The Design and Implementation of Open vSwitch and Tuple Space Search Related Issues

May 14, 2015

Outline

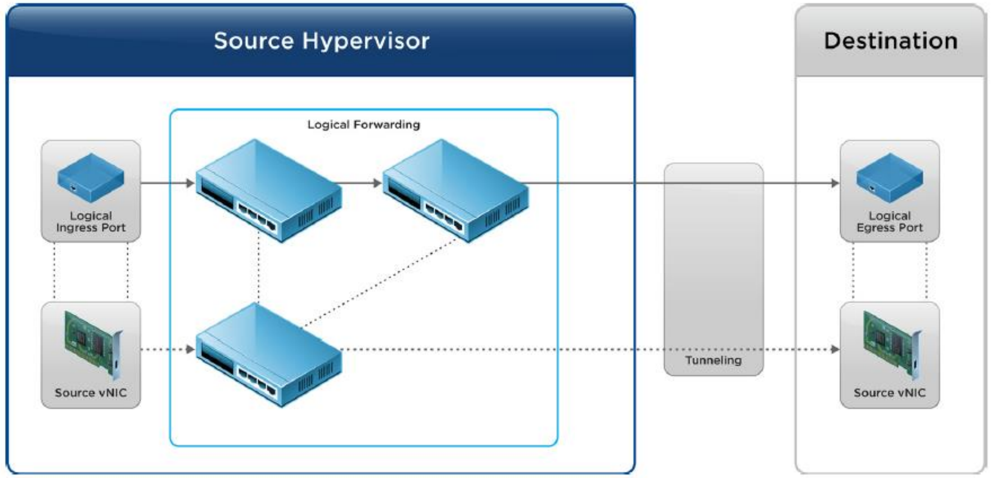
- Open vSwitch
  - Background
  - Design
    - Packet classification
    - Flow caching
    - Cache invalidation
  - Evaluation
- Tuple Space Search
  - Reduce hash lookup times

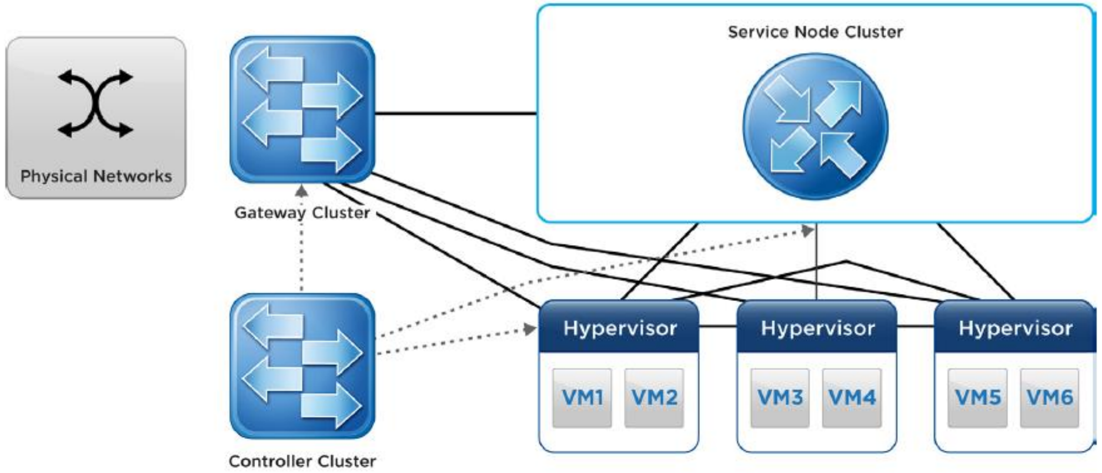
Outline

- Open vSwitch
  - Background
  - Design
    - Packet classification
    - Flow caching
    - Cache invalidation
  - Evaluation
- Tuple Space Search
  - Reduce hash lookup times

Open vSwitch

- Programmability demanded by network virtualization
  - Logical network abstraction, services and tools
  - Flexibility of general purpose processors
- Architectural tension: expressibility v.s. performance
  - Obtaining high performance without sacrificing generality





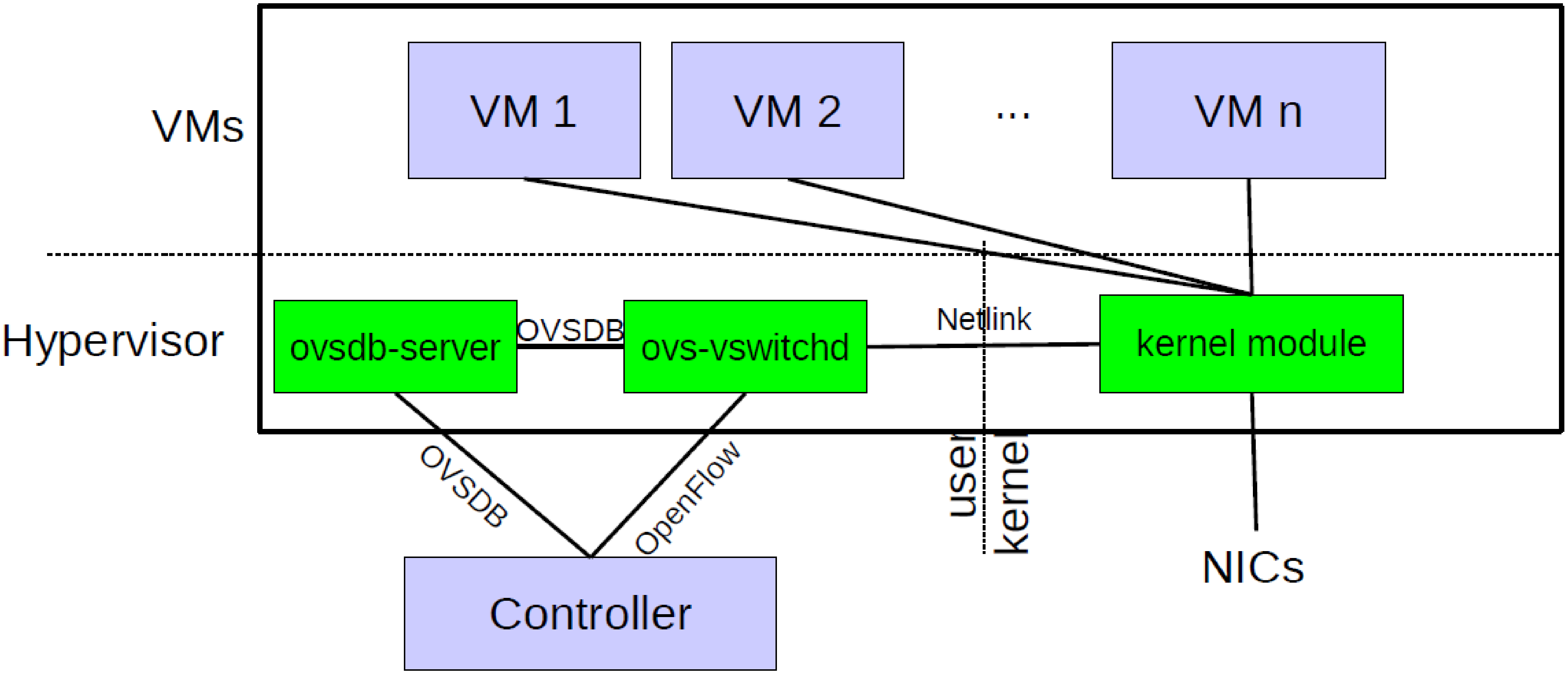
Design constraints

- Resource sharing
  - First maximizing resources available, second worst-case line rate
  - Caching: optimize the common-case over worst-case
- Placement
  - Edge and tunnel: in hypervisor with VMs, thousands of vswitches as peers
  - Update: forwarding state in constant flux
- SDN
  - High classification load: flexible and long packet processing pipelines

