Stefano Scanzio (stefano.scanzio[at]unimib.it)

Practice

Systems for Industry 4.0 and Environment (IoT)

www: https://www.skenz.it/iot

Presentation

Stefano Scanzio

skenz.it/ss

Stefano Scanzio

CNR – National Research Council of Italy

Contact:
Email: stefano.scanzio[at]unimib.it
Telegram: https://t.me/zioskenz

www: https://www.skenz.it/ss

(link to courses, CV, publications, theses)

Outline

Arguments

- MQTT
  - Lightweight message protocol designed for IoT
  - Integrated with [1]
- Wireshark
  - Network protocol analyzer for monitor network traffic
- JSON
  - data interchange format used for structured data representation and exchange
  - Integrated with [2]
- tkinter
  - Python library to design graphical interfaces
- URI
  - A protocol based on string to identify a resource on the internet

[1] https://robot.unipv.it/toolleeo/teaching/docs_iot/message_passing_handout.pdf
[2] https://robot.unipv.it/toolleeo/teaching/docs_iot/rest_api_handout.pdf

Outline (2)

Arguments

- HTTP and HTTPS
  - Protocols that enable data exchange between client and server
- request
  - Python library that simplifies sending HTTP requests
- REST API
  - Web service architecture using HTTP methods for creating, reading, updating, and deleting resources
  - Integrated with [2]
- flask
  - Lightweight Python web framework for building web applications with flexibility and minimal setup

[1] https://robot.unipv.it/toolleeo/teaching/docs_iot/message_passing_handout.pdf
[2] https://robot.unipv.it/toolleeo/teaching/docs_iot/rest_api_handout.pdf

Paho MQTT

- **Paho MQTT** is the most popular MQTT client library in the Python
  - Support of different versions: MQTT v5.0, MQTT v3.1.1, and v3.1
  - Open-source
  - Easy-to-use API
  - Actively developed and maintained
- Guide
  - https://pypi.org/project/paho-mqtt/
- Installation

```
pip3 install paho-mqtt
```

Paho MQTT Client

How to create a MQTT Client (`Client` class):

- Create a `Client` instance
- Connect to a broker (*`connect*()`* functions)
- Maintain network traffic flow with the broker (*`loop*()`* functions)
- Subscribe to a topic and receive messages (*`subscribe()`* function)
- Publish messages to the broker (*`publish()`* function)
- Disconnect from the broker (*`disconnect()`* function)
- Callbacks used to process events

Simple Subscriber

```
import paho.mqtt.client as mqtt
def on_message(client, userdata, msg):
    print(f"Received `{msg.payload.decode()}`
                from `{msg.topic}` topic")
client = mqtt.Client('demo_unimib_sub')
client.connect('test.mosquitto.org')
client.on_message = on_message
client.subscribe('ax4sg-ggss/#')
client.loop_forever()
```

Client name

URL of the broker

Subscribed resources

Summary of the example

- In the example, the provided Paho MQTT is used to create an **MQTT client** in Python.
- The client connects to an **MQTT broker**.
  - Subscribing to a specific topic.
- The client sets a callback function to process incoming messages.
- Finally, the client enters a loop to continuously receive and process arriving messages.

Simple Publisher

```python
import paho.mqtt.client as mqtt
client = mqtt.Client('demo_unimib_pub')
client.connect('test.mosquitto.org')
client.publish('ax4sg-ggss/temperature', 21.5)
```

Wildcards (for topic subscription)

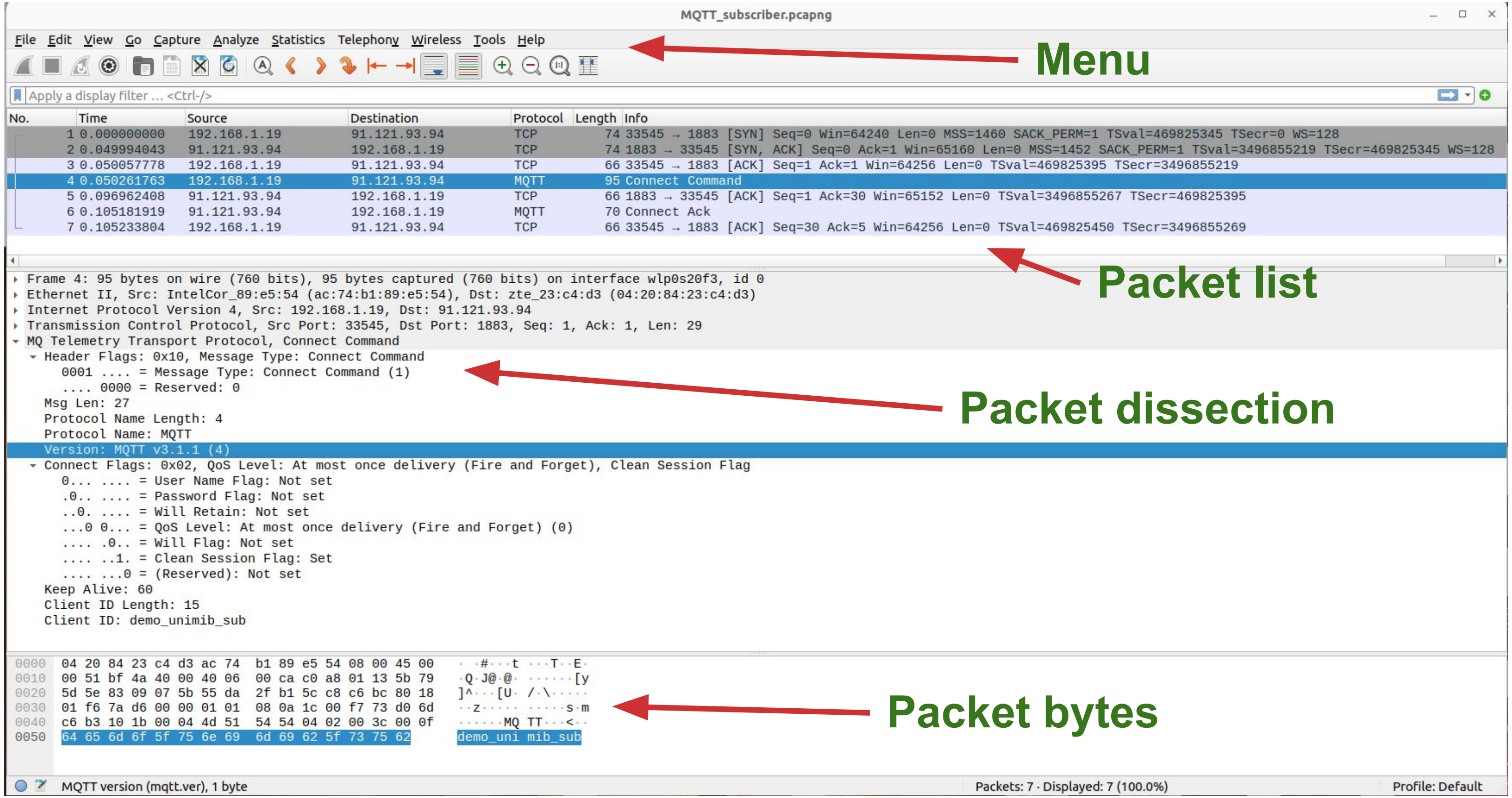
- + (Plus):
  - It represents a **single level** in a topic
  - Example: "`sensor/+/temperature`" corresponds to "`sensor/room1/temperature`" and "`sensor/kitchen/temperature`", but not to "`sensor/bedroom/humidity`".
  - Allows subscribing to groups of topics that share the same structure but differ in a specific level.
- # (Hash)
  - Represents **zero or more levels** in a topic (can only be used at the end).
  - Example: "`sensor/#`" corresponds to "`sensor/temperature`", "`sensor/humidity`", but also "`sensor/kitchen/temperature`".
  - Allows you to subscribe to all topics starting with a certain prefix.
  - It must be the last character in the topic, and may only be used once in a subscription.

Wireshark

- **Wireshark** is a network packet analyzer
  - **Captures packets** and provides many details
  - **Analyzes** what happen in a network cable or in a wireless communication
- Resources:
  - https://www.wireshark.org/
  - https://wiki.wireshark.org/

- Why Wireshark? From wireshark.org…
  - **Network administrators** use it to troubleshoot network problems
  - **Network security engineers** use it to examine security problems
  - **Quality assurance (QA)** engineers use it to verify network applications
  - **Developers** use it to debug protocol implementations
  - People use it **to learn** network protocol internals

Wireshark interface


MQTT_subscriber.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time         Source         Destination    Protocol Length Info
   1 0.000000000  192.168.1.19   91.121.93.94   TCP       74 33545 → 1883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=469825345 TSecr=0 WS=128
   2 0.049994043  91.121.93.94   192.168.1.19   TCP       74 1883 → 33545 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1452 SACK_PERM=1 TSval=3496855219 TSecr=469825345 WS=128
   3 0.050057778  192.168.1.19   91.121.93.94   TCP       66 33545 → 1883 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=469825395 TSecr=3496855219
   4 0.050261763  192.168.1.19   91.121.93.94   MQTT      95 Connect Command
   5 0.096962408  91.121.93.94   192.168.1.19   TCP       66 1883 → 33545 [ACK] Seq=1 Ack=30 Win=65152 Len=0 TSval=3496855267 TSecr=469825395
   6 0.105181919  91.121.93.94   192.168.1.19   MQTT      70 Connect Ack
   7 0.105233804  192.168.1.19   91.121.93.94   TCP       66 33545 → 1883 [ACK] Seq=30 Ack=5 Win=64256 Len=0 TSval=469825450 TSecr=3496855269
```

```
› Frame 4: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface wlp0s20f3, id 0
› Ethernet II, Src: IntelCor_89:e5:54 (ac:74:b1:89:e5:54), Dst: zte_23:c4:d3 (04:20:84:23:c4:d3)
› Internet Protocol Version 4, Src: 192.168.1.19, Dst: 91.121.93.94
› Transmission Control Protocol, Src Port: 33545, Dst Port: 1883, Seq: 1, Ack: 1, Len: 29
▾ MQ Telemetry Transport Protocol, Connect Command
  ▾ Header Flags: 0x10, Message Type: Connect Command
      0001 .... = Message Type: Connect Command (1)
      .... 0000 = Reserved: 0
    Msg Len: 27
    Protocol Name Length: 4
    Protocol Name: MQTT
    Version: MQTT v3.1.1 (4)
  ▾ Connect Flags: 0x02, QoS Level: At most once delivery (Fire and Forget), Clean Session Flag
      0... .... = User Name Flag: Not set
      .0.. .... = Password Flag: Not set
      ..0. .... = Will Retain: Not set
      ...0 0... = QoS Level: At most once delivery (Fire and Forget) (0)
      .... .0.. = Will Flag: Not set
      .... ..1. = Clean Session Flag: Set
      .... ...0 = (Reserved): Not set
    Keep Alive: 60
    Client ID Length: 15
    Client ID: demo_unimib_sub
```

```
0000  04 20 84 23 c4 d3 ac 74  b1 89 e5 54 08 00 45 00   · ·#···t ···T··E·
0010  00 51 bf 4a 40 00 40 06  00 ca c0 a8 01 13 5b 79   ·Q·J@·@· ······[y
0020  5d 5e 83 09 07 5b 55 da  2f b1 5c c8 c6 bc 80 18   ]^···[U· /·\·····
0030  01 f6 7a d6 00 00 01 01  08 0a 1c 00 f7 73 d0 6d   ··z····· ·····s·m
0040  c6 b3 10 1b 00 04 4d 51  54 54 04 02 00 3c 00 0f   ······MQ TT···<··
0050  64 65 6d 6f 5f 75 6e 69  6d 69 62 5f 73 75 62      demo_uni mib_sub
```

MQTT version (mqtt.ver), 1 byte | Packets: 7 · Displayed: 7 (100.0%) | Profile: Default

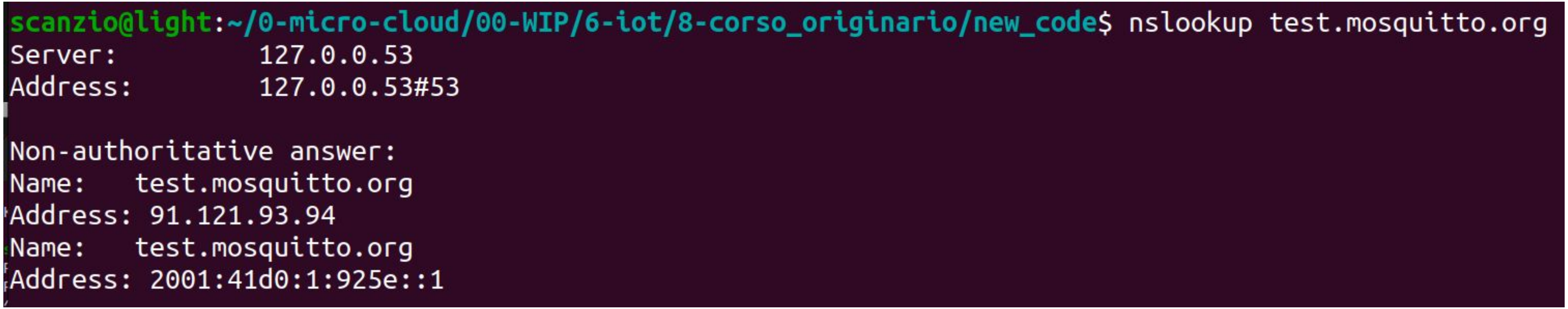
Menu

Packet list

Packet dissection

Packet bytes

Wireshark meets MQTT - Subscriber

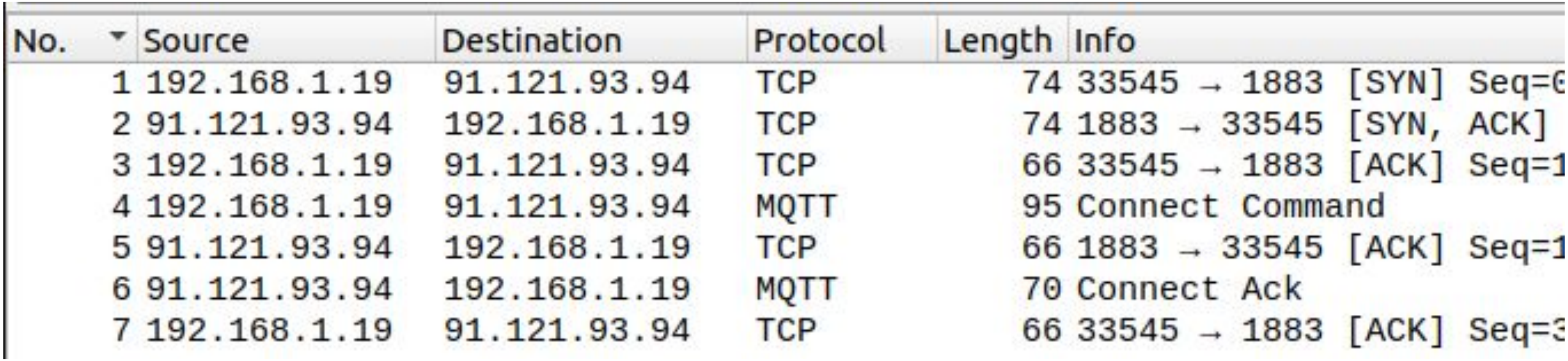


```
scanzio@light:~/0-micro-cloud/00-WIP/6-iot/8-corso_originario/new_code$ nslookup test.mosquitto.org
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   test.mosquitto.org
Address: 91.121.93.94
Name:   test.mosquitto.org
Address: 2001:41d0:1:925e::1
```

Execute the subscriber:

