Building efficient dataplanes in software

Luigi Rizzo
Universita` di Pisa, Italy

http:info.iet.unipi.it/~luigi/research.html

Overview

This talk is about fast and efficient software dataplanes for networking

Key ideas apply to hardware as well

Several misconceptions (user vs kernel, zero copy, hw vs sw, latency)

Actual systems way more complex than research prototypes

http:info.iet.unipi.it/~luigi/research.html

Summary

- metrics
- bottleneck identification
- classic I/O versus faster methods
- netmap features

Metrics

Throughput, latency, CPU efficiency are key metrics

Throughput: units of work per second

- Bit, byte, packet, segment, transaction ?

  Depends on the problem and layer where you operate

Throughput

Packet processing devices (switches, firewalls, some middleboxes) normally care about pps

Other apps and system care for bps (bulk I/O)

Most systems optimize for bps (and use large segments, to make the problem 2-3 orders of magnitude easier)

→ not a bad idea if possible!

Latency (and efficiency)

Time to deliver a unit of work from source to destination.

Two parts of the problem:

1. move data across
2. notify the recipient

#2 often takes 10-50x longer than #1

Many save the work by assuming the recipient is always spinning to wait for data (CPU efficiency goes down the drain)

http:info.iet.unipi.it/~luigi/research.html

Bottleneck identification

Our processing chains involve hardware, OS, user software

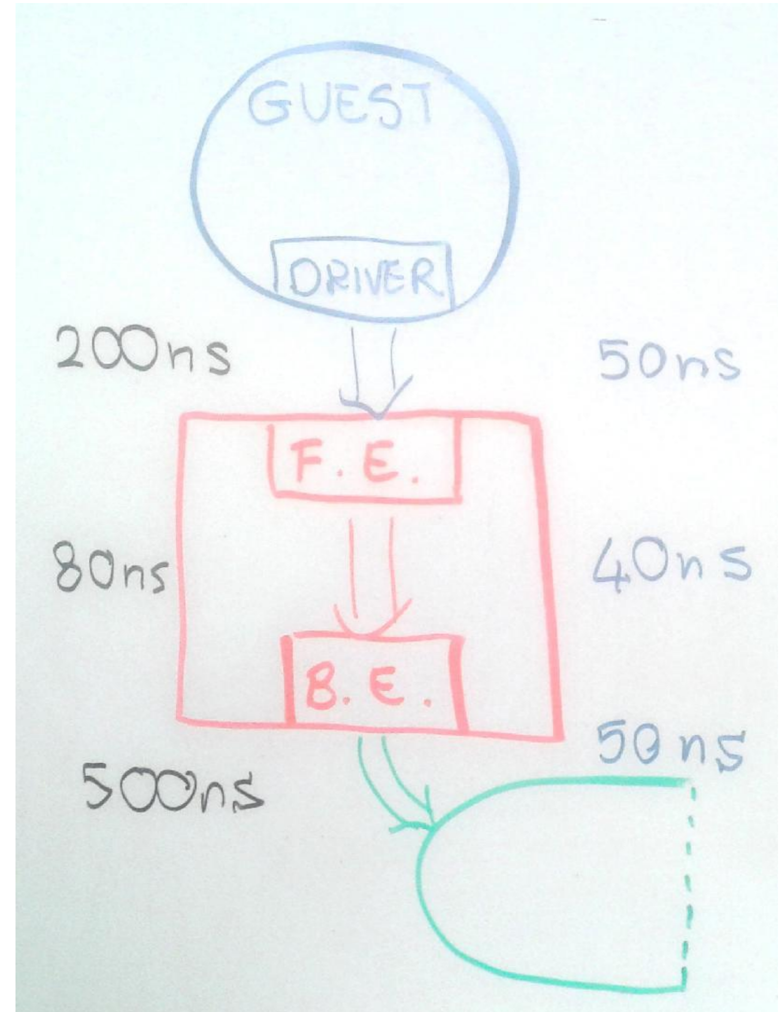
- performance bottlenecks may be in any of them
- and also in their interaction
- performance figures vary widely with workload

Understand how the system works before optimizing.

Example: per packet costs in Qemu

Left: before our changes

Right: with our changes (including replacing TAP with VALE switch)



Bottleneck identification (2)

Do not overestimate hardware

- HW switches usually deliver line rate in all conditions
  simpler problem, closed platforms, established testing standards
- Most NICS cannot do line rate with small packets
  users unlikely to find out due to OS or application limitations
- more expensive CPUs not always better
  multi socket systems have slower memory paths;
  OSes often scale badly with many cores

Reference numbers

Minimum frame size:

64 bytes (data+CRC) + 160 bits framing = 672 bits

- up to 14.88 Mpps on 10 Gbit/s NICs (67.2ns/frame)
- compare with 40+ ns for a L3 cache miss
- 40 Gbit/s NICs are 4x faster

Larger frames are less problematic

- 64 bytes -> 67.2 ns / 14.88 Mpps
- 1500 bytes -> 1200 ns / 0.800 Mpps
- 9000 bytes -> 7200 ns / 0.130 Mpps
- 64K bytes -> 51,200 ns / 0.020 Mpps

Latency can be much larger (500..2000ns)

http:info.iet.unipi.it/~luigi/research.html

Socket performance

Sockets deliver 1-2 Mpps (per core)

- system call
- memory allocation and copies
- repeated address lookups
- complex mbuf parsing
- device programming

- splitting to multiple cores not always possible (ordering)
- scaling can be sublinear (resource contention)

http:info.iet.unipi.it/~luigi/research.html

How to improve performance

When you are CPU bound: be more efficient

- remove useless operations
- move work to init-time when possible
- simplify data structures, reduce runtime decisions
- batching

Once you hit other HW limitations:

- optimize access to hardware
- find better hardware!

