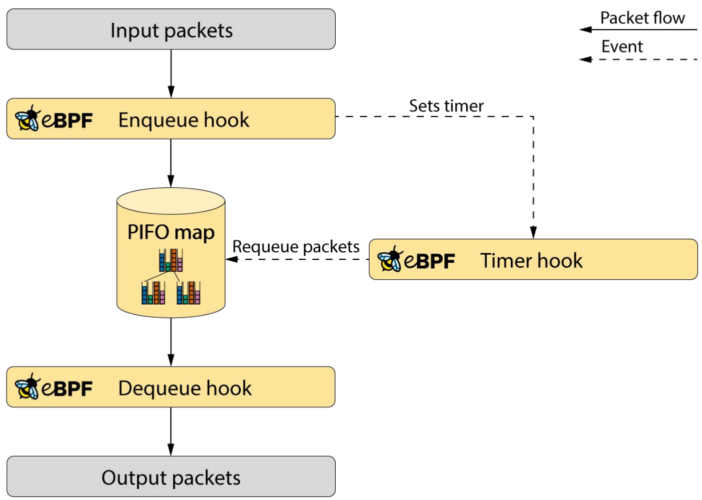
Bringing packet queueing to XDP

by Frey Alfredsson, Karlstad University
Jesper Dangaard Brouer, Red Hat
Toke Høiland-Jørgensen, Red Hat
Anna Brunström, Karlstad University
Per Hurtig, Karlstad University

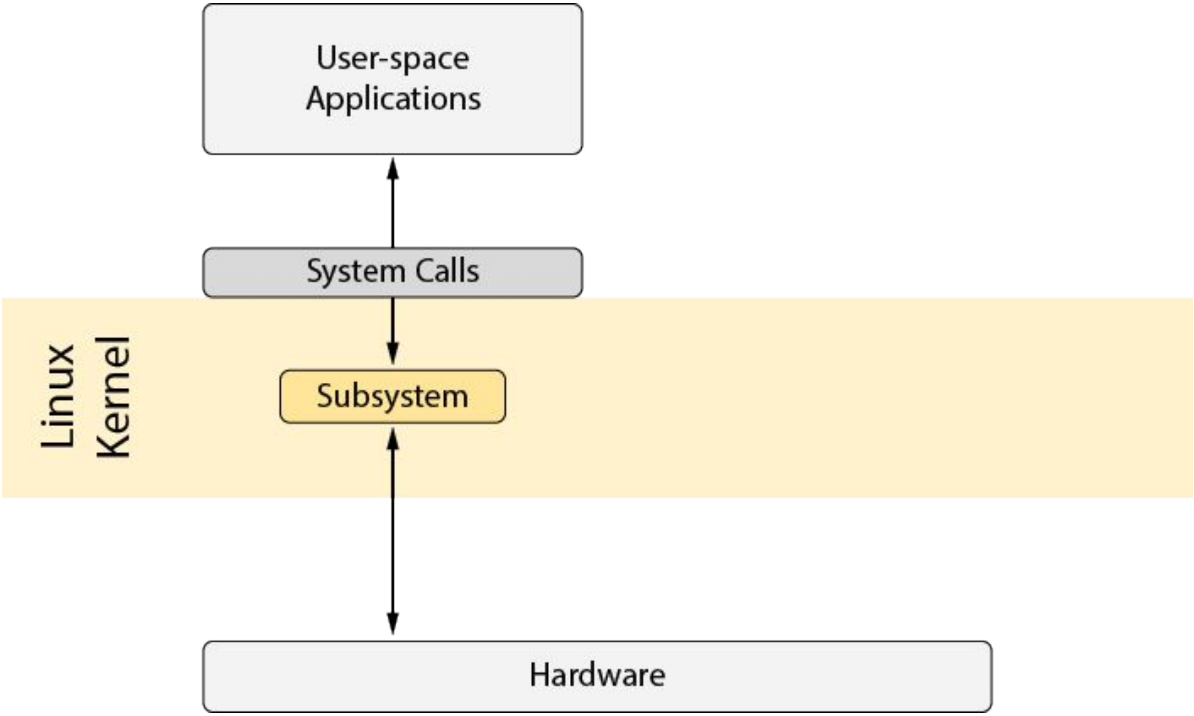


Attribution 4.0 International (CC BY 4.0)

Includes work from https://www.bpf.io

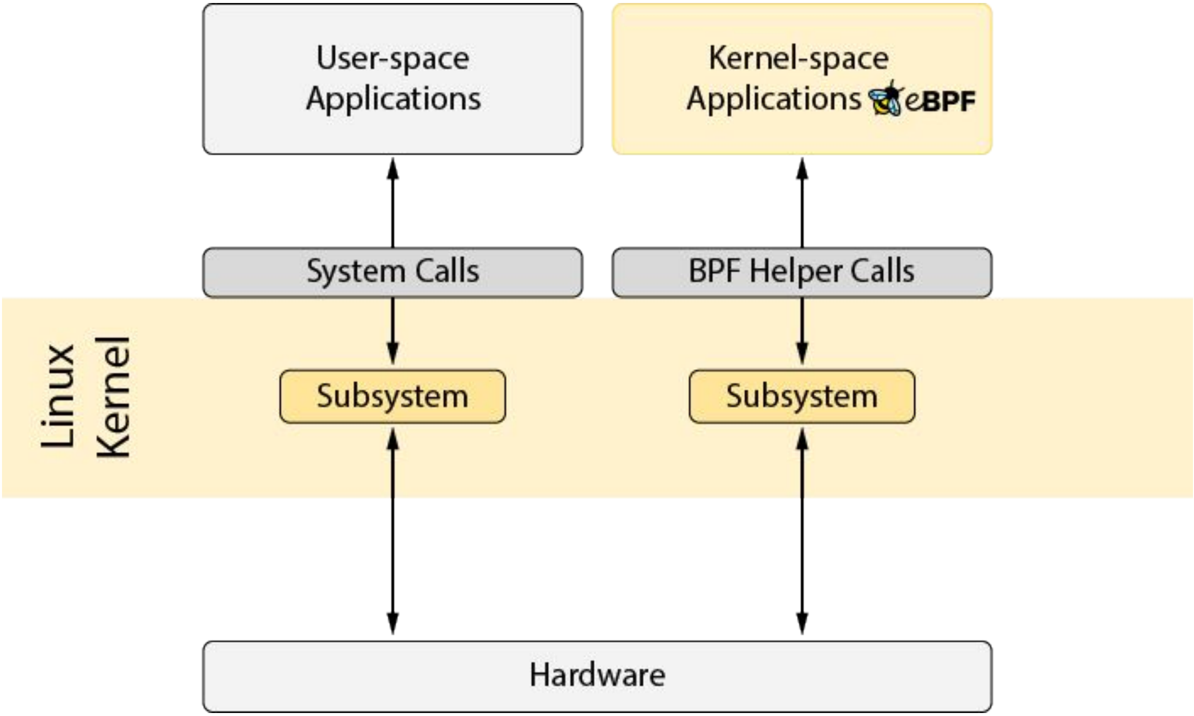
- BPF
- Packet Scheduling
- XDP
- Packet queueing to XDP
- PIFO
- FQ Example
- Summary

The BPF run-time environment



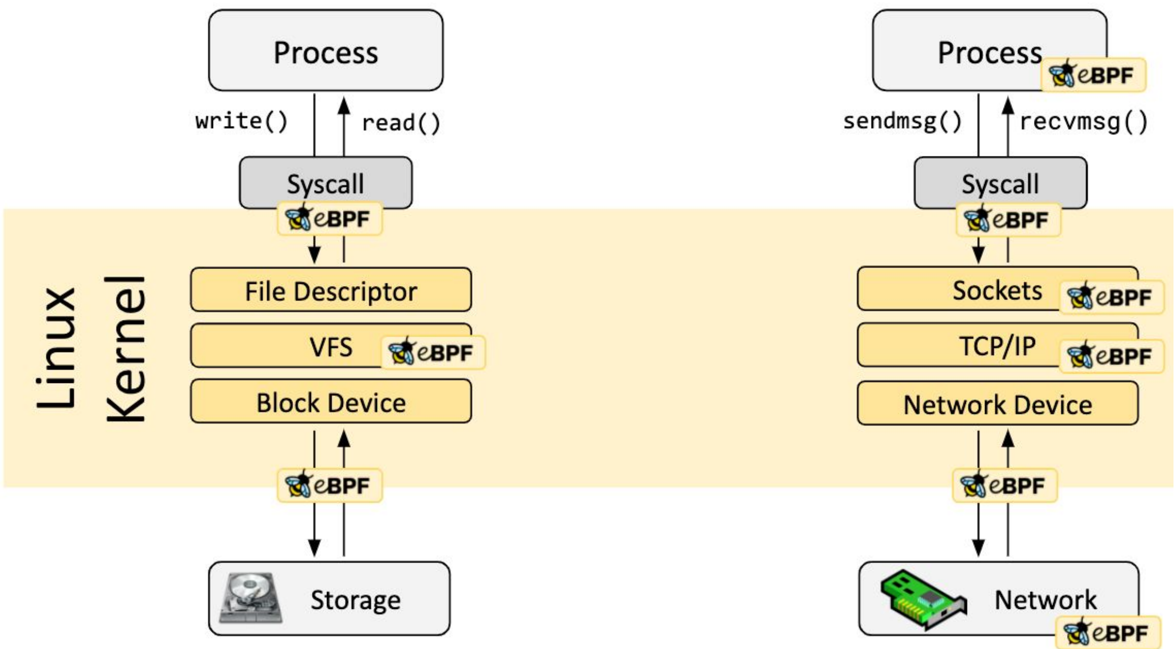
The BPF run-time environment

- BPF is a run-time environment to attach and run specialized code within the Linux kernel
- The BPF ecosystem comes with a full tool-chain:
  - Compilers
  - Loader libraries
  - Tools



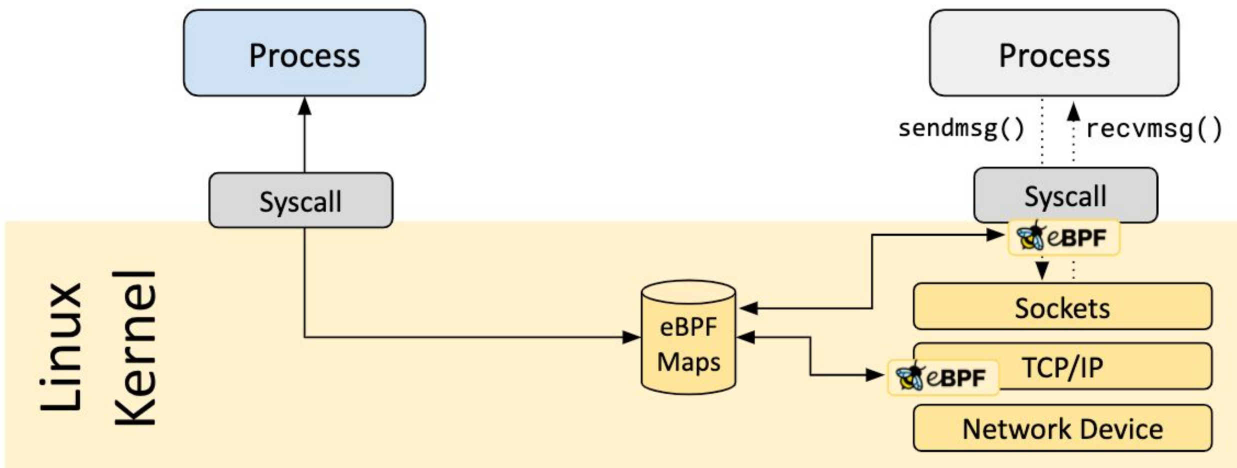
BPF hooks

- The BPF applications are attached to hook points within the Linux kernel
- Each hook has different sandbox rules
  - Which BPF helper functions can be called
  - What memory can be accessed



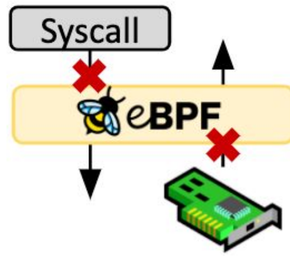
BPF inter-process communication

- This run-time provides us with:
  - Inter-process communication is done using BPF Maps
  - Inter-process communication between user- and kernel-space
  - Inter-process communication between BPF attached code

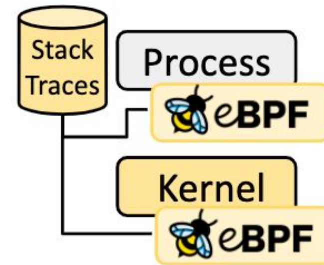


What can BPF do for us?