```
python3 00200-mqtt_basic_sub.py
```

Wireshark meets MQTT - Subscriber

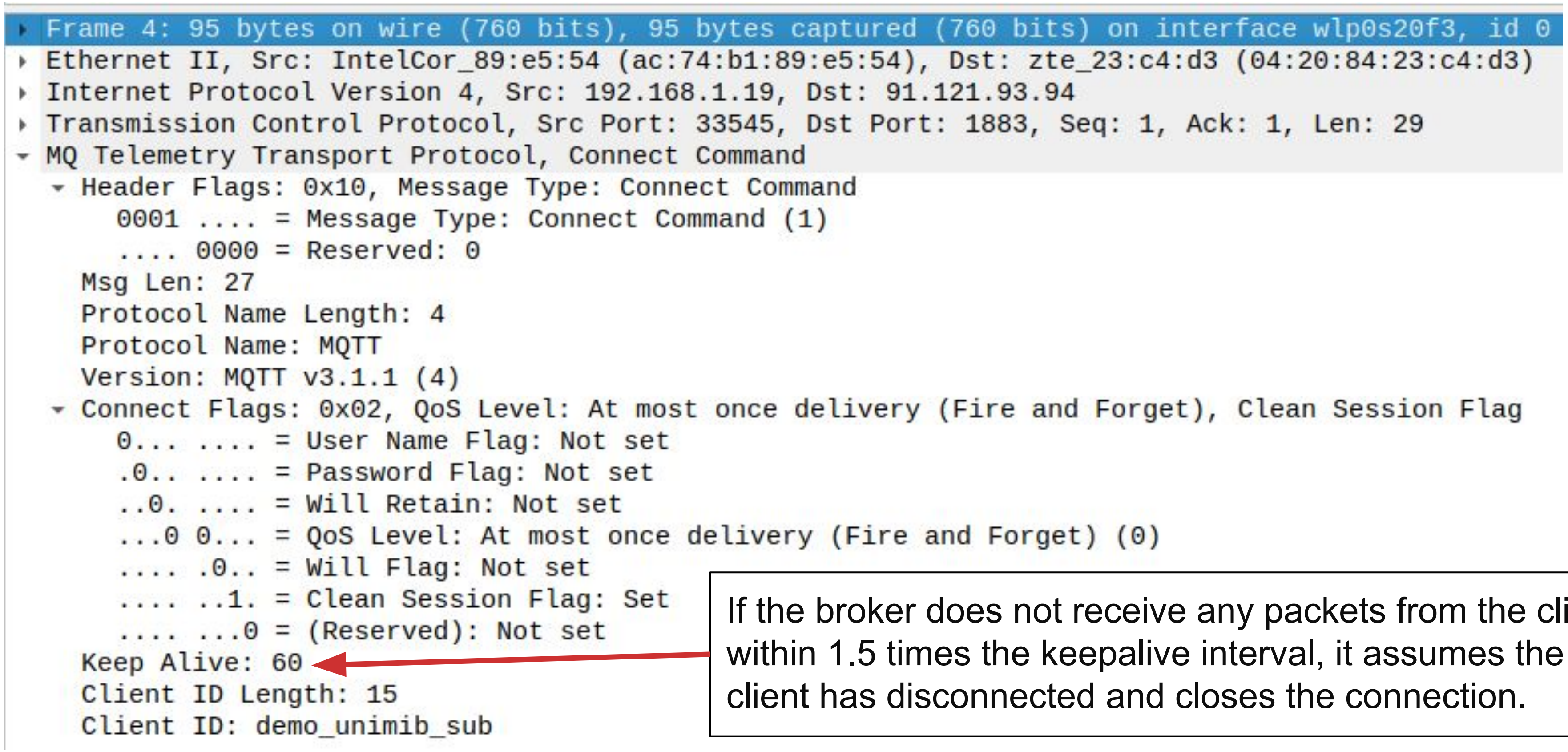


No.	Source	Destination	Protocol	Length	Info
1	192.168.1.19	91.121.93.94	TCP	74	33545 → 1883 [SYN] Seq=0
2	91.121.93.94	192.168.1.19	TCP	74	1883 → 33545 [SYN, ACK]
3	192.168.1.19	91.121.93.94	TCP	66	33545 → 1883 [ACK] Seq=1
4	192.168.1.19	91.121.93.94	MQTT	95	Connect Command
5	91.121.93.94	192.168.1.19	TCP	66	1883 → 33545 [ACK] Seq=1
6	91.121.93.94	192.168.1.19	MQTT	70	Connect Ack
7	192.168.1.19	91.121.93.94	TCP	66	33545 → 1883 [ACK] Seq=3

Execute the Subscriber:

```
python3 00200-mqtt_basic_sub.py
```

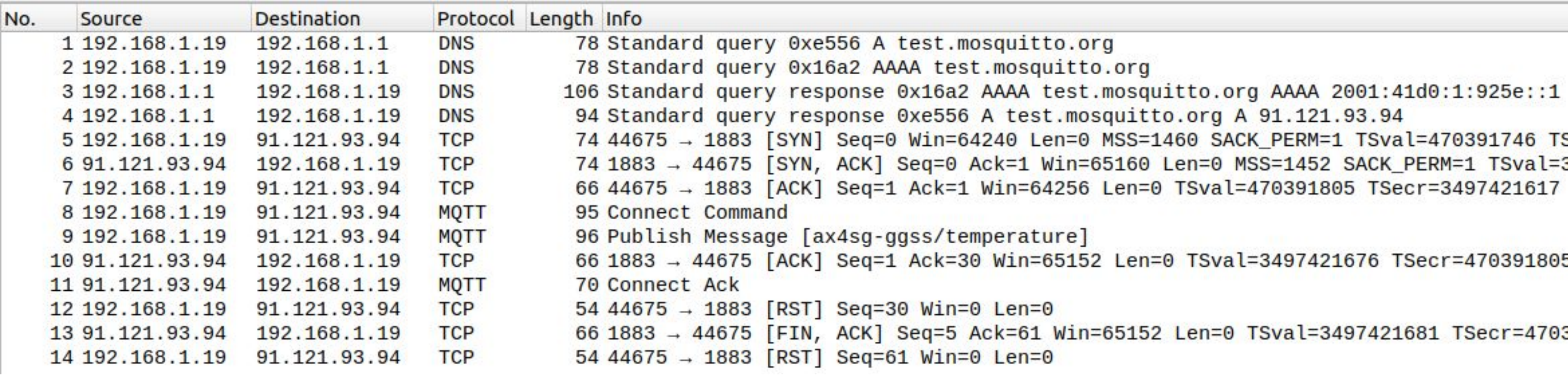
Wireshark meets MQTT - Connect command



```
▸ Frame 4: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: IntelCor_89:e5:54 (ac:74:b1:89:e5:54), Dst: zte_23:c4:d3 (04:20:84:23:c4:d3)
▸ Internet Protocol Version 4, Src: 192.168.1.19, Dst: 91.121.93.94
▸ Transmission Control Protocol, Src Port: 33545, Dst Port: 1883, Seq: 1, Ack: 1, Len: 29
▾ MQ Telemetry Transport Protocol, Connect Command
  ▾ Header Flags: 0x10, Message Type: Connect Command
      0001 .... = Message Type: Connect Command (1)
      .... 0000 = Reserved: 0
    Msg Len: 27
    Protocol Name Length: 4
    Protocol Name: MQTT
    Version: MQTT v3.1.1 (4)
  ▾ Connect Flags: 0x02, QoS Level: At most once delivery (Fire and Forget), Clean Session Flag
      0... .... = User Name Flag: Not set
      .0.. .... = Password Flag: Not set
      ..0. .... = Will Retain: Not set
      ...0 0... = QoS Level: At most once delivery (Fire and Forget) (0)
      .... .0.. = Will Flag: Not set
      .... ..1. = Clean Session Flag: Set
      .... ...0 = (Reserved): Not set
    Keep Alive: 60
    Client ID Length: 15
    Client ID: demo_unimib_sub
```

If the broker does not receive any packets from the client within 1.5 times the keepalive interval, it assumes the client has disconnected and closes the connection.

Wireshark meets MQTT - Publisher

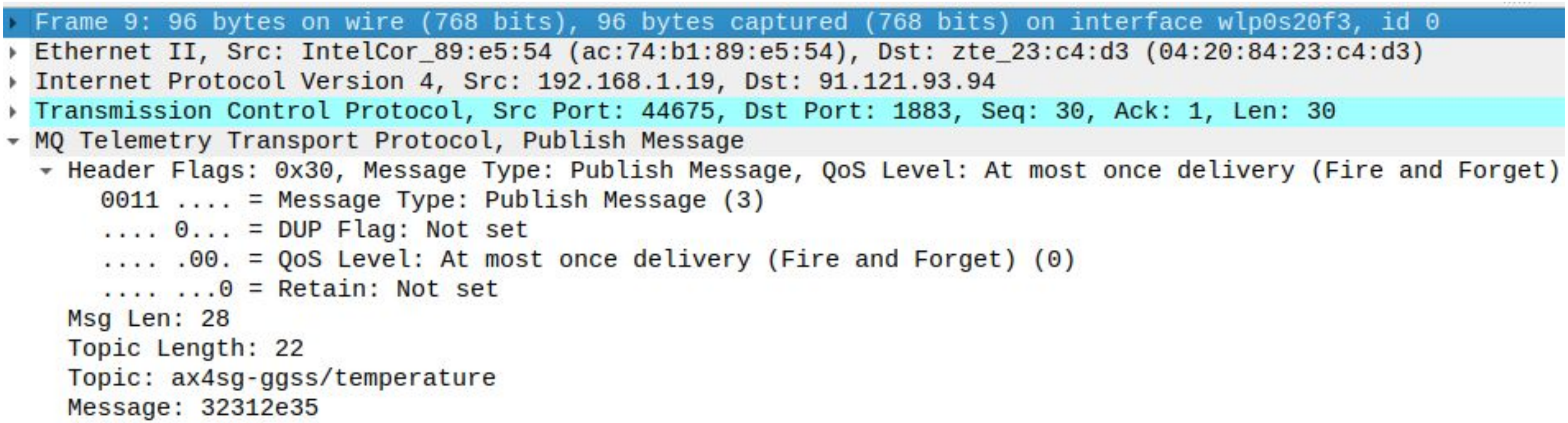


No.	Source	Destination	Protocol	Length	Info
1	192.168.1.19	192.168.1.1	DNS	78	Standard query 0xe556 A test.mosquitto.org
2	192.168.1.19	192.168.1.1	DNS	78	Standard query 0x16a2 AAAA test.mosquitto.org
3	192.168.1.1	192.168.1.19	DNS	106	Standard query response 0x16a2 AAAA test.mosquitto.org AAAA 2001:41d0:1:925e::1
4	192.168.1.1	192.168.1.19	DNS	94	Standard query response 0xe556 A test.mosquitto.org A 91.121.93.94
5	192.168.1.19	91.121.93.94	TCP	74	44675 → 1883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=470391746 TS
6	91.121.93.94	192.168.1.19	TCP	74	1883 → 44675 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1452 SACK_PERM=1 TSval=3
7	192.168.1.19	91.121.93.94	TCP	66	44675 → 1883 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=470391805 TSecr=3497421617
8	192.168.1.19	91.121.93.94	MQTT	95	Connect Command
9	192.168.1.19	91.121.93.94	MQTT	96	Publish Message [ax4sg-ggss/temperature]
10	91.121.93.94	192.168.1.19	TCP	66	1883 → 44675 [ACK] Seq=1 Ack=30 Win=65152 Len=0 TSval=3497421676 TSecr=470391805
11	91.121.93.94	192.168.1.19	MQTT	70	Connect Ack
12	192.168.1.19	91.121.93.94	TCP	54	44675 → 1883 [RST] Seq=30 Win=0 Len=0
13	91.121.93.94	192.168.1.19	TCP	66	1883 → 44675 [FIN, ACK] Seq=5 Ack=61 Win=65152 Len=0 TSval=3497421681 TSecr=4703
14	192.168.1.19	91.121.93.94	TCP	54	44675 → 1883 [RST] Seq=61 Win=0 Len=0

Execute the Publisher:

```
python3 00100-mqtt_basic_pub.py
```

Wireshark meets MQTT - Publish Message



```
▸ Frame 9: 96 bytes on wire (768 bits), 96 bytes captured (768 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: IntelCor_89:e5:54 (ac:74:b1:89:e5:54), Dst: zte_23:c4:d3 (04:20:84:23:c4:d3)
▸ Internet Protocol Version 4, Src: 192.168.1.19, Dst: 91.121.93.94
▸ Transmission Control Protocol, Src Port: 44675, Dst Port: 1883, Seq: 30, Ack: 1, Len: 30
▾ MQ Telemetry Transport Protocol, Publish Message
  ▾ Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
      0011 .... = Message Type: Publish Message (3)
      .... 0... = DUP Flag: Not set
      .... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
      .... ...0 = Retain: Not set
    Msg Len: 28
    Topic Length: 22
    Topic: ax4sg-ggss/temperature
    Message: 32312e35
```

Message ASCII (character):

32 (‘2’) 31 (‘1’) 2e (‘.’) 35 (‘5’) →”21.5”

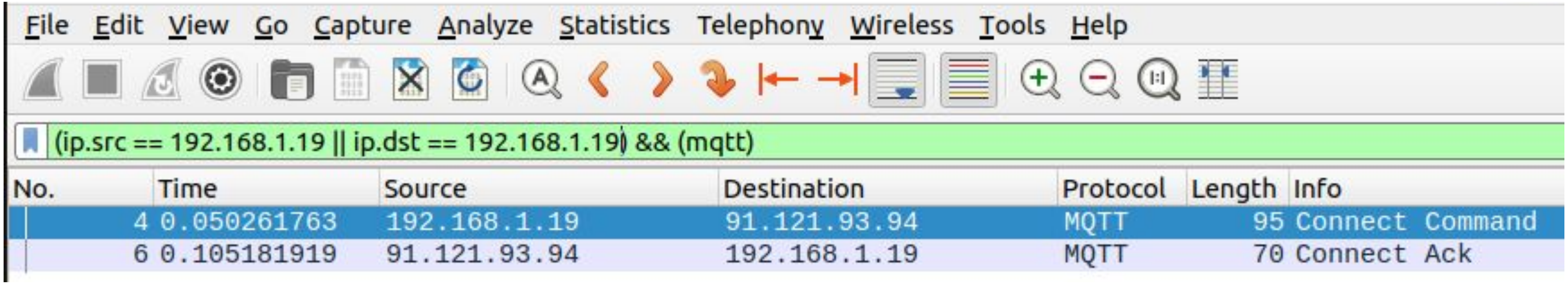
.pcap files

- Data captured by Wireshark can be stored in a **.pcap** (packet capture) file
  - Originally designed for tcpdump/libpcap
  - Widely used format
  - Contains raw network traffic

- You can download the first example here:
  - https://www.skenz.it/listing/iot/wireshark/MQTT_subscriber.pcap
- Other general examples of captures:
  - https://wiki.wireshark.org/uploads/27707187aeb30df68e70c8fb9d614981/http.cap
  - https://wiki.wireshark.org/SampleCaptures

Filters

- **Filters** permit the selection of specific packets
  - That meet certain search requirements
  - Possible questions about filters in the exam!
- Example applied to https://www.skenz.it/listing/iot/wireshark/MQTT_subscriber.pcap

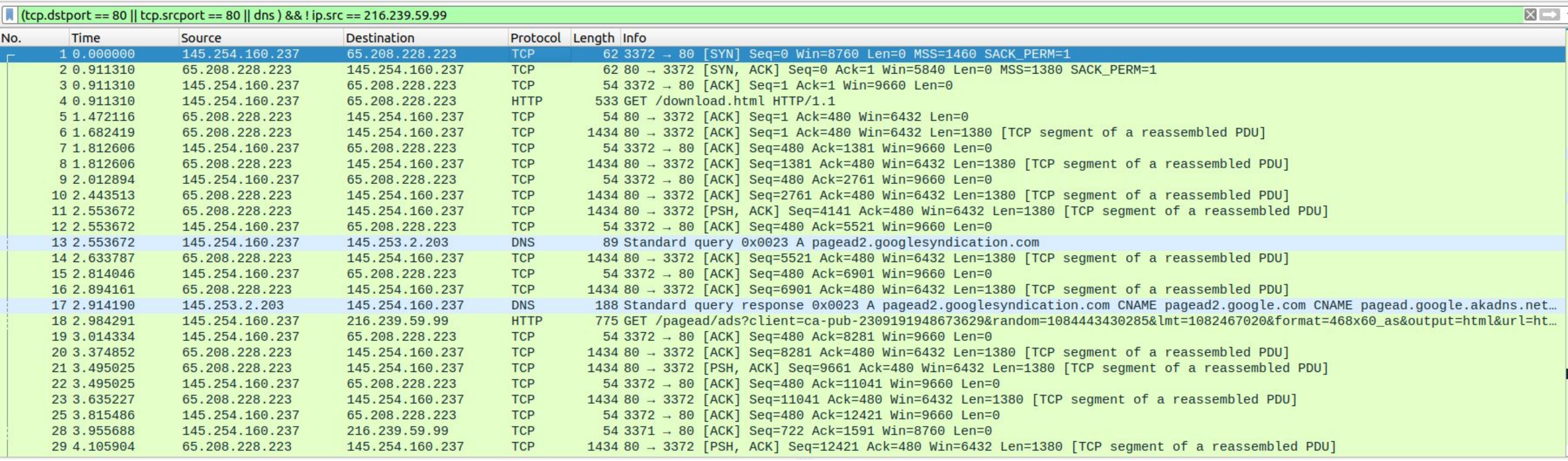


```
(ip.src == 192.168.1.19 || ip.dst == 192.168.1.19) && (mqtt)
```

No.	Time	Source	Destination	Protocol	Length	Info
4	0.050261763	192.168.1.19	91.121.93.94	MQTT	95	Connect Command
6	0.105181919	91.121.93.94	192.168.1.19	MQTT	70	Connect Ack

Filters (2)

- Example applied to https://wiki.wireshark.org/uploads/27707187aeb30df68e70c8fb9d614981/http.cap



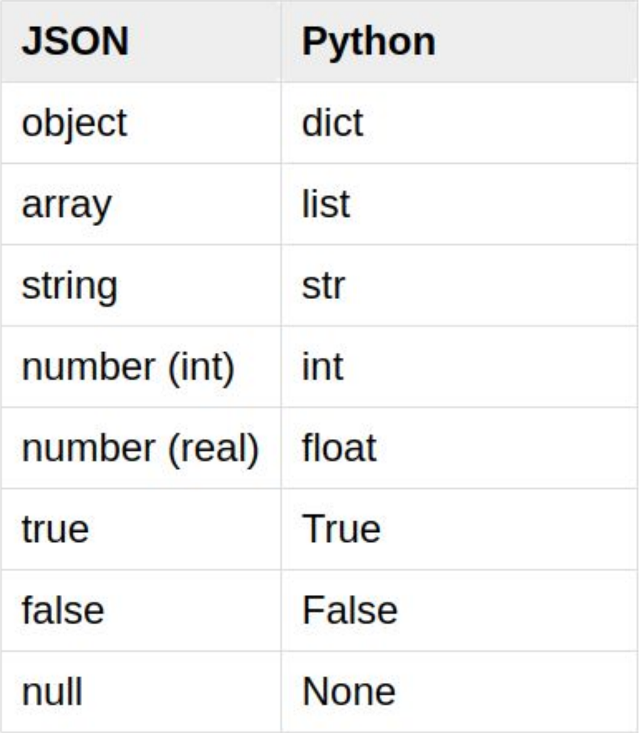
```
(tcp.dstport == 80 || tcp.srcport == 80 || dns ) && ! ip.src == 216.239.59.99
```

