

A night-time photograph of the Dubai skyline, featuring numerous illuminated skyscrapers and their reflections on the water in the foreground. The Burj Khalifa is the most prominent building in the center.

Real Time Packet Processing with FPGAs

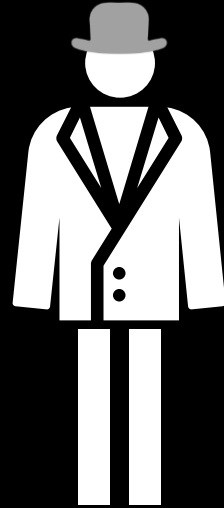
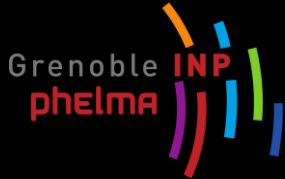
A network security toolbox with encryption features designed for FPGA logic-fabrics

Hack In The Box Dubai – 2018

Matteo Collura

About me

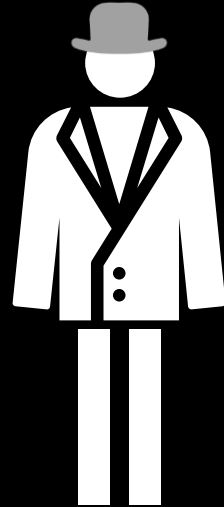
- ❖ M.Sc in Micro and Nanotechnologies for Integrated systems
- ❖ B.Sc in Electronics Engineering



About me

- ❖ Amateur and passionate about IT Security
- ❖ Speaker at International security conferences since 2013

DEFCON



blackhat
ARSENAL

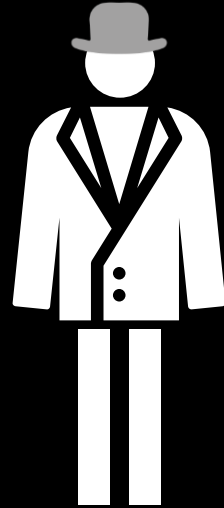


About me

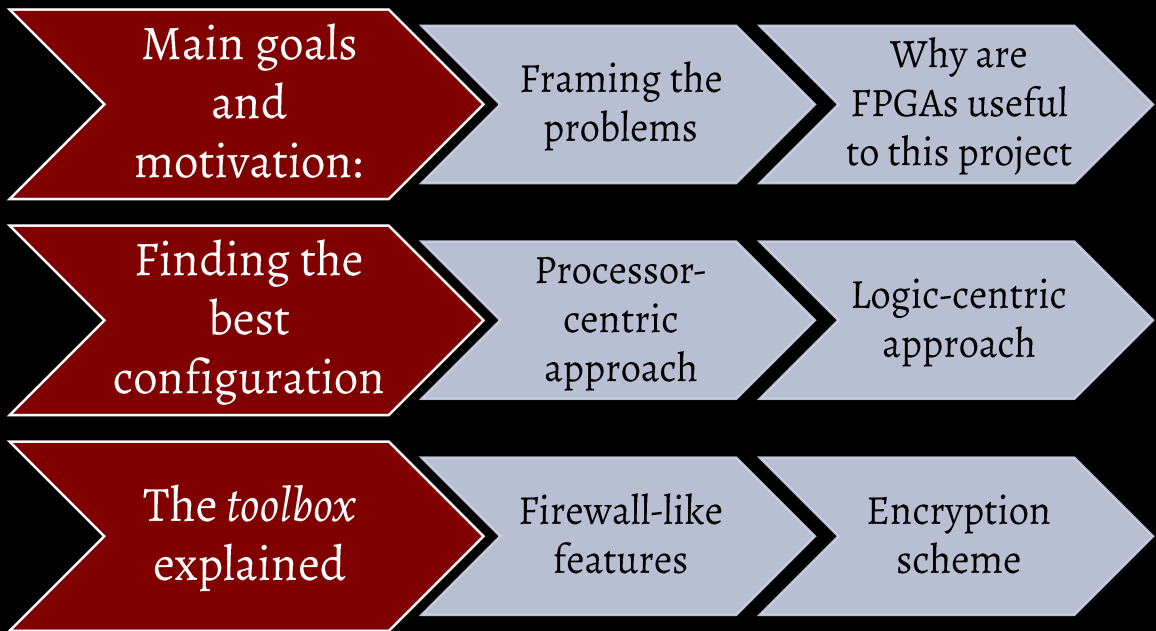
- ❖ Worked for 6-month Master's Thesis Project at Knowledge Resources GmbH



www.knowres.ch



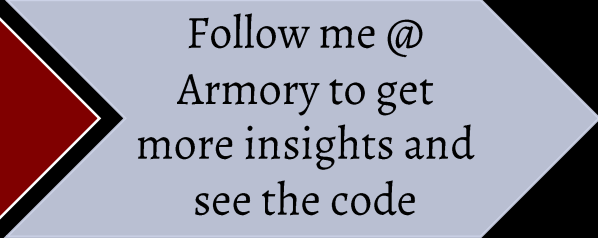
Outline



Outline

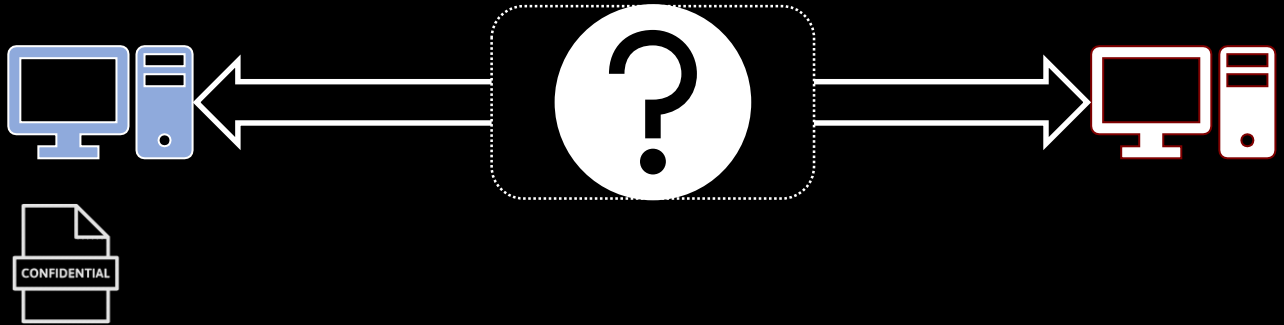


Results
and Demo

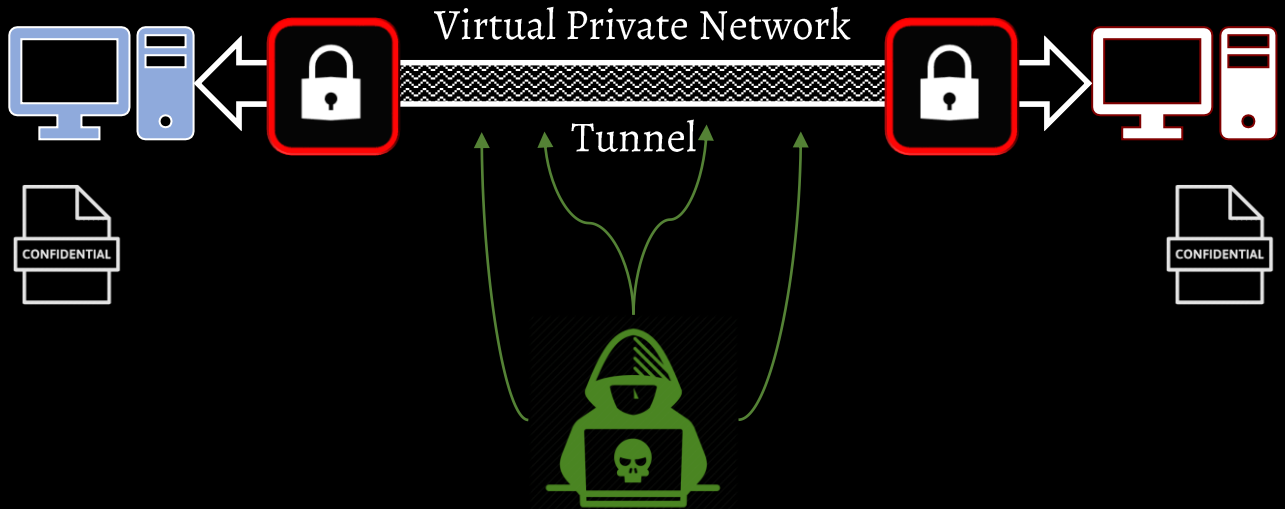


Follow me @
Armory to get
more insights and
see the code

How to deliver confidential information securely?



VPN



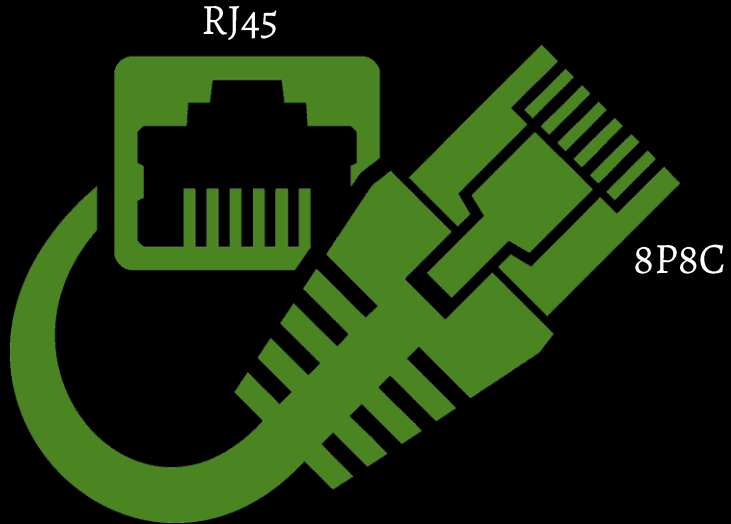
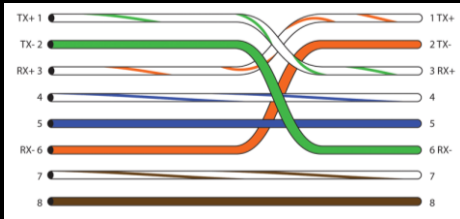
VPN



Real
Time? 

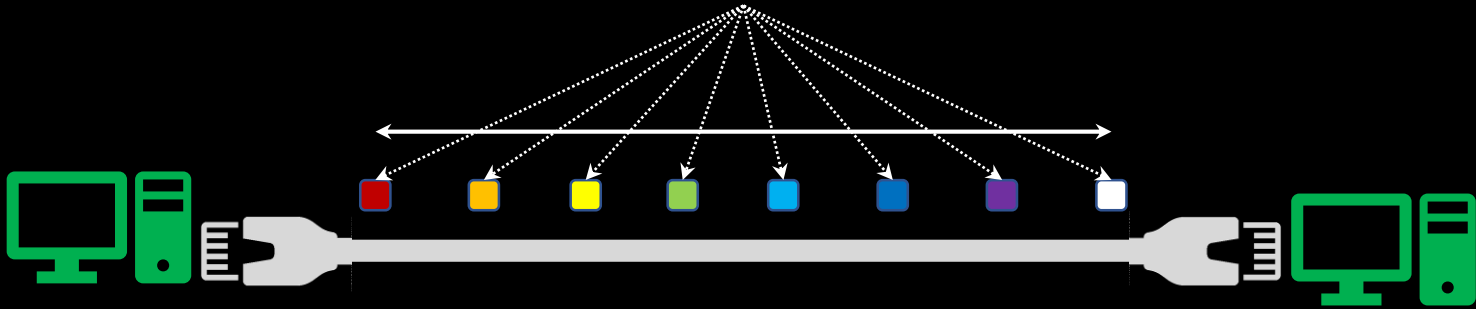
Ethernet connections run at 1Gbps commonly

IEEE 802.3 standard

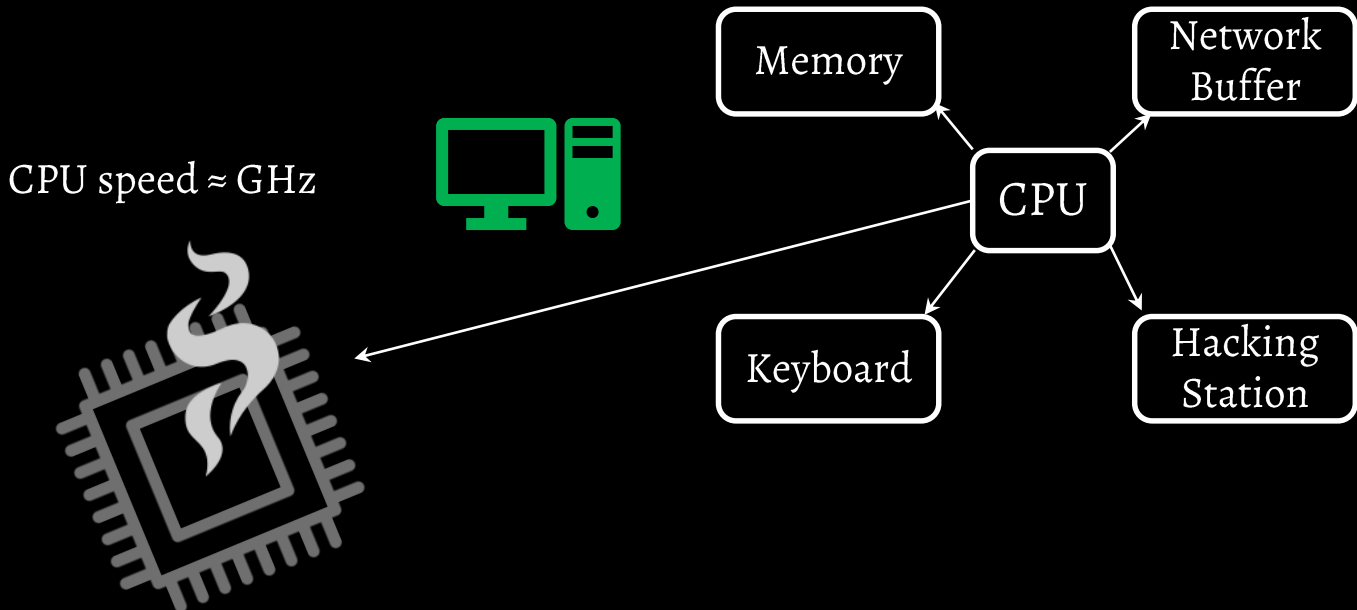


Ethernet connections run at 1Gbps commonly

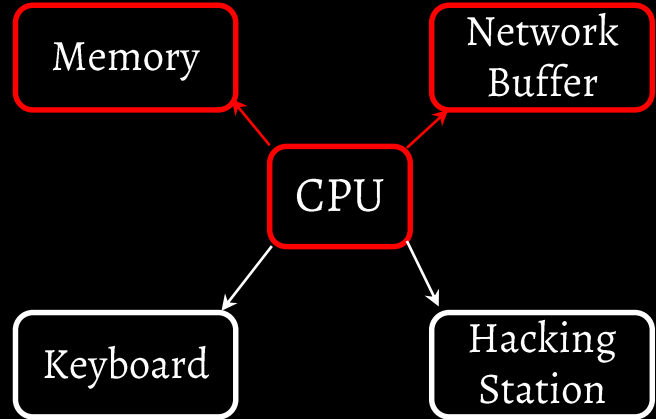
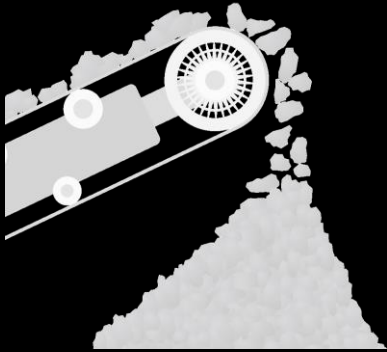
1 Byte every 8 ns



Von Neumann architecture



Von Neumann Bottleneck



FPGAs: a Non-Von Neumann class of devices



FPGAs: a Non-Von Neumann class of devices



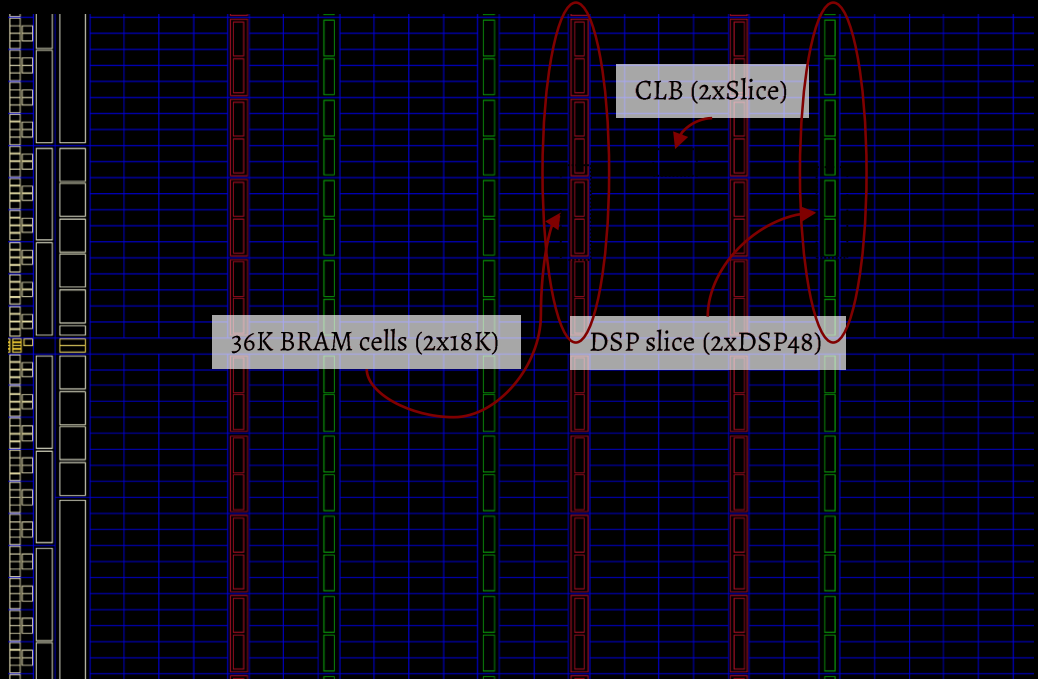
Non-Pipelined



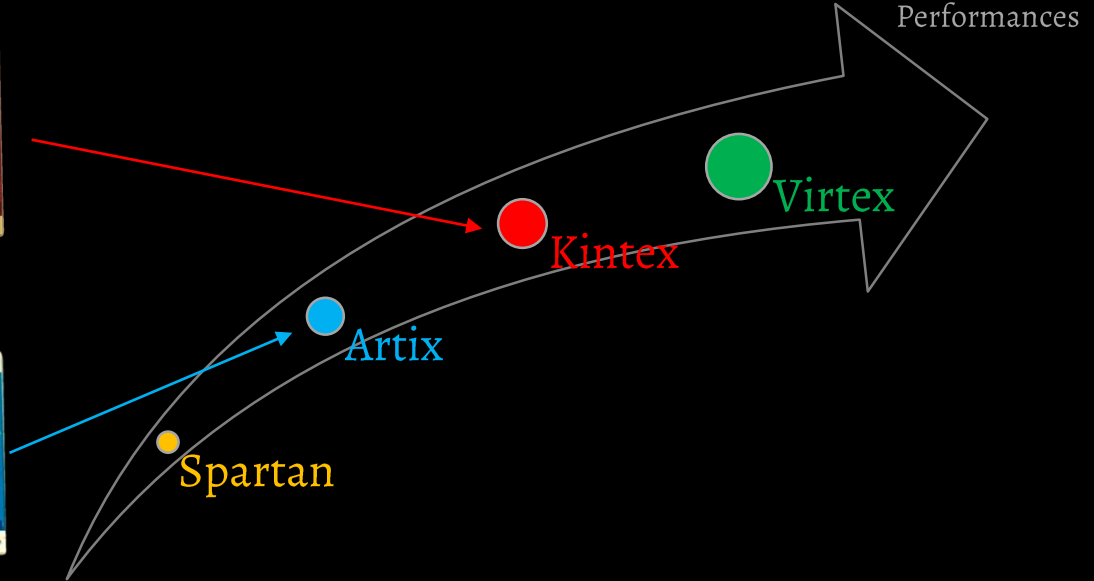
Pipelined



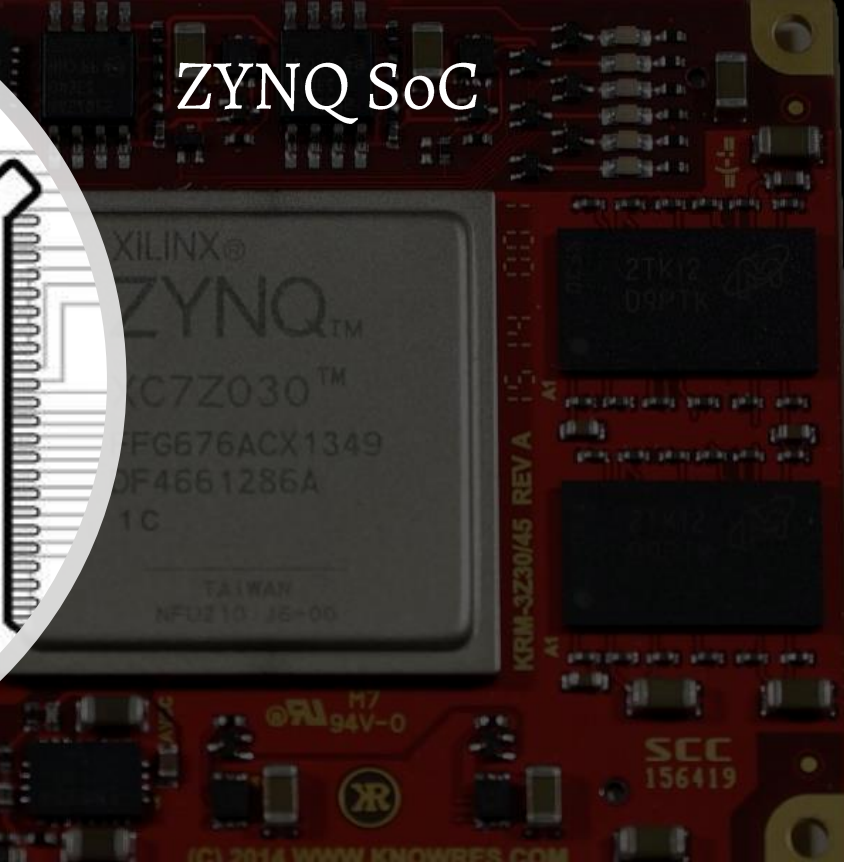
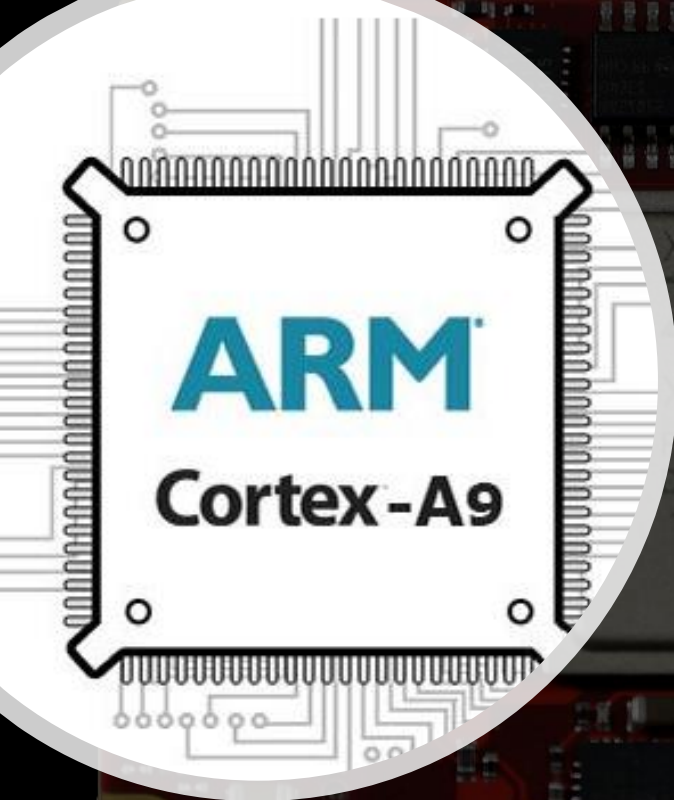
FPGAs: what's inside



