Azure Accelerated Networking: SmartNICs in the Public Cloud

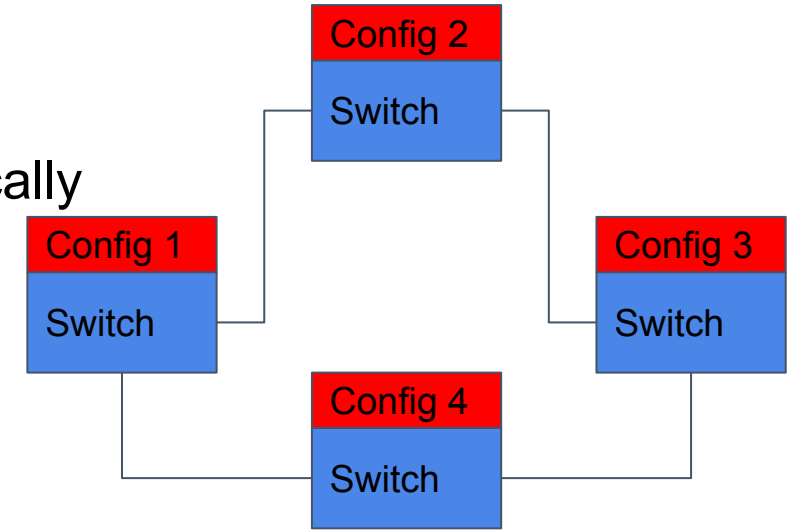
Developer | Hasnain Abdur Rehman| hasnain@bu.edu
Developer | Pierre-François Wolfe | pwolfe@bu.edu
Developer | Samyak Jain | samyakj@bu.edu
Developer | Suli Hu | sulihu@bu.edu
Developer | Yufeng Lin | yflin@bu.edu

Boston University CS & ECE

BOSTON UNIVERSITY

Traditional Network Policy Management

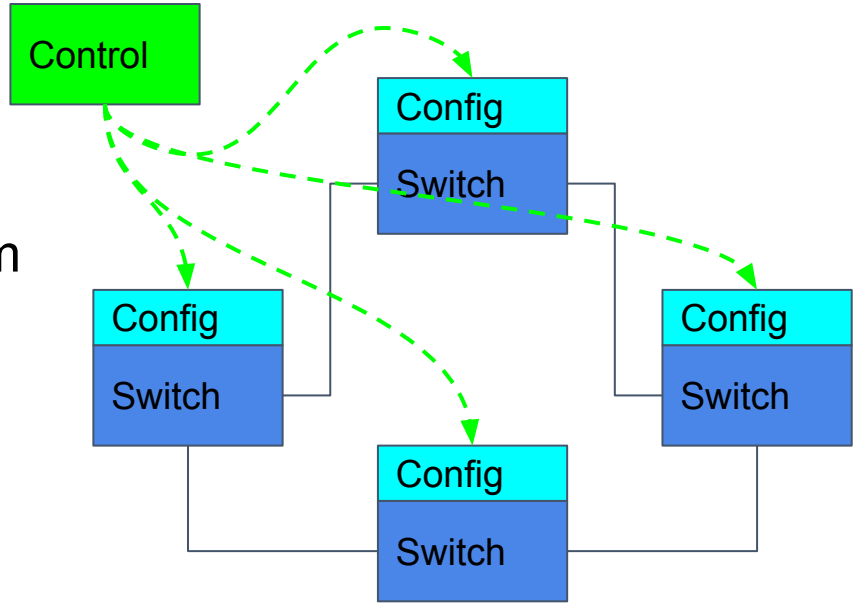
- Each switch can apply policy
- Each switch must be programmatically configured



BOSTON UNIVERSITY

Software Defined Network (SDN)

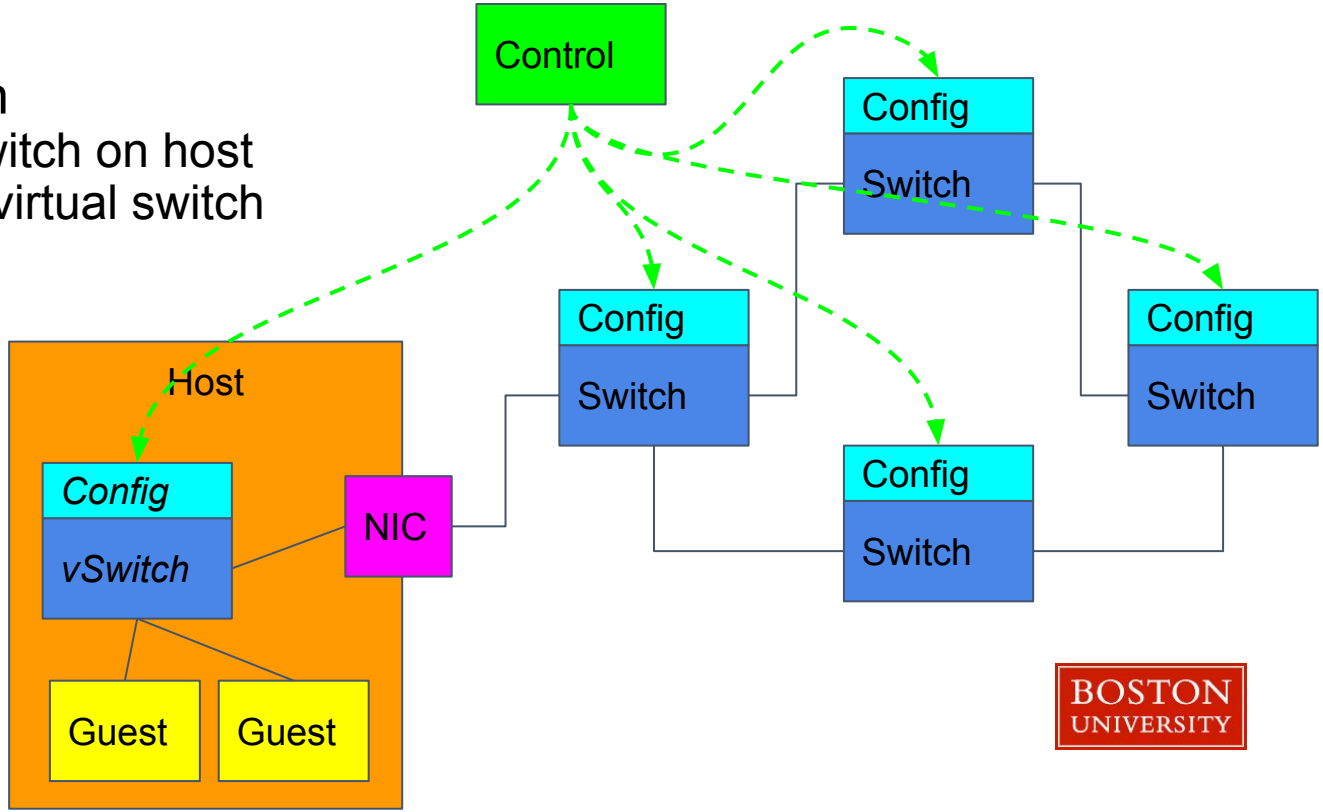
- Centralized Controller
- All switches can be updated from the controller



BOSTON UNIVERSITY

What about Virtualization?

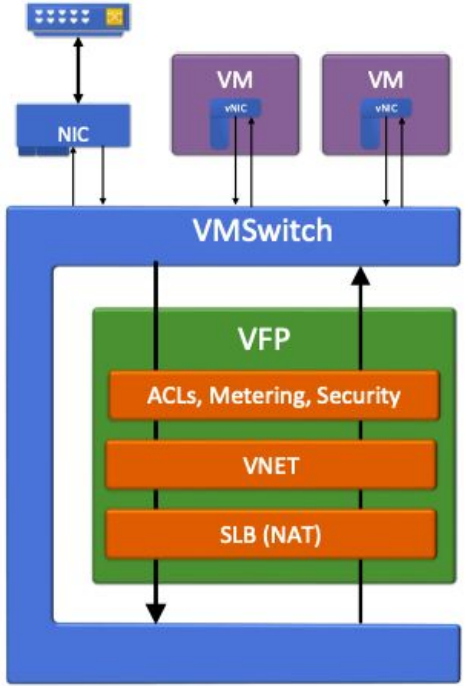
- Virtualization
  - Virtual switch on host
  - SDN via virtual switch



BOSTON UNIVERSITY

Virtual Filtering Platform (VFP)

