# Computer Network I

## Reti di Calcolatori I

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Informatica

Riccardo Caccavale

(riccardo.caccavale@unina.it)

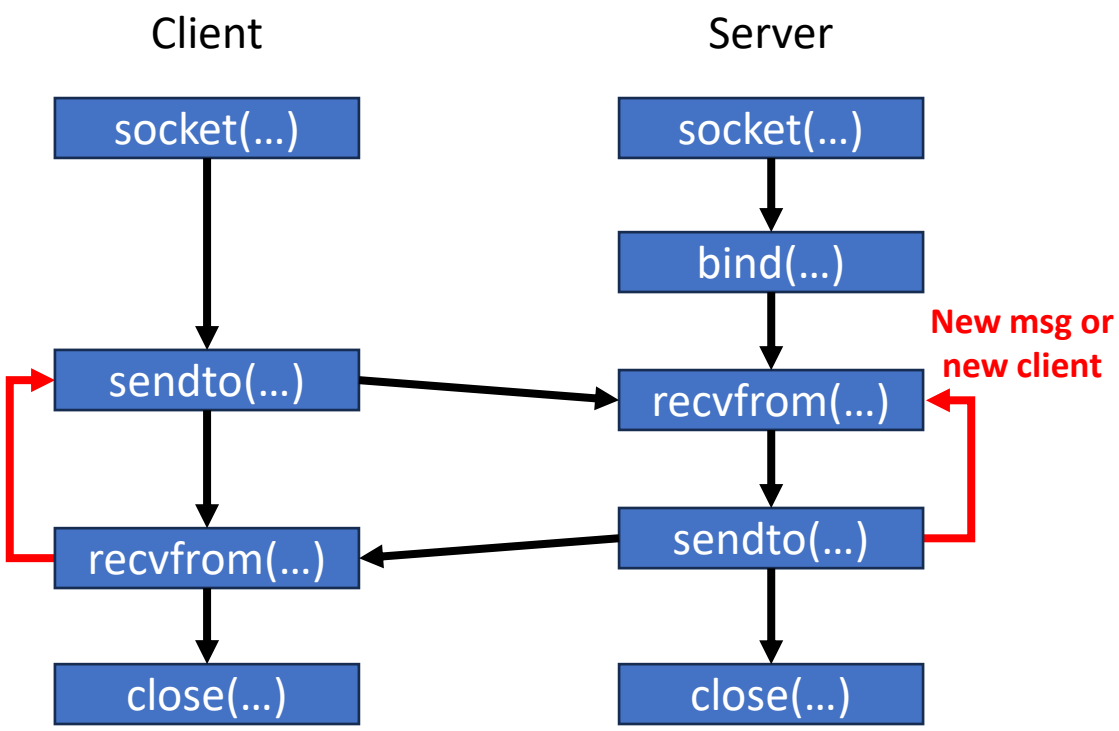
UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

DIETI

UNINA

# Application Layer

## UDP Socket Commands




- For UDP connection we need both client and server to open a socket.
- The server has to bind the socket to a specific port.
- Then we can exchange messages using sendto and recvfrom functions.
  - Here IP and port of the client are retrieved through the recvfrom function.
  - We may loop over sendto/recvfrom to keep alive the communication.
- When communication is over, both hosts close the socket.
  - Notice that if client only closes the socket, the server may accept other clients.

