Fast and Scalable Software Packet Processing for Online Packet Classification

James Daly, Valerio Bruschi, Leonardo Linguaglossa, Salvatore Pontarelli, Dario Rossi, Jerome Tollet, Eric Torng, Andrew Yourtchenko

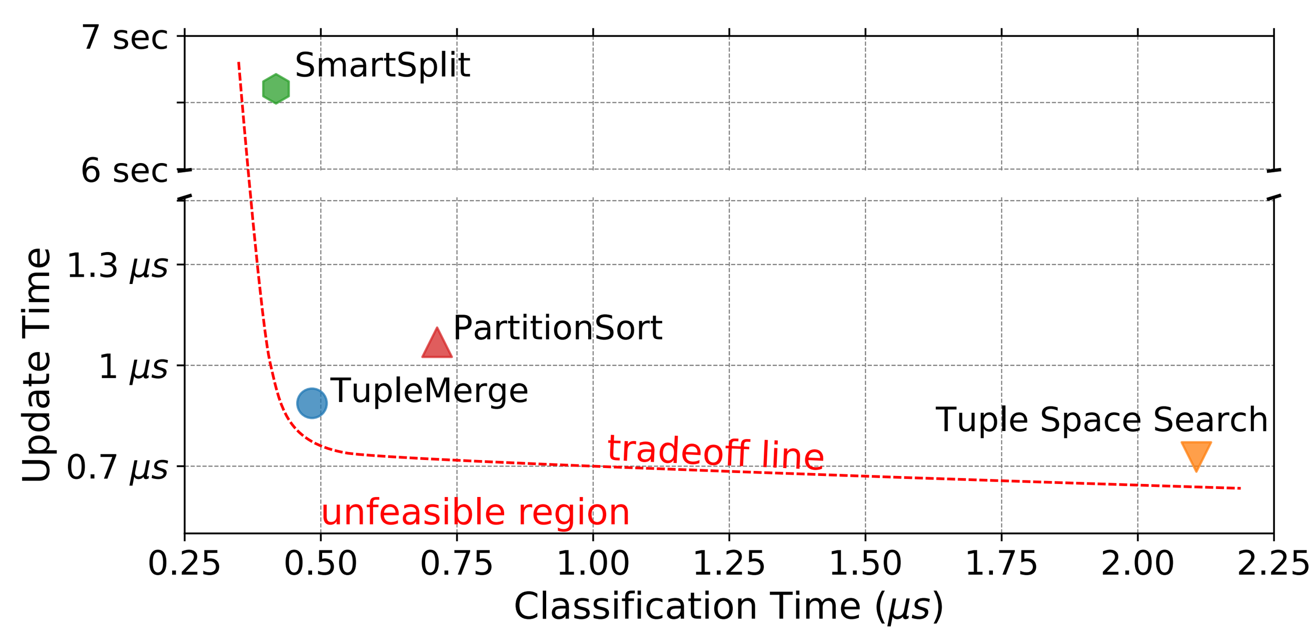
1. The Packet Classification Problem

Packet classification is a vital part of internet routing, firewalls, and other services. The object is to classify packets by applying a set of rules to the header fields of a packet.

In input to the packet classification problem there is a sequence of requests:

- **rule updates** (rule insertions or deletions) → *In SDN, a controller program* ***dynamically configures*** *the behavior of switches*
- **packets to classify** → Internet services have **real-time constraints**: If packets are not processed in a timely manner, they cause network congestion

2. Performance tradeoff (from the literature)



Several algorithms have been proposed, providing different tradeoffs

3. TupleMerge (TM)

Our approach, TupleMerge, is a new data structure that decomposes the original ruleset into smaller ones:

- It groups rules that share same characteristics in order to **perform classification as a several exact match queries**.
- It improves upon Tuple Space Search (TSS) by **relaxing the restrictions on which rules may be placed in the same table**.

TSS rulesets

Tuple1: (/24, /0, 0xFF, 0xFFFF, 0)	
Match	**Action**
R1=(160.80.73.∗, ∗, TCP, 80, ∗)	DROP

Tuple2: (/24, /0, 0xFF, 0, 0)	
Match	**Action**
R2=(160.80.73.∗, ∗, TCP, ∗, ∗)	ACCEPT
R3=(137.194.47.∗, ∗, TCP, ∗, ∗)	ACCEPT