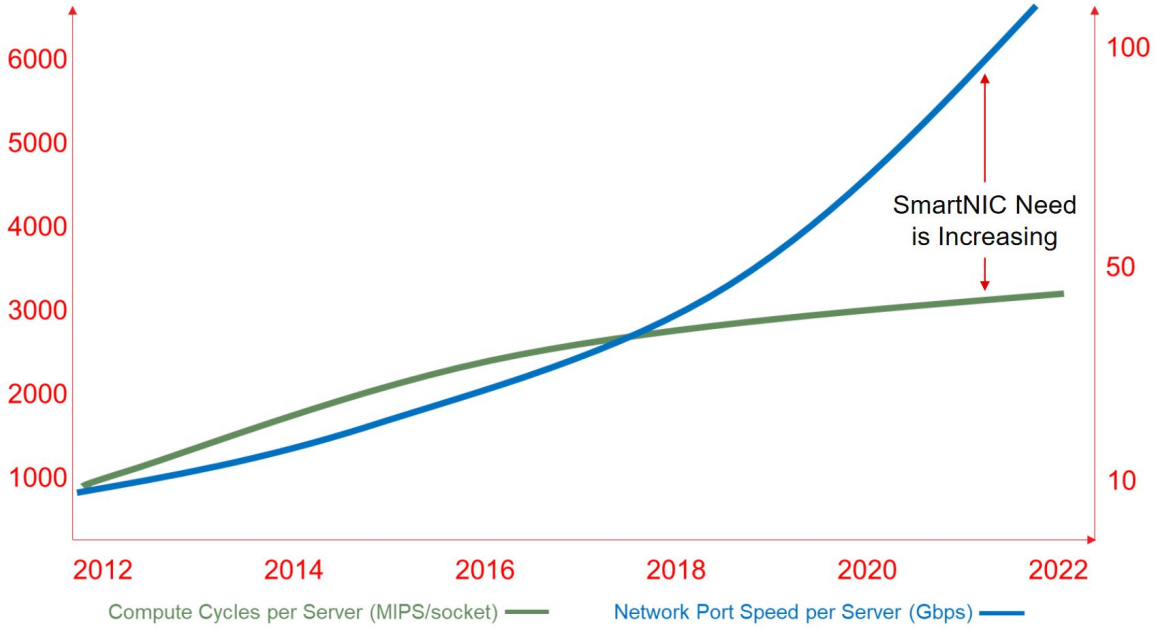
- Azure's programmable vSwitch implementing SDN stack
  - Acts as filtering engine for each VM's virtual NIC
  - Provides SDN programmability
- Uses flow tables for policy management (discussed later)
- Programmed by multiple Azure SDN controllers
- Handles all network I/O traffic to and from hardware
- Issues due to this
  - low throughput
  - high latency
  - increased processor utilisation



BOSTON UNIVERSITY

Trends in Compute and Network Speed

Compute cycles per server (MIPS/socket)



Network Port Speed per Server (Gbps)

How to solve this?

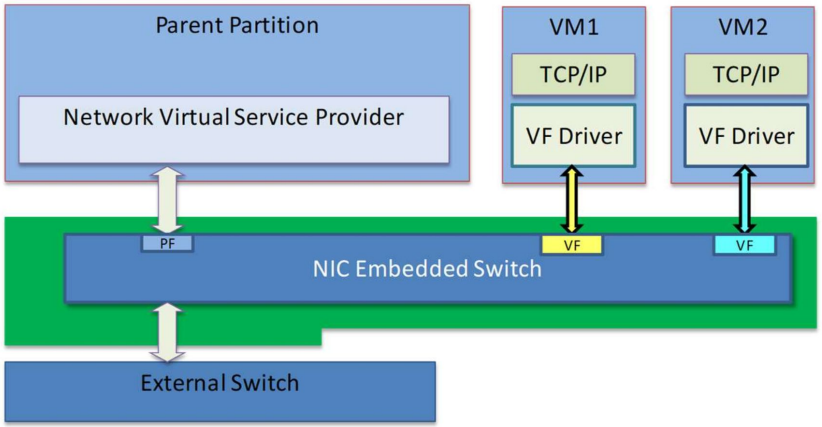
BOSTON UNIVERSITY

Image source: https://www.xilinx.com/publications/product-briefs/xilinx-smartNIC-media-briefing-final.pdf

How about with hardware Offload?

Solution: *Single Root I/O Virtualization (SR-IOV)*

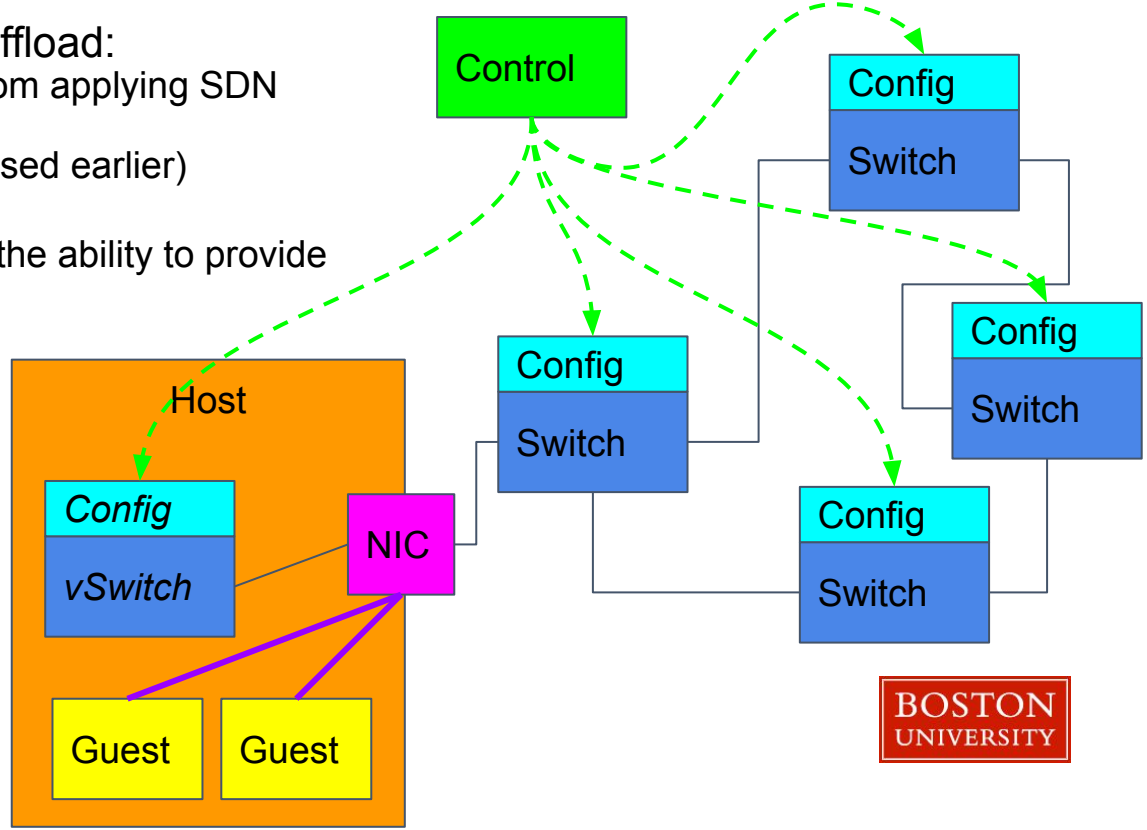
- Isolates PCIe resources to appear as separate physical devices for better manageability.
- Each VM connects to a unique Virtual Function (VF) isolating it from other VMs
- Bypasses host networking stack (“all or nothing” offload)
- Benefits
  - improved throughput
  - lower latency
  - reduced CPU utilisation
- Drawback
  - **NO SDN policy implementation!**



SR-IOV NIC

How about Hardware Offload providing VFP flexibility?

- Problem with hardware offload:
  - Prevents hypervisor from applying SDN policy
  - Like in SR-IOV (discussed earlier)
- Solution:
  - Hardware offload with the ability to provide SDN rules
  - Some options for this -
    - ASICs
    - FPGAs
    - SoCs
    - CPUs



BOSTON UNIVERSITY

Design goals

1. Don’t burn host CPU cores
   - Azure business depends on selling the VMs to customers as IaaS
   - Overheads due to running high-speed SDNs must be kept low for cost reduction
2. Maintain host SDN programmability of VFP
   - VFP flexibility should be maintained
   - Most rules don’t change during flow; offloading them is unnecessary
3. Achieve the latency, throughput and utilisation of SR-IOV hardware
4. Have a path to scale to 100GbE+
   - Initially, AccelNet deployed for 40GbE
   - SmartNIC should be scalable with increasing speed and number of VMs

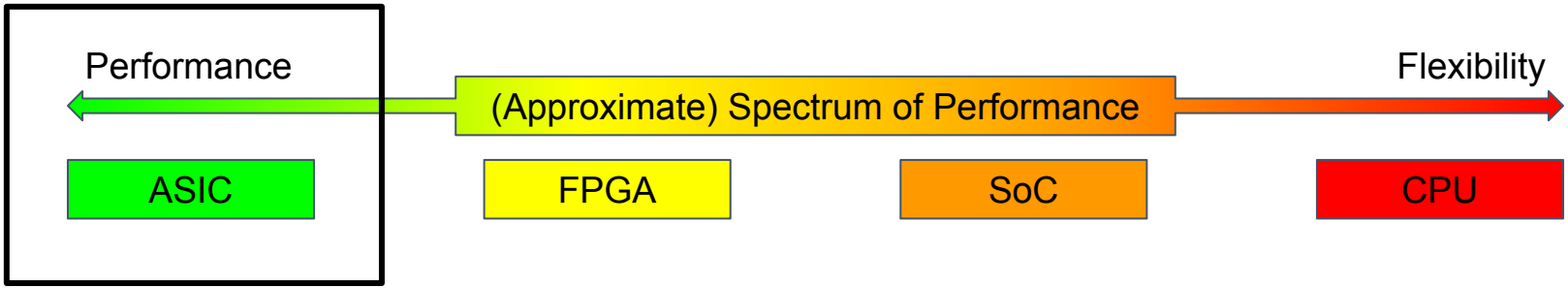
Additional Design Goals

5. Support new SDN primitives and workloads overtime
6. Rollout new functionality to the entire fleet
7. Provide high single-connection performance
8. Retain serviceability

Offloading SDN Network Operations

- Hardware Options
  - CPU (Central Processing Unit)
  - SoC (System on a Chip)
  - FPGA (Field Programmable Gate Array)
  - ASIC (Application-Specific Integrated Circuit)
- FPGA
  - How they function and are used
  - Azure SmartNIC architecture

Hardware Options



- Application-Specific Integrated Circuit (ASIC)
  - Advantages:
    - Greatest optimization and performance potential
  - Disadvantages:
    - Can't be changed after it's made
    - Longest development time

