PACKET MANIPULATION TOOLS FOR LINUX

Eric Vial (evial@lasige.di.fc.ul.pt)

Lasige Navtalk Session 2016

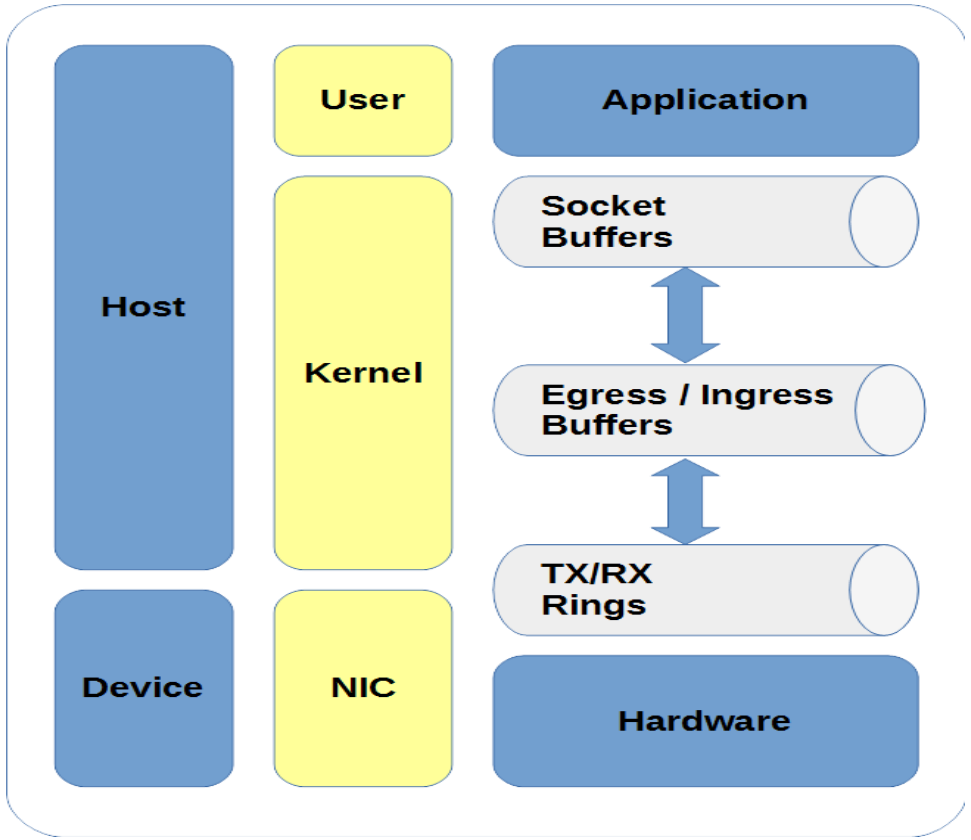
PLAN

1. Queuing mechanisms in Linux stack
2. Key principles of packet manipulation
3. Classification and example of tools
4. Usage in three European projects

INTRODUCTION AND MOTIVATION

- A packet manipulation tool allows to intercept, drop, duplicate, modify, forge, delay any Ethernet packets
- Packet manipulation has a wide range of use: from the protocol debugging to the test of MITM attacks.
- Manipulation tools are now widespread and no longer reserved for hacking purpose.
- Many user-made pieces of code are available from internet but many of them are highly specialized and hard to combine with other programs.
- Many software are based on no longer supported libraries. Low-level libraries require regular updates.