OS and network stack bypass

OS bypass: (DPDK, PF_RING, snabbswitch)

"We can do better than the OS"

- expose hardware to userspace
- direct access to HW

But: the OS is useful

- protection
- synchronization
- device independence

Netmap is "network stack bypass", not OS-bypass

http:info.iet.unipi.it/~luigi/research.html

Netmap goals and history

Goal: build a fast path between NIC and applications

- targeted to raw packet I/O
- userspace for convenience
- robust, easy to use, device independent

Evolution

- jun. 2011: first prototype and FreeBSD release
- feb. 2012: linux release
- jun. 2012: VALE (virtual software switch)
- jan. 2013: Qemu extensions
- dec. 2013: netmap pipes, monitor ports
- apr. 2014: bhyve support, mSwitch
- apr. 2015: ptnetmap (virtual passthrough)
- aug. 2015: Windows release

http:info.iet.unipi.it/~luigi/research.html

Data structures

netmap ports support, but do not demand, batching and zero copy

shared data structures: netmap port

protected kernel resources:
NIC, device drivers

http:info.iet.unipi.it/~luigi/research.html

Standard raw packet access

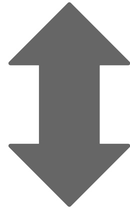
```
socket(...);
bind(...);
read();
write();
```


socket
bpf
PF_PACKET
PF_RING

300..1000 ns/pkt

http:info.iet.unipi.it/~luigi/research.html

Netmap NIC access



fd = open("/dev/netmap");
ioctl(fd, NIOCREGIF, "netmap:eth0");
mmap()

ioctl(), select(), poll(), epoll(), kevent()