Traditional ASIC Offload Tasks

- 1990s - **TCP offloads**: checksum/segmentation → TCP Offload Engine (TOE)
- 2000s - **Steering**: Receive-Side Scaling (RSS), Virtual Machine Queues (VMQ)
  - Multi-Core Scalability
- 2010s - **Encapsulation**: Network Virtualization using Generic Routing Encapsulation (NVGRE) Task Offload, Virtual Extensible LAN (VxLAN)
  - Stateless offloads for virtual networking
- **Afterwards...**
  - Generic Flow Tables (GFT) originally for ASICs for use with SR-IOV
  - However: SR-IOV → all-or-nothing offload

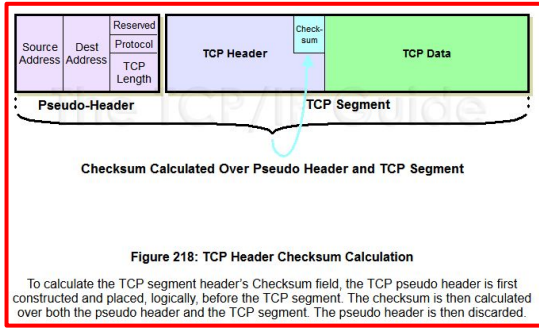
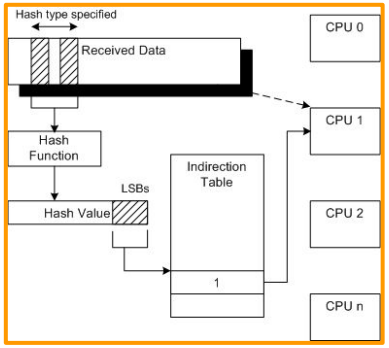
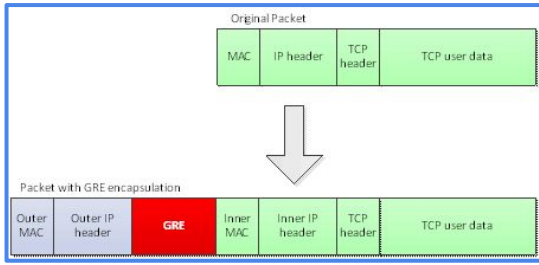


Figure 218: TCP Header Checksum Calculation

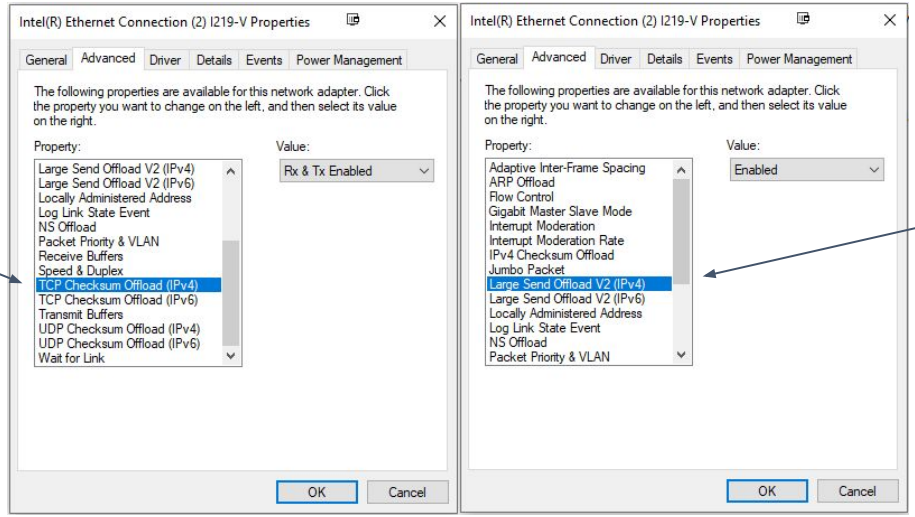
To calculate the TCP segment header's Checksum field, the TCP pseudo header is first constructed and placed, logically, before the TCP segment. The checksum is then calculated over both the pseudo header and the TCP segment. The pseudo header is then discarded.





Example of NIC capabilities

TCP Checksum Offload

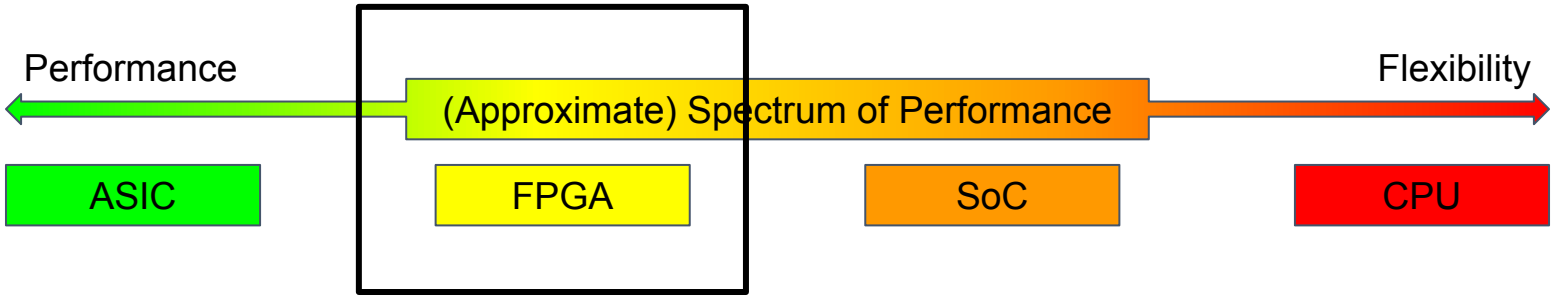


Segmentation Offload

Some offloads are supported in normal NICs today

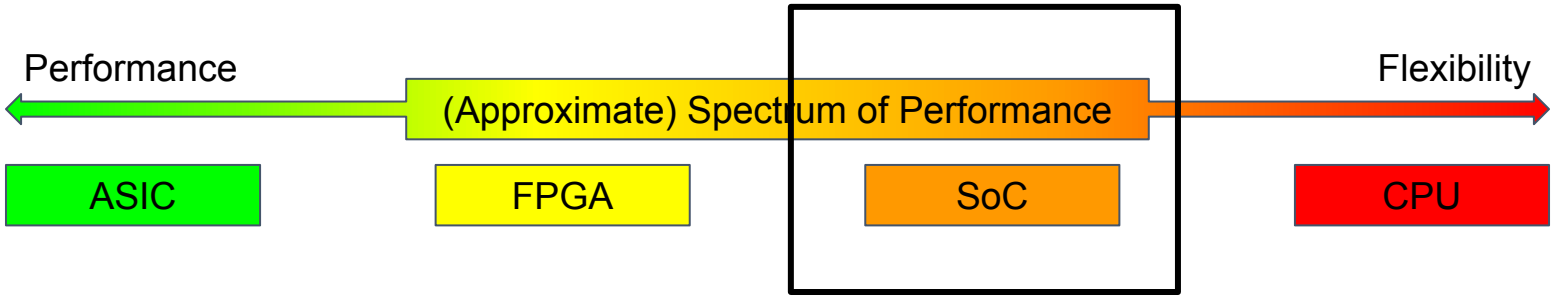
BOSTON UNIVERSITY

Hardware Options



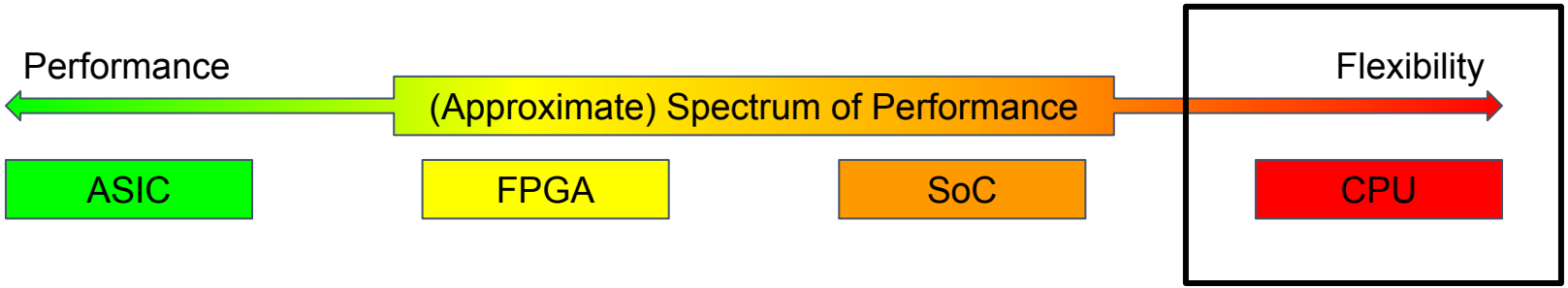
- Field Programmable Gate Array (FPGA)
  - Advantages:
    - High performance potential (low latency and parallel processing)
    - Modifiable/reconfigurable hardware
  - Disadvantages:
    - Still requires hardware knowledge for best performance

Hardware Options



- System on a Chip (SoC)
  - Advantages:
    - Software programmable (familiar to software developers, flexible)
  - Disadvantages:
    - Less performance
    - More limited scalability

Hardware Options



- Central Processing Unit (CPU)
  - Advantages:
    - Least effort required, current approach
  - Disadvantages:
    - To scale, more cores need to be burned for network processing

Is There Room for Different Types of SmartNIC?