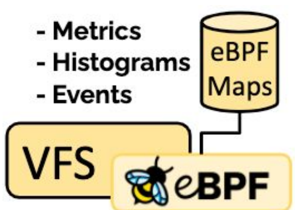
- **Security**
  - cgroups
  - IDS
  - IPS



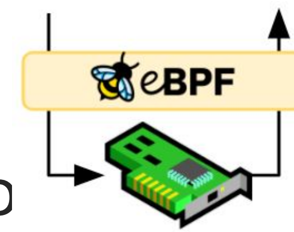
- **Tracing & Profiling**
  - Debugging
    - Software chain
    - Kernel internals
  - Applications
    - Performance
    - RAID size



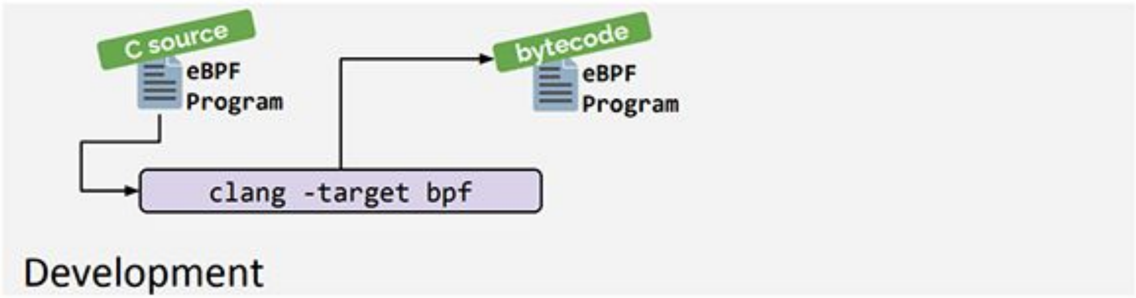
- **Observability & Monitoring**
  - Monitoring
    - I/O
    - Network
    - System calls
    - Process scheduling
    - Memory



- **Networking**
  - XDP
    - Acceleratio
    - Load balancing
    - DDoS prevention
  - XDP socket
    - Load balancing

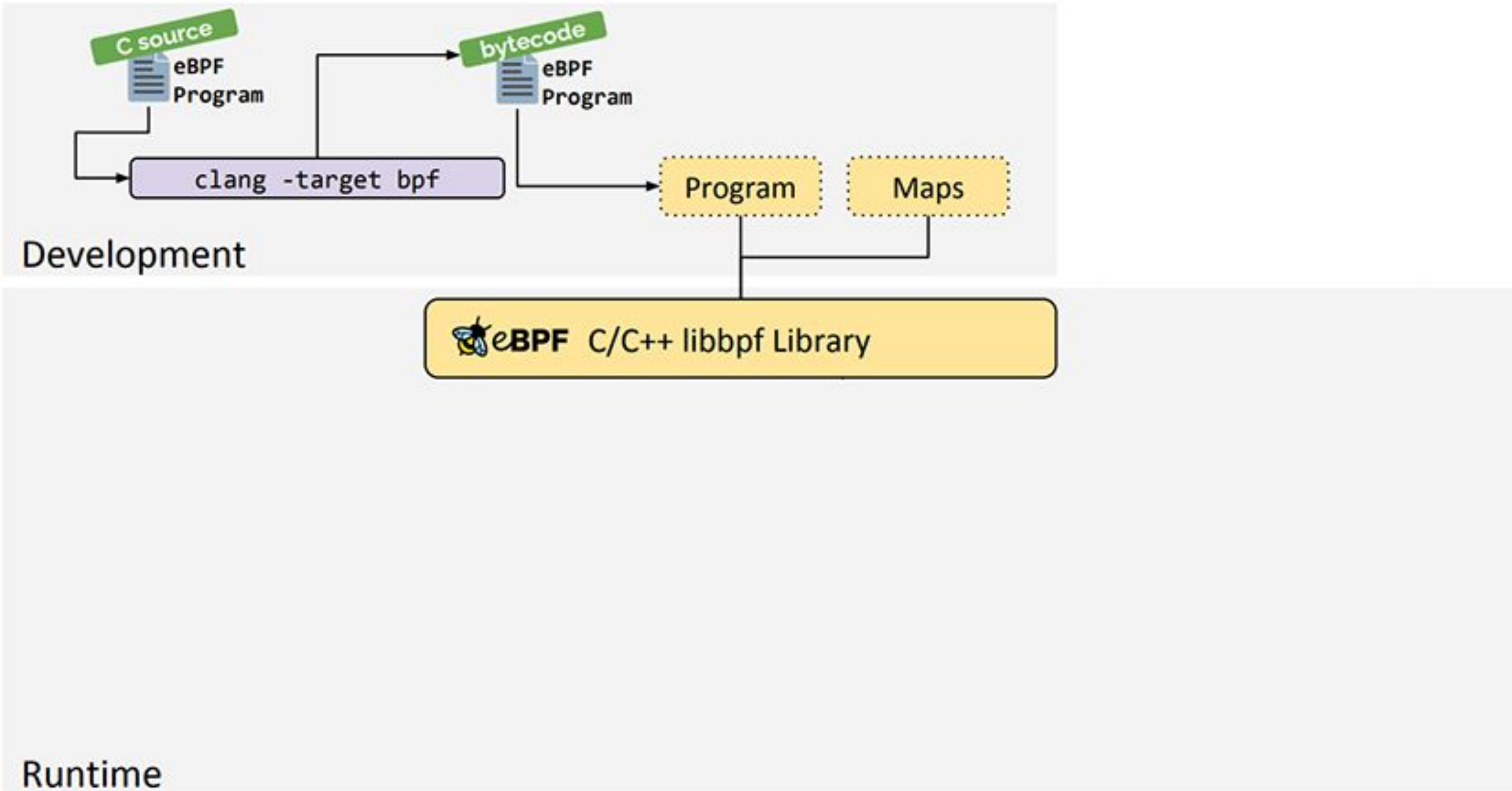


The BPF life-cycle (1/3)



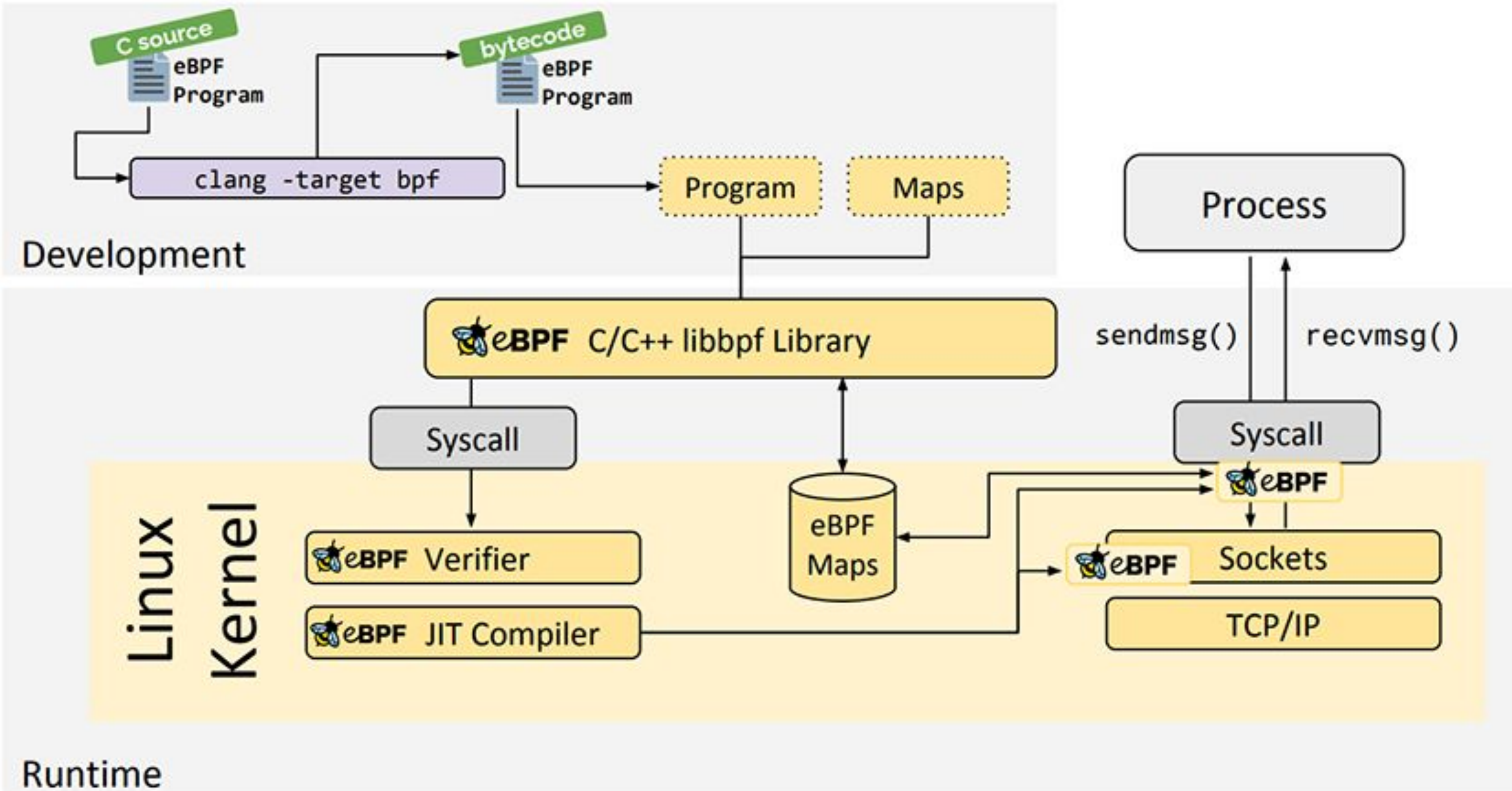
- Development environment:
  - Bytecode / Machine instruction language
  - Compilers and tools

