NetFPGA Summer Course

Presented by:

Noa Zilberman

Yury Audzevich

Technion

August 2 – August 6, 2015

http://NetFPGA.org

USING NETFPGA AS AN APPLICATION

Agenda

- NetFPGA as an application
- OpenFlow as an example
- OSNT
- BlueSwitch

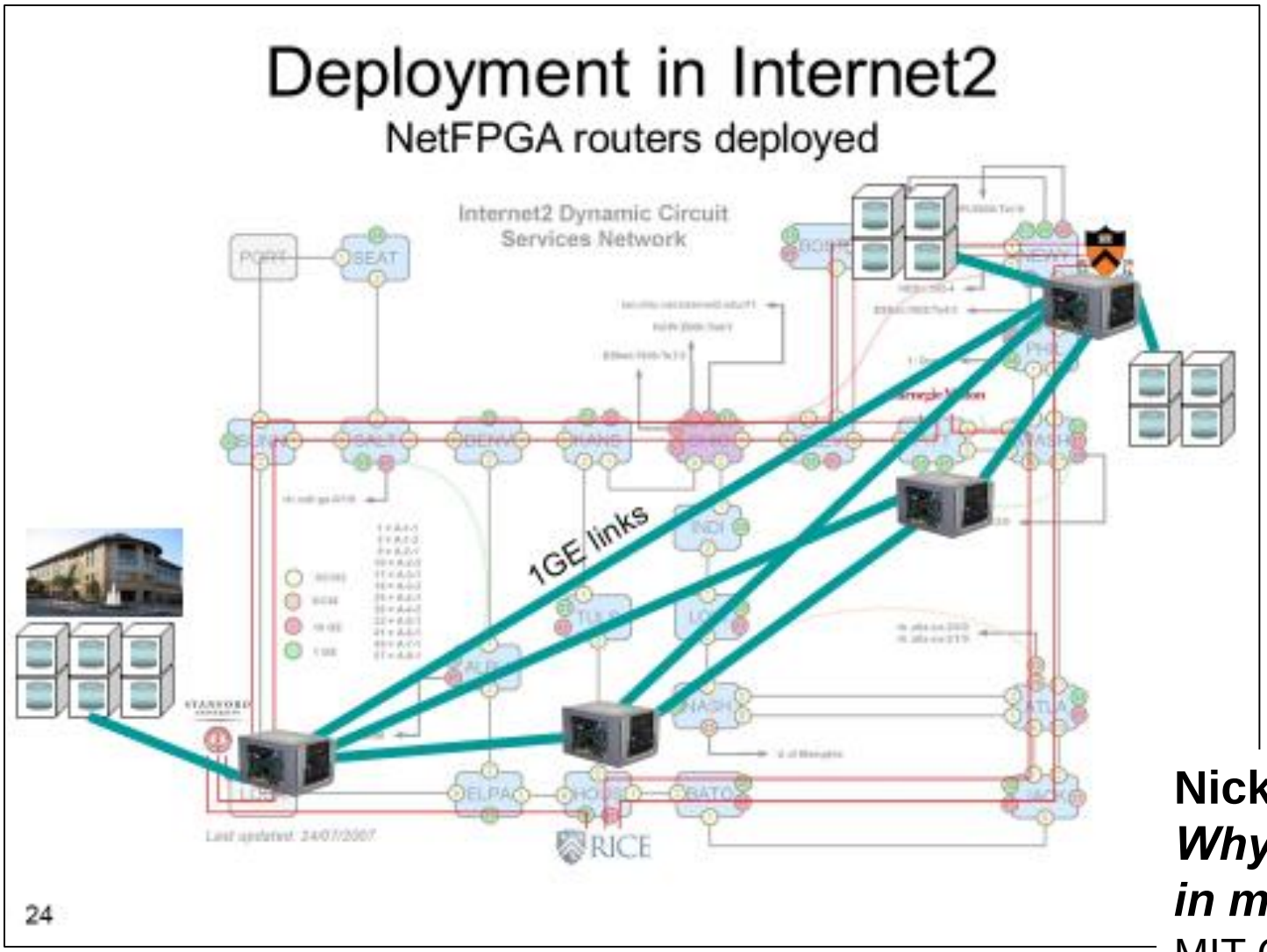
NetFPGA as an Application

- **Hardware development is just one aspect of research**
- **Many need flexible, open source platforms**
- **Idea: take a project developed over NetFPGA and be an end-user**

OpenFlow as an Example

- **Have you heard of Software Defined Networking?**
- **OpenFlow is a southbound interface between the data and a control plane**
- **NetFPGA enabled OpenFlow**
  - Provided a widely available open-source development platform
  - Capable of line-rate
- **NetFPGA was, until its commercial uptake, the reference platform for OpenFlow**

Early OpenFlow Deployments



Nick McKeown
Why can't I innovate in my wiring closet?
MIT CSAIL Colloquium,
April 17 2008

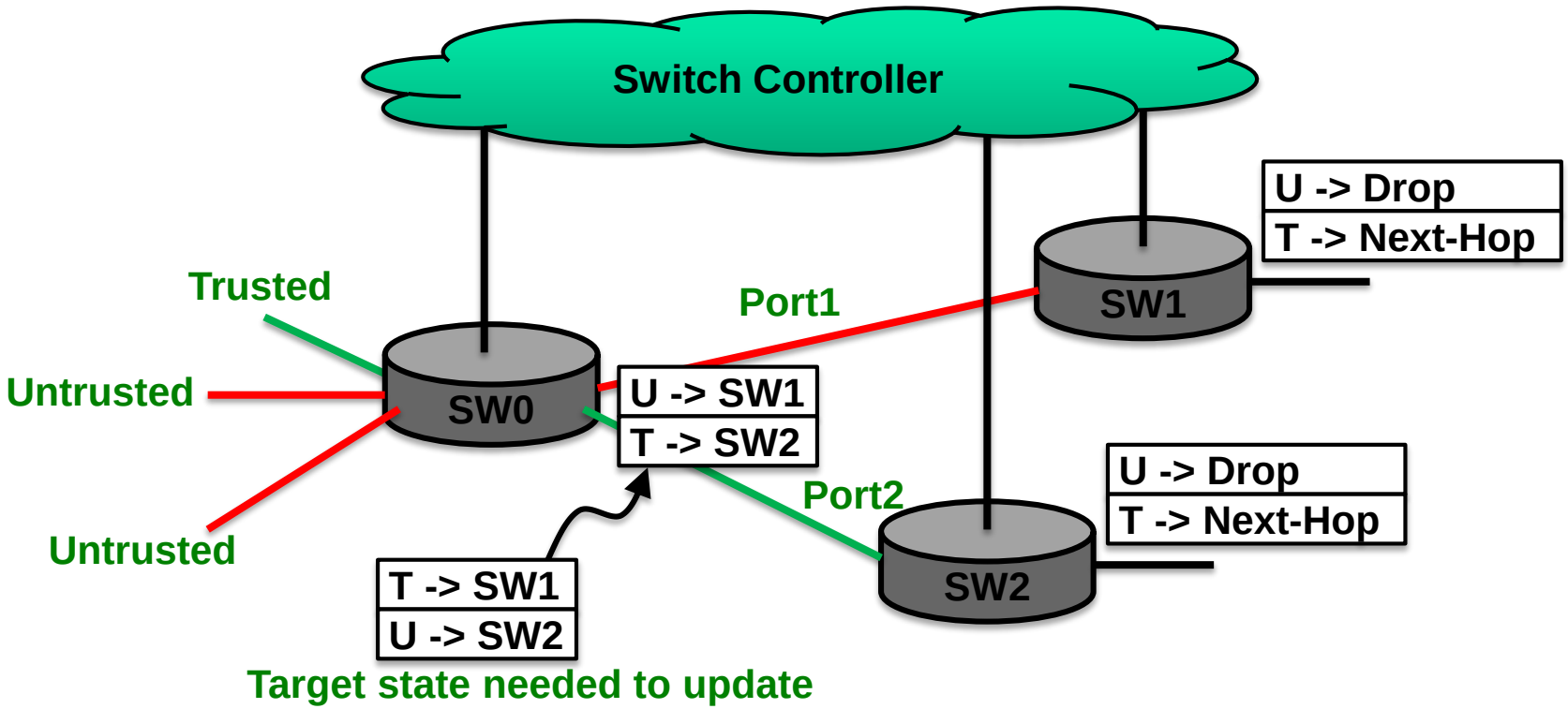
BLUESWITCH

BlueSwitch

- **An openFlow switch**
- **Parameterized modular design**
- **Multi-Table**
- **Provides packet consistency**
  - In the internal datapath of the switch
- **Supports openFlow v1.4 *Bundle* feature**
  - Atomic updates to switch configuration
- **Currently running over NetFPGA-10G**