```
No.  Time      Source           Destination      Protocol Length Info
   1 0.000000  145.254.160.237  65.208.228.223   TCP        62 3372 → 80 [SYN] Seq=0 Win=8760 Len=0 MSS=1460 SACK_PERM=1
   2 0.911310  65.208.228.223   145.254.160.237  TCP        62 80 → 3372 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1380 SACK_PERM=1
   3 0.911310  145.254.160.237  65.208.228.223   TCP        54 3372 → 80 [ACK] Seq=1 Ack=1 Win=9660 Len=0
   4 0.911310  145.254.160.237  65.208.228.223   HTTP      533 GET /download.html HTTP/1.1
   5 1.472116  65.208.228.223   145.254.160.237  TCP        54 80 → 3372 [ACK] Seq=1 Ack=480 Win=6432 Len=0
   6 1.682419  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [ACK] Seq=1 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
   7 1.812606  145.254.160.237  65.208.228.223   TCP        54 3372 → 80 [ACK] Seq=480 Ack=1381 Win=9660 Len=0
   8 1.812606  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [ACK] Seq=1381 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
   9 2.012894  145.254.160.237  65.208.228.223   TCP        54 3372 → 80 [ACK] Seq=480 Ack=2761 Win=9660 Len=0
  10 2.443513  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [ACK] Seq=2761 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
  11 2.553672  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [PSH, ACK] Seq=4141 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
  12 2.553672  145.254.160.237  65.208.228.223   TCP        54 3372 → 80 [ACK] Seq=480 Ack=5521 Win=9660 Len=0
  13 2.553672  145.254.160.237  145.253.2.203    DNS        89 Standard query 0x0023 A pagead2.googlesyndication.com
  14 2.633787  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [ACK] Seq=5521 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
  15 2.814046  145.254.160.237  65.208.228.223   TCP        54 3372 → 80 [ACK] Seq=480 Ack=6901 Win=9660 Len=0
  16 2.894161  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [ACK] Seq=6901 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
  17 2.914190  145.253.2.203    145.254.160.237  DNS       188 Standard query response 0x0023 A pagead2.googlesyndication.com CNAME pagead2.google.com CNAME pagead.google.akadns.net…
  18 2.984291  145.254.160.237  216.239.59.99    HTTP      775 GET /pagead/ads?client=ca-pub-2309191948673629&random=1084443430285&lmt=1082467020&format=468x60_as&output=html&url=ht…
  19 3.014334  145.254.160.237  65.208.228.223   TCP        54 3372 → 80 [ACK] Seq=480 Ack=8281 Win=9660 Len=0
  20 3.374852  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [ACK] Seq=8281 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
  21 3.495025  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [PSH, ACK] Seq=9661 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
  22 3.495025  145.254.160.237  65.208.228.223   TCP        54 3372 → 80 [ACK] Seq=480 Ack=11041 Win=9660 Len=0
  23 3.635227  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [ACK] Seq=11041 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
  25 3.815486  145.254.160.237  65.208.228.223   TCP        54 3372 → 80 [ACK] Seq=480 Ack=12421 Win=9660 Len=0
  28 3.955688  145.254.160.237  216.239.59.99    TCP        54 3371 → 80 [ACK] Seq=722 Ack=1591 Win=8760 Len=0
  29 4.105904  65.208.228.223   145.254.160.237  TCP      1434 80 → 3372 [PSH, ACK] Seq=12421 Ack=480 Win=6432 Len=1380 [TCP segment of a reassembled PDU]
```

JSON

- **JSON (JavaScript Object Notation)** is a lightweight data interchange format inspired by JavaScript
  - open standard
  - language independent format, but similar to C-like languages (C, C++, C#, Java, Perl, Python, and JavaScript)
  - human-readable, easy to parse and generate, useful for representing and storing structured data (and “serialize” them).
    - Lighter and faster than XML
  - specified by RFC 7159 (which obsoletes RFC 4627) and by ECMA-404
  - built-in package called json (`import json`)
  - https://json.org/
- Guide
  - https://docs.python.org/3/library/json.html

Type conversion



JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

json package

- `json.loads()`: From JSON string to python dictionary

```
import json
a = '{ "exams": ["IoT", "OS"], "students": 6000}'
b = json.loads(a)
print(b, type(b))
> {'exams': ['IoT', 'OS'], 'students': 6000} <class ‘dict’>
```

- `json.dumps()`: From a python object (e.g., dictionary, list, tuple) to a JSON string

```
import json
c = {"name":"Stefano", "age":44}
d = json.dumps(c)
print(type(c), type(d), d)
> <class 'dict'> <class 'str'> {"name": "Stefano", "age": 25}
```

From JSON to XML

- After installing `xmltodict` using `pip3 install xmltodict`

JSON must have only one root

```
import json
import xmltodict
a = '{"root": { "exams": ["IoT", "OS"], "students": 6000}}'
b = json.loads(a)
xml=xmltodict.unparse(b)
print(xml)
```

```
> <?xml version="1.0" encoding="utf-8"?>
<root><exams>IoT</exams><exams>OS</exams><students>6000</students></root>
```

From XML to JSON

```
import json
import xmltodict
c = xmltodict.parse(
"<root><exams>IoT</exams><exams>OS</exams><students>6000</students></root>")
d = json.dumps(c)
print(type(c), type(d), d)
```

```
> <class 'dict'> <class 'str'> {"root": {"exams": ["IoT", "OS"], "students":
"6000"}}
```

tkinter

- **Tkinter** is a module to create graphical interfaces
  - Easy to use
  - Cross platform: Windows, Linux, macOS
- Guide
  - https://docs.python.org/3/library/tkinter.html
- Installation

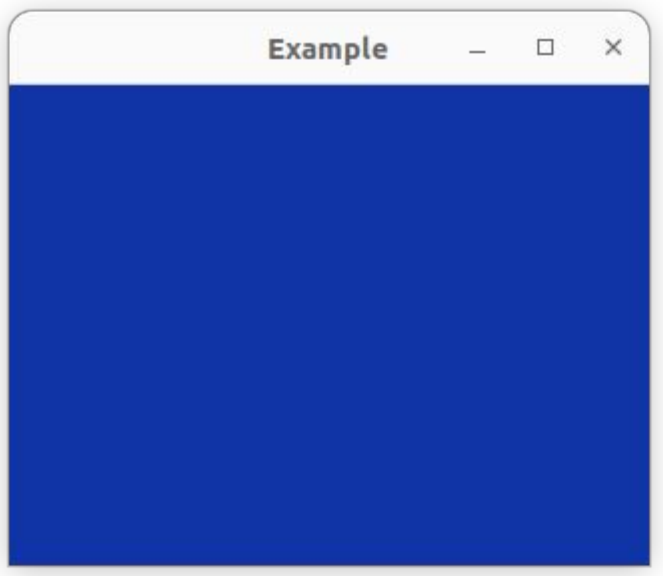
```
pip install tk
```

tkinter

- **Widget**: Graphical User Interface (GUI) elements
  - buttons
  - textboxes
  - labels
  - images
- **Windows**: container of widgets

Simple window

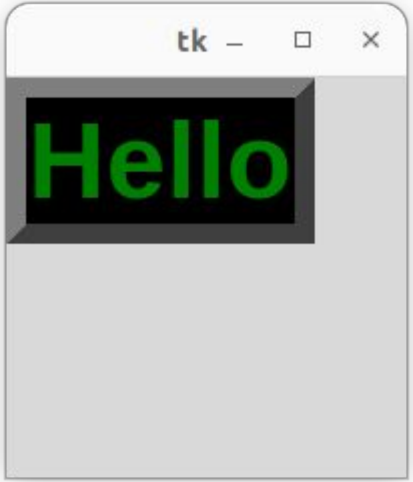
```
import tkinter

win = tkinter.Tk() # Instance of a window
win.geometry("320x240")
win.title("Example")
#win.config(background="red")
win.config(background="#1034A6") #Egyptian blue

win.mainloop() # Generate window, and listen for events
```



Label widget

- **Label widget to place text or images**

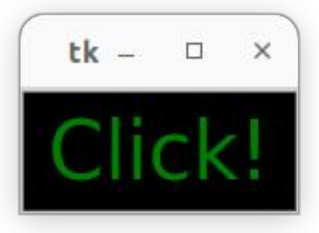
```
from tkinter import *

win = Tk()
label = Label(win,text="Hello",
              font=("Arial",40,"bold"),
              fg="green",
              bg="black",
              relief=RAISED,
              bd=10)
#label.pack() # Center of row
label.place(x=0,y=0)

win.mainloop()
```



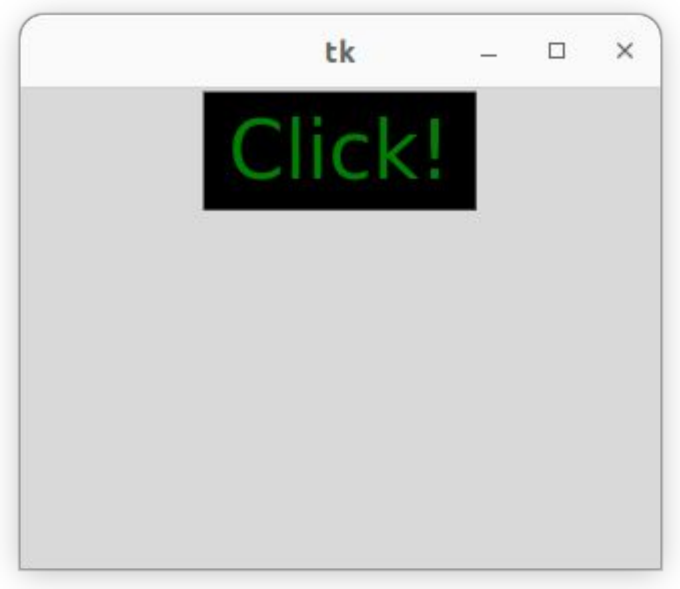
Button widget

```
from tkinter import *

count = 0
def on_click():
    global count; count += 1
    print(count)
    win.geometry("320x240")

win = Tk()
button = Button(win,
                text="Click!",
                command=on_click, font=("Comic Sans",30),
                fg="green", activeforeground="green", bg="black", activebackground="black")
button.pack()

win.mainloop()
```

Before Click!



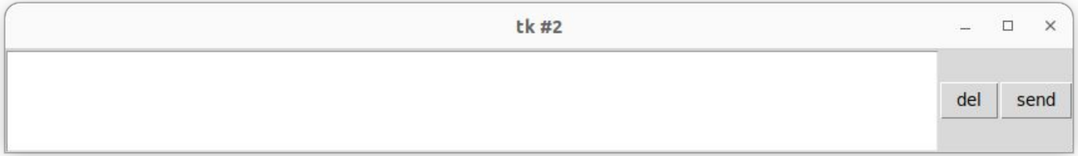
After Click!



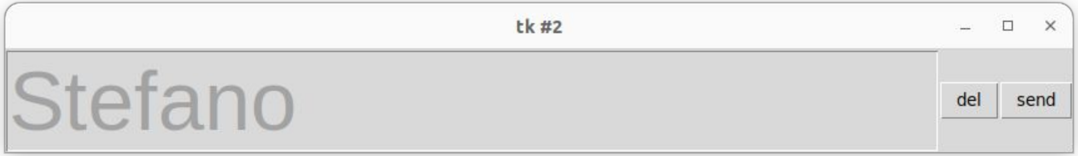
Entry widget

```
from tkinter import *

def send():
    str = txt.get(); print("Hello "+str)
    txt.config(state=DISABLED)
def delete():
    txt.delete(0,END)

win = Tk()
txt = Entry(win,font=("Arial",50)); txt.pack(side=LEFT)

sub_button = Button(win,text="send",command=send); sub_button.pack(side=RIGHT)
del_button = Button(win,text="del",command=delete); del_button.pack(side=RIGHT)

win.mainloop()
```

Before send

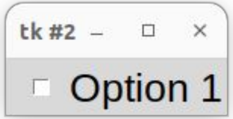


After send

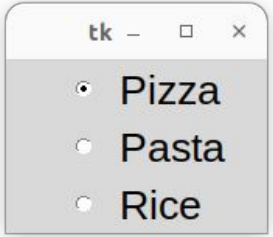


Other widgets

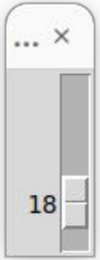
- **Check button**: `tkinter.Checkbutton()`



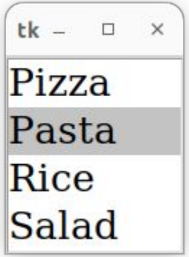
- **Radio button**: `tkinter.Radiobutton()`



- **Sliding scale**: `tkinter.Scale()`

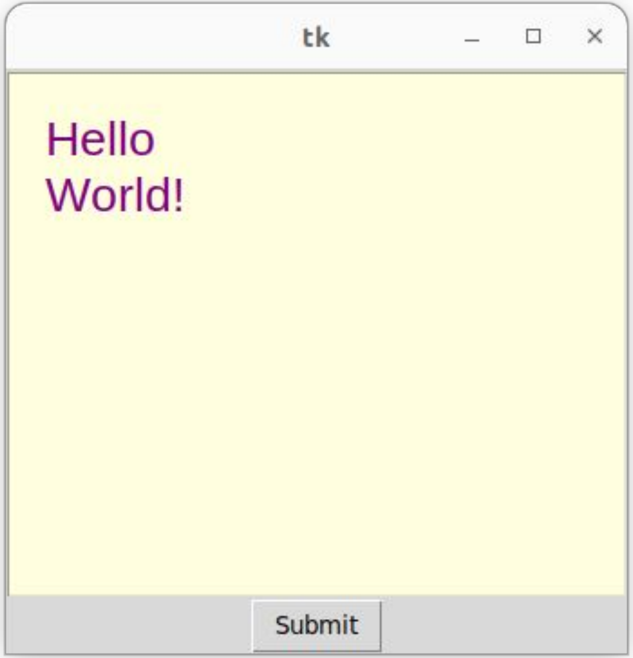


- **List box**: `tkinter.Listbox()`



Text widget

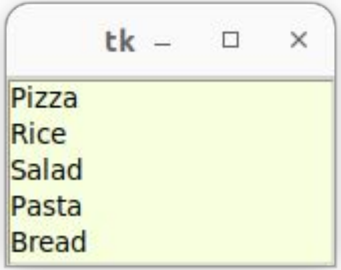
```
from tkinter import *
def submit():
   print(text.get(1.0,END)) # From the beginning 1.0 to the end

win = Tk()
text = Text(win,bg="light yellow",fg="purple",
           font=("Arial",20),
           height=8,width=20,
           padx=20,pady=20)
text.pack()

button=Button(win,text="Submit",command=submit)
button.pack()

win.mainloop()
```



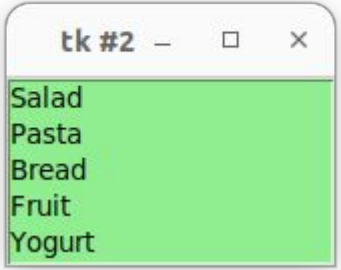
Hello
World!

Listbox widget

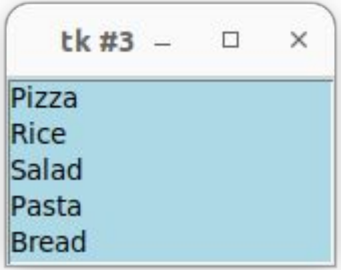
```
from tkinter import *
win = Tk()
listbox = Listbox(win, bg="#f7ffde", width=20, height=5)
listbox.pack()

listbox.insert(1,"Pizza"); listbox.insert(2,"Rice")
listbox.insert(3,"Salad"); listbox.insert(4,"Pasta")
listbox.insert(5,"Bread"); listbox.insert(6,"Fruit")
listbox.insert(END,"Yogurt")

listbox.yview_moveto(1.0)
listbox.config(bg="light green")

listbox.yview_moveto(0.0)
listbox.config(bg="light blue")

win.mainloop()
```



Pizza
Rice
Salad
Pasta
Bread



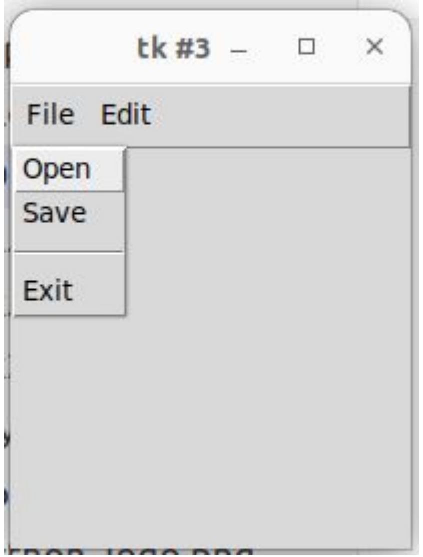
Salad
Pasta
Bread
Fruit
Yogurt



Pizza
Rice
Salad
Pasta
Bread

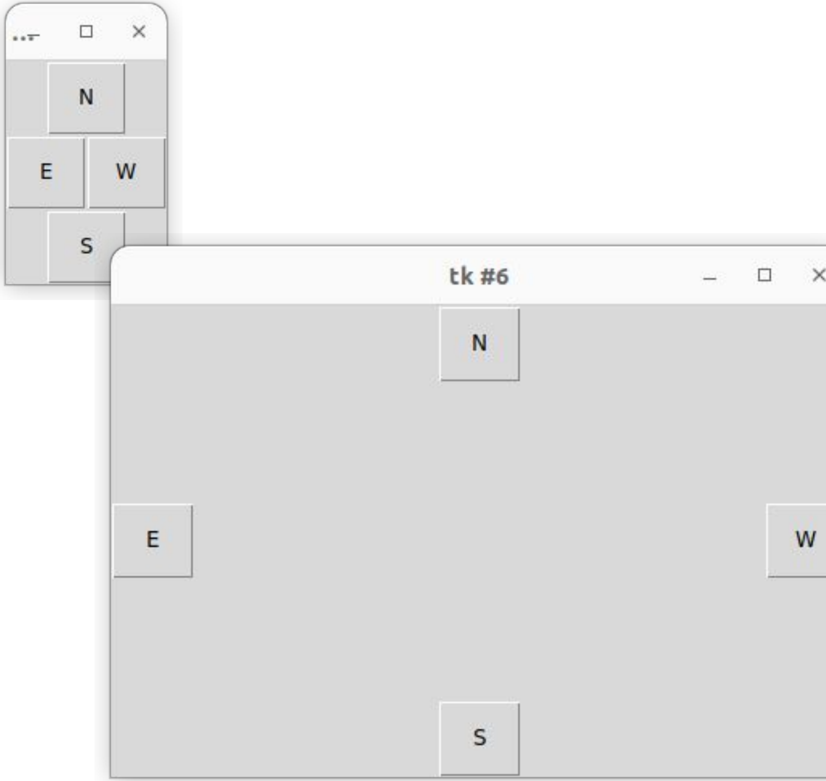
Menu bar

