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

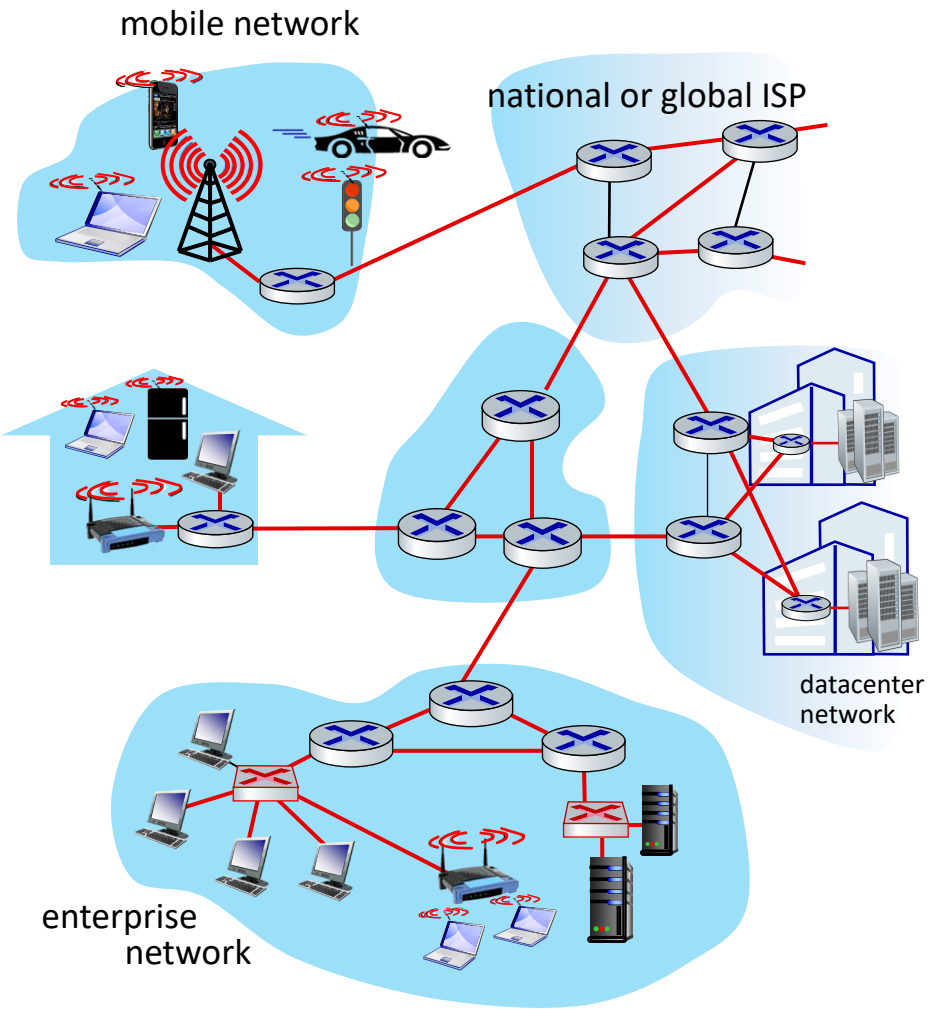
DIETI

UNINA

Network Programming

Introduction

- Network programming is the act of **creating network applications** that run on multiple, perhaps different, devices.
  - **Network applications** may be based on **different programming languages, code libraries, or OSs**.
  - **Devices** involved in a network applications **can be quite heterogeneous** (they may have different hardware, architecture, etc.).
- Such applications **rely on the network infrastructure** to communicate.
- Fortunately, the network infrastructure is **standardized**, we no need to worry about networking as most of the processes are “hidden” to us and are **managed (back-box) by sockets**.

