

# Network traffic analysis with Python

---

Nadja Deininger



# About me & this talk

---

- I'm a Python developer with an interest in networking and security
- I'm not an expert (yet)
- This is an introduction to the topic, suitable for complete beginners

## Contact:

Twitter: [@machine\\_person](https://twitter.com/machine_person)

Mail: [nadja@ef.gy](mailto:nadja@ef.gy)

Web: [machineperson.github.io](https://machineperson.github.io)

# How does it work?

---

Capture network packets using software

- Wireshark
- `tshark` or `tcpdump`

To get all packets on the network you might need to use **port mirroring**.

# What is a network packet?

---

When you send data over a network it will be sent in one or more units called packets.

Each packet contains control information (e.g. source, destination) together with the data you are sending.

Long messages may be split across multiple packets:

- Routers and switches have limited buffer sizes
- Transfer is not 100% reliable, some packets may be dropped

# Example packet: DNS request

Ethernet

IPv6

UDP

DNS

```
▶ Frame 2799: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface 0
Ethernet II, Src: Micro-St_50:78:65 (d8:cb:8a:50:78:65), Dst: Tp-LinkT_c0:26:98 (14:cc:20:c0:26:98)
▶ Destination: Tp-LinkT_c0:26:98 (14:cc:20:c0:26:98)
▶ Source: Micro-St_50:78:65 (d8:cb:8a:50:78:65)
  Type: IPv6 (0x86dd)
▶ Internet Protocol Version 6, Src: fdc9:2a33:8c43:0:5489:a4d7:2068:b55d, Dst: fdc9:2a33:8c43::1
  0110 .... = Version: 6
  ▶ .... 0000 0000 .... = Traffic class: 0x00 (DSCP: CS0, ECN: Not-ECT)
  .... 0000 0000 0000 0000 = Flow label: 0x00000
  Payload length: 40
  Next header: UDP (17)
  Hop limit: 64
  Source: fdc9:2a33:8c43:0:5489:a4d7:2068:b55d
  Destination: fdc9:2a33:8c43::1
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
▶ User Datagram Protocol, Src Port: 53945, Dst Port: 53
  Source Port: 53945
  Destination Port: 53
  Length: 40
  Checksum: 0x37e2 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 22]
▶ Domain Name System (query)
  [Response In: 2808]
  Transaction ID: 0x7952
  ▶ Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  ▶ Queries
    ▶ www.google.com: type A, class IN
0000 14 cc 20 c0 26 98 d8 cb 8a 50 78 65 86 dd 60 00 ... .&... .Pxe...
0010 00 00 00 28 11 40 fd c9 2a 33 8c 43 00 00 54 89 ... (.@... *3.C..T.
0020 a4 d7 20 68 b5 5d fd c9 2a 33 8c 43 00 00 00 00 ... .h.]... *3.C....
0030 00 00 00 00 00 01 d2 b9 00 35 00 28 37 e2 79 52 ..... .5.(7.yR
0040 01 00 00 01 00 00 00 00 00 00 03 77 77 77 06 67 ... .www.g
0050 6f 6f 67 6c 65 03 63 6f 6d 00 00 01 00 01      oogle.co m.....
```

# How can we do this in Python?

---

e.g. [pyshark](#) library

- Python wrapper for tshark, so tshark must be installed
- Python 3 only
- Uses tshark's parsing capabilities

Other libraries:

- [pypcapfile](#) - for analysing capture files
- [pypcap](#) - live packet capture, based on libpcap

# pyshark example - live capture

---

```
import pyshark

cap = pyshark.LiveCapture(interface="eth0")

for packet in capture.sniff_continuously(packet_count=5):
    print(packet)
```

# pyshark example - existing capture

```
import pyshark

cap = pyshark.FileCapture(filename)

packet = cap[0]

print(packet)

# Layer objects with control
information

link_layer = packet.layers[0]
```

```
>>> print(cap[3])
Packet (Length: 97)
Layer ETH:
  Type: IPv4 (0x0800)
  Destination: d8:cb:8a:50:78:65
  .... ..0 .... ..0 .... ..0 .... = IG bit: Individual address (unicast)
  Source: 14:cc:20:c0:26:98
  .... ..0 .... ..0 .... ..0 .... = LG bit: Globally unique address (factory default)
  Address: d8:cb:8a:50:78:65
  .... ..0 .... ..0 .... ..0 .... = IG bit: Individual address (unicast)
  .... ..0 .... ..0 .... ..0 .... = LG bit: Globally unique address (factory default)
  Address: 14:cc:20:c0:26:98
Layer IP:
  Flags: 0x4000, Don't fragment
  ..1. .... ..0 .... ..0 .... = Don't fragment: Set
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Header checksum: 0xdabe [validation disabled]
  Time to live: 239
  0000 00.. = Differentiated Services Codepoint: Default (0)
  Header checksum status: Unverified
  Protocol: TCP (6)
  ..0 0000 0000 0000 = Fragment offset: 0
  Destination: 10.2.0.133
  ..0. .... ..0 .... ..0 .... = More fragments: Not set
  Total Length: 83
  Identification: 0x1b5b (7003)
  .... 0101 = Header Length: 20 bytes (5)
  0100 .... = Version: 4
  .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Source: 34.253.104.7
  0... ..0 .... ..0 .... = Reserved bit: Not set
Layer TCP:
  Timestamps
  Source Port: 443
  .... ..0. .... 1... = Push: Set
  SEQ/ACK analysis
  Flags: 0x018 (PSH, ACK)
  Urgent pointer: 0
```



