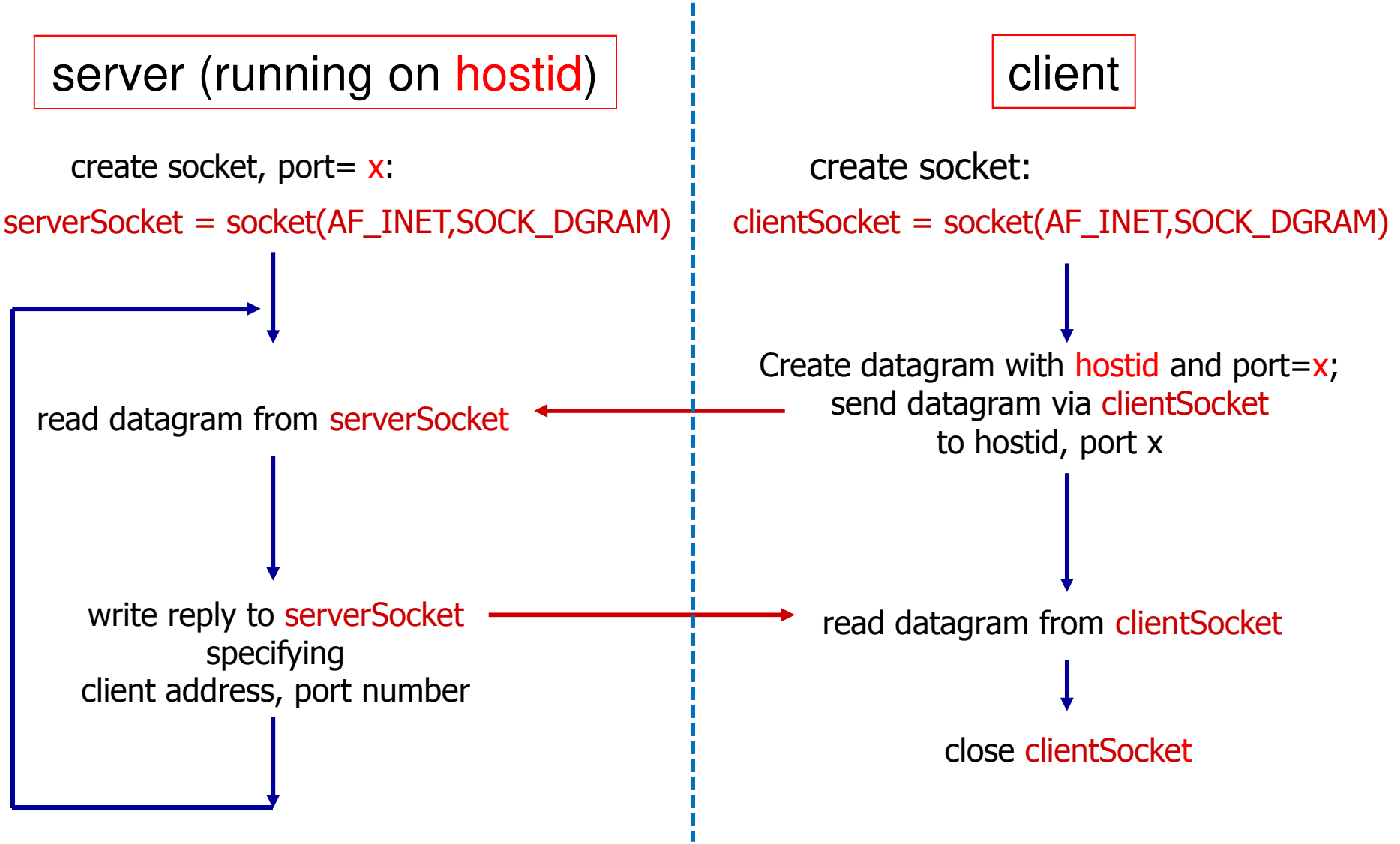
TCP client in Python 3 (3/3)

```
# ...
def read_args():
    global SERVER_ADDRESS, SERVER_PORT
    try:
        opts,args= getopt.getopt(sys.argv[1:],'ha:p:',['help','address=','port='])
    except getopt.GetoptError as err:
        print("ERROR: ", err)
        usage(); sys.exit()

    for opt, arg in opts:
        if opt in ('-h', '--help'):
            usage(); sys.exit()
        elif opt in ('-a', '--address'):
            SERVER_ADDRESS = arg
        elif opt in ('-p', '--port'):
            if (arg.isnumeric()) and (int(arg) >= 1024) and (int(arg) <= 65535):
                SERVER_PORT = int(arg)
            else:
                print("ERROR: -p option must be an integer in 1024..65535")
                usage(); sys.exit()
        else:
            print("Unrecognized argument !")
            usage(); sys.exit()

# ...
```