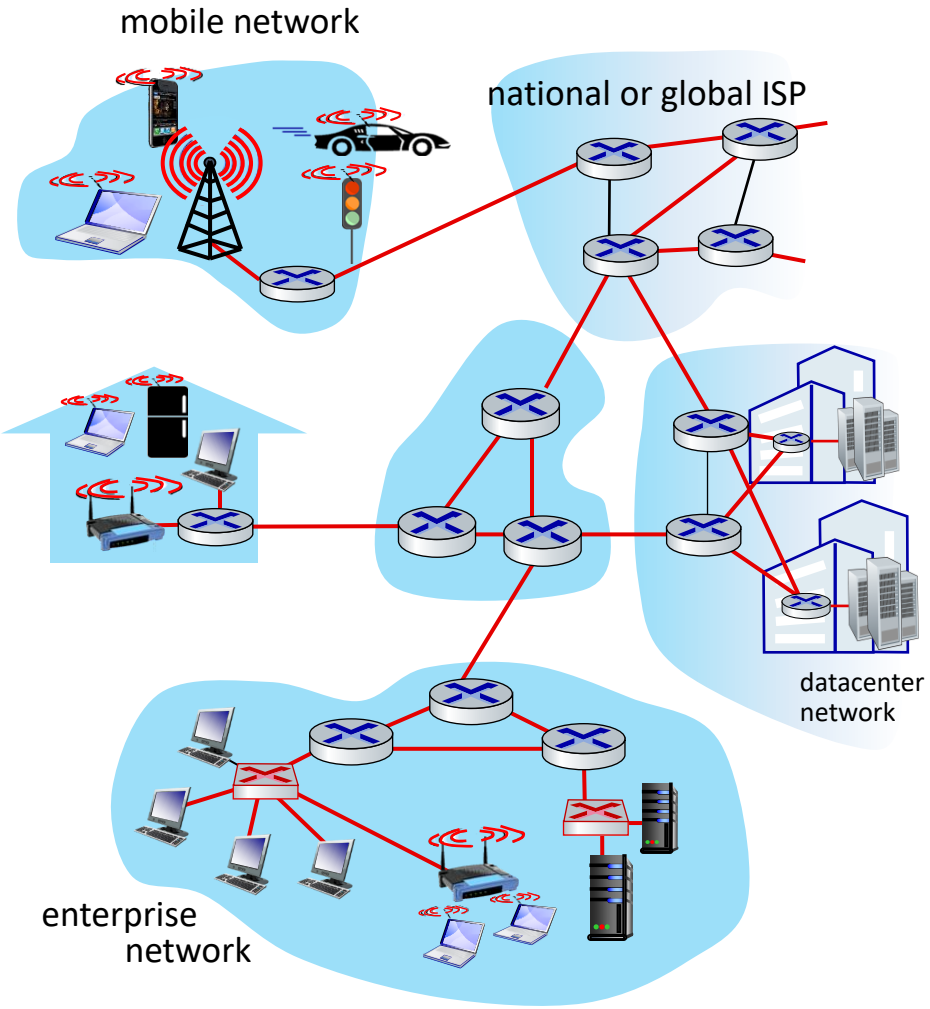


Network Programming

Introduction

- **Sockets are widely available** in different programming languages and OSs.