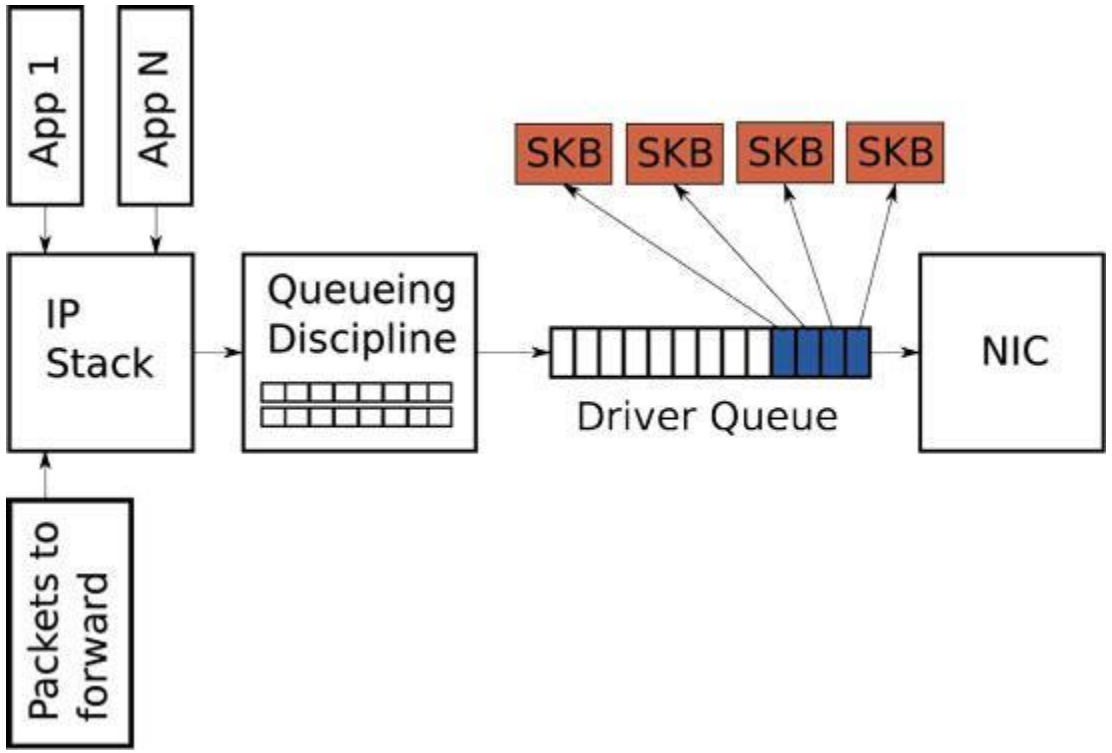
QUEUING IN LINUX STACK



- **Transmission**
  - Send socket buffer : use of write primitives to copy data, set_sockopt to set options.
  - Qdisc queue (Queuing Disciplines). FIFO queue by default.
  - TX ring queue shared between the NIC and the driver. FIFO queue.
- **Reception**
  - Receive socket buffer
  - Ingress buffer queue (only drop capability in case of congestion)
  - RX ring queue

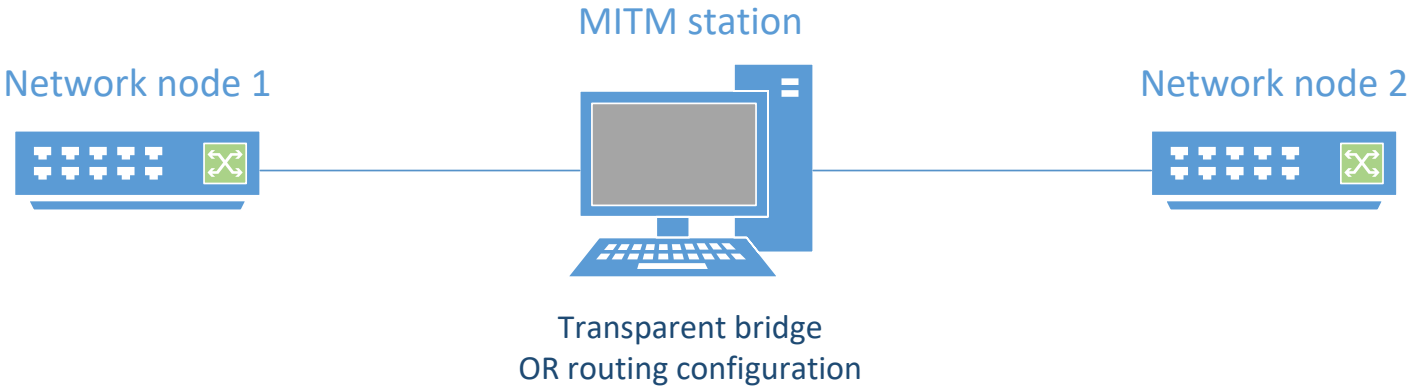
KERNEL PACKET DATAPATH

- Qdisc usual size is 1000 packets. Driver queue's size is driver-dependent (80 packets for e1000)
- Drivers queue does not contain packet data but descriptors which point to socket kernel buffers(SKBs).
- Communication between NIC and driver through Soft-IRQs, hard interruptions / NAPI (reception)
- Packet manipulation tools usually modify Qdisc and/or Driver queues behavior.



FIND THE RIGHT TEST ENVIRONMENT

- Packet manipulation could turn out complicated on controlled network environment where FWs or smart routers could detect and block abnormal packet contents or unusual traffic.
- Example of IP spoofing, many routers discard packets that couldn't possibly have originated from the interface where they are coming from.



