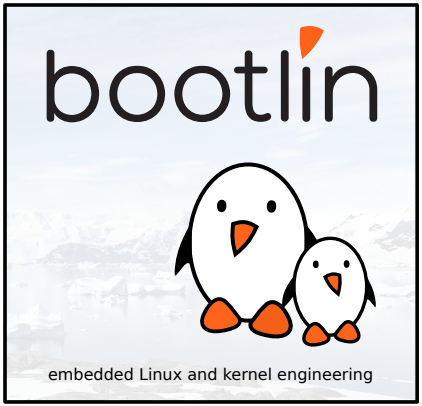
Offloading Network Traffic Classification

Maxime Chevallier

maxime.chevallier@bootlin.com



Maxime Chevallier

- Linux kernel engineer at Bootlin.
  - Linux kernel and driver development, system integration, boot time optimization, consulting...
  - Embedded Linux, Linux driver development, Yocto Project & OpenEmbedded and Buildroot training, with materials freely available under a Creative Commons license.
  - `https://bootlin.com`
- Contributions:
  - Worked on network (MAC, PHY, switch) engines.
  - Contributed to the Marvell EBU SoCs upstream support.
  - Also worked on SPI and real-time topics.

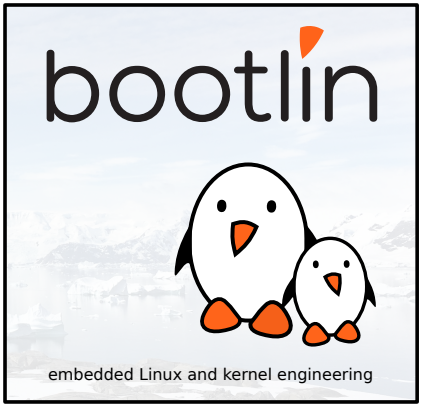
Preamble - goals

- Discover the classification operations in the kernel.
- Discover the hardware technologies used to offload packet classification
- Learn about the use cases for classification
- Based on **PPv2**'s behaviour and design, similar on other NICs

Introduction to Ingress Classification

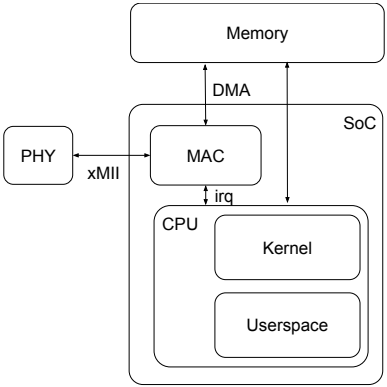
Maxime Chevallier

maxime.chevallier@bootlin.com



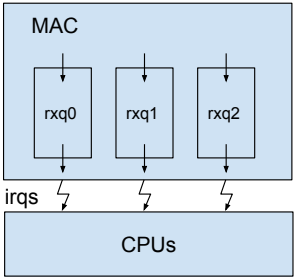
Packet path from the hardware to userspace

1. A frame arrives to the PHY
2. It is transferred to the MAC
3. The MAC performs offloaded operations
4. The packet is copied to RAM via DMA
5. The MAC raises an interrupt



MAC

- Upon receiving a frame, the packet goes through a **Packet Processor**
- The `MAC` receives the frames and places them into a buffer using `DMA`
- Descriptors for that buffer are placed into a **receive queue**
- Once the frame is received, the `MAC` raises an **interrupt**
- Receive queues can have dedicated interrupts, pinned to CPUs



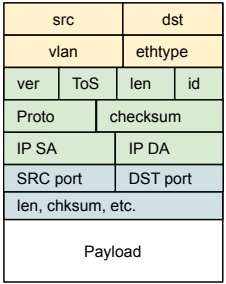
Kernel path

- The interrupt is handled, mostly in `softirq` context
- `napi` is used to coalesce interrupts
- Packet processing is done on the CPU that handled the interrupt
- Before going up the network stack, the packet goes through the `TC` subsystem
- L2 handling, to deal with `MAC` filtering and VLANs
- L3 handling, to deal with **routing**
- L4 handling, where we find the **socket** that will consume the payload