- ASIC Based
  - Excellent price-performance
  - Vendor development cost high
  - Programmable and extensible
    - Easy to program but flexibility is limited to pre-defined capabilities

- FPGA Based
  - Good performance but expensive
  - Very difficult to program
  - Workload specific optimization

- SOC Based

  System on Chip - NIC + CPU
  - Good price-performance
  - C Programmable Processors
  - Highest Flexibility
  - Easiest programmability

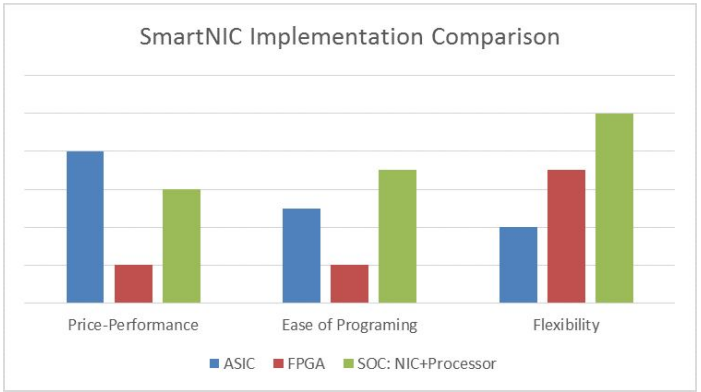


Image source: https://blog.mellanox.com/2018/08/defining-smartnic/

BOSTON UNIVERSITY

… Yes!

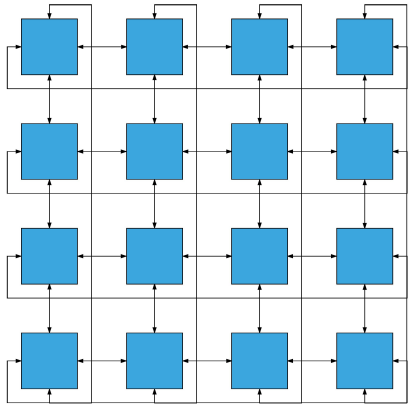
Capability	Workloads Accelerated	Foundational NIC	Intelligent NIC (iNIC)	SmartNIC
Entry Level Virtualization and Data Movement				
TCP/IP Acceleration	Enterprise workloads	✓	✓	✓
NIC Virtualization (SR-IOV)	Enterprise workloads	✓	✓	✓
Network Virtualization (VXLAN)	Multi-tenant workloads	✓	✓	✓
Data Transport Acceleration & Programmable Data Plane				
RoCE Acceleration	VM, Storage, Big Data, AI/ML		✓	✓
DPDK	Virtualized Network Functions		✓	✓
Spark Acceleration	Big Data		✓	✓
NVMe-over-Fabrics Storage	Storage		✓	✓
OVS Hardware Acceleration	Efficient, Scalable Virtualized Apps		✓	✓
QoS and ACL Acceleration	Web Servers / Content Distribution Networks		✓	✓
Flow monitoring/reporting	Visibility, Network Packet Broker, IBN		✓	✓
Flow match/action engine	Software Defined Networking		✓	✓
Fully Programmable Data Plane	Network Function Virtualization		✓	✓
ASAP^2 Virtual Switching/Routing	NAT, Load Balancing, stateless firewall		✓	✓
Smart Networking & Virtualization				
Virtual Switch Policy Engine	OVS Control Plane			✓
Analytics engine	DPI, Network Monitoring and Diagnostics			✓
Container Acceleration	Various (AI/ML, Big Data, Analytics)			✓
Smart Cloud Virtualization	Server Disaggregation & Resource Sharing			✓
Security, Compression, Network Function Virtualization, Storage				
Public Key Crypto, RNG	Authentication, Key Exchange			✓
Fault Domain Isolation/HA	Bare Metal Cloud			✓
Stateful IP/ACL filtering	Load Balancing IPD/IDS/UTM			✓
Storage: Hashing, ECC, Compression	Erasure Coding, Thin Provisioning, Dedup			✓
Encryption/Decryption	Data at Rest or on-the-fly			✓
Security VNF Offloading	Firewall, IDS, IPS, Anti-malware, Anti-DDoS			✓

It depends on the user and the task.

Image source: https://blog.mellanox.com/2018/08/defining-smartnic/

SmartNIC System Architecture

- Catapult (Original Version)
  - FPGAs mapped in 6x8 2D torus
  - Secondary network (within a rack)
- Azure SmartNIC
  - Directly connect to datacenter network
  - Bump-in-the-wire FPGA (between NIC and TOR)



2D Torus and Catapult Mapping

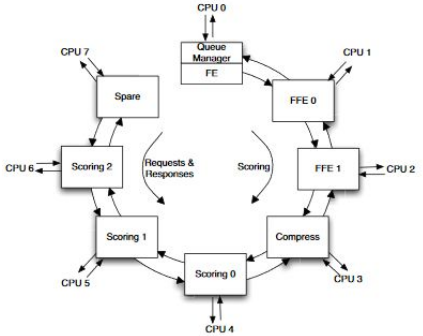
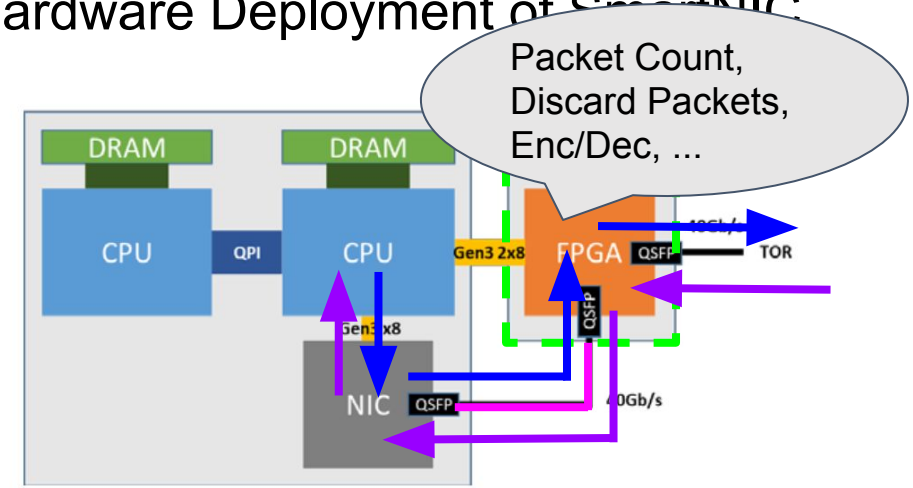
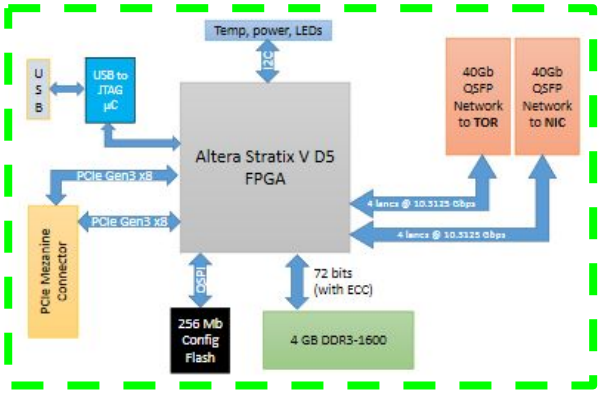


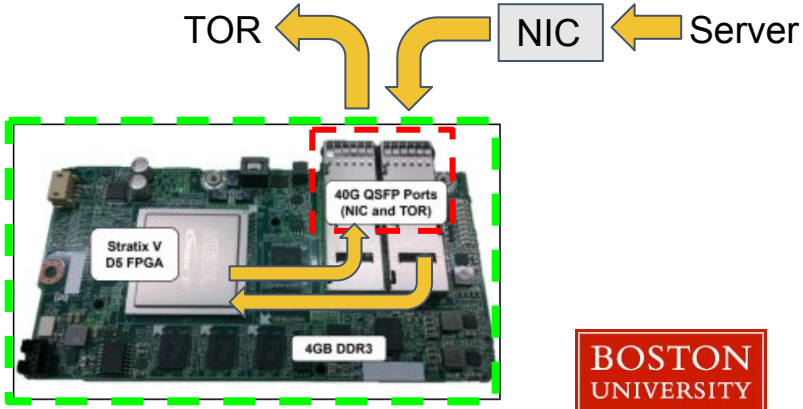
Image Source: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/Catapult_ISCA_2014.pdf
https://en.wikipedia.org/wiki/Torus_interconnect

Hardware Deployment of SmartNIC



(c) *bump-in-the-wire* architecture





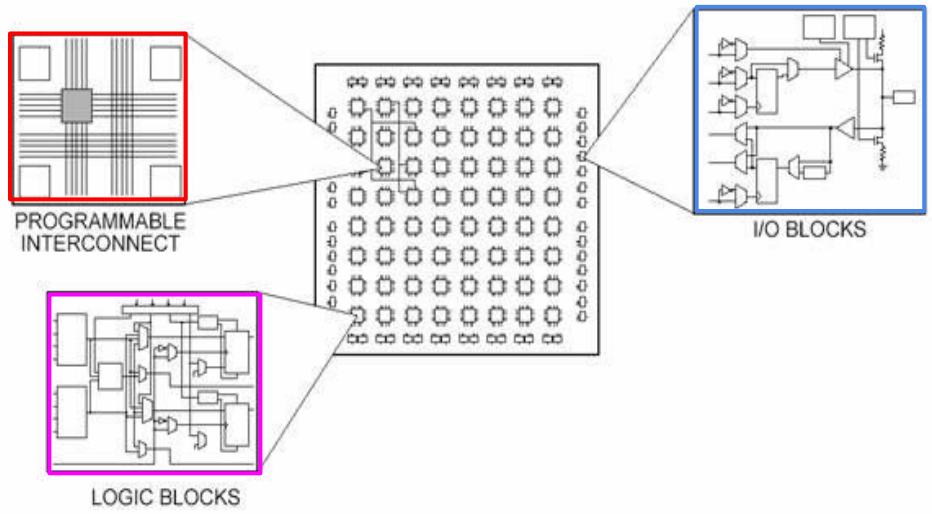
(a) Azure SmartNIC Gen1, 40GbE w/ external NIC

- Gen1: Very visible “bump-in-the-wire”
- Gen2: The “wire” is now on a PCB

BOSTON UNIVERSITY

What are Field Programmable Gate Arrays (FPGAs)?