CHOOSE YOUR STRATEGY

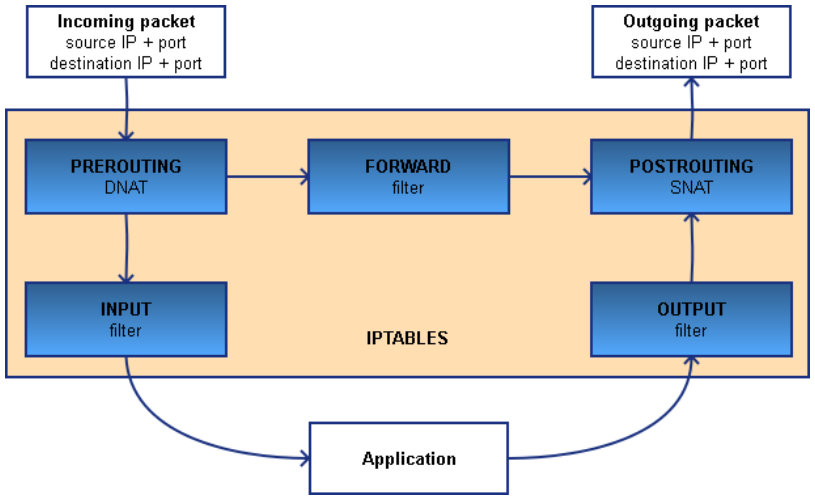
- **Option 1: Use Raw sockets**: build your Ethernet packet from scratch, add protocol layers, calculate checksums. Why is not sufficient?
  - It's very likely that the kernel security rules would block your packet if you create Ethernet/IP headers with unauthorised values.
  - Local FW will block a simple IP spoofing on most Linux distributions.
  - Disable security locks would turn out inefficient and time-consuming.
- **Option 2: Modify the Linux FW** : **Netfilter** is a framework composed by a set of kernel modules responsible for filtering and manipulating packets. **Nftables** has been introduced in latest versions (3.13 <) to replace Netfilter.
- **Option 3**: **Create your own kernel module** or use packet manipulation tools using kernel module insertion.

PLAY WITH THE LINUX FW

- **iptables** allows to configure the rule tables implemented in the Netfilter modules to manipulate the packets. **nft** is the similar tools for **nftables**.

```
iptables -t filter -A OUTPUT -j DROP -d 10.0.0.1
```

- **iptables** provides a way for the kernel modules to register callback functions with the kernel's networking stack (hook mechanisms).
- **Userspace libraries as libnetfilter-queue** offer API to manipulate packets which have been queued in the kernel filters.



CREATE A KERNEL MODULE

```
#include <linux/module.h>
#include <linux/kernel.h>

int init_module(void) {}
void cleanup_module(void) {}
```

- **modprobe**, **insmod**, **lsmod**, **rmmod** to manipulate modules in kernel
- Runs in kernel mode. Could lead to crash or even system damage if not coded properly.

- Short and stable code in kernel modules combined with application calls to module's API in user space provide safety and resilience but lead to some delay due to context switching.
- Module's code usually linked with Linux kernel sources. May require to recompile and reinstall the module in case of kernel update.

HARDWARE ACCELERATION WITH KERNEL MODULES

- Adapter uses offloading in order to reduce CPU load from the system, modern network adapters have offloading features which move some network processing load onto the network interface card.
- For example, the kernel can submit large (up to 64k) TCP segments to the NIC, which the NIC will then break down into MTU-sized segments. This particular feature is called TCP Segmentation Offload (TSO).
- Offloading settings are managed by the command "**ethtool**".
- A large number of offloading features are locked in NIC's driver (e.g. vlan) and cannot be performed directly by the kernel module.

CLASSIFICATION OF TOOLS

- **Packet sniffer**

  Intercept and store traffic without modifying its content or affecting its behaviour.

- **Traffic shaper**

  Emulate the physical properties of a network link: bandwidth, latency and packet loss rate.

- **Packet crafter**

  Create and modify packets in order to test and perform fuzzing techniques. Unlike sniffing, crafting requires to bypass or block the kernel security mechanisms.

Most manipulation tools combine two or even three features.

PACKET SNIFFING TOOLS

- Well-known **TCPDUMP** and **Wireshark** are based on **Libpcap**

```
tcpdump -i eth0 '(icmp[0] = 0) and (icmp[4:2] = 0x1f4)
```

- **Libpcap** does not allow packet interception or modification. **Libpcap's** API is easy to used and implemented in a large number of sniffing tools as a low-level library.
- Large choice of sniffer tools (most of them based on libpcap). Some of them are <u>hacking-oriented</u>: **ngrep**, **EtherApe, dsniff** …

TRAFFIC SHAPING TOOLS

- **TC** allows to manipulate the traffic control settings (Qdisc queue).

```
tc qdisc add dev eth0 root netem delay 100ms
```

- **DummyNet**

```
ipfw add pipe 2 in proto tcp
ipfw add pipe 3 in proto udp
```

```
ipfw pipe 2 config bw 2Mbit/s
ipfw pipe 3 config bw 300Kbit/s
```