TCP server in Python 3

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "0.0.0.0"
SERVER_PORT = 12000
# read command line arguments
read_args()
# create TCP socket
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket.bind((SERVER_ADDRESS, SERVER_PORT)) # bind socket to local port
serverSocket.listen(1)                           # put socket in passive mode
while True:
    # wait for incoming connections on accept(), new socket created on return
    print("Server waiting on (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
    connectionSocket, clientAddress = serverSocket.accept()
    # receive request on newly established connectionSocket
    request = connectionSocket.recv(1024)
    print("Received request from (%s, %d)" % (clientAddress[0], clientAddress[1]))
    # convert message to upper case
    response = request.upper()
    # send back modified string to client
    connectionSocket.send(response)
    print("Sent reply to (%s, %d)" % (clientAddress[0], clientAddress[1]))
    connectionSocket.close()
```

Client/server con UDP: schema



UDP client in Python 3

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "127.0.0.1"
SERVER_PORT = 12000
# read command line arguments
read_args()
# get user keyboard input
request = input("Input lowercase sentence: ")
# create UDP socket
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# send request to server
clientSocket.sendto(request.encode(), (SERVER_ADDRESS, SERVER_PORT))

# wait for server response
print("Waiting for server response...")
response, serverAddress = clientSocket.recvfrom(2048)

# Print out received response
print(response.decode())

# Close socket
clientSocket.close()
```

UDP server in Python 3

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "0.0.0.0"
SERVER_PORT = 12000
# read command line arguments
read_args()
# create UDP socket
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverSocket.bind((SERVER_ADDRESS, SERVER_PORT)) # bind socket to local port
serverSocket.listen(1)                           # put socket in passive mode
while True:
   print("Server waiting on (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
   # receive request
   request, clientAddress = serverSocket.recvfrom(2048)
   print("Received request from (%s, %d)" % (clientAddress[0], clientAddress[1]))
   # convert message to upper case
   response = request.upper()
   # send back modified string to client
   serverSocket.sendto(response, clientAddress)
   print("Sent reply to (%s, %d)" % (clientAddress[0], clientAddress[1]))
```

HTTP client in Python 3 (1/2)

```
import socket
from urllib.parse import urlparse

while True:
    url = input("Type the object's URL: ")
    if (url == ""):
        break
    elif (url == "*"):
        url = "http://127.0.0.1:8080/"

    o = urlparse(url)
    if (o.hostname != None):
        SERVER_ADDRESS = o.hostname
    else:
        print("Error in parsing URL <%s>. Retry." % url);  continue

    if (o.port != None):
        SERVER_PORT = o.port
    else:
        SERVER_PORT = 80

    if (o.path != None):
        PATH = o.path
    else:
        PATH = "/"
#...
```

HTTP client in Python 3 (2/2)

```
#...
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print("Connecting to server at (%s, %d) " % (SERVER_ADDRESS, SERVER_PORT))
    try:
        clientSocket.connect((SERVER_ADDRESS, SERVER_PORT))
    except OSError as err:
        print("Error. Retry."); continue

    request = "GET %s HTTP/1.0\r\n\r\n" % PATH
    print(request)
    clientSocket.send(request.encode('ascii'))

    print("Waiting for response ", end="... ")
    reply = clientSocket.recv(1500)
    print("received")
    clientSocket.close()

    print("Response:")
    print(reply.decode('ascii'))
```

HTTP server in Python 3

```
import http.server
import socketserver

Handler = http.server.SimpleHTTPRequestHandler
DEFAULT_PORT = 8080

try:
    port = int(sys.argv[1])
except:
    port = DEFAULT_PORT

print ("Serving at port %s/tcp ..." % port)

httpd = socketserver.TCPServer(("0.0.0.0", port), Handler)
httpd.serve_forever()...
```

UDP multicast listener in Python 3 (1)

```
import socket, struct, sys

MCAST_GROUP = '224.3.29.71'
MCAST_PORT = 12000

# Create the socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)

#allow multiple sockets to use the same PORT number
try:
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
except AttributeError:
    pass    # Some systems don't support SO_REUSEPORT

try:
    sock.bind(('0.0.0.0', MCAST_PORT))
except OSError:
    print("Multiple listeners not allowed. Exiting.")
    sys.exit()
```

UDP multicast listener in Python 3 (2)

```
# ...

# Join the multicast group on all interfaces
group = socket.inet_aton(MCAST_GROUP)
mreq = struct.pack('4sL', group, socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

# Receive/respond loop
while True:
    print('\nwaiting to receive message')
    data, address = sock.recvfrom(1024)

    print('received %s bytes from %s' % (len(data), address))
    print(data)

    print('sending acknowledgement to', address)
    sock.sendto(b'ack', address)
```

UDP multicast sender in Python 3 (1)

```
import socket, struct, sys

MCAST_GROUP = '224.3.29.71'
MCAST_PORT = 12000
message = b'very important data'

# Create the datagram socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Set a timeout so the socket does not block indefinitely when trying
# to receive data.
sock.settimeout(0.2)

# Set the time-to-live for messages to 1 so they do not go past the
# local network segment.
ttl = struct.pack('b', 1)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)
# ...
```

UDP multicast sender in Python 3 (2)

```
#...
try:
    # Send data to the multicast group
    print('sending <%s>' % message.decode())
    sent = sock.sendto(message, (MCAST_GROUP, MCAST_PORT))

    # Look for responses from all recipients
    while True:
        print('waiting to receive ...')
        try:
            data, server = sock.recvfrom(16)
        except socket.timeout:
            print('timed out, no more responses')
            break
        else:
            print('received <%s> from %s' % (data.decode(), server))

finally:
    print('closing socket')
    sock.close()
```