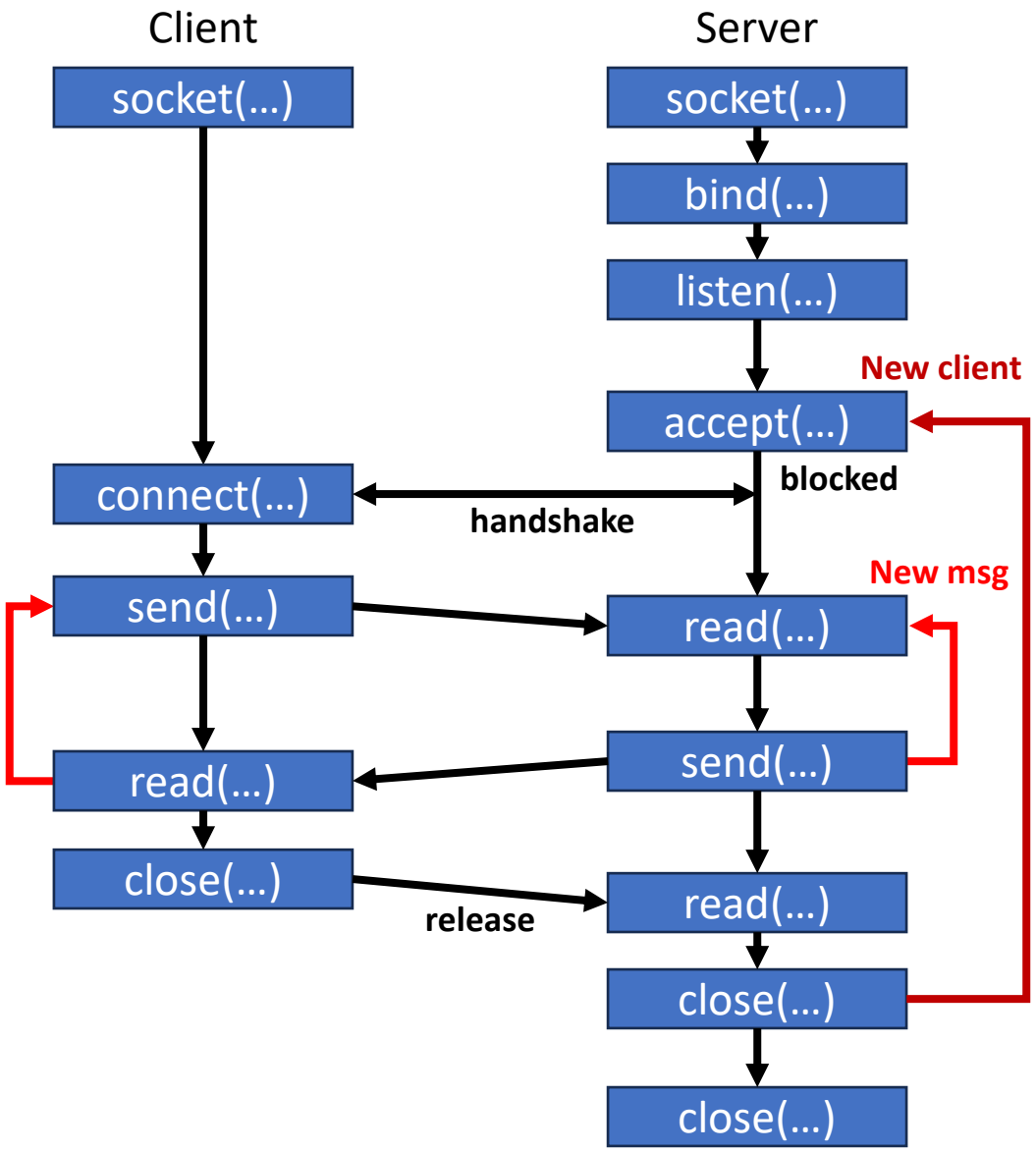
# Application Layer

## UDP Socket Programming (Keepalive Example C/C++)

Outsourced…

# Application Layer

## TCP Socket Commands




- For TCP connection we need both client and server to open a socket.
- The server has to bind the socket to a specific port, listen and eventually accept new connection. The client has to connect to the server.
- Then we can exchange messages using send and read functions.
  - Here IP and port of the client are retrieved from the newly created socket.
  - We may loop over send/read to keep alive the communication.
- When communication is over, both hosts close the socket.
  - Notice that the server should perform an additional read before to close the client-specific socket (connection release) otherwise we must wait about 4 minutes to reuse the port.
- The server may also go back to the accept function and welcome another client.

# Application Layer

## TCP Socket Programming (Keepalive Example C/C++)

Outsourced…

# Application Layer
## DNS Translation (C/C++)

- As just shown, we can create sockets between two hosts, but **we need to know (IP) address and port of the server**.

- On Internet, hosts are typically identified by **hostnames rather than by IP** addresses.

- Berkeley sockets (C/C++) only works with IP addresses.
  - This is not surprising as **sockets works on transport layer**, while **hostnames works on application layer**.

- If we are to connect with a host by hostname, **we need to use DNS translation** and get the associated IP address to be passed to the socket.

# Application Layer
## DNS Translation (C/C++)

- C/C++ offers some functions and structures that perform such translation for us, which are used **to contact DNS servers** and to retrieve addresses.

- In particular, we can use the function *gethostbyname()* to contact DNS servers.

- The above function **does not return just the IP address**. It returns a *hostent* structure that contains some info about the host:
  - Canonical name.
  - Possible aliases.
  - One or more addresses.