The BPF life-cycle (2/3)



- A runtime environment that loads the BPF code
  - Multiple libraries and programs exist
    - It's recommended to use libbpf today

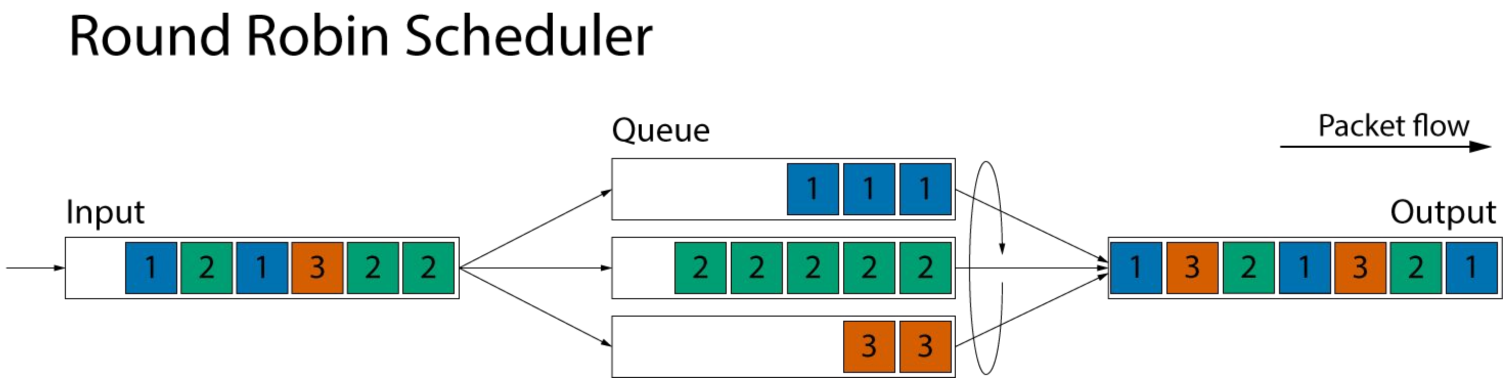
The BPF life-cycle (3/3)



- The kernel handles the BPF program by:
  - Verifying that it does not break the kernel
  - Attaching the BPF program to a hook

Packet Scheduling

- Packet Scheduling algorithm determines the order of packets being transmitted
  - A simple scheduler example:



  - In this example, packets are sorted by flows into different queues that are dequeued in a round robin fashion

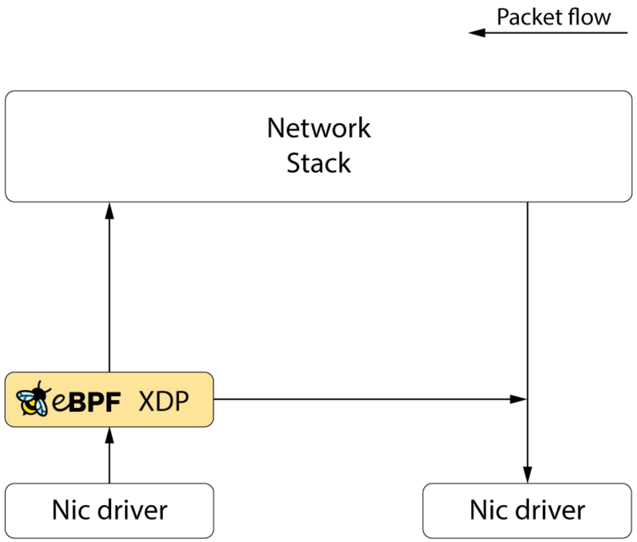
Packet scheduling and queue management

- Traffic scheduling policies
  - Provide all clients with equal throughput
  - Prioritize the production environment over the testing environment
  - Prioritize sparse flows

- Queue management
  - Bufferbloat mitigation
    - See https://www.bufferbloat.net

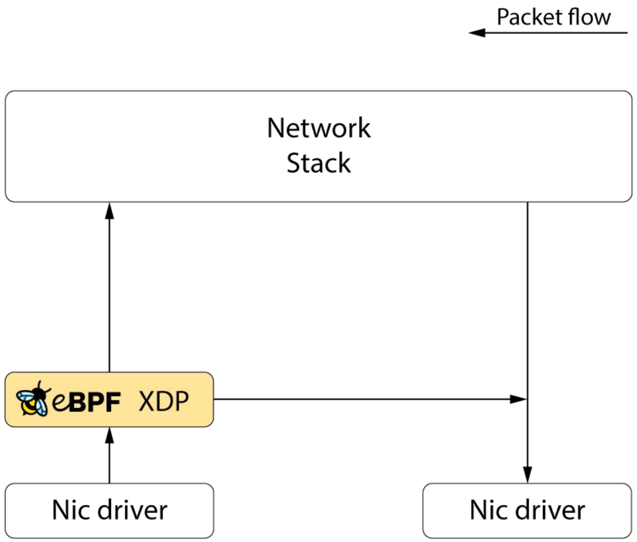
XDP – eXpress Data Path

- The eXpress Data Path is an in-kernel network fast-path
- XDP is a BPF hook that resides in-front of the network stack
- It provides the following operations:
  - Packet manipulation
  - Packet redirection
  - Packet drop
  - Monitoring



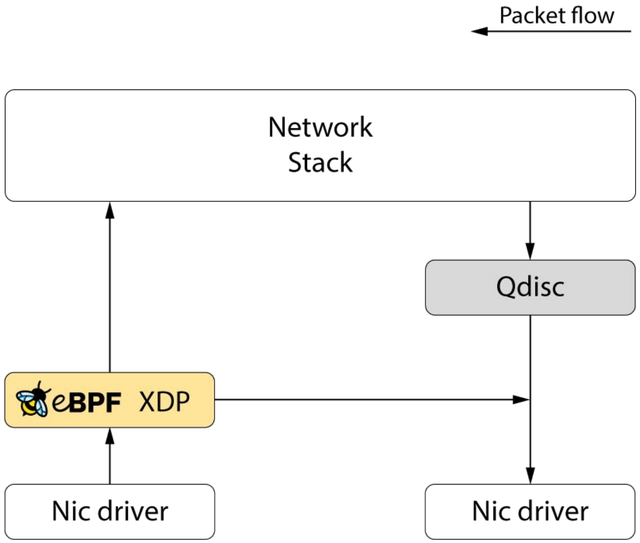
XDP – eXpress Data Path

- The eXpress Data Path is an in-kernel network fast-path
- XDP is a BPF hook that resides in-front of the network stack
- It provides the following operations:
  - Packet manipulation
  - Packet redirection
  - Packet drop
  - Monitoring
- XDP lacks packet scheduling capabilities!



XDP – eXpress Data Path

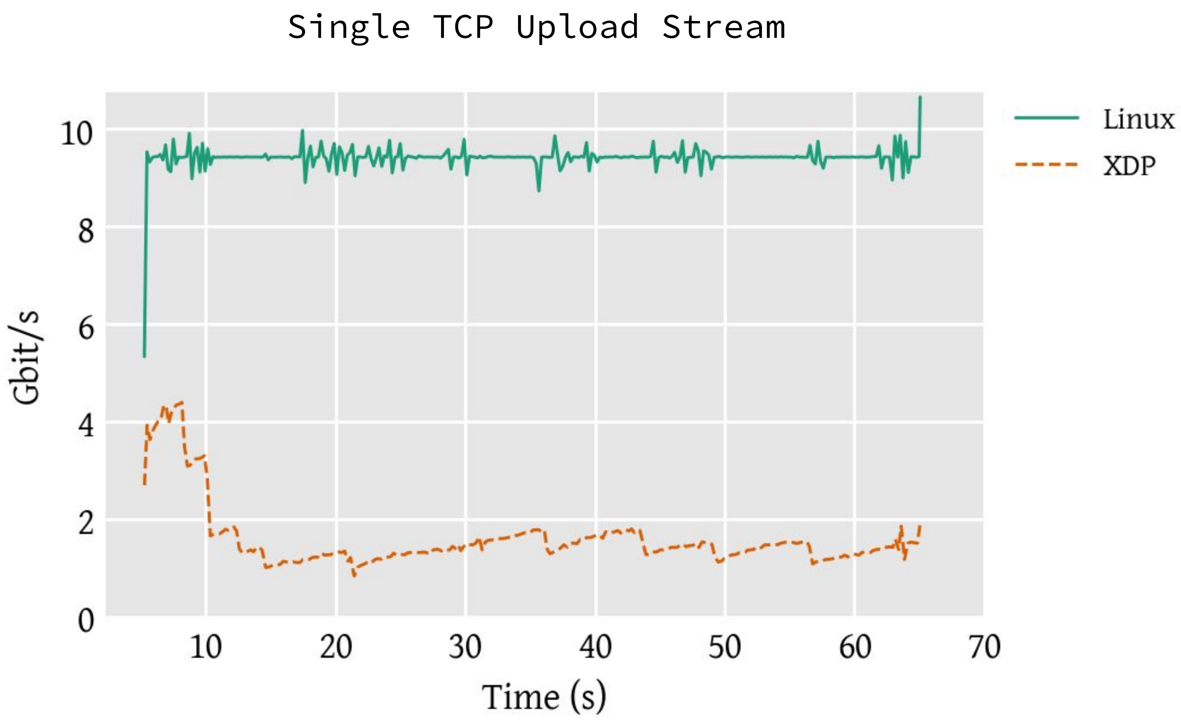
- The eXpress Data Path is an in-kernel network fast-path
- XDP is a BPF hook that resides in-front of the network stack
- It provides the following operations:
  - Packet manipulation
  - Packet redirection
  - Packet drop
  - Monitoring



- XDP lacks packet scheduling capabilities!
- Which the Linux kernel otherwise provides as Qdisc

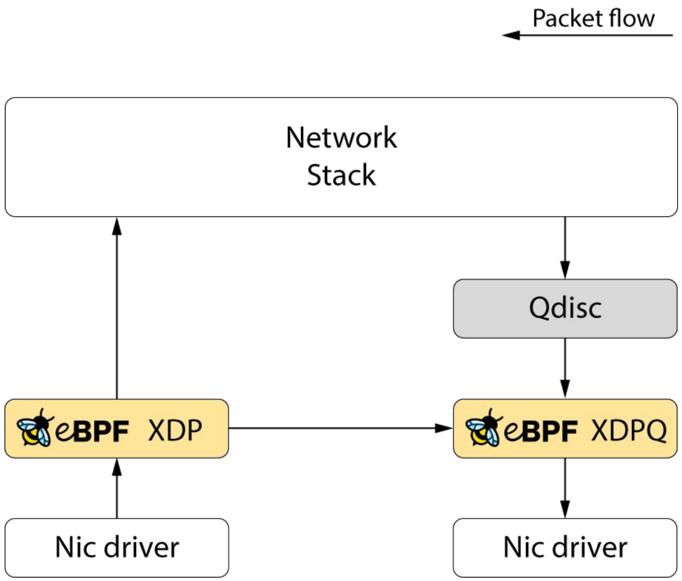
Simulation of the problem

- Test setup
  - 100 Gbps to 10 Gbps traffic
  - 10 ms propagation delay using netem