NIC

host

network stack

200-250 ns/pkt

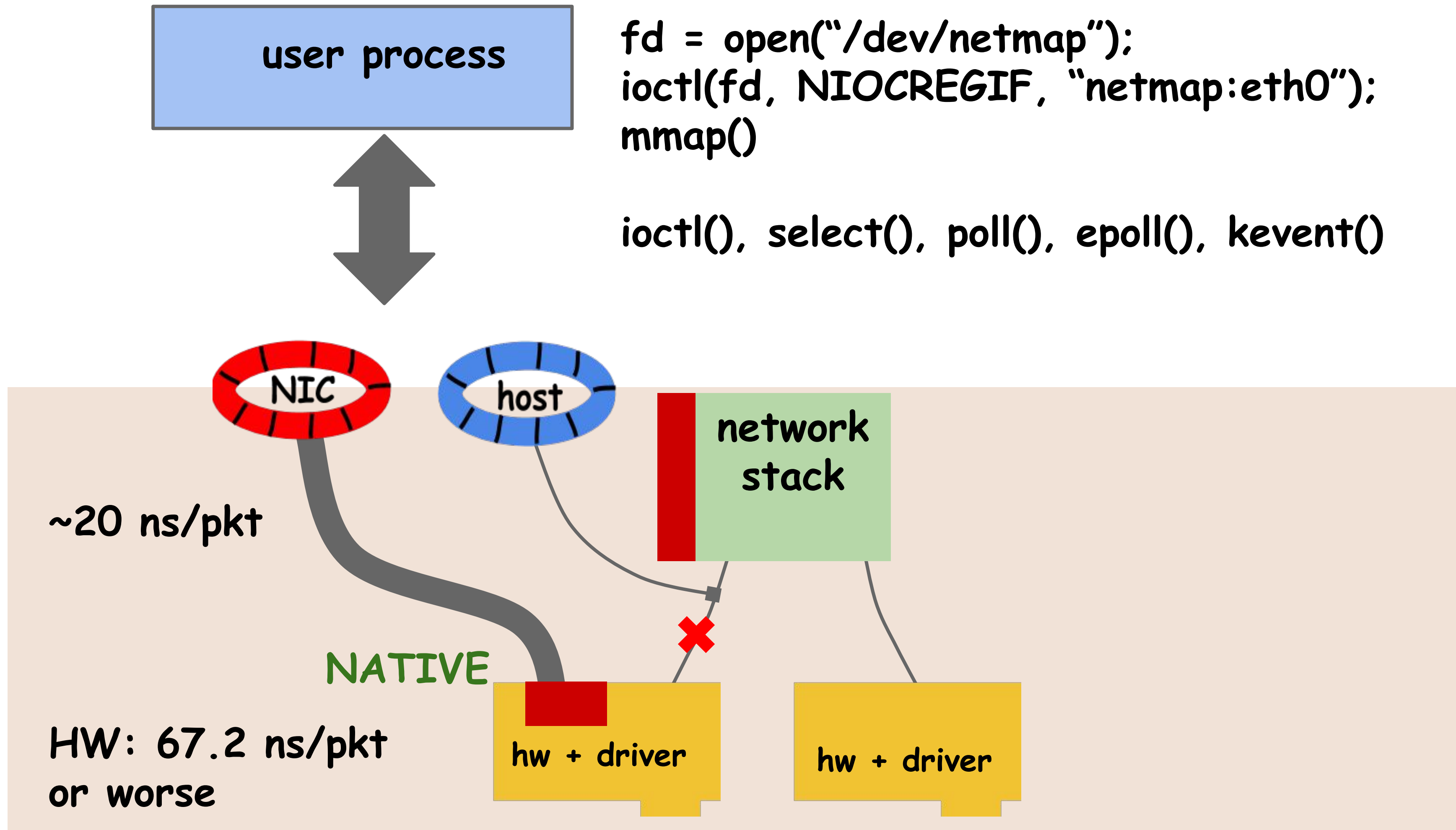
EMULATED

hw + driver

hw + driver

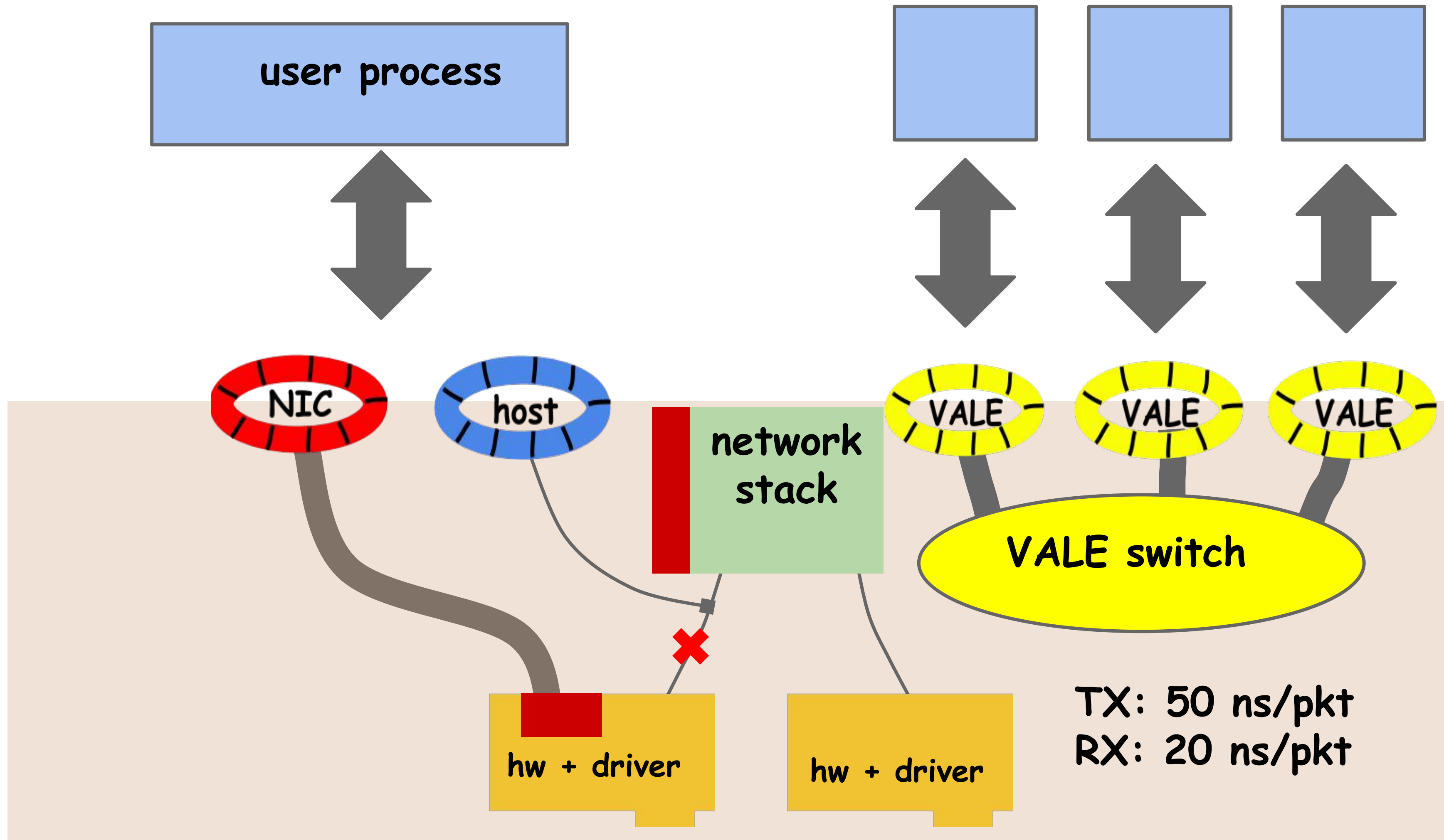
http:info.iet.unipi.it/~luigi/research.html

Native NIC access



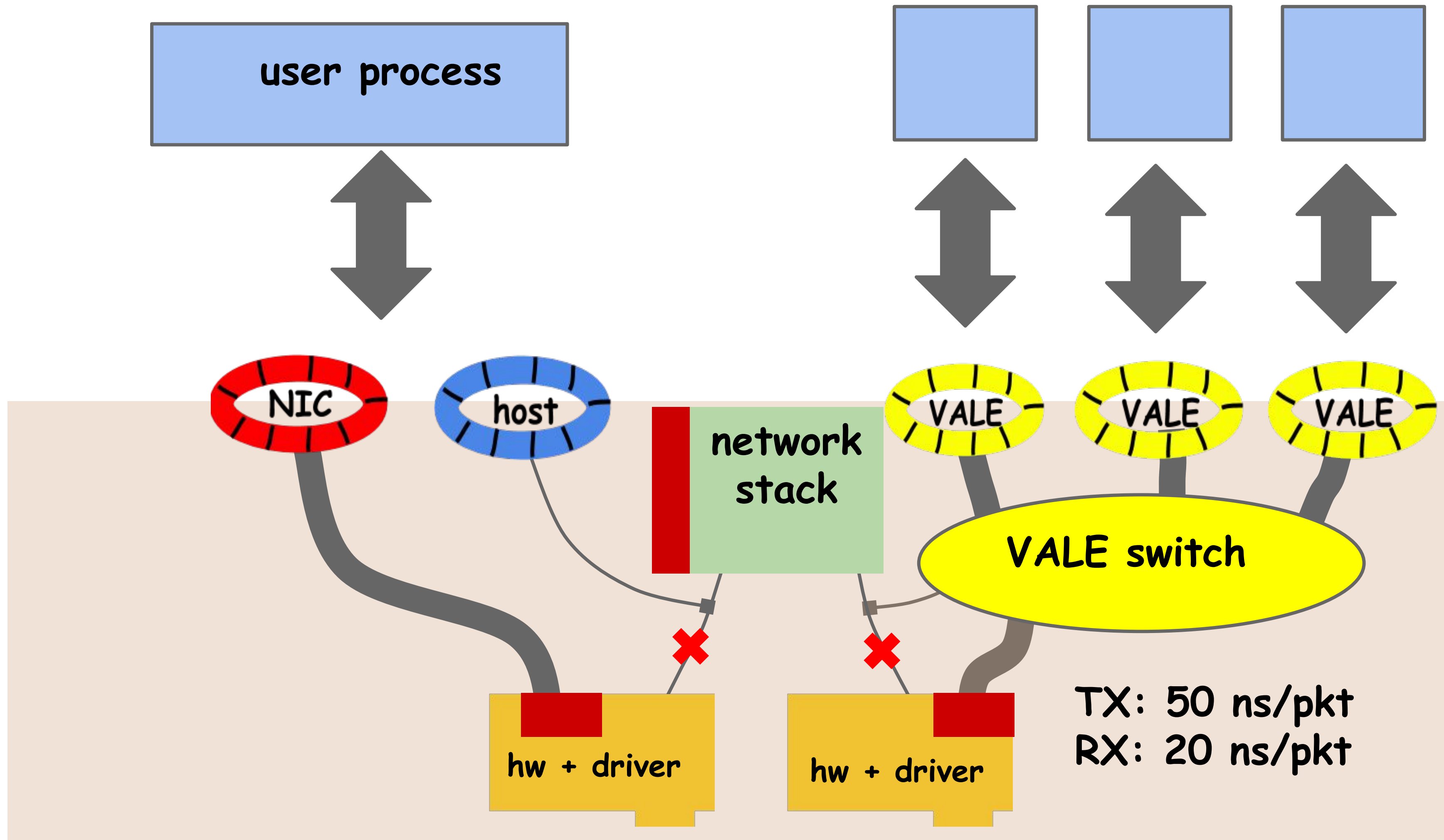
http:info.iet.unipi.it/~luigi/research.html