# pyshark example - finding protocols

```
protonums = {1: "ICMP",
             6: "TCP",
             17: "UDP",
             58: "IPv6-ICMP"}

ip_layer = packet.layers[1]
protocol = None

src_addr = ip_layer.src
dst_addr = ip_layer.dst

if ip_layer.version == "4":
    protocol = ip_layer.proto
elif ip_layer.version == "6":
    protocol = ip_layer.nxt

return {"src_addr": src_addr,
        "dst_addr": dst_addr,
        "protocol": protonums.get(int(protocol), protocol)}
```

# Potential applications

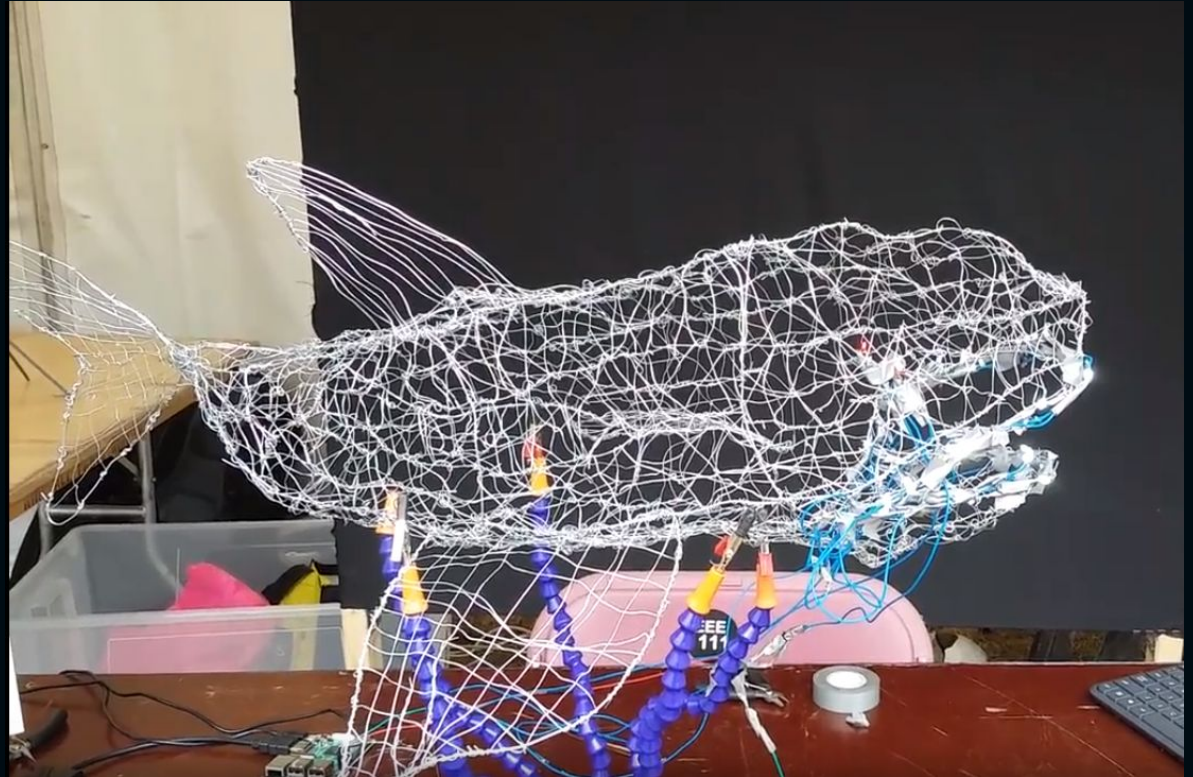
---

- Live capture with monitoring instrumentation
- Statistics on capture files
- Data visualisation
- ...or just light up some LEDs because I can...

# Meet Sharky...

- Wire shark model
- pre-recorded packet capture
- Python program on a Raspberry Pi that interprets the packets
- LEDs

→ Network-based blinkenlights!



Thank you!