```
from tkinter import *

def open_file(): print("Open file code")

win = Tk()
menubar = Menu(win)
win.config(menu=menubar)

file_menu = Menu(menubar,tearoff=0)
menubar.add_cascade(label="File",menu=file_menu)
file_menu.add_command(label="Open",command=open_file)
file_menu.add_command(label="Save")
file_menu.add_separator()
file_menu.add_command(label="Exit")

edit_menu = Menu(menubar,tearoff=0)
menubar.add_cascade(label="Edit",menu=edit_menu)
win.mainloop()
```



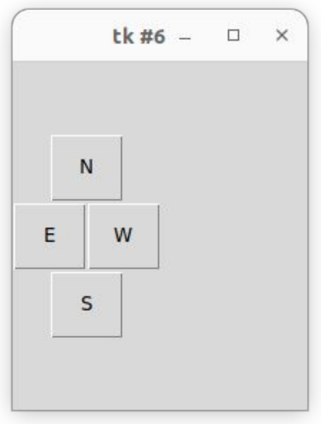
Why Frame?

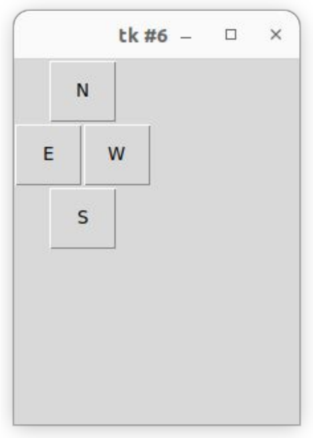
```
from tkinter import *

win = Tk()

buttonN=Button(win,text="N",width=3,height=2)
buttonS=Button(win,text="S",width=3,height=2)
buttonE=Button(win,text="E",width=3,height=2)
buttonW=Button(win,text="W",width=3,height=2)
buttonN.pack(side=TOP);
buttonS.pack(side=BOTTOM);
buttonE.pack(side=LEFT);
buttonW.pack(side=RIGHT);

win.mainloop()
```

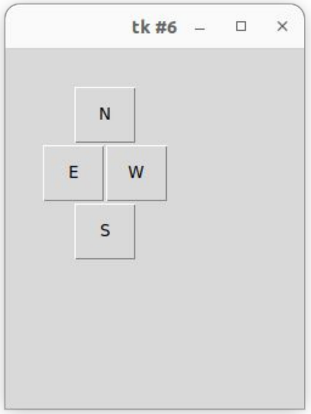


Frame

```
from tkinter import *

win = Tk()
frame=Frame(win)
frame.pack(side=LEFT)
#frame.pack(anchor=NW)
#frame.place(x=30,y=30)

buttonN=Button(frame,text="N",width=3,height=2)
buttonS=Button(frame,text="S",width=3,height=2)
buttonE=Button(frame,text="E",width=3,height=2)
buttonW=Button(frame,text="W",width=3,height=2)
buttonN.pack(side=TOP);
buttonS.pack(side=BOTTOM);
buttonE.pack(side=LEFT);
buttonW.pack(side=RIGHT);

win.mainloop()
```

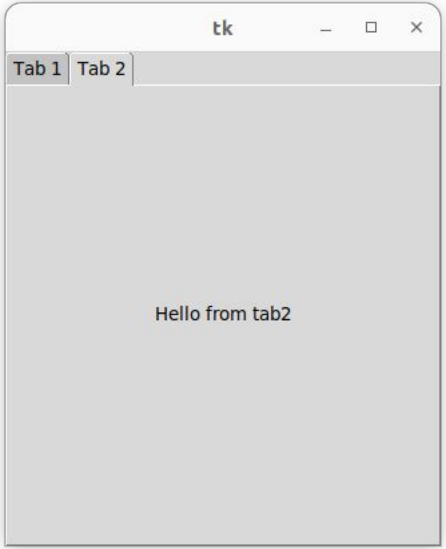






Tabs (Notebook widget)

```
from tkinter import *
from tkinter import ttk

win = Tk()
notebook = ttk.Notebook(win) # Widget to manage a collection of widget and displays
tab1 = Frame(notebook) # New Frame for tab1
tab2 = Frame(notebook) # New Frame for tab2
notebook.add(tab1,text="Tab 1")
notebook.add(tab2,text="Tab 2")
notebook.pack(expand=True,fill="both") # Fill all space

Label(tab1,text="Hello from tab1",width=40,heigh=20).pack()
Label(tab2,text="Hello from tab2",width=40,heigh=20).pack()

win.mainloop()
```



Hello from tab2

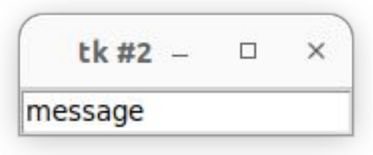
Window .protocol() method

```
from tkinter import *

win = Tk()

def on_closing():
   print("window closed!")
   win.quit()

win.protocol("WM_DELETE_WINDOW", on_closing)
# WM_TAKE_FOCUS  Other window protocol, triggered by focus

win.mainloop()
```

Widget .bind() method

- It binds a specified event occurred on the widget to a handler

```
from tkinter import *

win = Tk()

def on_message(event=None):
   print(my_msg.get())

my_msg = StringVar()
my_msg.set("message")
entry_field = Entry(win, textvariable=my_msg)
entry_field.bind("<Return>", on_message)
entry_field.pack()

win.mainloop()
```



message

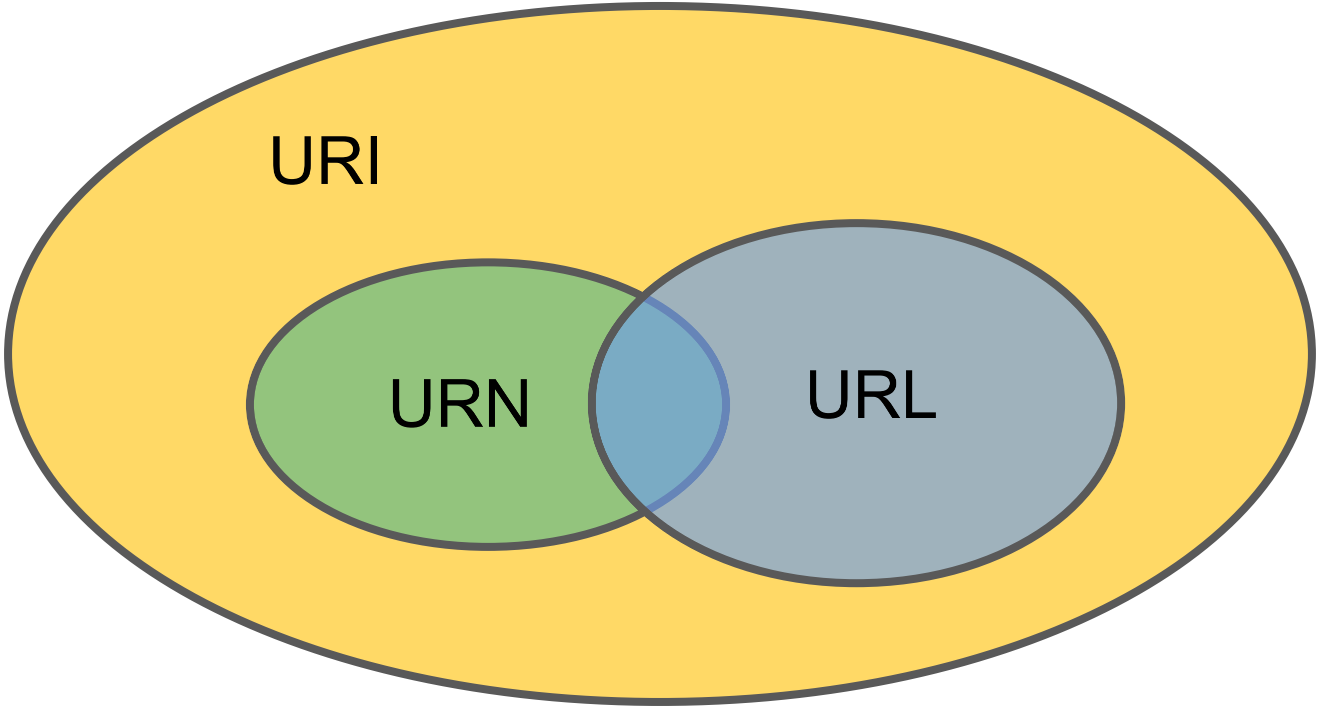
Other events:

- **<Button-1>**: Left mouse button click
- **<Button-3>**: Right mouse button click
- **<Enter>**: Mouse cursor enters the widget
- **<KeyPress>**: Any key is pressed
- **<KeyRelease>**: Any key is released
- **<Tab>**: Tab key is pressed
- **<FocusIn>**: Widget gains focus
- **<FocusOut>**: Widget loses focus
- **<Configure>**: Widget is resized or moved

URI

- **Universal Resource Identifiers** (URIs) are a syntax used to define the **names** and **position** of objects (resources) on the Internet (but not only).
    - formalization started in 1994
    - T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax, 2005. RFC3986. WWW: https://datatracker.ietf.org/doc/html/rfc3986
    - Define a mechanism and syntax for unified access to data resources (encoded as strings).
- WWW
    - Use URI to identify resources reachable through different protocols (e.g., HTTP, FTP, Telnet, etc.)
    - Use of URI in different context, to identify:
        - an image
        - an HTML page
        - a hyperlink
        - an Excel document
        - etc.

URI, URL, URN



URI, URL, URN

- **Uniform Resource Identifier** (URI): *mailto:myemail@skenz.it*
  - *mailto:myemail@skenz.it* (specifies a mail address) or *tel:+393001234567* (specify a phone number)
  - but also *https://www.skenz.it/ss*
- **Uniform Resource Locator** (URL): *https://www.skenz.it/ss*
  - It is a type of URI that specifies the **access method** and the **location**
  - For example: *https://www.skenz.it/ss/theses* tells that the resource is a webpage that can be accessed with the **https protocol** in the web server with **address skenz.it**, and it can be identified by **/ss/theses**
- **Uniform Resource Name** (URN): *urn:isbn:978-3659204821*
  - Designed to uniquely identify a resource based on its name rather than its physical location
  - It is a URI that **persistently identifies the resource**, but it does **not tell where or how to find it**
  - For example: *urn:isbn:978-3659204821* identifies that the resource is a book with a specific ISBN number

URL and URN

In other words:

- **URL** is an address that can be used immediately to access the resource
  - URLs contain **all the information** needed to access the information but are **not robust to changes in the access mechanism** (e.g., changing a directory)
- **URN** is a stable and definitive name of a resource but give no information on how to access it
  - *urn:isbn:978-3659204821* (Reference to the book "Speeding-up Artificial Neural Networks")
  - *urn:ietf:rfc:3986* (Reference to the IETF's RFC 3986)
  - *urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66* (Reference to UUID, version 1)
    - Universally Unique Identifier (UUID) is a label coded in 128 bits
    - Version 1: concatenates a MAC address (48 bits) and a timestamp (60 bits)
  - *urn:isan:0000-0000-2CEA-0000-1-0000-0000-Y* (Reference to a film)

URI characteristics

URIs are:

- **Extensible**: new schemas can be added (to enable new protocols)
- **Comprehensive**: all existing names are encodable and new protocols can be included
- **Printable**: URIs can be expressed in 7-bit ASCII encoding

Syntax

- High level view

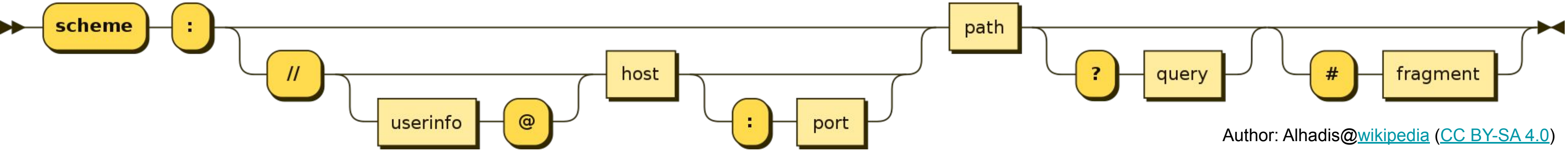
  scheme:specific_part

  - *scheme*
    - protocol (registered string)
    - specifies how to encode the specific_part

- More detailed view

  scheme:[//authority]path[?query][#fragment]

  (authority = [userinfo@]host[:port])



Author: Alhadis@wikipedia (CC BY-SA 4.0)

Syntax (2)

```
scheme:[//authority]path[?query][#fragment]
```

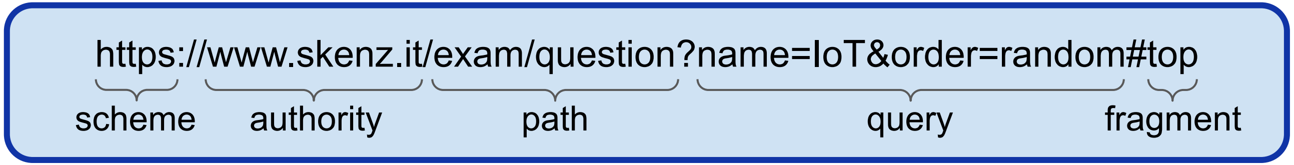
- The **scheme** can restrict the semantic and the syntax of the identifiers
  - case-insensitive (lowercase in practice)
- The **path** permits to define a hierarchy
  - each element separated by “/”
- Examples:

$$\underbrace{\text{prot}}_{\text{scheme}}\text{://}\underbrace{\text{example.net:123}}_{\text{authority}}\text{/}\underbrace{\text{class/animal}}_{\text{path}}\text{?}\underbrace{\text{name=lion}}_{\text{query}}\text{\#}\underbrace{\text{nose}}_{\text{fragment}}$$

$$\underbrace{\text{urn}}_{\text{scheme}}\text{:}\underbrace{\text{example:animal:lion:nose}}_{\text{path}}$$

URL Syntax

- Again, a URL is (a subset of) a URIs

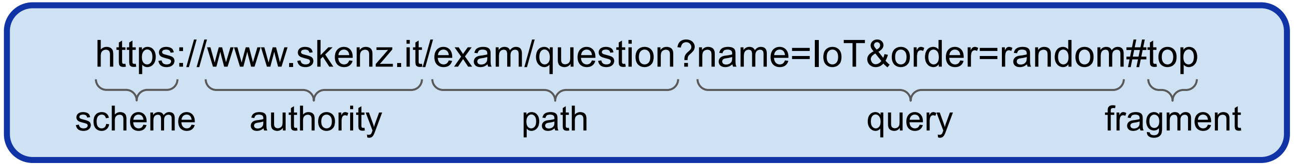


https://www.skenz.it/exam/question?name=IoT&order=random#top

- The **query**
  - Preceded by a “?” (question mark)
  - Assign **values** to an **attributes/keys**
    - “=” (equal) used to **pair an attribute with a value** (*attribute=value*)
    - “&” (ampersand) to **separate pairs** (*attibute1=value1&attribute2=value2*)
  - Used to transfer data to a server (e.g., derived by a webform)
- URL has a limited length
  - e.g., from 2 KB to 8 KB
  - URL too long -> HTTP status code: 414 Request-URI Too Long

URL Syntax (2)

- Again, a URL is (a subset of) a URIs



https://www.skenz.it/exam/question?name=IoT&order=random#top

scheme authority path query fragment

- The **fragment**
  - Refer a **particular element within the resource**
  - For example, `https://www.skenz.it/article#conclusion`
    - Points to the `conclusion` element within the `article`

URI encoding

Characters in URI are unreserved, reserved, and escaped.

- **Unreserved** characters
  - Uppercase, lowercase, and digits (included in US-ASCII)
  - Punctuation: - _ . ! ~ * ‘ ( )
- **Reserved** characters
  - Have specific function in URI: ; / ? : @ & = + $,
  - Escape used to identify these characters
  - Example:
    - \; is the character “;”
    - \\ is the character “\”

URI encoding (2)

- **Escaped** characters
    - All others
        - non US-ASCII, control characters, spaces, other { } | \ ^ [ ] ` < > # % “
    - For ASCII: %XX
        - XX two digits hexadecimal number representing the character
        - “ “ space character $32_{10}=20_{16}$ → %20 (in URI)
    - In UTF-8: %XX(XX)?(XX)?(XX)?
        - UTF-8 has a variable length of 2, 4, 6, or 8 digits hexadecimal numbers
        - € is coded as %E2%82%AC (in URI)

Connection between URI and HTTP/HTTPS