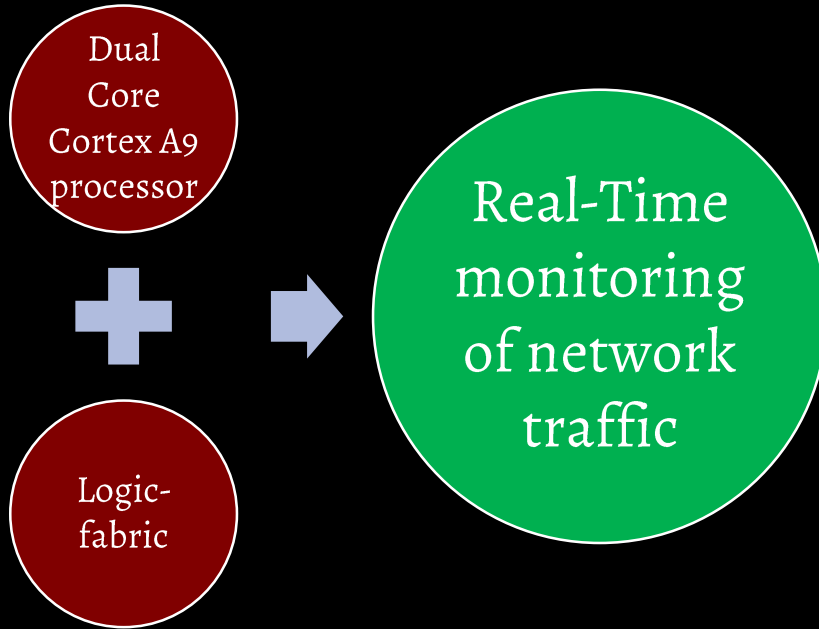
FPGAs: Xilinx fabric families



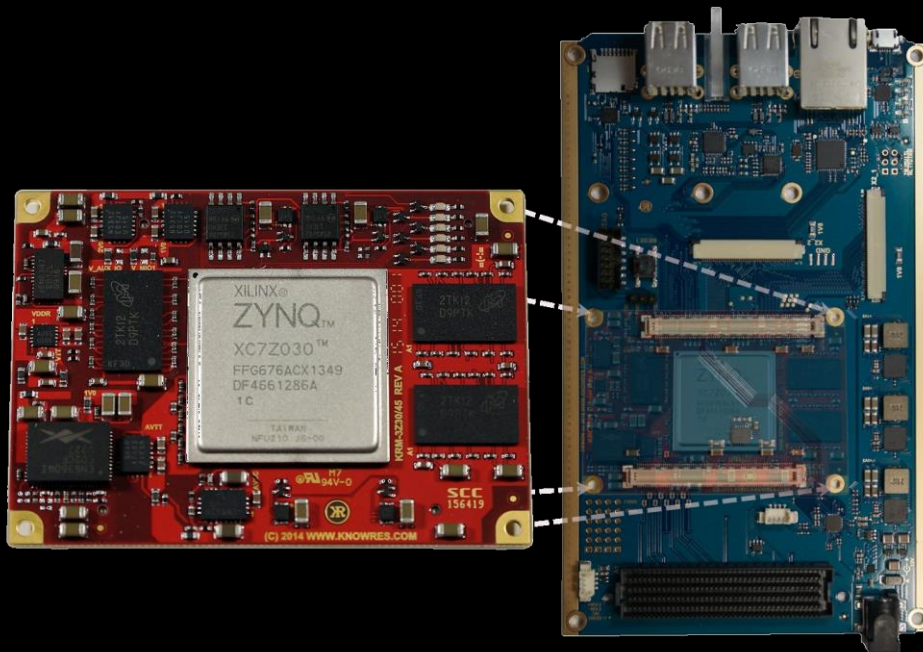
ZYNQ SoC



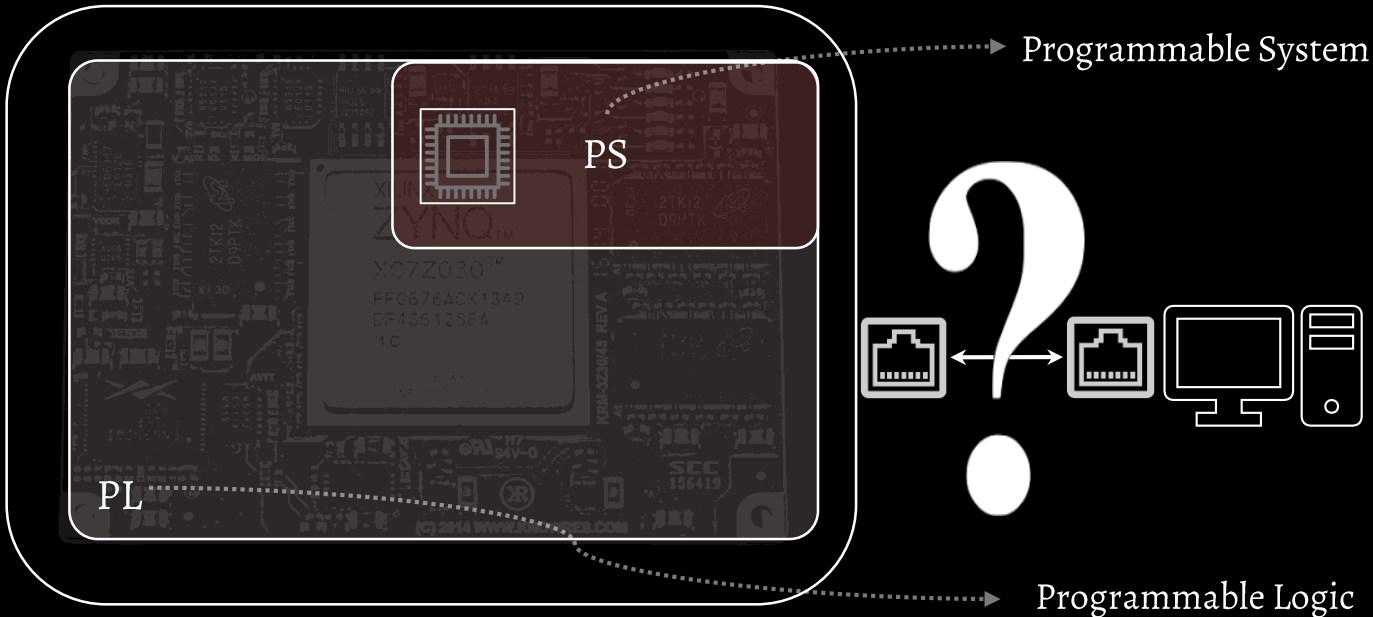
Summarizing



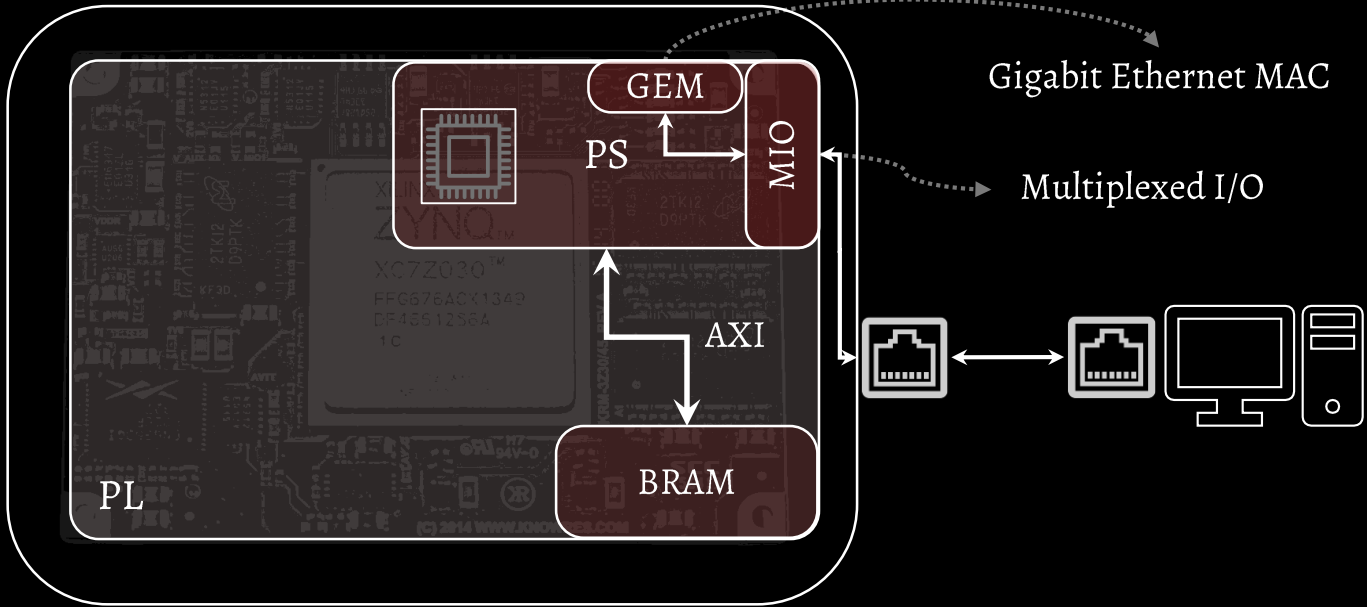
Carrier Evaluation Kit



ZYNQ architecture



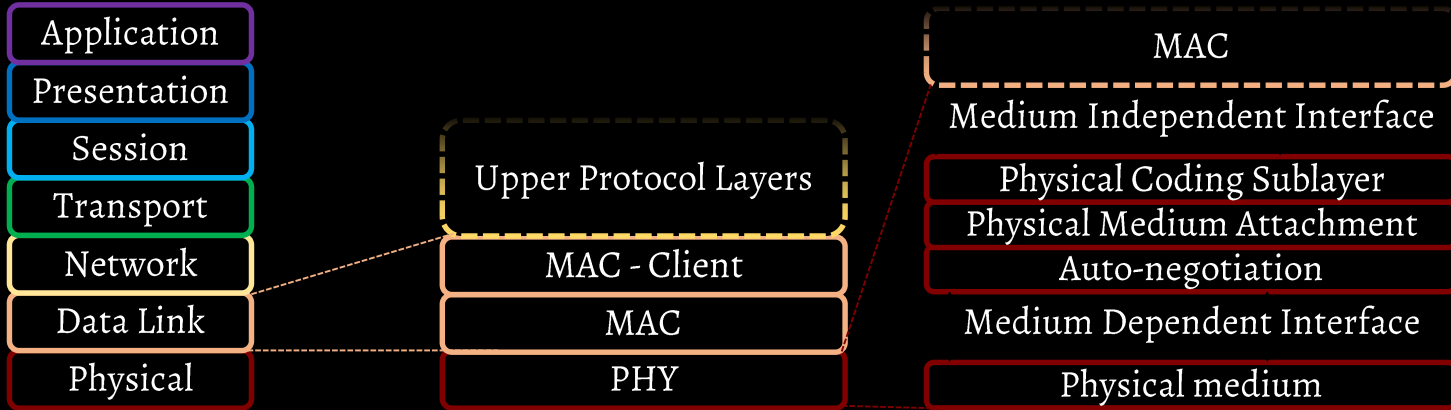
Routing Network Connections



ISO/OSI

OSI
reference model

IEEE 802.3
reference model



Useful libraries to implement network applications



LWIP (LightWeight IP)

- Good at Network+ (3+) OSI Level
- Dynamic allocation of memory

Too high
level of
abstraction

The
processor
is always
busy

Useful libraries to implement network applications

LWIP (Light Weight IP)

- Good at Network Level
- Dynamic allocation of memory

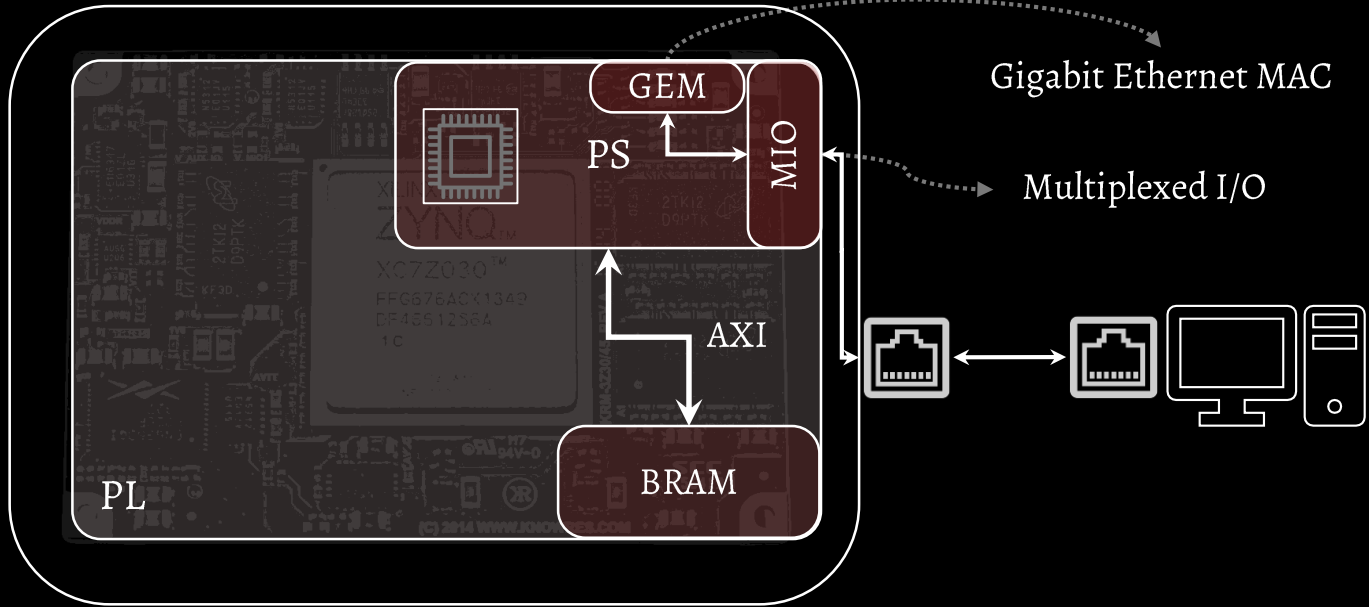
Captures ALL the packets

XEmacPS

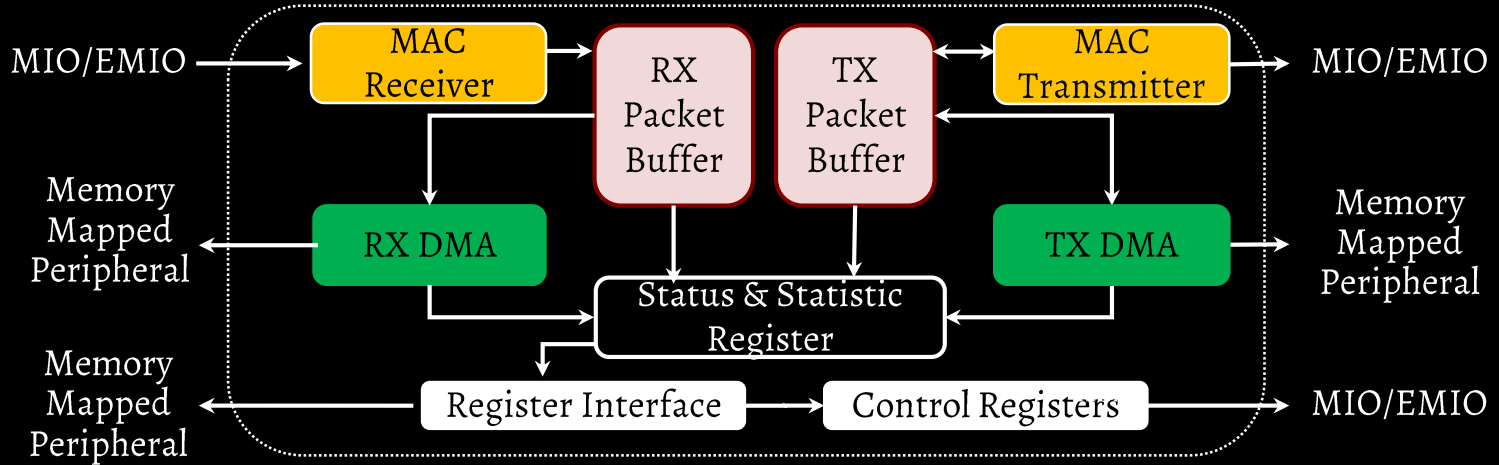
- Low level library, Data Link (MAC) OSI Level
- Interacts directly with GEM and DMA

A bit complicated

Routing Network Connections



Gigabit Ethernet MAC



Gigabit Ethernet MAC

RX/TX Buffer

Fills up quickly

Packet size is
overestimated but
well organized
structure

DMA Engine

4-state Finite
State Machine

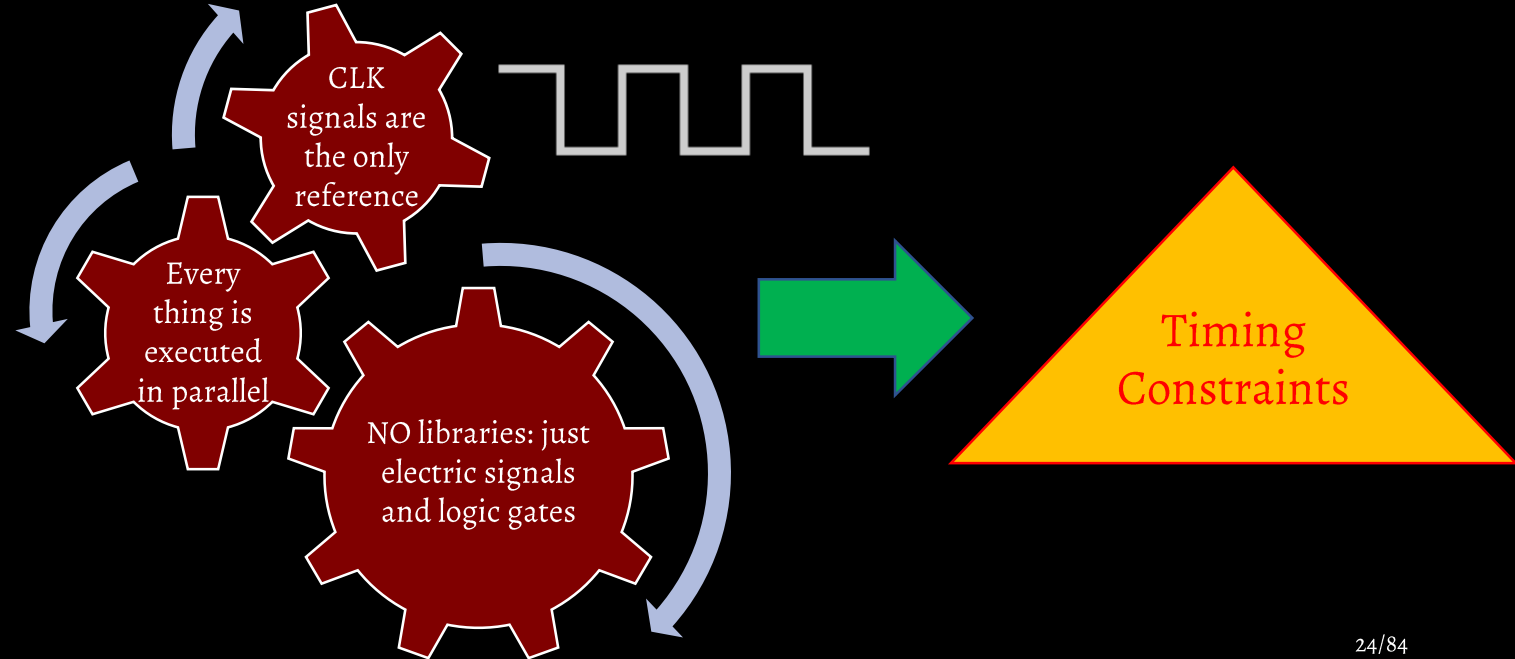
Driven by ARM
processor

BRAM

“Slow” (Hundreds
of MHz)

Too many blocks
will mess up the
timing
constraints.