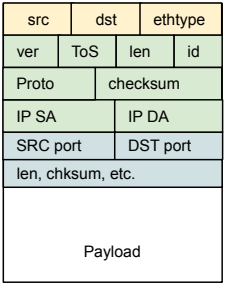
Classification

- Classification consists in **identifying packets of interest**
- We can then perform **actions** on these packets
- We first need to **dissect** the packet
- Determining the various attributes of interest isn't straightforward
- All fields don't have a fixed offset in the packet

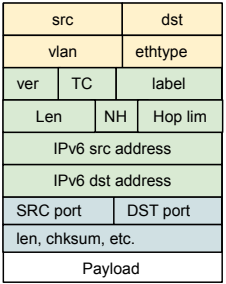
VLAN, IPv4, TCP



IPv4, TCP



VLAN, IPv6, TCP



Flows

- A **flow** characterizes a group of packets that have a common source and destination.
- We group packets based on common attributes, such as :
  - The source and destination IP addresses (2-tuple)
  - The L4 protocol, source and destination ports (5-tuple)
- We manipulate flows to avoid reordering and optimize locality
- We need to extract the required information from the headers.

TC flower

- **T**raffic **C**ontrol
- Used for traffic shaping, scheduling, policing and filtering
- In our case, we'll focus on the **tc flower** ingress filter
- **tc flower** is a **classifier**, which uses either software or hardware
- `tc qdisc add dev eth0 ingress`
- `tc filter add dev eth0 protocol ip parent ffff: flower ip_proto tcp dst_port 80 action drop`

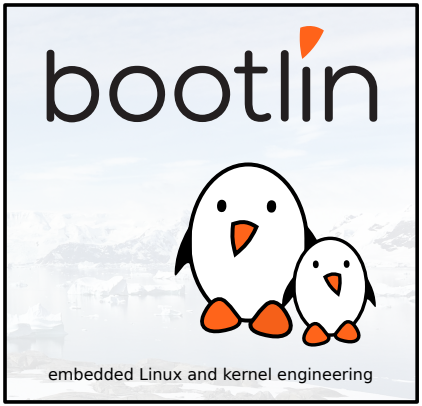
ethtool

- `ethtool` is used to interact with network drivers
- `ethtool -N` can be used to configure **n-tuple** filters
- It acts on specific **flow types** : `tcp4`, `udp6`, `ether`, etc.
- Rules are **ordered**, the first one that matches takes precedence
- `ethtool -N eth0 flow-type tcp4 dst-port 80 action -1 loc 0`
- Actions can be :
  - Steer to a Receive Queue
  - Steer to a RSS context
  - Drop

Offloading Classification

Maxime Chevallier

maxime.chevallier@bootlin.com



When and why

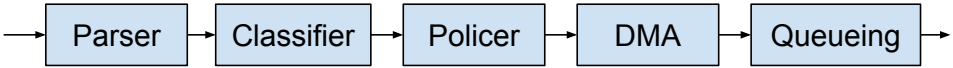
- Reduce CPU load
- Spread traffic across CPUs with per-cpu interrupts
- Early drop in case of Denial-of-Service attack
- Early redirection with switches

We must however be careful :

- The kernel might not see important packets
- The kernel might want to have access to the first packet of new flows
- Counters are not up to date anymore

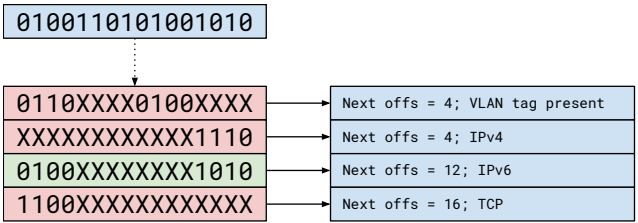
Hardware design

- Need to extract the required fields from the headers at wire speed
- These fields aren't always at a know position
- We need fast ways to lookup these fields, using a **parser**
- The attributes extracted by the parser are then used for **classification**