```
http://host[:port]/path[?query][#fragment]
https://host[:port]/path[?query][#fragment]
```

- **host:** IP or DNS address of the resource
- **port:** where the server is listening
  - Default: 80 for HTTP, 443 for HTTPS
- **path:** hierarchic path name to identify the resource
- **query:** the object of research on a specific resource
- **fragment:** identification of a sub-part of the object (the server ignores this part because the return of sub-parts is the responsibility of the client)

Hypertext Transfer Protocol (HTTP)

- **Hypertext Transfer Protocol (HTTP)**
  - **Application layer** protocol
  - For the exchange of documents
- Characteristics
  - **Client-server**
    - **client** activate the connection and request services
    - **server** accept the connection and provides the resource
      - possibly identifies the client
  - **Generic**
    - independent of the format in which the resources are transmitted
  - **Stateless**
    - The server does not need to maintain information that persists between one connection and the next

Versions

- **HTTP/0.9** (1991): Easy client-server protocol only for request HTML resources (**old version**)
- **HTTP/1.0** (1996, RFC 1945): protocol becomes generic and stateless (**old version**)
- **HTTP/1.1** (1997, RFC 2068 and RFC 2616): reuse of a TCP connection to request multiple resources
- **HTTP/2** (2015, RFC 7540): more efficient and push capability to send resources from the server to the client
- **HTTP/3** (2022, RFC 9114): improve of HTTP/2 to support QUIC+UDP

HTTP request

```
Method URI Version CR+LF
[Header]*
CR+LF
Body
```

- **Method**: is the requested action
  - *GET*, *HEAD*, *POST*, *PUT*, *POST*
  - *OPTIONS*, *DELETE*, *TRACE*, *CONNECT* (less common, not described here)
- **URI**: identifies the resource
- **Version**: HTTP1.0 or HTTP/1.1
- **CR+LF**: CR (Carriage Return, ASCII 13) and LF (Line Feed, ASCII 10)
- **Header**: are lines describing the resource
  - **key: value** coules
- Body: the message/resource in the MIME format

Example of an HTTP request

```
GET /beta.html HTTP/1.1
Referer: http://www.alpha.com/alpha.html
Connection: Keep-Alive
User-Agent: Mozilla/4.61 (Macintosh; I; PPC)
Host: www.alpha.com:80
Accept: image/gif, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

Common HTTP request methods

- **GET**
  - used to request a resource from the server
- **HEAD**
  - similar to GET
  - server replies only with headers, but **body is not included**
  - Used to check: validity and accessibility of URI, coherence of the cache
- **DELETE**
  - Used to remove information

Common HTTP request methods (2)

- **PUT**
  - to transmit information from the client to the server (usually to replace existing resources)
  - the argument is a pre-existing resource to which information is added/modified
  - **idempotent**: multiple identical requests should have the same effect as a single request (no risk to create the same resource more than once)

- **POST**
  - like PUT, to transmit information from the client to the server (usually to create new resources)
  - **not idempotent**: sending the same POST request multiple times might result in different outcomes

Headers

- Standardized in RFC 822
- Types
  - general for the transmission, referred to the entity transmitted, to the request made, or to the response generated
- Examples:
  - **Date**: date and hour of the transmission
  - **MIME-Version**: always 1.0
  - **Transfer-Encoding**: The format used in the transmission
  - **Cache-Control**: cache mechanism requested or suggested
  - **Connection**: specifies if connection should be maintained after the current transaction
  - **Content-Type**: the MIME type of the body
  - **Content-Length**: length in bytes of the body
  - **Expires**: a date after which the resource is considered no longer valid (for cache)

Headers (2)

- Examples:
  - **Last-Modified**: date and hour of last modification of the resource
  - **User-Agent**: client that originates the request
  - **Host**: domain name and port to which the connection is made
    - The URI in the request is only the local part
    - If the server contains more than one website, the “host:” header permits to distinguish the website to which the request refers

```
GET /ss HTTP/1.1
...
Host: www.skenz.it:80
```

Example of an HTTP response

```
GET /index.html HTTP/1.1
Host: www.cs.unibo.it:80



HTTP/1.1 200 OK
Date: Fri, 26 Nov 1999 11:46:53 GMT
Server: Apache/1.3.3 (Unix)
Last-Modified: Mon, 12 Jul 1999 12:55:37 GMT
Accept-Ranges: bytes
Content-Length: 3357
Content-Type: text/html
<HTML> …. </HTML>
```

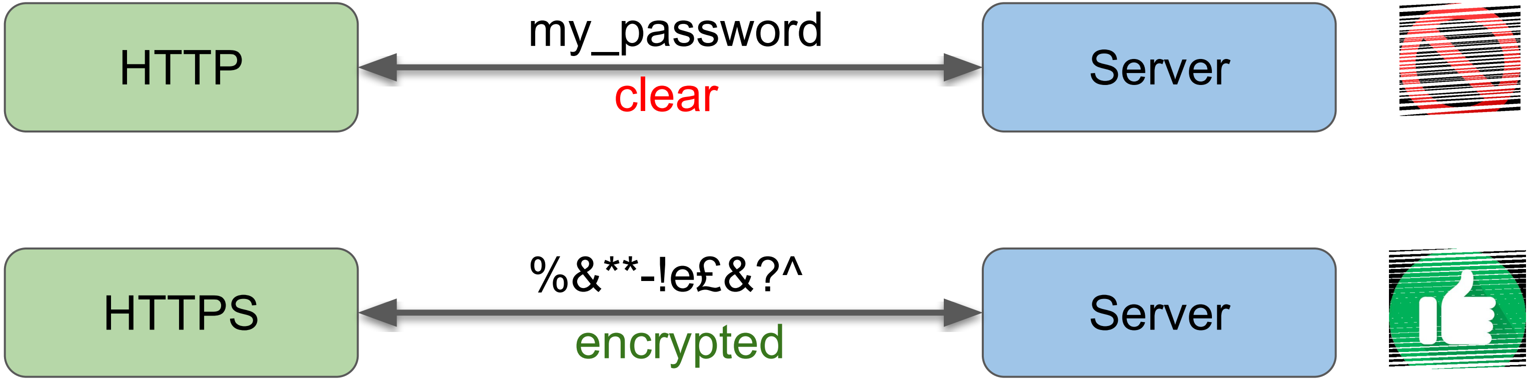
Status line

- **Status line** is the first line of the response message
    - HTTP/1.1 200 OK (in the example of the previous slide)
- **Status number code**
    - First digit identify the class of response
        - **1xx: Informational** - not used
        - **2xx: Success** - requested action correctly accepted
        - **3xx: Redirection** - further action required (to complete the request)
        - **4xx: Client error** - request performed by the client is invalid
        - **5xx: Server error** - server cannot execute the client request
    - Examples of common status codes

200 OK	**401** Unauthorized
301 Resource moved permanently	**403** Forbidden
302 Resource moved temporarily	**404** Not found
400 Bad request	**500** Internal Server Error

Hypertext Transfer Protocol Secure (HTTPS)

- Extension of the HTTP protocol
- HTTP encrypted with Transport Layer Security (TLS)



HTTP

my_password

clear

Server

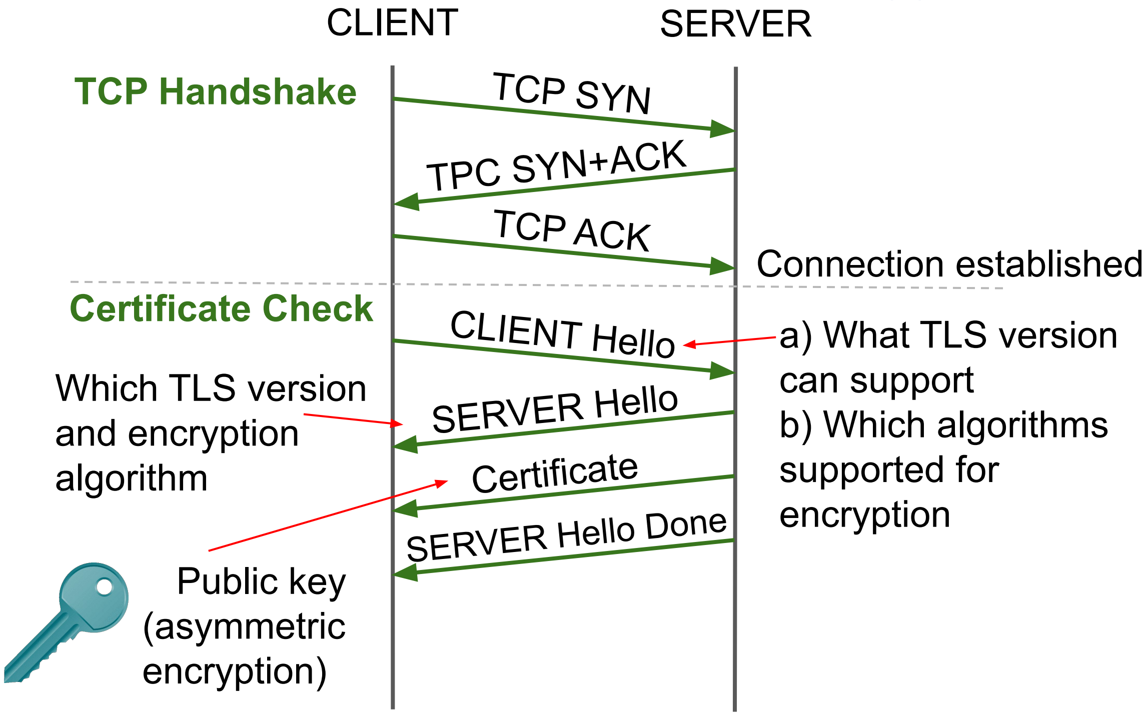
HTTPS

%&**-!e£&?^

encrypted

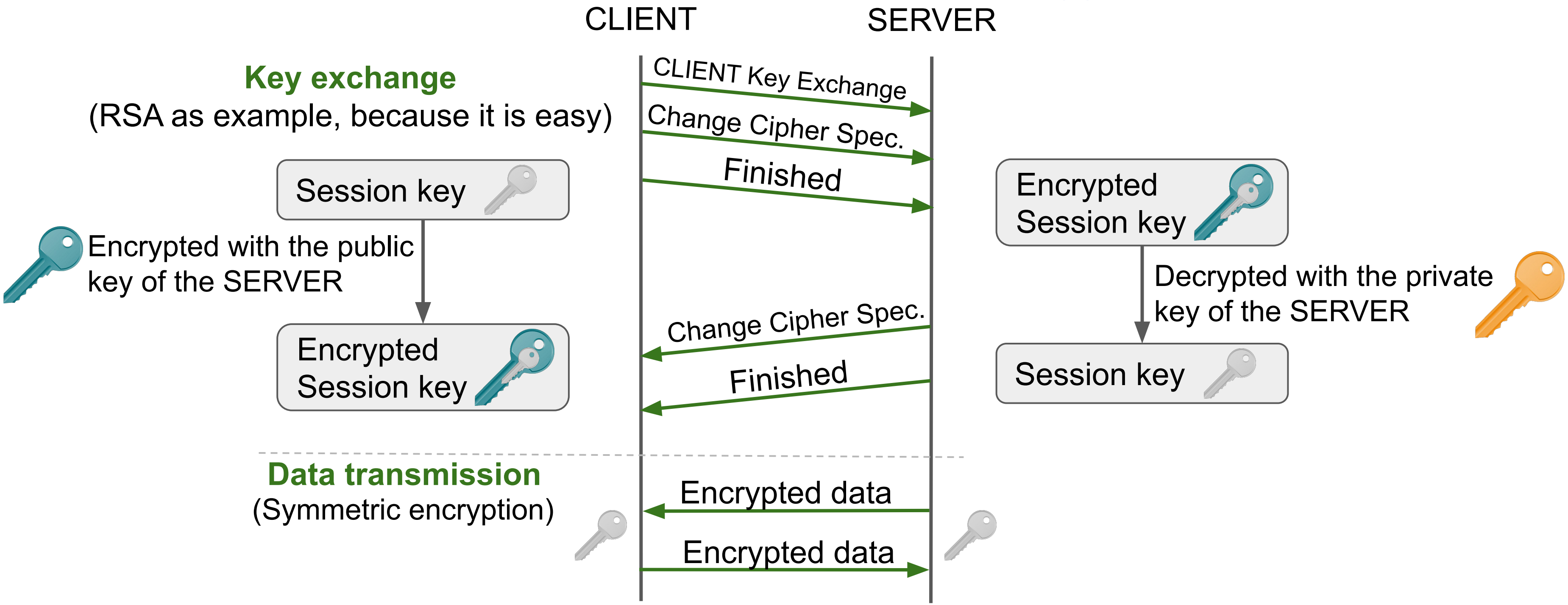
Server

Message exchanges


CLIENT
SERVER
TCP Handshake
TCP SYN
TPC SYN+ACK
TCP ACK
Connection established
Certificate Check
CLIENT Hello
SERVER Hello
Certificate
SERVER Hello Done
a) What TLS version can support
b) Which algorithms supported for encryption
Which TLS version and encryption algorithm
Public key (asymmetric encryption)

At the end of this step, they choose the encryption algorithm (and the server communicate its key)

Message exchanges



requests

- Requests is a simple (but elegant) **HTTP library**
  - Send HTTP/1.1 requests extremely easily
  - No need to manually add query strings to URL
  - No need to form-encode your POST data
- Guide
  - https://pypi.org/project/requests/
  - https://requests.readthedocs.io/en/latest/
- Installation

```
pip install requests
```

math.js

- math.js is web service that
  - allow evaluation of mathematical expression
  - using GET or POST requests
- Guide
  - https://api.mathjs.org/
- GET (Try it by clicking on the URL)
  - https://api.mathjs.org/v4/?expr=2*(7-3) (2*(7-3) = 8)
  - https://api.mathjs.org/v4/?expr=2%2F3&precision=3 (2/3 with precision 3 significant digits = 0.667)
- POST
  - Request:
    - content-type: application/json
    - `{"expr":["a = 2 + 3", "5 * 2"], "precision": 3}`
  - Response:
    - `{"result":["5","10"], "error":null}`

requests in practice

- Request a generic webpage (GET method)

```
r = requests.get("https://www.skenz.it/ss")
print(r.text)
print(r.headers)
print(r.status_code)
```

Output: HTML of the webpage

HTTP header

200 (corresponding to OK)

- math.js GET request

```
r = requests.get("http://api.mathjs.org/v4/", params={"expr": "3*2"})
print(r.text)
```

Output: 6

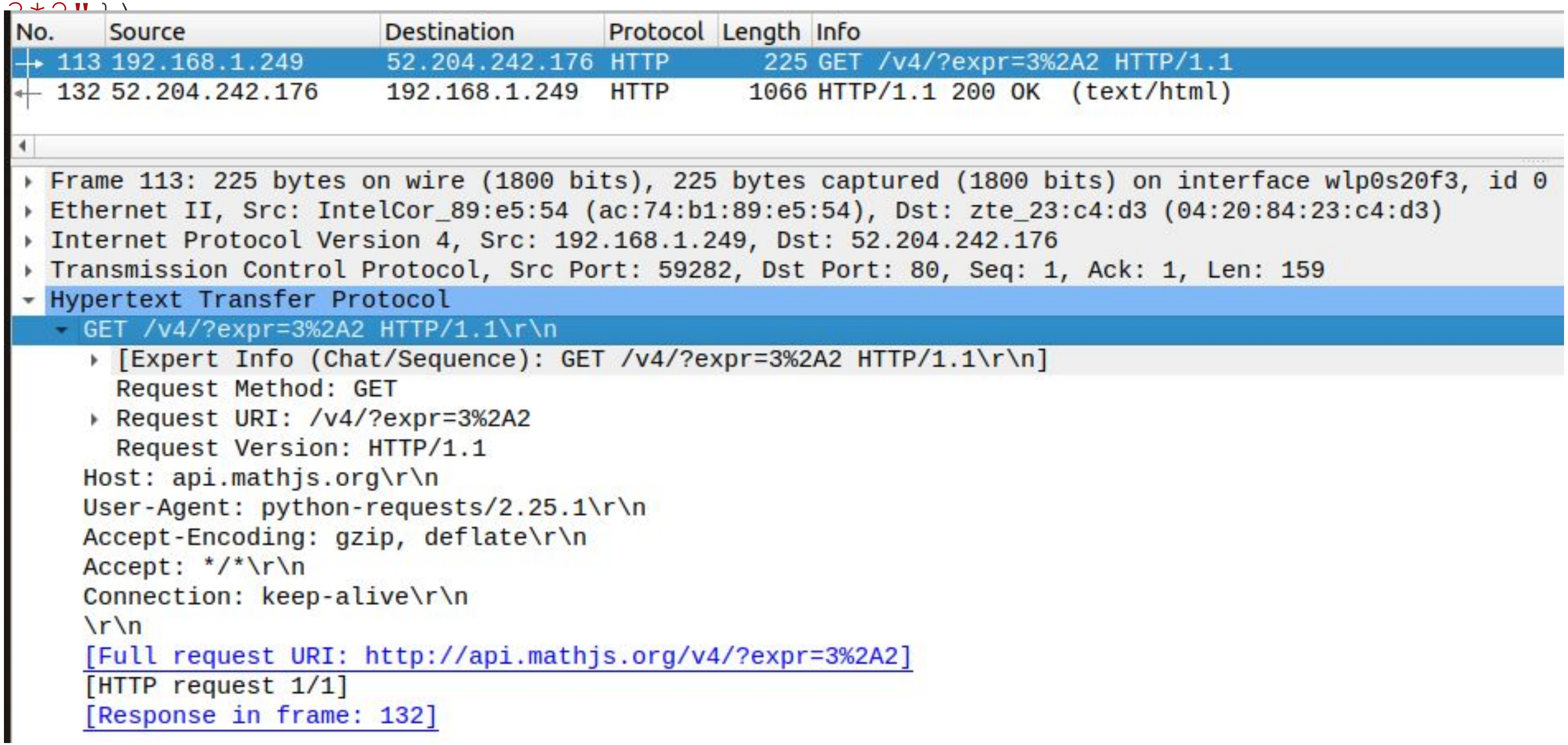
- math.js POST request

```
r = requests.post("http://api.mathjs.org/v4/",
data='{"expr":["3*2"]}',headers={'Content-Type': 'application/json'} )
print(r.text)
```