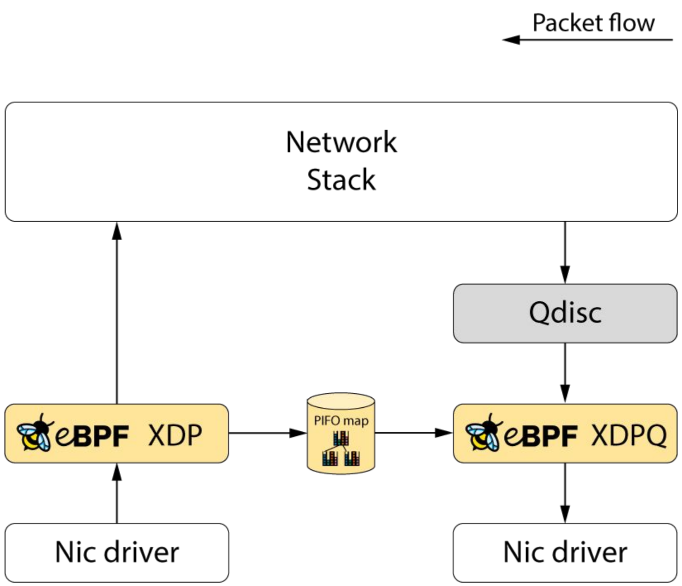
Packet Queuing for XDP

- We are adding programmable queuing capabilities to XDP by:
  - Providing a dequeue hook to XDP



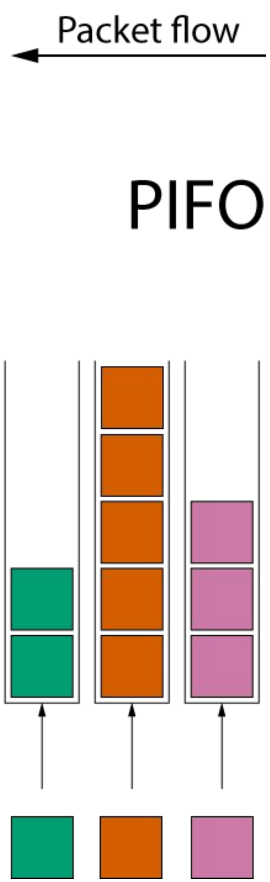
Packet Queuing for XDP

- We are adding programmable queuing capabilities to XDP by:
  - Providing a dequeue hook to XDP
  - Allowing XDP to redirect packets to a new BPF map scheduling data structure



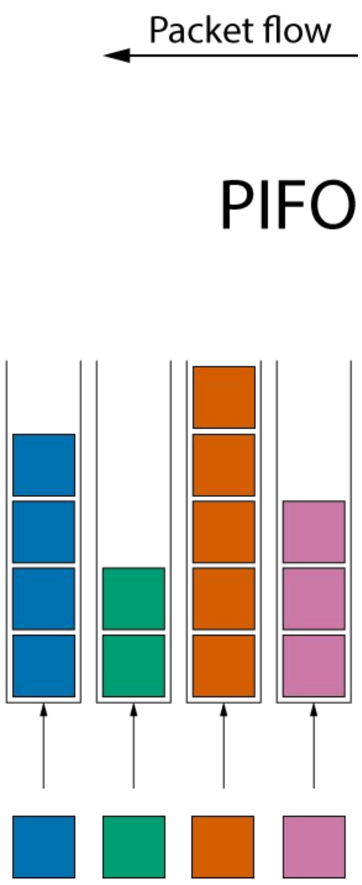
PIFO – Push-In First-Out

- PIFO is a data structure for programmable packet scheduling:



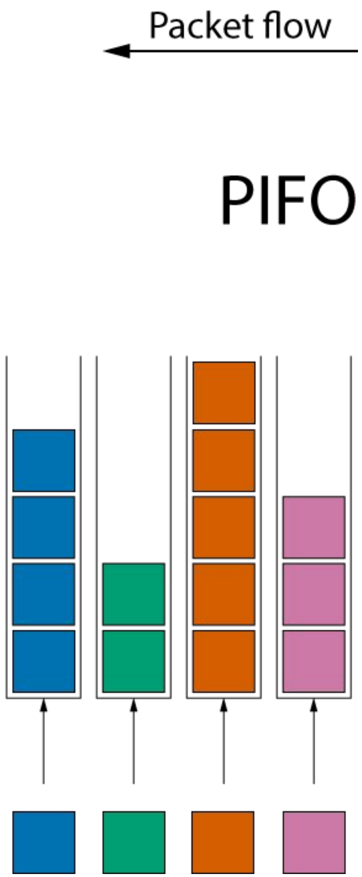
PIFO – Push-In First-Out

- PIFO is a data structure for programmable packet scheduling:
  - More known in the hardware world



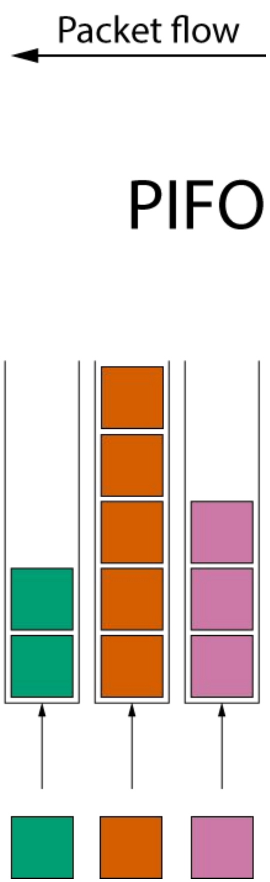
PIFO – Push-In First-Out

- PIFO is a data structure for programmable packet scheduling:
  - More known in the hardware world
  - A PIFO is a set of queues



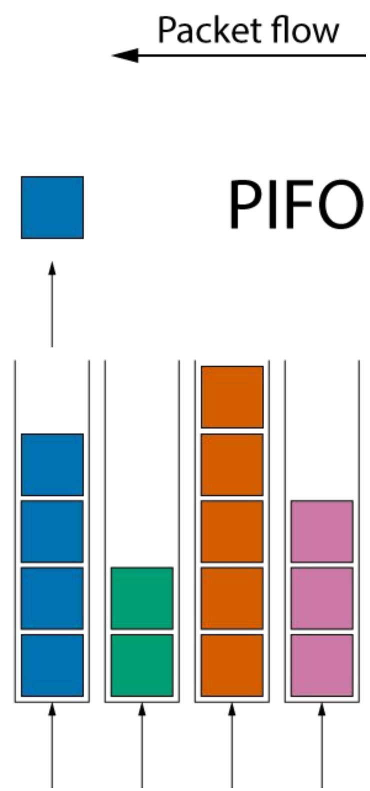
PIFO – Push-In First-Out

- PIFO is a data structure for programmable packet scheduling:
  - More known in the hardware world
  - A PIFO is a set of queues
  - Packets can be pushed in any order into the queues



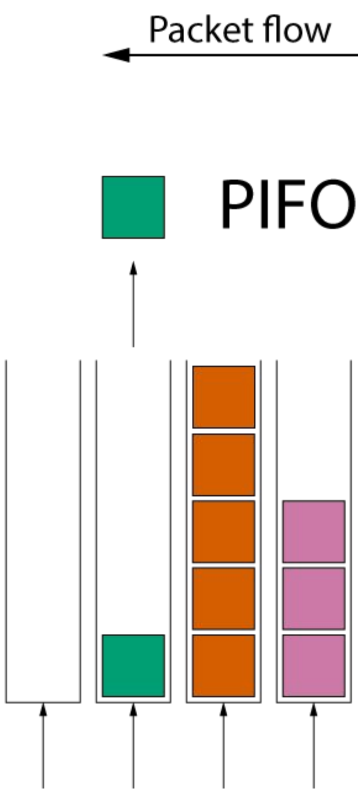
PIFO – Push-In First-Out

- PIFO is a data structure for programmable packet scheduling:
  - More known in the hardware world
  - A PIFO is a set of queues
  - Packets can be pushed in any order into the queues
  - However, packets can only be retrieved from the head of the data structure



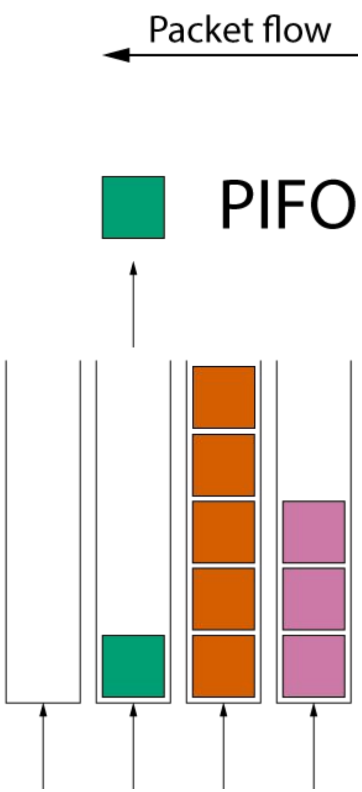
PIFO – Push-In First-Out

- PIFO is a data structure for programmable packet scheduling:
  - More known in the hardware world
  - A PIFO is a set of queues
  - Packets can be pushed in any order into the queues
  - However, packets can only be retrieved from the head of the data structure



PIFO – Push-In First-Out

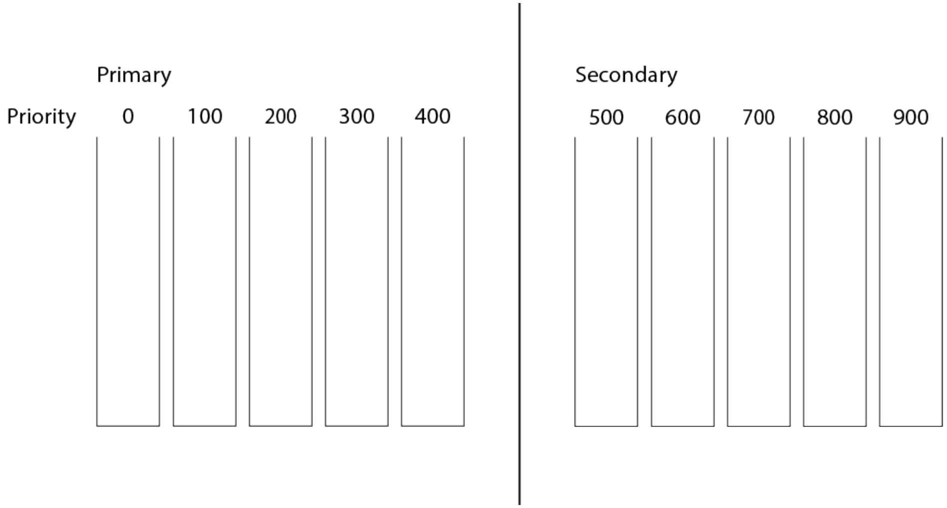
- PIFO is a data structure for programmable packet scheduling:
  - More known in the hardware world
  - A PIFO is a set of queues
  - Packets can be pushed in any order into the queues
  - However, packets can only be retrieved from the head of the data structure
  - PIFOs do not allow rearranging the packets after queueing them