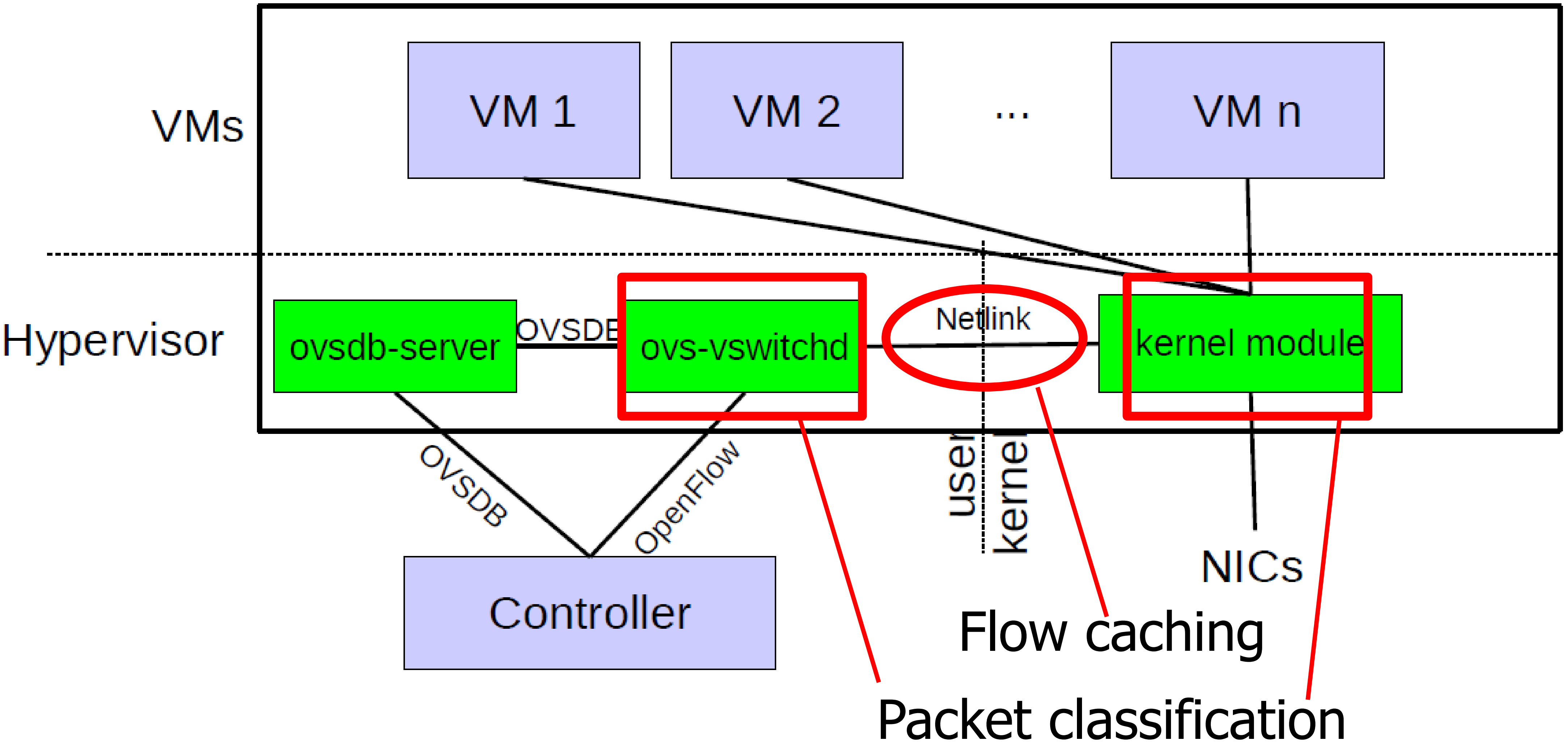
Outline

- Open vSwitch
  - Background
  - Design
    - Packet classification
    - Flow caching
    - Cache invalidation
  - Evaluation
- Tuple Space Search
  - Reduce hash lookup times

Open vSwitch Architecure



Open vSwitch Architecure

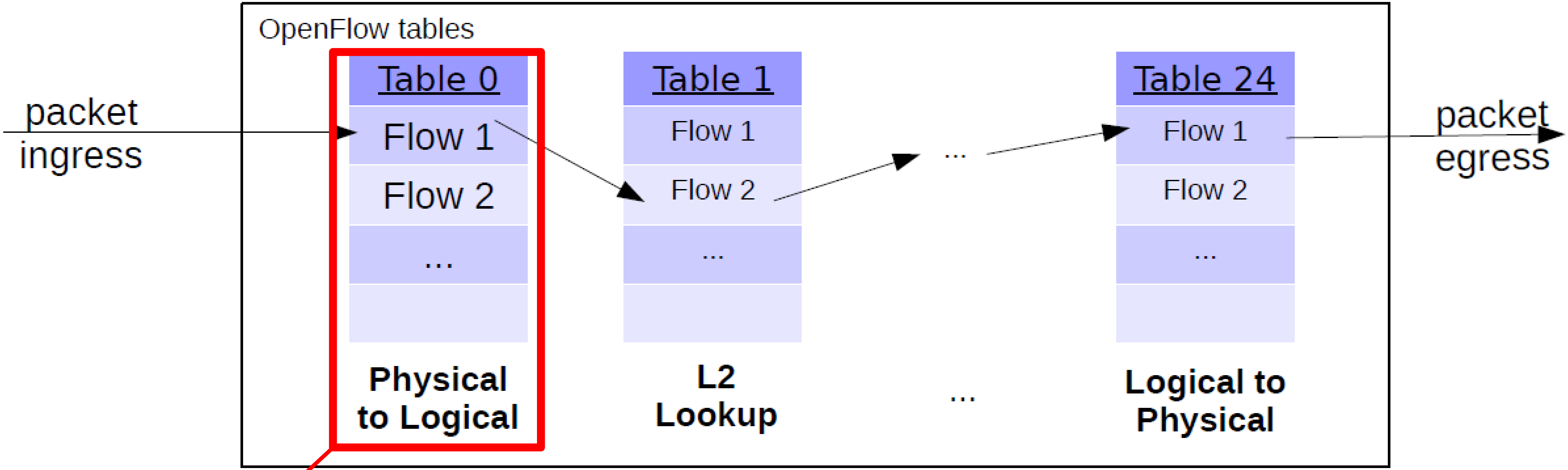


Packet classification

- Classification is expensive on general-purpose CPUs
- Many-field: the context of OpenFlow
- Long pipelines in userspace
  - Cross-producting would significantly increase the flow table sizes
    - n_1 values of field A, $n_2$values of field B, $n_1 \times n_2$ flows in general cases
  - Developer preference to modularize the pipeline design

```
172  cookie=0xfff2fff100000030, duration=1013600.62s, table=0, n_packets=4, n_bytes=256,
     idle_age=65534, hard_age=65534, priority=33000,arp,dl_vlan=47,arp_tpa=192.168.10.1
    actions=learn(table=1,idle_timeout=300,priority=33000,cookie=0xfff2fff100000030,in_p
    ort=72,NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],output:NXM_OF_IN_PORT[]),set_skb_mark:0x30,
    strip_vlan,output:72
173  cookie=0xfff2fff100000040, duration=82649.927s, table=1, n_packets=1974, n_bytes=82
    908, idle_timeout=300, idle_age=1, hard_age=1, priority=33000,dl_dst=e6:1e:ef:68:d7:
    2d actions=output:1
```

Packet classification



OpenFlow Pipeline

- Algorithm: Tuple space search, both kernel and userspace

Tuple Space Search in OVS

- A tuple is a vector of field lengths
  - Hash key: concatenating the bits in each field
  - Hash table: mapping filters of the tuple
- Tuple space Search
  - Multi hash table lookups, the highest priority chosen

Rule	Specification	Tuple
*R*1	(00*,00*)	(2,2)
*R*2	(0**,01*)	(1,2)
*R*3	(1**,0**)	(1,1)
*R*4	(00*,0**)	(2,1)
*R*5	(0**,1**)	(1,1)
*R*6	(***,1**)	(0,1)