Output: `{"result":["6"],"error":null}`

Wireshark GET request

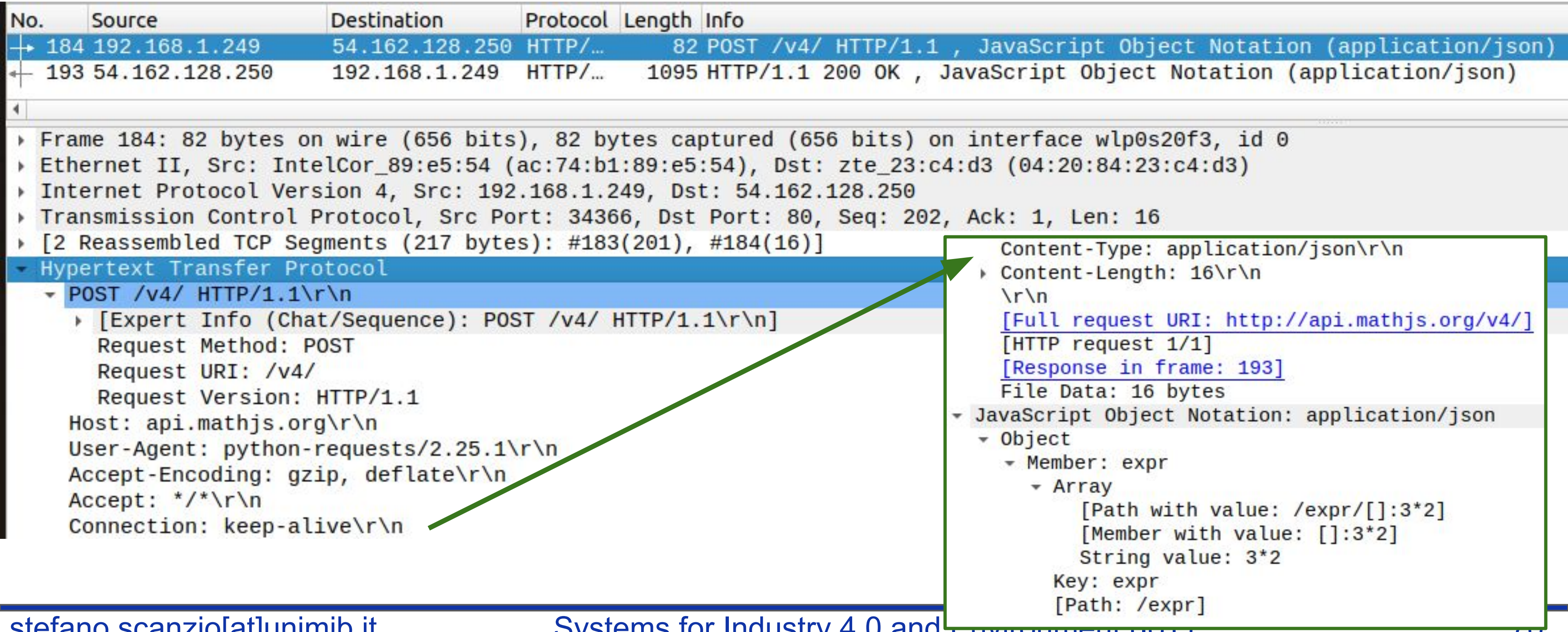
- GET: `r = requests.get("http://api.mathjs.org/v4/", params={"expr": "3*2"})`



```
No.  Source         Destination     Protocol  Length  Info
113  192.168.1.249  52.204.242.176  HTTP         225  GET /v4/?expr=3%2A2 HTTP/1.1
132  52.204.242.176 192.168.1.249   HTTP        1066  HTTP/1.1 200 OK  (text/html)

Frame 113: 225 bytes on wire (1800 bits), 225 bytes captured (1800 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_89:e5:54 (ac:74:b1:89:e5:54), Dst: zte_23:c4:d3 (04:20:84:23:c4:d3)
Internet Protocol Version 4, Src: 192.168.1.249, Dst: 52.204.242.176
Transmission Control Protocol, Src Port: 59282, Dst Port: 80, Seq: 1, Ack: 1, Len: 159
Hypertext Transfer Protocol
  GET /v4/?expr=3%2A2 HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /v4/?expr=3%2A2 HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /v4/?expr=3%2A2
    Request Version: HTTP/1.1
  Host: api.mathjs.org\r\n
  User-Agent: python-requests/2.25.1\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept: */*\r\n
  Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://api.mathjs.org/v4/?expr=3%2A2]
  [HTTP request 1/1]
  [Response in frame: 132]
```

Wireshark POST request

- POST: `r = requests.post("http://api.mathjs.org/v4/", data='{"expr":["3*2"]}',headers={'Content-Type': 'application/json'} )`



```
No.  Source          Destination     Protocol Length Info
184  192.168.1.249   54.162.128.250  HTTP/…       82 POST /v4/ HTTP/1.1 , JavaScript Object Notation (application/json)
193  54.162.128.250  192.168.1.249   HTTP/…     1095 HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
```

```
▸ Frame 184: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: IntelCor_89:e5:54 (ac:74:b1:89:e5:54), Dst: zte_23:c4:d3 (04:20:84:23:c4:d3)
▸ Internet Protocol Version 4, Src: 192.168.1.249, Dst: 54.162.128.250
▸ Transmission Control Protocol, Src Port: 34366, Dst Port: 80, Seq: 202, Ack: 1, Len: 16
▸ [2 Reassembled TCP Segments (217 bytes): #183(201), #184(16)]
▾ Hypertext Transfer Protocol
  ▾ POST /v4/ HTTP/1.1\r\n
    ▸ [Expert Info (Chat/Sequence): POST /v4/ HTTP/1.1\r\n]
      Request Method: POST
      Request URI: /v4/
      Request Version: HTTP/1.1
    Host: api.mathjs.org\r\n
    User-Agent: python-requests/2.25.1\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept: */*\r\n
    Connection: keep-alive\r\n
```

```
    Content-Type: application/json\r\n
  ▸ Content-Length: 16\r\n
    \r\n
    [Full request URI: http://api.mathjs.org/v4/]
    [HTTP request 1/1]
    [Response in frame: 193]
    File Data: 16 bytes
▾ JavaScript Object Notation: application/json
  ▾ Object
    ▾ Member: expr
      ▾ Array
          [Path with value: /expr/[]:3*2]
          [Member with value: []:3*2]
          String value: 3*2
        Key: expr
        [Path: /expr]
```

REST API

A **Rest API** (also called **RESTful API** or RESTful web API), is basically an **architectural style** for an application program interface (API), that **uses HTTP requests** to access and use data.

An API for a website is code that allows two software programs to communicate with each other. These offered services are:

- Scalable
- Stateless
- Easy to maintain

Purposes of REST API

The **purposes of REST API** are:

- Providing a **standardized way** to for clients to interact with server resources over the internet.
- Enables the creation of **scalable and maintainable web services** that can be easily integrated into different applications and platforms.
- REST APIs are widely used in **mobile app and web development**, for building distributed systems and microservices architectures.

Resources in REST API

In **REST API** resources have the following **characteristics**:

- **Definition**: Everything in a RESTful API is a resource, which can be accessed using a unique Uniform Resource Identifier (URI).
- **Example**: Resources can represent real-world objects such as users, products, documents, etc.
- **URI Structure**: Each resource is identified by a unique URI, which serves as its address on the web.

Status codes in REST API

REST API uses HTTP status codes:

- **Purpose**: HTTP codes indicates the success or failure of a request.

- **Common status codes**:
  - 200: OK - Request succeeded.
  - 201: Created - Resource was successfully created.
  - 404: Not Found - Resource not found on the server.
  - 500: Internal Server Error - Server encountered an unexpected condition.

- **Response**: servers return appropriate status codes along with responses to clients about the outcome of their requests.

REST APIs Commands

REST APIs uses several commands to manage resources:

- **GET**: Retrieve a resource or a collection of resources from the server.
- **POST**: Create a new resource on the server.
- **PUT**: Update an existing resource on the server or create a new one if it does not exist.
- **DELETE**: Delete a resource from the server.

- Example Usage: Clients use these HTTP methods to perform actions on server resources.

Data Formats in REST API

Most common **data formats** in REST API is:

- **application/json**: Lightweight data interchange format, easy to read, write, parse and generate.
- **application/xml**: Markup language similar to HTML, commonly used for data exchange between web services.
- application/x-wbe+xml
- application/x-www-form-urlencoded
- multipart/form-data

Examples of REST API

Example 1: User Resource

- Endpoint: ‘`/api/users`’
- HTTP Methods:
  - GET ‘`/api/users`’: Retrieve all users.
  - POST ‘`/api/users`’: Create a new user.

Example 2: Specific User Resource

- Endpoint: ‘`/api/users/{id}`’
- HTTP Methods:
  - GET ‘`/api/users/{id}`’: Retrieve a specific user.
  - PUT ‘`/api/users/{id}`’: Update a specific user.
  - DELETE ‘`/api/users/{id}`’: Delete a specific user.

Flask

- **Flask** is a lightweight and flexible web framework for Python.
  - Open-source
  - Easy-to-use API
  - Actively developed and maintained
  - Flexible and customizable
  - Ideal for building up to medium-size web applications
  - Extensive documentation and active community support
  - Usable to **easily create RESTful API**
- Guide
  - https://flask.palletsprojects.com/en/3.0.x/
- Installation

```
pip install Flask
```

Flask components

Flask main components are:

- **Routes**: URL to which the application responds
- **Templates**: mix of HTML and placeholders to generate web pages dynamically
- **Requests Handling**: python code that handle incoming HTTP requests
- **Extensions**: additional packages or libraries that provide extra functionality to Flask
- **Configuration**: setting up variables for Flask configuration
- **Error Handling**: handling errors that occur during the request-response cycle.

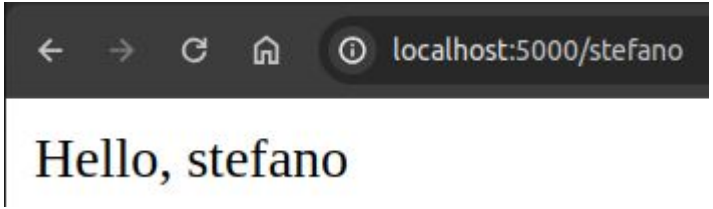
Routes

Basically, **Routes map URLs**. Routing defines the URLs that your application responds to.

- Routes in Flask define the URLs at which your application's functions (view functions) can be accessed.
- **‘`@app.route()`’ decorator**, defines a new route, where ‘`app`’ is the Flask application instance.
- **Routes can include dynamic parts**, specified within ‘`<>`’, allowing for variable data to be passed to the view function.

Route example

```
from flask import Flask

app = Flask(__name__)

@app.route('/<name>')
def index(name):
   return 'Hello, '+str(name)

if __name__ == '__main__':
  app.run()
```



Hello, stefano

Templates

Templates are basically HTML files that separate the presentation layer from the Python code that manages the application.

- Flask uses the **Jinja2 template engine**, which is a modern and widely used templating engine for Python.
    - Documentation: https://jinja.palletsprojects.com/en/3.1.x/
- **Templates increases the dynamism** of the code by inserting data into placeholders and looping over data structures.
- In Flask, template syntax uses double curly braces `{{ }}` for expressions and `{% %}` for control structures.

Template example

```
<!DOCTYPE html>
<html>
<head>
   <title>{{ title }}</title>
</head>
<body>
   <h1>Hello, {{ name }}!</h1>
   <ul>
       {% for item in items %}
           <li>{{ item }}</li>
       {% endfor %}
   </ul>
</body>
</html>
```

Transforming a template into the corresponding HTML

‘index.html’ template:

```
<!DOCTYPE html>
<html lang="en">
<head>
       <title>{{ title }}</title>
</head>
<body>
   <h1>Hello, {{ name }}!</h1>
   <p>Welcome to our website!</p>
</body>
</html>
```

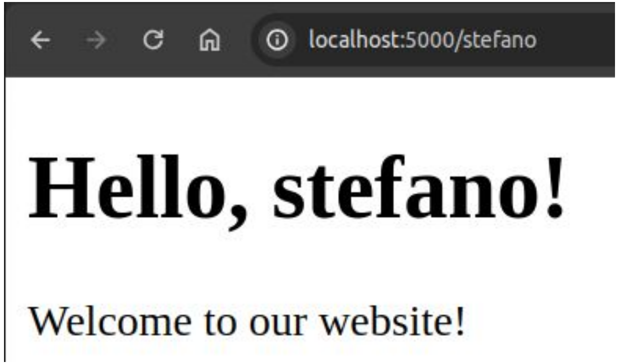
two placeholders: `{{title}}`, `{{name}}` → will be replaced with actual data when rendered.

- make sure your index.html template file is located inside a directory named `templates` in the same directory as your Flask application file

Transforming a template into the corresponding HTML

Flask route that renders the previous template:

```
from flask import Flask, render_template     # import Flask class and the render_template function.

app = Flask(__name__)

@app.route('/<name>')
def index(name):
    # defining values
    title = "Welcome to My Website"
    return render_template('index.html', title=title, name=name)

if __name__ == '__main__':
    app.run(debug=True)
```



Hello, stefano!

Welcome to our website!

When you run this Flask application and navigate to the `/stefano` URL, Flask will render the template with the provided data, resulting in an HTML page where `{{title}}` and `{{name}}` is replaced with "Welcome to My Website" and “stefano" respectively.

Request Handling

Flask provides user-friendly/simple methods to handle incoming requests and access request data:

- The ‘`request`’ object allows you to access data sent with the request, such as form data, query parameters, and file uploads.
- ‘`request.method`’ used to determine the HTTP method used (GET, POST, …).
- ‘`request.form`’: returns a dictionary-like object containing form data submitted with a POST request.
- ‘`request.args`’: returns a dictionary-like object containing the query parameters in the URL.
- ‘`request.headers`’: returns a dictionary-like object containing the request headers.

Request Handling example

```python
from flask import request
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        # Validate username and password
    else:
        return render_template('login.html')
```

Extensions

Extensions in Flask are a third-party packages offering various services and providing additional functions, such as:

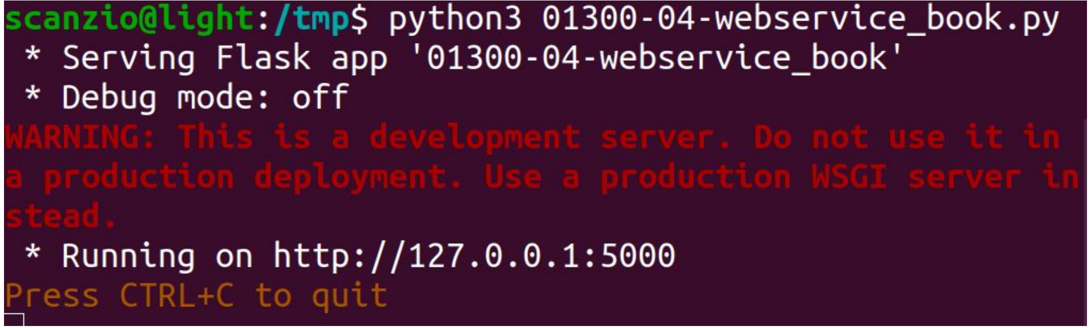
- database integration
- authentication and authorization
- form validation
- caching
- email sending

Example RESTful

Flask

- Download the example: https://www.skenz.it/listing/iot/examples/01300-04-webservice_book.py

```
from flask import Flask, jsonify, request


# Generation of a Flask instance
app = Flask(__name__)


# Sample data (you can have a database)
books = [
    {"id": 1, "title": "Narcis", "author": "John Doe"},
    {"id": 2, "title": "The Glass Bead Game", "author": "Hermann Hesse"},
]


if __name__ == '__main__':
    app.run()
```