Implementation notes: Eiffel extensions to PIFO

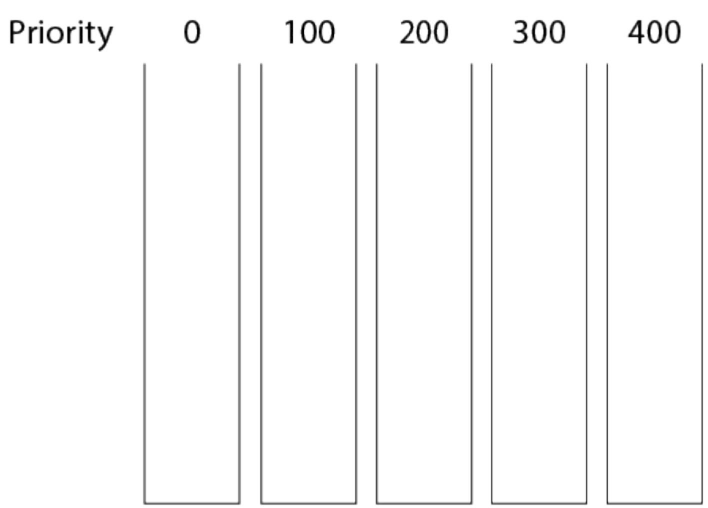
- PIFOs can queue flows and other data structures
  - A flow could be a FIFO

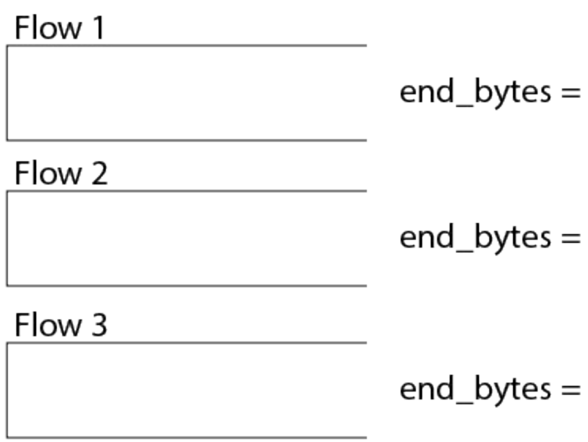
- A PIFO can internally cycle between two PIFOs for schedulers with increasing priorities



Fair Queuing (FQ)

```
// FQ scheduling algorithm
if pkt.flow_id in flows:
   flow = get_flow(pkt.flow_id, flows);
   prio = max(time_bytes, flow.end_bytes);
else:
   flow = new_flow(pkt, time_bytes);
   add_flow(flow, flows);
   prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```

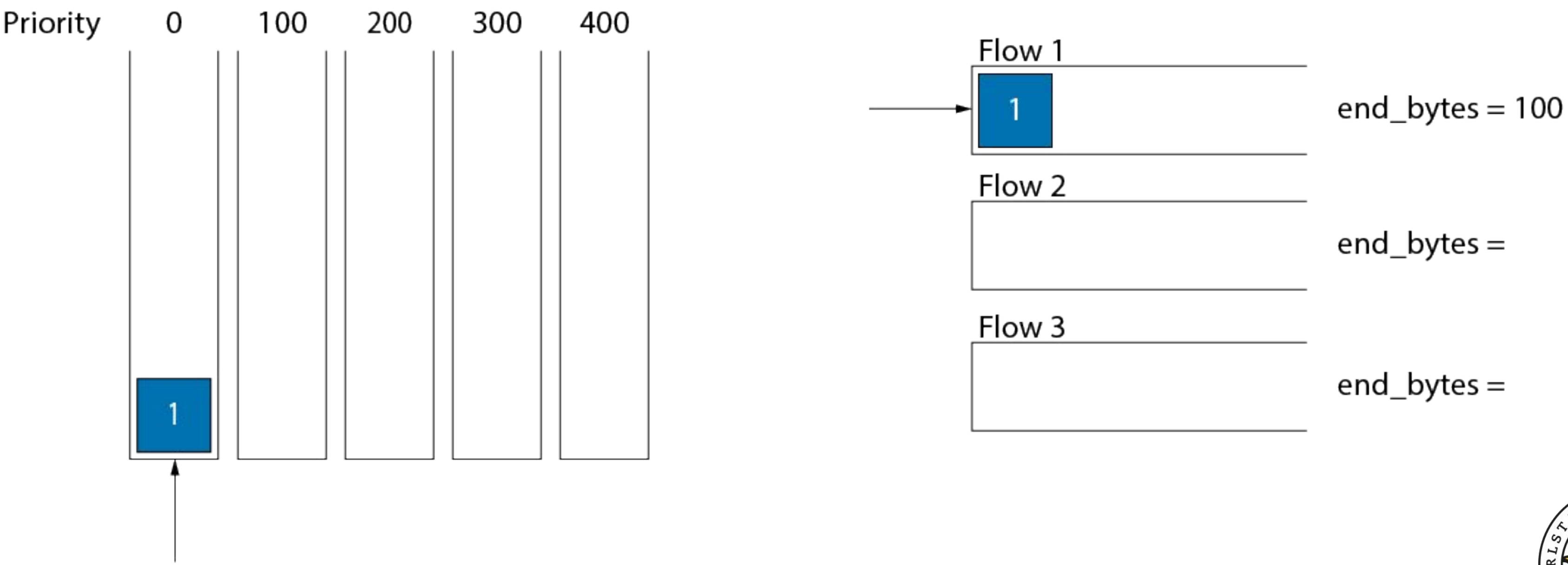





```
time_bytes = 0
```

Fair Queuing (FQ)

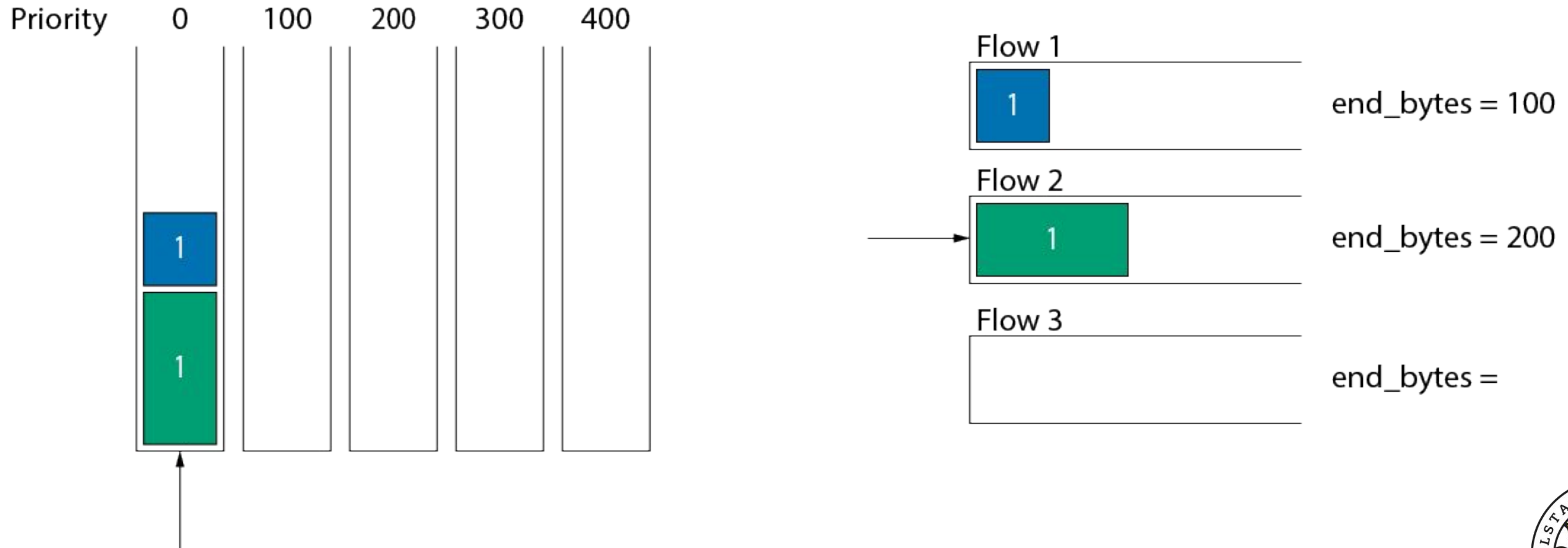
```
// FQ scheduling algorithm
if pkt.flow_id in flows:
   flow = get_flow(pkt.flow_id, flows);
   prio = max(time_bytes, flow.end_bytes);
else:
   flow = new_flow(pkt, time_bytes);
   add_flow(flow, flows);
   prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```



time_bytes = 0

Fair Queuing (FQ)

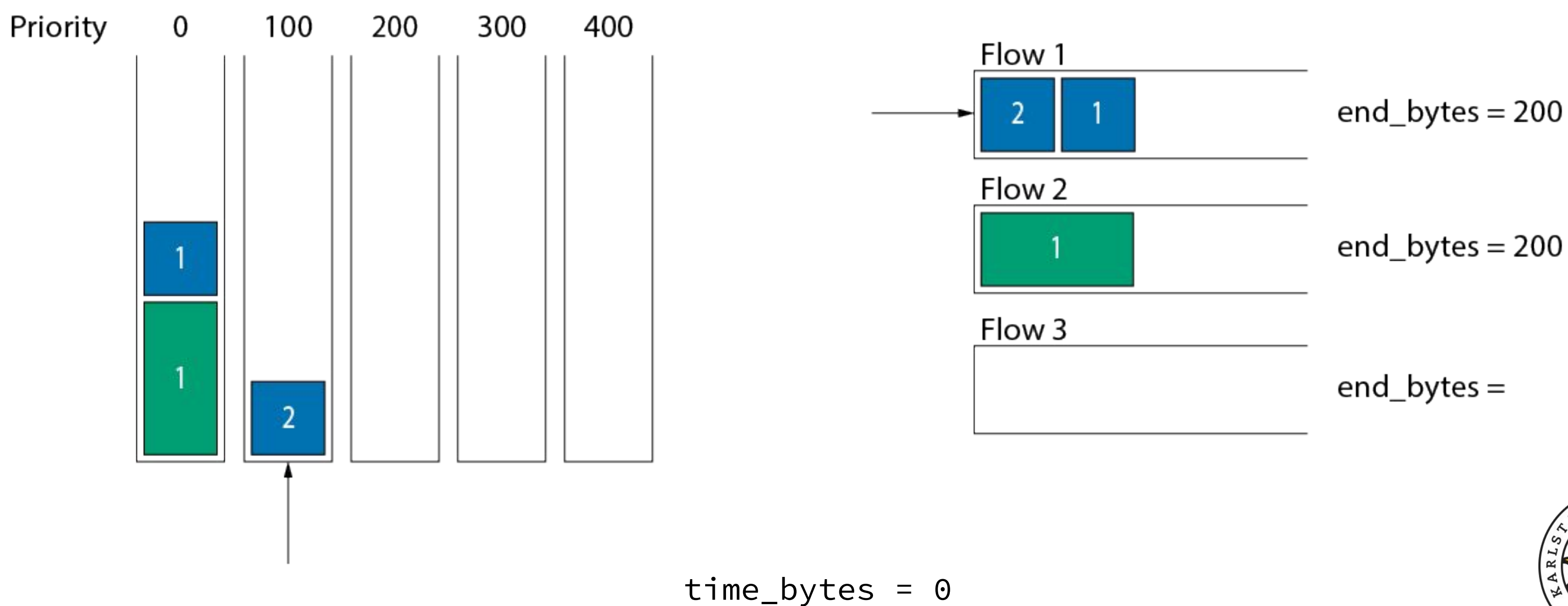
```
// FQ scheduling algorithm
if pkt.flow_id in flows:
    flow = get_flow(pkt.flow_id, flows);
    prio = max(time_bytes, flow.end_bytes);
else:
    flow = new_flow(pkt, time_bytes);
    add_flow(flow, flows);
    prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```