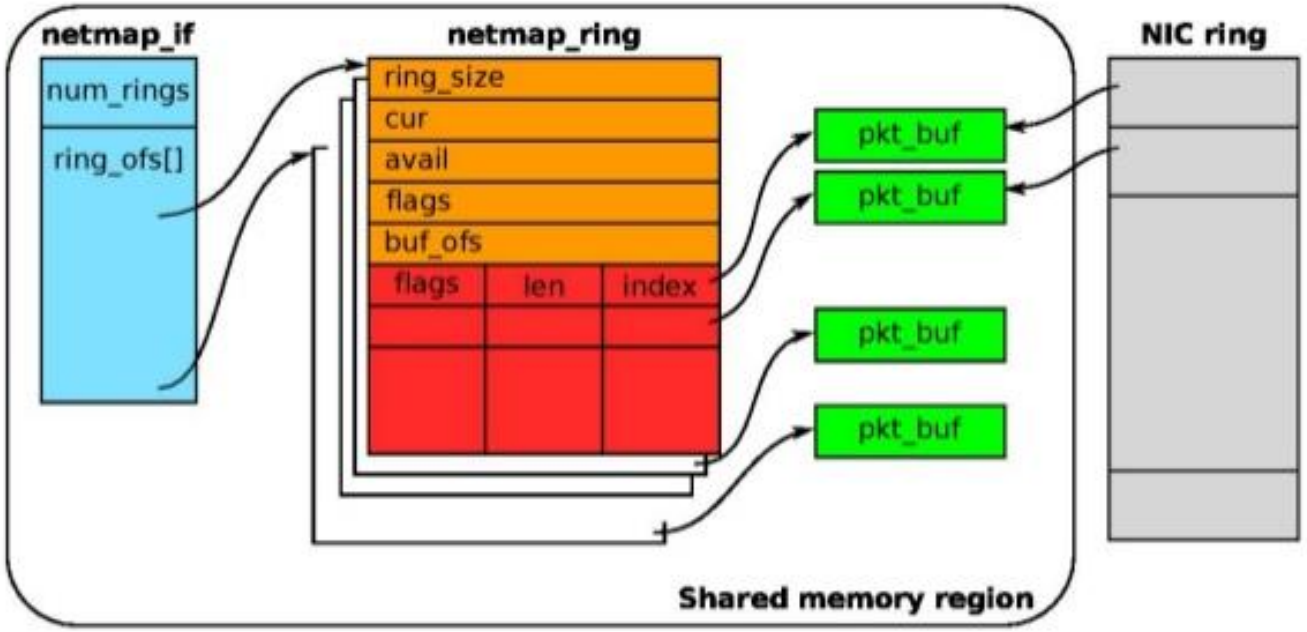
  - Based on kernel modules. Available by default in some Linux distributions.
  - Original version developed until 2012.
  - From 2013, new version based on **Netmap** running at 6.5 Mpps.

PACKET CRAFTING TOOLS

- **Raw sockets**: limited use.
- **Ettercap** : suite for MITM attacks.
- **Scapy** : interactive tool written in Python able to forge and decode packets of a wide number of protocols.
- **Netfilter / Nftables** : all libraries able to create callback functions in FW kernel modules and handle queued packets (e.g. libnetfilter-queue)
- **Netmap**: low-level C library to manipulate packets