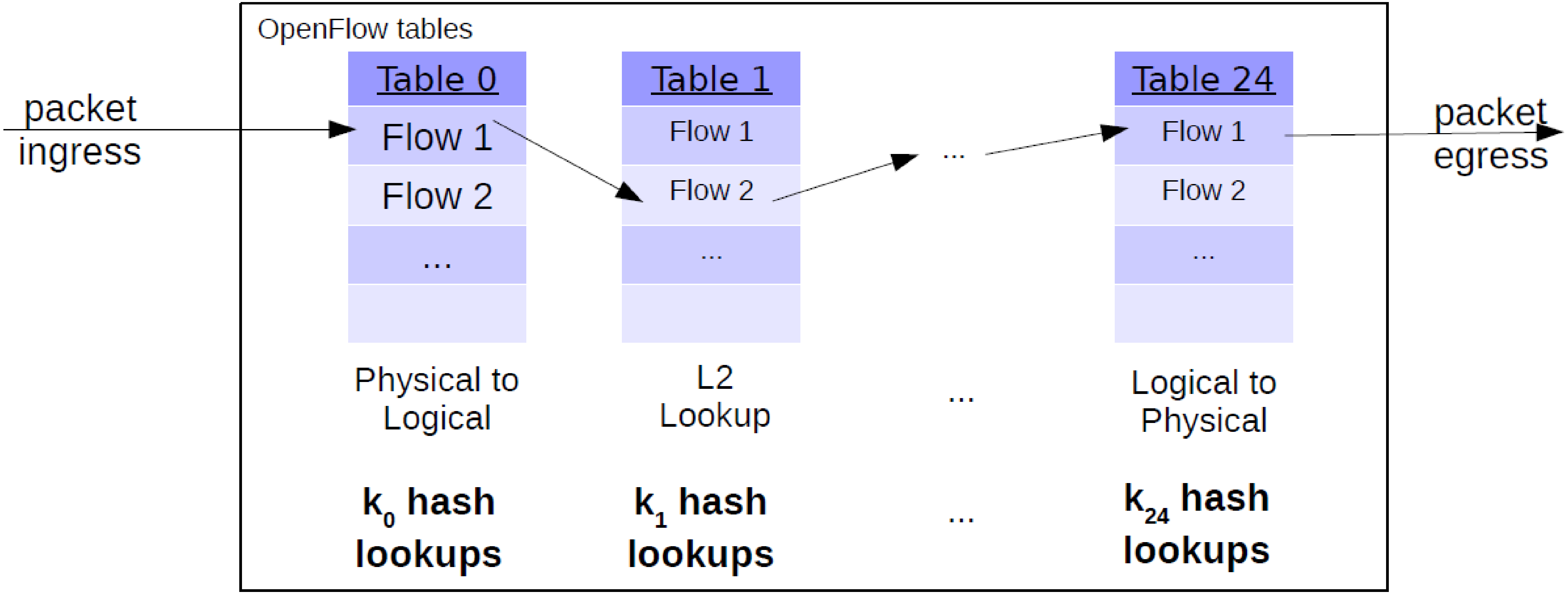
Tuple	Hash table entries
(0,1)	{*R*6}
(1,1)	{*R*3,*R*5}
(1,2)	{*R*2}
(2,1)	{*R*4}
(2,2)	{*R*1}

Tuple Space Search in OVS

- Perform well in practice
- Three attractive properties over decision tree methods
  - Efficient constant-time update (a single hash table operation)
    - Flow change: multiple times per second per hypervisor
  - Generalizing to an arbitrary number of packet header fields
  - Linear memory usage in the number of flows

Decision tree classifiers come with complex tree update logic and in practice developers may favor simplicity over optimality. *--Flow Caching , HotSDN 14*

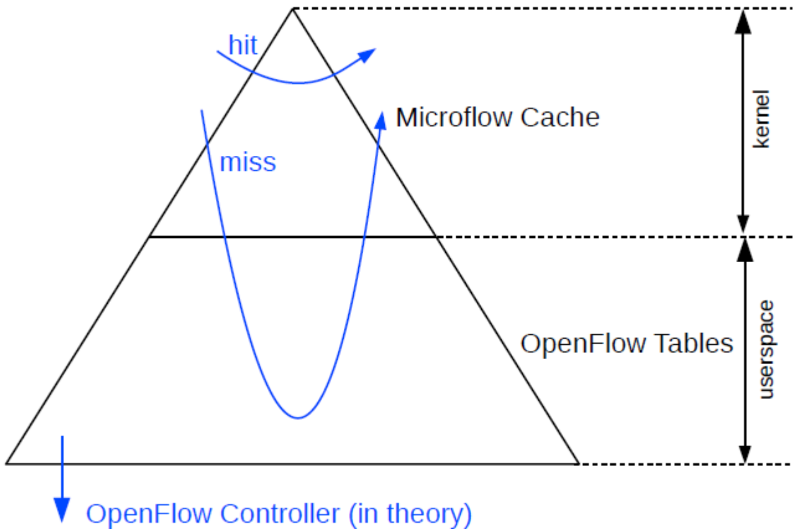
Tuple Space Search in OVS

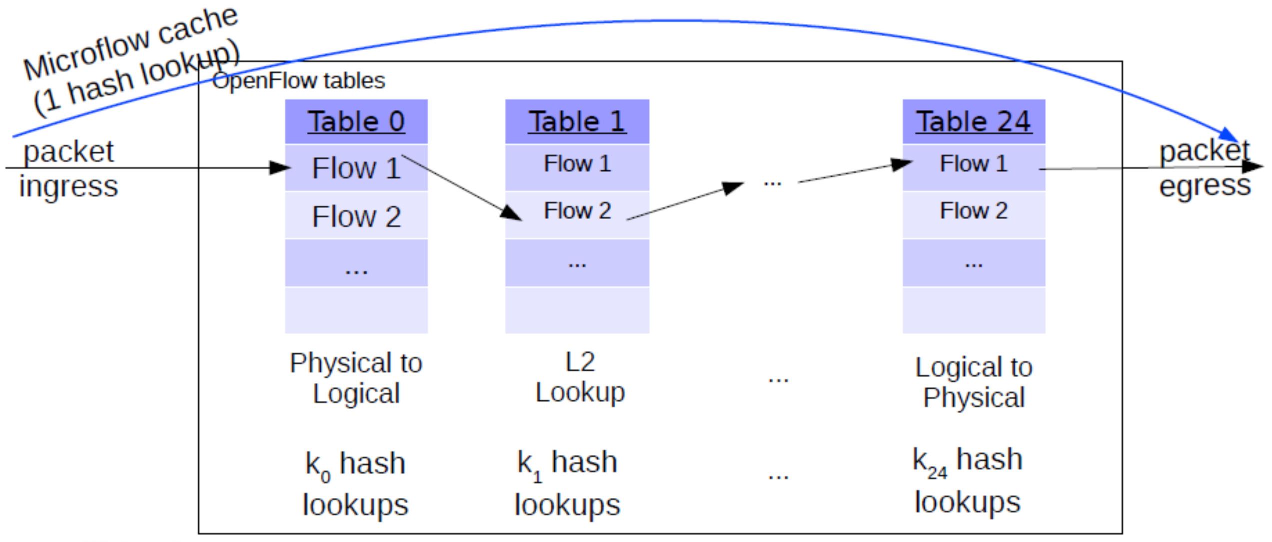


100+ hash lookups per packet for tuple space search?

Flow caching in kernel

- Microflow
  - A single cache entry exact matches all fields
  - Suitable for hash table
  - Low hit rate: short-lived flows, port scan, p2p, network testing





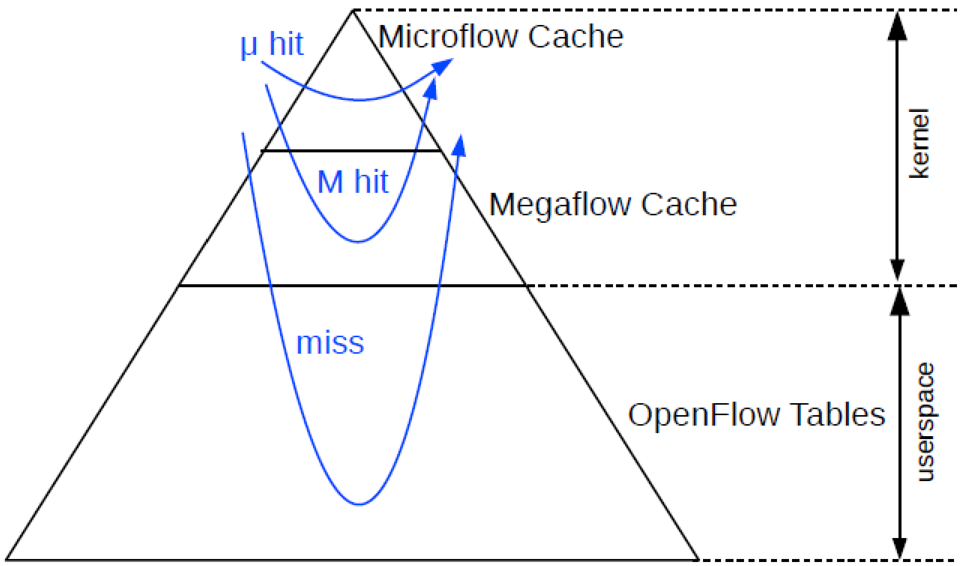
From 100+ hash lookups per packet, to just 1!