```
time_bytes = 0
```

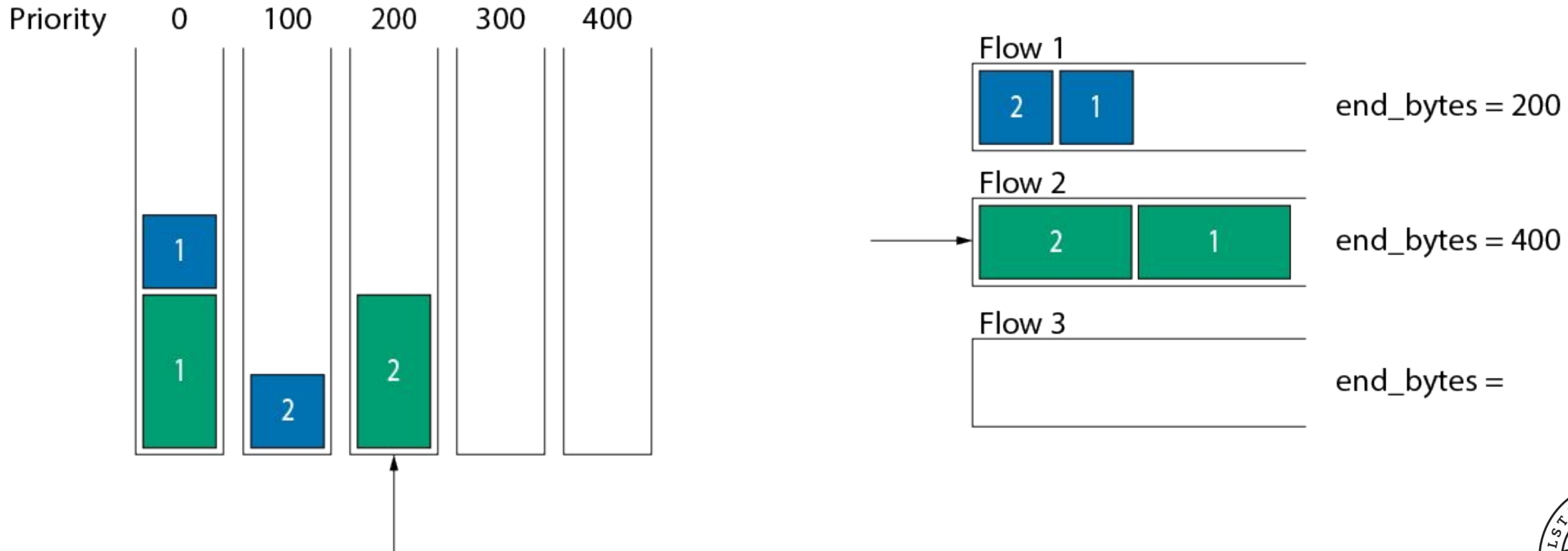
Fair Queuing (FQ)

```
// FQ scheduling algorithm
if pkt.flow_id in flows:
   flow = get_flow(pkt.flow_id, flows);
   prio = max(time_bytes, flow.end_bytes);
else:
   flow = new_flow(pkt, time_bytes);
   add_flow(flow, flows);
   prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```



Fair Queuing (FQ)

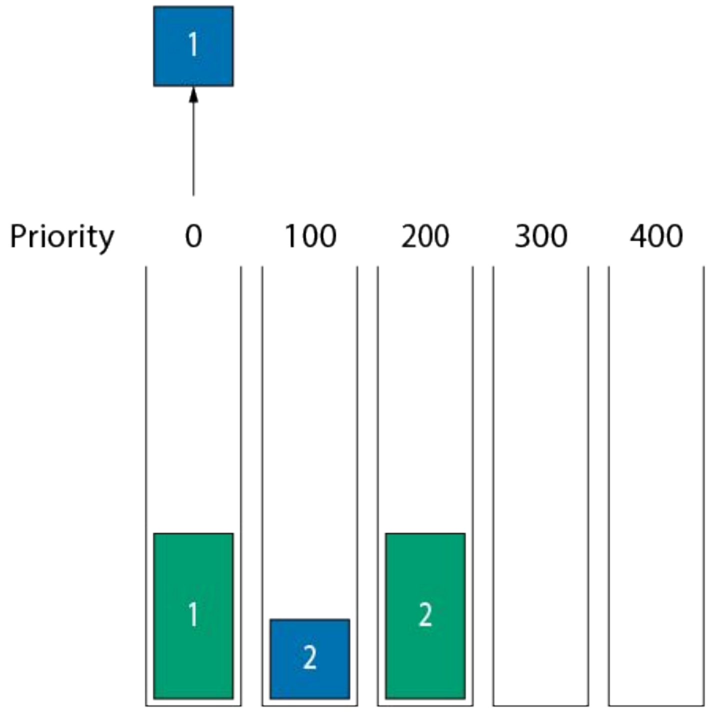
```
// FQ scheduling algorithm
if pkt.flow_id in flows:
   flow = get_flow(pkt.flow_id, flows);
   prio = max(time_bytes, flow.end_bytes);
else:
   flow = new_flow(pkt, time_bytes);
   add_flow(flow, flows);
   prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```

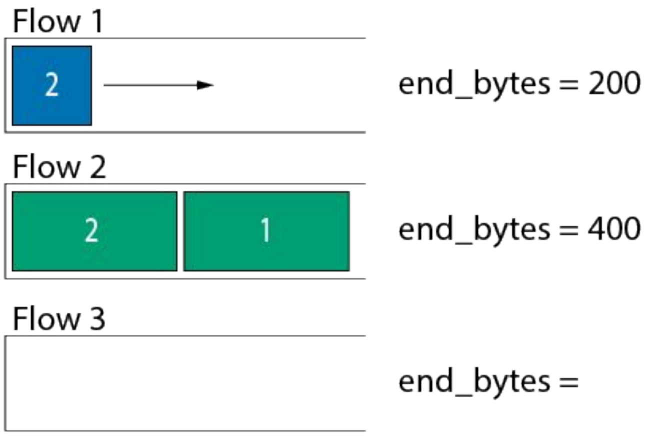


`time_bytes = 0`

Fair Queuing (FQ)

```
// FQ scheduling algorithm
if pkt.flow_id in flows:
   flow = get_flow(pkt.flow_id, flows);
   prio = max(time_bytes, flow.end_bytes);
else:
   flow = new_flow(pkt, time_bytes);
   add_flow(flow, flows);
   prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```