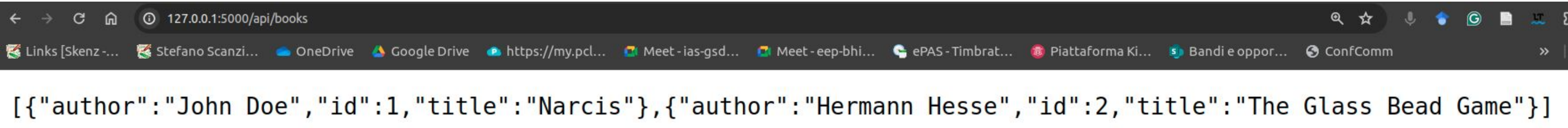


```
scanzio@light:/tmp$ python3 01300-04-webservice_book.py
 * Serving Flask app '01300-04-webservice_book'
 * Debug mode: off
WARNING: This is a development server. Do not use it in
a production deployment. Use a production WSGI server in
stead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

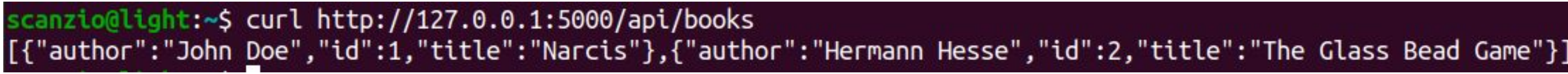
Example RESTful - GET

Flask

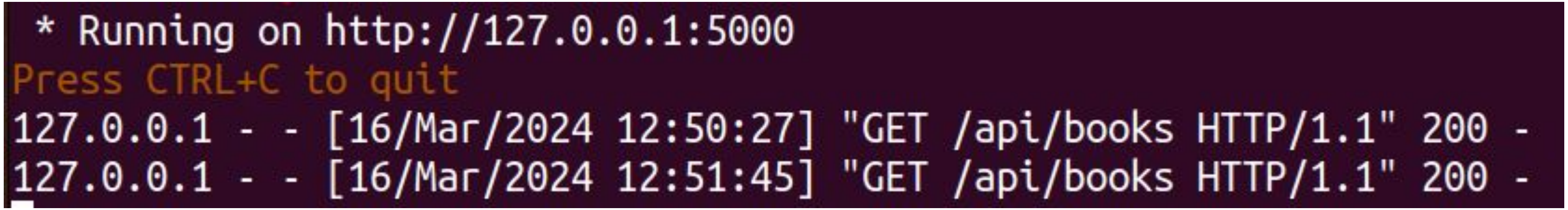
```python
# Route to get all books
@app.route('/api/books', methods=['GET'])
def get_books():
    return jsonify(books)
```



```
[{"author":"John Doe","id":1,"title":"Narcis"},{"author":"Hermann Hesse","id":2,"title":"The Glass Bead Game"}]
```



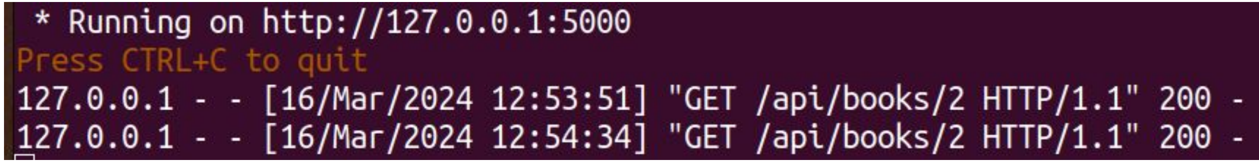
```
scanzio@light:~$ curl http://127.0.0.1:5000/api/books
[{"author":"John Doe","id":1,"title":"Narcis"},{"author":"Hermann Hesse","id":2,"title":"The Glass Bead Game"}]
```



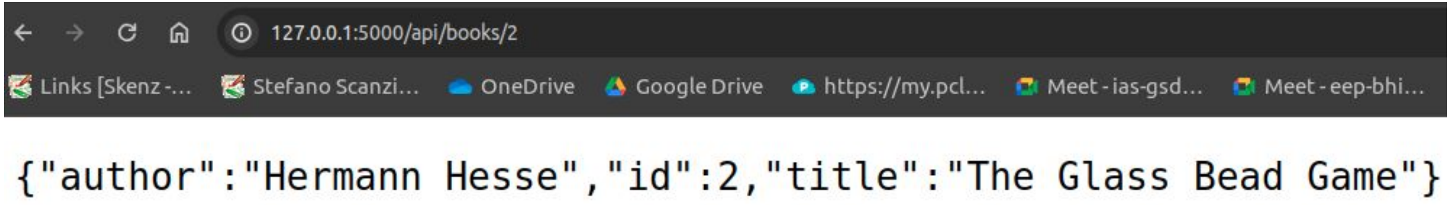
```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [16/Mar/2024 12:50:27] "GET /api/books HTTP/1.1" 200 -
127.0.0.1 - - [16/Mar/2024 12:51:45] "GET /api/books HTTP/1.1" 200 -
```

Example RESTful - GET (2)

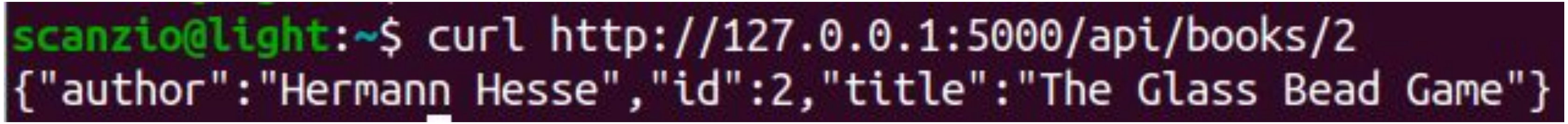
```python
# Route to get a specific book by book_id
@app.route('/api/books/<int:book_id>', methods=['GET'])
def get_book(book_id):
    # Search for the given book with the given book_id
    for book in books:
        if book['id'] == book_id:
            return jsonify(book), 200

    # If no book found with the given book_id, return an error message
    return jsonify({"error": "Book not found"}), 404
```



```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [16/Mar/2024 12:53:51] "GET /api/books/2 HTTP/1.1" 200 -
127.0.0.1 - - [16/Mar/2024 12:54:34] "GET /api/books/2 HTTP/1.1" 200 -
```



```
{"author":"Hermann Hesse","id":2,"title":"The Glass Bead Game"}
```

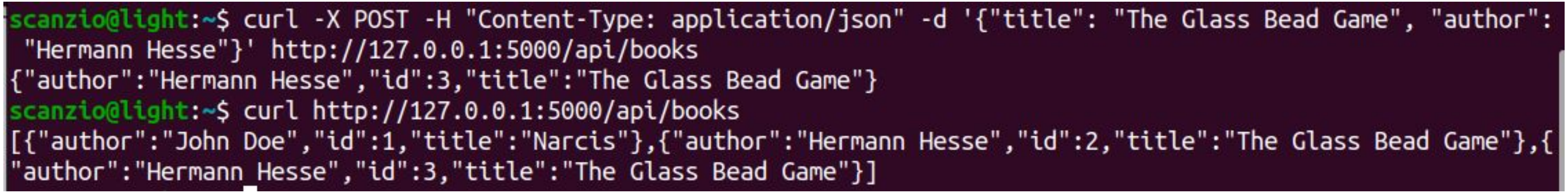


```
scanzio@light:~$ curl http://127.0.0.1:5000/api/books/2
{"author":"Hermann Hesse","id":2,"title":"The Glass Bead Game"}
```

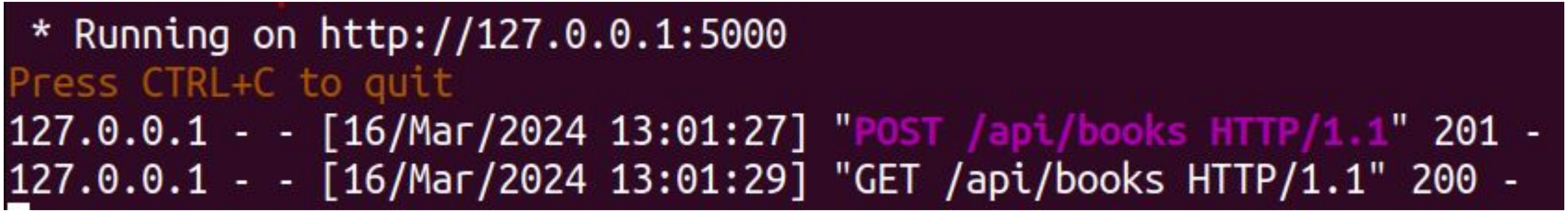
Example RESTful - POST

flask

```python
# Route to add a new book
@app.route('/api/books', methods=['POST'])
def add_book():
    data = request.json
    new_book = {"id": len(books) + 1, "title": data['title'], "author": data['author']}
    books.append(new_book)
    return jsonify(new_book), 201
```



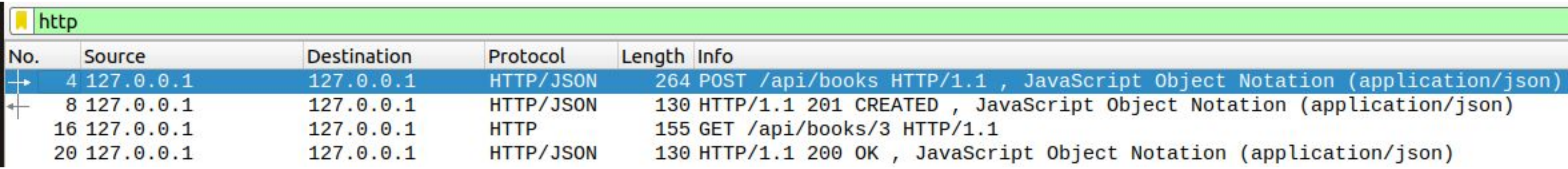
```
scanzio@light:~$ curl -X POST -H "Content-Type: application/json" -d '{"title": "The Glass Bead Game", "author":
 "Hermann Hesse"}' http://127.0.0.1:5000/api/books
{"author":"Hermann Hesse","id":3,"title":"The Glass Bead Game"}
scanzio@light:~$ curl http://127.0.0.1:5000/api/books
[{"author":"John Doe","id":1,"title":"Narcis"},{"author":"Hermann Hesse","id":2,"title":"The Glass Bead Game"},{
"author":"Hermann Hesse","id":3,"title":"The Glass Bead Game"}]
```



```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [16/Mar/2024 13:01:27] "POST /api/books HTTP/1.1" 201 -
127.0.0.1 - - [16/Mar/2024 13:01:29] "GET /api/books HTTP/1.1" 200 -
```

Wireshark

- After the execution of the following command:
  - `curl -X POST -H "Content-Type: application/json" -d '{"title": "The Glass Bead Game", "author":"Hermann Hesse"}' http://127.0.0.1:5000/api/books` **(POST)**
  - `curl http://127.0.0.1:5000/api/books/3` **(GET)**
- Download the Wireshark log:
  - https://www.skenz.it/listing/iot/wireshark/FLASK_POST_GET.pcap

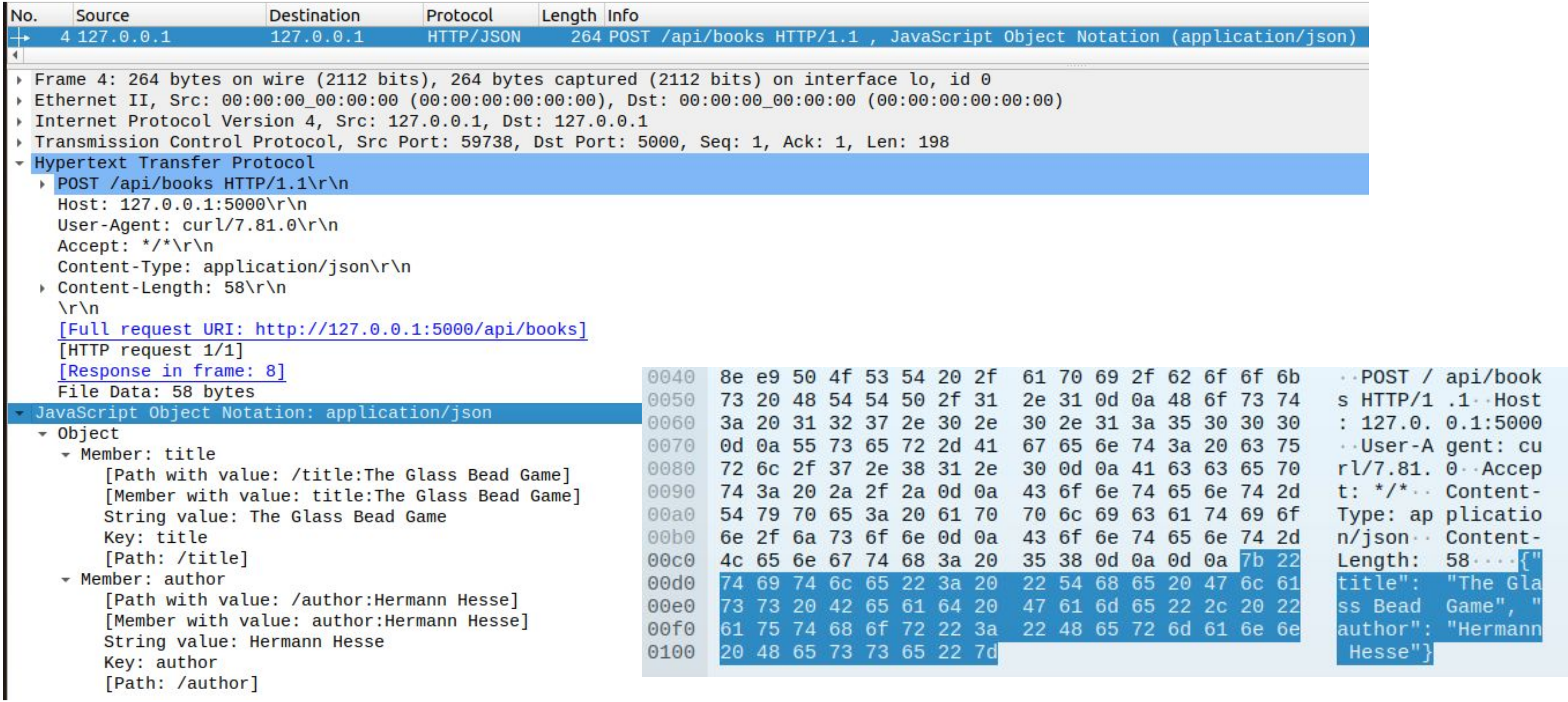


http

No.	Source	Destination	Protocol	Length	Info
4	127.0.0.1	127.0.0.1	HTTP/JSON	264	POST /api/books HTTP/1.1 , JavaScript Object Notation (application/json)
8	127.0.0.1	127.0.0.1	HTTP/JSON	130	HTTP/1.1 201 CREATED , JavaScript Object Notation (application/json)
16	127.0.0.1	127.0.0.1	HTTP	155	GET /api/books/3 HTTP/1.1
20	127.0.0.1	127.0.0.1	HTTP/JSON	130	HTTP/1.1 200 OK , JavaScript Object Notation (application/json)

Wireshark: POST

Flask



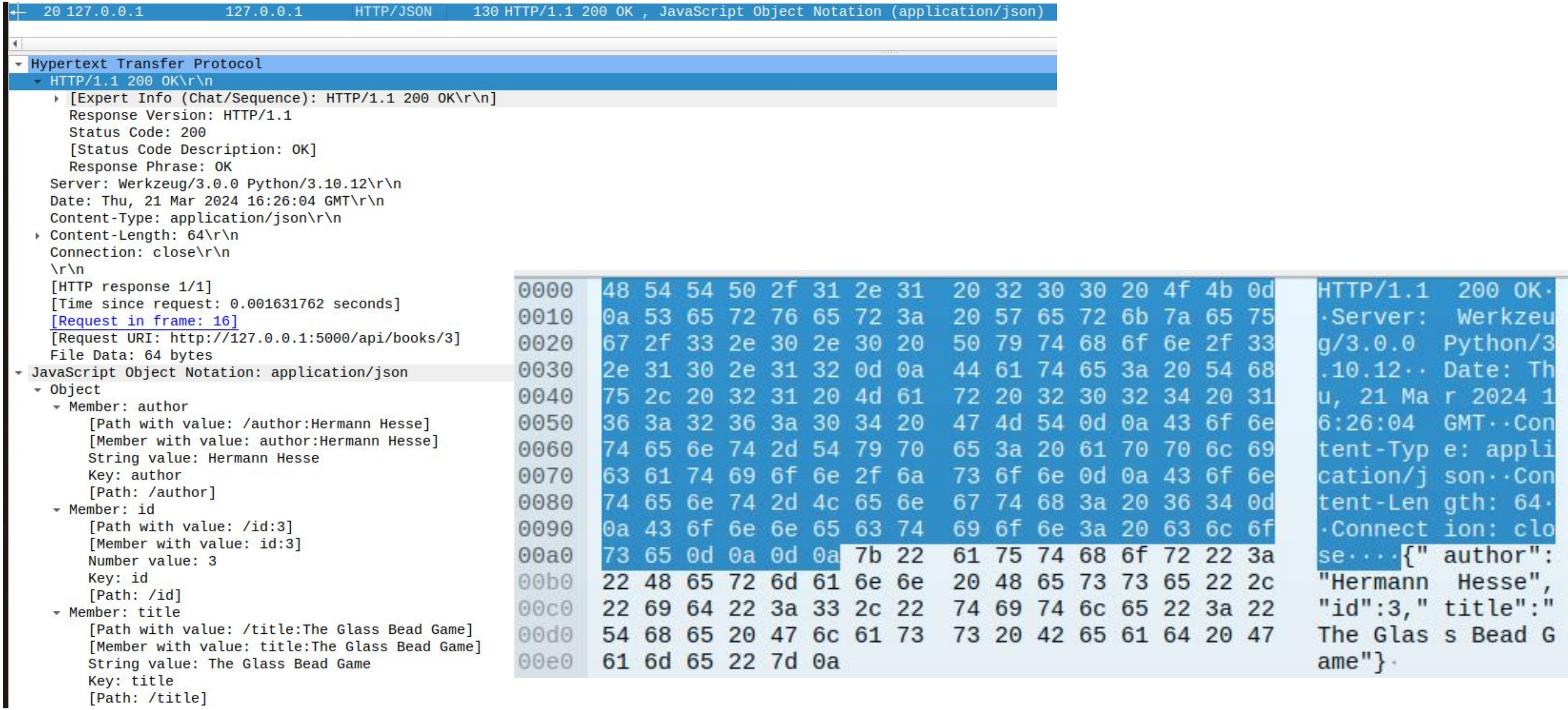
No.	Source	Destination	Protocol	Length	Info
4	127.0.0.1	127.0.0.1	HTTP/JSON	264	POST /api/books HTTP/1.1 , JavaScript Object Notation (application/json)