Flow caching in kernel

- Microflow
  - A single cache entry exact matches all fields
  - Suitable for hash table
  - Low hit rate: short-lived flows, port scan, p2p, network testing
- Megaflow
  - A single flow table supporting generic matching
  - Multi hash table lookup, still simpler and lighter
    - No priorities: TSS terminates as soon as it finds any match
    - Only one megaflow classifier in kernel

Flow caching in kernel

- Megaflow
  - A single flow table supporting generic matching
  - Multi hash table lookup, still simpler and lighter
  - Retaining the microflow cache as a first-level cache

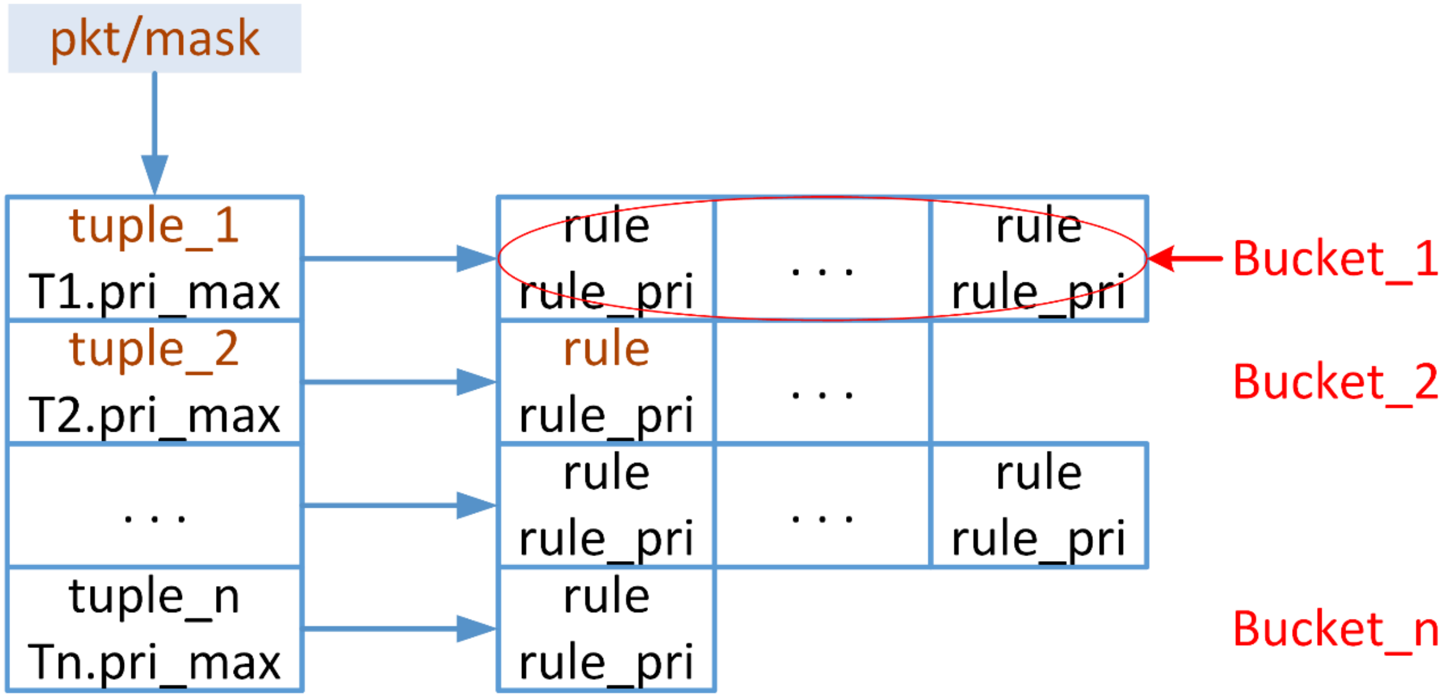


Dual Cache

Caching-aware Packet Classification

- Refine the tuple search algorithm
- Constructing megaflow entries while classifying in userspace
- An online algorithm to generate optimal, least specific megaflows is hard to implement
- Generating increasingly good approximations
  - Tuple priority sorting
  - Staged lookup
  - Prefix tracking

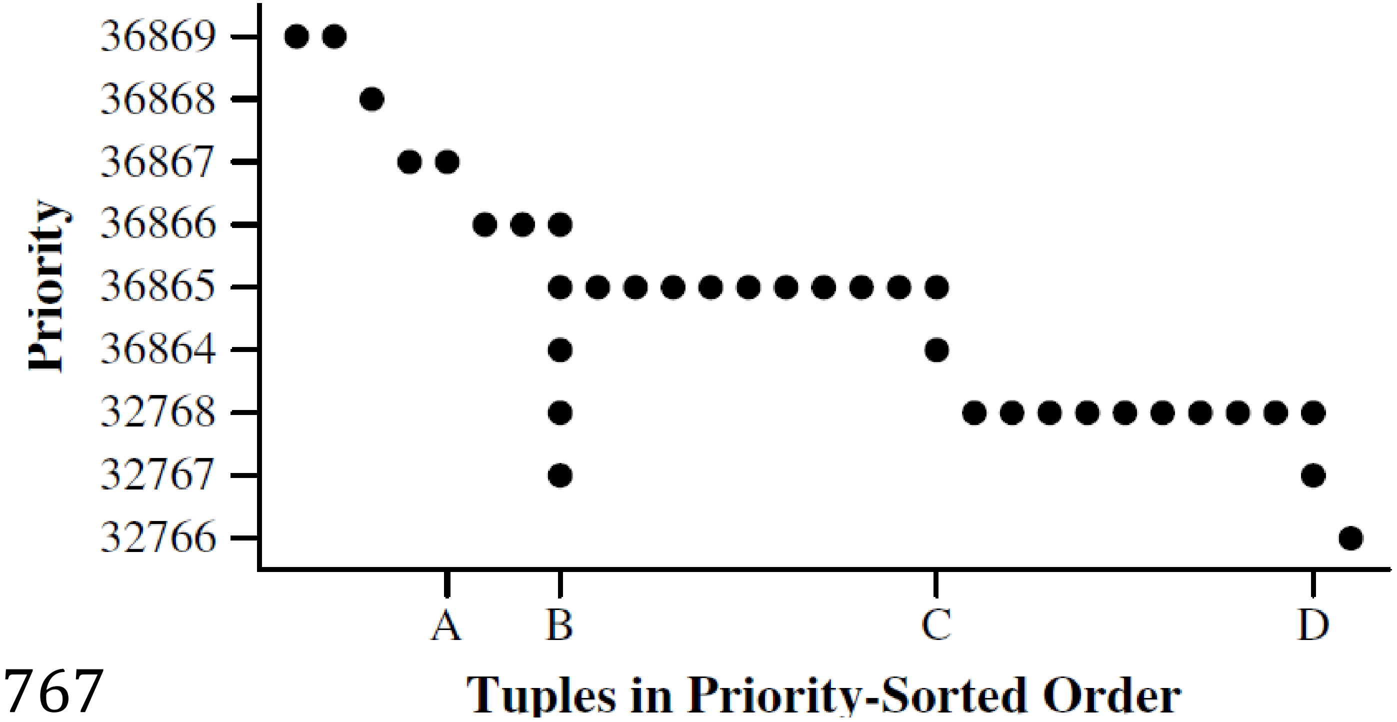
Tuple Priority Sorting



- Search tuples from greatest to least max priority
- Search can terminate as soon as $BestMatch.pri \geq Tuple.pri\_max$

Tuple Priority Sorting

- An example from production deployment
  - Vmware' NVP controller
  - A table of 29 tuples
  - 26 with single priority
  - 2 with two unique priority
  - 1 from 32767 to 36866
  - Worst case: $T.pri\_max > 32767$



Staged Lookup

- Megaflow must match all the bits of fields even search fails
- A field varying from flow to flow
  - Tcp destination port: exactly specified
  - Generated megaflow only matches a single tcp stream as microflow
- Search a tuple on a subset of its fields
  - Terminate when the tuple could not match

Staged Lookup

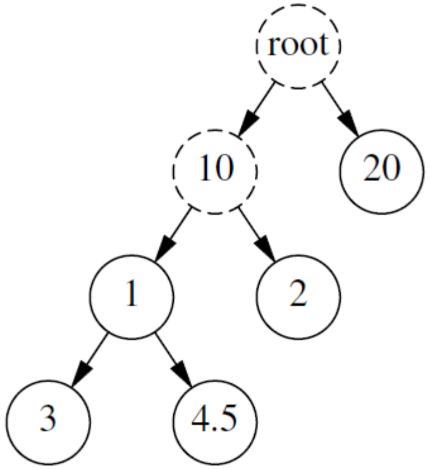
- Trie or per-field hash
  - increase memory access from $O(1)$ to $O(n)$
- Statically divides fields into four groups
  - Metadata (e.g. ingress port), L2, L3 and L4
- Staged Lookup
  - Four hash tables: metadata, metadata L2, metadata L2 L3, all fields
  - Hash computed incrementally from one stage to the next
  - Hash computation a significant cost (profiling shown)