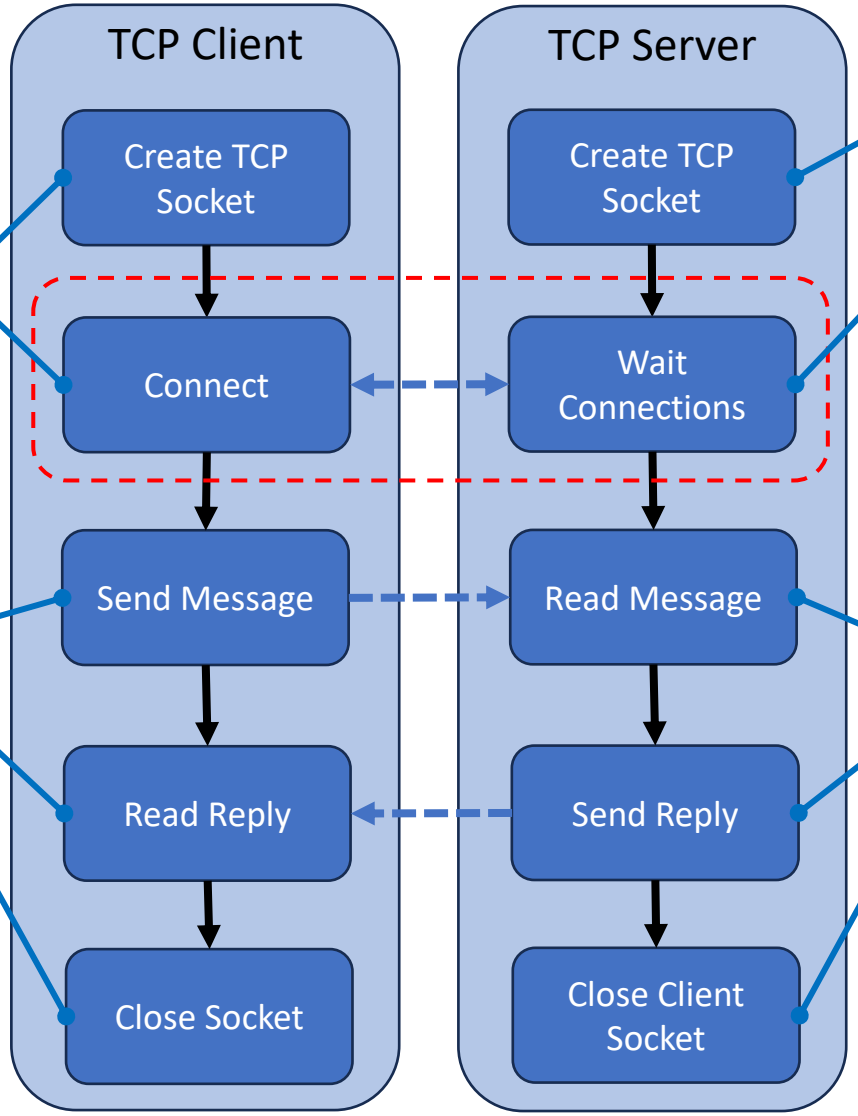
- Network programming often requires skills in **multiple programming areas and tools**:
  - Different programming languages.
    - Java, Python, C/C++, etc.
  - Operating Systems.
    - Linux, Windows, etc.
  - ISO/OSI stack and protocols.
    - HTTP, TCP, UDP, etc.
  - Data exchange formats.
    - JSON, YAML, XML, etc.
  - REST APIs.
  - …



Network Programming

TCP Socket Programming (Simple Example in Java)

```java
import java.net.*;
import java.io.*;

public class TCP_client {
   public static void main(String[] args) {
      if (args.length < 2) return;

      String client_name = args[0];
      String hostname = "127.0.0.1";
      int port = Integer.parseInt(args[1]);

      try (Socket socket = new Socket(hostname, port)) {

         OutputStream output = socket.getOutputStream();
         PrintWriter writer = new PrintWriter(output, true);

         InputStream input = socket.getInputStream();
         BufferedReader reader = new BufferedReader(
                  new InputStreamReader(input));

         writer.println("Hello from " + client_name);

         String msg = reader.readLine();
         System.out.println(msg);

         socket.close();
         System.out.println("Connection closed");

      } catch (UnknownHostException ex) {
         System.out.println("Server not found: " + ex.getMessage());
      } catch (IOException ex) {
         System.out.println("I/O error: " + ex.getMessage());
      }
   }
}
```