- User configurable
  - HDL (Hardware Definition Language)
  - HLS (High Level Synthesis)
- Hardware Elements
  - CLBs (Configurable Logic Blocks)
    - Flip-Flops
    - LUTs (Look-up tables)
    - Block RAM
    - Multipliers/DSP (Digital Signal Processing)
  - Interconnect
  - Input/Output



BOSTON UNIVERSITY

Image source: https://www.ni.com/en-us/innovations/white-papers/08/fpga-fundamentals.html

HDL Example - AND Gate

Input 1	Input 2	output
0	0	0
0	1	0
1	0	0
1	1	1


```
-- VHDL Example
library ieee;
use ieee.std_logic_1164.all;

entity example_and is
  port (
    input_1   : in  std_logic;
    input_2   : in  std_logic;
    and_result : out std_logic
    );
end example_and;

architecture rtl of example_and is
  signal and_gate : std_logic;
begin
  and_gate    <= input_1 and input_2;
  and_result <= and_gate;
end rtl;
```

Defining the Interface (Inputs and Outputs)

Internal Signal

Combinational Logic

```
// Verilog Example
module example_and_gate
  (
    input_1,
    input_2,
    and_result);

input  input_1;
input  input_2;
output and_result;

wire   and_temp;

assign and_temp = input_1 & input_2;

assign and_result = and_temp;

endmodule
```


BOSTON UNIVERSITY

Reference: https://www.nandland.com

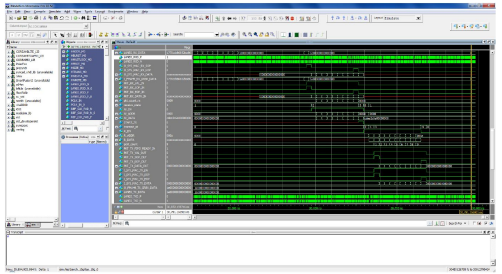
Example FPGA Programming Flow

1. Write HDL (or HLS → HDL)
   a. Behavioral
   b. RTL (Register-Transfer Level)
2. Simulate/Verify Behavior
   a. e.g. Modelsim
3. Synthesize
   a. e.g. Vivado, Quartus
4. Implement Design
5. Timing Analysis
6. Generate bitstream
7. Program
8. In-Circuit (hardware) Test

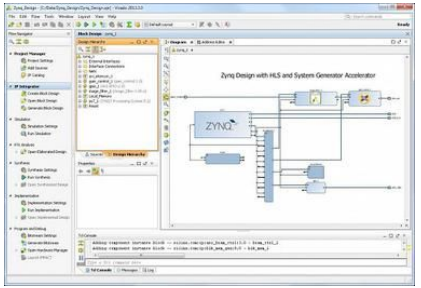
HDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in   std_logic;
    clk  : in   std_logic;
    a    : in   std_logic_vector;
    b    : in   std_logic_vector;
    q    : out  std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
  assert(a'length >= b'length)
    report "Port A must be the longer vector if different sizes!"
    severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
    begin
      if (aclr = '1') then
        q_s <= (others => '0');
      elsif rising_edge(clk) then
        q_s <= ('0'&signed(a)) + ('0'&signed(b));
      end if; -- clk'd
    end process;