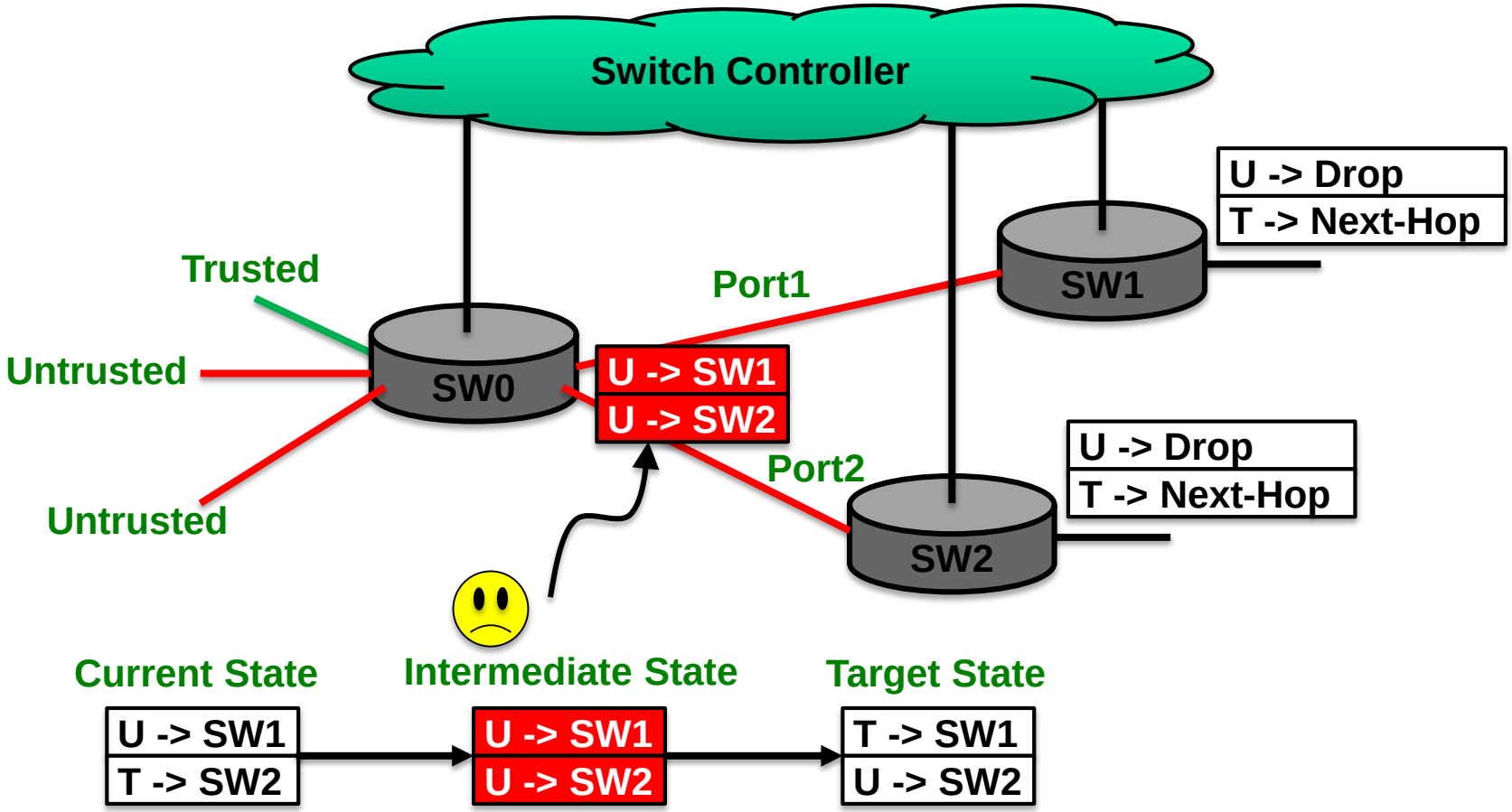
Inconsistent Policy Update Problem

- Inconsistent policy update in SDN - Security and Resilience



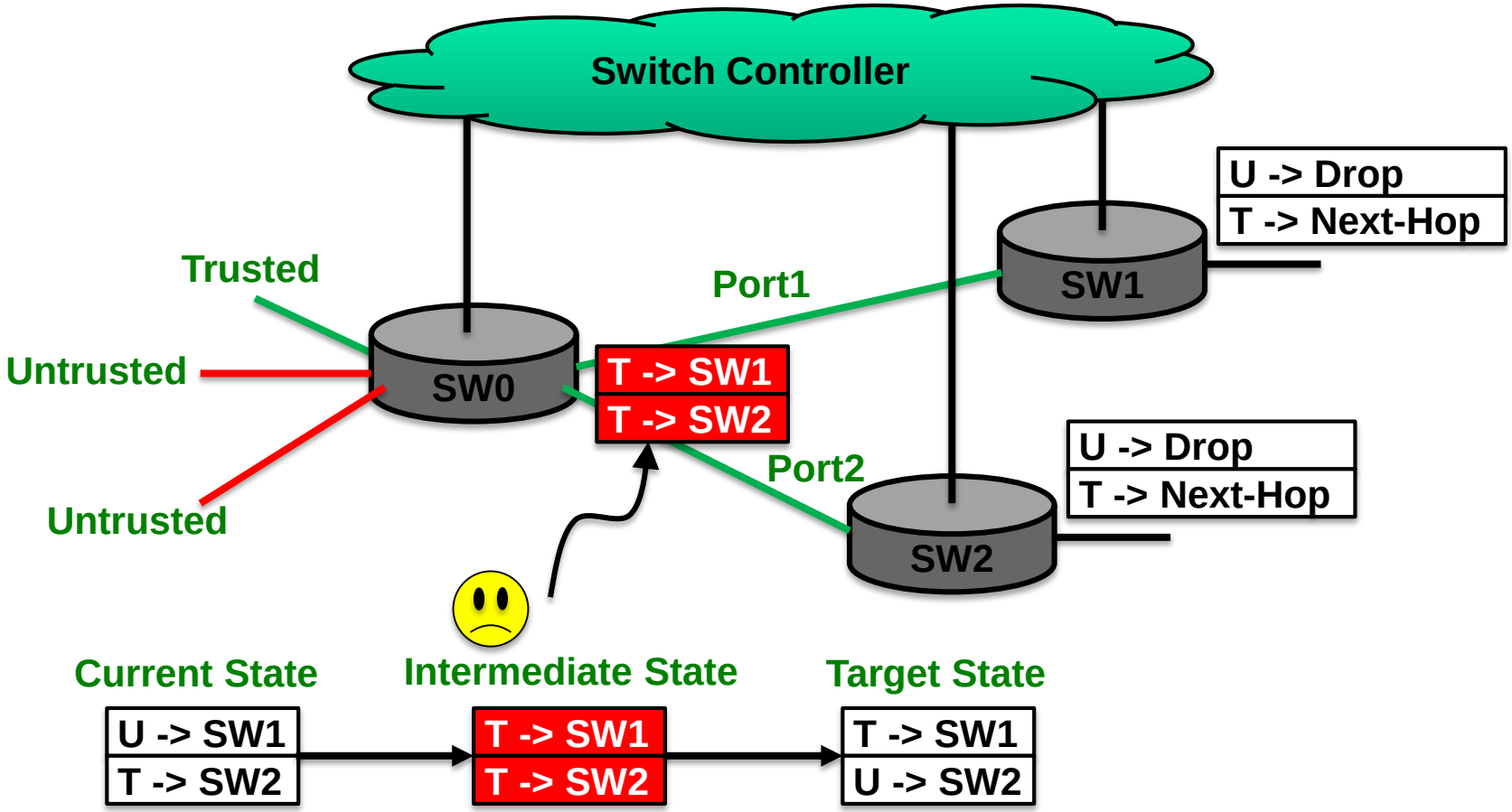
Inconsistent Policy Update Problem

- Risky Rule Update I – Update per Rule



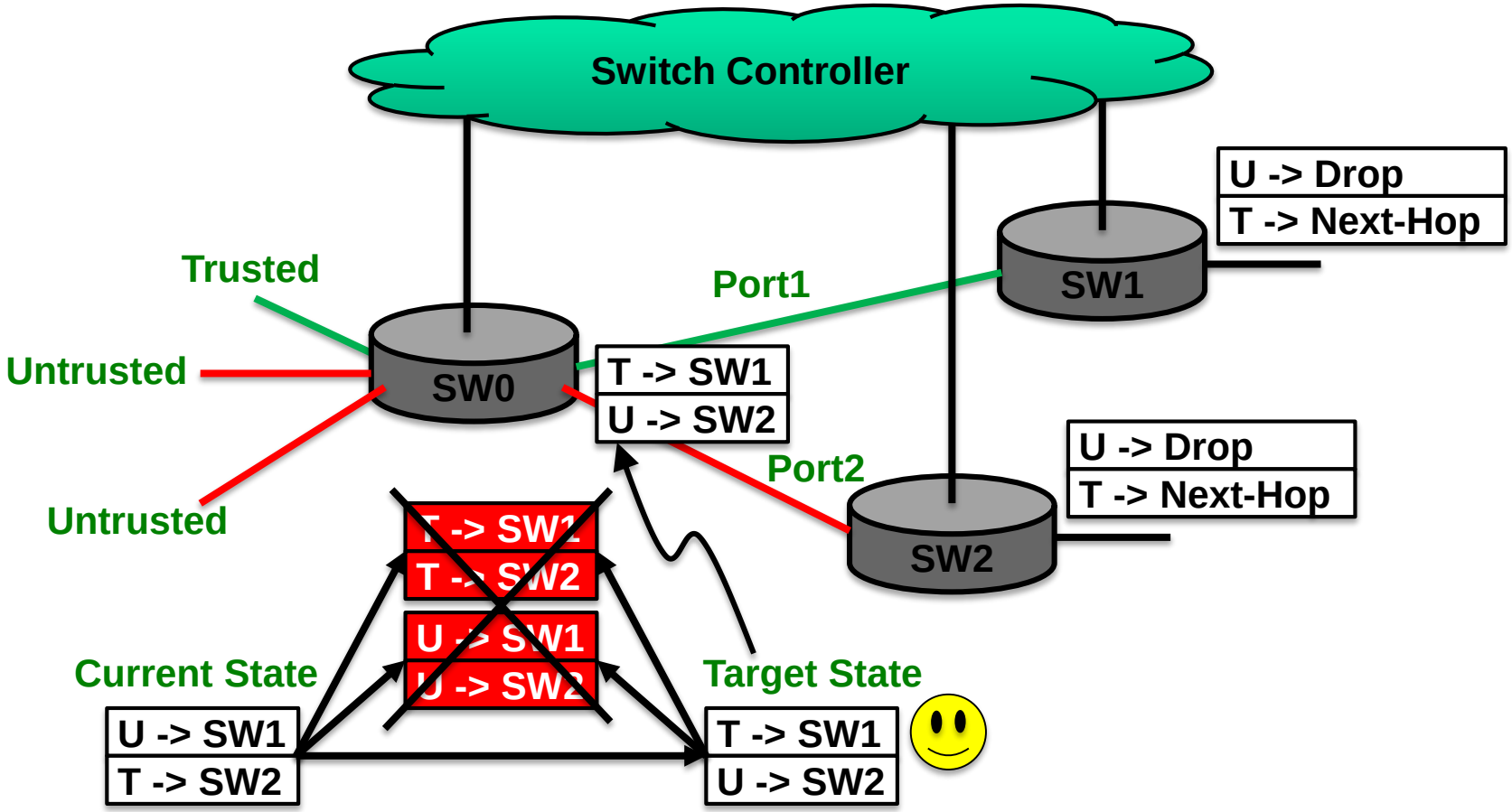
Inconsistent Policy Update Problem

- Risky Rule Update II – Update per Rule



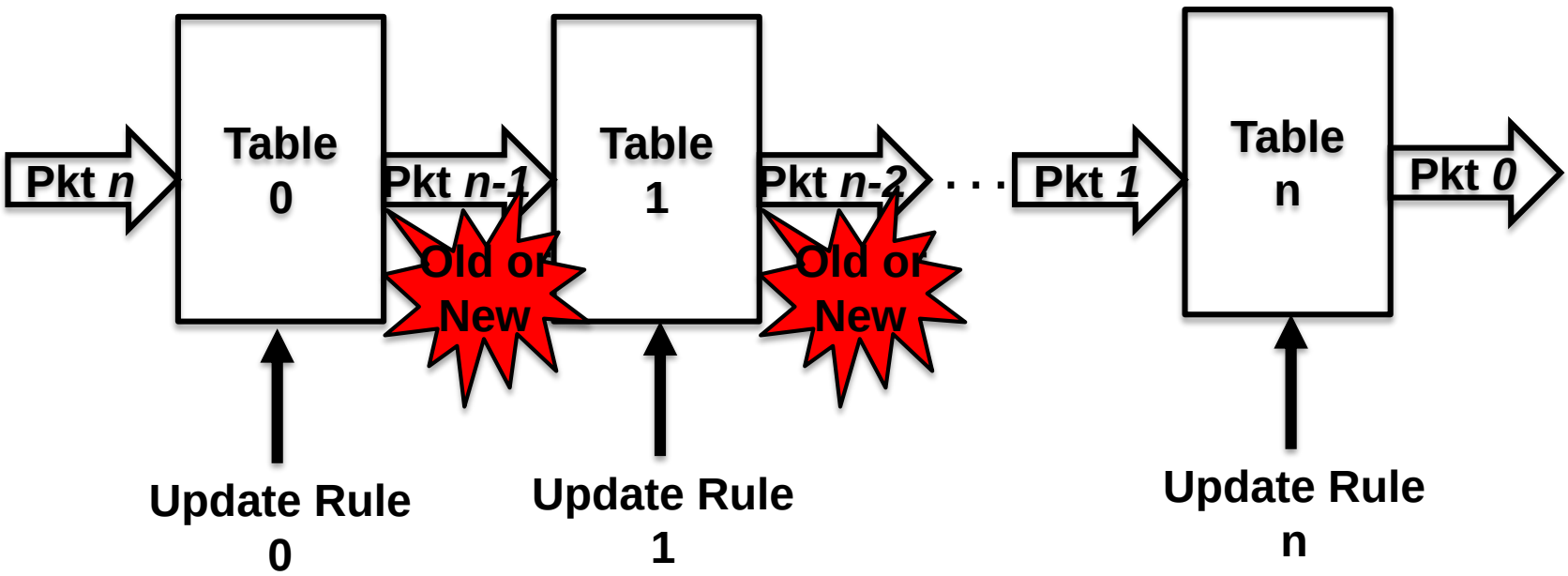
Inconsistent Policy Update Problem

- Safe Atomic Update – Update All Rules



