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 2

**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

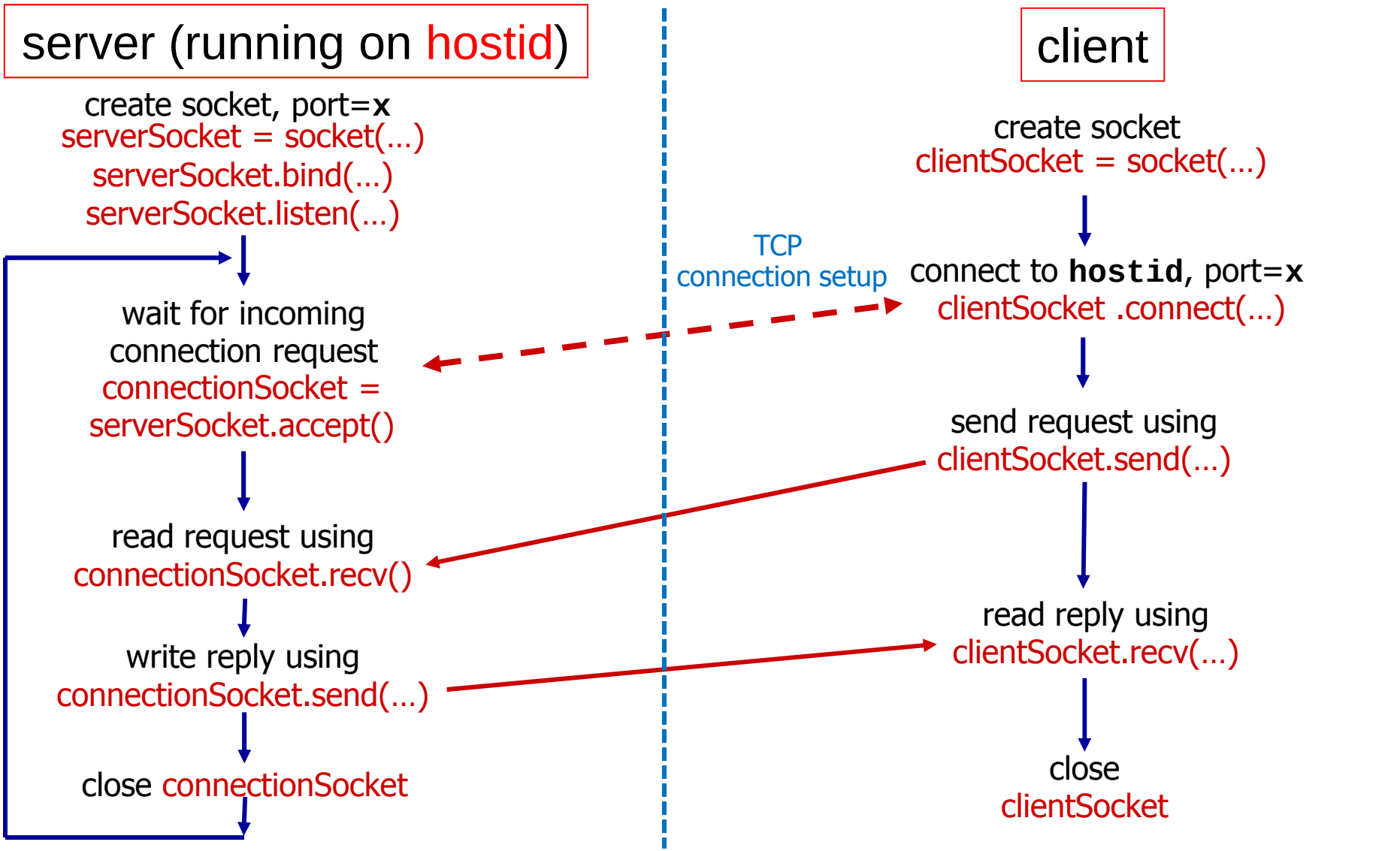
Nota di copyright per le slide COMICS

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



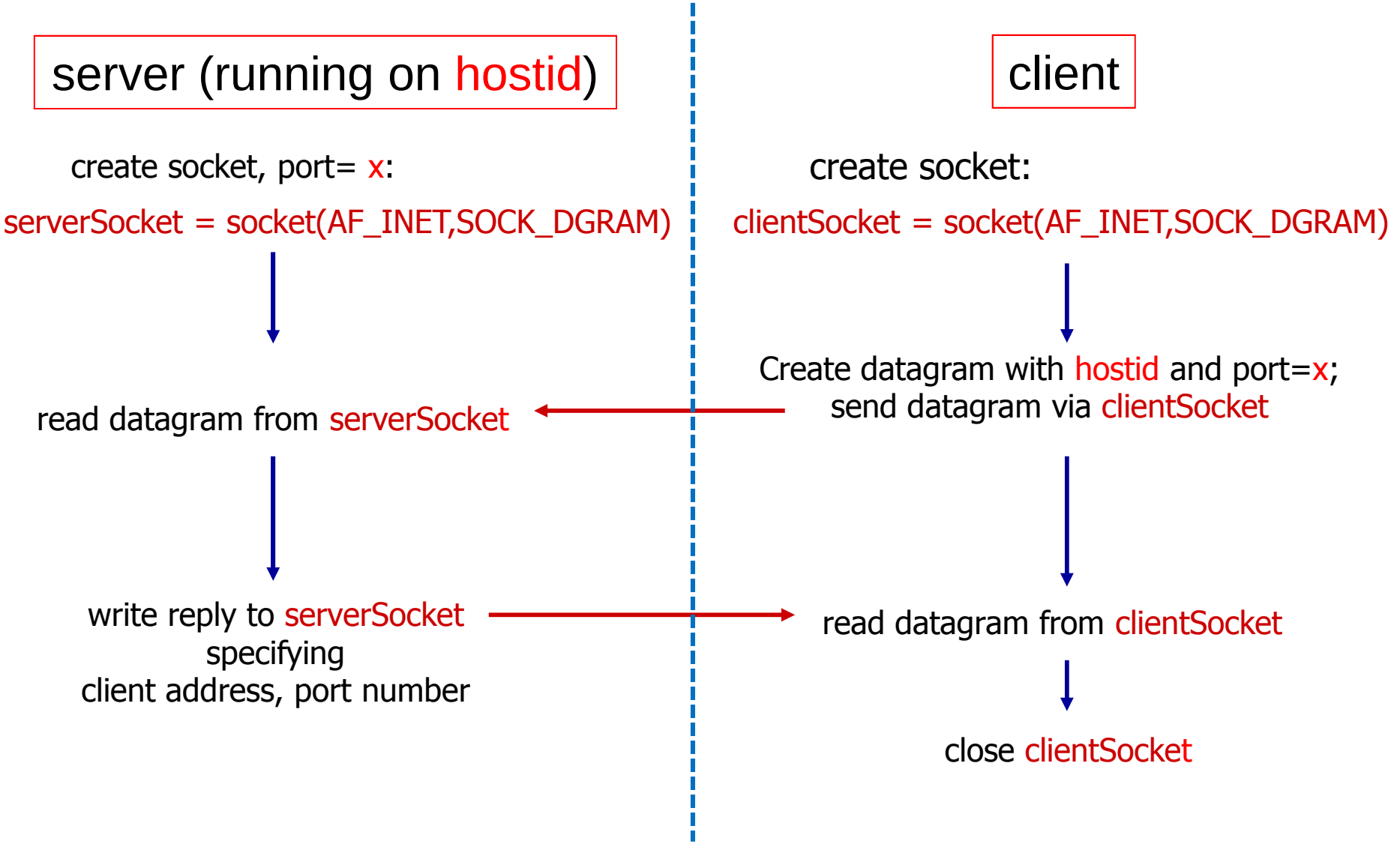
TCP client in Python 2

```
# -*- coding: utf-8 -*-
# include Python's socket library
from socket import *
serverName = 'localhost'
serverPort = 12000

# create TCP socket
clientSocket = socket(AF_INET, SOCK_STREAM)
# connect socket to remote server at (serverName, serverPort)
clientSocket.connect((serverName, serverPort))
# get user keyboard input
sentence = raw_input('Input lowercase sentence:')
# Send sentence into socket, no need to specify server IP and port
clientSocket.send(sentence)

# read reply message from socket into modifiedMessage string
modifiedSentence = clientSocket.recv(1024)

# Print out received modifiedMessage string
print modifiedSentence
# Close socket
clientSocket.close()
```