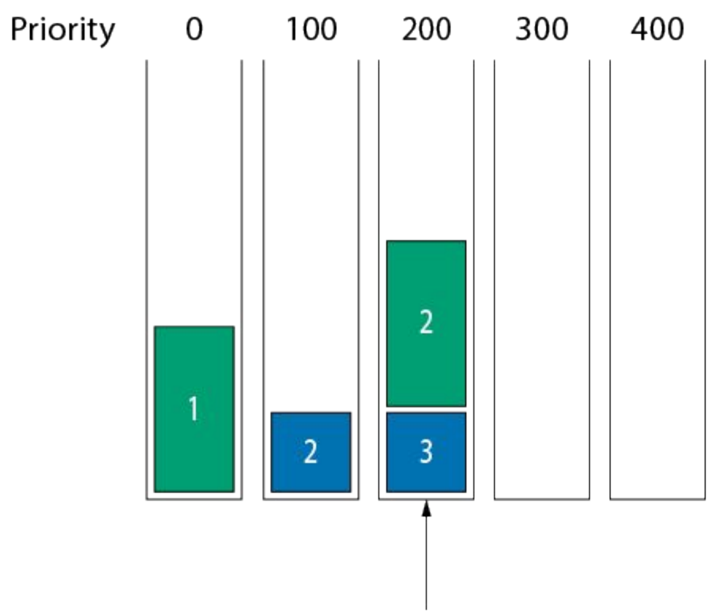


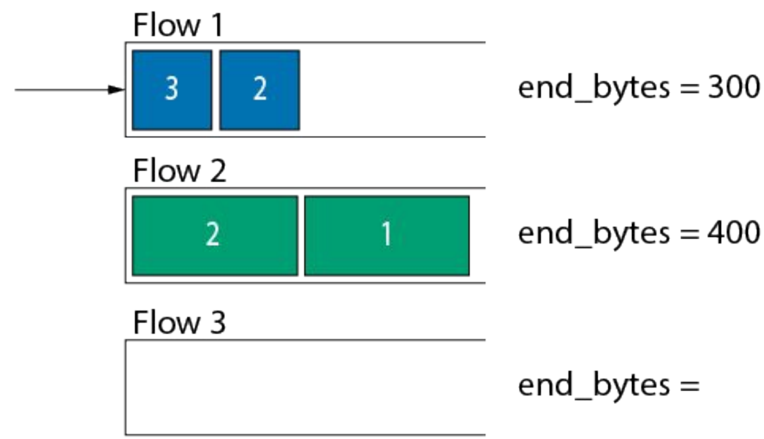


time_bytes = 0

Fair Queuing (FQ)


```
// FQ scheduling algorithm
if pkt.flow_id in flows:
   flow = get_flow(pkt.flow_id, flows);
   prio = max(time_bytes, flow.end_bytes);
else:
   flow = new_flow(pkt, time_bytes);
   add_flow(flow, flows);
   prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```

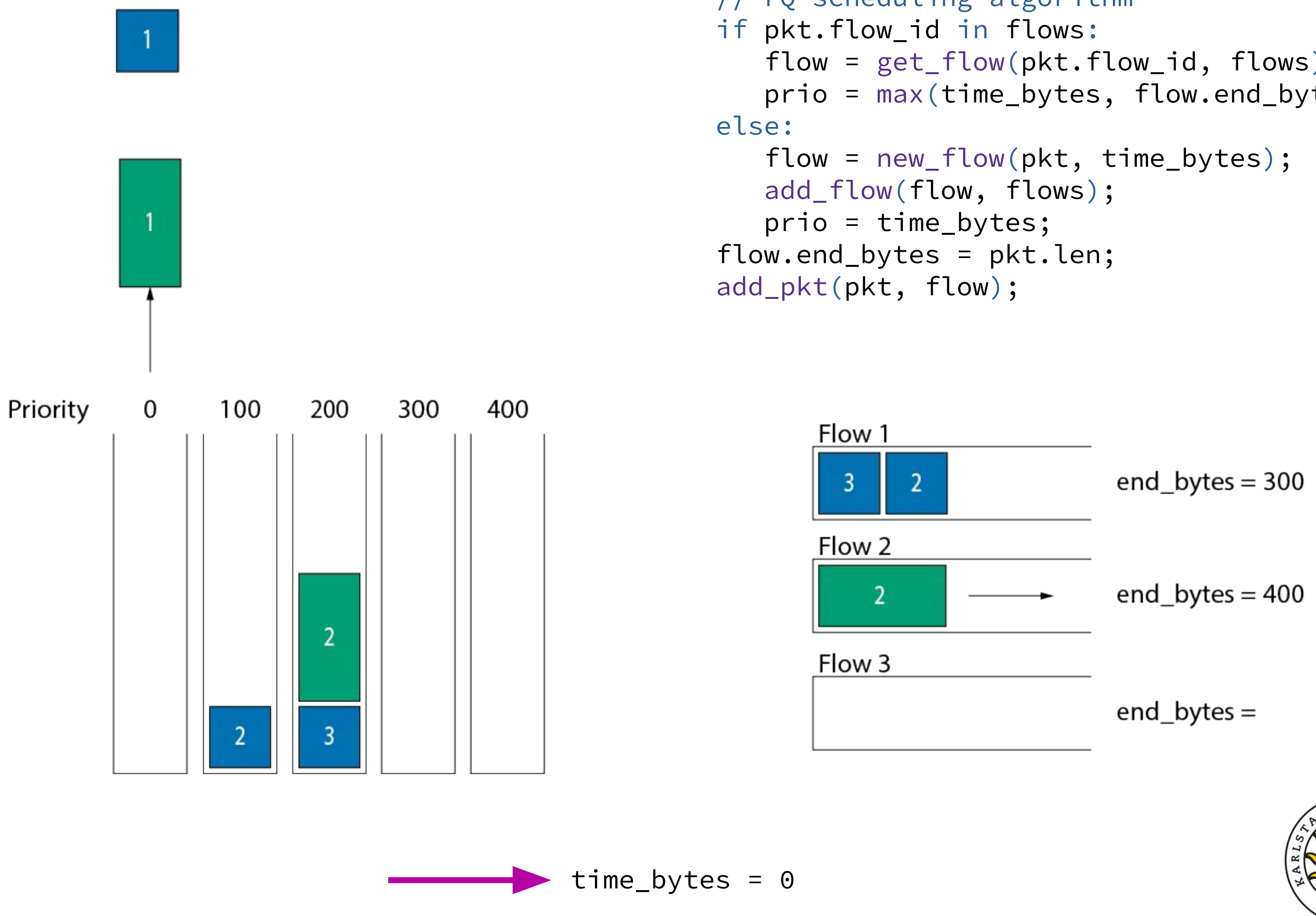




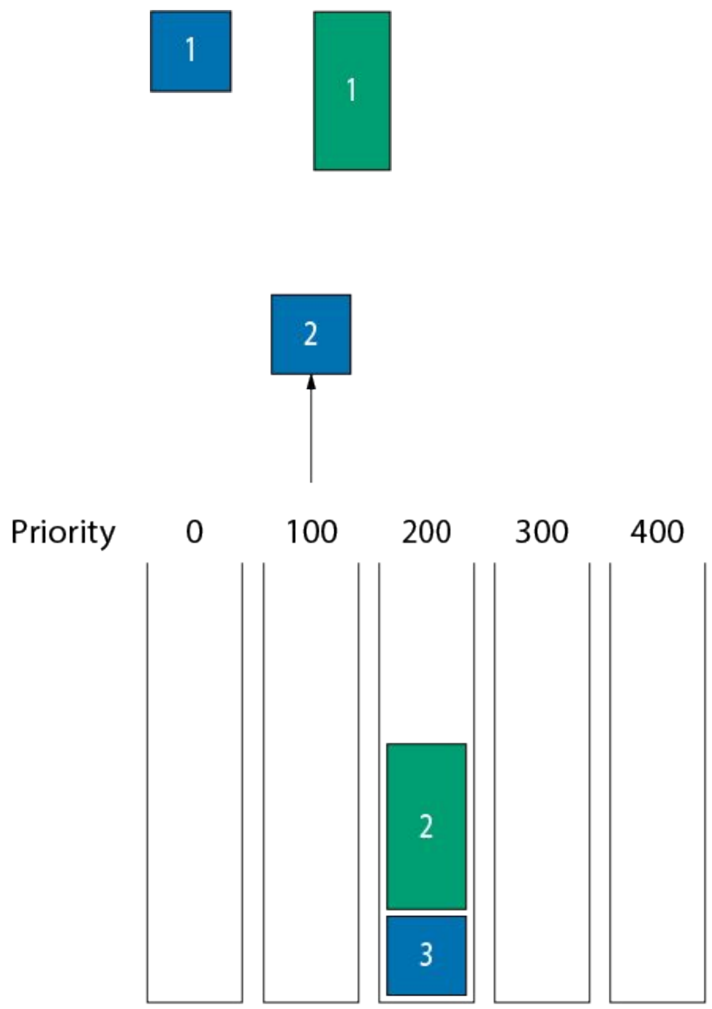
time_bytes = 0

Fair Queuing (FQ)

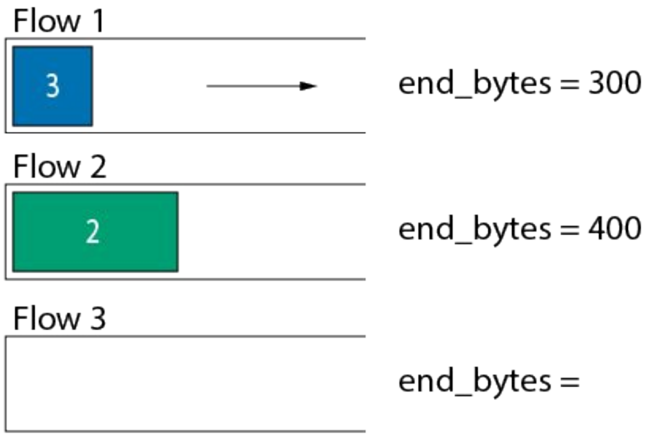
```
// FQ scheduling algorithm
if pkt.flow_id in flows:
   flow = get_flow(pkt.flow_id, flows);
   prio = max(time_bytes, flow.end_bytes);
else:
   flow = new_flow(pkt, time_bytes);
   add_flow(flow, flows);
   prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```



Fair Queuing (FQ)

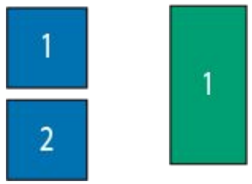



```
// FQ scheduling algorithm
if pkt.flow_id in flows:
   flow = get_flow(pkt.flow_id, flows);
   prio = max(time_bytes, flow.end_bytes);
else:
   flow = new_flow(pkt, time_bytes);
   add_flow(flow, flows);
   prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```

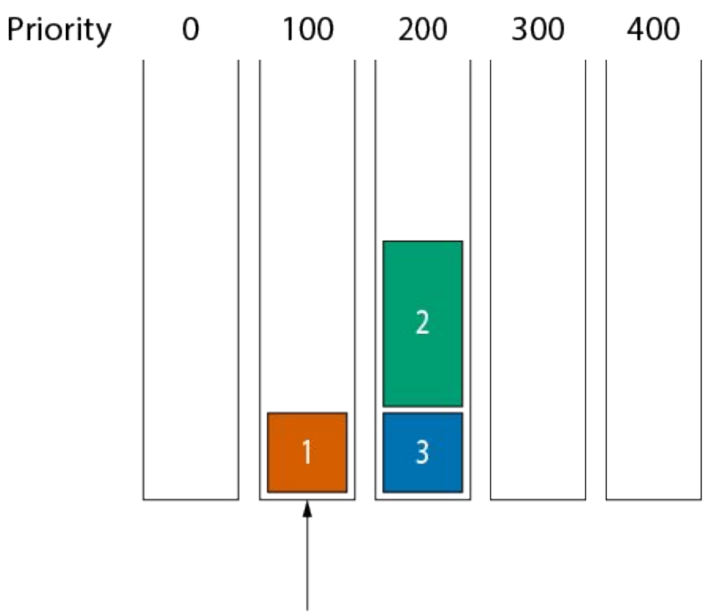


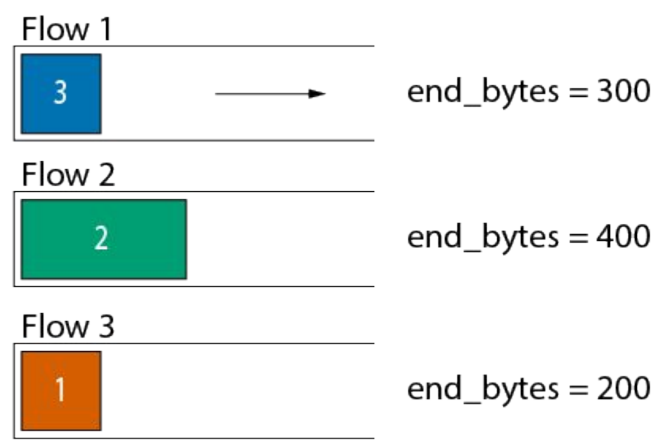
time_bytes = 100

Fair Queuing (FQ)