end signed_adder_arch;
```

Simulate



Synthesize
Implement
Timing Check
Generate bitstream



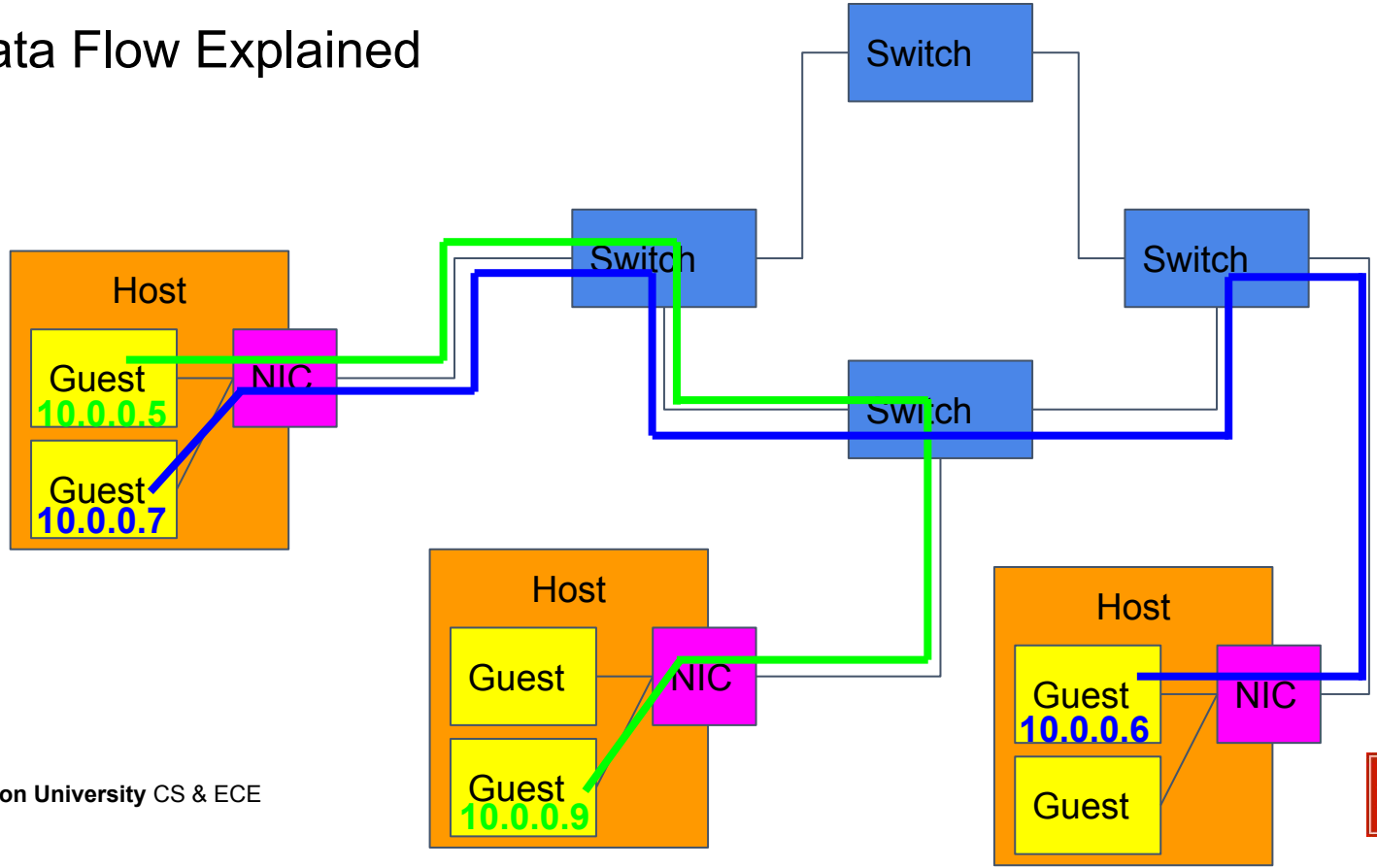
Program
Test

Boston University CS & ECE

Image sources:
https://www.ni.com/en-us/innovations/white-papers/08/fpga-fundamentals.html
https://www.microsemi.com/product-directory/dev-tools/4900-modelsim
https://en.wikipedia.org/wiki/Xilinx_Vivado

*notional flow, not actually the same project

Data Flow Explained



BOSTON UNIVERSITY

Handling Flow Tables - The Key to Hardware Acceleration

- Cascading Flow Tables for Rules (Software)
- Single Action Determined
- Cached on FPGA in Hash Table

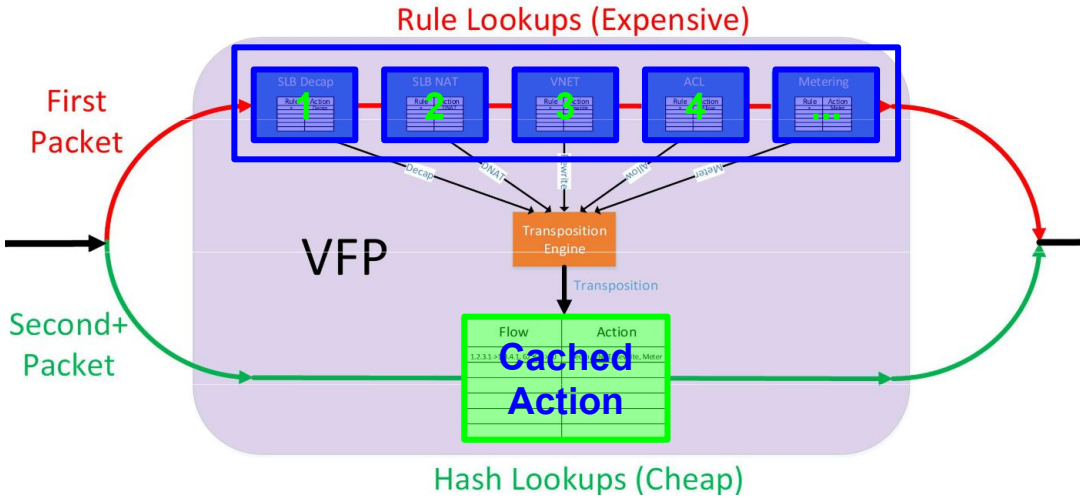
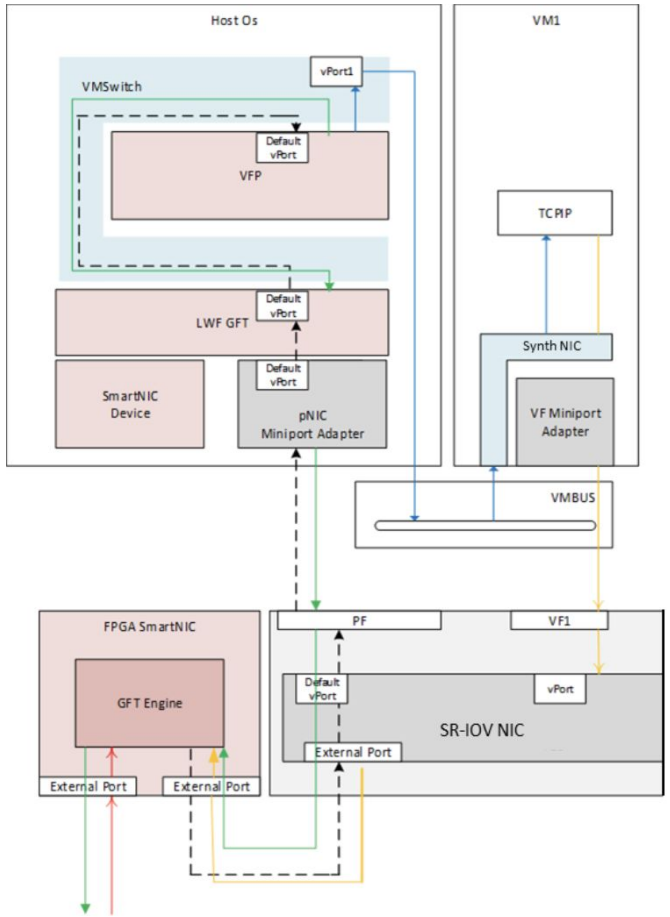


Image Source: https://www.usenix.org/sites/default/files/conference/protected-files/nsdi18_slides_firestone.pdf

AccelNet System Design

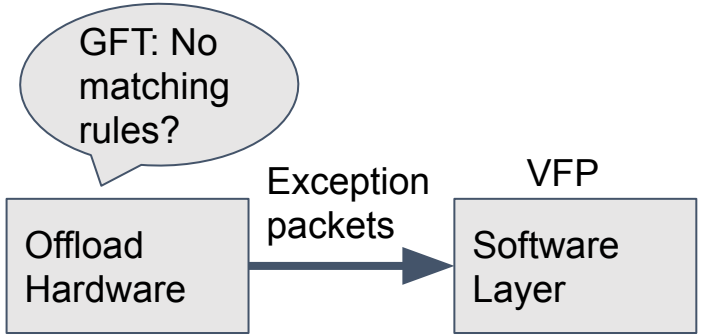
- Software design
- FPGA pipeline design
- Flow tracking and reconciliation



Software Design of AccelNet

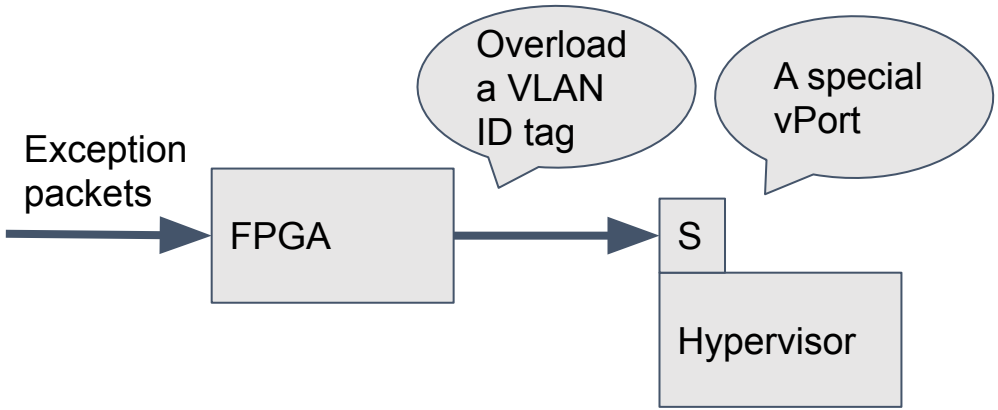
Acronym Recap: Generic Flow Table (GFT)
Virtual Filtering Platform (VFP)

- Control Operations
- Handle exception packets
- Establish special virtual port (Vport)
- Terminate stale connection



BOSTON UNIVERSITY

What happened when FPGA receives exception packets?

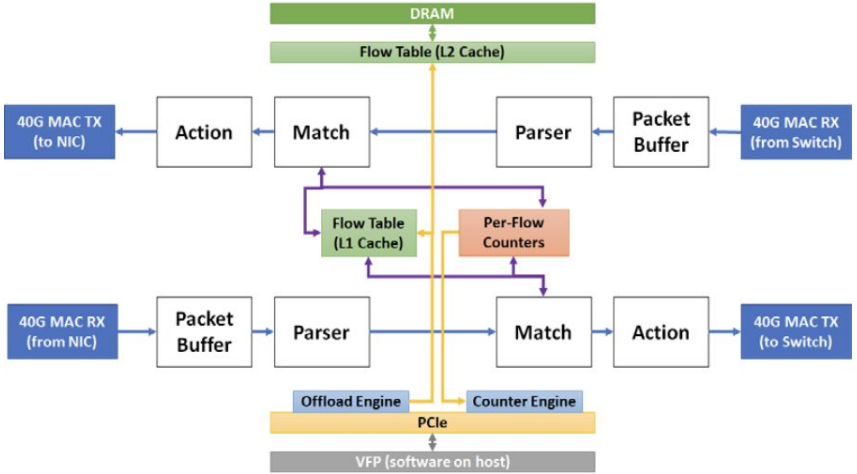


(802.1Q VLAN ID tag was overloaded)

BOSTON UNIVERSITY

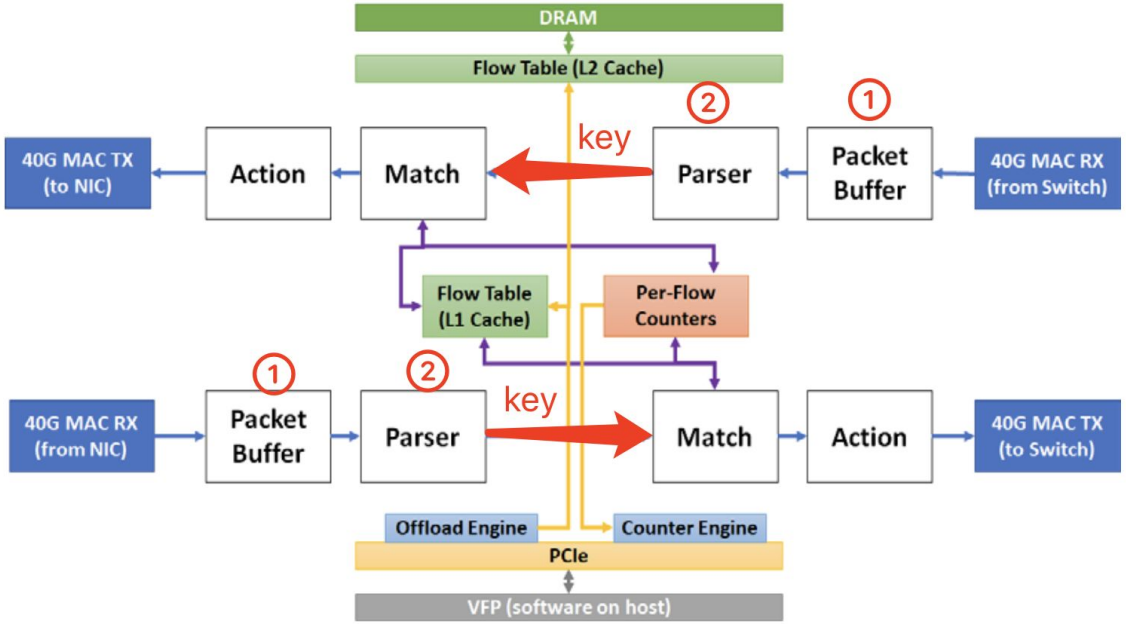
FPGA pipeline design

- Pipeline Stages:
  - a store and forward packet buffer
  - a parser
  - a flow lookup and match
  - a flow action



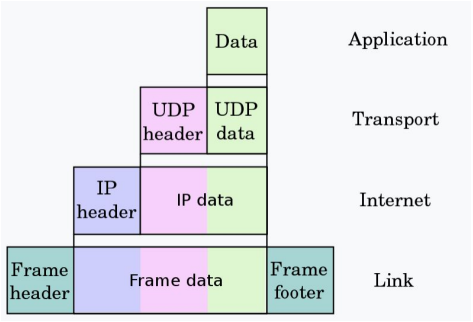
BOSTON UNIVERSITY

① Packet Buffer Stage & ② Parser Stage



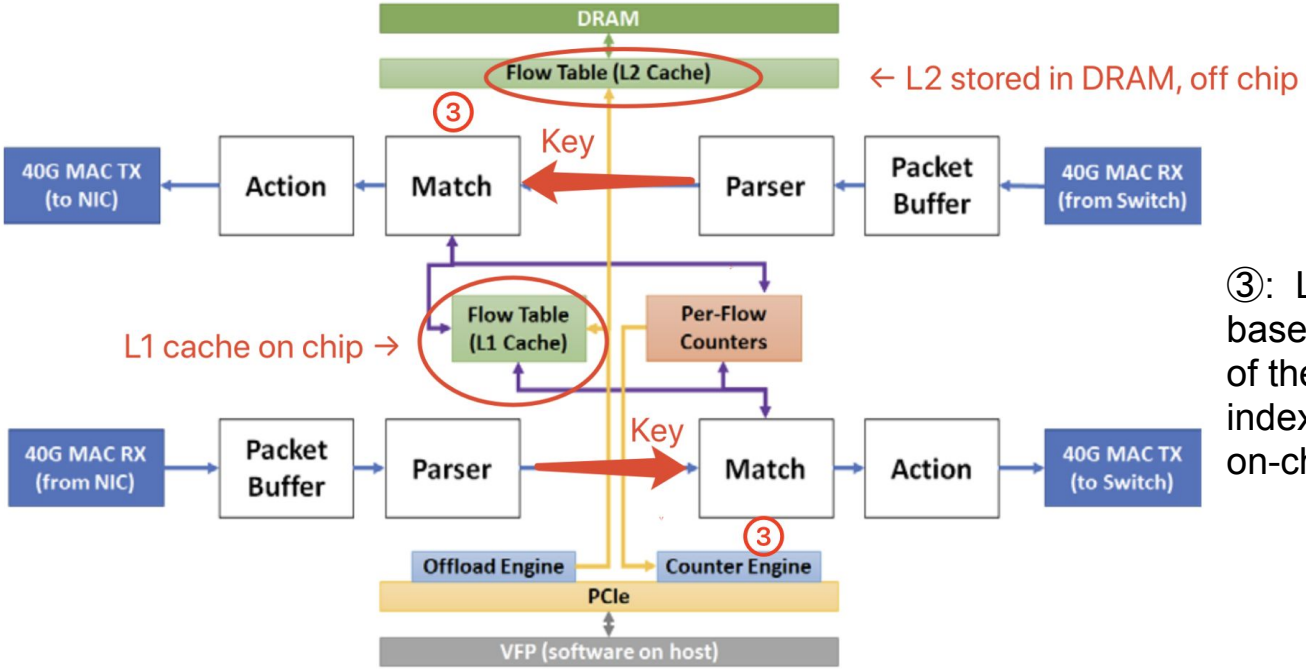
①: Packets store temporary here and forward them to next stage

②: Combine the header information of a packet, generate unique for each flow and pass it to next stage



Encapsulation

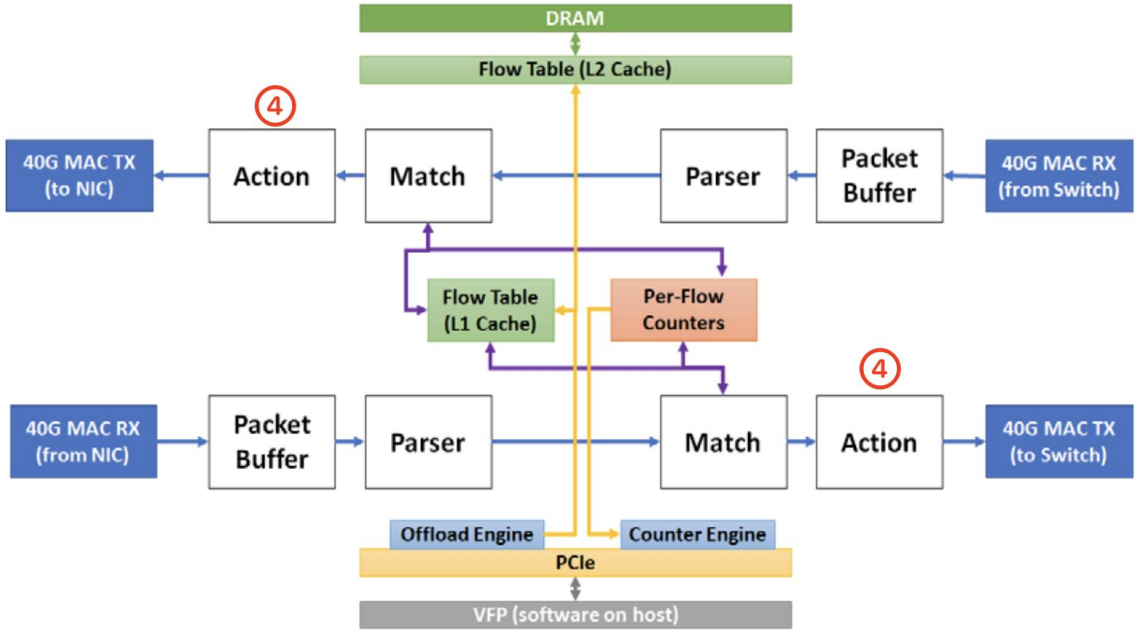
③ The Match Stage



③: Look up rules for packet based on the key. Compute hash of the key and use it as cache index. It has 2 caches, one is on-chip with another one off-chip.

BOSTON UNIVERSITY

④ The action stage

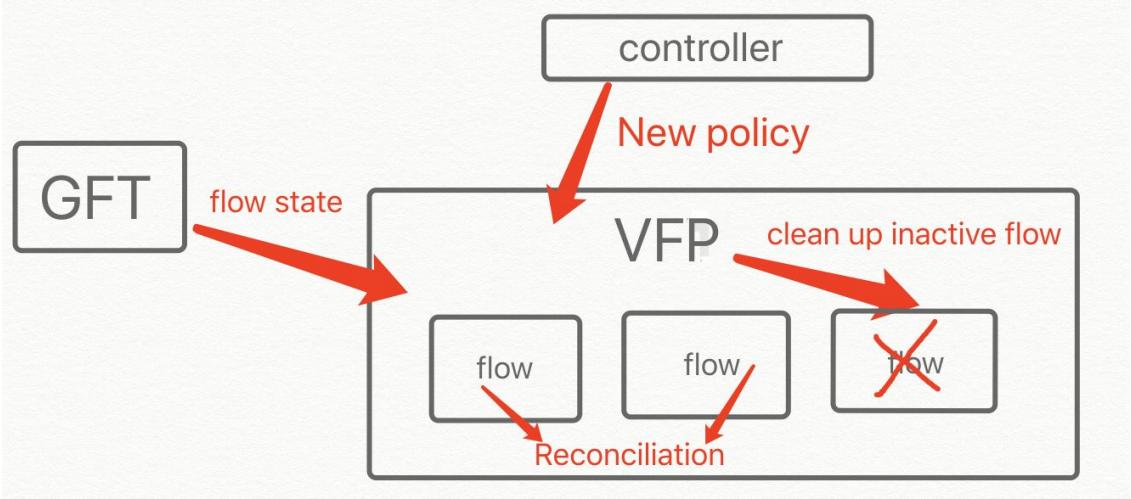


④: Take the parameters looked up from flow table, perform transformations on the packet header.

BOSTON UNIVERSITY

Flow tracking and reconciliation

- Virtual Filtering Platform (VFP)
  - Overlying controllers
  - Monitor layers