VALE switch



http:info.iet.unipi.it/~luigi/research.html

VALE switch + NIC/host

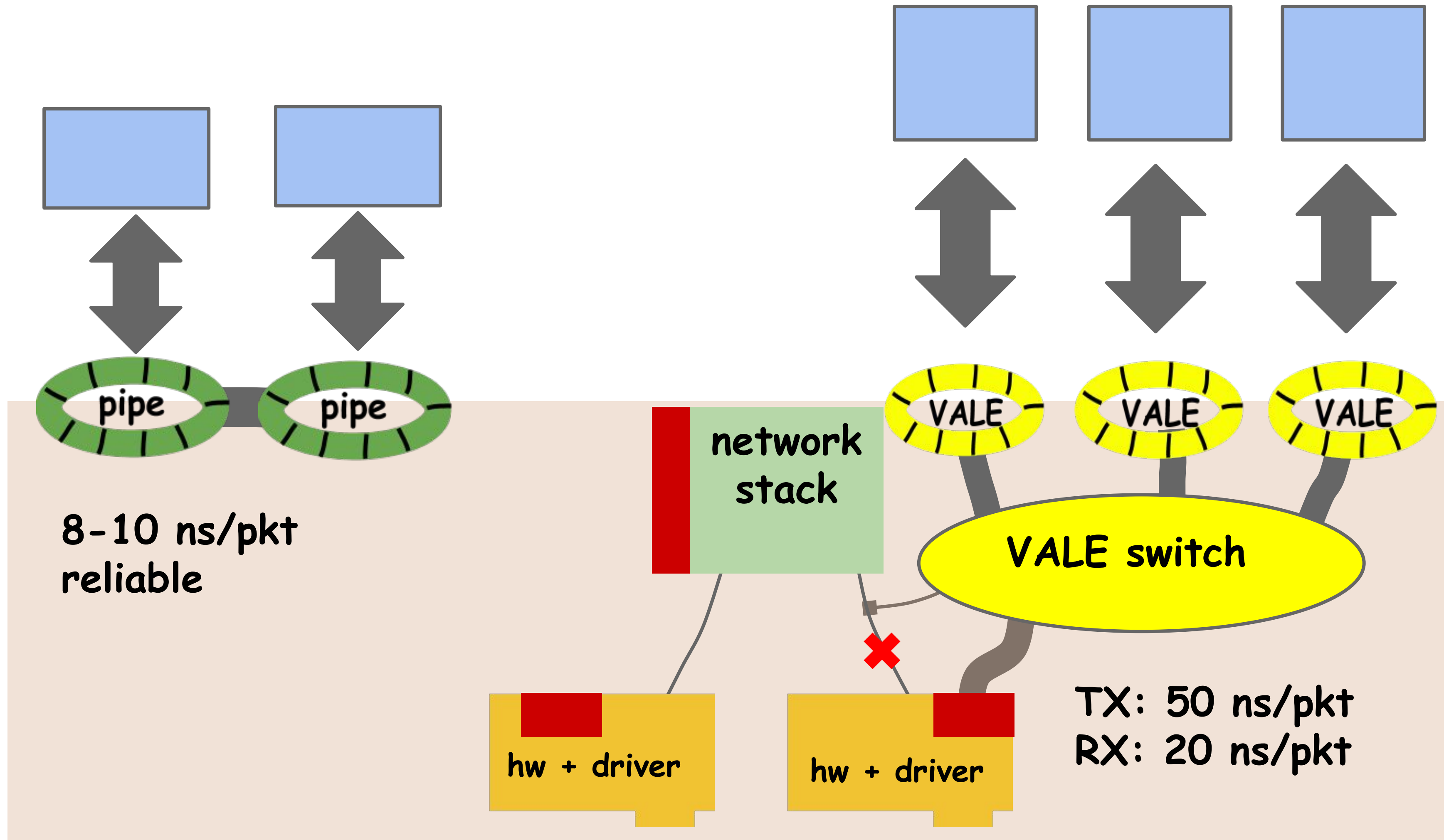


http:info.iet.unipi.it/~luigi/research.html

Custom logic, VALE dataplane

http:info.iet.unipi.it/~luigi/research.html

Netmap pipes



http:info.iet.unipi.it/~luigi/research.html

Netmap monitor

Can create a port that mirrors traffic from another netmap port

- can only read
- can decide whether to monitor tx, rx or both
- traffic available only after the master port has processed it
- excess traffic is dropped

