NS 2018 Paper Reading

Péter Garamvölgyi 2018280070

Outline

- **GPU-based packet classification [1]**
  - packet classification overview
  - problem statement
  - GPU programming
  - bloom filters
  - GSwitch
  - results & evaluation
- safe network updates [3]
  - problem statement
  - inconsistent update example
  - update mechanisms
  - mathematical model
  - results & evaluation

packet classification - overview

given a rule-/filter-set...

... and a packet (usually characterized by the standard 5-tuple) ...

... decide which rules match ...

... and execute the associated actions.

packet classification - example filter set

Filter				*Action*	
SA	*DA*	*Prot*	*DP*	*FlowID*	*PT*
11010010	*	TCP	[3:15]	0	3
10011100	*	*	[1:1]	1	5
101101*	001110*	*	[0:15]	2	8†
10011100	01101010	UDP	[5:5]	3	2
*	*	ICMP	[0:15]	4	9†
100111*	011010*	*	[3:15]	5	6†
10010011	*	TCP	[3:15]	6	3
*	*	UDP	[3:15]	7	9†
11101100	01111010	*	[0:15]	8	2
111010*	01011000	UDP	[6:6]	9	2
100110*	11011000	UDP	[0:15]	10	2
010110*	11011000	UDP	[0:15]	11	2
01110010	*	TCP	[3:15]	12	4†
10011100	01101010	TCP	[0:1]	13	3
01110010	*	*	[3:3]	14	3
100111*	011010*	UDP	[1:1]	15	4

source: [2]

problem statement

- virtualization and SDN made software switches popular again
- software-based packet classification has unique challenges
  - large, dynamic rule tables
  - multidimensional tuples
- packet classification has become the bottleneck

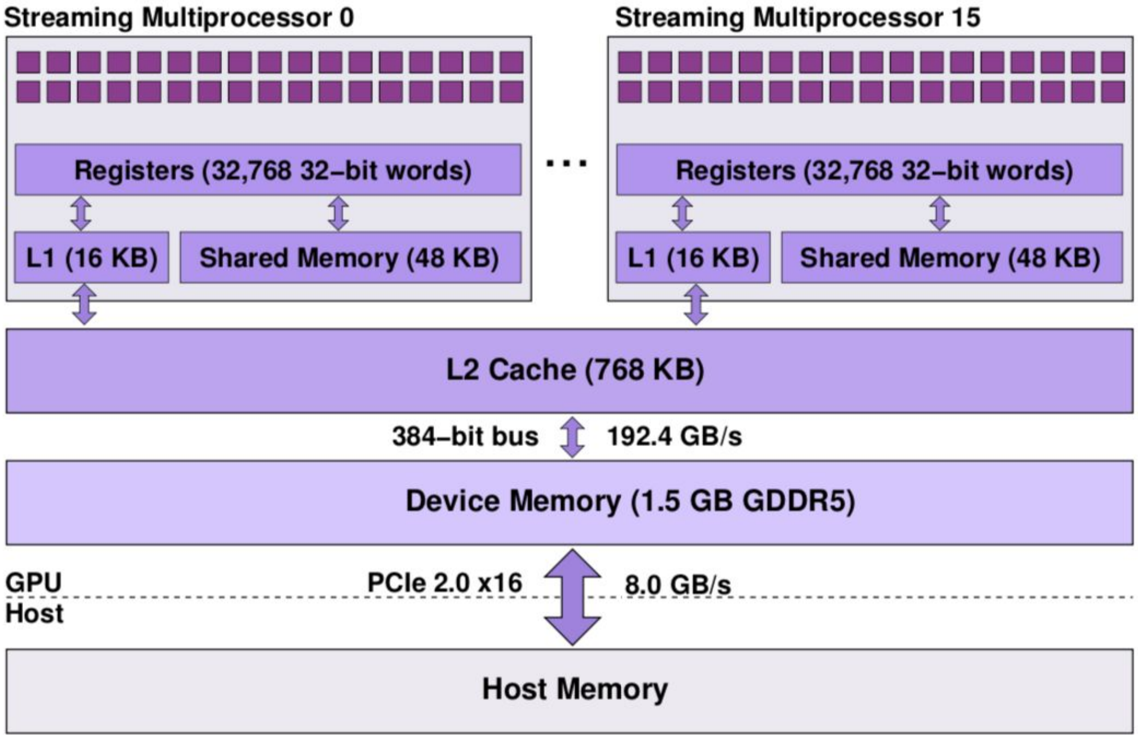
paper contributions

1. GPU-accelerated packet classification
2. bloom search algorithm
3. GSwitch

GPGPU - General Purpose GPU programming

- OpenCL, CUDA
- each thread executes the same program (*kernel*)
- utilization and performance relies on multiple factors
  - synchronization
  - memory access patterns (coalesced reads)
  - caching (on-chip SRAM, shared memory)
  - pipelining (copy & execution)
  - work allocation