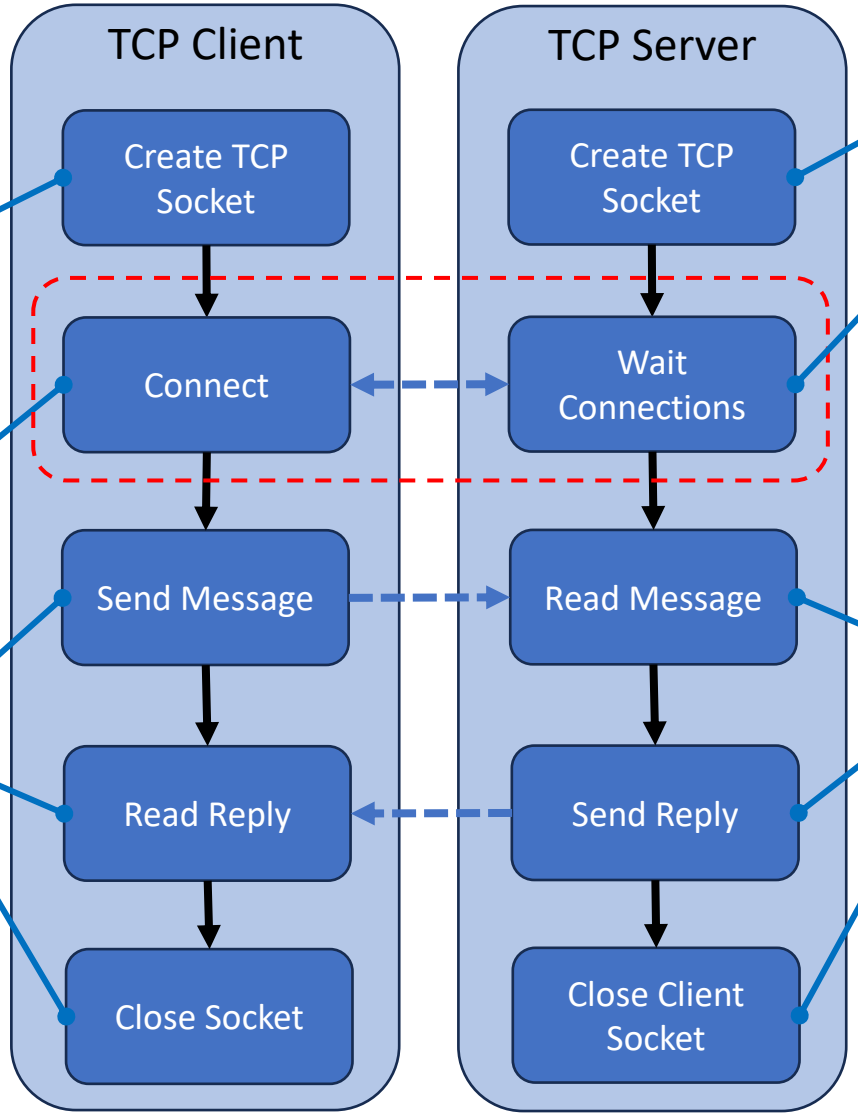

```java
import java.io.*;
import java.net.*;

public class TCP_server {
   public static void main(String[] args) {
      if (args.length < 1) return;

      int port = Integer.parseInt(args[0]);

      try (ServerSocket welcome_socket = new ServerSocket(port)) {

         System.out.println("Server is listening on port " + port);

         while (true) {
            Socket new_socket = welcome_socket.accept();

            InputStream input = new_socket.getInputStream();
            BufferedReader reader = new BufferedReader(
                     new InputStreamReader(input));

            OutputStream output = new_socket.getOutputStream();
            PrintWriter writer = new PrintWriter(output, true);

            System.out.println("New client connected");

            String msg = reader.readLine();
            System.out.println("Received: " + msg);

            writer.println("Hello from Server");

            new_socket.close();
            System.out.println("Connection closed");
         }

      } catch (IOException ex) {
         System.out.println("Server exception: " + ex.getMessage());
         ex.printStackTrace();
      }
   }
}
```

Network Programming

TCP Socket Programming (Example in C/C++ and Java)

```
// client-side (includes omitted)

int main() {
   int sockfd, status;
   char buffer[1024];
   const char *hello = "Hello from client";
   struct sockaddr_in servaddr;

   if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
      perror("socket creation failed");
      exit(EXIT_FAILURE);
   }

   memset(&servaddr, 0, sizeof(servaddr));

   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(8080);
   servaddr.sin_addr.s_addr = INADDR_ANY;

   int n;

   if ((status = connect(sockfd, (struct sockaddr*)&servaddr,
            sizeof(servaddr))) < 0) {
      printf("\nConnection Failed \n");
      return -1;
   }

   send(sockfd, hello, strlen(hello), 0);
   std::cout<<"Hello message sent."<<std::endl;

   n = read(sockfd, buffer, 1024);
   std::cout<<"Received \""<<buffer<<"\""<<std::endl;

   close(sockfd);

   return 0;
}
```

TCP client in C/C++ from previous lesson!




```
import java.io.*;
import java.net.*;

public class TCP_server {
   public static void main(String[] args) {
      if (args.length < 1) return;

      int port = Integer.parseInt(args[0]);

      try (ServerSocket welcome_socket = new ServerSocket(port)) {

         System.out.println("Server is listening on port " + port);

         while (true) {
            Socket new_socket = welcome_socket.accept();

            InputStream input = new_socket.getInputStream();
            BufferedReader reader = new BufferedReader(
                  new InputStreamReader(input));

            OutputStream output = new_socket.getOutputStream();
            PrintWriter writer = new PrintWriter(output, true);

            System.out.println("New client connected");

            String msg = reader.readLine();
            System.out.println("Received: " + msg);

            writer.println("Hello from Server");

            new_socket.close();
            System.out.println("Connection closed");
         }

      } catch (IOException ex) {
         System.out.println("Server exception: " + ex.getMessage());
         ex.printStackTrace();
      }
   }
}
```

Network Programming

Data Exchange

- In the previous examples we have just seen simple programs **exchanging strings** (plain text), but this is hardly a realistic case.