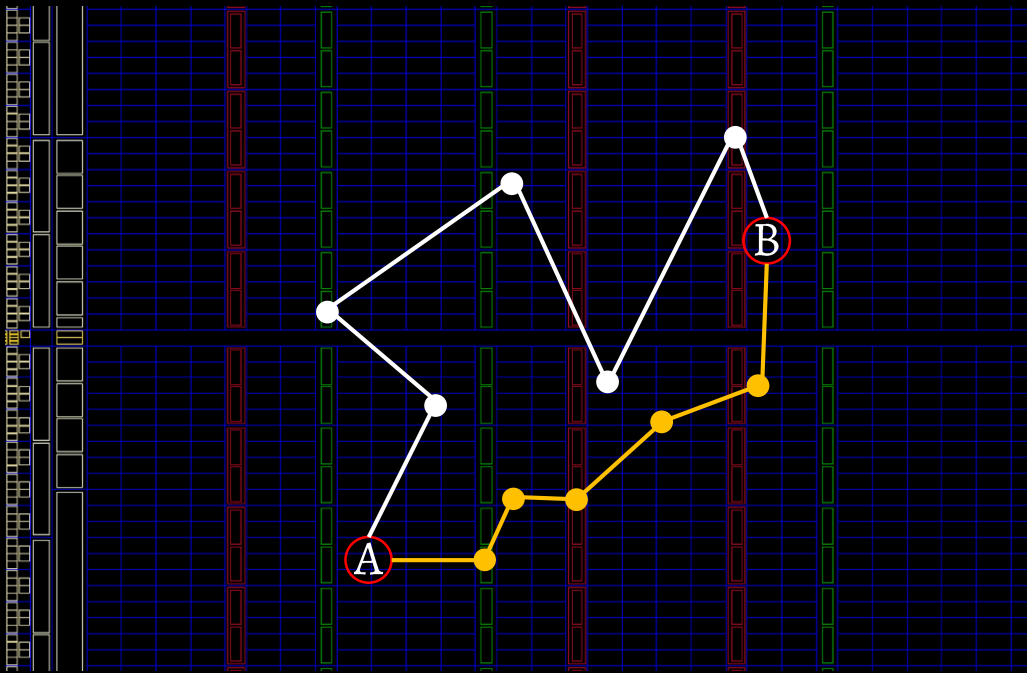
Towards a fully hardware implementation



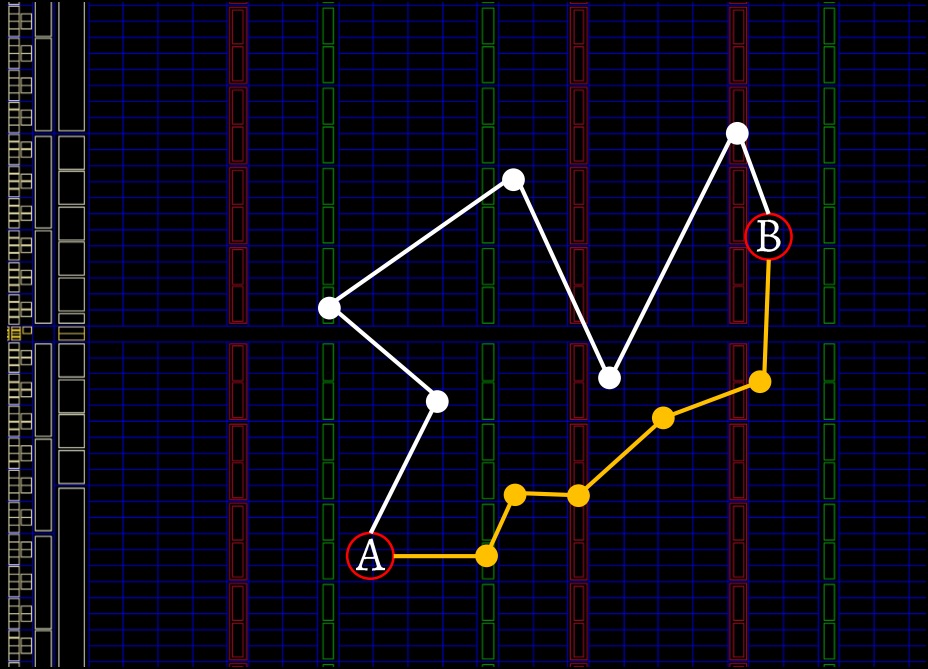
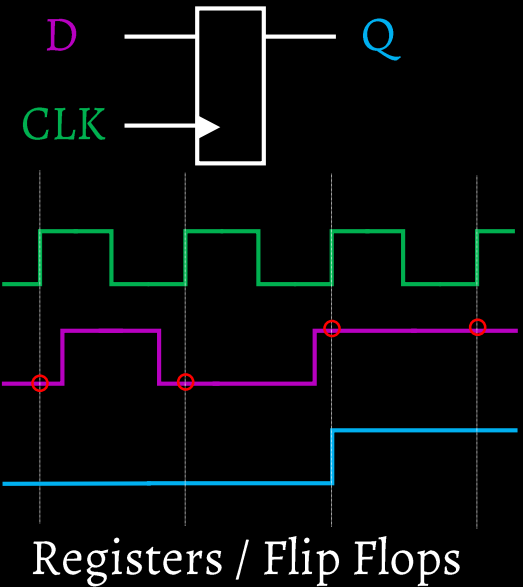
Timing Constraints



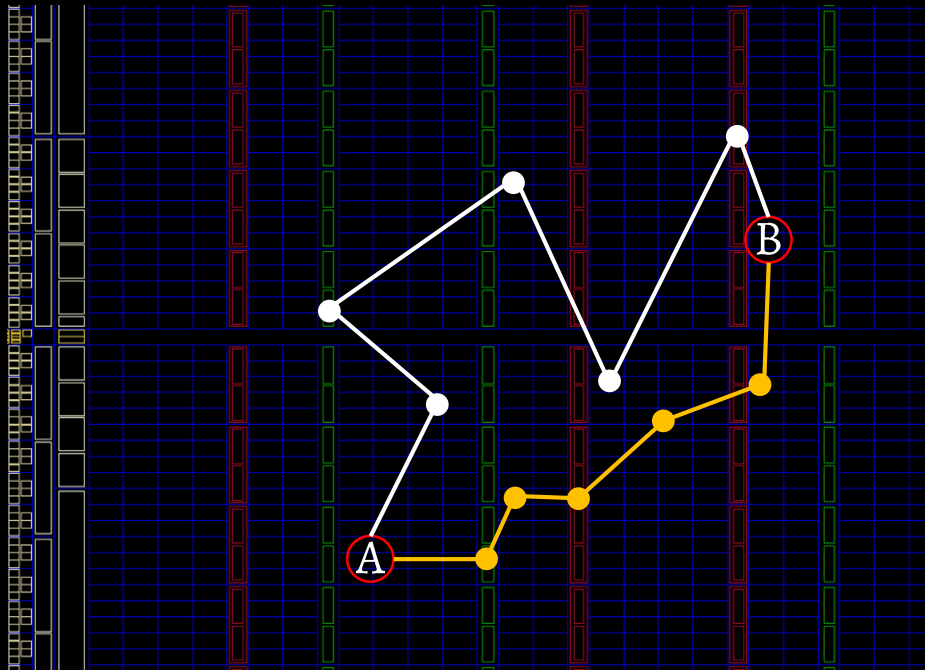
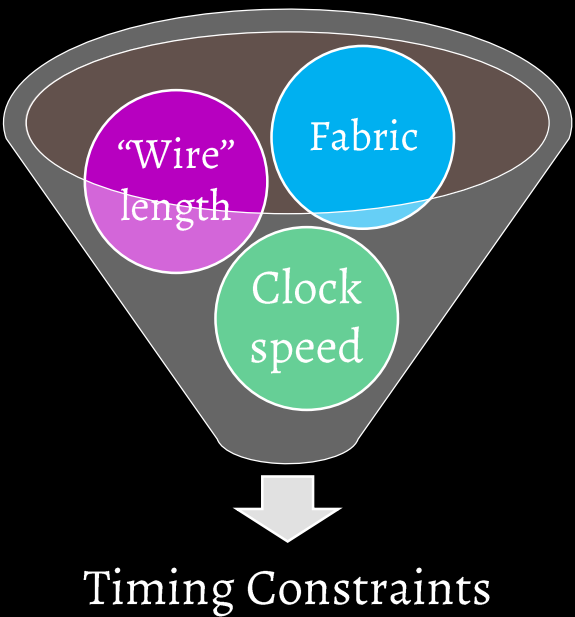
Do use memory waypoints wisely!



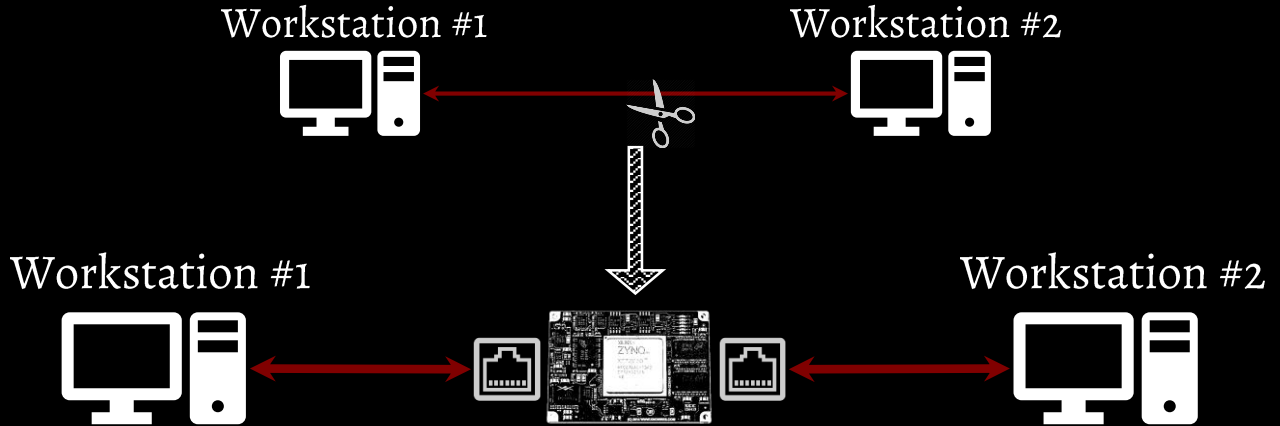
Timing Constraints



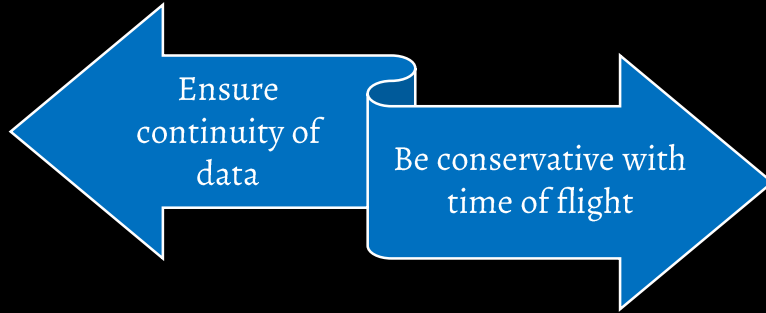
Timing Constraints – Dependencies



Primary goal: the Ethernet extension cord



Primary goal: the Ethernet extension cord



Workstation #1



Workstation #2

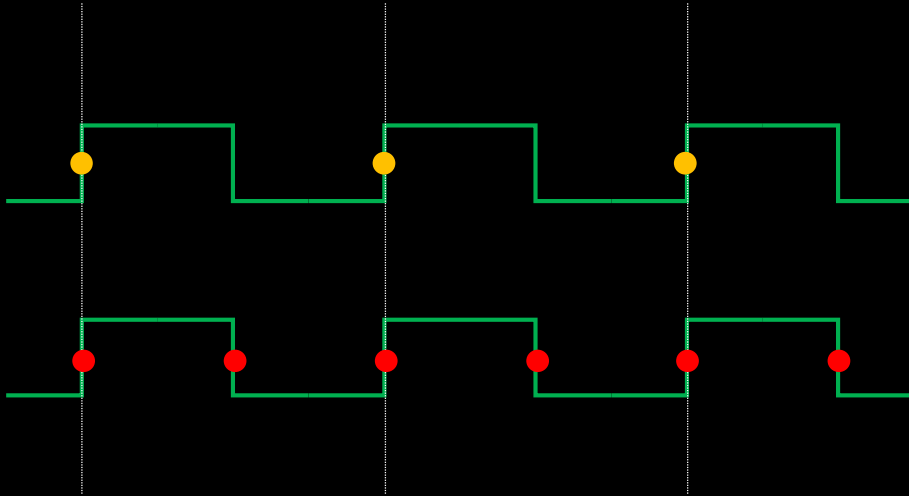


FPGA expansion modules



Ethernet at “ground” level

Single Data
Rate (SDR)

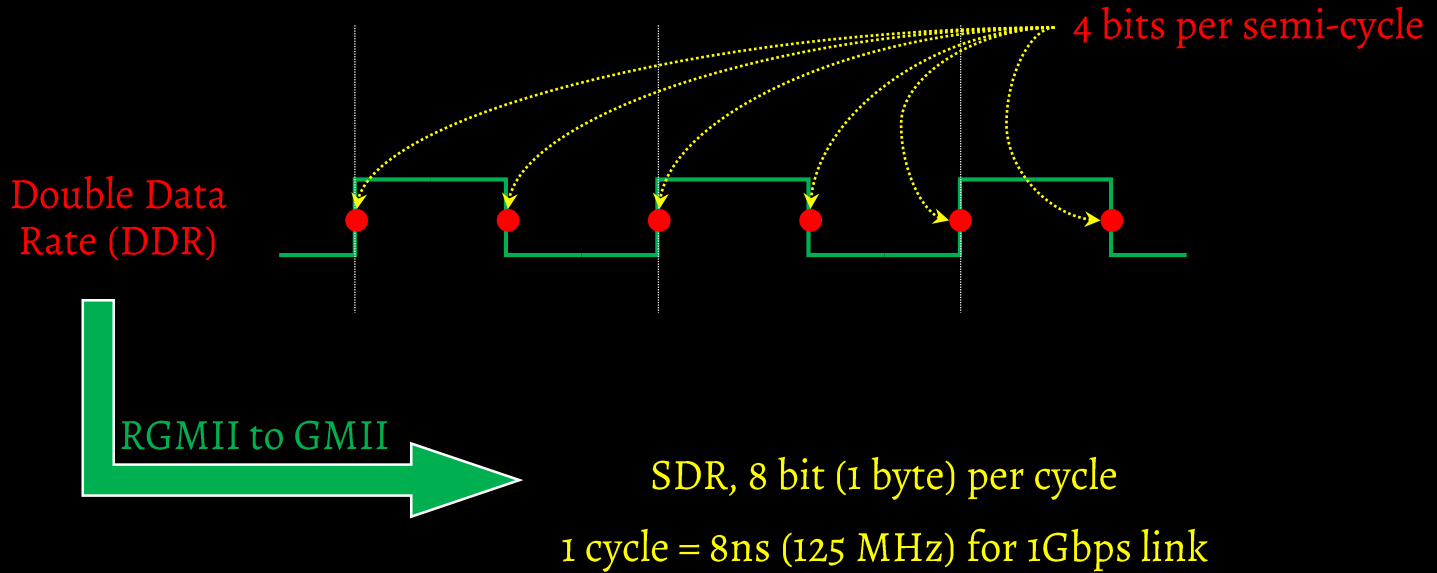


One signal
per clock
cycle

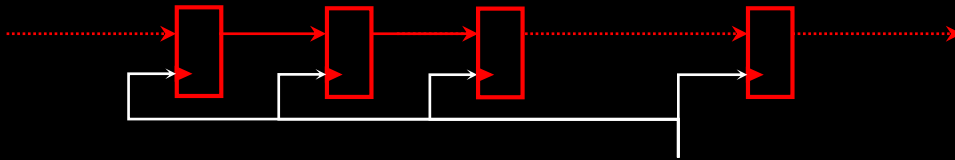
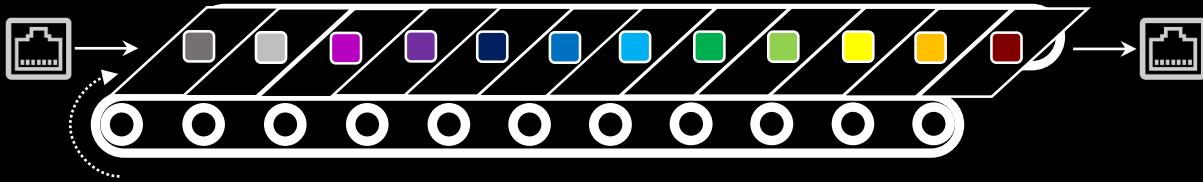
Double Data
Rate (DDR)

Two signals
per clock
cycle

Ethernet at “ground” level

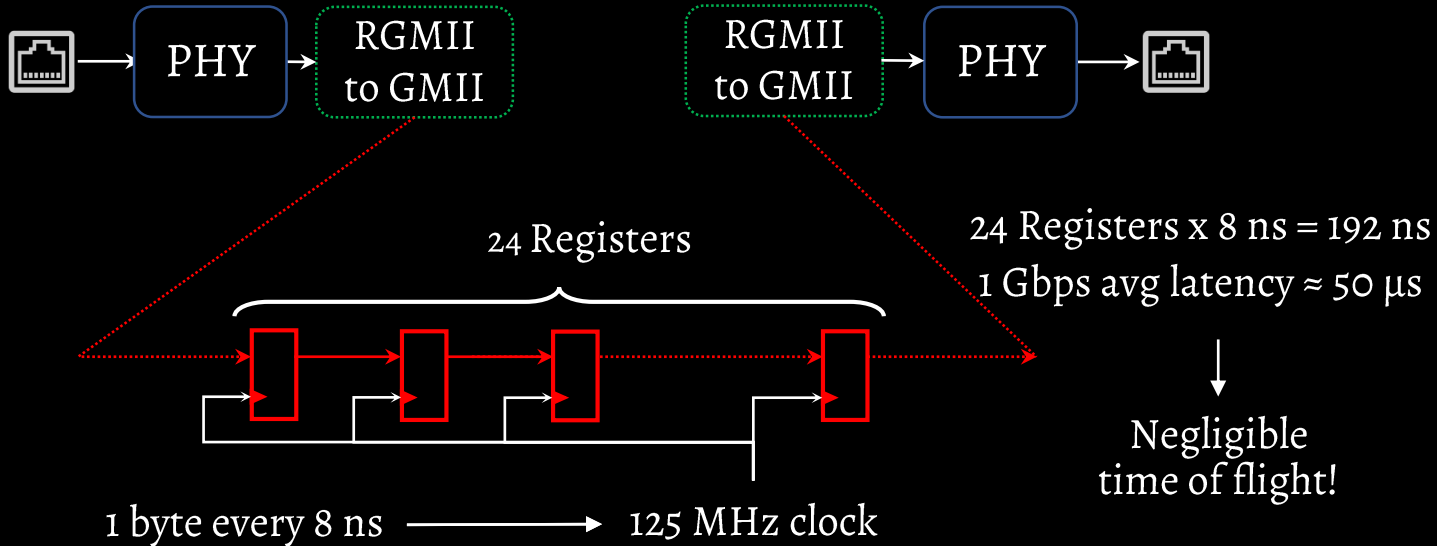


Conveyor belt model



1 byte every 8 ns 125 MHz clock

Conveyor belt model



So far so good! What's next?