GPGPU - GPU architecture



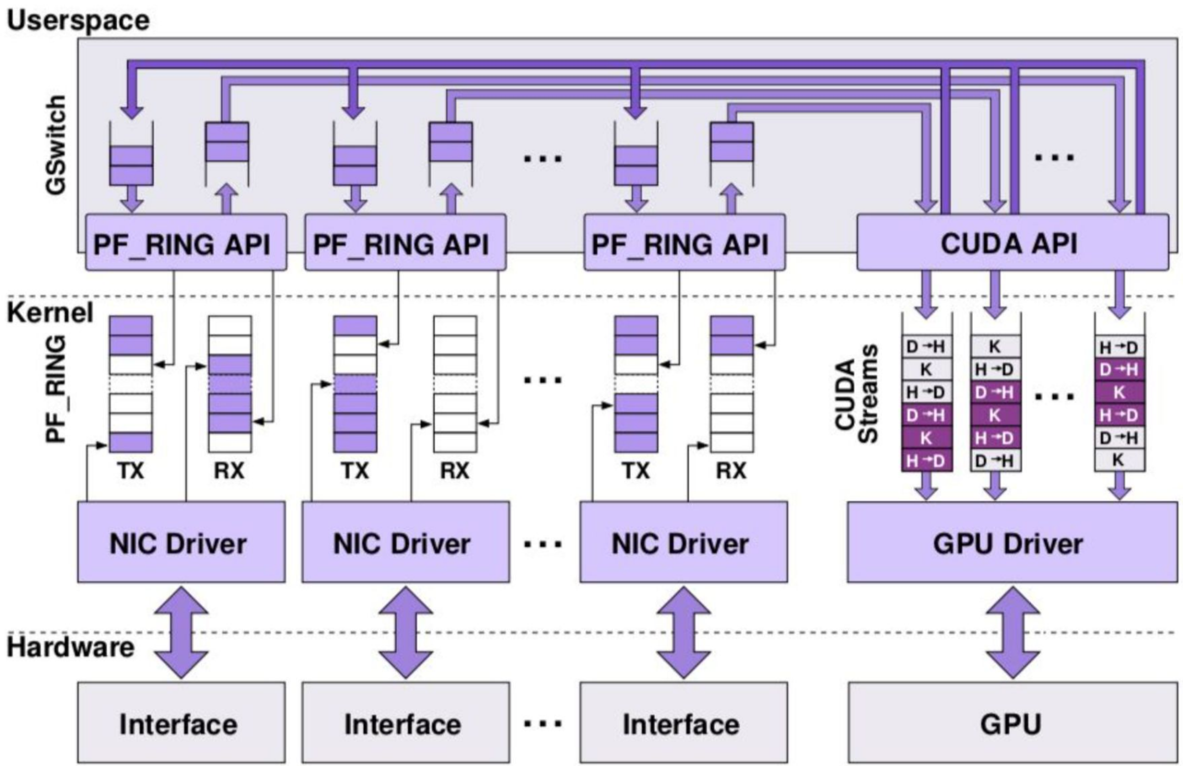
GPGPU - implemented algorithms

- linear search
  - go through all rules
- tuple search
  - divide rules into classes defined by a mask
  - lookup in each class hash table
- bloom search

bloom filters

- probabilistic data structure
- quickly (constant time) tell if element is not in set
- might get false positives, rate is tunable
- bloom search
  - quickly decide if need to check class based on filter