NETMAP

- Developed by Luigi Rizzo (University of Pisa, Italy) like DummyNet

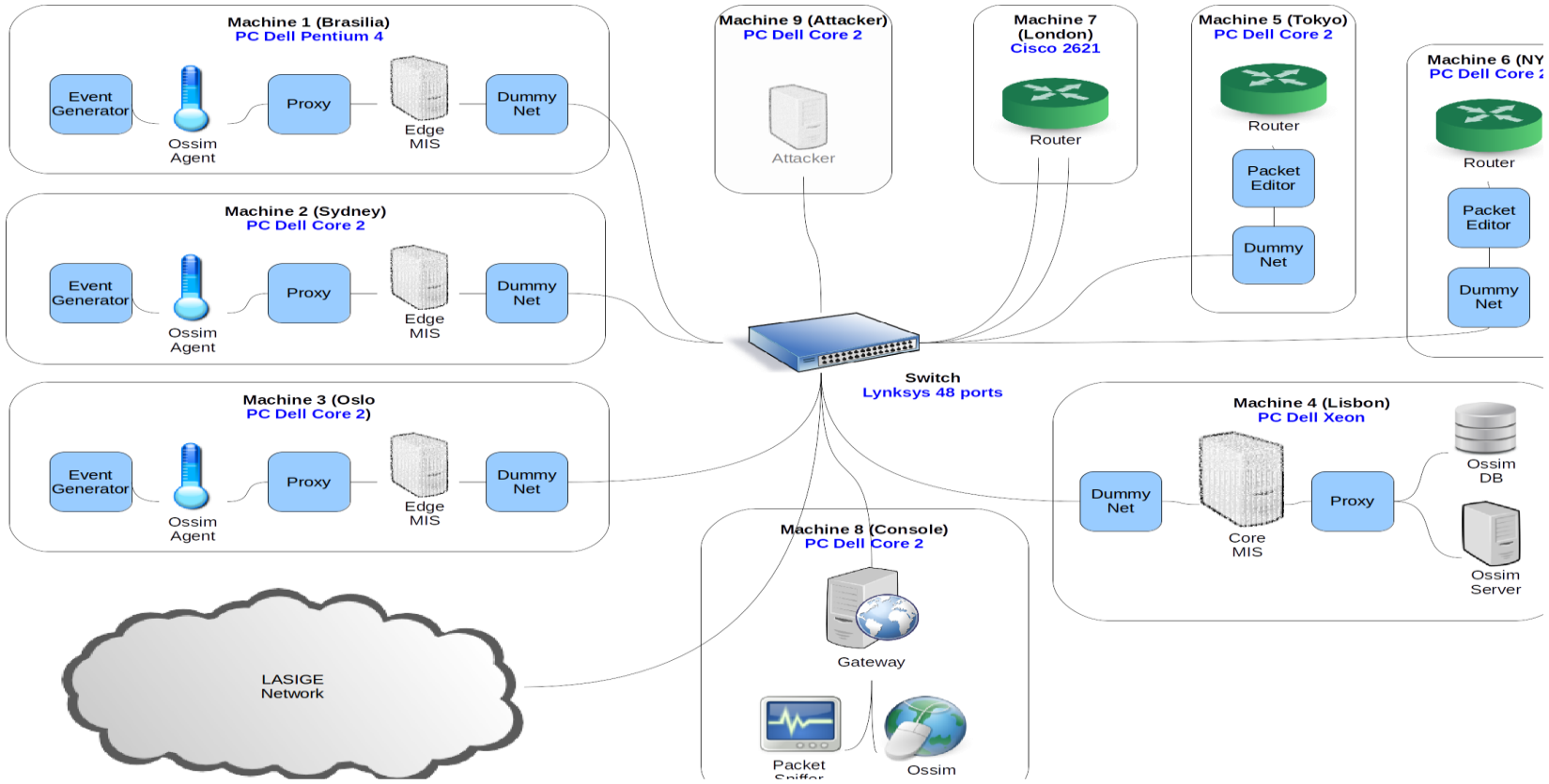


- Shadow copy of the NIC's ring (netmap ring)
- "poll" method to synchronize buffers with NIC's ring.
- 15Mpps with 10Gbs interface

TOOLS IMPLEMENTATION IN EUROPEAN PROJECTS

- **MASSIF** project
  - Dummynet
- **KARYON** project
  - Netplay 1 based on libnetfilter_queue
- **SEGRID** project
  - Netplay 2 based on Netmap

MASSIF TESTBED ARCHITECTURE



NETPLAY 1.0 OVERVIEW

- Based on **libnetfilter-queue** library and **iptables**

```
iptables -t mangle -A PREROUTING -m mark --mark 0xe -j ACCEPT
iptables -t mangle -A PREROUTING -p all -j NFQUEUE --queue-num 0
```

- Uses a text definition language to describe both the filter rules and the actions to be performed if a packet is matched: save, forward, modify, drop, delay, inject (before or after IP defragmentation).
- Can keep data from previous matching and modify actions at run-time
- Provides flexibility but remains slow compared to full-compiled tools
- Ethernet header modification not supported.
- Latest **libnetfilter-queue** version in 2012.