Firewall-like
features

Deep packet
inspection

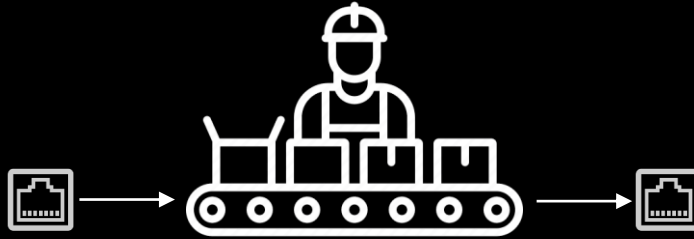
Encryption/
Decryption
system

Network Packets



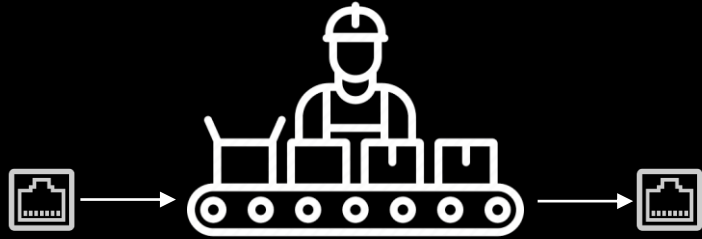
Information travels via
network packets according
to IEEE 802.3 standard

Assembly chain model

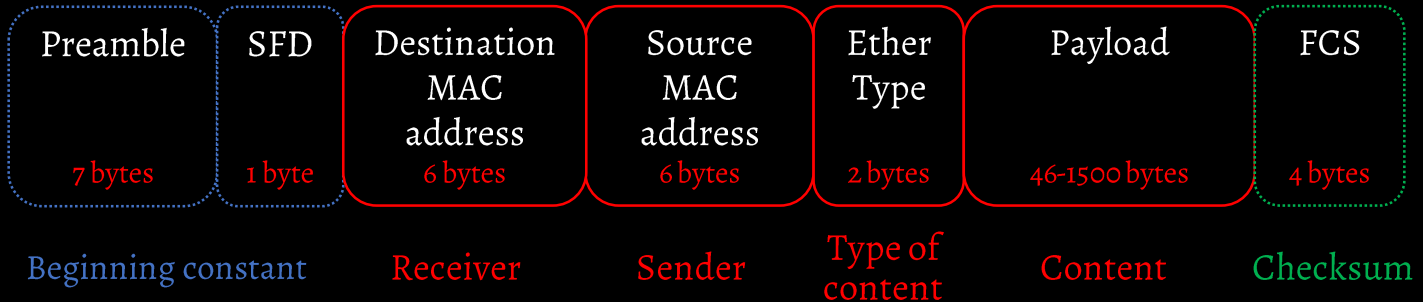


On-the-flight
manipulation of
packets

Ethernet Frame structure

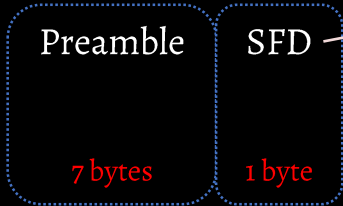


IEEE 802.3 standard



Preamble detector

- Constant pattern
"0101010101....1101"
→ 0x55.....0xD5
- Helps clock
synchronization



Data valid from GMII interface

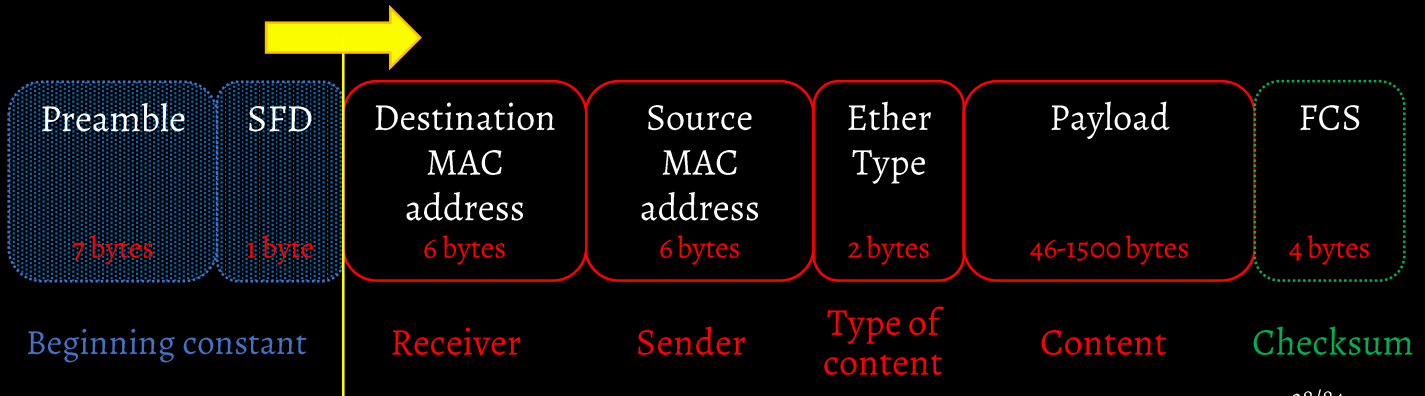
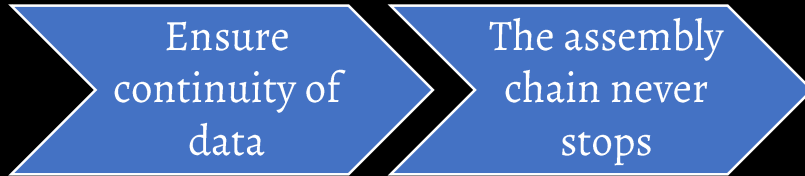
Initialization of CRC engine

Initialization of crypto engine

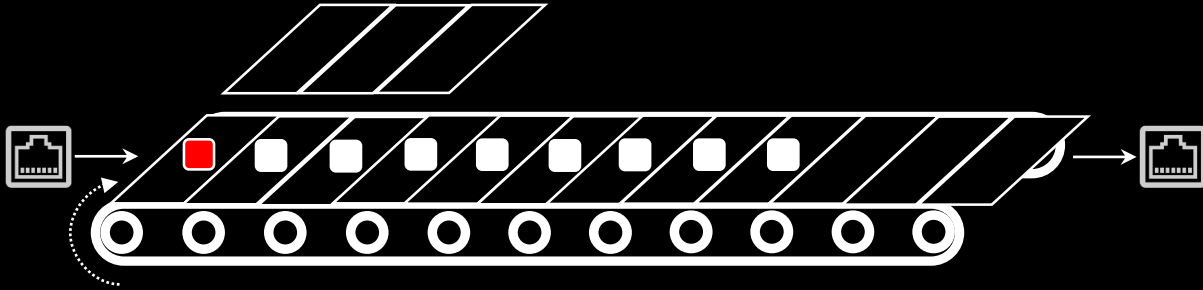
```
when IDLE =>
  IS_CURRENT_STATE_IDLE <= '1';
  if (DV_IN = '1') then
    counter_next <= "001";
    state_next <= PREAMBLE_DETECT;
    crc_rst <= '1';
  else
    state_next <= IDLE;
  end if;

when PREAMBLE_DETECT =>
  if (PIPELINE(SIZE_OF_PIPE-1) = "01010101") then
    if (counter_reg = "011") then
      if (D_IN = "11010101") then
        state_next <= PREAMBLE_OK;
        crypto_start <= '1';
      else
        counter_next <= (OTHERS => '0');
        state_next <= IDLE;
      end if;
    else
      state_next <= PREAMBLE_DETECT;
    end if;
  else
    counter_next <= (OTHERS => '0');
    state_next <= IDLE;
  end if;
```

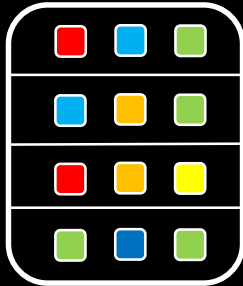
MAC and EtherType filters



MAC and EtherType filters



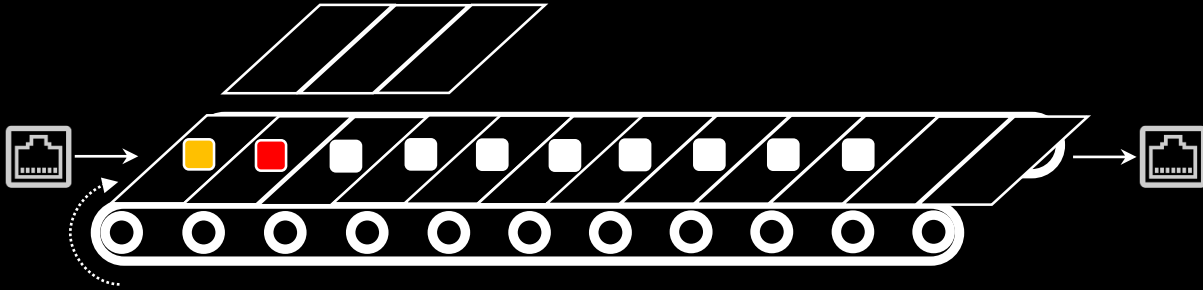
Blacklist



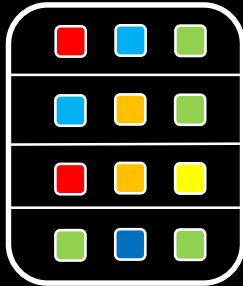
Search algorithm progress:

0%

MAC and EtherType filters



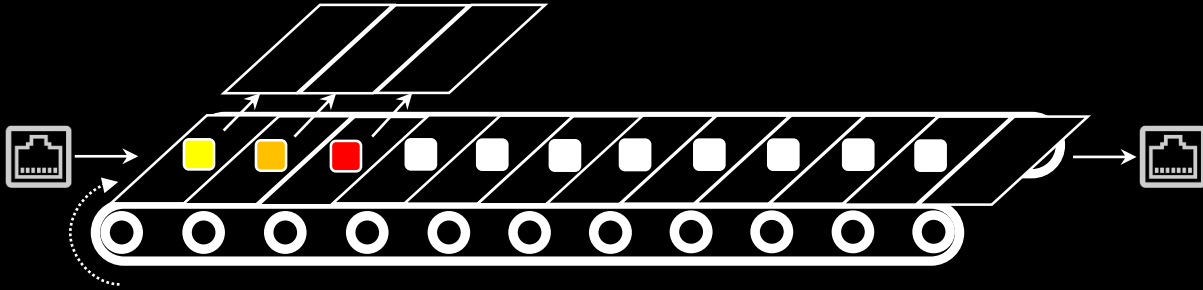
Blacklist



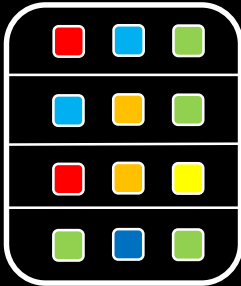
Search algorithm progress:

0%

MAC and EtherType filters



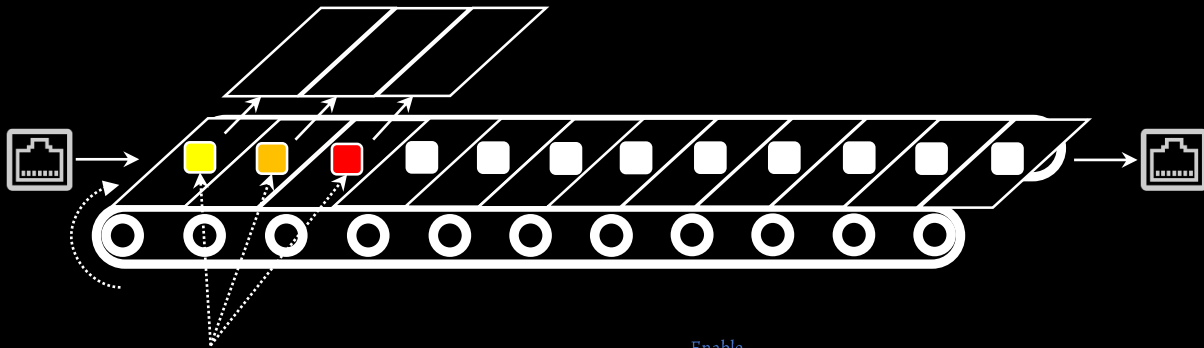
Blacklist



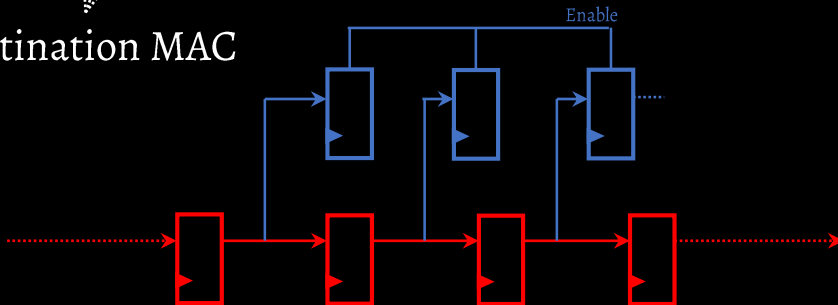
Search algorithm progress:

0%

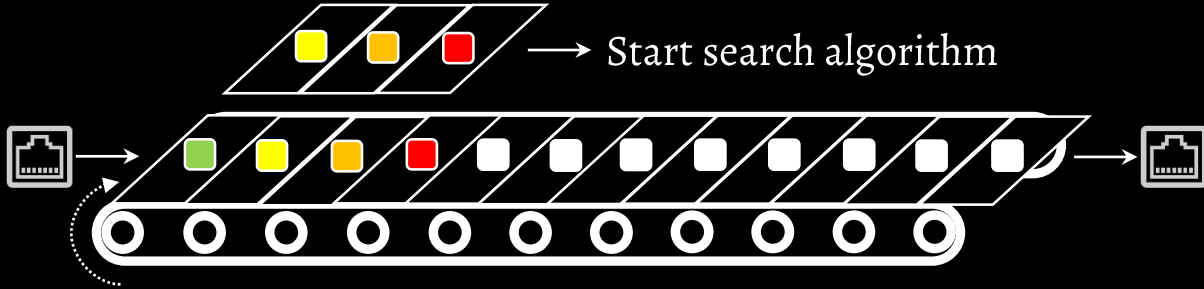
MAC and EtherType filters



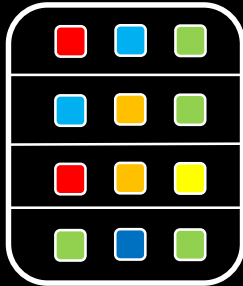
Destination MAC



MAC and EtherType filters



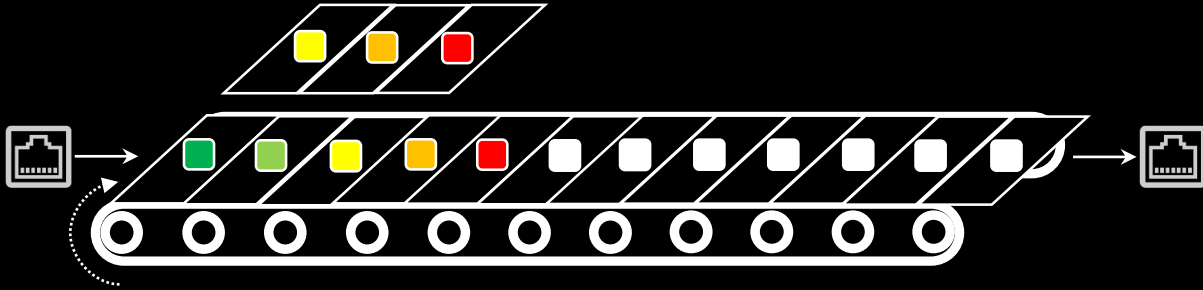
Blacklist



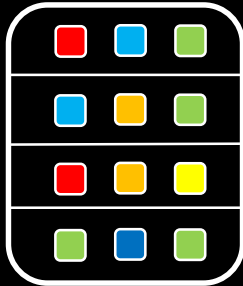
Search algorithm progress:

20%

MAC and EtherType filters



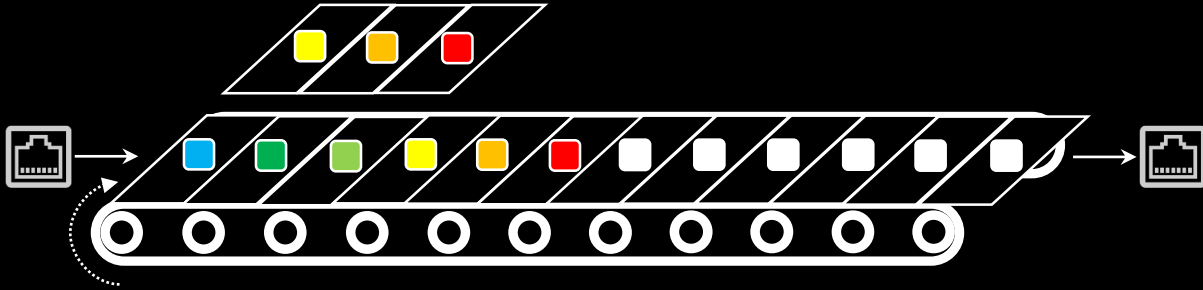
Blacklist



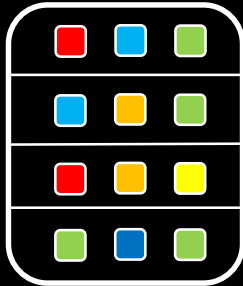
Search algorithm progress:

40%

MAC and EtherType filters

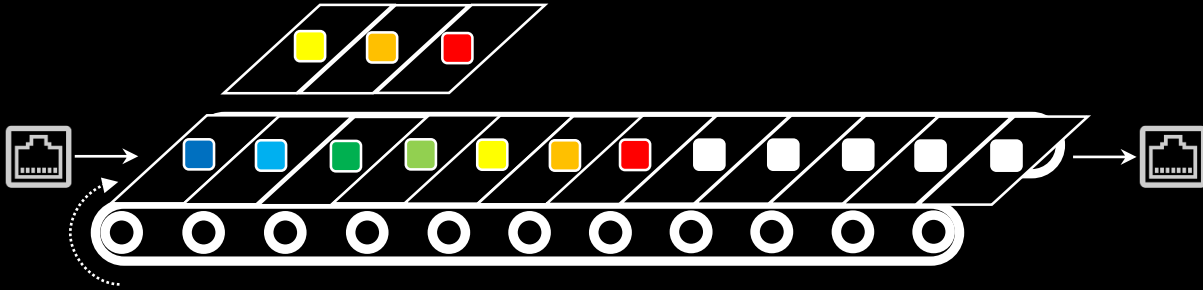


Blacklist

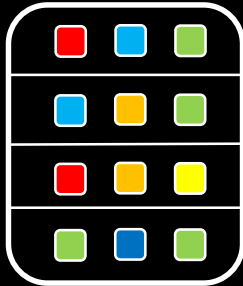


Search algorithm progress:
60%

MAC and EtherType filters

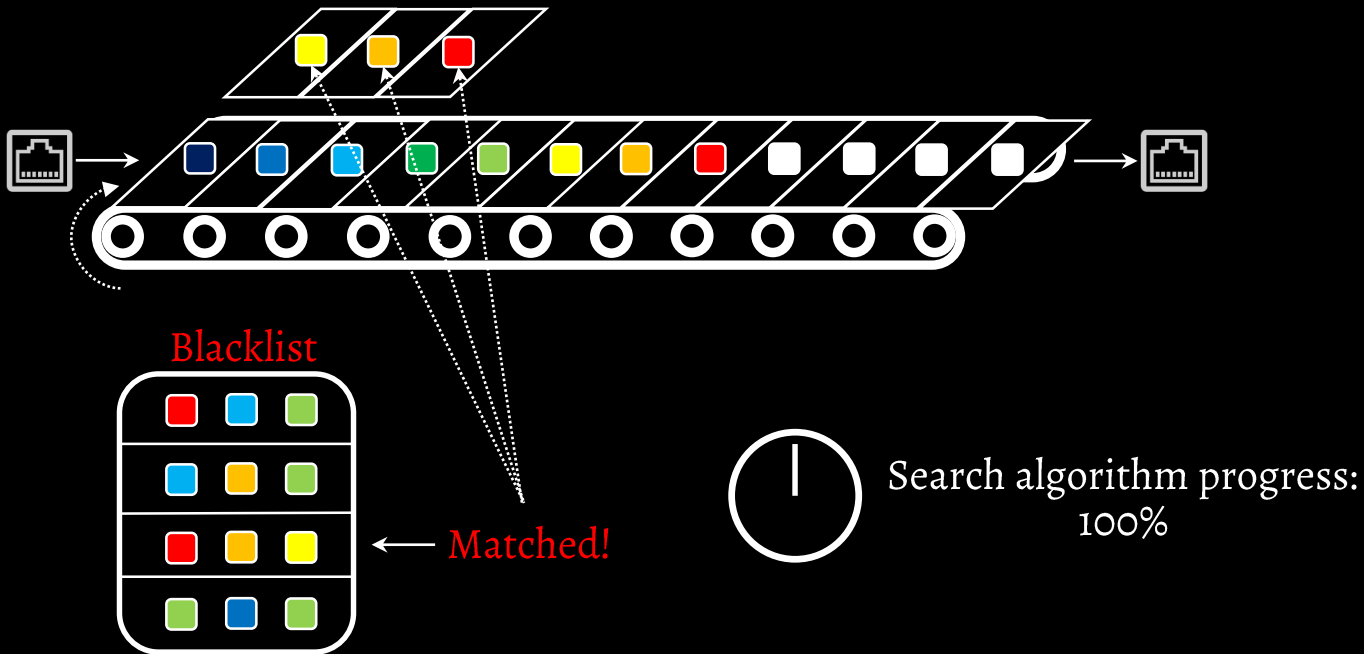


Blacklist

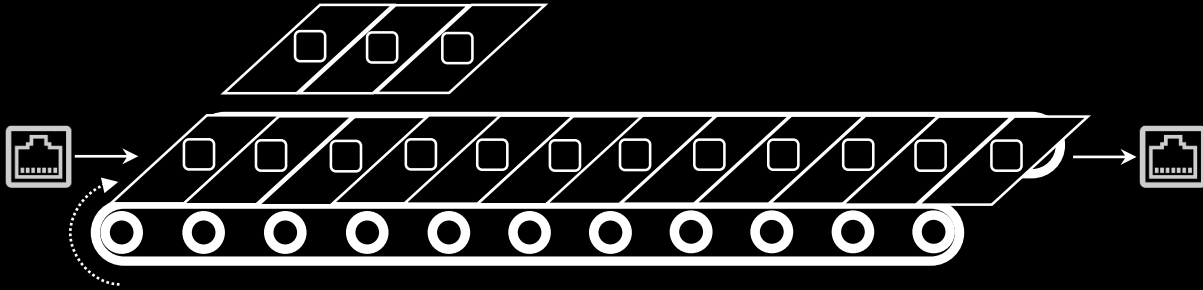


Search algorithm progress:
80%

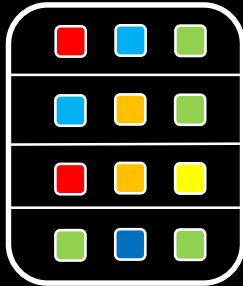
MAC and EtherType filters



MAC and EtherType filters



Blacklist



Matched!