Port naming

Names identify ports

netmap:eth0	NIC, all queues
netmap:eth0-3	NIC, queue #3
netmap:eth0^	host port
netmap:eth0*	all queue and host port
valeXX:yy	VALE port on switch XX
valeXX:yy{NN	pipe, master side
valeXX:yy}NN	pipe, slave side
netmap:eth0+[z][rt]	monitor port (zero copy, tx, rx)

http:info.iet.unipi.it/~luigi/research.html

Isolation

Ports can be allocated to same or different memory regions. Defaults:

- all **NIC/host** ports in the **same** region
  (in the future: separate memory regions)
- each **VALE** port in a **separate region**
- **netmap pipes** share memory **according to the basename**

http:info.iet.unipi.it/~luigi/research.html

Performance

Nota bene:

- these are best-case, single core numbers, computed with moderately large batches

- **ns/pkt is a better metric than Mpps (additive)**

- at the rates of interest, memory accesses can change performance a lot
- still important to know what the limit is

http:info.iet.unipi.it/~luigi/research.html

Performance (2)

Basic I/O (netmap in OR out, device): 20 ns/pkt

- 14.88 Mpps, one core, 900 MHz, 64 byte frames
- (no data touching)
- many NICs cannot do line rate due to their own hw limitations
- PCIe bus accesses also problematic with unaligned accesses

Performance (3)

VALE switch (one data copy in the switch)

- 50 ns/pkt (20 Mpps), 64 bytes
- 250ns/pkt (4 Mpps, 50 Gbit/s), 1500 bytes
- scales to memory bandwidth with multiple senders

netmap pipes (point to point, zero copy)

- 8-10 ns/pkt (100-120 Mpps)
- mostly insensitive to packet size

http:info.iet.unipi.it/~luigi/research.html

Application performance

Per packet time is sum of application and I/O time

- I/O intensive apps have the greatest benefits
- CPU-intensive apps see less gain
- same for DPDK, PF_RING-DNA etc.

Examples:

- netmap-click: 10 Mpps
- netmap-libpcap: 10 Mpps
- netmap-OVS: 3-4 Mpps
- netmap-ipfw: 6 Mpps (filtering), 2 Mpps (dummynet)
- VM-VM: 4-6 Mpps (Qemu/bhyve, netmap mode)
- VM-host: 6-12 Mpps (Qemu/bhyve)
- VM-passthrough: same as bare metal

http:info.iet.unipi.it/~luigi/research.html

Application design strategy

When I/O was an unsurmountable cost:

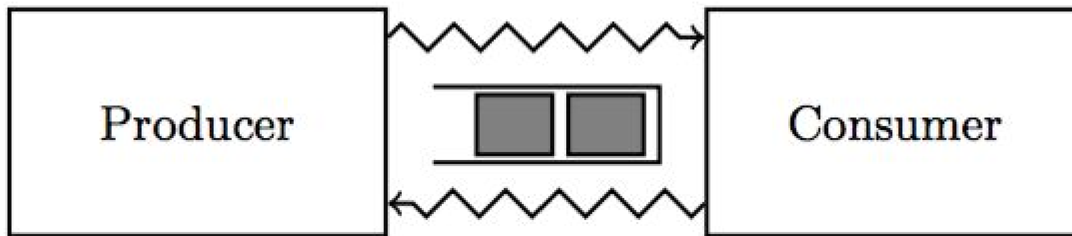
- \+ split traffic, process in parallel
- \+ multiqueue and flow steering come to help
- \- reordering ? centralized data structures ?

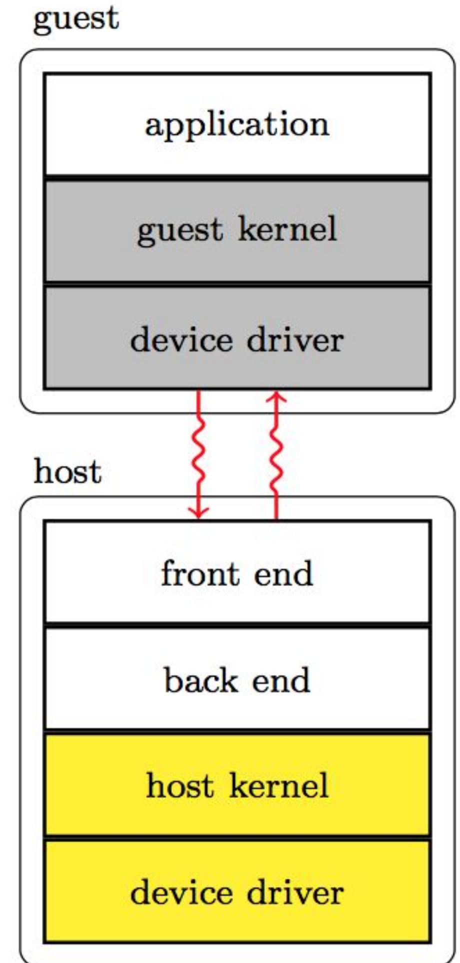
With fast I/O:

- \+ a single thread does everything, if possible
- \+ otherwise build pipelines
- \+ natural approach e.g. for VM networking
- \+ easily solves reordering and coherency
- \- more memory pressure
- \- synchronization between pipeline stages

Network I/O in VMs

virtualization platforms have pipelines with stages communicating through queues and some synchronization mechanism