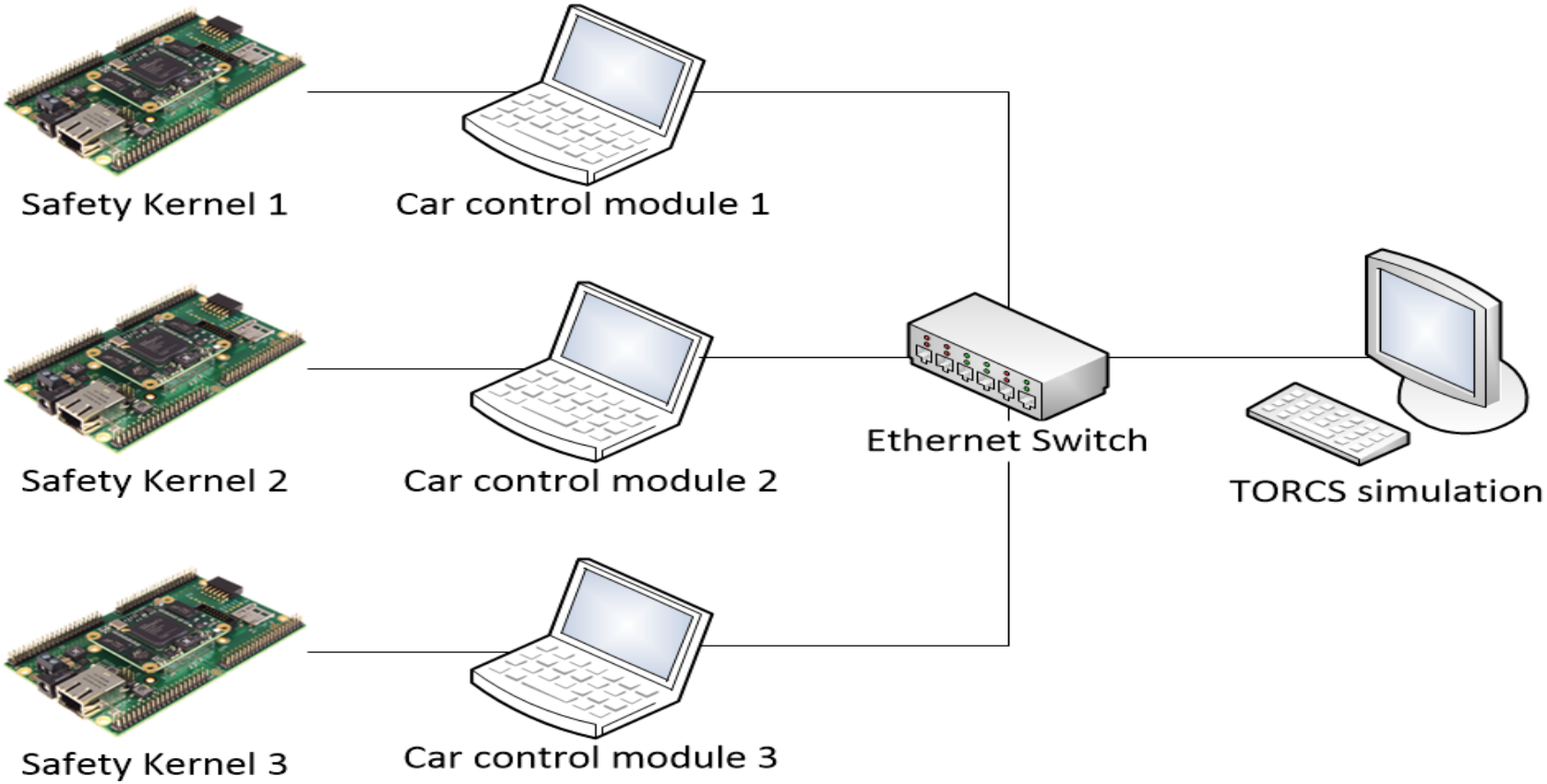
NETPLAY 1.0
RULE AND ACTION DEFINITION

```
# Example1 : dropping one from three IP packets
ip {
     id
     equal id%3 0 { drop }
}
```

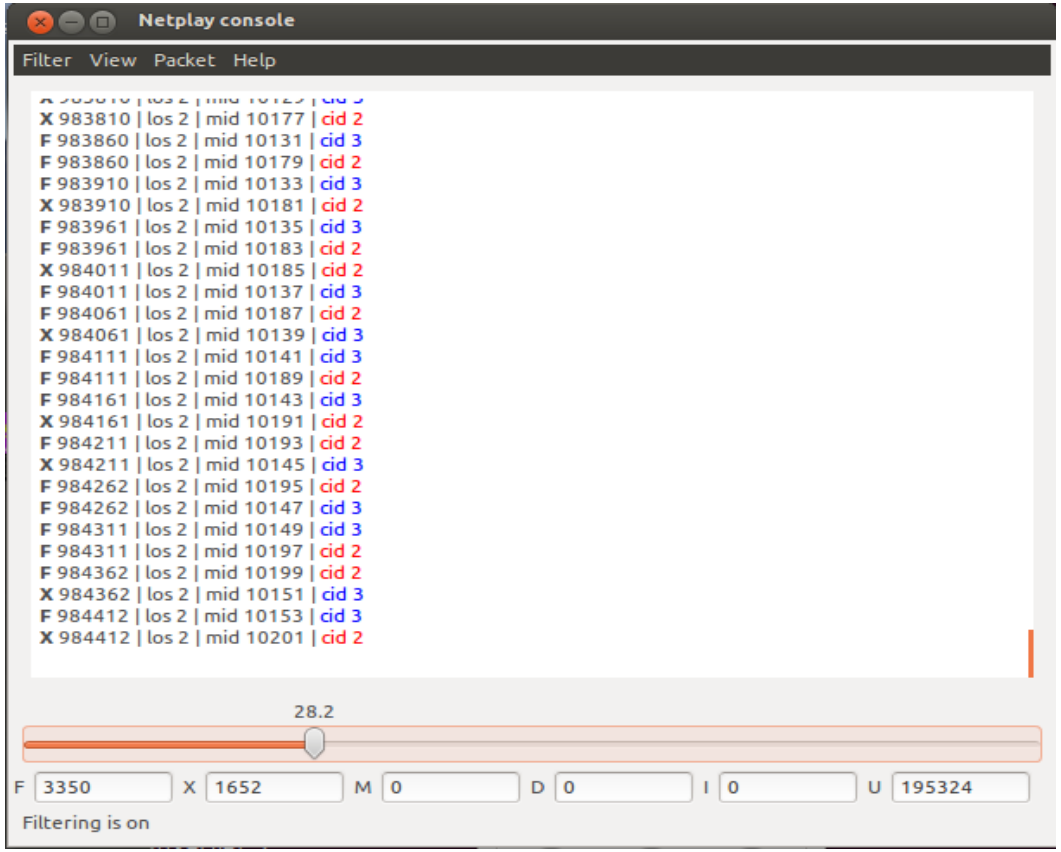
```
#Example 2: printing UDP payload size
ip {
     udp {
          length { set $size length/1024 }
          print $size+"KB"
     }
}
```

```
# Example 3: printing field in UDP payload
ip {
     udp {
          data {
               area {
                    length 4
                    name "code"
               }
               area {
                    offset 4
                    length 16
                    format "string"
                    inf code 10 { print }
               }
          }
     }
}
```