Problem in Multi-Table OF Switch

- OpenFlow Switch Multi-Table Inconsistency Problem



Configuration Consistency

- **No commodity switch hardware is consistent**
- **Transitions from state *A* to *B* can move through intermediate (non-deterministic) states**
- **Not a new problem but SDN can fix this with principled hardware/software co-design**

Consistency in Blueswitch

- **Consistent double-buffered multi-flow-table structure**

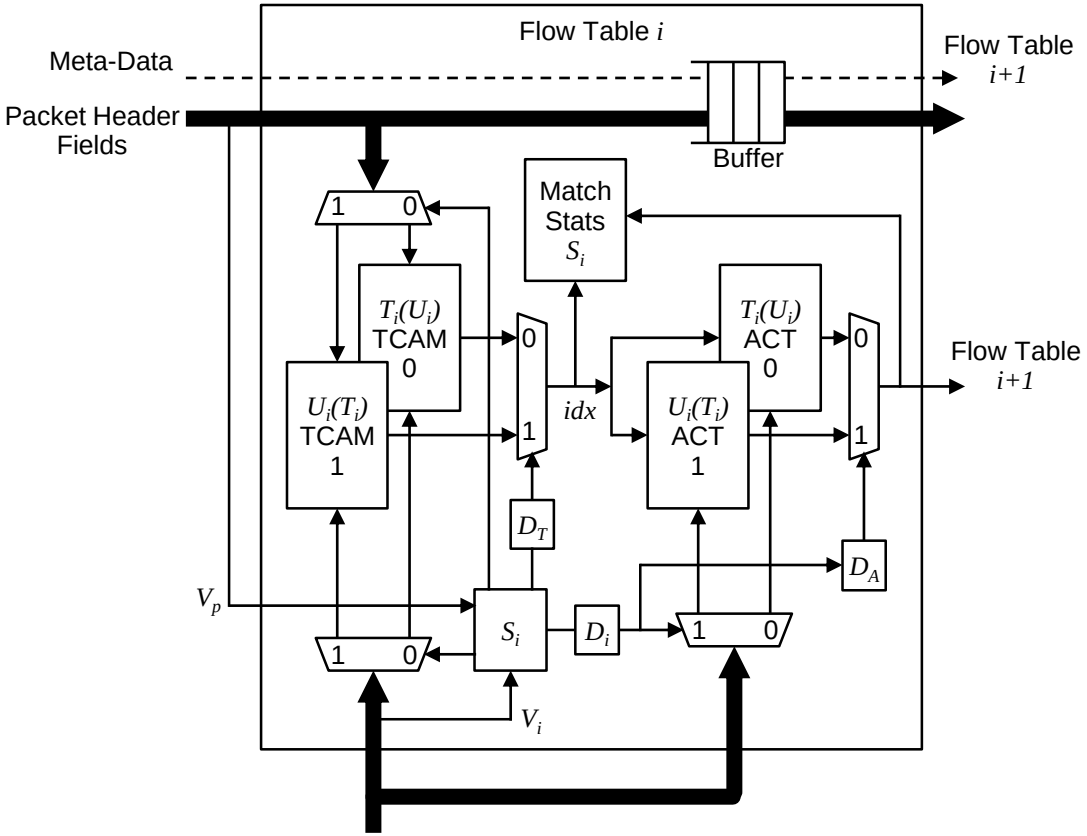
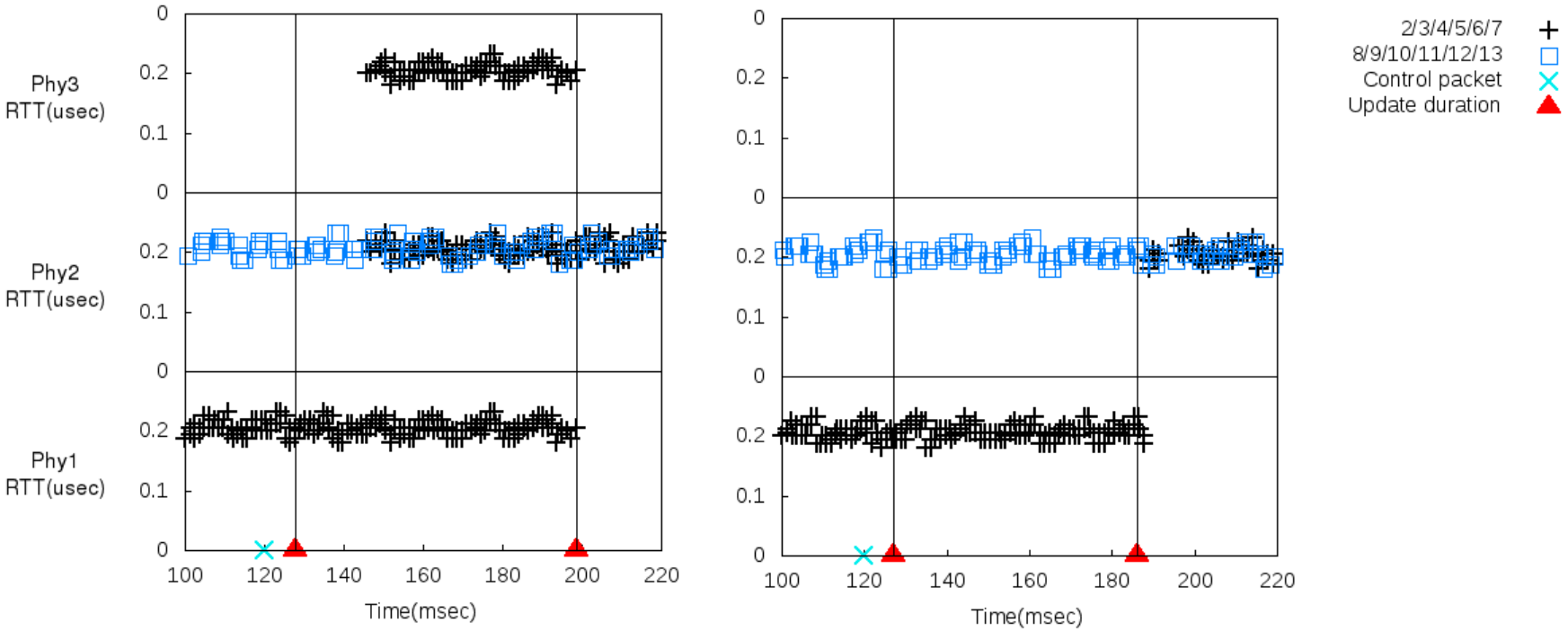