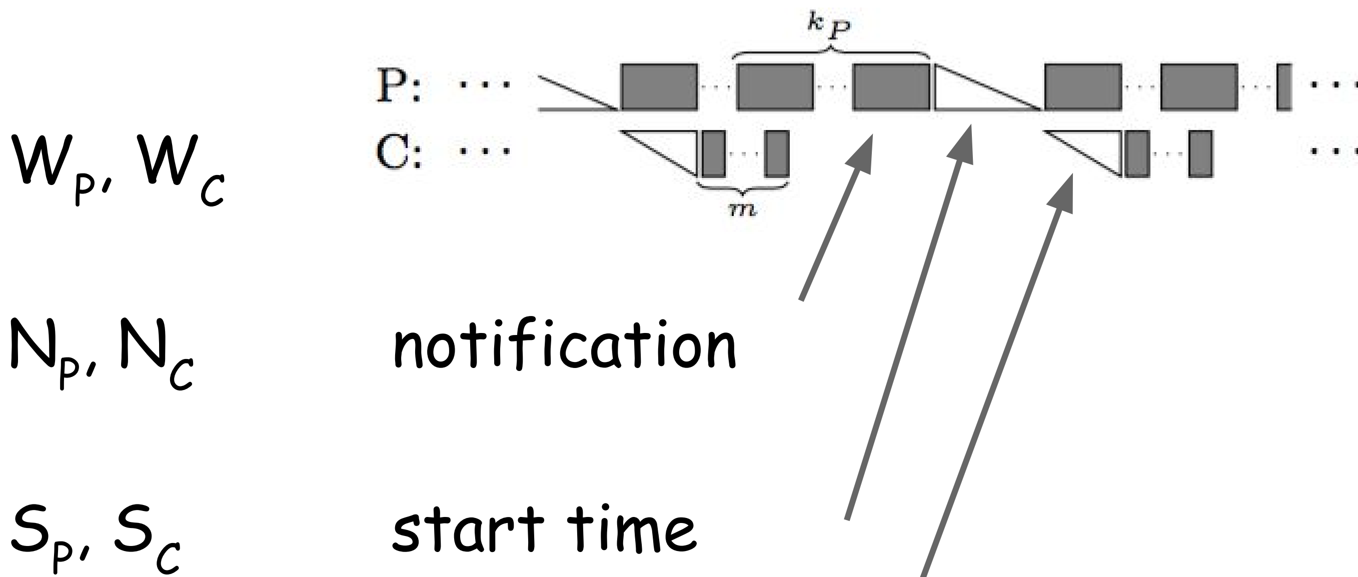




Start/stop costs often dominate individual packet processing costs

http:info.iet.unipi.it/~luigi/research.html

Producer-consumer interaction



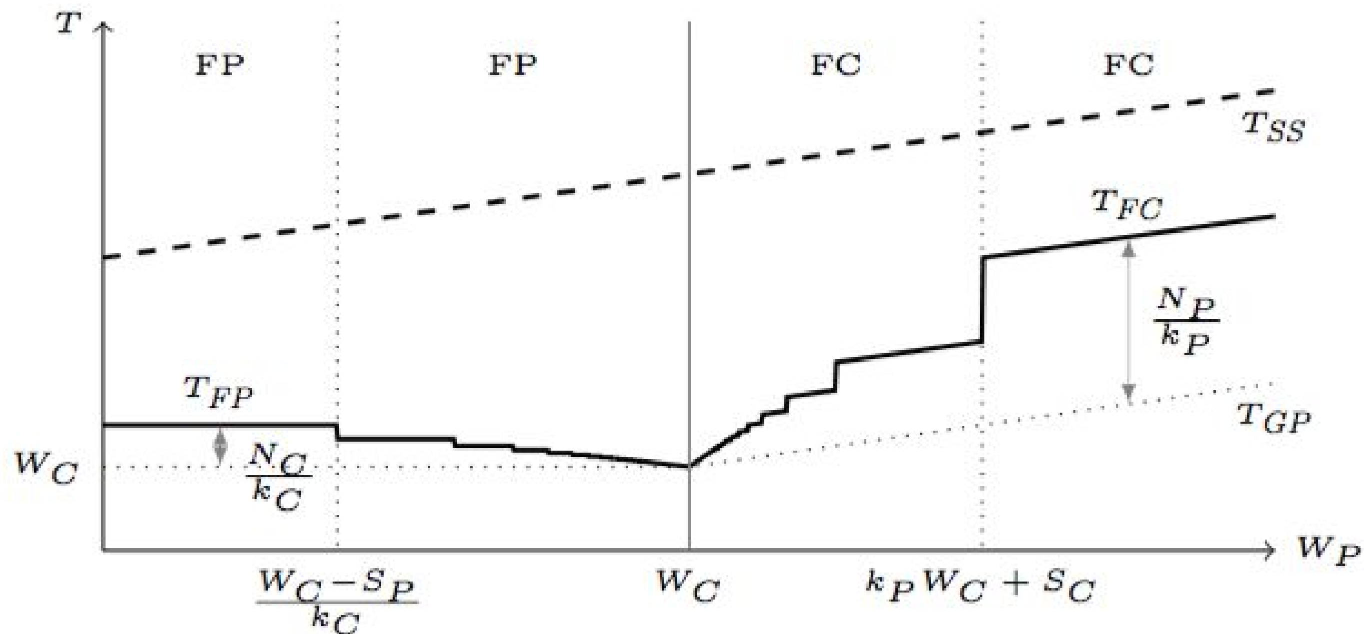
Depending on the values we have different regimes, a faster stage may slow down the entire pipeline.

http:info.iet.unipi.it/~luigi/research.html

Effects of pipeline imbalance

Time per packet (lower is better) is optimal when producer and consumer are balanced

- slow down the fast stage to improve throughput!
- see our work at ANCS'16



Network I/O in VMs

Individual packet I/O require interrupts and "vm exits" to talk to the switch in the host

- both are 10-50x more expensive than on bare metal

Amortize them with "paravirtualized" device models and drivers:

- establish a shared memory channel between guest and host
- first interaction pays the vm exit cost, then a thread spins for a while waiting for events
- virtio, vmxnet and other devices use this mechanism

Hypervisor netmap support

(single core, best case, large batches, aligned packets, ...)