- Real network applications typically **exchange structured data** (structures, lists, etc.) so, a more complex representation of the information could be useful.

- Several **data exchange** (or interchange) **languages** (or formats) have been used in the years to provide **a standardized way of representing and exchanging structured data** for network applications.

- Because of their human-readability, **most common** (and open) formats used for data exchange are:
  - XML.
  - YAML.
  - JSON.

Network Programming

Data Exchange: XML

- The **XML** (eXtensible Markup Language) is a language designed to be **easily understandable by both humans and computers** (similar to HTML).
  - Typical file extension: .xml

- The core concept of XML are the **tags**. A **tag** is a markup construct wrapped in angle brackets (<…>), each content in an XML file is surrounded by the **start-tag** (<TAGNAME>) and the **end-tag** (</TAGNAME>).
  - Contents inside tags may be **simple values** (string, numbers, bool, etc.) or **other tags**.

- Differently from HTML, in XML **few tags are pre-defined**, we may **create tags at our discretion**.
  - XML documents are **conventionally** wrapped within <xml> ... </xml> tags, but these are rarely implemented for data exchange.

Network Programming

Data Exchange: XML

- This is a simple example of how to represent data about a person. Here we have:
  - A **root tag** <person> that collects the data about one person. It contains 3 inner tags:
    - The tag <personID> specifies the **ID of the person** (e.g., of a database) and contains a number.
    - The tag <firstName> contains a string with the **name of the person**.
    - The tag <lastName> contains a string with the **surname of the person**.

```
<person>
   <personID> 77 </personID>
   <firstName> John </firstName>
   <lastName> Doe </lastName>
</person>
```

- It is important to see that **all contents** (no matter how trivial) are wrapped inside start-tag and end-tag.

Network Programming

Data Exchange: XML

- It could be uncomfortable to specify all contents within tags. XML offers the possibility to **specify attributes and self-closing tags** to simplify content definition.

- A **self-closing tag** is a **standalone tag** that closes itself:

<TAGNAME />

- The **attributes** are **name-value pairs** that exist **within a start-tag or self-closing tag**. The value of attributes must be inside double quote (").

Start-tag with attribute

<TAGNAME ATT_NAME = "ATT_VALUE" >

...

</TAGNAME>

or

Self-closing tag with attribute

<TAGNAME ATT_NAME = "ATT_VALUE" />

Network Programming

Data Exchange: XML

- Following the previous example, we can use attributes and self-closing tags to represent a person in different ways:
  - Inserting some attributes into the start-tag.
  - Inserting all attributes into a self-closing tag.

```
<person personID = "77" >
   <firstName> John </firstName>
   <lastName> Doe </lastName>
</person>
```

```
<person personID = "77" firstName = "John" lastName = "Doe" />
```

Network Programming

Data Exchange: XML

- We can specify a group of people by nesting multiple person into an additional <people> tag.

```
<people>
   <person personID = "77" >
      <firstName> John </firstName>
      <lastName> Doe </lastName>
   </person>
   <person personID = "78" >
      <firstName> Alice </firstName>
      <lastName> Doe </lastName>
   </person>
</people>
```

Network Programming

Data Exchange: YAML

- The **YAML** (Yaml Ain't Markup Language) is a language designed to be **minimal and human-readable**.
  - Typical file extension: .yaml or .yml

- Differently from XML, in YAML **contents are defined through spaces and indentation** (as in python).
  - Note: **tab-based indentation is not allowed**, only whitespaces must be used.

- In YAML **names and values are separated by column** (NAME: VALUE) and there is a specific syntax to define elements such as **structures, lists or dictionaries**.