Table update interface (from API via DMA/PCIe)

Blueswitch consistent rule update

Inconsistent and consistent data-plane packet behavior results during new policy update



HW Implementation Results

- **Results on NF10**

Resource	Reference Switch	Reference Router	Blueswitch (overhead)
Slices	18.3K	20.3K	22.9K (25%)
Slice Registers	44.3K	47.2K	57.2K (29%)
LUTs	39.0K	43.1K	43.3K (11%)
LUT-Flip Flop	56.6K	61.7K	72.4K (28%)
No. of TCAMs	1	1	6

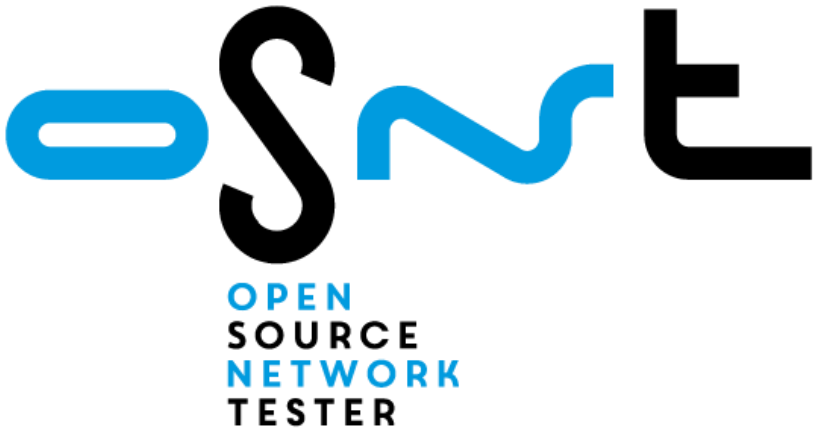
BlueSwitch – More Information

- Han J.H *et al* - Blueswitch: Enabling provably consistent configuration of network switches, ANCS 2015
- BlueSwitch source code - NetFPGA GitHub repository
- OpenVSwitch for BlueSwitch - https://github.com/pmundkur/ovs

OSNT

Long development cycles and high cost create a requirement for open-source network testing

- **Open-source hardware/software co-design**
  - **For research and teaching community**



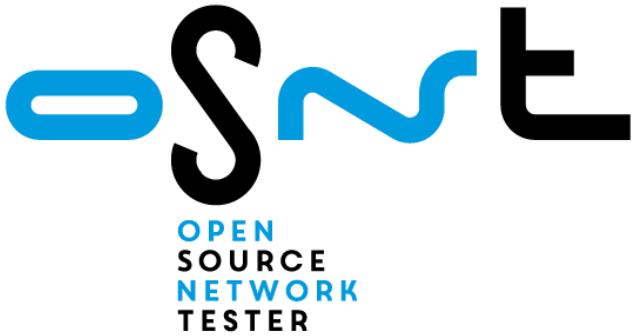
- flexible
- scalable
- community-based

www.osnt.org

- **the first OSNT prototype is based upon the NetFPGA-10G open-source hardware platform**

- **OSNT is portable across a number of HW platforms**
  - maximizing reuse
  - minimizing reimplementation costs (as new HW, physical interfaces become available)

- **we invite everyone from the community to audit our implementation and adapt it to your needs**



- **NetFPGA platform enabled the first prototype of OSNT.**
- **The open nature of NetFPGA ecosystem represents the best starting point for open HW/SW community-oriented projects.**
- **OSNT aims to build a community as NetFPGA did.**

OSNT architecture on NetFPGA-10G

OSNT flexibility provides support for a wide range of use-cases

- **OSNT-TG**
  - a single card, capable of generating packets on four 10GbE ports
  - to test a single networking system or a small network

- **OSNT-MON**
  - a single card, capable of capturing packets arriving through four 10GbE ports
  - to provide loss limited capture system with both high-resolution and high precision timestamping

OSNT architecture on NetFPGA-10G

- **Hybrid OSNT**
  - the combination of Traffic Generator and Traffic Monitor into single FPGA device and single card
  - to perform full line-rate, per-flow characterization of a network (device) under test

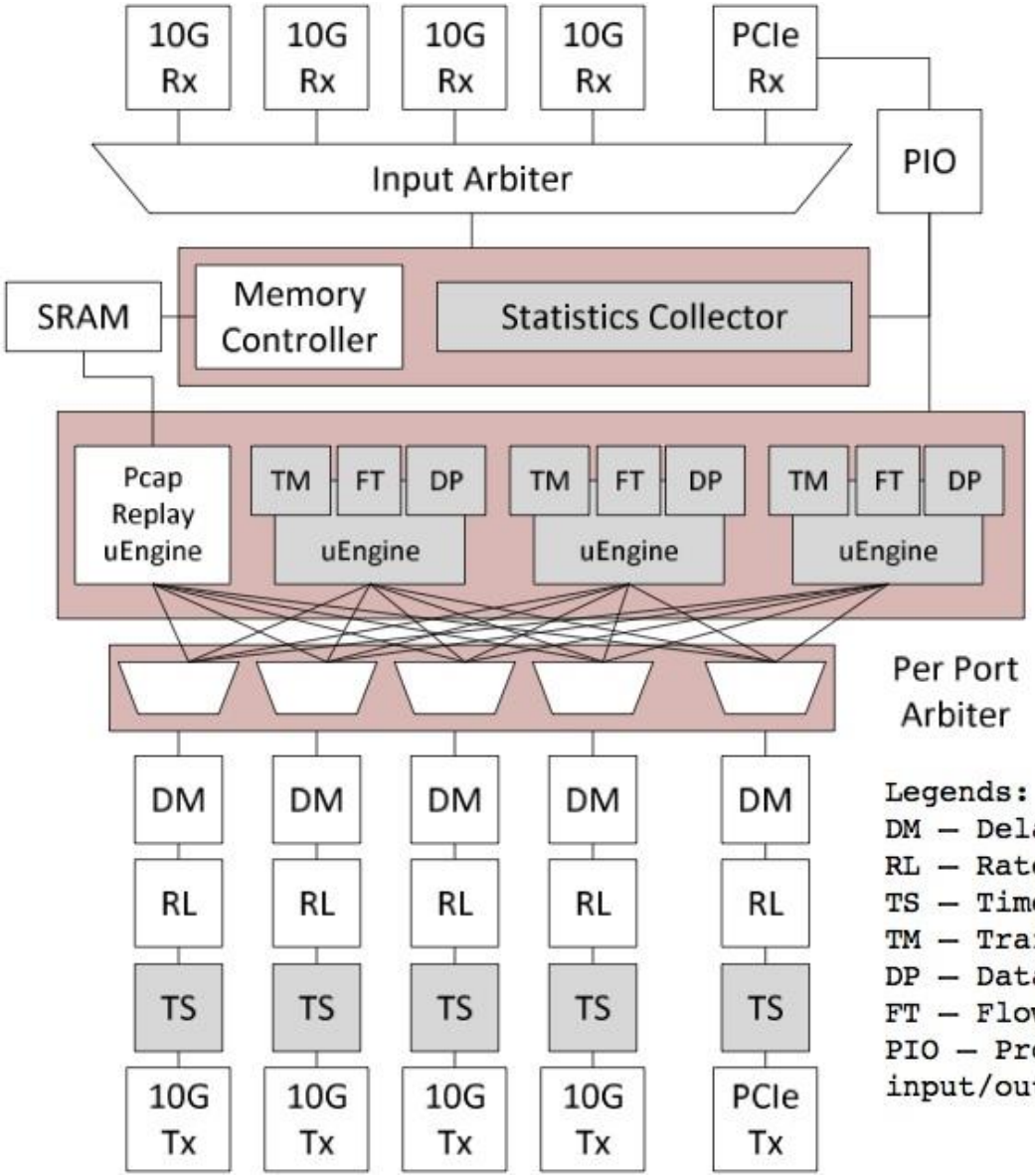
- **Scalable OSNT**
  - our approach for coordinating large numbers of multiple generators and monitors synchronized by a common time-base
  - still largely under work

OSNT-TG

The OSNT-TG generates packets according user-defined parameters

- **PCAP replay function**
- **micro-engines generate packets according (TBD)**
  - traffic model
  - list of flow values (header templates)
  - data patterns
- **generation process may depend on**
  - packet size
  - inter-packet delay