QEMU: up to 6-8 Mpps G-G, 12 Mpps G-H

- basic netmap support in-tree (3-4 Mpps)
- more flag, PV netmap in guest, indirect buffers not committed yet

bhyve: ~8 Mpps G-H

- full netmap and virtio support

Xen: 6-10 Mpps G-H

- first approach, replace xen rings with VALE
- current approach: netmap extension for netfront/netback
- use VALE in DOM-0

Virtual netmap passthrough

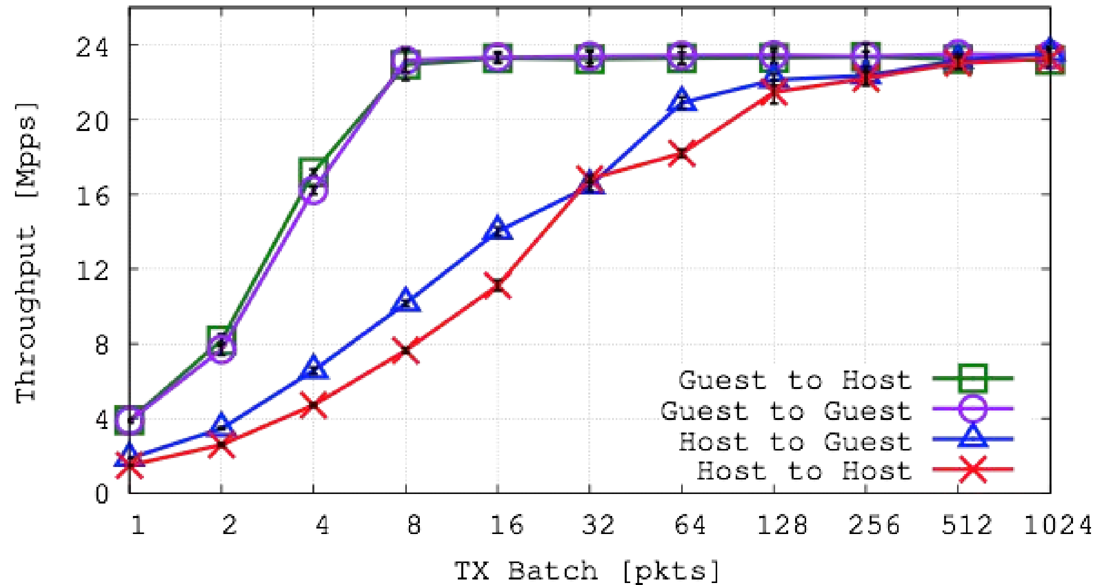
The conventional frontend/backend architecture still requires data copies

PCI passthrough is a popular approach to achieve bare-metal performance in VMs
- bound to specific HW availability
- communication goes through the PCIe bus

Better option: use passthrough on netmap ports

ptnetmap performance

64 byte frames

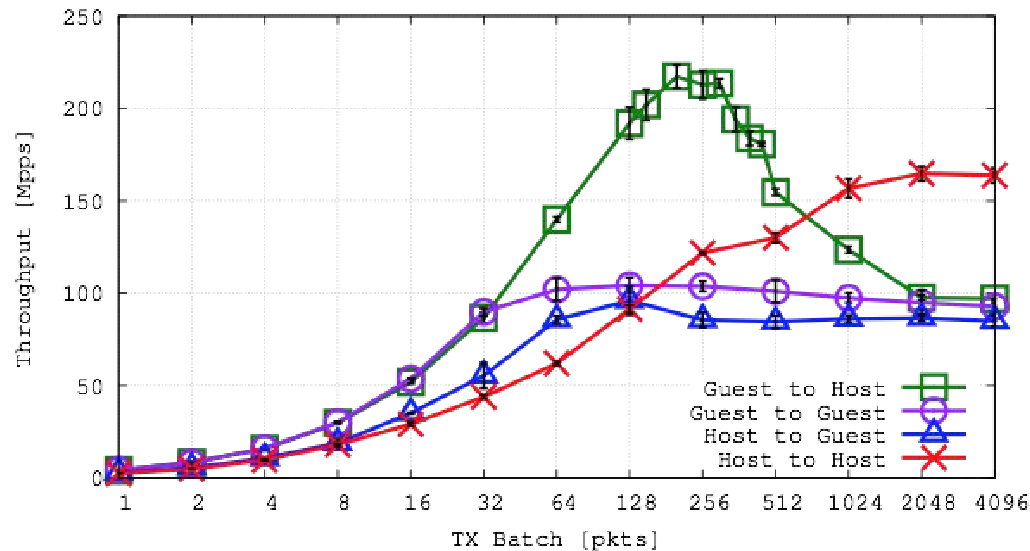


VALE gives ports isolation (data copy)

- the bottleneck is the sender, which executes the data copy
- a sender on the guest is faster thanks to the helper thread

