Tuple Space Explosion:

A Denial-of-Service Attack Against a Software Packet Classifier

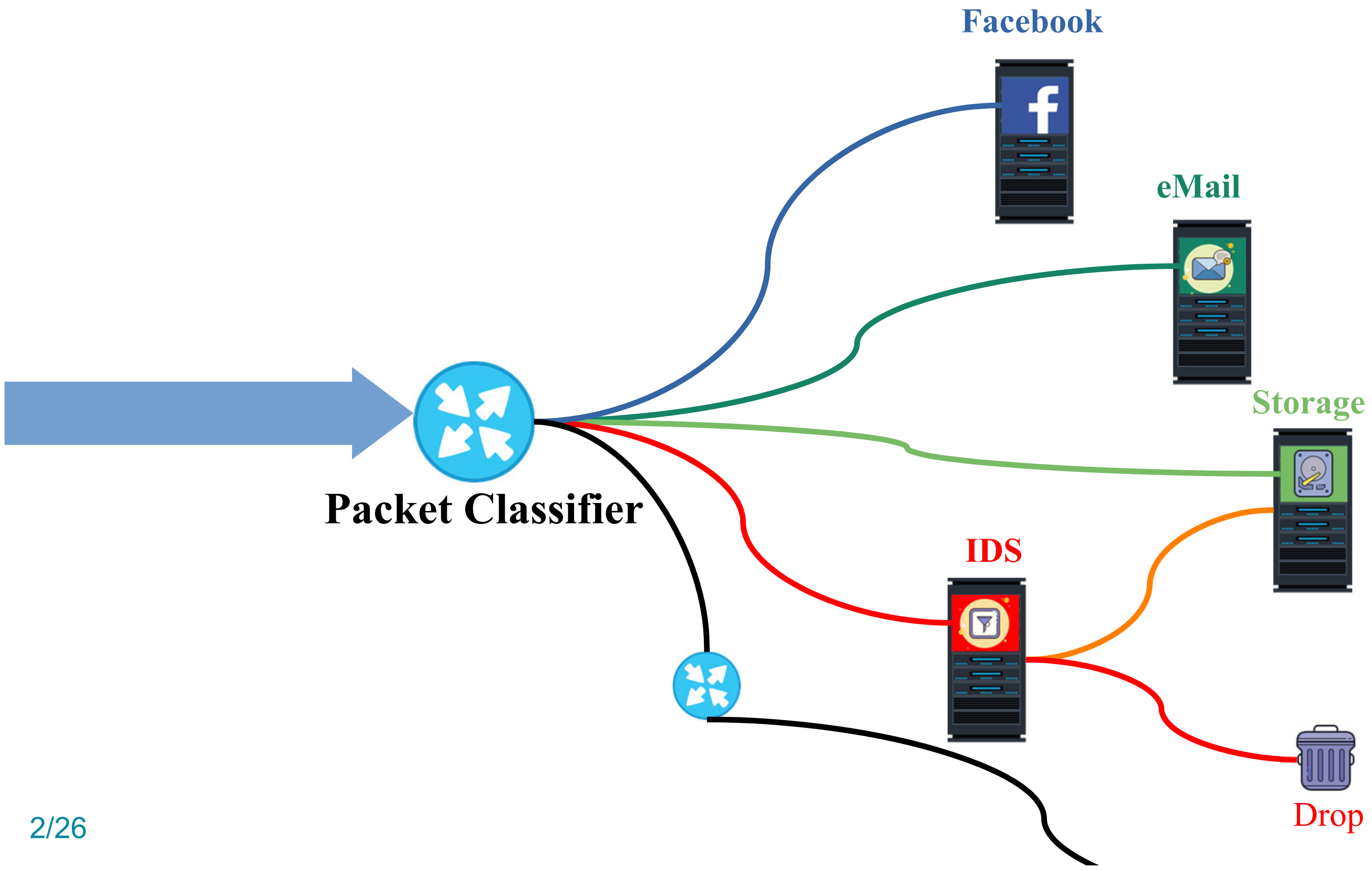
Levente Csikor, Min Suk Kang, Dinil Mon Divakaran