OSNT-TG architecture

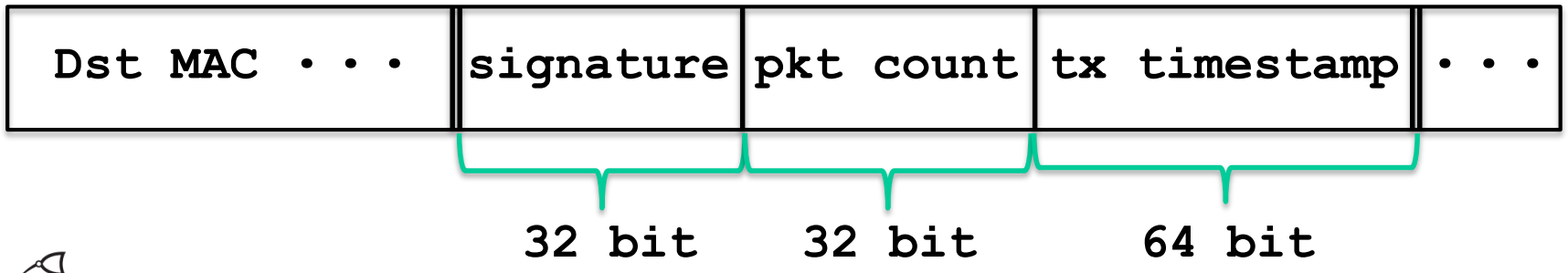


- DM and RL guarantee the output packet rate is the one assigned by the user
- 27MB of SRAM used to store the packets

OSNT-TG timestamp

Evaluating device functionalities using packet level information requires accurate timestamping functionality

- **timestamping just before the transmit 10GbE MAC**
- **configurable offset**
- **timing-related measurements**
  - latency
  - jitter



OSNT timestamp

free-running counter?

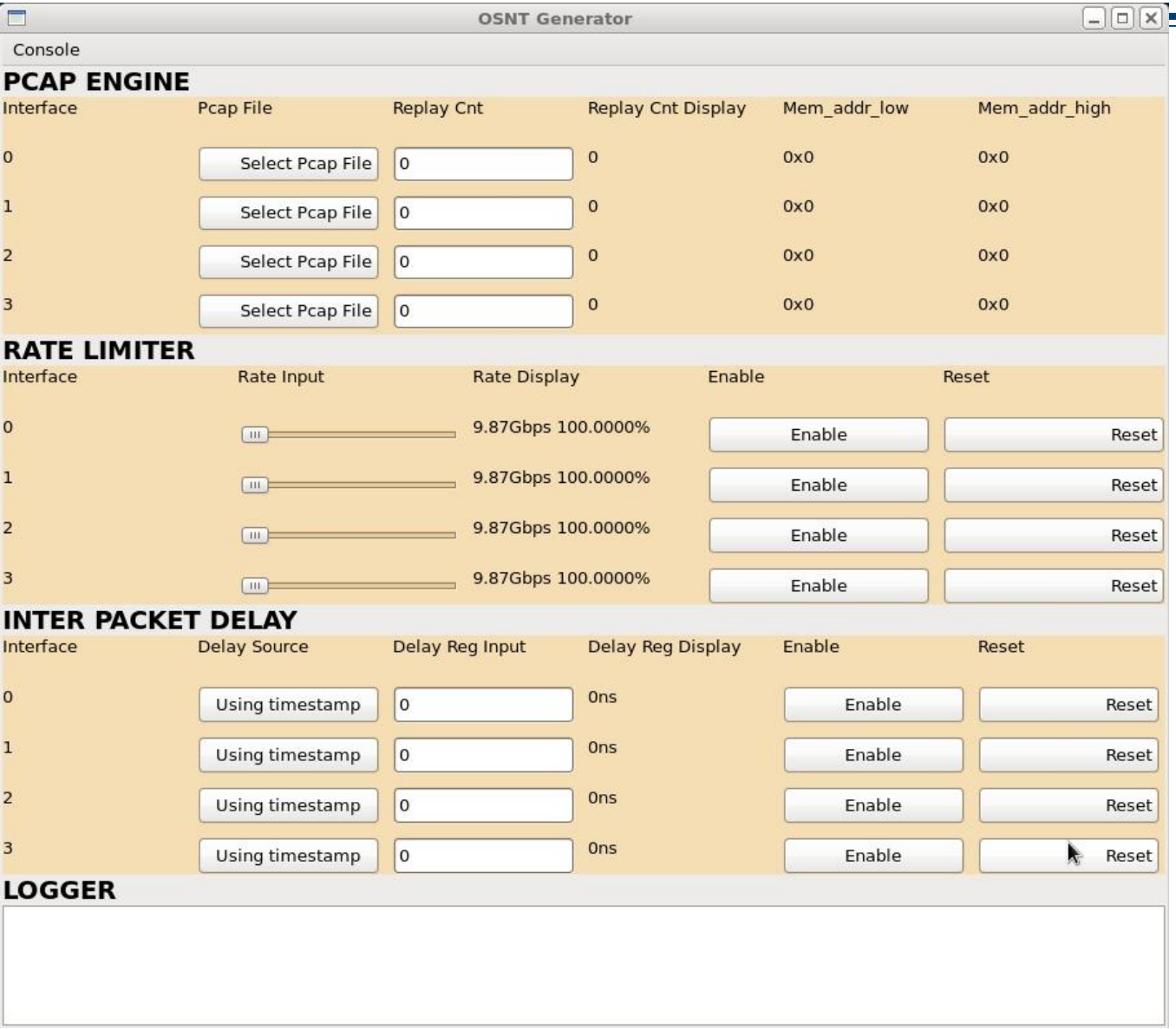
- **we could use a 64-bit counter driven by the 160MHz system clock (naïve solution)**
  - provides no means by which to correct oscillator frequency drift
  - produces timestamps expressed in unit of 6.25 ns
  - fixed-point representation of time in seconds more useful to host

OSNT timestamp

a more accurate solution…

- **DDS (Direct Digital Synthesis)**
  - technique by which arbitrary variable frequencies can be generated using FPGA-friendly logic (how DAG works)
  - need a time reference to correct DDS rate
  - optimal solution: PPS from GPS receiver

OSNT-TG GUI



- python GUI
- basic functionality management
- logger to track down last events

OSNT-TG evaluation

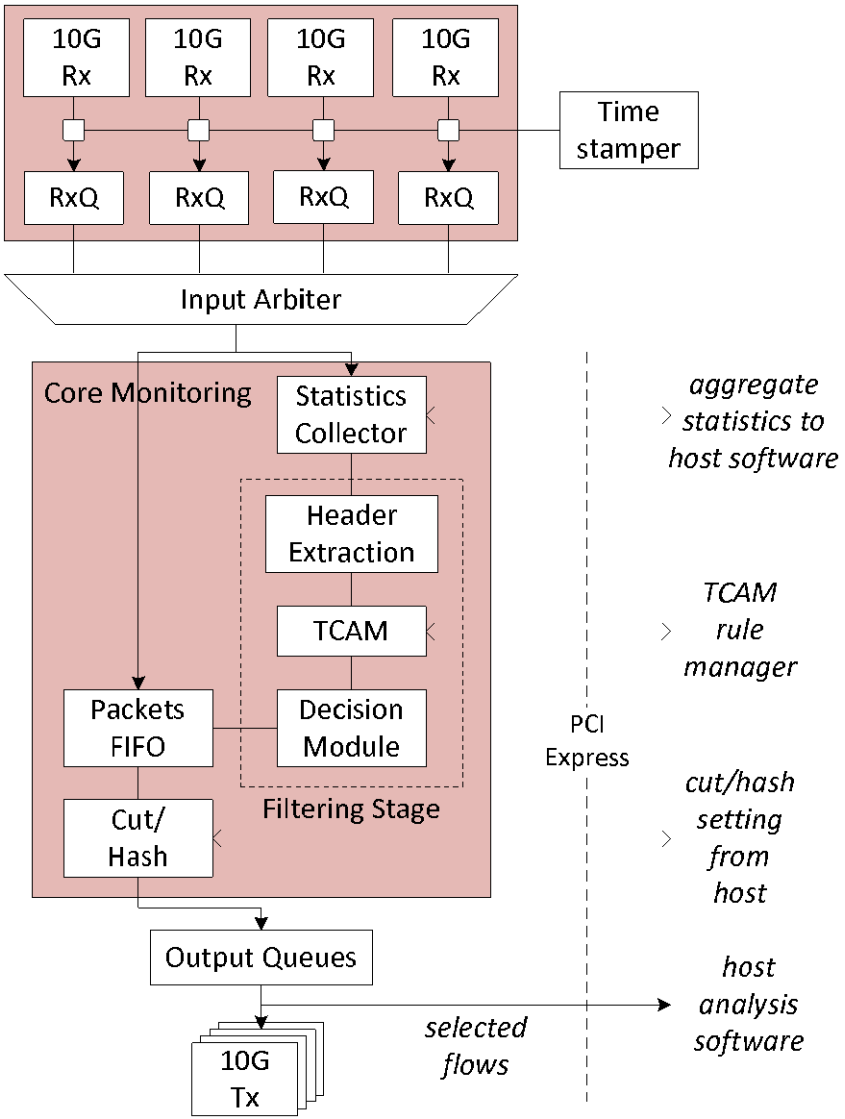
- performance tests against IXIA box
- full line rate regardless packet length on 2 ports
- full line rate over the 4 ports is work in progress (main limitation due to the Virtex5 FPGA resources)
- IFG (Inter Frame Gap) is statically set to 96 bit