Tuple3: (/0, /0, 0xFF, 0xFFFF, 0)	
Match	**Action**
R4=(∗, ∗, TCP, 22, ∗)	DROP

TupleMerge rulesets

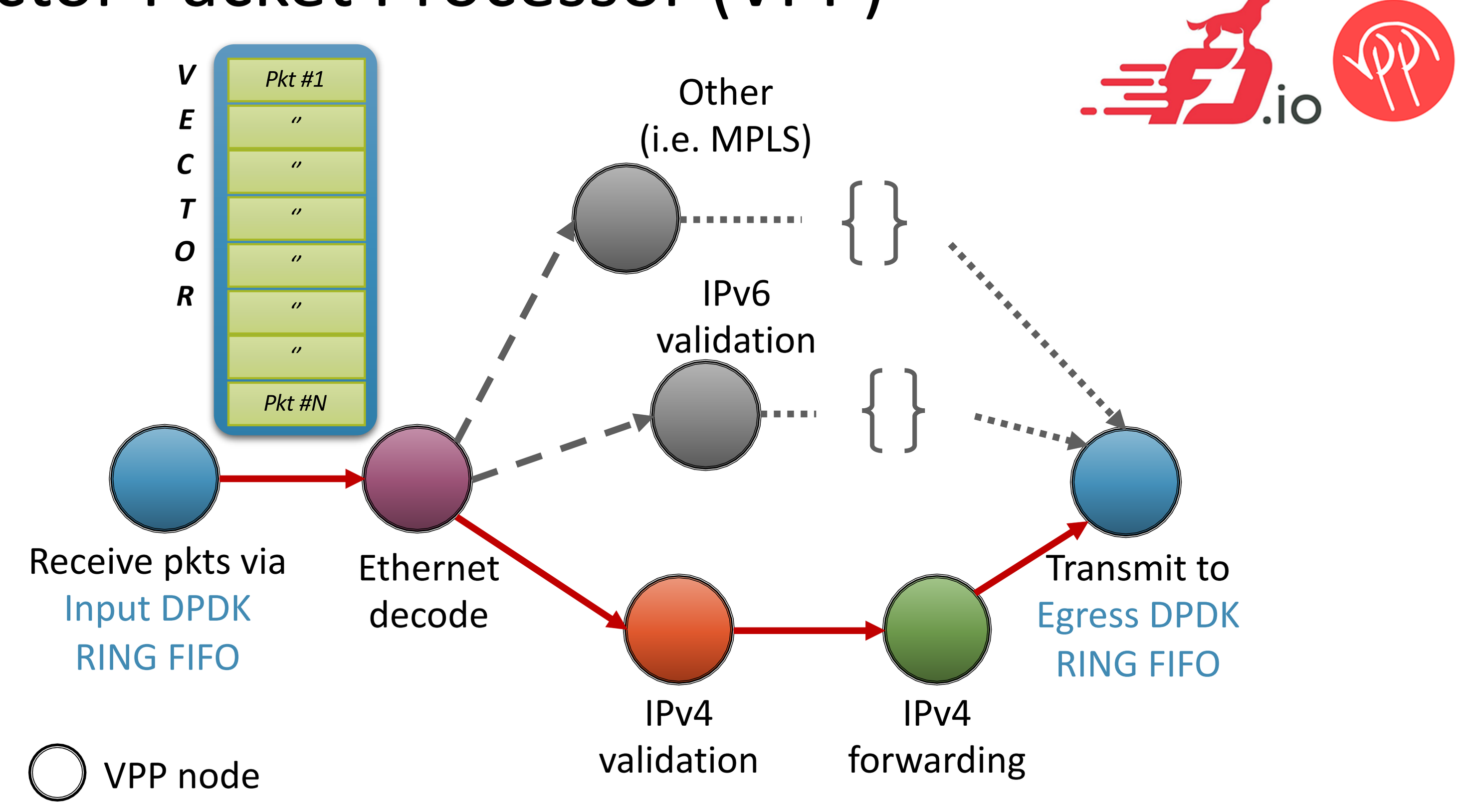
Tuple1: (/22, /0, 0xFF, 0, 0)	
Match	**Action**
R1=(160.80.73.∗, ∗, TCP, 80, ∗)	DROP
R2=(160.80.73.∗, ∗, TCP, ∗, ∗)	ACCEPT
R3=(137.194.47.∗, ∗, TCP, ∗, ∗)	ACCEPT

Tuple2: (/0, /0, 0xFF, 0xFFFF, 0)	
Match	**Action**
R4=(∗, ∗, TCP, 22, ∗)	DROP

Two important key differences:

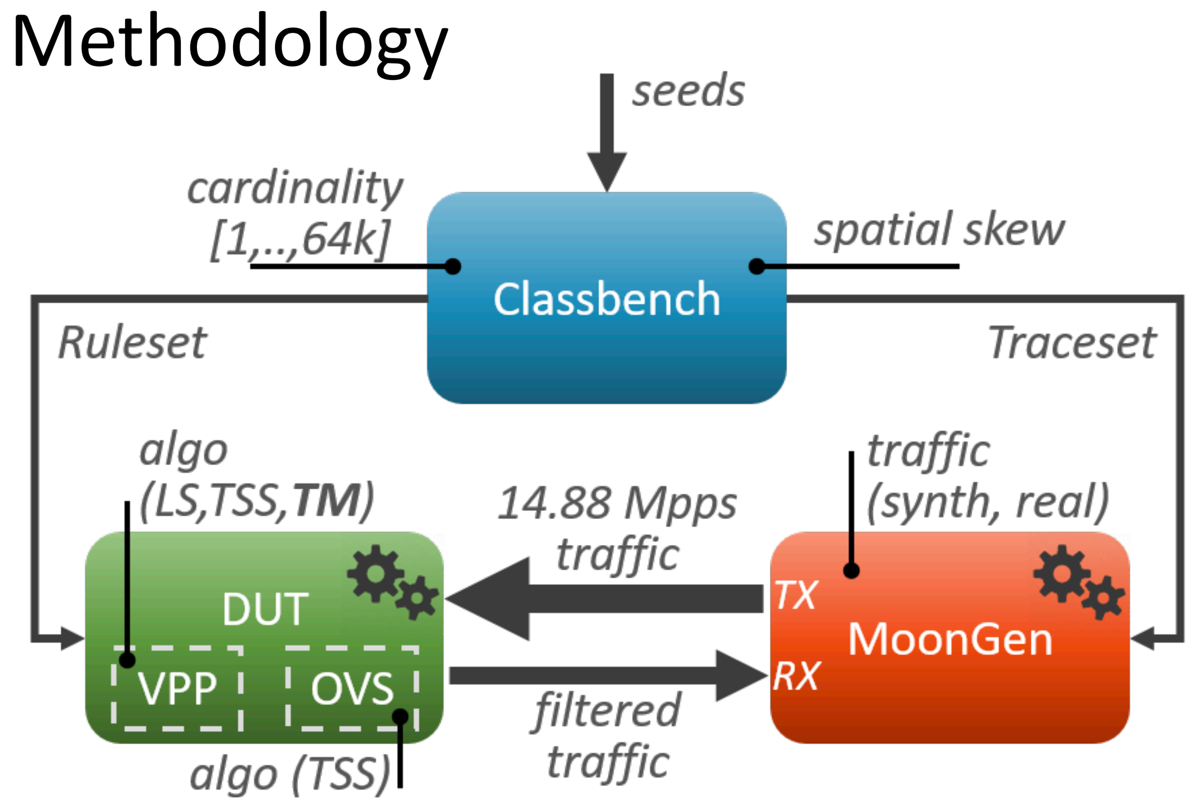
- Similar tuples are merged togheter
  - a rule R may be placed in T if **mask_T ⊆ mask_R**
- The number of collisions inside a same Table is controlled
  - if there are too many collisions, TM creates more specific tables

4. Vector Packet Processor (VPP)



- Located in **user-space** -> Kernel Bypass techniques
- Flexibility of a **modular router**
- Exploit coherence and locality of **L1-I cache**
  - -> cache miss can occur only for the first packet of the batch
- **<u>Share framework overhead across several packets.</u>**