Attila Kőrösi, Dávid Haja, Balázs Sonkoly, Dimitrios P. Pezaros, Stefan Schmid, Gábor Rétvári

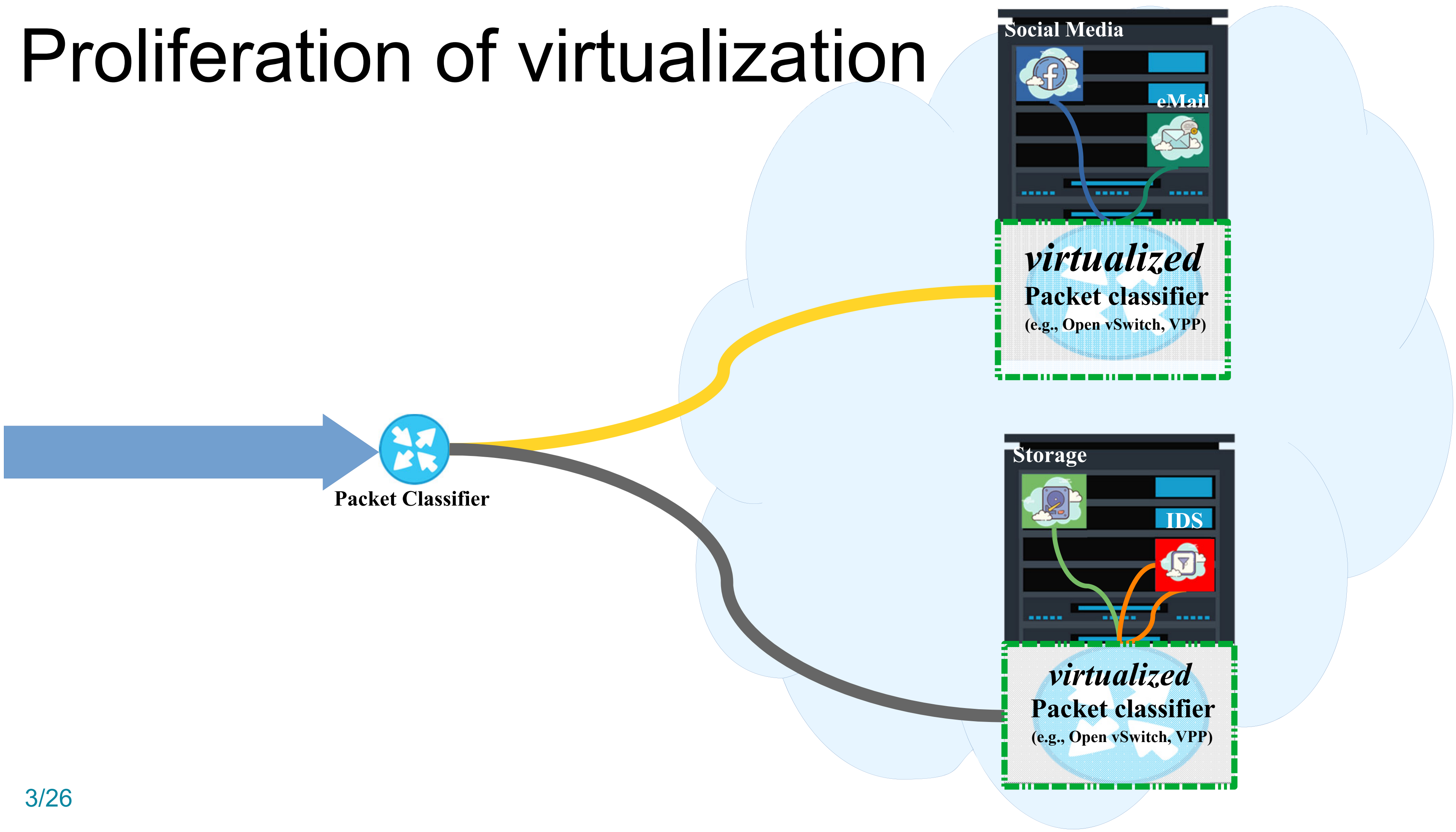
CoNEXT 2019

Dec 11, 2019

Packet Classification in the Past



Proliferation of virtualization