TCP server in Python 2

```
# -*- coding: utf-8 -*-
# include Python's socket library
from socket import *
serverPort = 12000
# create TCP socket
serverSocket = socket(AF_INET, SOCK_STREAM)
# bind socket to local port number 12000
serverSocket.bind(('', serverPort))
# put socket in passive mode
serverSocket.listen(1)
print "The server is ready to receive"
# Loop forever
while 1:
    # server waits for incoming connections on accept()
    # for incoming requests, new socket created on return
    connectionSocket, addr = serverSocket.accept()
    # receive sentence on newly established connectionSocket
    sentence = connectionSocket.recv(1024)
    # convert message to upper case
    modifiedSentence = sentence.upper()
    # send back modified string to client
    connectionSocket.send(modifiedSentence)
```

Client/server con UDP: schema



UDP client in Python 2

```
# -*- coding: utf-8 -*-
# include Python's socket library
from socket import *
serverName = 'localhost'
serverPort = 12000

# create UDP socket
clientSocket = socket(AF_INET, SOCK_DGRAM)

# get user keyboard input
message = raw_input('Input lowercase sentence:')

# Attach server name, port to message; send into socket
clientSocket.sendto(message, (serverName, serverPort))

# read reply message from socket into modifiedMessage string
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

# Print out received modifiedMessage string
print modifiedMessage
# Close socket
clientSocket.close()
```

UDP server in Python 2

```
# -*- coding: utf-8 -*-
# include Python's socket library
from socket import *
serverPort = 12000
# create UDP socket
serverSocket = socket(AF_INET, SOCK_DGRAM)

# bind socket to local port number 12000
serverSocket.bind(('', serverPort))
print "The server is ready to receive"

# Loop forever
while 1:
    # Read from UDP socket into message
    # getting client IP and port
    message, clientAddress = serverSocket.recvfrom(2048)
    # Convert message to upper case
    modifiedMessage = message.upper()
    # Send back modified string to client
    serverSocket.sendto(modifiedMessage, clientAddress)
```

UDP multicast sender in Python 2 (1)

```
import socket
import struct
import sys

message = 'very important data'
multicast_group = ('224.100.100.100', 12000)

# 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 2 (2)

```
#...
try:

    # Send data to the multicast group
    print >>sys.stderr, 'sending "%s"' % message
    sent = sock.sendto(message, multicast_group)

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

finally:
    print >>sys.stderr, 'closing socket'
    sock.close()
```

UDP multicast listener in Python 2 (1)

```
import socket
import struct
import sys

MCAST_GROUP = '224.3.29.71'
ANY = '0.0.0.0'
MCAST_PORT = 10000

# Create the socket
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM,socket.IPPROTO_UDP)
#allow multiple sockets to use the same PORT number (only works in Linux)
try:
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
except AttributeError:
    pass # Some systems don't support SO_REUSEPORT

# Bind to the server address
sock.bind((ANY, MCAST_PORT))

#...
```

UDP multicast listener in Python 2(2)

```
#...
# Tell the operating system to add the socket to 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 >>sys.stderr, '\nwaiting to receive message'
    data, address = sock.recvfrom(1024)

    print >>sys.stderr,'received %s bytes from %s'%(len(data), address)
    print >>sys.stderr, data

    print >>sys.stderr, 'sending acknowledgement to', address
    sock.sendto('ack', address)
```