  - Flows do reconciliation to update actions
- Generic flow table (GFT)
  - Keep track of per-connection data usage
    - TCP sequence
    - Connection state

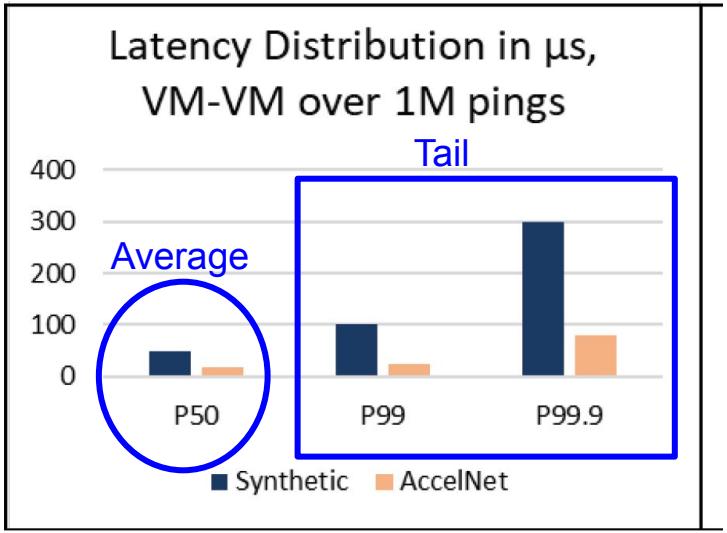


Performance

- What was measured (and presented) ?
  - VM-VM latency distribution
  - Single Connection Throughput
  - An Example - SQL Query Time
  - Gateway Forwarding Latency

Performance - VM to VM Latency Distribution

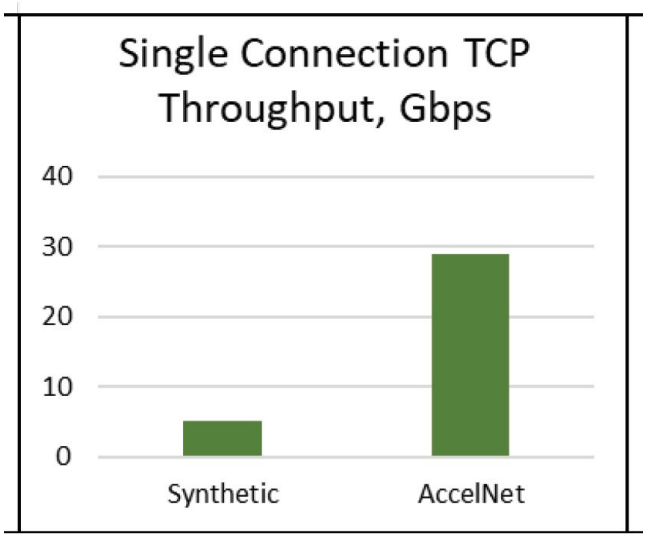
- Sent 1 Million, 4 bytes long pings sequentially, and measured the response time.



BOSTON UNIVERSITY

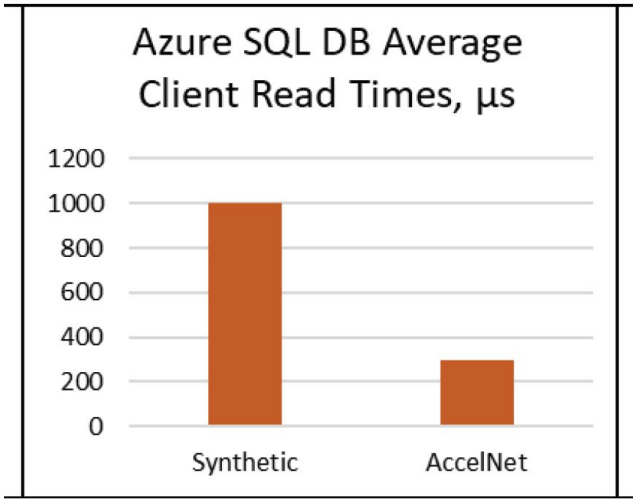
Performance - Single Connection TCP Throughput

- Consistently measured ~31 Gbps on a single connection with 0% CPU utilization.



Performance - SQL DB Client Read Times

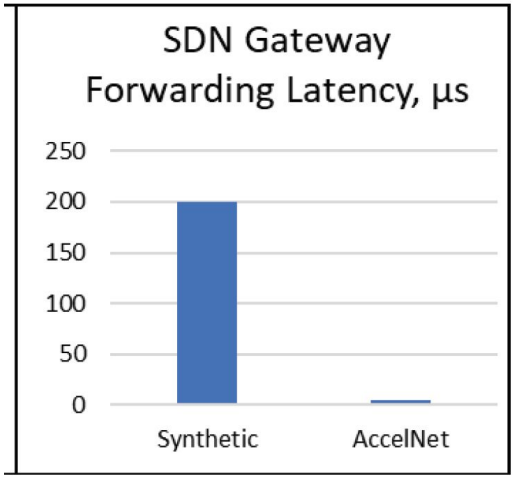
- Average Read times dropped from ~1ms to around 300us.
- Tasks depending on ‘burst performance’ ran 2x faster !



BOSTON UNIVERSITY

Performance - SDN Gateway forwarding latency

- SDN gateway forwarding latency minimized.
- Line rate forwarding now possible !



BOSTON UNIVERSITY

Performance

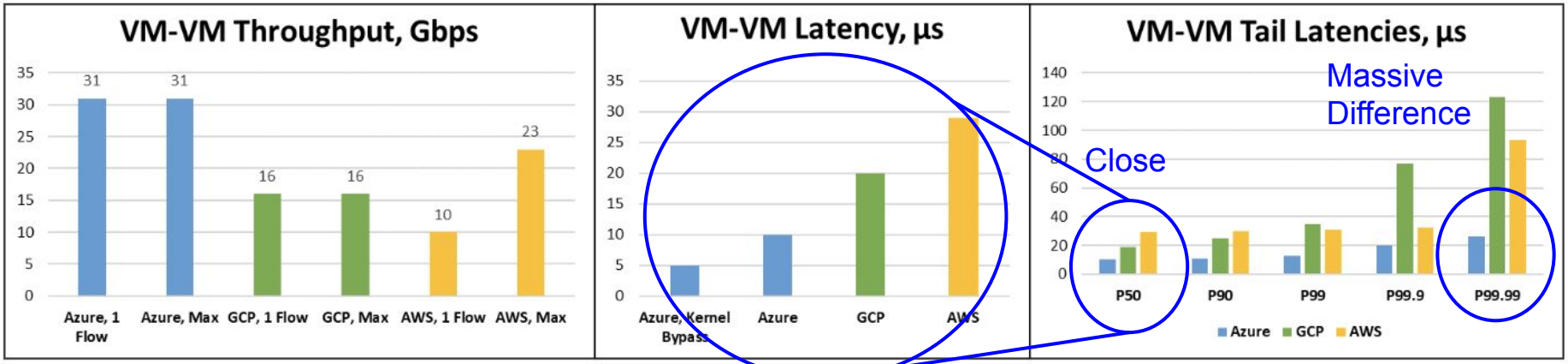


Figure 6: Performance of AccelNet VM-VM latencies vs. Amazon AWS Enhanced Networking and Google GCP Andromeda on Intel Skylake generation hardware.

BOSTON UNIVERSITY

Serviceability

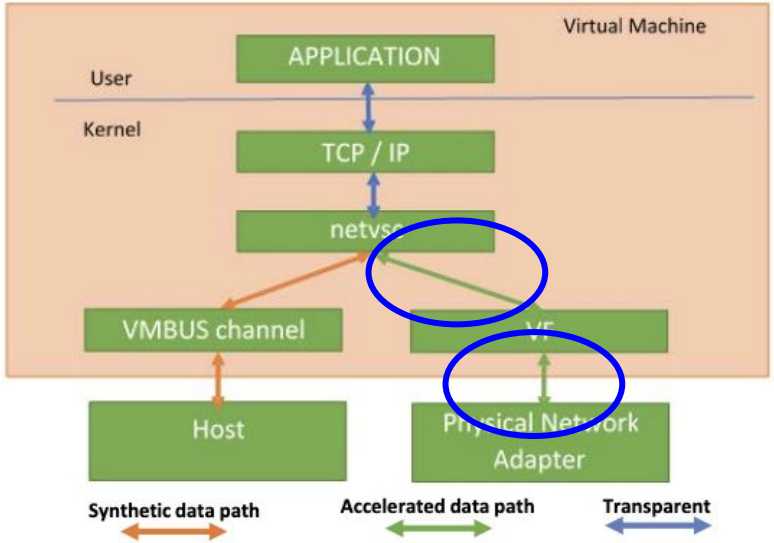


Figure 7: Transparent bonding between an SR-IOV in face and a synthetic interface

BOSTON UNIVERSITY

Monitoring and Diagnostics

Monitoring:

- 500+ Metrics per Host
  - Alerts/Actions
  - Ex: Packet Counting, unusual traffic patterns, how packets were parsed, latency from timestamps, ...
- Constantly tweaking thresholds and actions with more data

Diagnostic:

- Programmable packet capture and tracing services at every interface.
- Hardware timestamps

Example: GFT

Analysis and Conclusion

- Accomplishments
- SDN Evolution over time.
- FPGAs in the Datacenter
- Some key points to note.
- Related Work
- Future Work

Accomplishments

1. **Stopped burning CPU cores** for AccelNet network processing
2. **Upgradeable SDN policies in VFP** agnostic of hardware
3. FPGA results in **latency of <1us** and **line-rate processing**
   - Line-rate achieved on a single connection
4. **GFT** on FPGAs **updated successfully**.
5. Changes rolled out to **heterogeneous hardware**
6. Expected **scalability for 100Gb+**