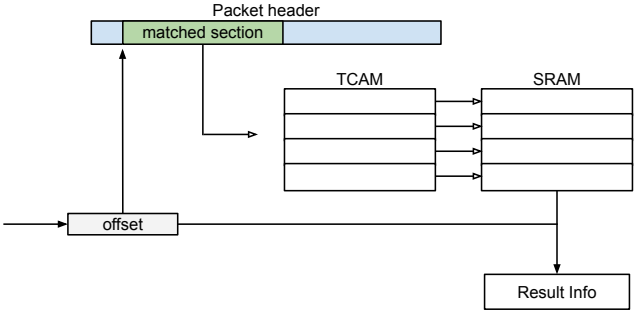
Ternary Content Addressable Memory

- Very fast lookups, but takes place on the die
- Addressed by value, returns the index of the first match
- Match on a ternary value : `0`, `1` and `X`
- The matched pattern is extracted from the header starting from an offset
- The returned index is used to lookup a SRAM containing match actions



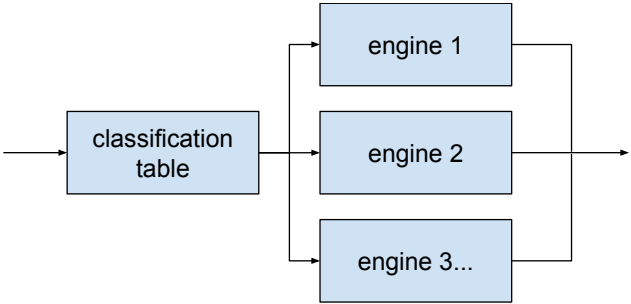
Parser

- Acts as a **dissector**
- Extract useful information from the packet header
- Take into account the various offsets due to DSA, VLAN and L3/L4
- Used as a pre-step for classification
- Often hardcoded in a firmware or a driver
- Multiple iterations per packet, flags are accumulated



Classifier

- Uses information from the **parser**
- Can use several engines to classify and perform actions :
  - TCAM engines, for exact matches
  - Hash-based engines, for rate limiting and RSS
  - Logic engines for complex rules
- A final policing step decides what to do based on results from engines
- Not all these possibilities can be expressed by the generic frameworks



Receive Side Scaling

- Spread traffic across multiple CPUs
- Compute a hash from specific fields from the header
  - **s** : Source IP, **d** : Destination IP
  - **f** : Source port, **n** : Destination port
  - **v** : VLAN tag, **m** : Destination MAC
- Make sure that traffic from the same flows ends up on the same CPU
- Spreading is configured using an **RSS Table**

- `ethtool -N eth2 rx-flow-hash tcp4 sdfn`
- `ethtool -X eth0 weight 2 1 1 0`

PPv2 example

- PPv2 is found on Marvell SoCs, such as the Armada 70xx and 80xx
- Has a TCAM parser with 256 entries, performing up to 16 matches on 11B
- Classifier has one 512 instruction table, subdivided in subflows
- Has 4 classification engines :
  - **C2** : TCAM match engine, 8B keys, 256 entries
  - **C3** : Exact match engine, 12B keys, 4K entries
  - **C4** : Classification and Marking engine, uses if-then-else constructs
  - **C3Hx** : Computes hashes, for RSS and C3 lookups.
- Can perform drop (in parser or classifier), steer to queue or RSS, limit traffic, modify and redirect packets.
- Parser and Classifier is shared between multiple ports

PPv2 : Current support

- Support for basic RSS
- Support steering on 2-tuple, 5-tuple and VLAN tag
- MAC and VLAN filtering, performed by the parser
- Support steering to RSS tables
- All Parser and Classifier configuration is done by the kernel, no firmware involved
- Only `C2` and `C3Hx` engines are used, others are way too complex

Conclusion

- Offloading classification requires a lot of hardware configuration
- Most of the time, we need to limit ourselves to a subset of what the HW can do
- There are ongoing efforts to solve the issue of stats reporting
- Performance and power consumption improvements make it worth it
- In most cases, a firmware is in charge of configuring most of the tables

Thank you!
Questions? Comments?

Maxime Chevallier — maxime.chevallier@bootlin.com

https://bootlin.com/pub/conferences/2019/elce/chevallier-network-classification-offload/