Prefix Tracking

1	**10.1.2.3/32**
2	**10/8**

- ACL with high priority applied to a specific host
  - Forcing all megaflows to match on a full IP address
  - Packet: 10.5.6.7 → safe megaflow: 10.5/14, but 10.5.6.7/32 instead
- Optimization of prefixes using a trie structure
  - LPM lookup before tuple space search

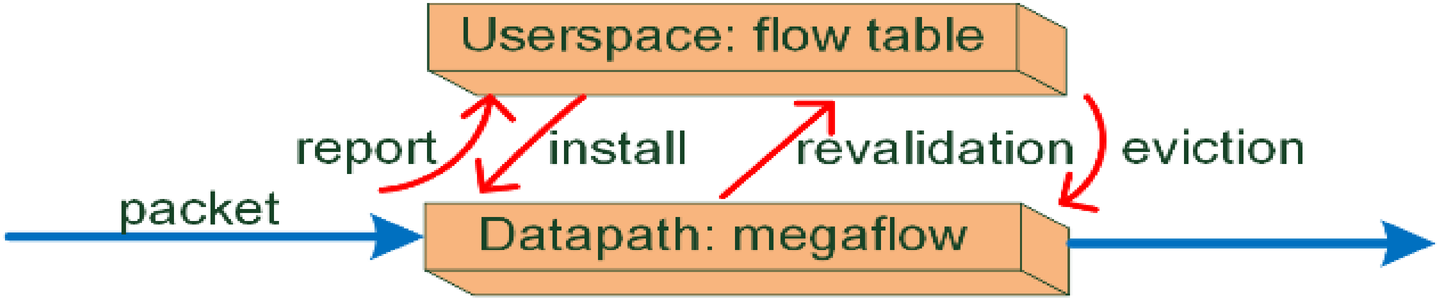
20	/8
10.1	/16
10.2	/16
10.1.3	/24
10.1.4.5	/32



10.1.3.5	10.1.3/24
20.0.5.1	20/8
103.5.1	10.3/16
30.10.5.2	30/5

Cache invalidation

- Precisely identify the megaflow needed to change
  - Online efficient (time and space) analysis remains an open problem (HSA)
- OVS method
  - Examine every datapath flow through the userspace flow tables
  - Multiple dedicated threads for cache revalidation
  - Max cache size about 200K to ensure revalidation time under 1 second



Outline

- Open vSwitch
  - Background
  - Design
    - Packet classification
    - Flow caching
    - Cache invalidation
  - Evaluation
- Tuple Space Search
  - Reduce hash lookup times

Evaluation

- Performance in production
  - 24 hours OVS data polled every 10 minutes over 1000 hypervisor

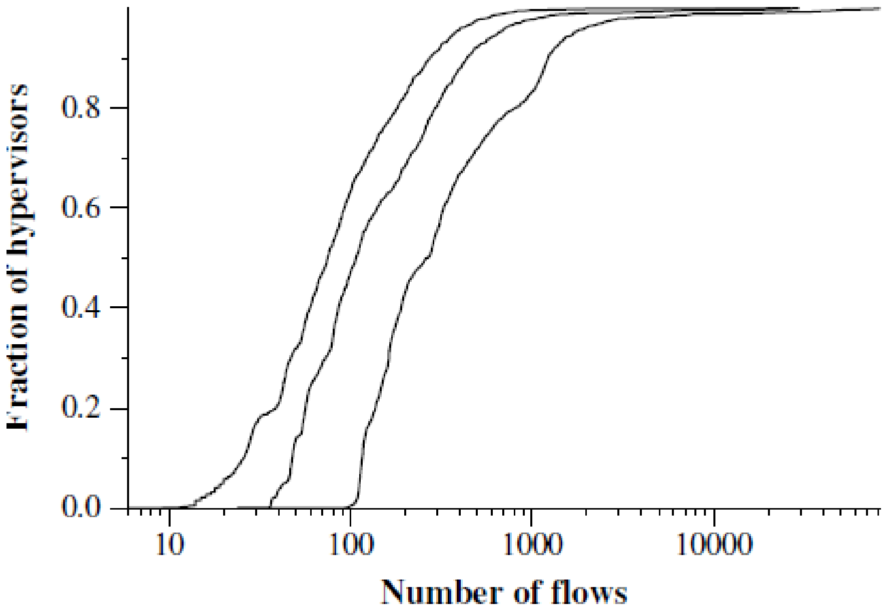


Figure 4: Min/mean/max megaflow flow counts observed.

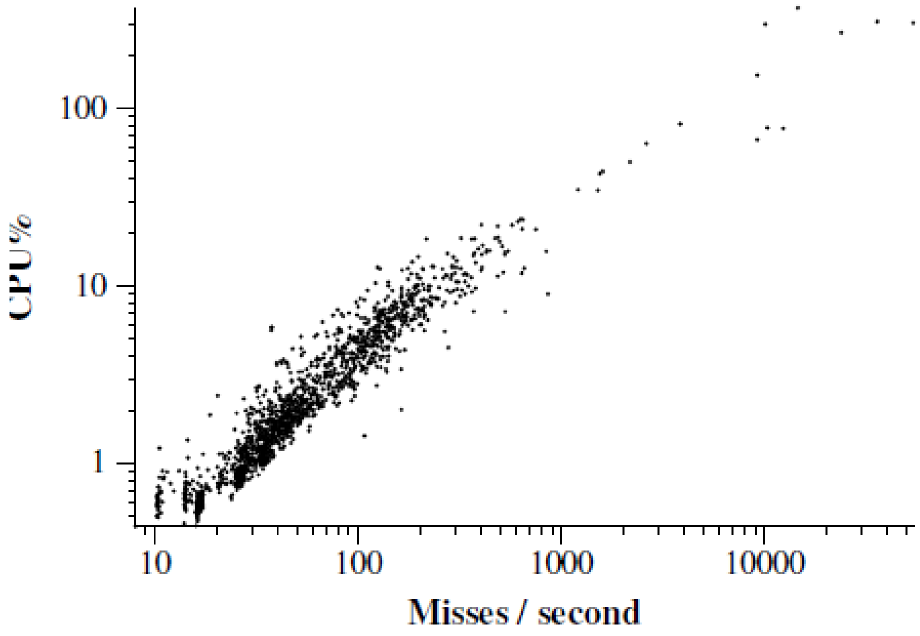


Figure 7: Userspace daemon CPU load as a function of misses/s entering userspace.

Evaluation

- Cache hit rates

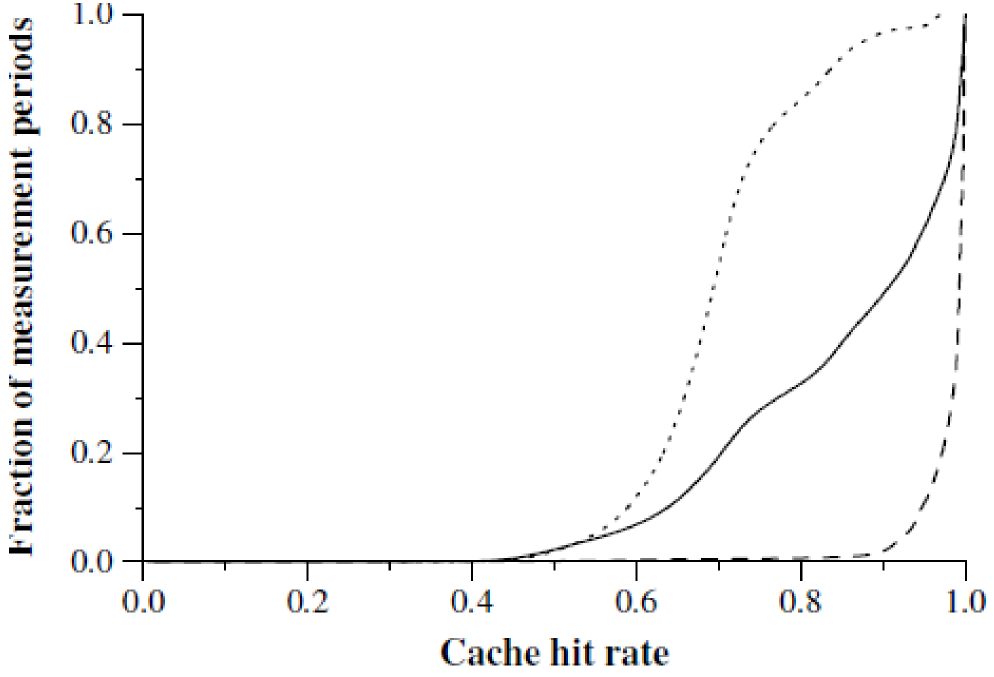


Figure 5: Hit rates during all (solid), busiest (dashed), and slowest (dotted) periods.