OSNT-MON

The OSNT-MON provides four main functions

- **packet capture**
- **packet filtering permitting selection of traffic-of-interest (5-tuple)**
- **high precision, accurate, packet timestamping**
- **high-level traffic statistics**

OSNT-MON architecture



- timestamp before the receive queues
- statistic collector (packets, bytes, IP, UDP, TCP..)
- extensible packet parser able to recognize VLAN
- TCAM for packet filtering
- cut/hash feature

OSNT-MON architecture

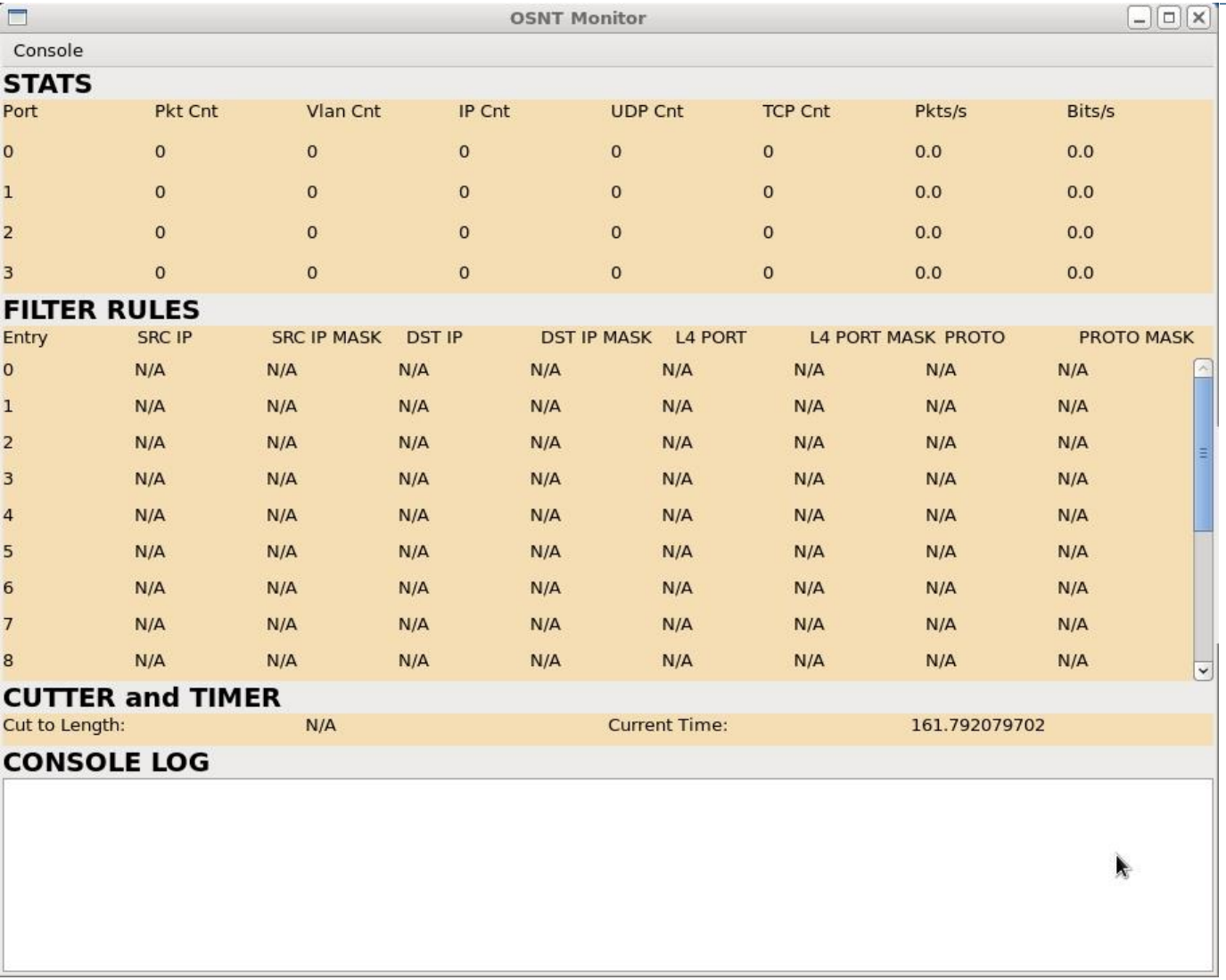
the NetFPGA-10G PCIe lacks the bandwidth to record all traffic

- **two traffic-thinning approaches**
  - packet filtering
  - snap length

- **5-Tuple filter stage (packets that match a rule are copied with their HW timestamp to the host)**

- **possibility of recording a fixed-length part of each packet along with a hash of the discarded part**

OSNT-MON GUI



- python GUI
- basic functionality management
- logger to track down last events

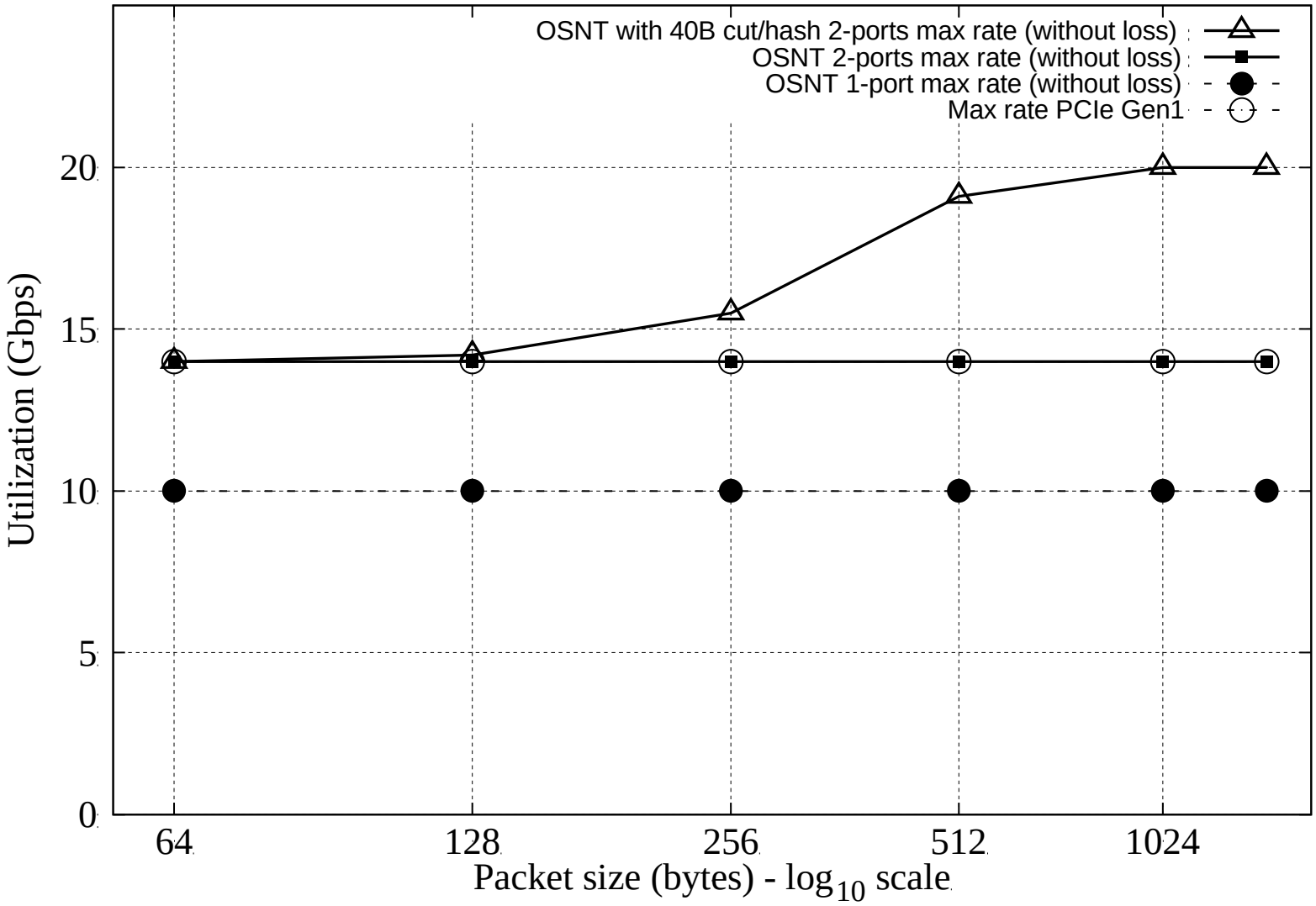
OSNT-MON SW plane

- **libpcap based tools do not work directly with OSNT: the device driver secures performance by bypassing the kernel network stack**
- **a modified version of libpcap with nanosecond granularity is provided to records PCAP traces in nanosecond resolution (if the appropriate user-space SW is written)**