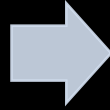
Drop the whole frame



MAC and EtherType filters – Preparation

Smart Lookup in memory

- Arrange the Blacklist properly
- Dicotomic (Tricotomic) search



Ascending order pre-sort

- Handled by the CPU

MAC Address Library

1. 00:01:03:02:30:22
2. 02:23:43:C2:B4:FF
3. Co:CA:Co:1A:00:00
4. FF:FF:FF:FF:FF:AC

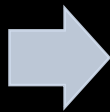


Qsort is useful!

MAC and EtherType filters

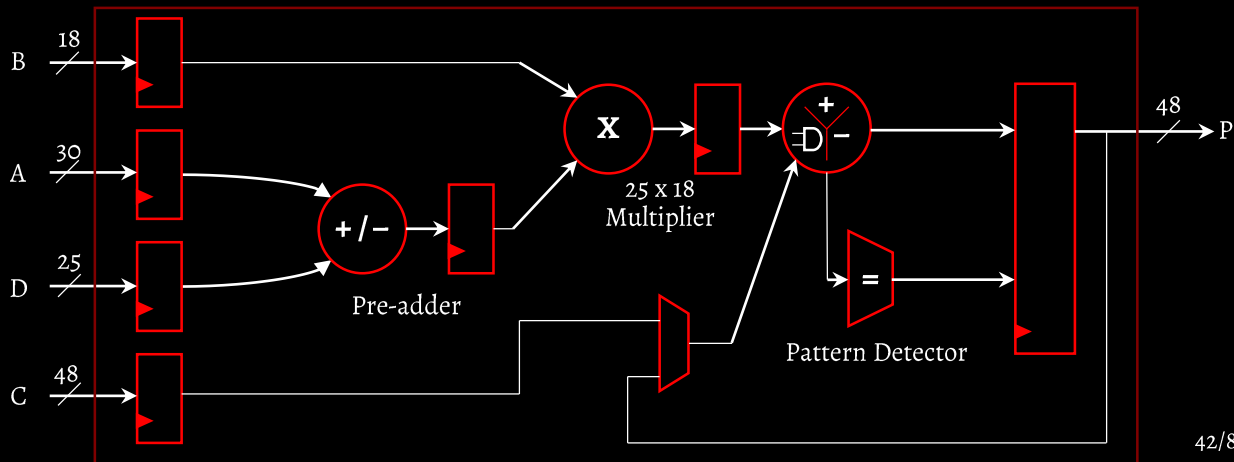
Fetch and Match

- The incoming MAC is compared with the blacklist entries

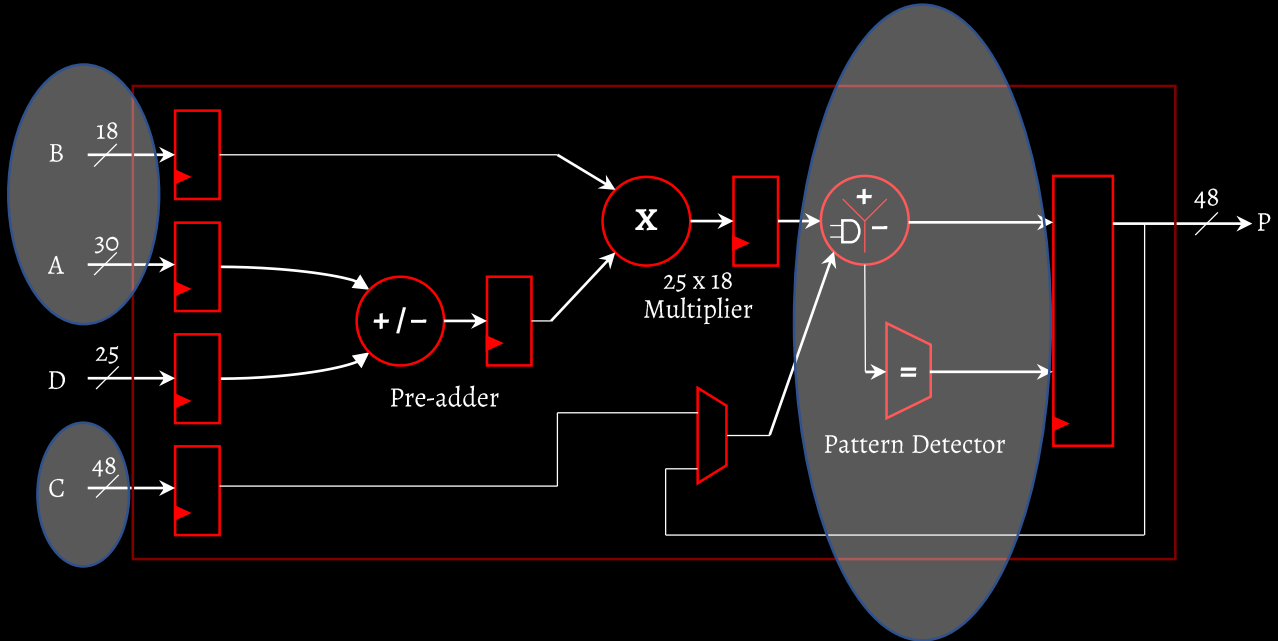


DSP makes fast comparisons

- Up to 48 bit entries



DSP48



MAC and EtherType filters

Smart Lookup in memory

- Arrange the Blacklist properly
- Dicotomic (Tricotomic) search

Fetch and Match

- The incoming MAC is compared with the blacklist entries

Ascending order pre-sort

- Handled by the CPU

DSP makes fast comparisons

- Up to 48 bit entries

Choose wisely the variable size

MAC address

00:10:FA:CC:Co:1A

MAC address – 48 bits

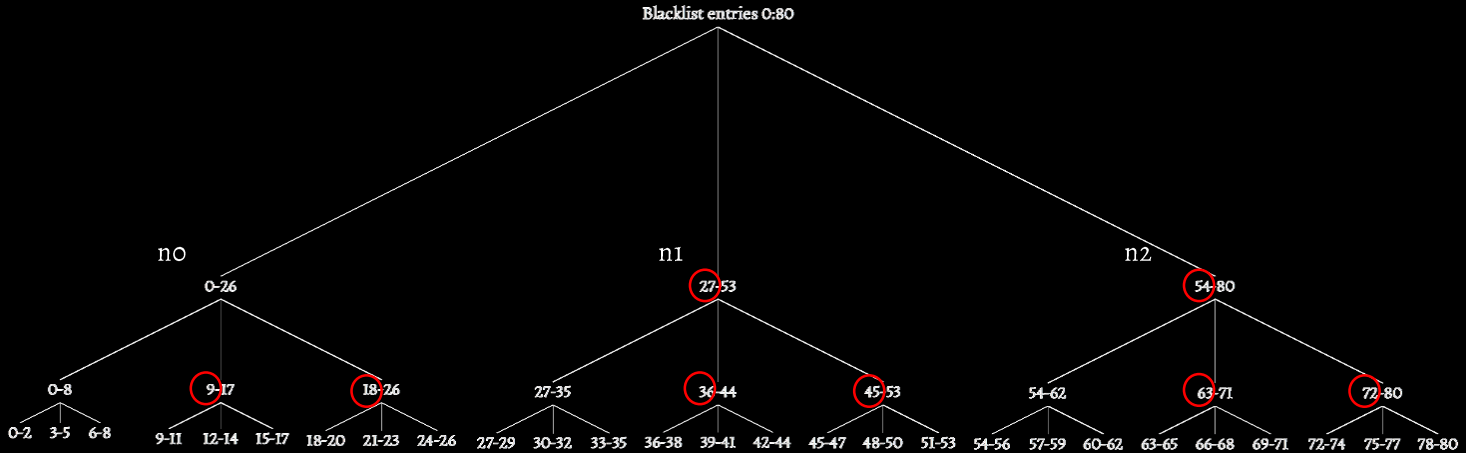
Organizationally Unique Identifier (OUI) – 24 bits

Do save time!

Compare 24-bit
operands

DSP will evaluate
2 comparisons
per time

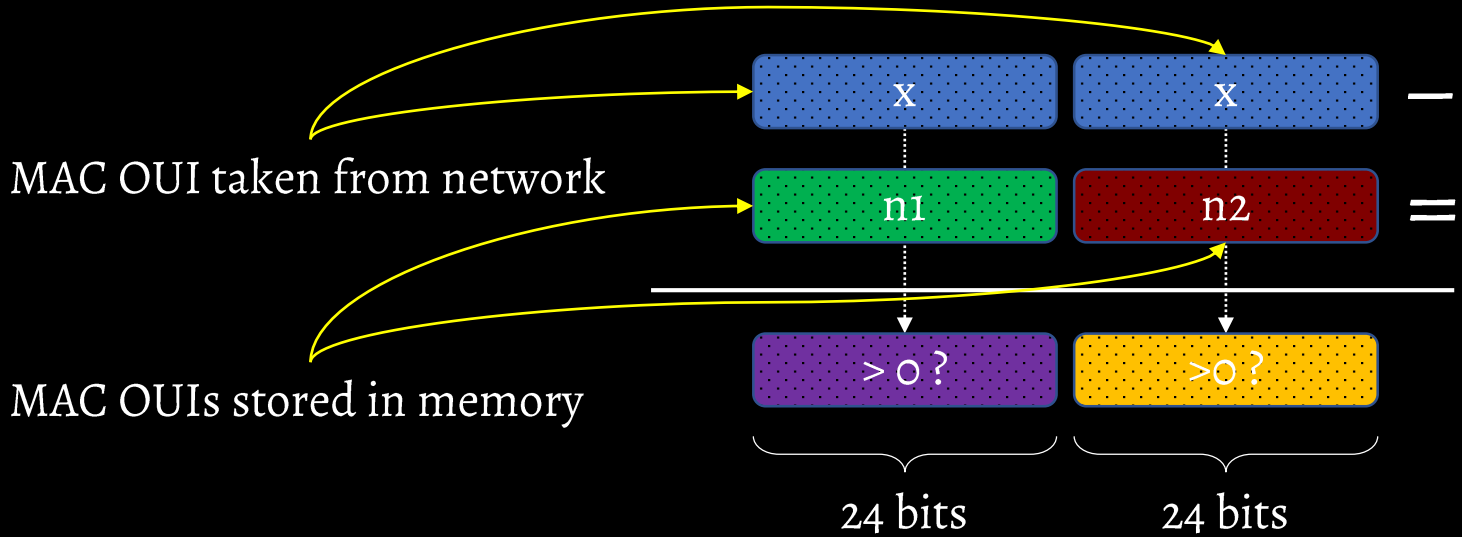
Search Algorithm version 1 – tricotomic



$$x > n1 ? \rightarrow x - n1 > 0$$

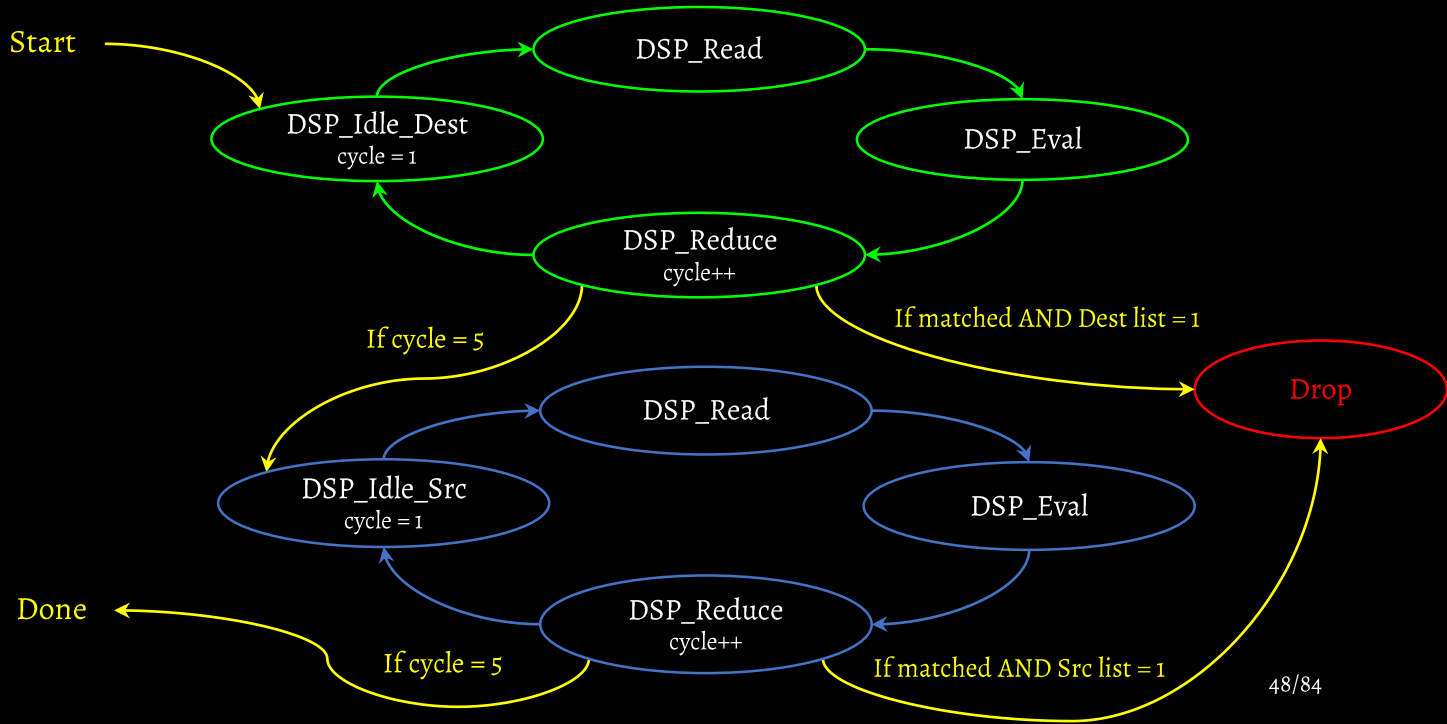
$$x > n2 ? \rightarrow x - n2 > 0$$

Search Algorithm version 1 – tricotomic



$$\begin{aligned}x > n1 ? &\rightarrow x - n1 > 0 \\x > n2 ? &\rightarrow x - n2 > 0\end{aligned}$$

MAC filter state machine



MAC and EtherType filters configuration

Set up MAC filter for Source and Destination address:

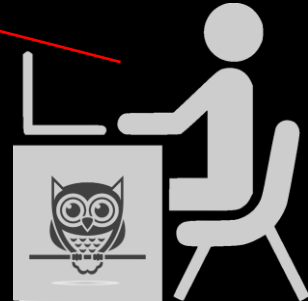
```
Source :      Packets coming from the chosen MAC are flushed
Destination : Packets directed towards the chosen MAC are flushed
Would you like to add all the inserted MACs to:
(1) Source blacklist
(2) Destination blacklist
(3) Both Source and Destination blacklist
(4) I would like to choose one by one
```

Please type per each MAC:

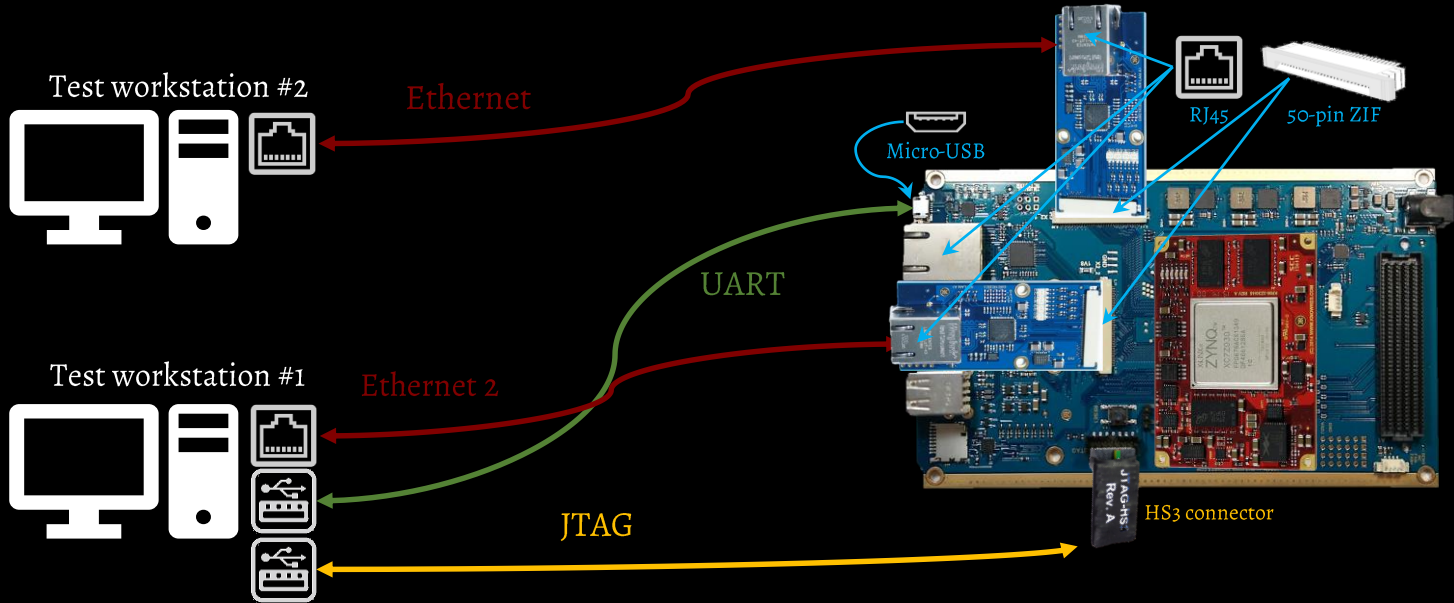
```
(1) --> Source blacklist
(2) --> Destination blacklist
(3) --> Both Source and Destination blacklist
```

```
0 | 00:0A:35:XX:XX:XX DESTINATION
1 | 0A:0A:0A:XX:XX:XX SOURCE
2 | 12:23:45:XX:XX:XX SOURCE & DESTINATION
3 | 12:31:23:XX:XX:XX █
```

- Up to 81 entries → 3^4
- Source and/or Destination MAC attribute is defined per each entry



Demo : MAC filtering



Demo : MAC filtering

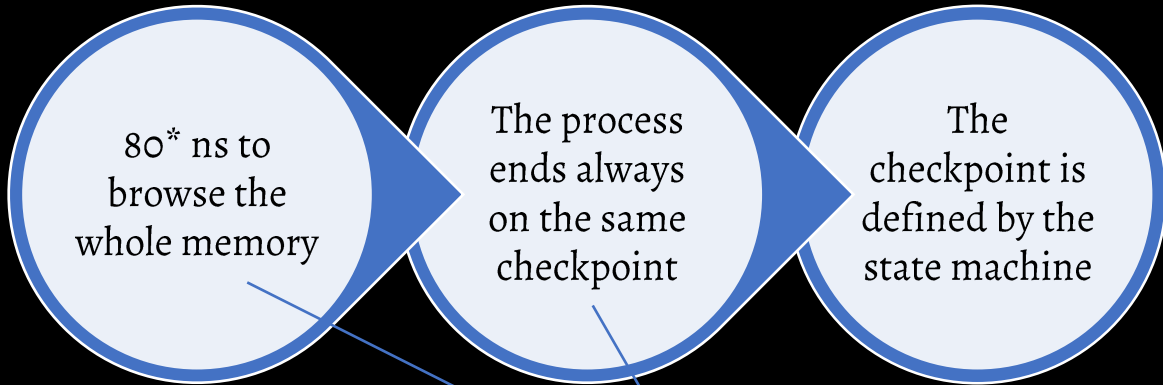


Demo: MAC filtering

The image displays two side-by-side Wireshark network capture windows. The left window, titled 'Capturing from Ethernet', shows a list of network packets. The right window, also titled 'Capturing from Ethernet 2', shows a similar list of packets. Red arrows originate from specific entries in the left window and point to corresponding entries in the right window, illustrating the results of MAC filtering. The arrows point from packets 117, 118, 119, 120, 121, 122, 123, 124, and 125 in the left window to packets 216, 217, 218, 219, 220, 221, 222, 223, and 224 in the right window, respectively. These packets are all of type DHCP and involve the Xilinx_01:02:03 MAC address.