# Application Layer
## DNS Translation (C/C++)

- The hostent structure is defined as follows:

```
#include <netdb.h>

struct  hostent {
    char *  h_name;             // original name of the host (canonical)
    char ** h_aliases;          // list of aliases (terminated by a NULL pointer)
    int     h_addrtype;         // family of the address (typically AF_INET)
    int     h_length;           // length of the address (typically 4 bytes)
    char ** h_addr_list;        // list of addresses (terminated by a NULL pointer)
};

// for compatibility reasons
#define h_addr h_addr_list[0]
```

- A DNS may provide a list of addresses (which may be randomly ordered if the query is sent to round-robin DNS servers).
  - We can get the address of the first element of the array (h_addr_list[0] or h_addr).

- The returned addresses can be casted to *in_addr* structure used by sockets.

# Application Layer
## DNS Translation (C/C++)

- The gethostbyname() function is defined as follows:

```
#include <netdb.h>

struct hostent *host_info = gethostbyname(const char *name);
```

- Where:
  - name: is a string containing the hostname to be translated.
  - host_info: is a pointer to the hostent structure containing the information about host.
    - This is NULL if error occurred.

- Notice that there is also a similar function *gethostbyaddr()* that returns a hostent structure for a specified IP address.

# Application Layer

## DNS Translation (C/C++)

```
#include <iostream>
#include <cstring>
#include <string>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <stdio.h>
```

- Include libraries to allow DNS translation.

```
int main(int argc, char **argv){
   struct hostent *server_info;
   char *server_name;

   if(argc < 2){
      std::cout<<"No hostname specified"<<std::endl;
      return 0;
   }
   else {
      server_name = argv[1];
   }

   server_info = gethostbyname(server_name);
   if (server_name == NULL){
      std::cout << "could Not resolve hostname " << server_name << " :(" << std::endl;
      return 0;
   }
```

- Initialization.
- Get hostname as argument.
- Invoke DNS and check if a response has been received.

# Application Layer

## DNS Translation (C/C++)

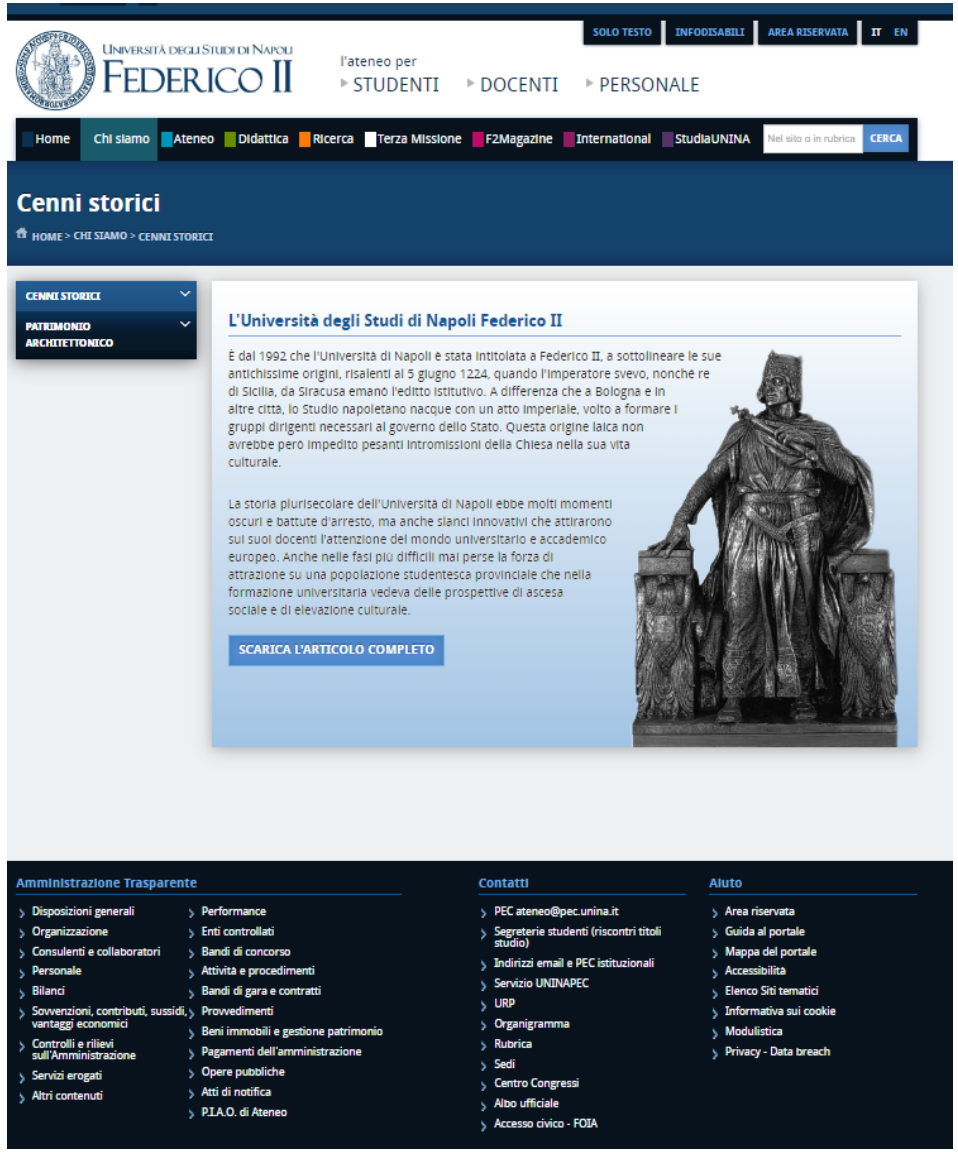
```
    //plot output
    std::cout<<"Canonical name:"<<std::endl;
    std::cout<<"\t"<<server_info->h_name<<std::endl;

    std::cout<<"Aliases:"<<std::endl;
    int i = 0;
    while(server_info->h_aliases[i] != NULL){
        std::cout<<"\t"<<server_info->h_aliases[i]<<std::endl;
        i++;
    }

    std::cout<<"IPs:"<<std::endl;
    i = 0;
    while(server_info->h_addr_list[i] != NULL){
        std::cout<<"\t"<< inet_ntoa( *( (struct in_addr *) server_info->h_addr_list[i] ) ) <<std::endl;
        i++;
    }

    return 0;
}
```