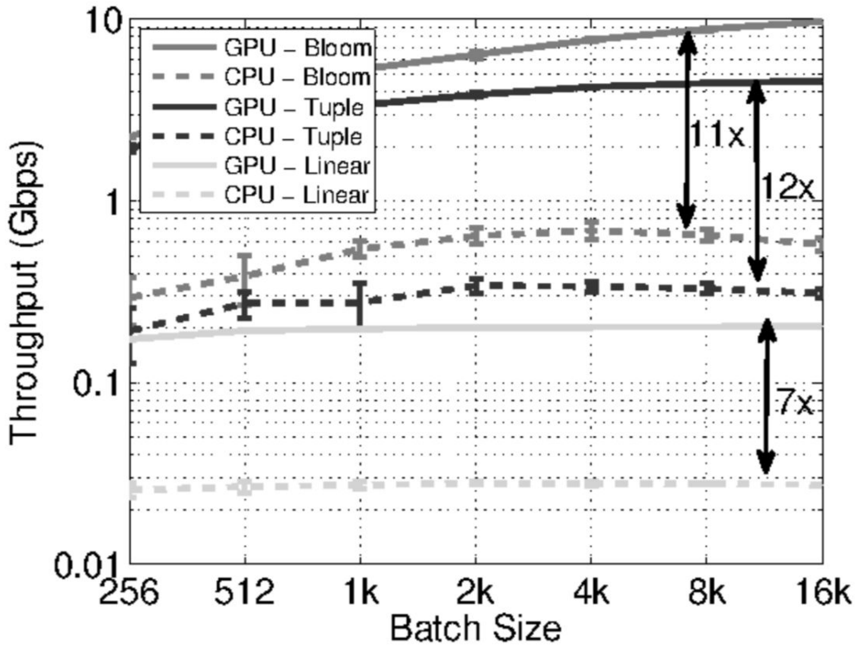
GSwitch

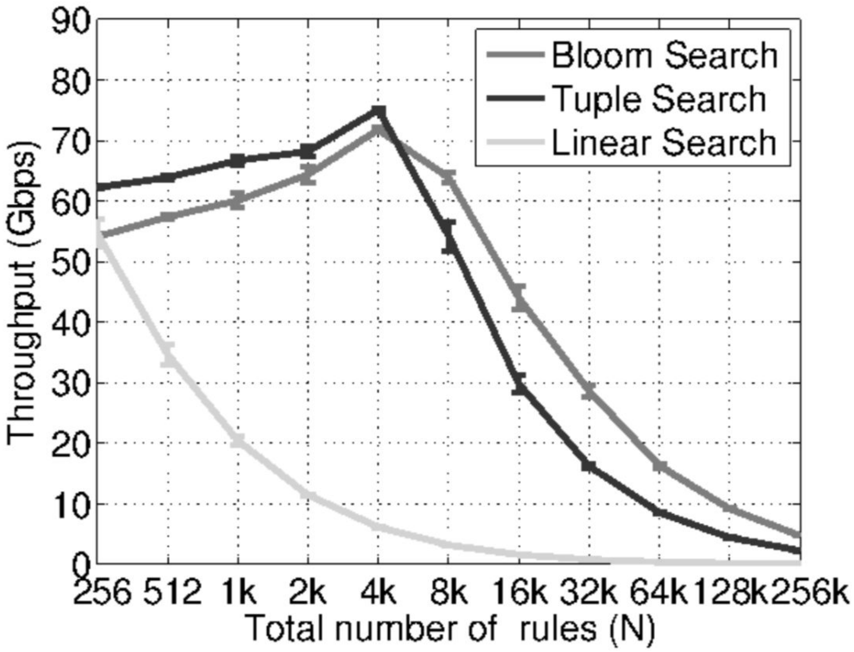

Userspace
GSwitch
PF_RING API
PF_RING API
PF_RING API
CUDA API
Kernel
PF_RING
TX
RX
TX
RX
TX
RX
CUDA Streams
D→H
K
H→D
NIC Driver
NIC Driver
NIC Driver
GPU Driver
Hardware
Interface
Interface
Interface
GPU

results and evaluation

- single CPU, single GPU of comparable price
- GPU micro-benchmarks to tune parameters
- speedup: 7x (linear search), 11x (tuple search), 12x (bloom search)
- “64-byte frames at 30 Gbps and a maximum per-packet latency of 500 µs”.
- shift performance bottleneck from packet classification back to packet I/O

results and evaluation





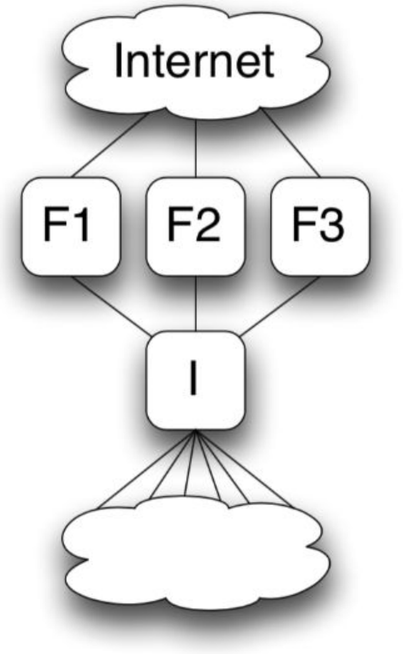
Outline

- GPU-based packet classification [1]
  - packet classification overview
  - problem statement
  - GPU programming
  - bloom filters
  - GSwitch
  - results & evaluation
- **safe network updates [3]**
  - problem statement
  - inconsistent update example
  - update mechanisms
  - mathematical model
  - results & evaluation

problem statement

- computer networks are complex
- updates involve changing configurations on several nodes
- updates often introduce incorrect transient states
- instability, outages, performance disruptions, security vulnerabilities

inconsistent updates - example



configuration 1		
	type	**action**
I	U, G	forward F1
	S	Forward F2
	F	Forward F3
F1	SSH	deny
	*	allow
F2	*	allow
F3	*	allow

configuration 2		
	type	**action**
I	U	forward F1
	G	Forward F2
	S, F	Forward F3
F1	SSH	deny
	*	allow
F2	SSH	deny
	*	allow
F3	*	allow

source: [3]

inconsistent update 1.