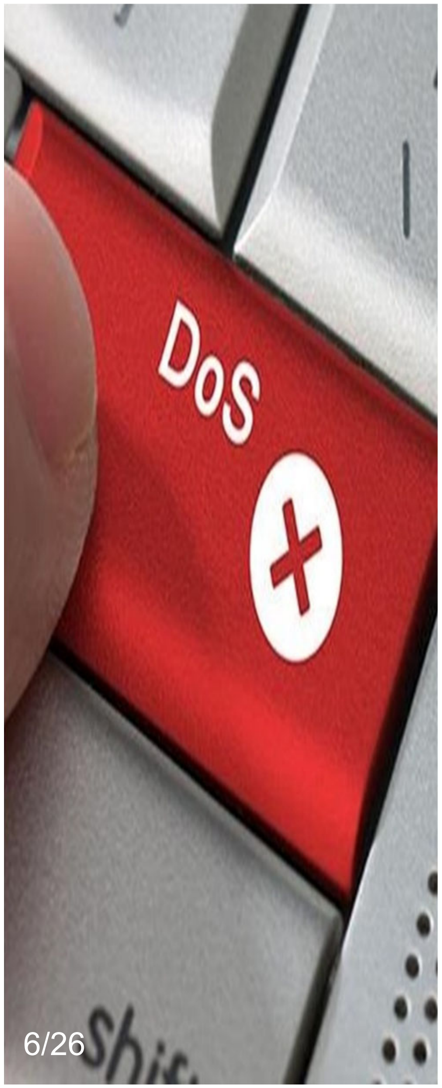
Algorithmic packet classification is expensive on general purpose processors...

Ben Pfaff et al. *"The Design and Implementation of Open vSwitch"*, USENIX NSDI 2015.