- Plotting output from hostent structure:
  - Canonical name.
  - List of aliases
  - List of IPs.

# Application Layer
## HTTP Socket Example

- We will now create a C++ code implementing from scratch a HTTP request (HEAD) for a web page.

- The web page we will check is the “cenni storici” from the UNINA website:
  - http://www.unina.it/chi-siamo/cenni-storici



# Application Layer

## HTTP Socket Example (C/C++)

```
#include <iostream>
#include <cstring>
#include <string>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <netdb.h>
#include <stdio.h>
```

- Include libraries to allow TCP connection to the web server.

```
int main(){
   int socket_desc;
   struct sockaddr_in serv_addr;
   struct hostent *server;
   char buffer[4096];

   socket_desc = socket(AF_INET, SOCK_STREAM, 0);
   if (socket_desc < 0){
      std::cout << "failed to create socket" << std::endl;
      return 0;
   }
```

- Initialize variables and create the TCP socket. For a HTTP request 3 elements are specified:
  - Hostname of the web server.
  - Port number.
  - Resource to be retrieved (object's path).
- Note: here we use the C++ cout for simplicity.

# Application Layer
## HTTP Socket Example (C/C++)

```
server = gethostbyname("www.unina.it");
if (server == NULL){
   std::cout << "could Not resolve hostname :("
      << std::endl;
   close(socket_desc);
   return 0;
}

bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(80);
bcopy((char *)server->h_addr, (char *)&serv_addr.sin_addr.s_addr, server->h_length);

if (connect(socket_desc, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0){
   std::cout << "connection failed :(" << std::endl;
   close(socket_desc);
   return 0;
}
```

- Connection to the web server. Here we use the following function:

  server = gethostbyname(host.c_str());

  that invokes DNS and to gets the IP of the server from the hostname (inside the *hostent* struct).

- The IP from the returned structure is copied into the *serv_addr* structure for the following connect function.

# Application Layer
## HTTP Socket Example (C/C++)

```
const char *request = "HEAD /chi-siamo/cenni-storici HTTP/1.1\r\nHost: www.unina.it\r\nConnection: close\r\n\r\n";

if (send(socket_desc, request, strlen(request), 0) < 0){
   std::cout << "failed to send request..." << std::endl;
   close(socket_desc);
   return 0;
}
std::cout<<"message sent:"<<std::endl;
std::cout<<request<<std::endl;

int n = recv(socket_desc, buffer, sizeof(buffer), 0);
```

- Messaging with the web server. The HTTP request is defined into the request string.
- Here we are creating a non-persistent connection since we just want to check one page.

```
std::cout<<"received "<<n<<" bytes:"<<std::endl;
std::cout<<buffer;

close(socket_desc);

return 0;
```

- Close socket and plot the received page (HTML).