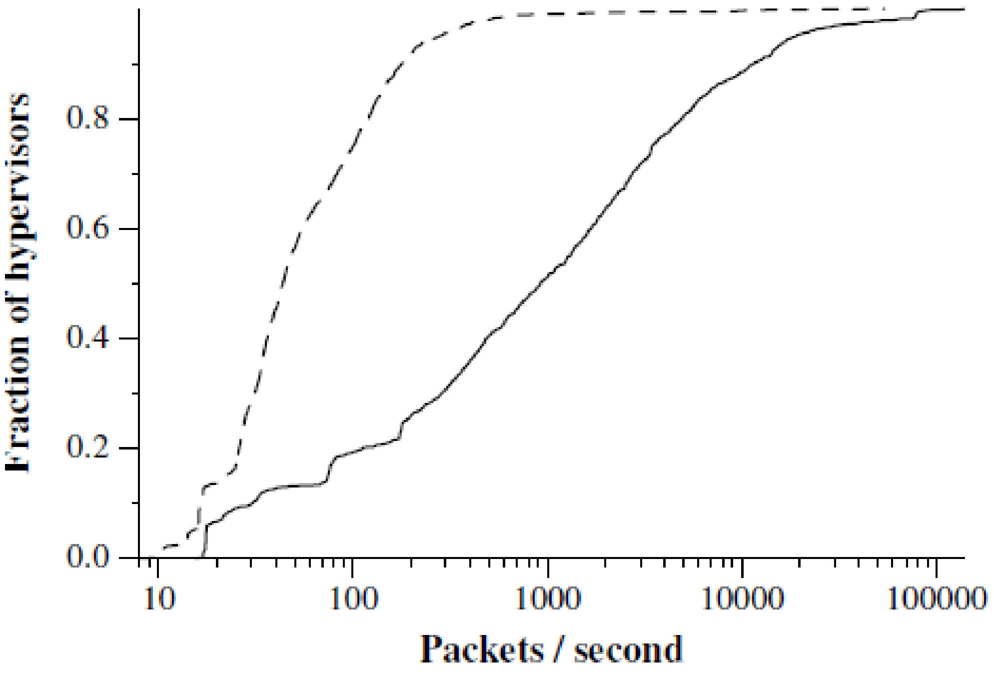


Figure 6: Cache hit (solid) and miss (dashed) packet counts.

Evaluation

- Caching Microbenchmarks
  - Server: two 8-core, 2.0GHz Xeon and two Intel 10Gb NICs
  - Netperf TCP_CRR test: 400 netperf sessions in parallel
  - Flow tables:

arp				(1)
ip	ip_dst=11.1.1.1/16			(2)
tcp	ip_dst=9.1.1.1	tcp_src=10	tcp_dst=10	(3)
ip	ip_dst=9.1.1.1/24			(4)

Evaluation

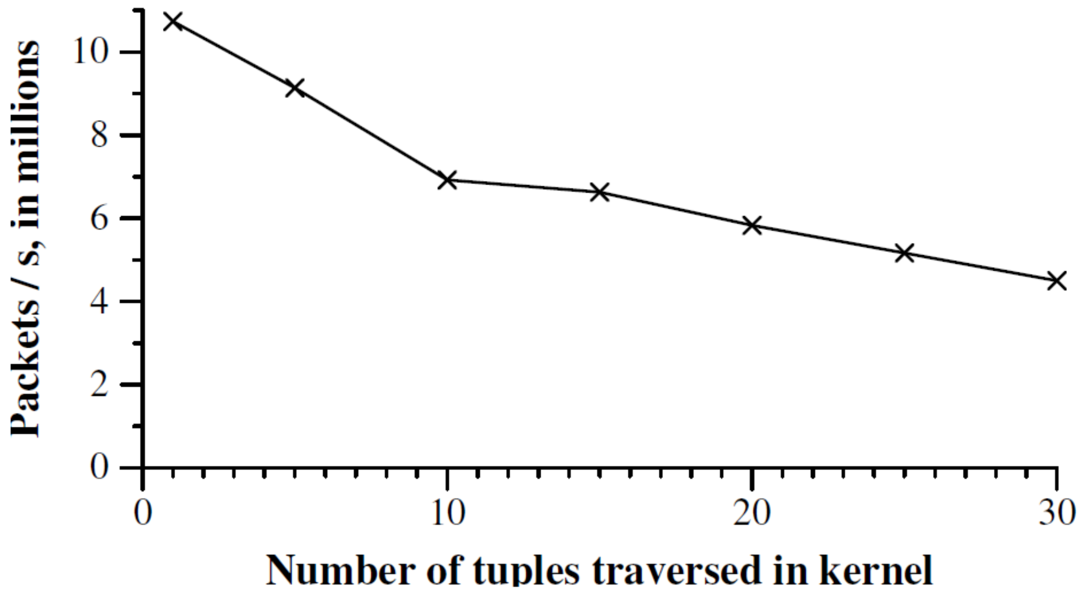
- Cache and optimization benefit

Microflows	Optimizations	ktps	Tuples/pkt	CPU%
Enabled	Enabled	120	1.68	0/ 20
Disabled	Enabled	92	3.21	0/ 18
Enabled	Disabled	56	1.29	38/ 40
Disabled	Disabled	56	2.45	40/ 42

Optimizations	ktps	Flows	Masks	CPU%
Megaflows disabled	37	1,051,884	1	45/ 40
No optimizations	56	905,758	3	37/ 40
Priority sorting only	57	794,124	4	39/ 45
Prefix tracking only	95	13	10	0/ 15
Staged lookup only	115	14	13	0/ 15
All optimizations	117	15	14	0/ 20

Evaluation

- Forwarding performance for long-lived flows
  - 50K entries randomly generated: 6.8M hash lookups/s
  - Microflow enabled: 10.6Mpps
  - Microflow disabled



Ongoing and Future

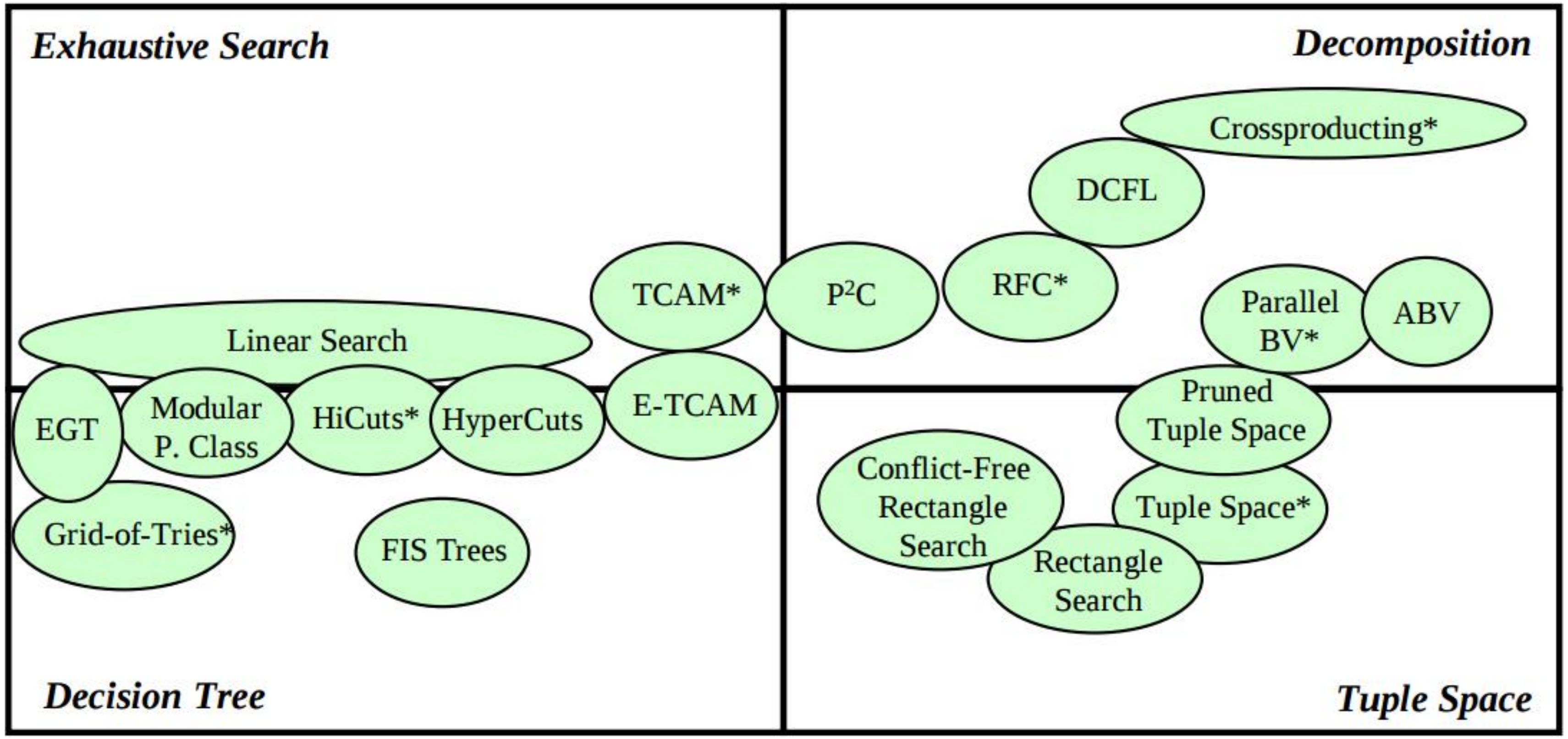
- Stateful packet processing
  - Local L3 deamon processing ARPs: socket communication with overhead
  - IP reassemble, transport connection tracking
  - New OpenFlow action: invoke a kernel module providing state metadata
- Userspace networking
  - Bypass kernel: DPDK, netmap
- Hardware Offloading
  - Enable NICs to accelerate kernel flow classification