KARYON DEMO ARCHITECTURE

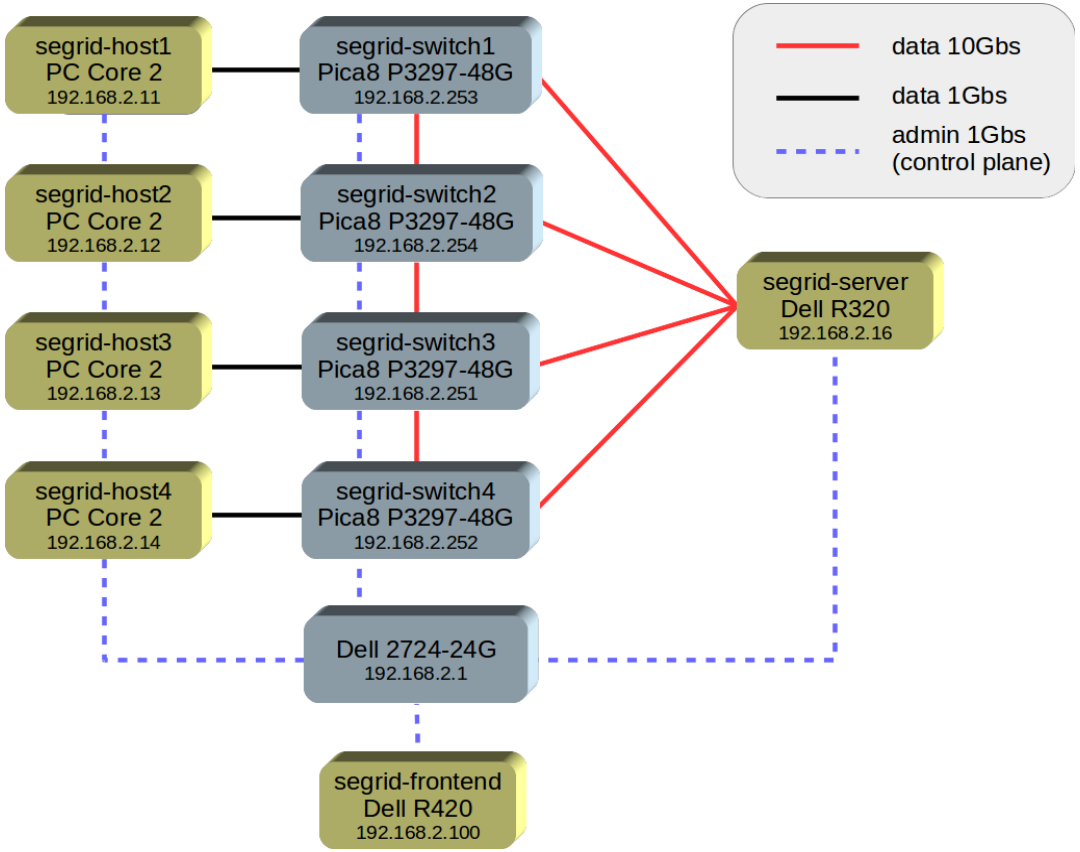


NETPLAY 1 CONSOLE



- Console developed in GTK+. GUI component state can be read in filter rule definitions.
- Filter rule matching data fields (Level Of Service, MID, CID) in UDP payload.
- Single action consisting in dropping packets according to current loss rate (fault injection).
- Loss rate adjusted at run-time using slide button control.