```
▸ Frame 4: 264 bytes on wire (2112 bits), 264 bytes captured (2112 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 59738, Dst Port: 5000, Seq: 1, Ack: 1, Len: 198
▾ Hypertext Transfer Protocol
  ▸ POST /api/books HTTP/1.1\r\n
    Host: 127.0.0.1:5000\r\n
    User-Agent: curl/7.81.0\r\n
    Accept: */*\r\n
    Content-Type: application/json\r\n
  ▸ Content-Length: 58\r\n
    \r\n
    [Full request URI: http://127.0.0.1:5000/api/books]
    [HTTP request 1/1]
    [Response in frame: 8]
    File Data: 58 bytes
▾ JavaScript Object Notation: application/json
  ▾ Object
    ▾ Member: title
        [Path with value: /title:The Glass Bead Game]
        [Member with value: title:The Glass Bead Game]
        String value: The Glass Bead Game
        Key: title
        [Path: /title]
    ▾ Member: author
        [Path with value: /author:Hermann Hesse]
        [Member with value: author:Hermann Hesse]
        String value: Hermann Hesse
        Key: author
        [Path: /author]
```

```
0040  8e e9 50 4f 53 54 20 2f  61 70 69 2f 62 6f 6f 6b   ··POST / api/book
0050  73 20 48 54 54 50 2f 31  2e 31 0d 0a 48 6f 73 74   s HTTP/1 .1··Host
0060  3a 20 31 32 37 2e 30 2e  30 2e 31 3a 35 30 30 30   : 127.0. 0.1:5000
0070  0d 0a 55 73 65 72 2d 41  67 65 6e 74 3a 20 63 75   ··User-A gent: cu
0080  72 6c 2f 37 2e 38 31 2e  30 0d 0a 41 63 63 65 70   rl/7.81. 0··Accep
0090  74 3a 20 2a 2f 2a 0d 0a  43 6f 6e 74 65 6e 74 2d   t: */*·· Content-
00a0  54 79 70 65 3a 20 61 70  70 6c 69 63 61 74 69 6f   Type: ap plicatio
00b0  6e 2f 6a 73 6f 6e 0d 0a  43 6f 6e 74 65 6e 74 2d   n/json·· Content-
00c0  4c 65 6e 67 74 68 3a 20  35 38 0d 0a 0d 0a 7b 22   Length:  58····{"
00d0  74 69 74 6c 65 22 3a 20  22 54 68 65 20 47 6c 61   title":  "The Gla
00e0  73 73 20 42 65 61 64 20  47 61 6d 65 22 2c 20 22   ss Bead  Game", "
00f0  61 75 74 68 6f 72 22 3a  22 48 65 72 6d 61 6e 6e   author": "Hermann
0100  20 48 65 73 73 65 22 7d                             Hesse"}
```

Wireshark: Response

Flask

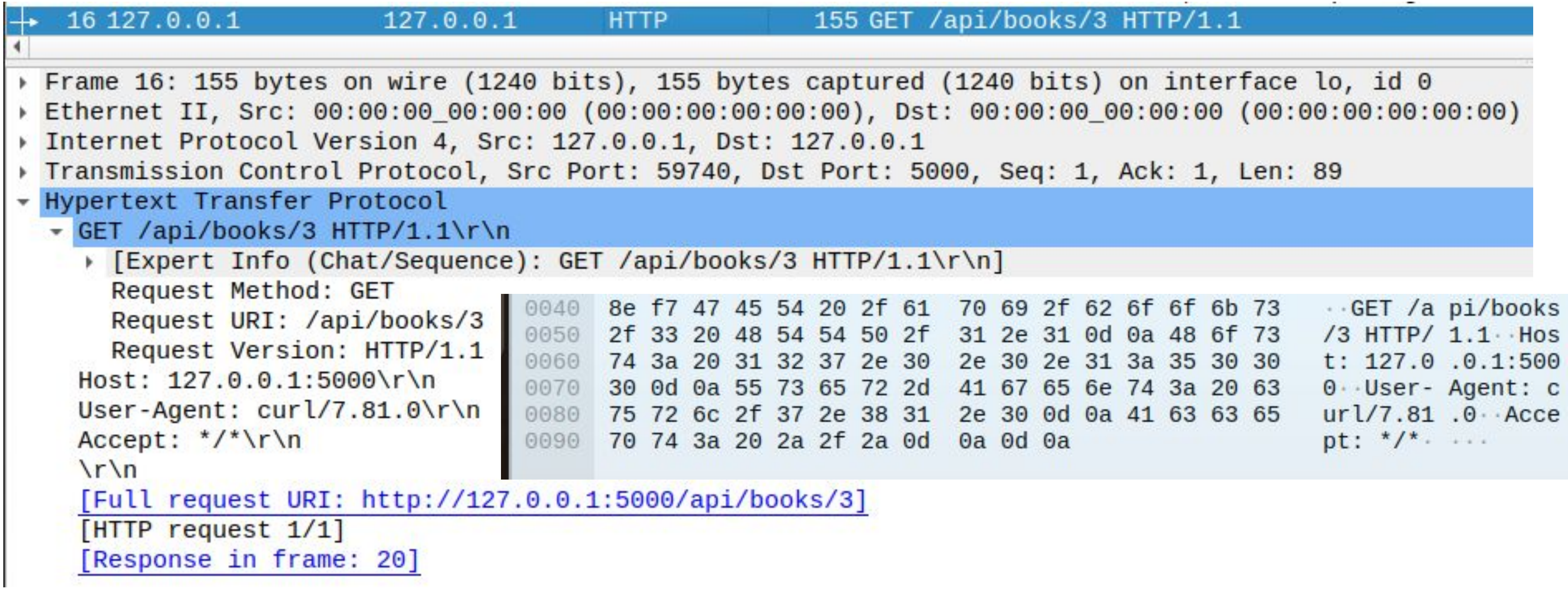


```
20 127.0.0.1          127.0.0.1          HTTP/JSON     130 HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
```

```
▾ Hypertext Transfer Protocol
  ▾ HTTP/1.1 200 OK\r\n
    ▸ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
    Server: Werkzeug/3.0.0 Python/3.10.12\r\n
    Date: Thu, 21 Mar 2024 16:26:04 GMT\r\n
    Content-Type: application/json\r\n
  ▸ Content-Length: 64\r\n
    Connection: close\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.001631762 seconds]
    [Request in frame: 16]
    [Request URI: http://127.0.0.1:5000/api/books/3]
    File Data: 64 bytes
▾ JavaScript Object Notation: application/json
  ▾ Object
    ▾ Member: author
        [Path with value: /author:Hermann Hesse]
        [Member with value: author:Hermann Hesse]
        String value: Hermann Hesse
        Key: author
        [Path: /author]
    ▾ Member: id
        [Path with value: /id:3]
        [Member with value: id:3]
        Number value: 3
        Key: id
        [Path: /id]
    ▾ Member: title
        [Path with value: /title:The Glass Bead Game]
        [Member with value: title:The Glass Bead Game]
        String value: The Glass Bead Game
        Key: title
        [Path: /title]
```

```
0000  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK·
0010  0a 53 65 72 76 65 72 3a  20 57 65 72 6b 7a 65 75   ·Server:  Werkzeu
0020  67 2f 33 2e 30 2e 30 20  50 79 74 68 6f 6e 2f 33   g/3.0.0  Python/3
0030  2e 31 30 2e 31 32 0d 0a  44 61 74 65 3a 20 54 68   .10.12··  Date: Th
0040  75 2c 20 32 31 20 4d 61  72 20 32 30 32 34 20 31   u, 21 Ma r 2024 1
0050  36 3a 32 36 3a 30 34 20  47 4d 54 0d 0a 43 6f 6e   6:26:04  GMT··Con
0060  74 65 6e 74 2d 54 79 70  65 3a 20 61 70 70 6c 69   tent-Typ e: appli
0070  63 61 74 69 6f 6e 2f 6a  73 6f 6e 0d 0a 43 6f 6e   cation/j son··Con
0080  74 65 6e 74 2d 4c 65 6e  67 74 68 3a 20 36 34 0d   tent-Len gth: 64·
0090  0a 43 6f 6e 6e 65 63 74  69 6f 6e 3a 20 63 6c 6f   ·Connect ion: clo
00a0  73 65 0d 0a 0d 0a 7b 22  61 75 74 68 6f 72 22 3a   se····{" author":
00b0  22 48 65 72 6d 61 6e 6e  20 48 65 73 73 65 22 2c   "Hermann  Hesse",
00c0  22 69 64 22 3a 33 2c 22  74 69 74 6c 65 22 3a 22   "id":3," title":"
00d0  54 68 65 20 47 6c 61 73  73 20 42 65 61 64 20 47   The Glas s Bead G
00e0  61 6d 65 22 7d 0a                                   ame"}·
```

Wireshark: GET

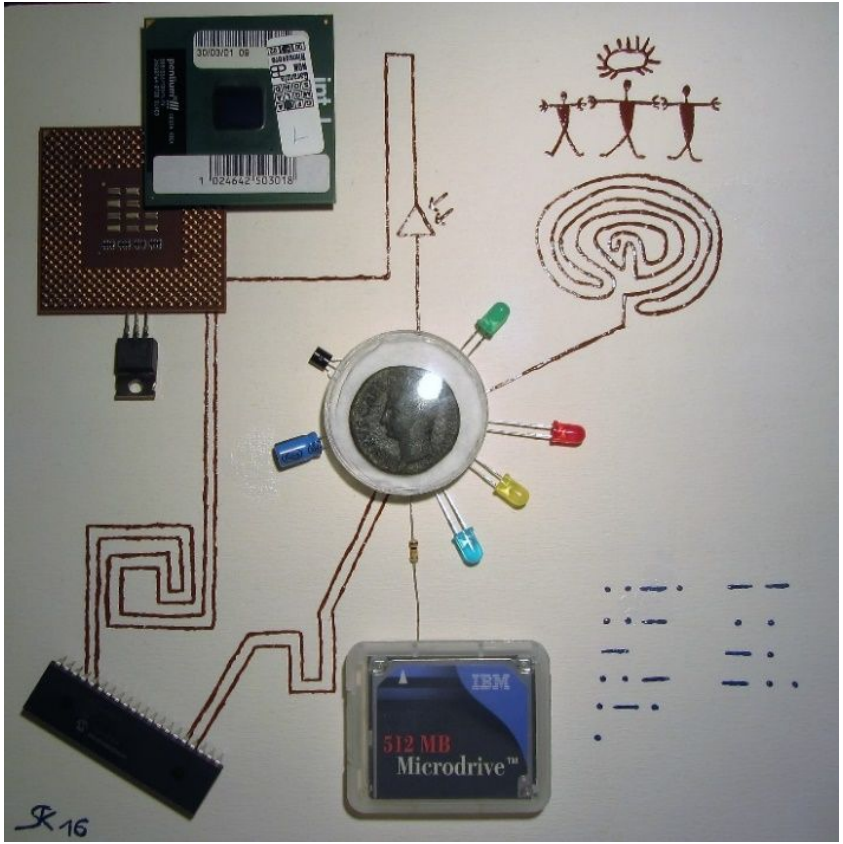


```
16 127.0.0.1          127.0.0.1          HTTP      155 GET /api/books/3 HTTP/1.1

▸ Frame 16: 155 bytes on wire (1240 bits), 155 bytes captured (1240 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 59740, Dst Port: 5000, Seq: 1, Ack: 1, Len: 89
▾ Hypertext Transfer Protocol
  ▾ GET /api/books/3 HTTP/1.1\r\n
    ▸ [Expert Info (Chat/Sequence): GET /api/books/3 HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /api/books/3
      Request Version: HTTP/1.1
    Host: 127.0.0.1:5000\r\n
    User-Agent: curl/7.81.0\r\n
    Accept: */*\r\n
    \r\n
    [Full request URI: http://127.0.0.1:5000/api/books/3]
    [HTTP request 1/1]
    [Response in frame: 20]
```

```
0040  8e f7 47 45 54 20 2f 61  70 69 2f 62 6f 6f 6b 73   ··GET /a pi/books
0050  2f 33 20 48 54 54 50 2f  31 2e 31 0d 0a 48 6f 73   /3 HTTP/ 1.1··Hos
0060  74 3a 20 31 32 37 2e 30  2e 30 2e 31 3a 35 30 30   t: 127.0 .0.1:500
0070  30 0d 0a 55 73 65 72 2d  41 67 65 6e 74 3a 20 63   0··User- Agent: c
0080  75 72 6c 2f 37 2e 38 31  2e 30 0d 0a 41 63 63 65   url/7.81 .0··Acce
0090  70 74 3a 20 2a 2f 2a 0d  0a 0d 0a                  pt: */*· ···
```

Thank you !!!

Futuristic IoT device…



Stefano Scanzio

skenz.it/ss

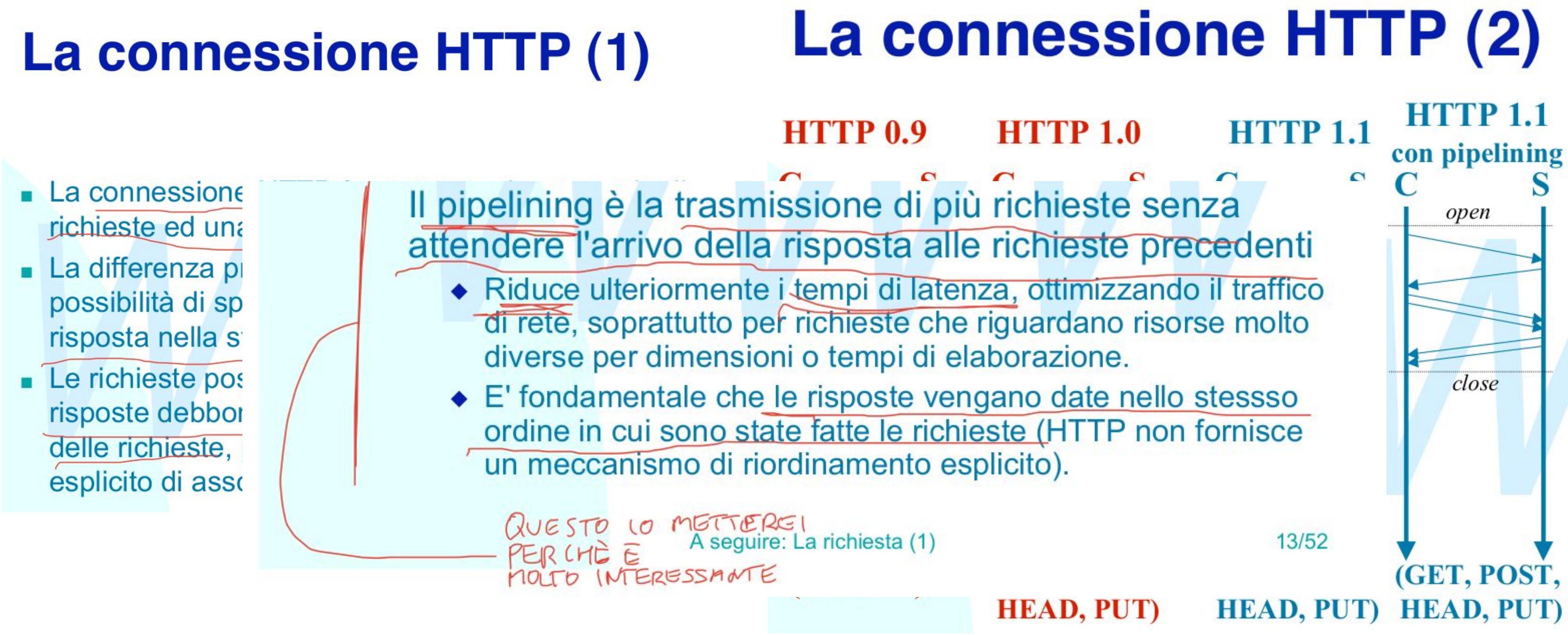
```
sudo apt install mosquitto

sudo /etc/init.d/mosquitto stop

sudo /etc/init.d/mosquitto start

sudo /etc/init.d/mosquitto restart

sudo systemctl restart mosquitto

/etc/mosquitto/mosquitto.conf
```

Schedule

- Week 1: Introduction to python
- Week 2: Python advanced (a)
- Week 3:
    - Python advanced (b)
    - Message passing protocol (01-message_passing_handout.pdf)
- Week 4:
    - Paho MQTT in python
        - First examples about Paho MQTT (00100, 00200, 00300, 00400)
    - Wireshark
    - tkinter
        - MQTT chat with graphical interface (00600)

Schedule (2)

- Week 5:
  - REST API and Data Formats (02-rest_api_handout.pdf)
  - JSON (example 00500)
  - URI
  - HTTP/HTTPS
  - request (example 00800)
- Week 6:
  - Example of MQTT services (00900, 01000, 01100)
  - flask
  - Web services (examples 01200, 01400)
  - If time something about Linux



La connessione HTTP (1)

- La connessione
  richieste ed una
- La differenza pr
  possibilità di sp
  risposta nella s
- Le richieste pos
  risposte debbon
  delle richieste,
  esplicito di asso

La connessione HTTP (2)

Il pipelining è la trasmissione di più richieste senza attendere l'arrivo della risposta alle richieste precedenti

- ◆ Riduce ulteriormente i tempi di latenza, ottimizzando il traffico di rete, soprattutto per richieste che riguardano risorse molto diverse per dimensioni o tempi di elaborazione.
- ◆ E' fondamentale che le risposte vengano date nello stessso ordine in cui sono state fatte le richieste (HTTP non fornisce un meccanismo di riordinamento esplicito).

QUESTO LO METTEREI
PERCHÈ È
MOLTO INTERESSANTE

A seguire: La richiesta (1)

13/52