Outline

- Open vSwitch
  - Background
  - Design
    - Packet classification
    - Flow caching
    - Cache invalidation
  - Evaluation
- Tuple Space Search
  - Reduce hash lookup times

Tuple Space Search Direction

- Rationale
  - decomposes a classification query into a number of exact match queries



Tuple Space Search Direction

- Rationale
  - decomposes a classification query into a number of exact match queries
- Cons
  - Only prefix supported, range can be encoded but no overlap
  - The number of tuples could be very large (cross-product)
  - Hashing makes the time complexity of searches and updates nondeterministic
- Pros
  - Efficient incremental update
  - Arbitrary number of packet header fields

Tuple Space Search Direction

- Reduce hash lookup times
  - Tuple pruning algorithm
  - Rectangle Search in 2D
  - Rule rewriting
  - Bloom filter
- Increase hash lookup performance
  - General or dedicated methods
- Software switches

Reduce hash lookup times

- Tuple pruning algorithm
  - Observation: no address D has more than 6 matching prefixes
  - $32 \times 32 = 1024 \rightarrow 6 \times 6 = 36$
  - First do independent matches in each dimension
  - Cross-product each dimension to generate tuple and probe the tuples

Rule Database	FW1-100	FW1-1k	FW1-5k	FW1-10k	FW1	IPC1	ACL1
Number of rules	92	971	4653	9311	266	1550	752
Number of tuples	26	42	52	57	36	179	44

Rules from Classbench

(Srinivasan et al, Packet classification using tuple space search, SIGCOMM' 99)

Encoded rule expansion

- Motivation
  - No IP address prefix contains more than 5 nesting prefixes
  - Reduce hash table numbers by encoding rules based on prefix nesting
- Expanding a rule to a designating length combination
  - Expand the length combination of R2 from {3,2,0,2,8} to {3,3,2,2,8}

R_2	000*	10*	*	10*	TCP	act_1

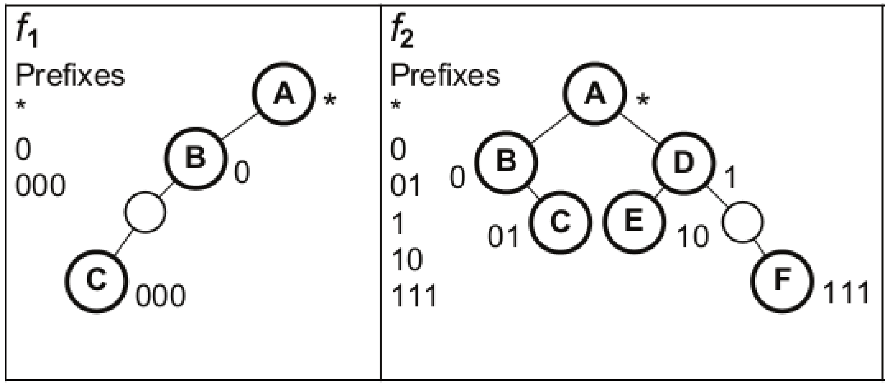
$$\begin{array}{ll}
(000*, 100*, 00*, 10*, \mathrm{TCP}), & (000*, 101*, 00*, 10*, \mathrm{TCP}) \\
(000*, 100*, 01*, 10*, \mathrm{TCP}), & (000*, 101*, 01*, 10*, \mathrm{TCP}) \\
(000*, 100*, 10*, 10*, \mathrm{TCP}), & (000*, 101*, 10*, 10*, \mathrm{TCP}) \\
(000*, 100*, 11*, 10*, \mathrm{TCP}), & (000*, 101*, 11*, 10*, \mathrm{TCP})
\end{array}$$

(Wang, Scalable Packet Classification for Datacenter Networks, JSAC 14)

Encoded rule expansion

- Nesting identify Encoded

Rule	f_1	f_2	f_3	f_4	f_5	Action
R_0	000*	111*	10*	*	UDP	act_0
R_1	000*	111*	01*	10*	UDP	act_0
R_2	000*	10*	*	10*	TCP	act_1
R_3	000*	10*	*	01*	TCP	act_2
R_4	000*	10*	10*	11*	TCP	act_1
R_5	0*	111*	10*	01*	UDP	act_0
R_6	0*	111*	10*	10*	UDP	act_0
R_7	0*	1*	*	*	TCP	act_2
R_8	*	01*	*	*	TCP	act_2
R_9	*	0*	*	01*	UDP	act_0
R_{10}	*	*	*	*	UDP	act_3
R_{11}	*	*	*	*	TCP	act_4



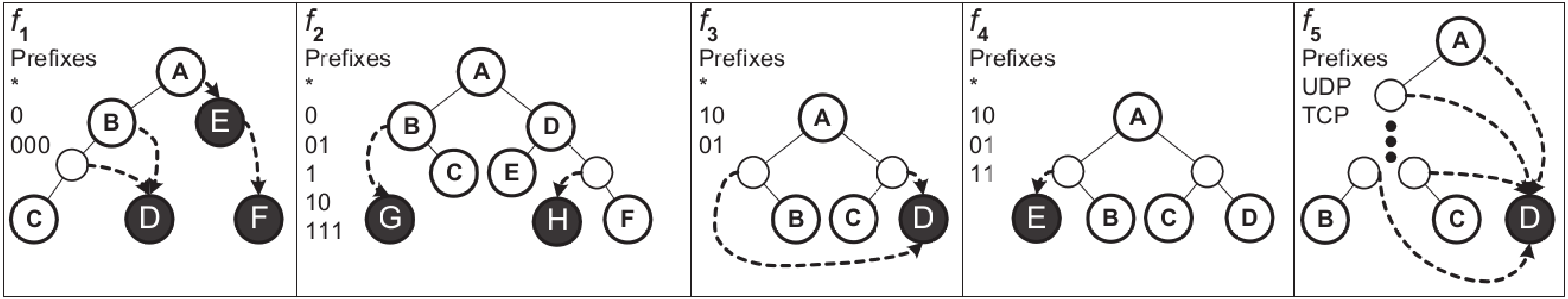
Rule	f_1	f_2	f_3	f_4	f_5	Action
R_0	ABC*	ADF*	AC*	A*	AB	act_0
R_1	ABC*	ADF*	AB*	AC*	AB	act_0
R_2	ABC*	ADE*	A*	AC*	AC	act_1
R_3	ABC*	ADE*	A*	AB*	AC	act_2
R_4	ABC*	ADE*	AC*	AD*	AC	act_1
R_5	AB*	ADF*	AC*	AB*	AB	act_0
R_6	AB*	ADF*	AC*	AC*	AB	act_0
R_7	AB*	AD*	A*	A*	AC	act_2
R_8	A*	ABC*	A*	A*	AC	act_2
R_9	A*	AB*	A*	AB*	AB	act_0
R_{10}	A*	A*	A*	A*	AB	act_3
R_{11}	A*	A*	A*	A*	AC	act_4