SEGRID TESTBED PLATFORM



- SDN switches (Pica8 3297) running Debian-based OS
- Server equipped with 10Gb SFP+ interfaces
- Copper Twinaxial 10Gbs links
- Attacker machine (1Gbs PC) can be installed between any equipment (transparent bridge)

NETPLAY 2 OVERVIEW

- Based on the Netmap fast library (kernel module and NIC driver updates).
- Netplay 1 was a little framework with GUI. Netplay 2 is a C library.
- Objective is performance (pure speed) rather than flexibility.
- Works at layer-2 level (Ethernet header including 802.1Q fields).
- Unlike Netplay 1, filter rules and actions are now included in a C skeleton and compiled before execution.
- Can be deployed on 10Gbs network (Intel X710 driver support).
- Two execution modes: with or without IP defragmentation

NETPLAY 2 CONFIGURATION

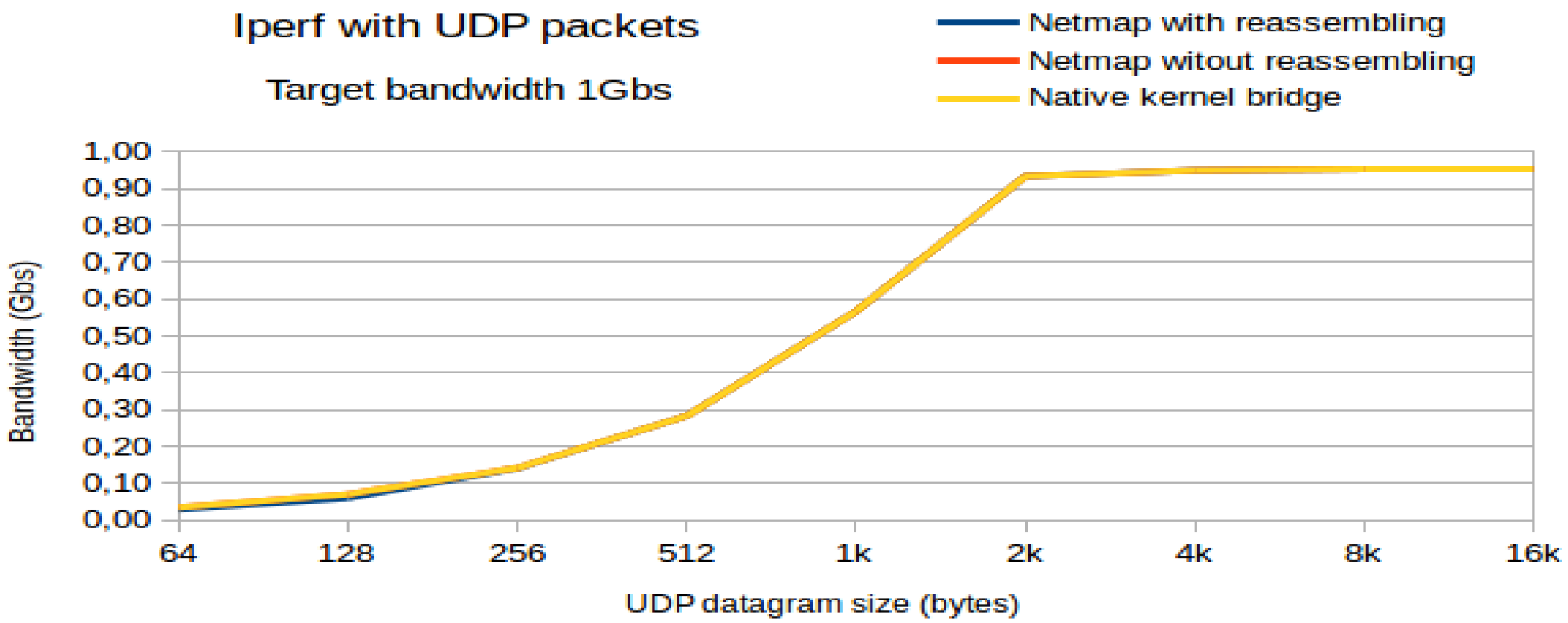
```
// Interface.c
#include "interface.h"

void np_interface_init(NPContext *ctx) {}
void np_interface_handle(NPContext *ctx, NPPacket *packet) {}
void np_interface_close(NPContext *ctx) {}
```

```
// Interface.h
void np_ip_set_source(NPPacket *packet, char *data);
bool np_tcp_set_range(NPPacket *packet, char *buffer,
int start, int length);
void np_packet_delay(NPPacket *packet, int delay);
void np_udp_set_checksum(NPPacket *packet);
```

- GET and SET primitives to read/write any fields in protocol headers and payloads (API with 112 methods)
- Current version supports Ethernet, ICMP, LLDP, IP, UDP and TCP.
- Only requires the installation of the netmap and drivers kernel modules before running.

NETPLAY 2.0 VS LINUX BRIDGE



CONCLUSION AND FUTURE WORK

- With 1Gbs links, measurements don't show any software limitations. Performance bounds are due to the packet switching capacity of the network device.
- Perform new test with 10Gbs links on the Segrid testbed platform to verify the Netmap theoretical 15Mpps limit and see how the switching performance is affected by the fragment reassembling and zero copy mechanisms.
- Enhance Netplay's API with new protocols and functionalities.

REFERENCES

- Netfilter: http://www.netfilter.org/projects/netfilter
- Netmap : http://info.iet.unipi.it/~luigi/netmap
- DummyNet: http://info.iet.unipi.it/~luigi/dummynet

THANKS…