7. Production **servicing** for FPGA image and driver without negative impact

SDN Evolution on SmartNIC

- Unanticipated feature additions are possible
  - Stateful tunneling and state tracking
  - Extend TCP state machine tracking
  - Packet forwarding/duplication actions
- New actions, ex: virtualized RDMA support
- New functions into hardware for faster connection setup
- Line-rate datapath diagnostics and packet tracing

Some Key Points to Note:

- **Design with serviceability in mind -- right from the start !**
- **Design Higher** level layers to be **agnostic** to **lower** level layers.
- Decreased latency variance leads to **diminishing transient load issues.**
  - Not possible with CPU alone, Hardware offloading needed.
  - Hence, other cloud providers might need to do this as well.
- **DRAM is the most frequently failing** part in the cloud, while other components are reasonably reliable.

Agile Methodology

- Hardware development, e.g. for ASICs, has traditionally been **different** from software development.
- Typically, a ‘waterfall’ methodology is used.
  - ‘All in one go’
- With hardware software co-design, an **Agile approach is needed**.
  - **Hardware Devs should be in the same team as Software Devs**.
  - **Knowledge Sharing** and **Collaboration** is fundamental.
  - Do **live**, real world **testing**, collect **data**, and **integrate** learnings into subsequent versions.

Commentary on FPGAs in the Datacenter

Challenges:

- Higher Dev. Effort
- Not (initially) Cloud-ready
- HLS not as effective
  - HDL very effective given hardware engineers
- Hard to recommend for small-scale clouds

Benefits:

- At Azure scale Dev. cost is worthwhile investment
  - Better performance than CPU
  - Better efficiency than CPU
  - Better flexibility than ASIC

Conclusions

- Microsoft Azure FPGA SmartNIC
  - SDN policy can be offloaded to hardware for great performance gains.
  - Low and predictable latencies
  - Higher throughput on single connections
- Broader Effects/Concepts
  - The cloud previously made their custom software...
  - ....now they are making custom hardware only for its own benefit too
- Custom hardware
  - What works for Microsoft may not be ideal for other providers and almost certainly will not be used by other enterprises
  - The cloud providers have so much power and importance now we will continue to see them define hardware that serves their needs alone

Thank you

...any questions?

BOSTON UNIVERSITY