Network Programming

Data Exchange: YAML

- This is a simple example of how to represent data about a person. Here we have:
  - A “person” **structure** that contains 3 inner fields:
    - The field “personID” specifies the **ID of the person** (e.g., of a database) as a number.
    - The field “firstName” contains a string with the **name of the person**.
    - The field “lastName” contains a string with the **surname of the person**.

```
person:
   personID: 77
   firstName: John
   lastName: Doe
```

- It is possible to notice that **quotes are not necessary** to discriminate values.

Network Programming

Data Exchange: YAML

- The YAML we can also represent **lists** and **dictionaries**.

- A **list** can be represented in 2 ways:
  1. As a **structure whose elements are preceded by a dash** (-).
  2. As a sequence of comma-separated **elements inside squared brackets** ([E1, E2, …]).

- A **dictionary** is represented as a sequence of **comma-separated name-value couples inside curly brackets** ({ N1: V1, N2: V2, … }).

Network Programming

Data Exchange: YAML

- Extending the previous example, we can define a **list of people containing 2 person structures**.
- We can define a **list of names**.
- We can define **fields of a structure** (the person in this case) **as a dictionary**.
  - This method is **widely used for data exchange**.

```
people:
   - person:
        personID: 77
        firstName: John
        lastName: Doe
   - person:
        personID: 78
        firstName: Alice
        lastName: Doe
```

```
names: [John, Alice, Bob]
```

```
person: {personID: 77, firstName: John, lastName: Doe}
```

Network Programming

Data Exchange: JSON

- The **JSON** (JavaScript Object Notation) is a simple format for data exchange which is **human-readable**, **easily parsed** by machines, and is **based on C-like conventions** for data representation.
  - Typical file extension: .json

- JSON is somehow a **tradeoff between simplicity and effectiveness**, it is considered an **ideal data-interchange language** (one of the most used).
  - C-like convention is also well-known by programmers.

Network Programming

Data Exchange: JSON

- As in YAML, **JSON data** is specified as **a pair of name and value divided by a colon** (:) symbol where the name is a string (inside double quotes “”):

  “DATANAME”: DATAVALUE

- **JSON objects** are **groups of data inside curly brackets** where data items are separated by commas ({ “N1”: V1, “N2”: V2, … }).

- **JSON arrays** are represented as comma-separated **data or objects** within **square brackets** ([“N1”: V1, “N2”: V2, …]).

Network Programming

Data Exchange: JSON

- Differently from YAML, the **values for JSON data** have c-like syntax for different types. Possible values are:
  - **String** (inside double quotes):
    - “name” : “Bob”
  - **Number** (integer, float, double):
    - “age” : 27
    - “weight” : 60.5
  - **Array** (containing generic JSON data):
    - “pets” : [“cat”, “dog”]
    - “siblings” : [ ]
  - **Boolean** (true/false):
    - “isAlive” : true
  - **Null** (e.g., not available):
    - “phoneNumber” : null
  - **JSON object**.

Network Programming

Data Exchange: JSON

- This is an example of how to represent data about a person. Here we have:
  - A **root object** that contains the “person” data, whose value is an additional JSON object having 3 fields:
    - The “personID” specifies the **ID of the person** (e.g., of a database) and contains a number.
    - The “firstName” contains a string with the **name of the person**.
    - The “lastName” contains a string with the **surname of the person**.

```
{
   "person": {
      "personID": 77,
      "firstName": "John",
      "lastName": "Doe"
   }
}
```

- In this case the **indentation is not necessary** (as in C), but it is just used for a better visualization.

Network Programming

Data Exchange: JSON

- Also in this case, we can define a list of people by putting multiple JSON objects within a JSON array.

```
{
   "people": [
      "person": {
         "personID": 77,
         "firstName": "John",
         "lastName": "Doe"
      },
      "person": {
         "personID": 78,
         "firstName": "Alice",
         "lastName": "Doe"
      }
   ]
}
```

Network Programming

Data Exchange: JSON

- Also in this case, we can define a list of people by putting multiple JSON objects within a JSON array.

- We may also simplify the syntax by removing the “person” data.
  - Notice that JSON array may contain JSON data as well as JSON objects.

```
{
   "people": [
      {
         "personID": 77,
         "firstName": "John",
         "lastName": "Doe"
      },
      {
         "personID": 78,
         "firstName": "Alice",
         "lastName": "Doe"
      }
   ]
}
```

Network Programming

JSON Examples (Java and C++)

Outsourced…