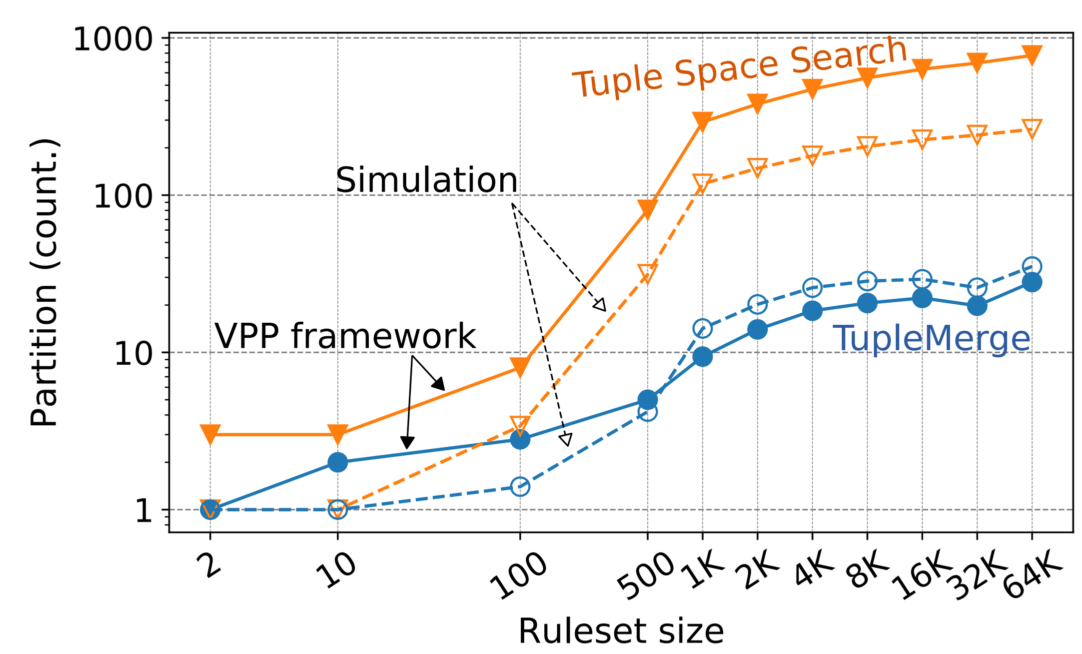
5. Test Methodology



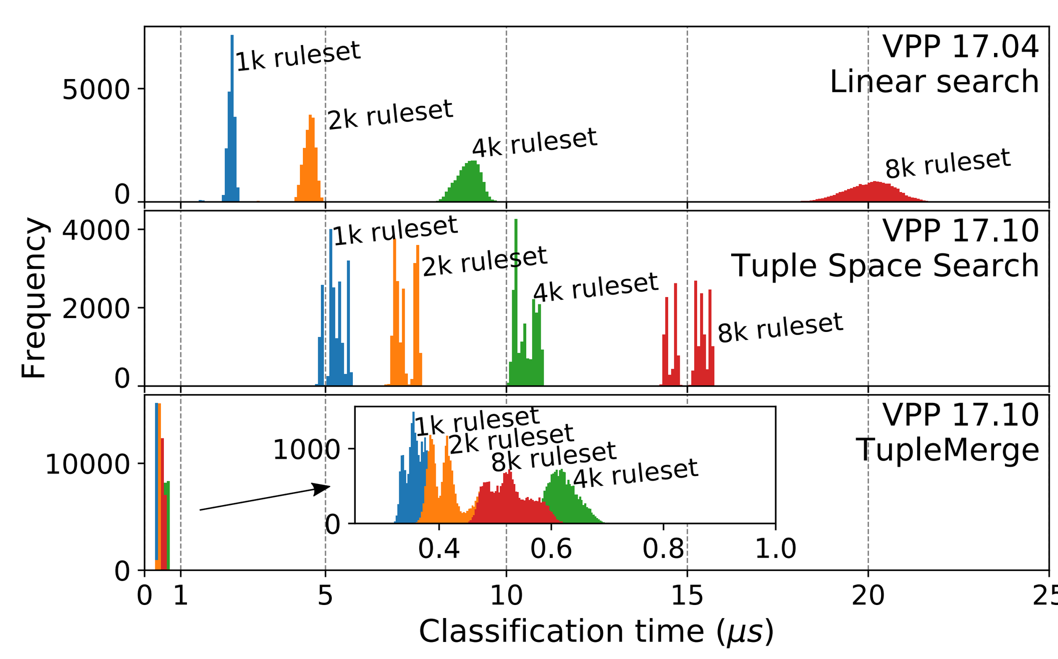
Test methodology is available: https://github.com/TeamRossi/VPP-ACL

6. Data Structure Validation

- VPP TSS requires **170% more partitions** than the original TSS
  - due to IP fragmentation
- VPP TM requires **72% less partitions** than the orginal TSS



The distribution of the per-packet classification time allows to determine whether or not classification time is stable:

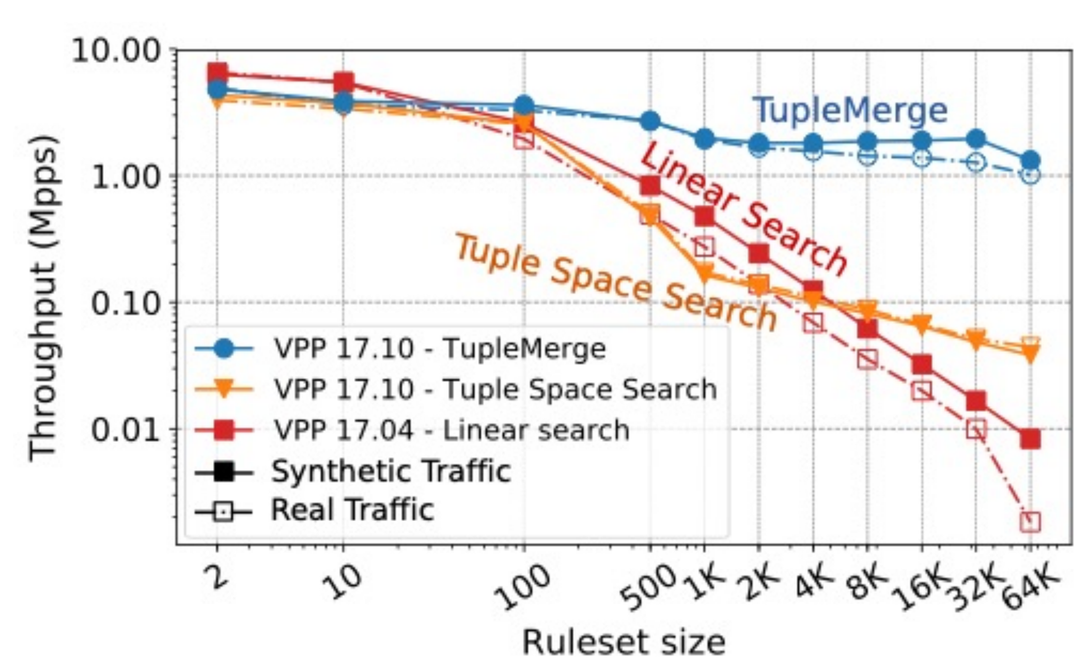


- TM classification times are all well **under 1μs**
- TM classification times are quite tight **in the range of 0.2μs to 0.7μs**, as ruleset size increases

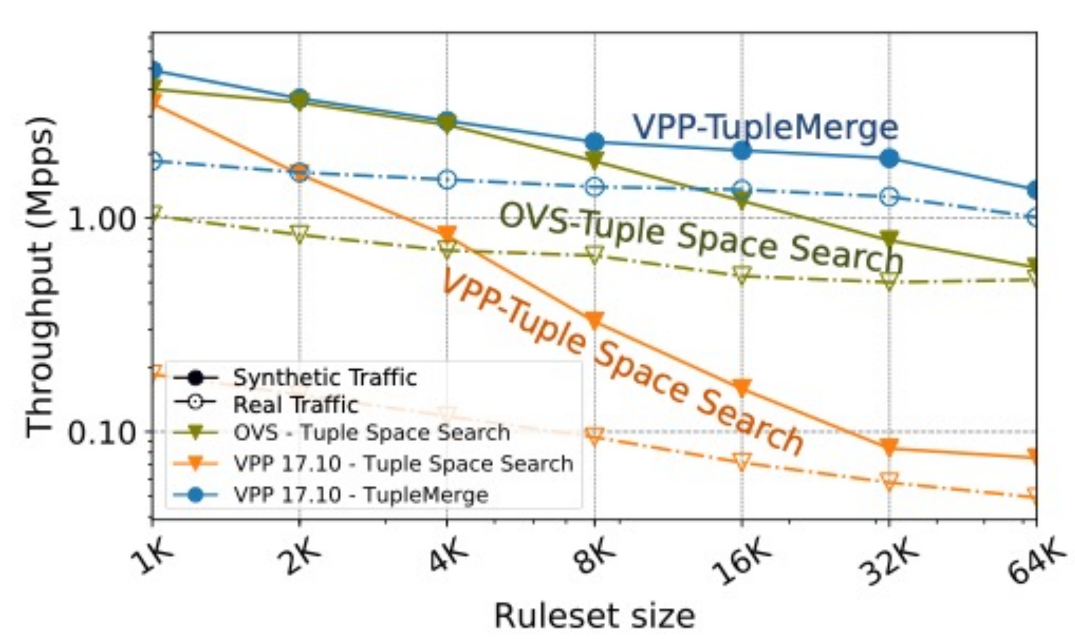
7. Experimental results

VPP TM achieves **consistently superior performance** to the previous state of the art:

- the **performance gap increases** as ruleset size increases
- **TM maintains a throughput of at least 1 Mpps** for all ruleset sizes and workloads.



We compare VPP-TM against the state of the art, represented by OVS-TSS. *In these experiments, we use OVS's three-tiered cache and the front-end cache we developed for VPP.*



- VPP TM classifies packets **12.63× faster,** on average, than OVS TSS

To Notice that OVS is designed for OpenFlow compliant rules and is limited to handle port ranges. Rather than subjecting OVS TSS to range expansion, we clearly favor OVS TSS and we modify each rule so that each port range other than all ports is a single port number.