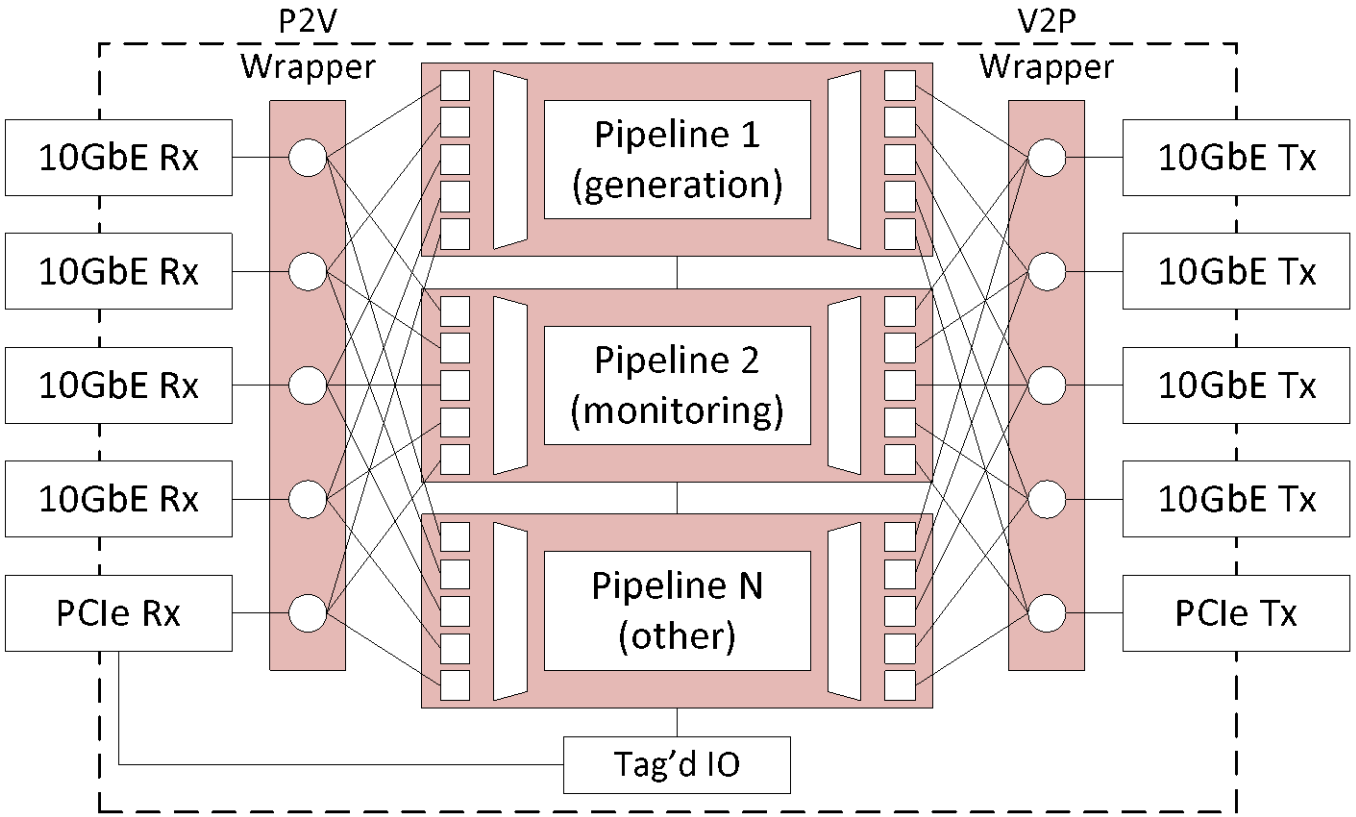
OSNT-MON SW plane

- **CLI-based approach (for those who do not like GUIs)**
  - set rules
  - check statistics
  - set snap-length
  - enable GPS correction

OSNT-MON evaluation



Hybrid OSNT



- multiple pipelines can co-exist
- it is possible to generate/monitor at the same time on a given port

Device Testing with OSNT

what can we do from here?

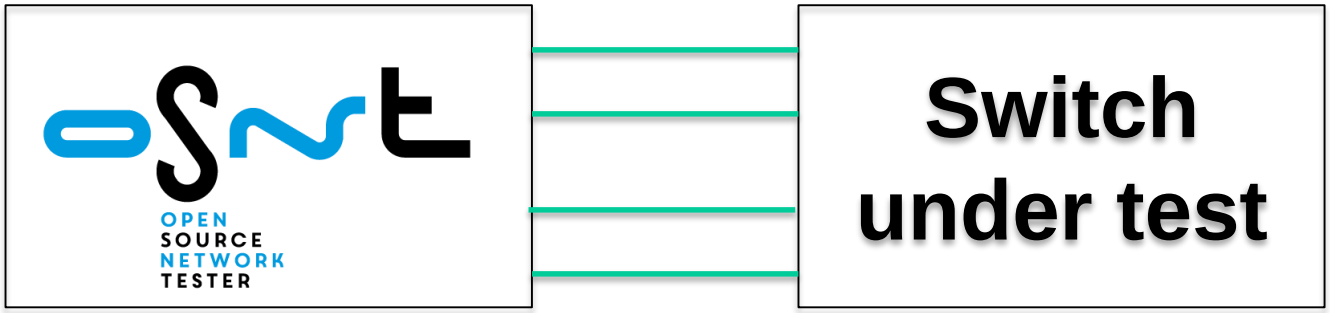
how can we effectively use OSNT?

- traffic characterization (OSNT is an high precision traffic capture system)
- networking device testing (OSNT is an high performance traffic generator)
- adapt OSNT to your needs (OSNT is open, OSNT is a starting point)
- **What about using OSNT for switch performance evaluation/characterization? (i.e., latency)**

Device Testing with OSNT

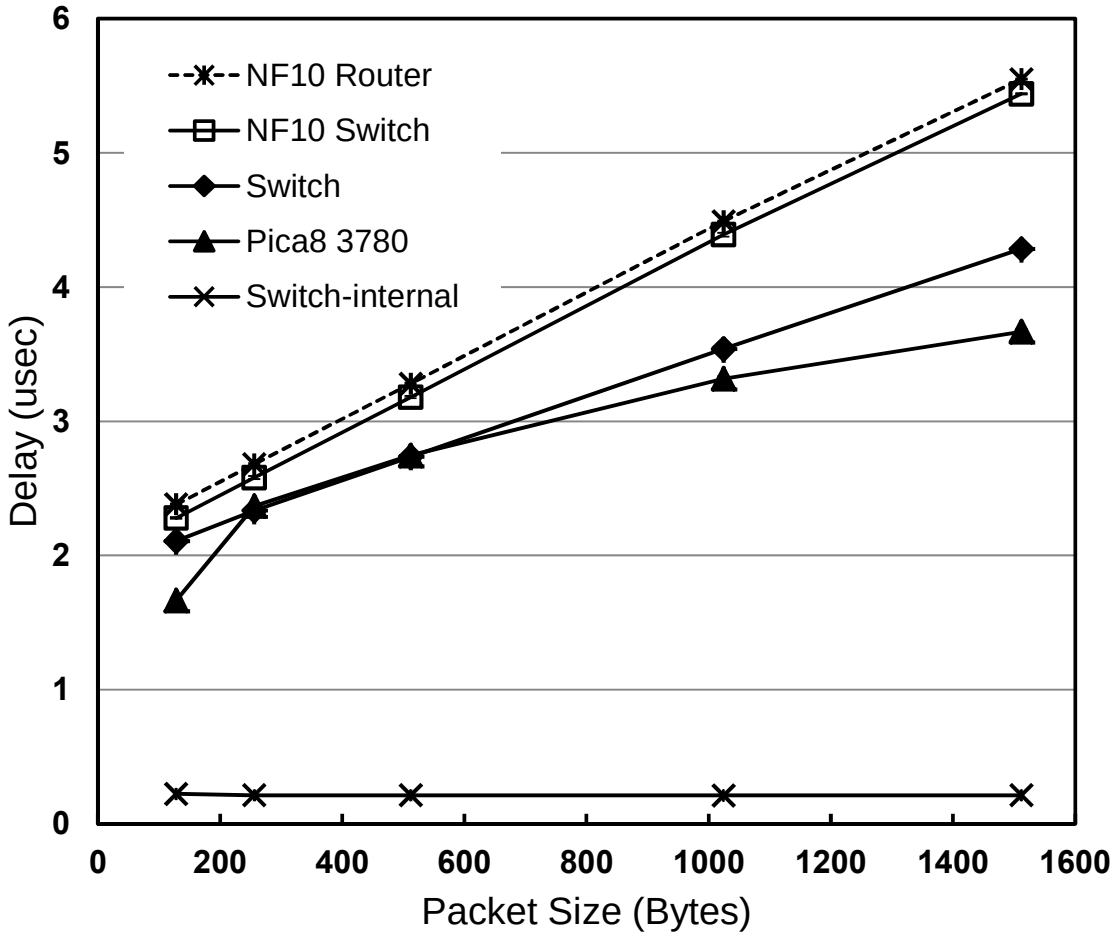
how is it possible to characterize a networking device latency with OSNT?

- we can embed the transmission timestamp into the packet
- OSNT can send packets at high rates and wait them back
- Compare the TX timestamp with the RX one.



Device Testing with OSNT

woooot!!!!! I can accurately measure switching latency!



ok…this is cool, but what's next?