configuration 1		
	type	action
I	U, G	forward F1
	S	Forward F2
	F	Forward F3
F1	SSH	deny
	*	allow
F2	*	allow
F3	*	allow

temp 1		
	type	action
I	U, G	forward F1
	S	Forward F2
	F	Forward F3
F1	SSH	deny
	*	allow
F2	**SSH**	**deny**
	*****	**allow**
F3	*	allow

configuration 2		
	type	action
I	U	forward F1
	G	Forward F2
	S, F	Forward F3
F1	SSH	deny
	*	allow
F2	SSH	deny
	*	allow
F3	*	allow

source: [3]

inconsistent update 2.

configuration 1		
	type	**action**
I	U, G	forward F1
	S	Forward F2
	F	Forward F3
F1	SSH	deny
	*	allow
F2	*	allow
F3	*	allow

temp2		
	type	**action**
I	**U**	forward F1
	G	Forward F2
	S, F	Forward F3
F1	SSH	deny
	*	allow
F2	*	allow
F3	*	allow

configuration 2		
	type	**action**
I	U	forward F1
	G	Forward F2
	S, F	Forward F3
F1	SSH	deny
	*	allow
F2	SSH	deny
	*	allow
F3	*	allow

source: [3]

correct update

configuration 1		
	type	**action**
I	U, G S F	forward F1 Forward F2 Forward F3
F1	SSH *	deny allow
F2	*	allow
F3	*	allow

temp1		
	type	**action**
I	U, G **-** **S,** F	forward F1 Forward F2 Forward F3
F1	SSH *	deny allow
F2	*	allow
F3	*	allow

temp2		
	type	**action**
I	U, G - S, F	forward F1 Forward F2 Forward F3
F1	SSH *	deny allow
F2	**SSH** *****	**deny** **allow**
F3	*	allow

configuration 2		
	type	**action**
I	U G S, F	forward F1 Forward F2 Forward F3
F1	SSH *	deny allow
F2	SSH *	deny allow
F3	*	allow

source: [3]

goal: consistent, safe network updates

- network abstraction
  - developers should not need to design step-by-step network update
  - offer high-level primitives and let the system make sure it is correct

- consistency levels
  - per-packet consistency (vs. atomic)
  - per-flow consistency (e.g. HTTP load balancing)

update mechanisms

- **2-phase update** (per-packet consistent)
    1. pre-install new config on all nodes with version number (*unobservable update*)
    2. unlock new policy on ingress ports by stamping packets (*one-touch update*)
- **switch rules with timeouts** (per-flow consistency)
    1. pre-install new config on all nodes
    2. install new config on ingress switches with low priority
    3. set soft timeout on the old config

mathematical modeling

$$
\begin{array}{lll}
\text{Bit} & b & ::= 0 \mid 1 \\
\text{Packet} & pk & ::= [b_1, \ldots, b_k] \\
\text{Port} & p & ::= 1 \mid \cdots \mid k \mid \mathit{Drop} \mid \mathit{World} \\
\text{Located Pkt} & lp & ::= (p, pk) \\
\text{Trace} & t & ::= [lp_1, \ldots, lp_n] \\
\text{Update} & u & \in \mathit{LocatedPkt} \rightharpoonup \mathit{LocatedPkt}\ \mathit{list} \\
\text{Switch Func.} & S & \in \mathit{LocatedPkt} \to \mathit{LocatedPkt}\ \mathit{list} \\
\text{Topology Func.} & T & \in \mathit{Port} \to \mathit{Port} \\
\text{Port Queue} & Q & \in \mathit{Port} \to (\mathit{Packet} \times \mathit{Trace})\ \mathit{list} \\
\text{Configuration} & C & ::= (S, T) \\
\text{Network State} & N & ::= (Q, C)
\end{array}
$$

results and evaluation

- implemented on top of OpenFlow/Kinetic
  - simple interface: `per_packet_update`, `per_flow_update`
  - automatically implement transition based on topology, config, consistency level
  - store version tag in VLAN field
- Mininet network simulation (fat-tree, small-world, random topologies)
  - routing, multicast test scenarios
  - 20-100% overhead (in terms of rules installed)

References

[1] Varvello, Matteo et al. “Multi-Layer Packet Classification with Graphics Processing Units.” CoNEXT (2014).

[2] Taylor, D. E.. “Survey and taxonomy of packet classification techniques.” ACM Comput. Surv. 37 (2005).

[3] Reitblatt, Mark et al. “Abstractions for network update.” SIGCOMM (2012).

Thank you!