Tuple	Length Combination	Rules
$\mathbb{T}_0$	(3, 3, 2, 1, 2)	R_0
$\mathbb{T}_1$	(3, 3, 2, 2, 2)	R_1, R_4
$\mathbb{T}_2$	(3, 3, 1, 2, 2)	R_2, R_3
$\mathbb{T}_3$	(2, 3, 2, 2, 2)	R_5, R_6
$\mathbb{T}_4$	(2, 2, 1, 1, 2)	R_7
$\mathbb{T}_5$	(1, 3, 1, 1, 2)	R_8
$\mathbb{T}_6$	(1, 2, 1, 2, 2)	R_9
$\mathbb{T}_7$	(1, 1, 1, 1, 2)	R_{10}, R_{11}

(Wang, Scalable Packet Classification for Datacenter Networks, JSAC 14)

Encoded rule expansion

- Expanding encoded rules
  - Binary tries with complementary nodes
  - R 2 = (ABC∗,ADE∗,A∗,AC∗,AC) to {3,3,2,2,2}
  - (ABC∗,ADE∗,AB∗,AC∗,AC), (ABC∗,ADE∗,AC∗,AC∗,AC), (ABC∗,ADE∗,AD∗,AC∗,AC)
  - With nesting identify: to {3,3,2,2,2}, rule number 2048 → 507



(Wang, Scalable Packet Classification for Datacenter Networks, JSAC 14)

Encoded rule expansion

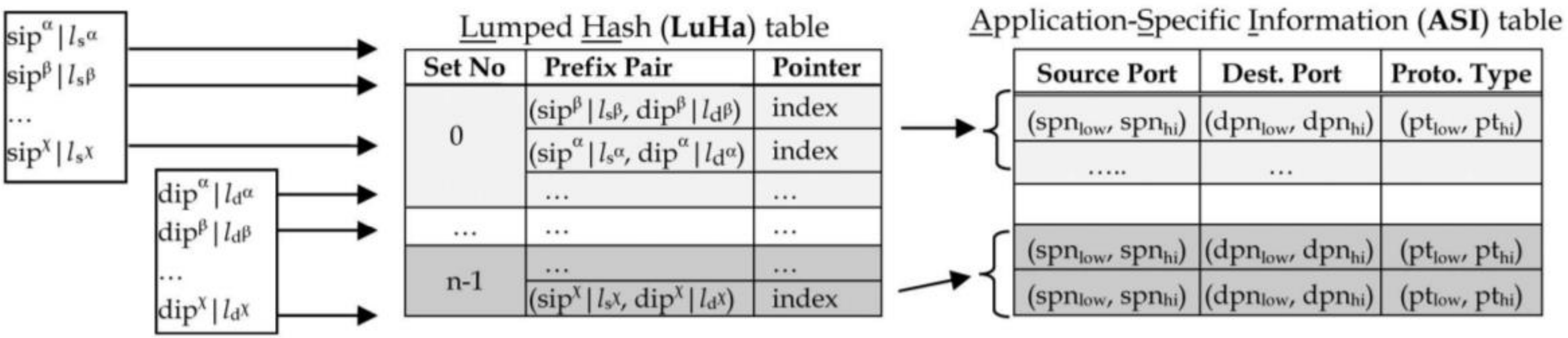
- Minimizes the cost of reducing the number of length combinations
  - The number of distinct length combinations is reduced to a predefined threshold T while minimizing the number of generated rules
  - Optimization: p-median problem
  - From 8 to 4 tuples

Tuple	Rule Specifications: $(f_1, f_2, f_3, f_4, f_5)$	Matching Rules
$\mathbb{T}_1$	(ABC∗, ADF∗, AB∗, AC∗, AB)	R_1
	(ABC∗, ADE∗, AC∗, AD∗, AC)	R_4
	(ABC∗, ADF∗, AC∗, AB∗/AC∗/AD∗/AE∗, AB)	$R_0, (R_5, R_6)$
	(ABC∗/ABD∗, ADF∗, AC∗, AB∗, AB)	R_5
	(ABC∗/ABD∗, ADF∗, AC∗, AC∗, AB)	R_6
	(ABC∗, ADE∗, AB∗/AC∗/AD∗, AC∗, AC)	R_2
	(ABC∗, ADE∗, AB∗/AC∗/AD∗, AB∗, AC)	R_3
$\mathbb{T}_4$	(AB∗, AD∗, A∗, A∗, AC)	R_7
$\mathbb{T}_5$	(A∗, ABC∗, A∗, A∗, AC)	$R_8, (R_{11})$
	(A∗,ABG∗/ABC∗/ADE∗/ADH∗/ADF∗,A∗,A∗,AB)	R_{10}
	(A∗,ABG∗/ABC∗/ADE∗/ADH∗/ADF∗,A∗,A∗,AC)	R_{11}
$\mathbb{T}_6$	(A∗, AB∗, A∗, AB∗, AB)	R_9

(Wang, Scalable Packet Classification for Datacenter Networks, JSAC 14)

Hashing round-down prefixes

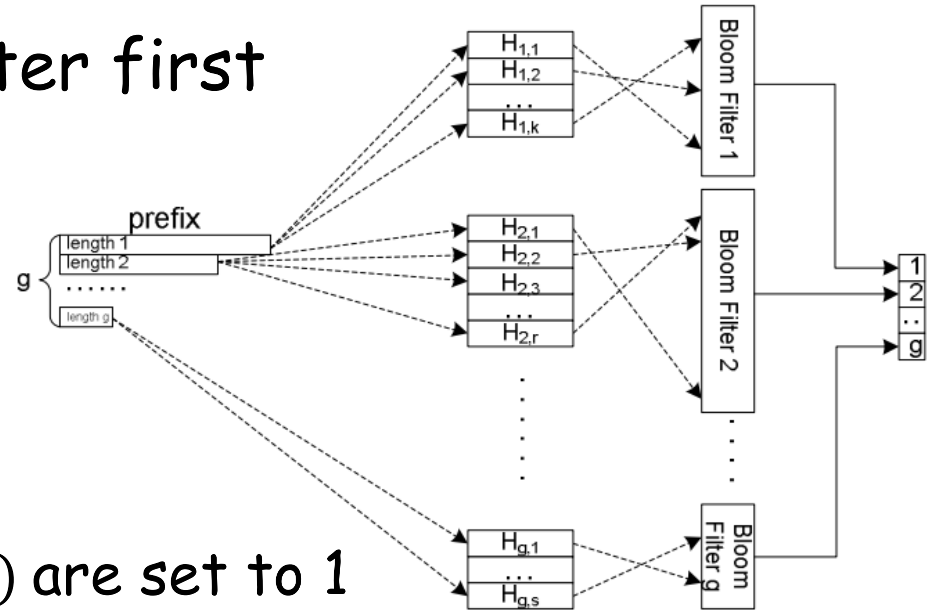
- Rounding down prefixes to a small number of designated prefix lengths
- Collapsing hash units to one lumped hash table



(Pong, HaRP: Rapid Packet Classification via Hashing Round-Down Prefixes, TPDS 11)

Reduce hash lookup times

- Accelerate per table lookup using bloom filter first
- Bloom filter first
  - Space-efficient: on-chip memory
  - A tuple: n rules, m bits array, k hash functions
  - rule r_i, the bits of positions $h_1(r_i), h_2(r_i), .., h_k(r_i)$ are set to 1
  - Match: all hashed positions are 1, false positive
- After match founed, check the conrresponding hash table in off-chip memory



(Varvello et al, Multi-Layer Packet Classification with Graphics Processing Units, CoNEXT' 14)

(Song et al, Ipv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards, INFOCOMM' 09)

Deterministic hash lookup

- Cuckoo hashing
  - Collision makes hash operation nondeterministic
  - n rules, s tables each with d slots, s hash functions
  - Compute $h_1(r_i), h_2(r_i), .., h_s(r_i)$, if one slot empty, insert r_i, otherwise select table q, $h_q(r_i) = h_q(r_j)$ replace r_j with r_i, then reinsert r_j
  - Rehash (new function) when cycle happens
  - Worst case complexity $O(s)$

(Zhou et al, Scalable, high performance ethernet forwarding with cuckooswitch, CoNEXT' 13)
(Varvello et al, Multi-Layer Packet Classification with Graphics Processing Units, CoNEXT' 14)
(Pfaff et al, The Design and Implementation of Open vSwitch, NSDI' 15)

Thanks

May 14, 2015