No.	Time	Source	Destination	Protocol	Length	Info
86	140.194082	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
87	140.252488	169.254.11.54	239.255.255.250	UDP	698	519
88	140.548464	169.254.11.54	239.255.255.250	UDP	698	519
89	140.565119	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
90	140.570102	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
91	140.745132	169.254.11.54	239.255.255.250	UDP	698	519
92	141.070370	169.254.11.54	169.254.255.255	BROWSER	219	Req
93	141.268164	169.254.11.54	239.255.255.250	UDP	698	519
94	141.309126	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
95	141.323095	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
96	141.729131	169.254.11.54	239.255.255.250	UDP	698	519
97	142.289188	169.254.11.54	224.0.0.251	MDNS	496	Sta
98	142.570065	169.254.11.54	169.254.255.255	BROWSER	219	Req
99	142.708154	169.254.11.54	239.255.255.250	UDP	698	519
100	142.797103	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
101	142.827093	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
102	143.697162	169.254.11.54	239.255.255.250	UDP	698	519
103	144.708465	169.254.11.54	239.255.255.250	UDP	698	519
104	144.797125	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
105	144.827114	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
106	144.996016	fe80::1c1a:f420:3ab...ff02::1:2		DHCPv6	158	Sol
107	145.697177	169.254.11.54	239.255.255.250	UDP	698	519
108	146.708474	169.254.11.54	239.255.255.250	UDP	698	519
109	146.798122	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
110	146.828448	fe80::1c1a:f420:3ab...ff02::c		UDP	718	519
111	146.927108	0.0.0.0	255.255.255.255	DHCP	342	DHC
112	147.209223	169.254.11.54	224.0.0.251	MDNS	112	Sta
113	147.697135	169.254.11.54	239.255.255.250	UDP	698	519
114	150.290163	169.254.11.54	224.0.0.251	MDNS	496	Sta
115	152.791628	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	502	Bo
116	153.791634	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	814	Bo
117	154.791637	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	1089	Bo
118	155.791637	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	261	Bo
119	156.791632	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	270	Bo
120	157.791637	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	178	Bo
121	158.791636	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	327	Bo
122	159.791640	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	1487	Bo
123	160.791639	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	394	Bo
124	160.996041	fe80::1c1a:f420:3ab...ff02::1:2		DHCPv6	158	S
125	161.791640	c0:ca:c0:1a:00:00	Xilinx_01:02:03	IPv4	671	Bo
126	163.164098	0.0.0.0	255.255.255.255	DHCP	342	DHC
127	166.287222	169.254.11.54	224.0.0.251	MDNS	496	Sta
128	174.274345	169.254.11.54	224.0.0.251	MDNS	112	Sta
129	192.996291	fe80::1c1a:f420:3ab...ff02::1:2		DHCPv6	158	Sol
130	195.162552	0.0.0.0	255.255.255.255	DHCP	342	DHC
131	198.286454	169.254.11.54	224.0.0.251	MDNS	467	Sta
132	199.154122	0.0.0.0	255.255.255.255	DHCP	342	DHC

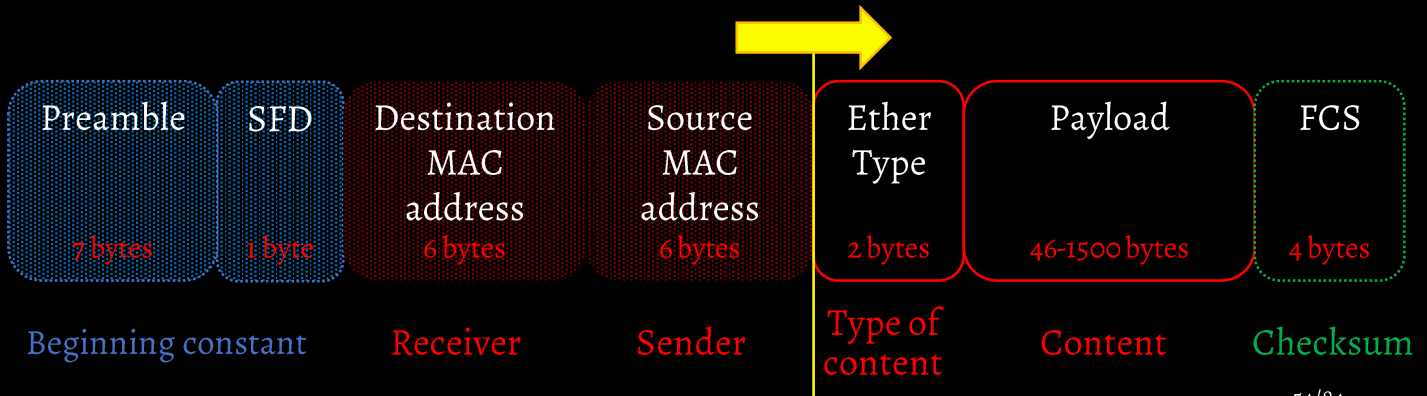
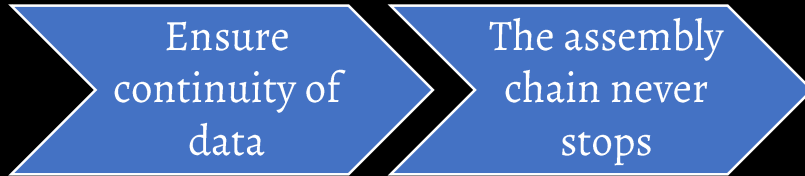
MAC filter conclusions



* IMPROVABLE TO 64 ns

80 ns = 10 bytes * 8 ns

EtherType filter



EtherType filter configuration

CURRENT LIST:

```
# | EtherType_ID
-----
0 | 0x0800
```

Do you want to add a new EtherType ID to your filter?
You still have 29 entries left

(y)es, please (n)o, that's enough for today

Ok, fine.

Set up EtherType filter to work as a Whitelist or Blacklist

Whitelist : Only the packets with matching EtherTypes are accepted

Blacklist : Packets with matching EtherTypes are flushed

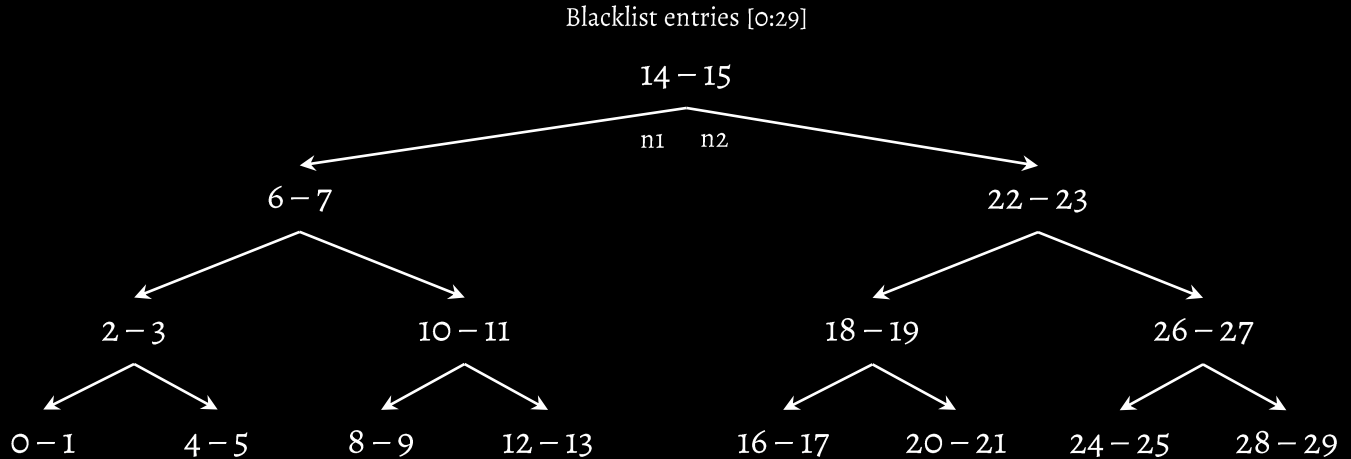
Would you like to set up a Blacklist or a Whitelist?

(1) Blacklist
(2) Whitelist

- Up to 30 entries
- Blacklist/Whitelist mode



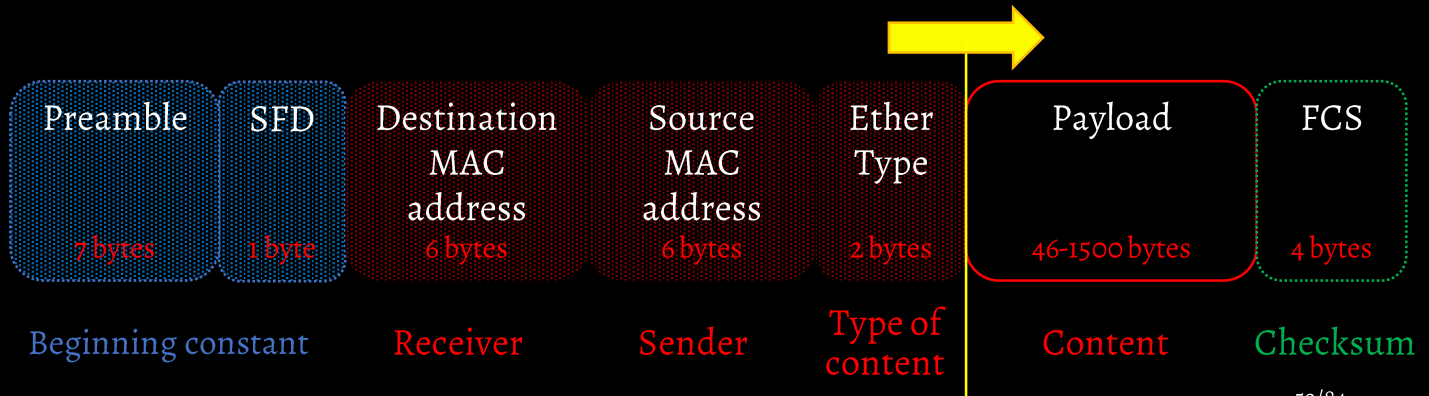
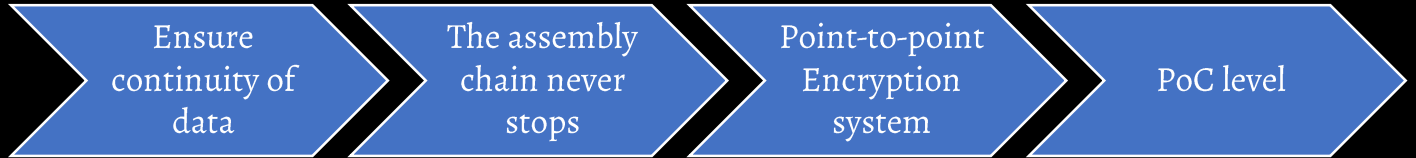
Search Algorithm version 2



$x > n1 ? \rightarrow x - n1 > 0$

$x > n2 ? \rightarrow x - n2 > 0$

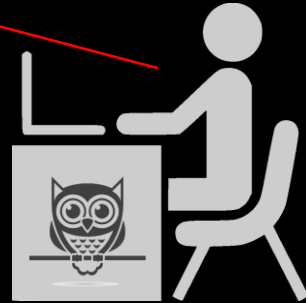
Encryption Environment



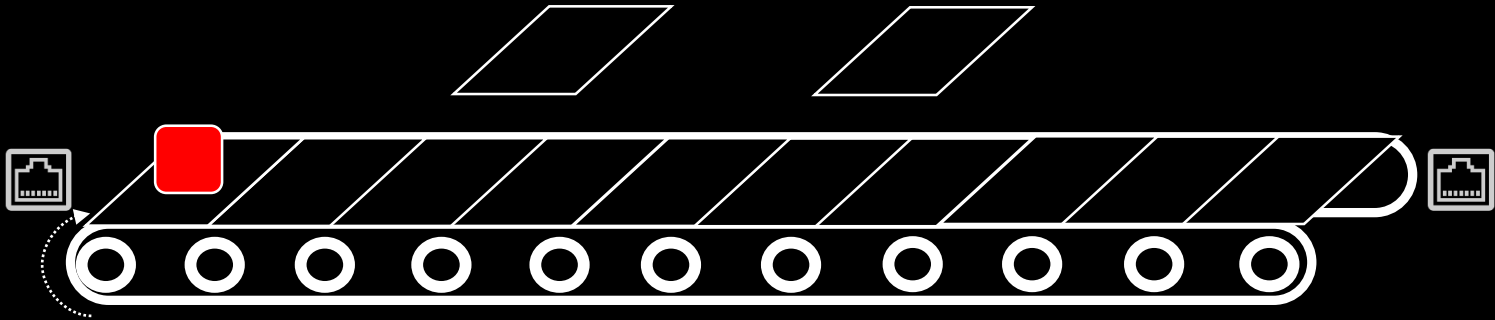
Encryption configuration

```
-----  
----|  Crypto Engine Setup  |----  
-----  
Do you want to setup the Crypto Engine? Press (y)es or (n)o  
Alright!  
  
What is the port exposed the outer world?  
Type 1 (parallel wrt board) or 2 (perpendicular)  
  
You chose port #2  
  
It's time to choose a password. Maximum characters allowed: 32  
Type your password here below. The ENTER key will validate your password  
  
*****  
Please, type your password once again to confirm.  
*****  
  
Congratulations: it's time to generate the seed now
```

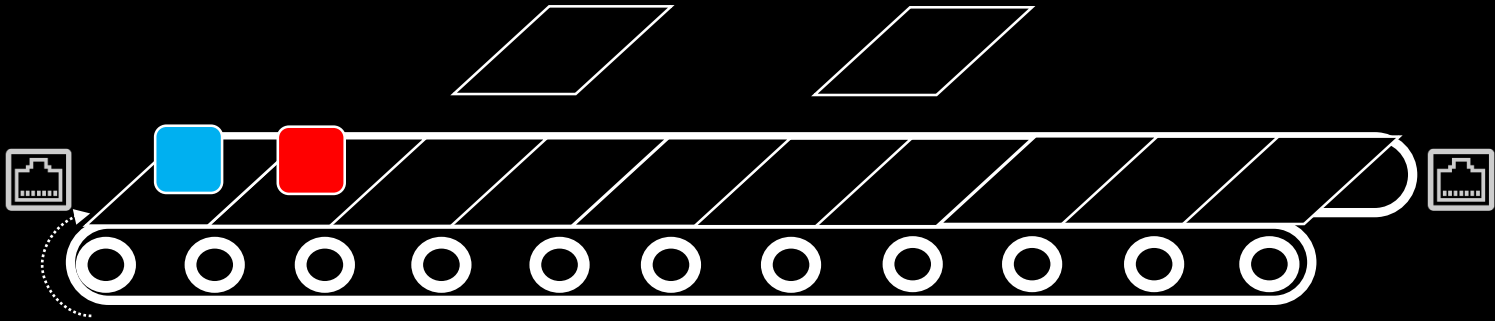
- Password input by the user
- A key is generated to encrypt data, one byte per time