ptnetmap throughput

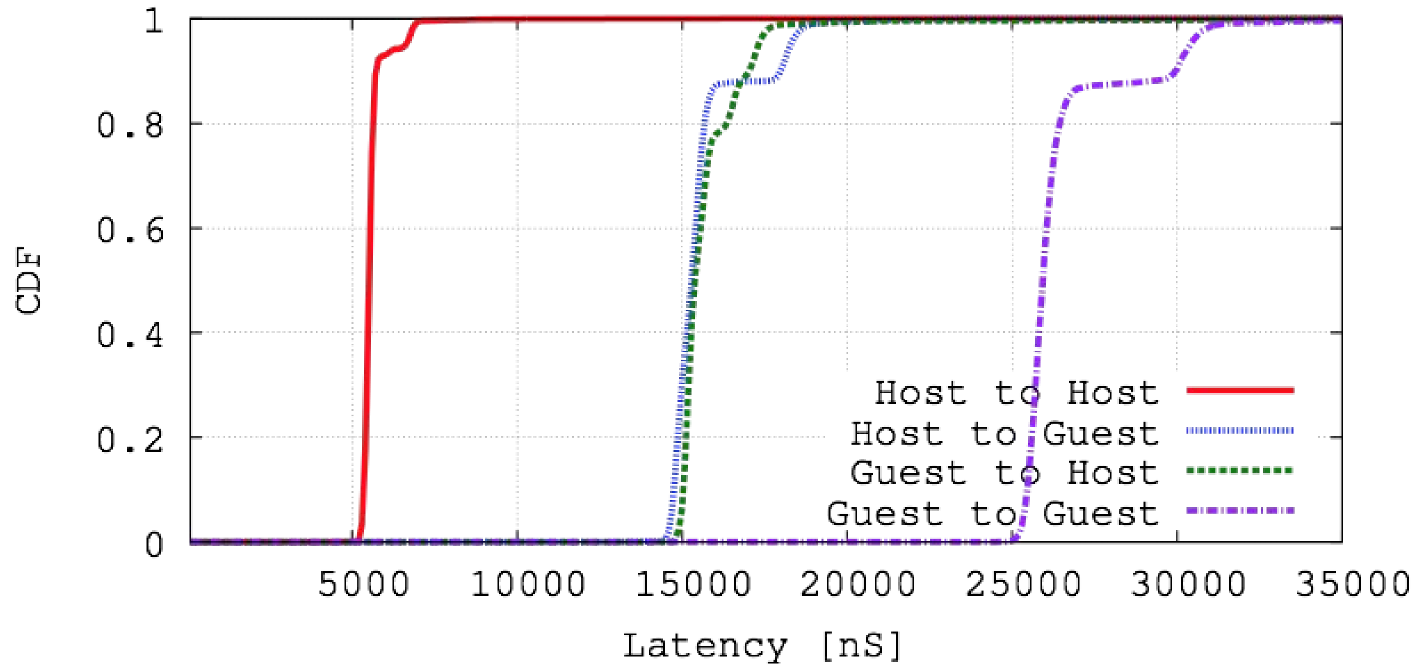
Throughput on pipes



True zero-copy, good for mutually trusted peers

- very little work, much higher throughput
- more balanced bottlenecks
- *Short queue regime* in Guest-Host case

RTT latency, blocking



This is the behaviour on the first packet of a batch

- **the kernel threads and the guest are sleeping, waiting for a notification or an interrupt**

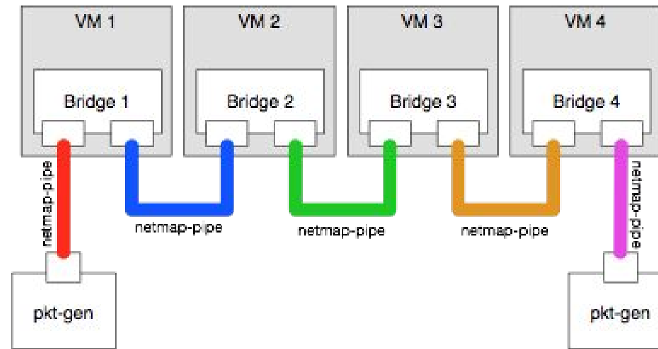
RTT latency, non blocking

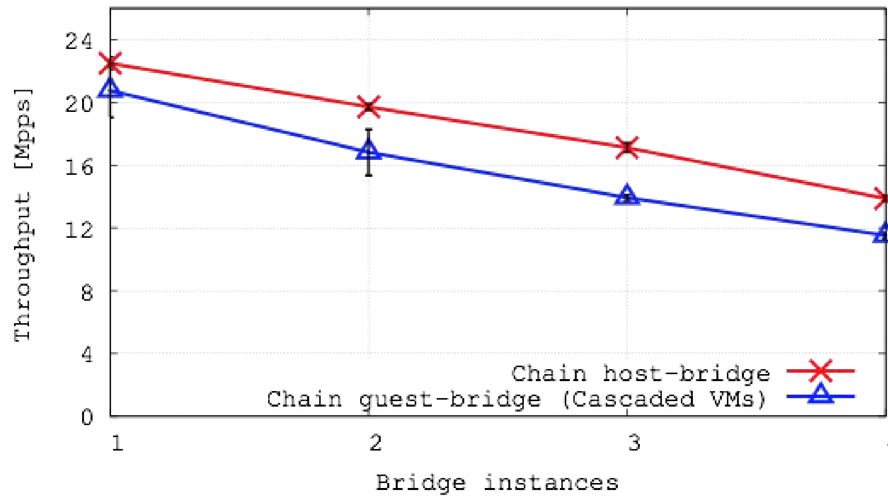
CDF
1
0.8
0.6
0.4
0.2
0
5000 10000 15000 20000 25000 30000 35000
Latency [nS]
Host to Host
Host to Guest
Guest to Host
Guest to Guest

Behaviour for subsequent packets in a batch:

- **kernel threads are active, thus avoiding all interrupts, VM exits, and process wakeups**

Chained VM performance





http://info.iet.unipi.it/~luigi/

Conclusions and current work

Network I/O is essentially a solved problem

- now look at application design: parallel vs pipeline
- design to amortize communication latency
- example: PSPAT, packet scheduling with parallel transmission (see my research page)

Move away from packet abstraction

- only useful on the link. VMs and VNFs use streams.

http:info.iet.unipi.it/~luigi/research.html

Acknowledgements

Funding and support (over time):

Intel Research, EU FP7 (Change, Openlab), ACM, Netapp, NEC, EU H2020 (SSICLOPS)

Developers:

Luigi Rizzo, Giuseppe Lettieri, Michio Honda, Matteo Landi, Gaetano Catalli, Vincenzo Maffione, Stefano Garzarella, Alessio Faina

http:info.iet.unipi.it/~luigi/research.html