- **participate, contribute to the open source network testing community**
- **extend OSNT with new features**

yes, ok..but…

- **Where can we go from here?**
- **How can we fully exploit OSNT?**

the effective integration of the OpenFlow protocol in production requires a flexible and high-precision open-source measurement platform which provide a deep understanding of switch capabilities

OFLOPS

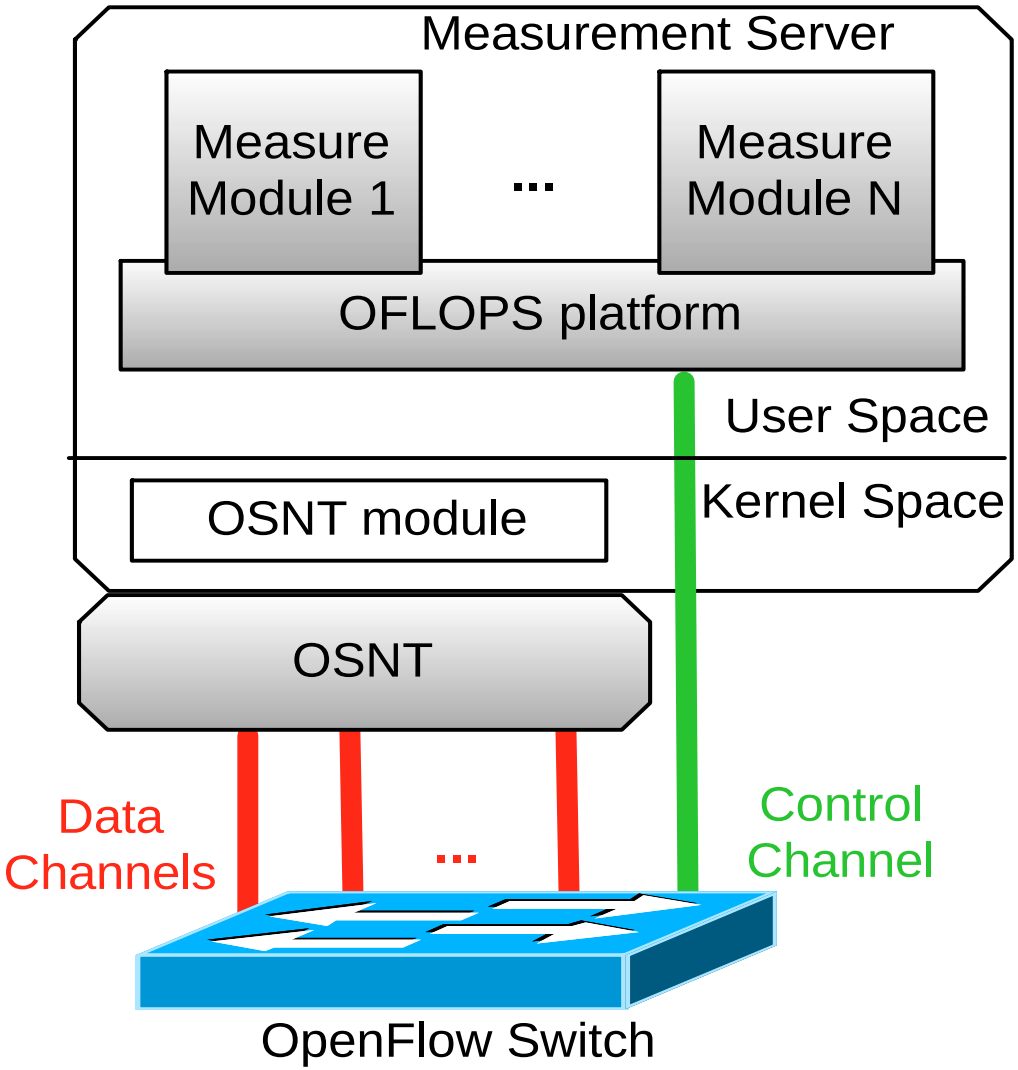
- **Holistic switch evaluation framework.**
  - API to interact with switch: SNMP, control and data plane.
  - Designed to minimize measurement noise.
- **Exploit OSNT traffic generation and capture capabilities (throughput, accuracy).**
- **Measurement modules to define experiment scenario and measurement.**
- **Use Cases:**
  - C. Rotsos, et.al. *OFLOPS: an open framework for openflow switch evaluation*. PAM'12
  - D. Pediaditakis, et.al. *Faithful reproduction of network experiments*. ANCS '14
  - J. Han, et.al. *Blueswitch: Enabling provably consistent configuration of network switches*. ANCS'15
  - C. Rotsos, et.al. *OFLOPS-Turbo: Testing the Next-Generation OpenFlow switch*. ICC'15

OFLOPS

- NetFPGA enables OSNT
- OSNT enables OFLOPS-Turbo

OFLOPS-Turbo

OFLOPS-turbo design

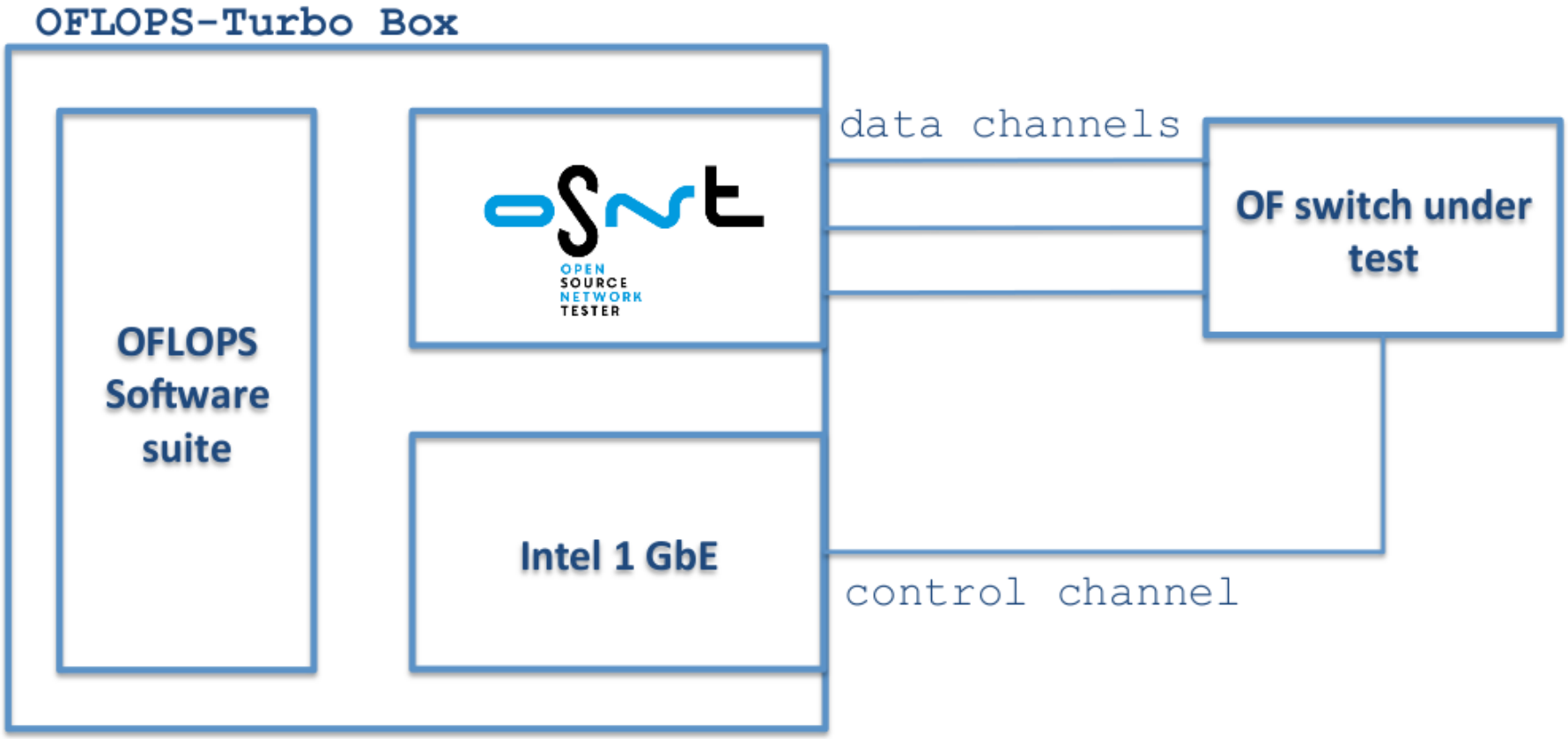


what can we do from here?

how can we effectively use OFLOPS-Turbo?

- **OpenFlow flow table insertion measurements**
- **OpenFlow flow table modification measurements**
- **Create your own test in SW and test multi Gigabit switches!**

Let's consider a testing scenario



Conclusion

Acknowledgments (I)

NetFPGA Team at University of Cambridge (Past and Present):

Andrew Moore, David Miller, Muhammad Shahbaz, Martin Zadnik
Matthew Grosvenor, Yury Audzevich, Neelakandan Manihatty-Bojan,
Georgina Kalogeridou, Jong Hun Han, Noa Zilberman, Gianni Antichi,
Charalampos Rotsos, Marco Forconesi, Jinyun Zhang, Bjoern Zeeb

NetFPGA Team at Stanford University (Past and Present):

Nick McKeown, Glen Gibb, Jad Naous, David Erickson,
G. Adam Covington, John W. Lockwood, Jianying Luo, Brandon Heller, Paul Hartke, Neda Beheshti, Sara Bolouki, James Zeng,
Jonathan Ellithorpe, Sachidanandan Sambandan, Eric Lo

All Community members (including but not limited to):

Paul Rodman, Kumar Sanghvi, Wojciech A. Koszek,
Yahsar Ganjali, Martin Labrecque, Jeff Shafer, Eric Keller ,
Tatsuya Yabe, Bilal Anwer, Yashar Ganjali, Martin Labrecque,
Lisa Donatini, Sergio Lopez-Buedo

Kees Vissers, Michaela Blott, Shep Siegel, Cathal McCabe

Acknowledgements (II)

Disclaimer: *Any opinions, findings, conclusions, or recommendations expressed in these materials do not necessarily reflect the views of the National Science Foundation or of any other sponsors supporting this project.*

This effort is also sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-11-C-0249. This material is approved for public release, distribution unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.