Platform for programmable heterogeneous virtual middleboxes
The team

Laurent Mathy *(Supervisor)*
Cyril Soldani *(Language / code analysis / private NFV)*
Emmanouil Psanis *(Synchronization)*
Me, Tom Barbette *(System / Packet Processing)*
Stir network innovation

Deploy middleboxes in the Cloud
Platform

It needs to be

- **flexible** (for various current and future middleboxes)
- **secure** (isolation between tenants w.r.t data and resources)
- **fast** (low latency, high throughput)
- **easy to use**
Heterogeneous

We develop a platform for programmable heterogeneous virtual middleboxes.
Virtual

**Consolidate** middleboxes
- Full-VM not the right abstraction for performance
- Fine grained (eg. Tailtrie: multiple FIB on GPU with partial similar data)

**Migrate** them between the underlying hardware
- Virtualize data structures for heterogeneous hardware (Tile, NetFPGA, GPU, x86)

We develop a platform for programmable heterogeneous virtual middleboxes
What will be the platform?
Why not?

- General purpose
- Have all the needed *softwares*
- Provide *isolation* through process mechanism
- Paper about running things on *GPU, FPGA, ...*
Kernel Network Stack is Slow

“Too much” general purpose
Lot of systems calls to receive and send packets
User-space <-> Kernel Space copy
First work

Lot of I/O frameworks available

<table>
<thead>
<tr>
<th>Framework</th>
<th>Packet_mmap</th>
<th>PacketShaper I/O</th>
<th>NetSlices</th>
<th>Netmap</th>
<th>PF_RING ZC</th>
<th>DPDK</th>
<th>OpenOnload</th>
</tr>
</thead>
<tbody>
<tr>
<td>No user-space copy</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Multiqueue</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>I/O Batching</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Kernel bypass</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Zero-copy forwarding</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Zero-copy buffering</td>
<td>X</td>
<td>X</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>Devices family supported</td>
<td>ALL</td>
<td>1</td>
<td>ALL</td>
<td>8</td>
<td>4 ZC / ALL (non-ZC)</td>
<td>11</td>
<td>All SolarFlare</td>
</tr>
<tr>
<td>Pcap library</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>Header change</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Socket library</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Last kernel version supported</td>
<td>Last</td>
<td>Last</td>
<td>2.6.35</td>
<td>Last</td>
<td>Last</td>
<td>Last</td>
<td>Last</td>
</tr>
<tr>
<td>License</td>
<td>GPLv2</td>
<td>GPLv2</td>
<td>BSD</td>
<td>BSD</td>
<td>Proprietary</td>
<td>BSD</td>
<td>Proprietary</td>
</tr>
<tr>
<td>IXGBE version</td>
<td>Last</td>
<td>2.6.28</td>
<td>Last</td>
<td>Last</td>
<td>Last</td>
<td>Last</td>
<td>N/A</td>
</tr>
</tbody>
</table>

![Graph showing throughput vs packet size for different I/O frameworks](image-url)
Those **frameworks** deliver **RAW** packets **quickly** to **user-space**

but we need to **do something** with them ...
Click **modular router**

**Flexible**, easy-to-use, component-based configuration

Existing reusable **elements** for common networking functions

Available in **user-space** with some of the above frameworks

Possibility of **hardware offloading** of some functions
Second work

Enhance click

Multiple I/O Framework **integrations**, but perfectible
Globally enhance Click with **multiqueue** support, **batching**, **zero-copy**, better **multithreading**...
FastClick

• Review the need for high speed I/O frameworks more suited than kernel API and compared them

• Using proposed ideas and new ideas of our own, we showed that FastClick was fit for purpose as a high speed userspace packet processor and opens the door for implementation of middleboxes and NFV

• FastClick is available at http://fastclick.run.montefiore.ulg.ac.be/

Routing test case
Current work

Add **flow** support to FastClick

Middlebox functionalities needs it:

- Attack spitted across multiple fragmented packets
- HTTP Reconstruction for ad-removal
- Defacing detection
- Proxy-caching
- DPI

...
Flows

Classification of packet into micro-flows
   Eg. :
      - TCP micro-flows
      - IP Pair
      - Packets from one AS to another AS
      - ?

Wait for more packets mechanism
Flow-local storage
The Click way

- Multiple classifiers
- Multiple flow table (each element has a hash table to access a per-flow space)

Also “the industry” way: multiple VM, reclassifying each time
One big classifier

Flow Control Block

Flow rule data

Flow Local Storage

Space computed at configuration time

class DPIData  {
    StateMachine s;
    Packet* awaiting_packets;
}

class NAT Data  {
    U32 old_ip;
    U32 new_ip;
    u16 old_dport;
    u16 new_dport;
}
Flow detection
One big classifier
Wait for more packets mechanism

“Run to completion|buffer” execution model
Try to process one packet through all elements in the pipeline.
Either:
- Execution finish in an output element
- Element asks for more packets, keeping that one in a buffer (per sub-flow, per element)
Process next packet

→ Avoid the cost of a context switch we would have with a socket
Current work

- Flow system (nearly finished)
- Quick packet classification
  - Fast rule specialization
  - Multi-dimensional
  - Partially offloadable

After that:
Make click elements compatible with some hardware devices (GPU, NetFPGA, …)
Handle the migration of those elements
→ Optimisation problem, data consistency, …
Questions, (possibly) answers and discussion

A programmable middlebox platform that can run on top of “what's available”, with fine grained virtualization and efficient consolidation.