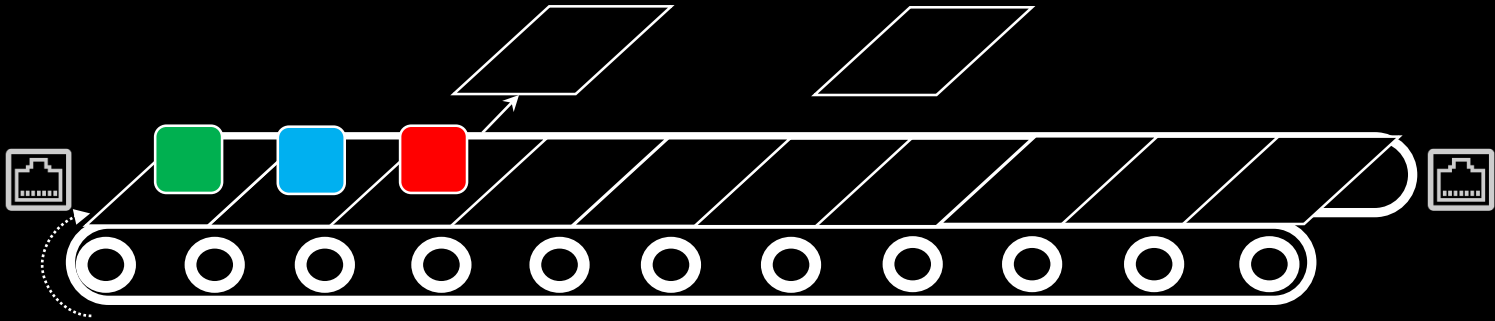
Encryption schema



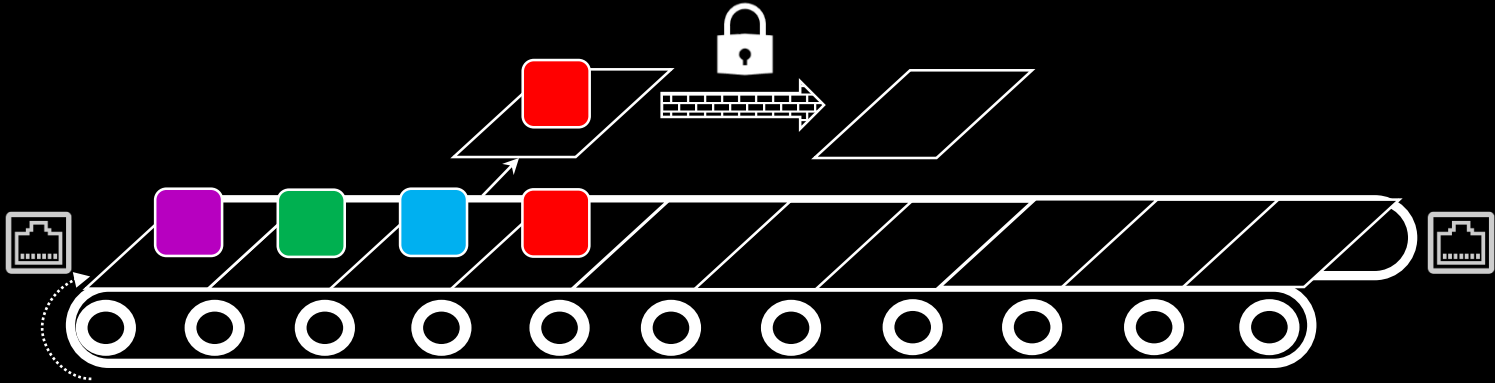
Encryption schema



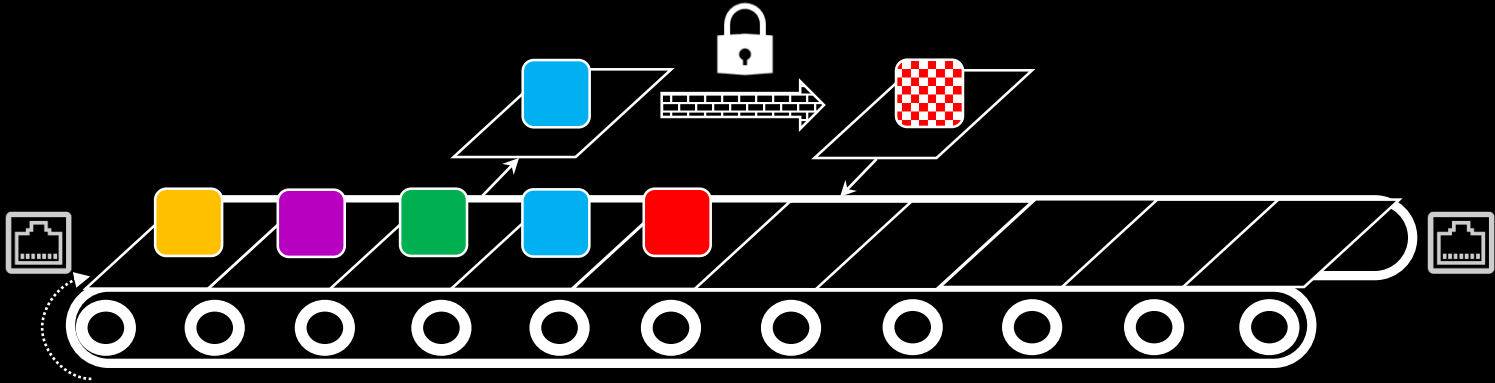
Encryption schema



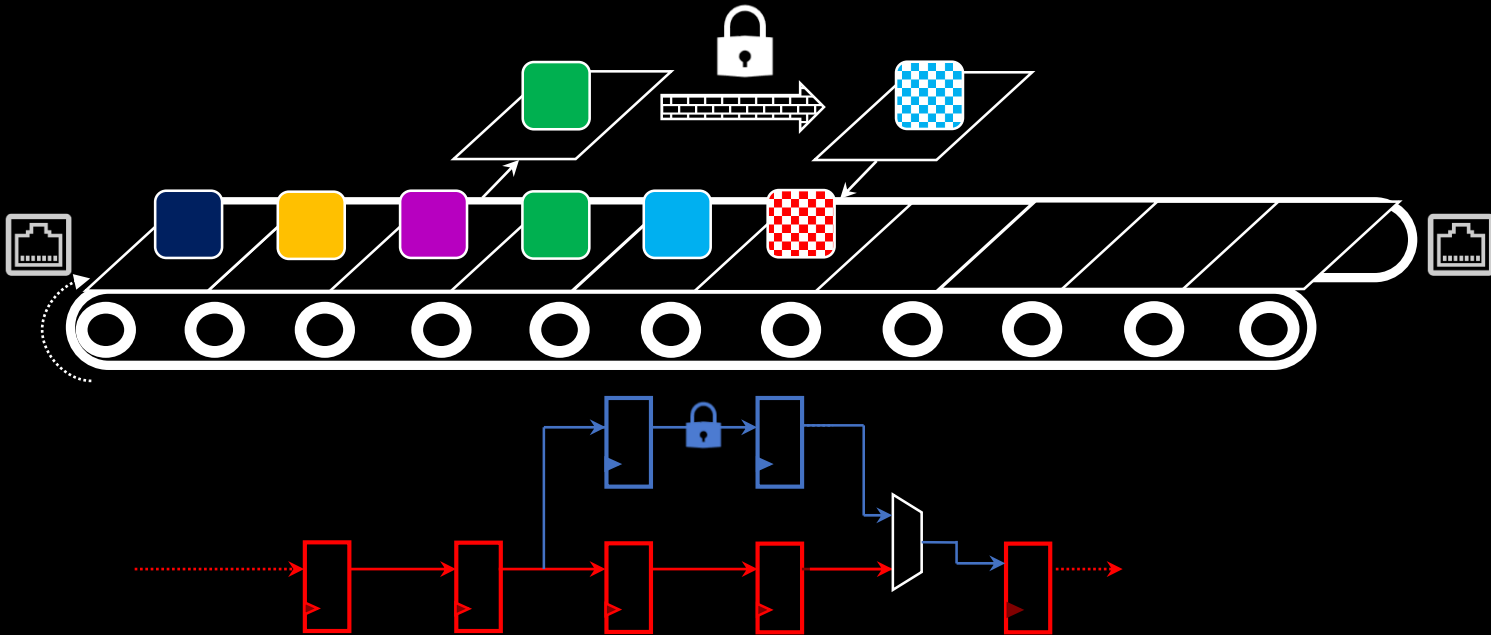
Encryption schema



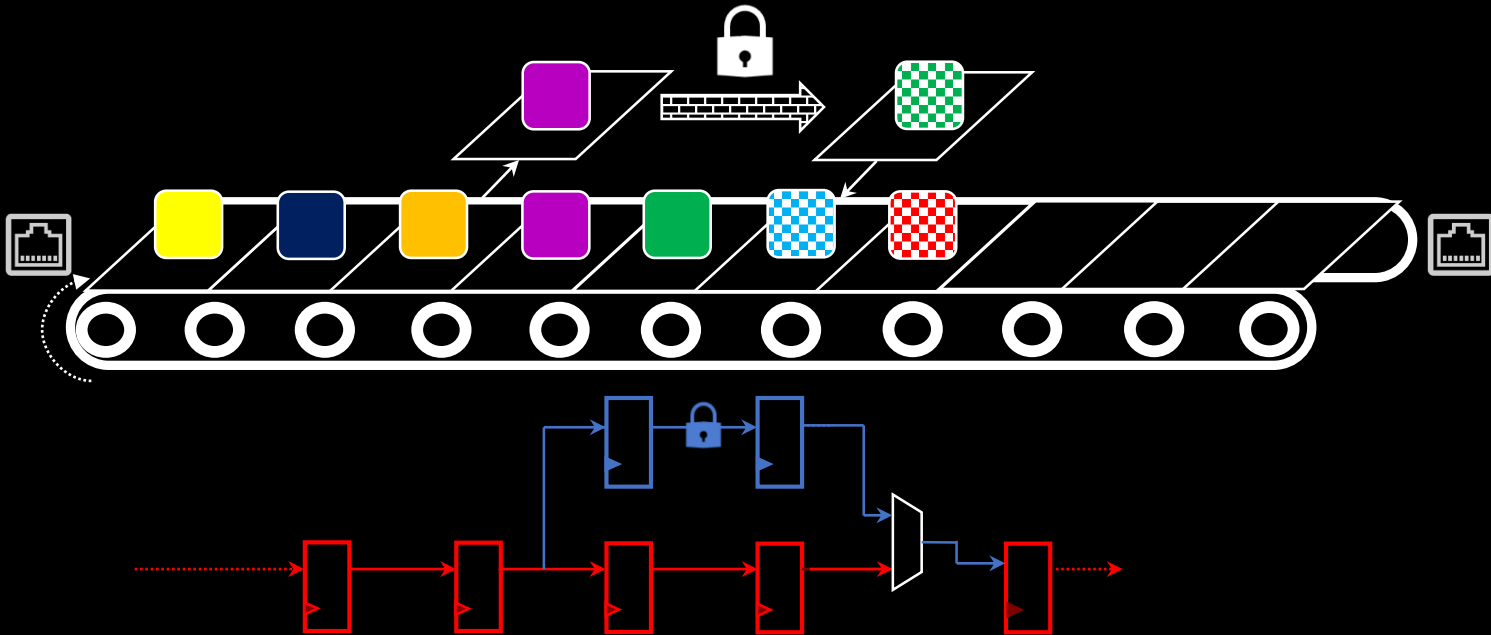
Encryption schema



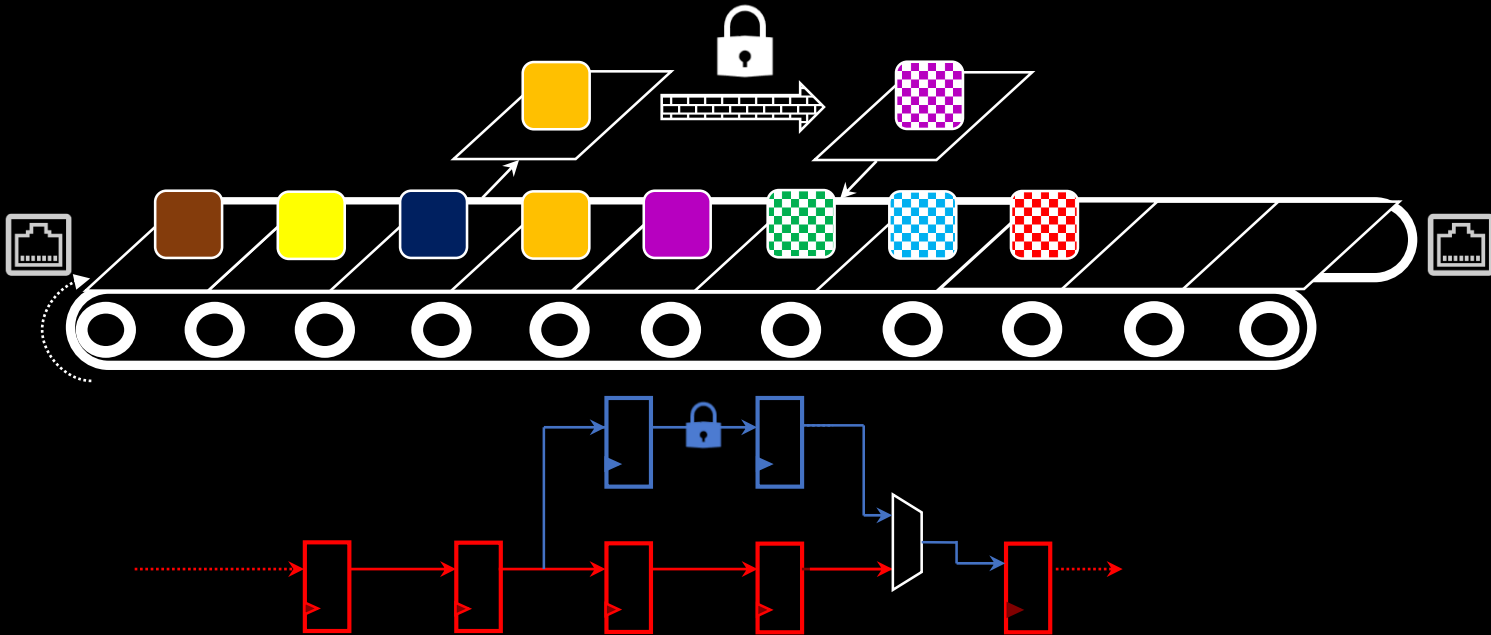
Encryption schema



Encryption schema



Encryption schema

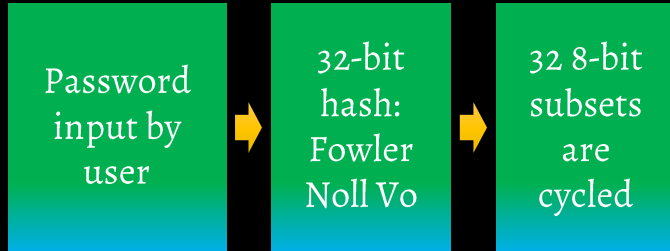


Encryption architecture

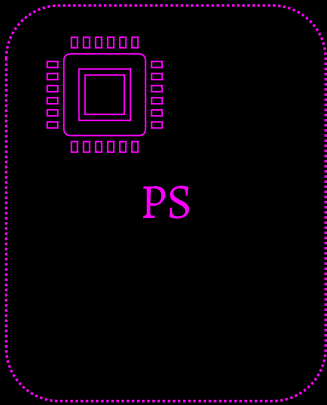


The following slides can
seriously hurt Crypto
specialists

Encryption architecture



Encryption architecture



JohnDo3

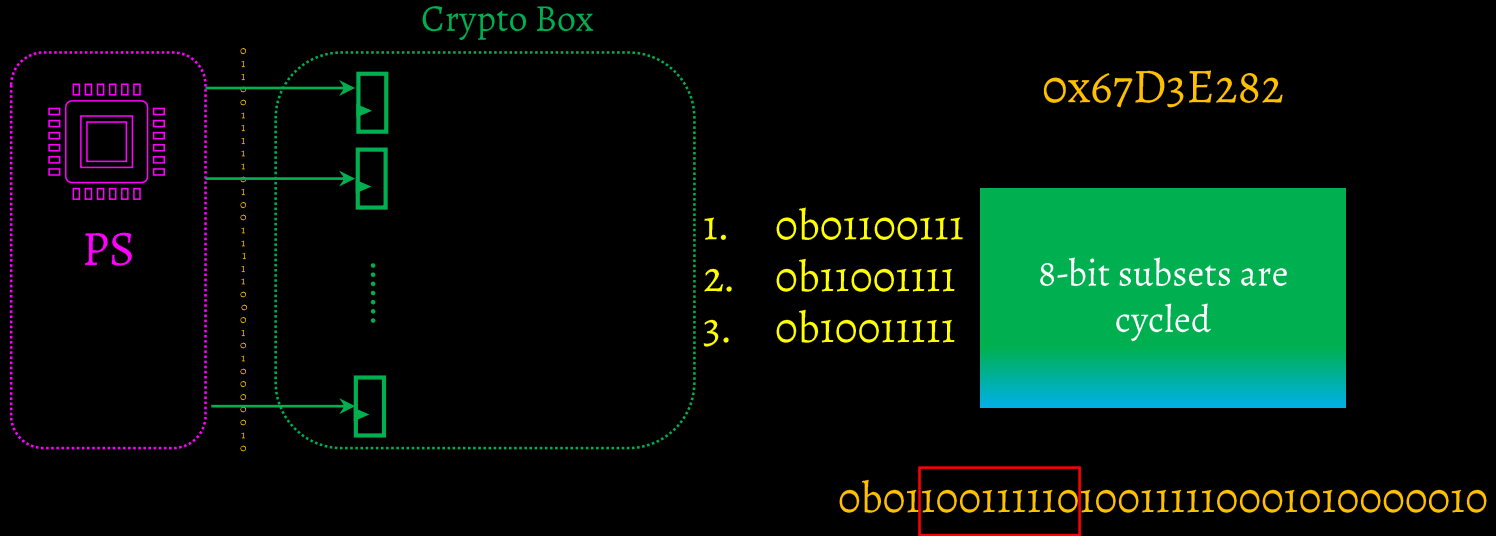
0x67D3E282

Password
input by user

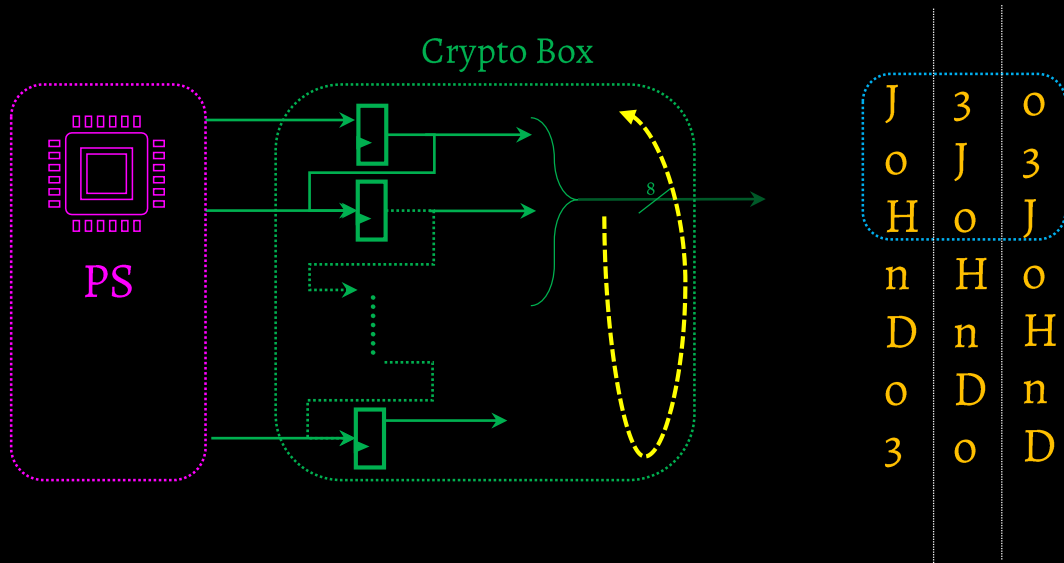


32-bit hash: Fowler
Noll Vo

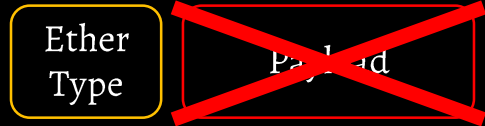
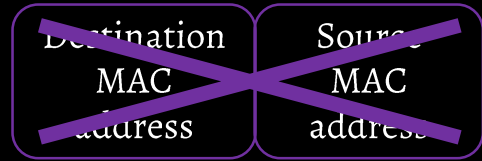
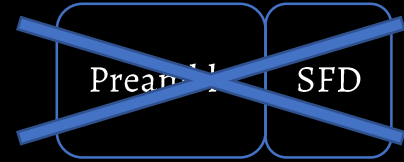
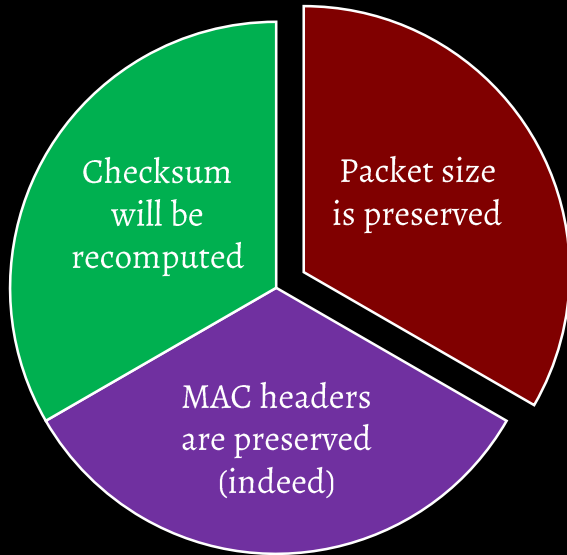
Encryption architecture



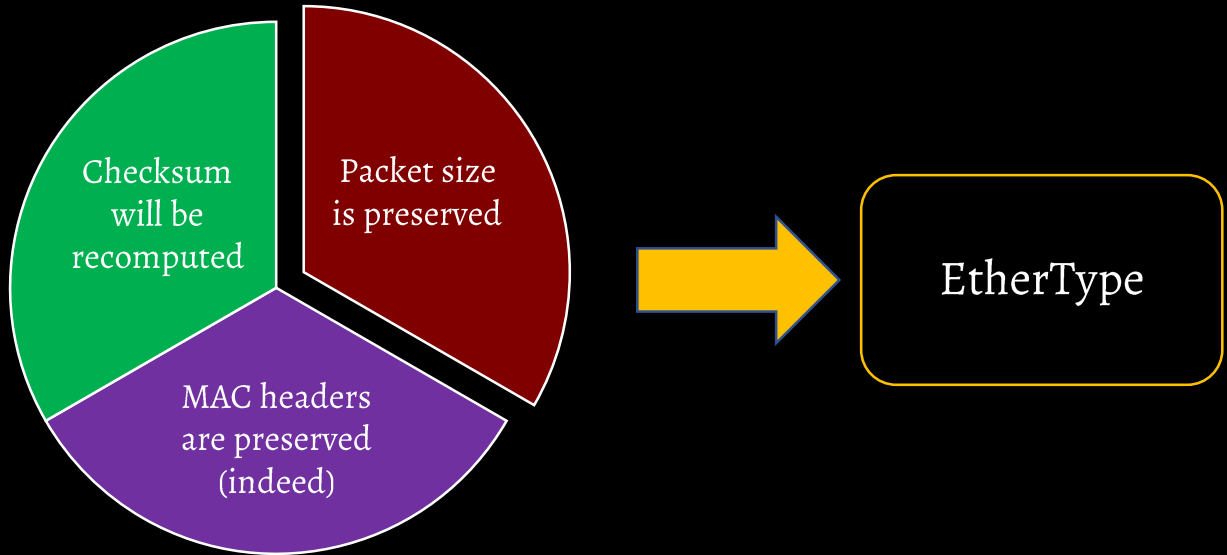
Encryption architecture



Encryption architecture – “Signature”



Encryption architecture – “Signature”



Encryption architecture – “Signature”

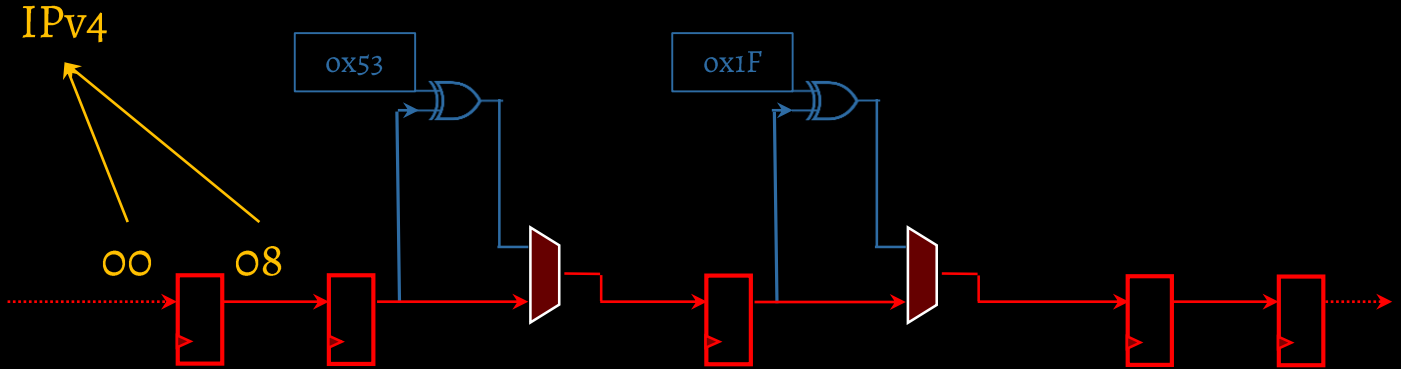
Match EtherTypes
with another ID



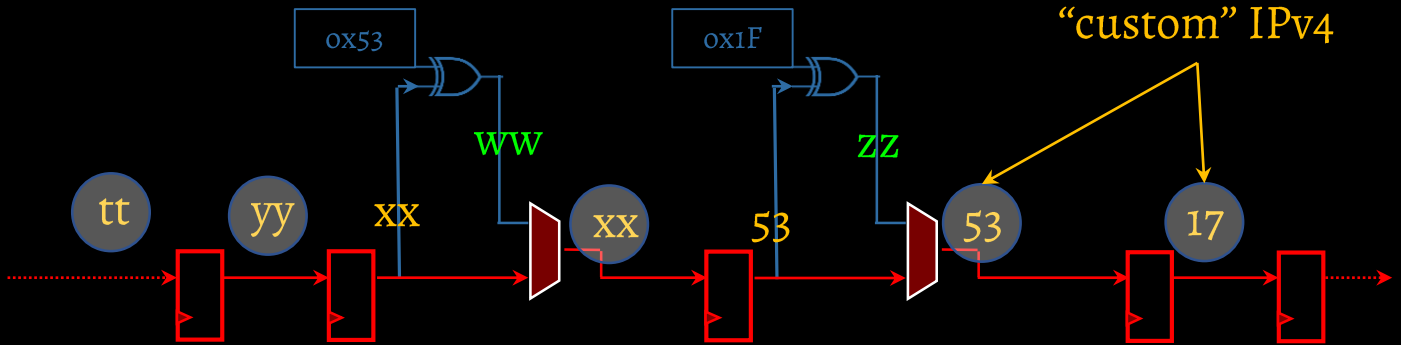
Only IPv4 is
encoded atm

EtherType	Protocol
0x0800	Internet Protocol version 4 (IPv4)
0x0806	Address Resolution Protocol (ARP)
0x0842	Wake-on-LAN ^[9]
0x22F3	IETF TRILL Protocol
0x22EA	Stream Reservation Protocol
0x6003	DECnet Phase IV
0x8035	Reverse Address Resolution Protocol
0x809B	AppleTalk (Ethernalk)
0x80F3	AppleTalk Address Resolution Protocol (AARP)
0x8100	VLAN-tagged frame (IEEE 802.1Q) and Shortest Path Bridging IEEE 802.1aq with NNI compatibility ^[10]
0x8137	IPX
0x8204	QNX Qnet
0x86DD	Internet Protocol Version 6 (IPv6)
0x8808	Ethernet flow control
0x8809	Ethernet Slow Protocols ^[11] such as the Link Aggregation Control Protocol

Encryption architecture – “Signature”



Encryption architecture – “Signature”