```
// FQ scheduling algorithm
if pkt.flow_id in flows:
   flow = get_flow(pkt.flow_id, flows);
   prio = max(time_bytes, flow.end_bytes);
else:
   flow = new_flow(pkt, time_bytes);
   add_flow(flow, flows);
   prio = time_bytes;
flow.end_bytes = pkt.len;
add_pkt(pkt, flow);
```

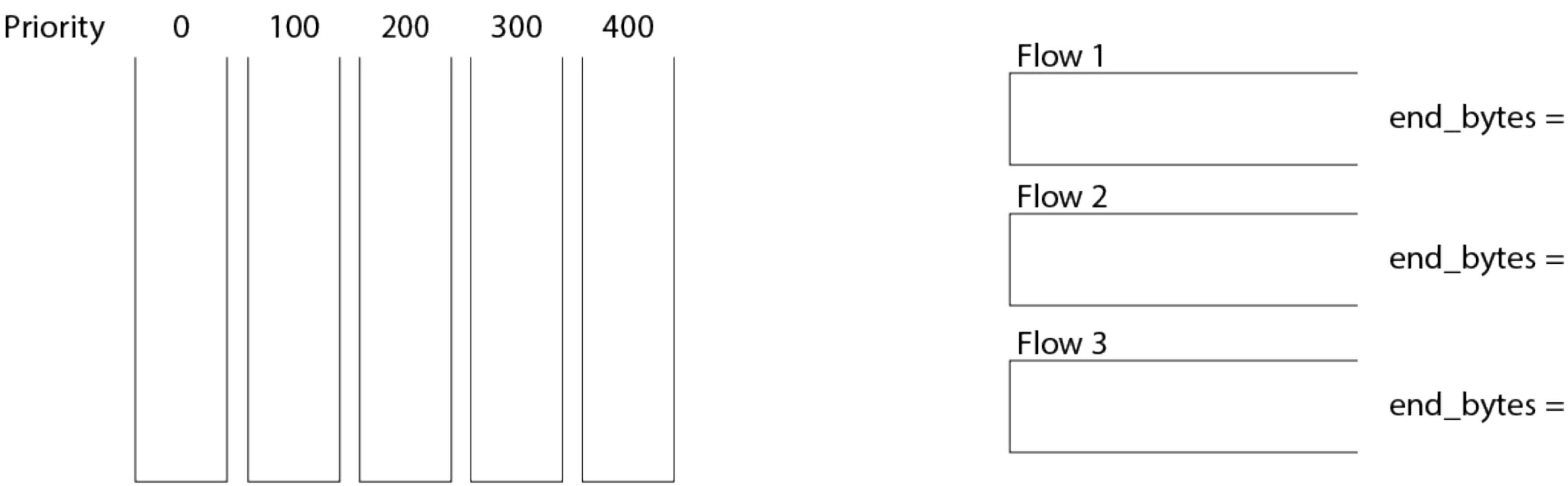




time_bytes = 100

Weighted Fair Queuing (WFQ)

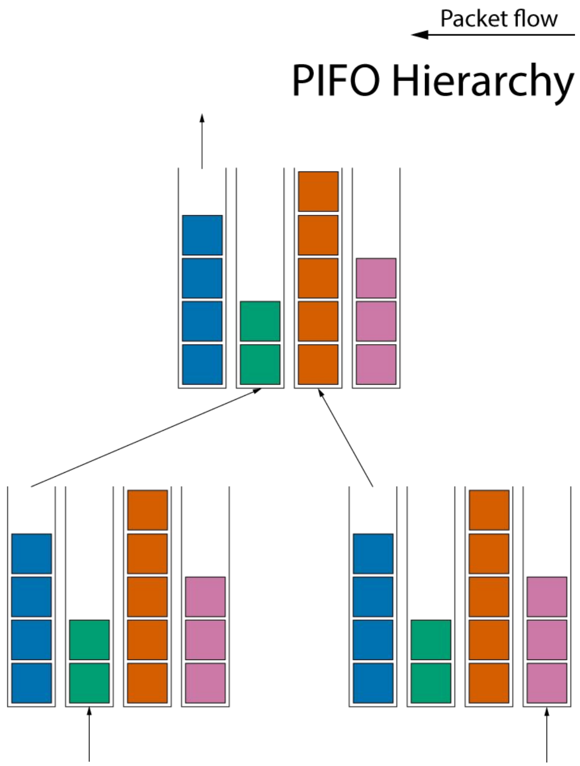
```
// WFQ scheduling algorithm
if pkt.flow_id in flows:
    flow = get_flow(pkt.flow_id, flows);
    prio = max(time_bytes, flow.end_bytes);
else:
    flow = new_flow(pkt, time_bytes);
    add_flow(flow, flows);
    prio = time_bytes;
flow.end_bytes = pkt.len / flow.weight;
add_pkt(pkt, flow);
```




```
time_bytes = 0
```

PIFO Hierarchy

- More complex packet scheduling algorithms can be constructed by creating a hierarchy of PIFOs

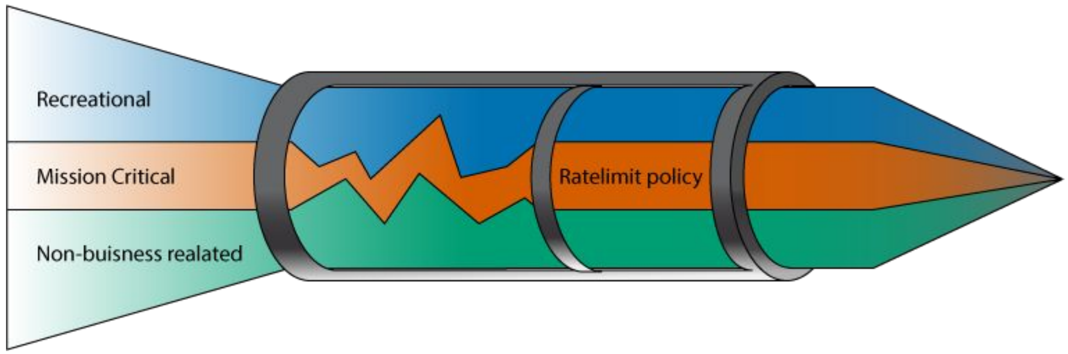


Future Work

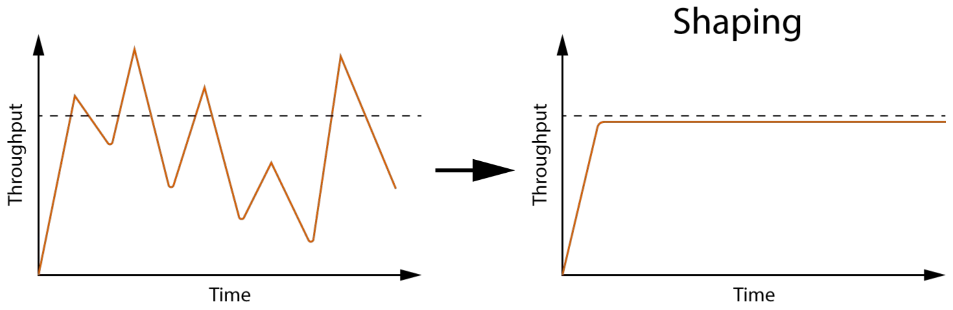
- Compare

Policing and Shaping

- Rate limiting a different type of packet scheduling practice were throughput is capped:

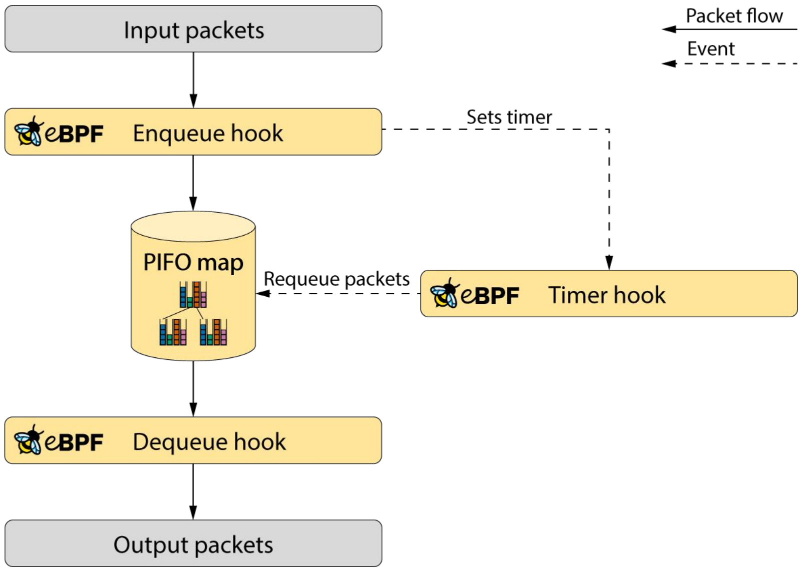


- Shaping algorithms rely on timers and have the capability of delaying packets.



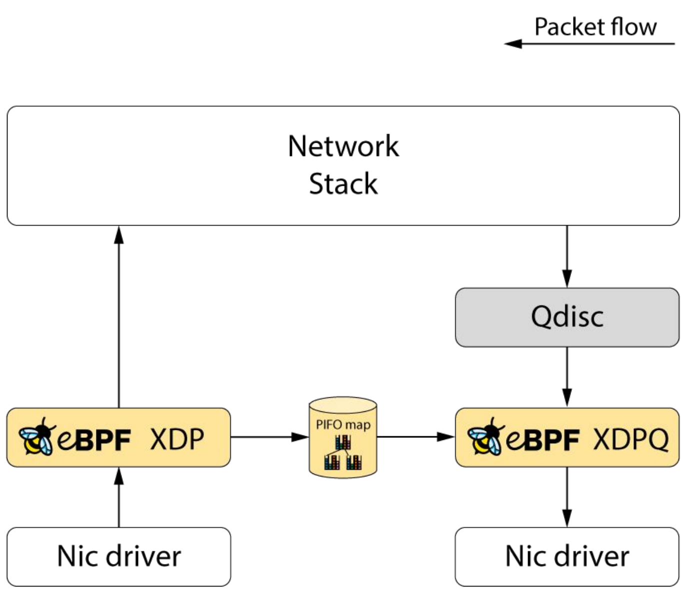
XDP queuing with shaping

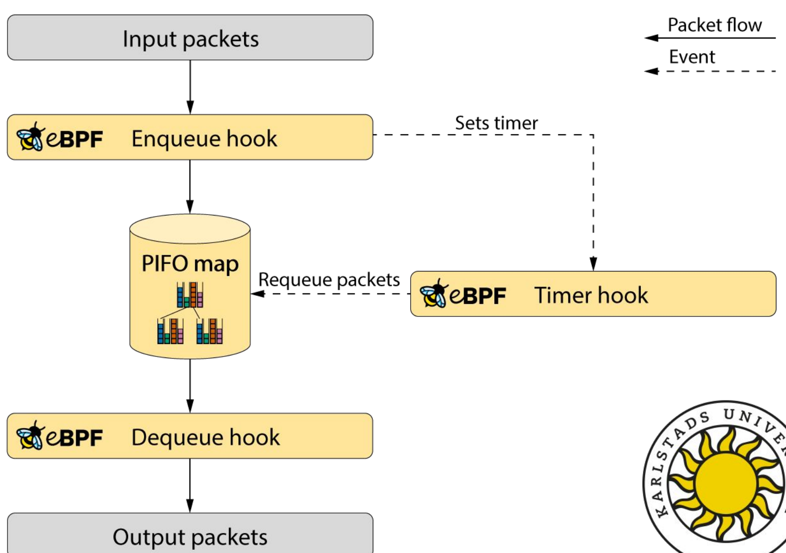
- We intend to provide shaping in the future using the new BPF timer API
  - XDP hook enqueues packets into a delay PIFO
  - Using the timer hook, we can requeue the delayed packets from the delay PIFO and into the active PIFO



Summary

- Bringing packet queueing to XDP
  - We are adding programmable packet scheduling capabilities to XDP by providing:
    - A new XDP dequeue hook
    - A new BPF PIFO map
  - Future work is adding shaping though BPF timers





Summary

- New XDP Dequeue hook and PIFO map:
  - https://git.kernel.org/pub/scm/linux/kernel/git/toke/linux.git/log/?h=xdp-queueing-05
- Scheduler examples and testing framework will be available at:
  - https://github.com/xdp-project/bpf-examples
- Papers:
  - PIFO: Sivaraman, Anirudh, et al. “Programmable packet scheduling at line rate”
  - Eiffel extension: Saeed, Ahmed, et al. “Eiffel: Efficient and flexible software packet scheduling”