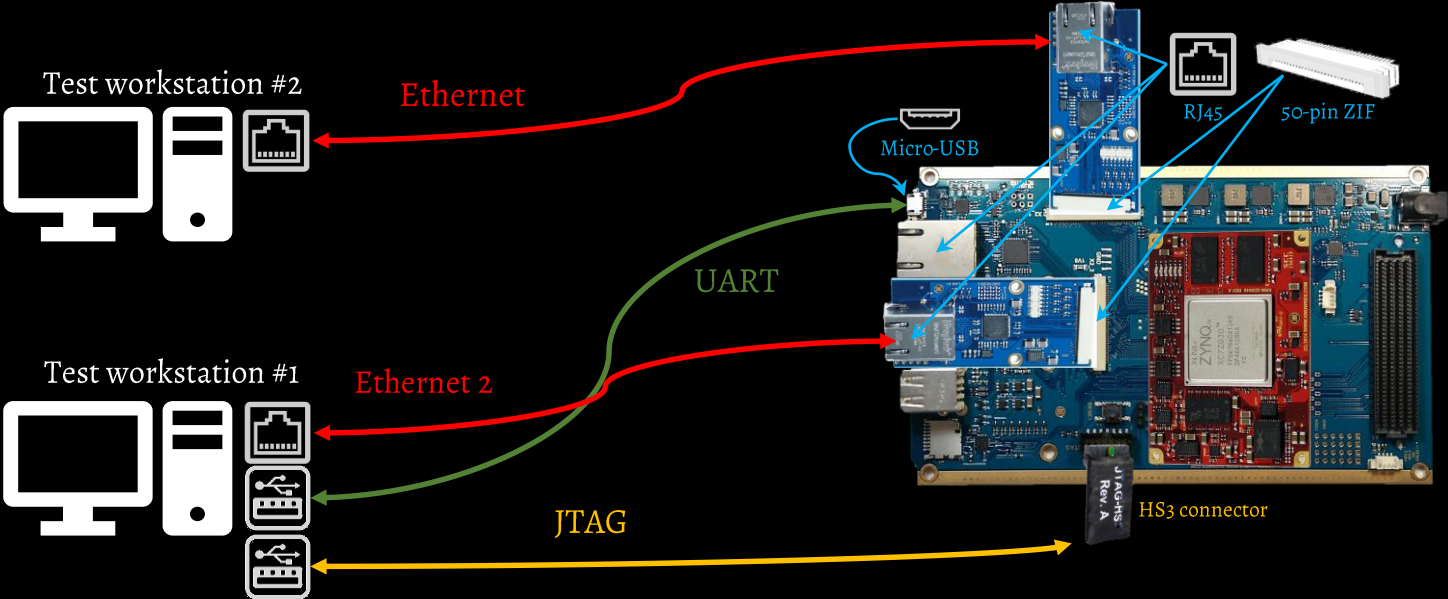
Encryption architecture – “Signature”



Encrypted IPv4 packets have
custom EtherType

The board automatically detects
and decrypts such packets

Demo : Encryption



Demo : Encryption

```
-----  
----|  Crypto Engine Setup  |----  
-----
```

```
Do you want to setup the Crypto Engine? Press (y)es or (n)o  
Alright!
```

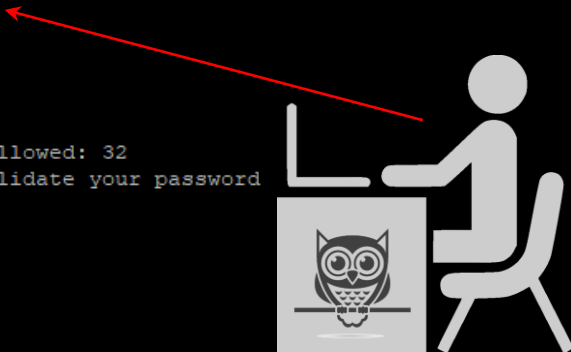
```
What is the port exposed the outer world?  
Type 1 (parallel wrt board) or 2 (perpendicular)
```

```
You chose port #2
```

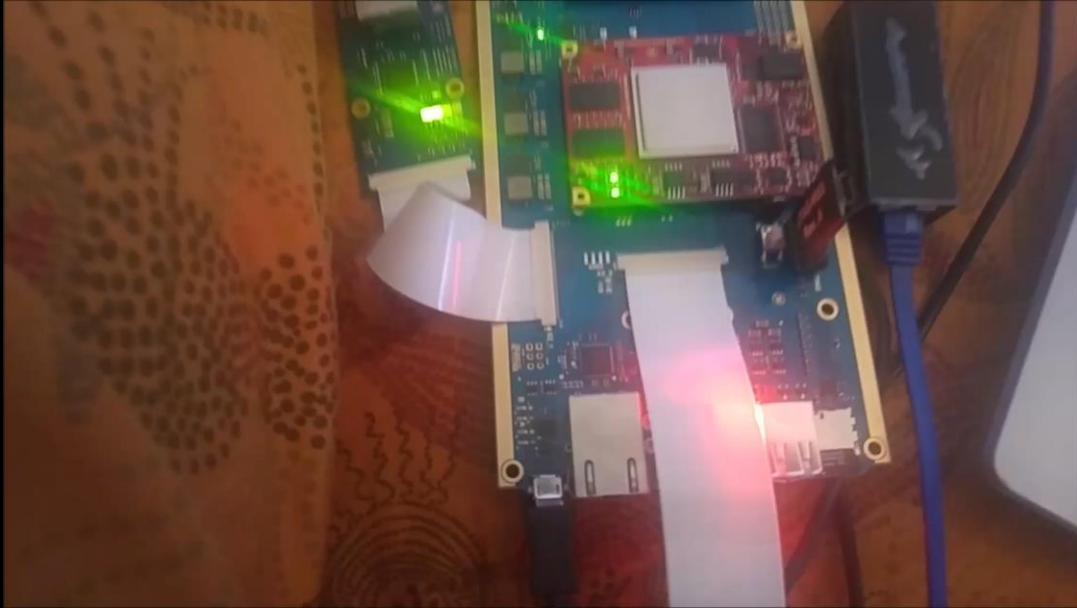
```
It's time to choose a password. Maximum characters allowed: 32  
Type your password here below. The ENTER key will validate your password
```

```
*****  
Please, type your password once again to confirm.  
*****
```

```
Congratulations: it's time to generate the seed now
```



Demo: Encryption



Demo: Encryption

The image displays two side-by-side screenshots of the Wireshark network traffic analysis tool. Both windows are titled "Capturing from Ethernet".

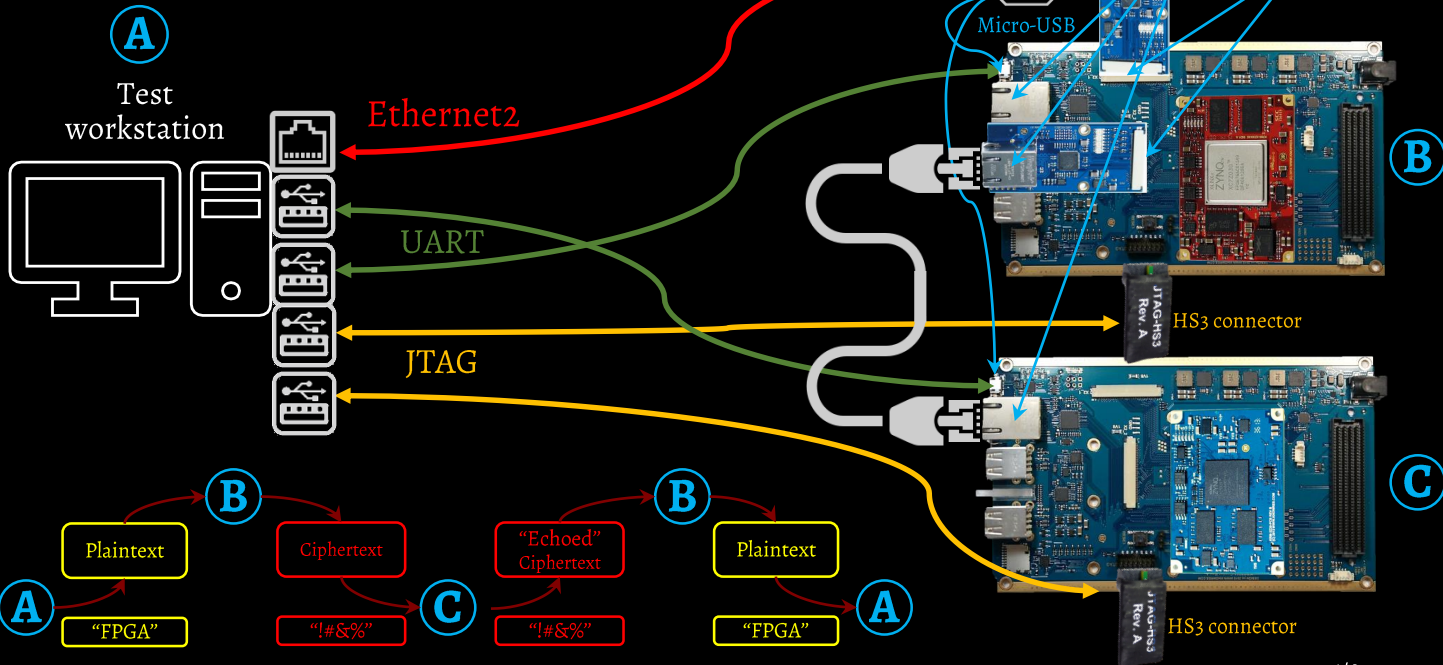
The left window shows a list of captured packets. Packet 14 is selected, showing the following details:

No.	Time	Source	Destination	Protocol	Length	Info
14	130.600358	c0:ca:c0:1a:00:00	Xilinx_01:02:03	Ethernet II	116	Bogus

The right window shows the details of packet 14. The "Internet Protocol Version 4" section is highlighted in red, indicating that the packet is an IPv4 packet.

Frame 14: 116 bytes on wire (928 bits), 116 bytes captured (928 bits) on interface Ethernet II, Src: c0:ca:c0:1a:00:00 (c0:ca:c0:1a:00:00), Dst: Xilinx_01:02:03 (00:00:00:00:00:00), Length: 116, Protocol: Internet Protocol Version 4

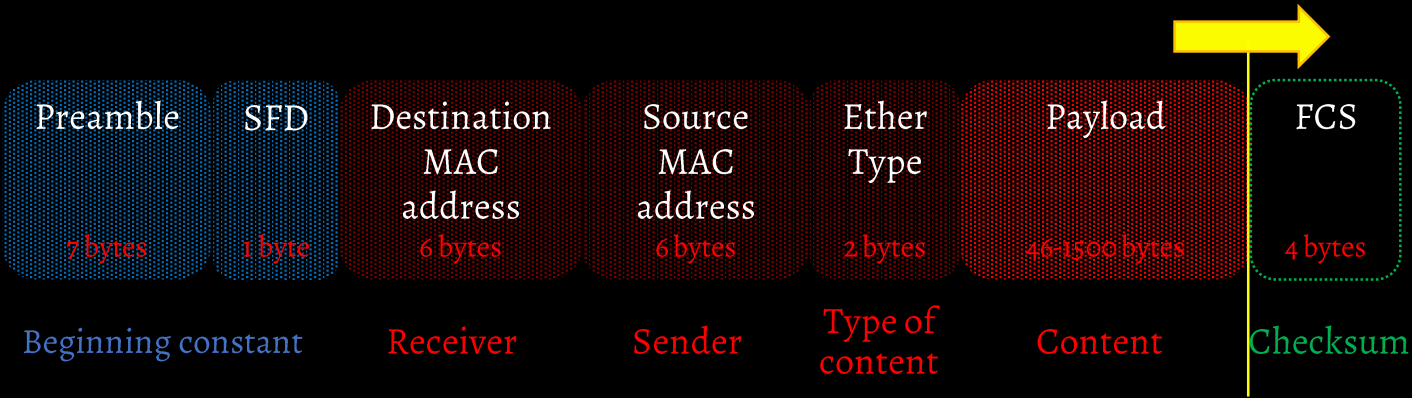
Demo: Decryption



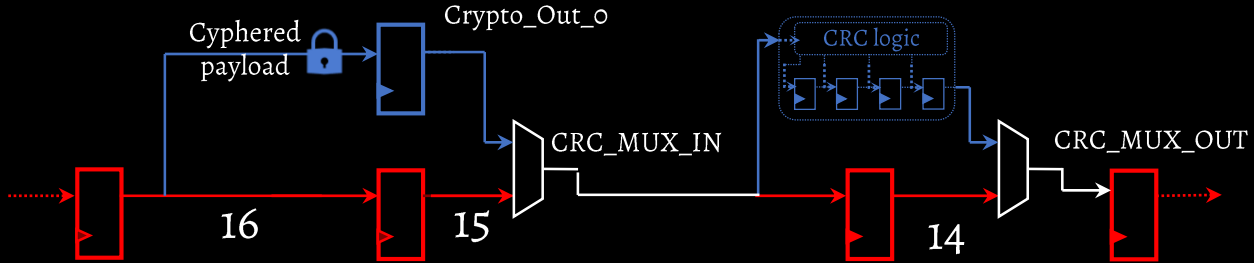
Demo: Decryption



Checksum generation



Checksum generation

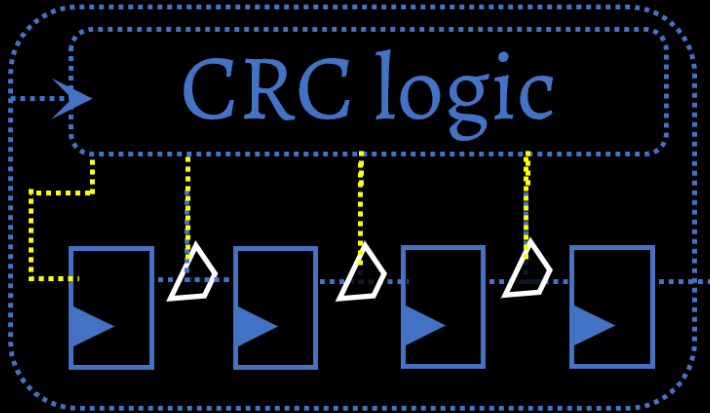


```
CRC_MUX_OUT <= CRC_OUT_o when crc_append = '1' else PIPELINE(14);  
CRC_MUX_IN <= CRYPTO_OUT_o when is_crypto_detected_reg = '1' else PIPELINE(15);
```

Flag raised when the last payload's byte crosses PIPELINE(14)

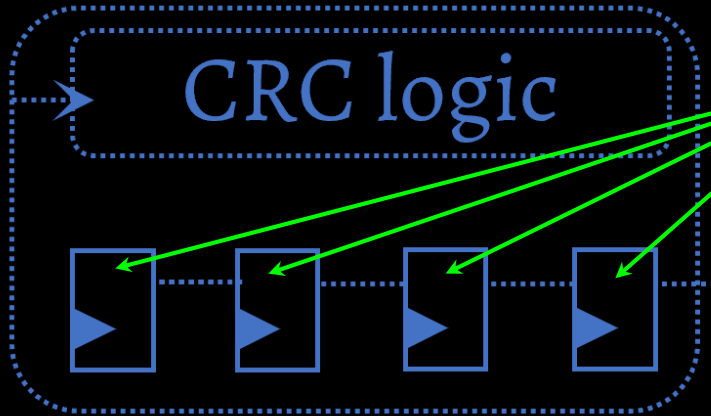
Flag raised when the custom IPv4 sequence is detected

Checksum generation



```
CRC_IN_3 <= crc_out(31 downto 24) when crc_append = '0' else (OTHERS => '0');  
CRC_IN_2 <= crc_out(23 downto 16) when crc_append = '0' else CRC_OUT_3;  
CRC_IN_1 <= crc_out(15 downto 8) when crc_append = '0' else CRC_OUT_2;  
CRC_IN_0 <= crc_out(7 downto 0) when crc_append = '0' else CRC_OUT_1;
```

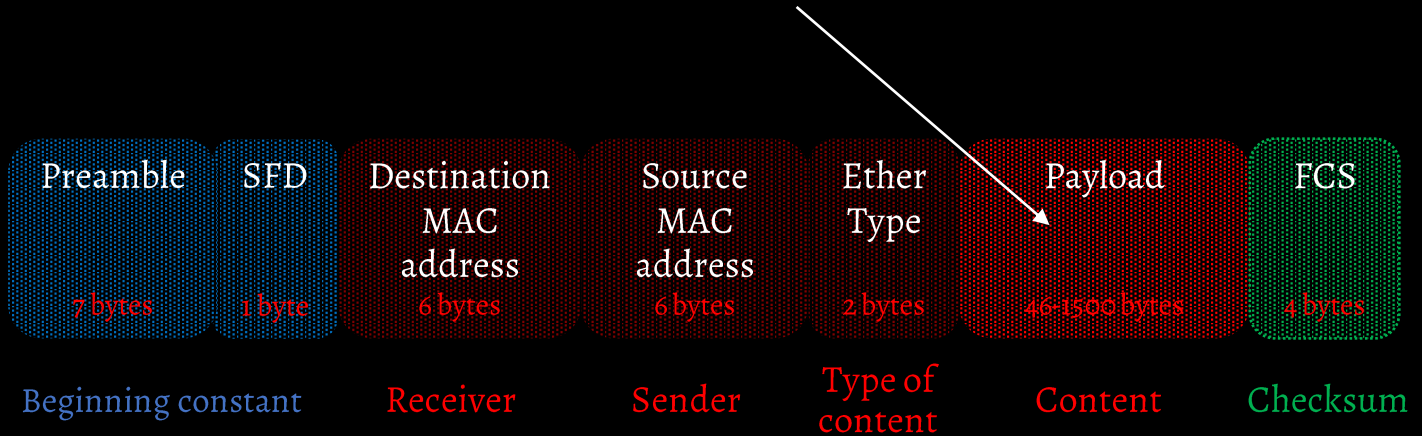
Checksum generation



```
CRC_OUT_i: ff_gen  
  generic map (NBITS =>  
    DEPTH)  
  port map(  
    CLK => CLK,  
    RST => RST,  
    DIN => CRC_IN_i,  
    DOUT => CRC_OUT_i  
  ); (OTHERS => '0');
```

32-bit checksum stored in 4 chunks

Moving towards a higher level...



Moving towards a higher level...

The screenshot shows the Wireshark interface with a list of 12 ICMP Echo (ping) packets. The selected packet (No. 7) is expanded to show its details. The 'Internet Control Message Protocol' section is highlighted in yellow. A blue arrow points from the text 'Byte 0x18' to the value '18' in the hex dump, which represents the offset to the ICMP Echo request data.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.89	192.168.0.1	ICMP	74	Echo (ping) request id=0x0200, seq=2304/9, ttl=128 (no
2	0.000028	192.168.0.89	192.168.0.1	ICMP	74	Echo (ping) request id=0x0200, seq=2304/9, ttl=128 (rep
3	0.003914	192.168.0.1	192.168.0.89	ICMP	74	Echo (ping) reply id=0x0200, seq=2304/9, ttl=255 (req
4	1.000271	192.168.0.89	192.168.0.1	ICMP	74	Echo (ping) request id=0x0200, seq=2560/10, ttl=128 (no
5	1.000587	192.168.0.89	192.168.0.1	ICMP	74	Echo (ping) request id=0x0200, seq=2560/10, ttl=128 (re
6	1.004632	192.168.0.1	192.168.0.89	ICMP	74	Echo (ping) reply id=0x0200, seq=2560/10, ttl=255 (re
7	2.001210	192.168.0.89	192.168.0.1	ICMP	74	Echo (ping) request id=0x0200, seq=2816/11, ttl=128 (no
8	2.001235	192.168.0.89	192.168.0.1	ICMP	74	Echo (ping) request id=0x0200, seq=2816/11, ttl=128 (re
9	2.005477	192.168.0.1	192.168.0.89	ICMP	74	Echo (ping) reply id=0x0200, seq=2816/11, ttl=255 (re
10	3.002313	192.168.0.89	192.168.0.1	ICMP	74	Echo (ping) request id=0x0200, seq=3072/12, ttl=128 (no
11	3.002342	192.168.0.89	192.168.0.1	ICMP	74	Echo (ping) request id=0x0200, seq=3072/12, ttl=128 (re
12	3.006210	192.168.0.1	192.168.0.89	ICMP	74	Echo (ping) reply id=0x0200, seq=3072/12, ttl=255 (re

Internet Protocol Version 4, Src: 192.168.0.89, Dst: 192.168.0.1

- 0100 = Version: 4
- 0101 = Header Length: 20 bytes (5)
- > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
- Total Length: 60
- Identification: 0xc644 (50756)
- > Flags: 0x0000
- Time to live: 128
- Protocol: ICMP (1)
- Header checksum: 0xf2d1 [validation disabled]
- [Header checksum status: Unverified]
- Source: 192.168.0.89
- Destination: 192.168.0.1
- > Internet Control Message Protocol

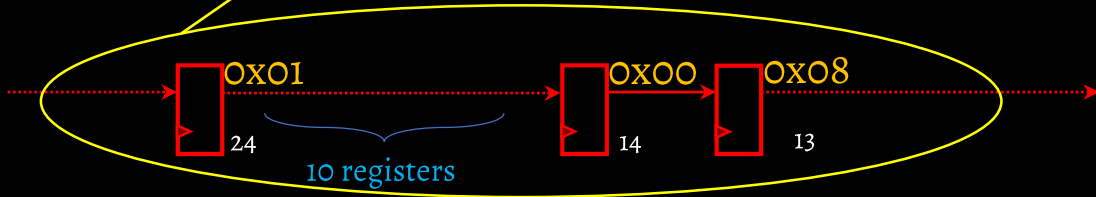
```
0000 00 0c ce 13 b9 a0 00 22 5f 3f 98 91 08 00 45 00  ....?_...E-
0010 00 3c c6 44 00 00 00 01 f2 d1 c0 a8 00 59 c0 a8  -<D...:  ....Y..
0020 00 01 08 00 40 5c 02 00 0b 00 61 62 63 64 65 66  -...@!...abcdef
0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76  ghijklmn opqrstuv
0040 77 61 62 63 64 65 66 67 68 69                    wabcedfg hi
```

Byte 0x18

Moving towards a higher level...

Killing ICMP packets!

```
when ICMP_KILL =>
  if (is_icmp = '1') then
    state_next <= DROP;
  else
    state_next <= WAIT_TILL_DONE;
  end if;
```



Achievements

- Transparent modification of data
- Real-time equivalent
- Pipelining allows multiple firewall-feature implementation
- The whole time of flight (192 ns) is below avg latency
- P2P Encryption

Future Upgrades

- Filter upper layer protocols (IP, etc.)
- Deep Packet Inspection (custom word lookup in payload, etc.)

Questions?



Thank you for your attention!



Follow me on LinkedIn: [linkedin.com/in/matteocollura](https://www.linkedin.com/in/matteocollura)