In this talk

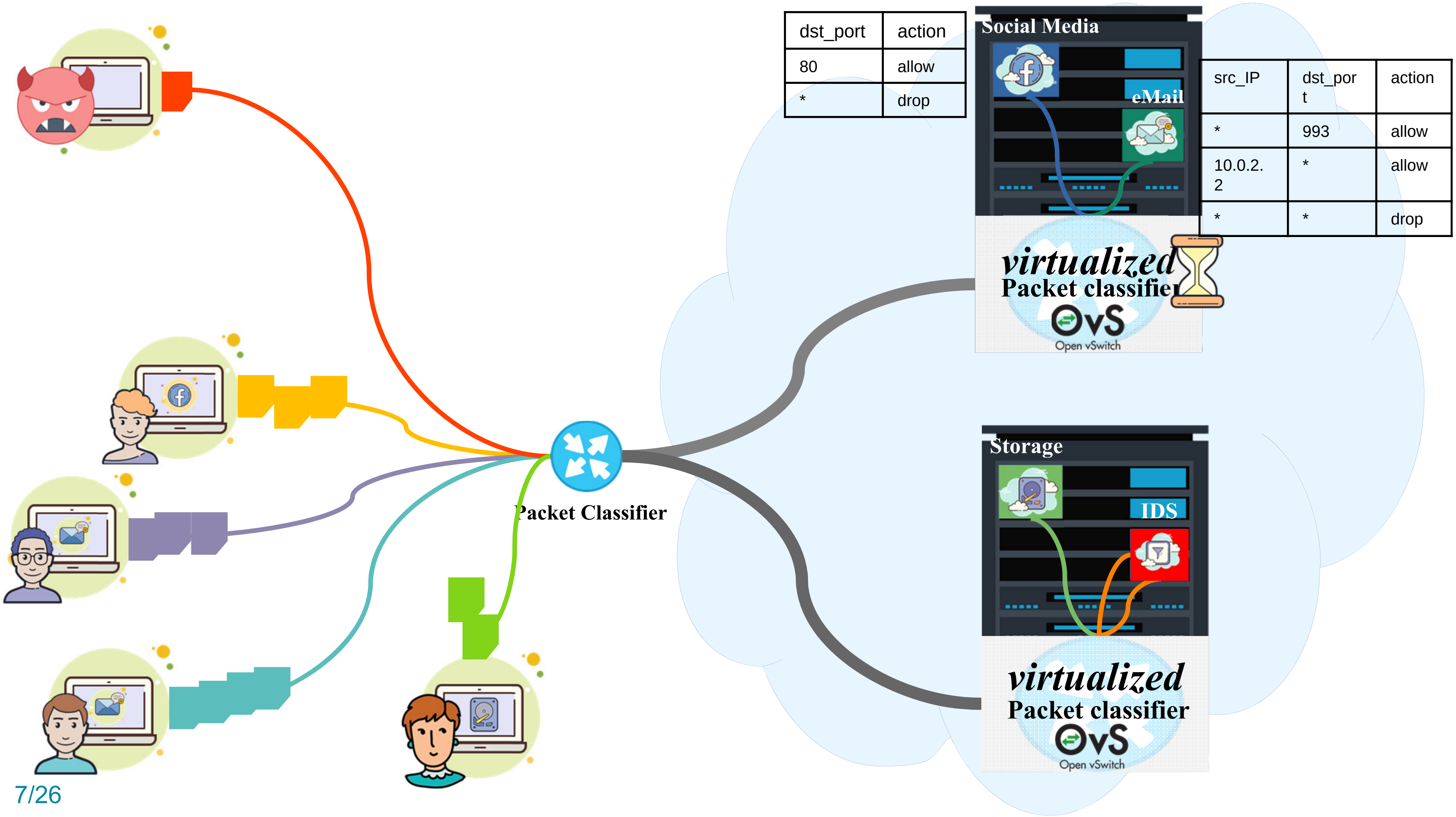
- Tuple Space Explosion (TSE): ***Family of novel Denial-of-Service (DoS) attacks*** against the ***de facto packet classifier*** algorithm (Tuple Space Search scheme) used in Open vSwitch, VPP, GSwitch, etc.
- **Remote adversary** can degrade the performance **to 12% of the baseline** (10 Gbps) with only **672 *kbps (!)* attack traffic**
- **Co-located adversary** can virtually bring down the **performance to 0%**

- Attack traffic is particularly hard to **filter out**:
  - **no attack signature** (packets w/ random headers)
  - **low-rate** (thousands of packets per second)
  - **legitimate packets**
- Countermeasures



Threat model

- **System model:**
  - typical multi-tenant cloud
  - OVS is used for packet processing
  - tenants use the Cloud Management System (CMS) to set up their ACLs to
  - access-control, redirect, log, etc.
- **Attacker's goal**
  - send some packet towards the virtual switch that when subjected to the ACLs will exhaust resources
- **Attacker's capability**
  - craft and send arbitrary packets to a target OVS
  - No privilege of the target (General TSE)
  - Co-locate with the target (Colocated TSE)



dst_port	action
80	allow
*	drop

src_IP	dst_port	action
*	993	allow
10.0.2.2	*	allow
*	*	drop

Explosion in the Tuple Space

- **Problem**: more masks → slower packet classification
- **Tuple Space Explosion phenomenon:**

1) **16-bit** TCP destination port → **16 masks**
2) **32-bit** source IP address → **32 masks**

- And that's only ONE *allow rule* on ONE *header*

- Multiple `allow rules` on multiple header fields result in an exponential growth → cross-product

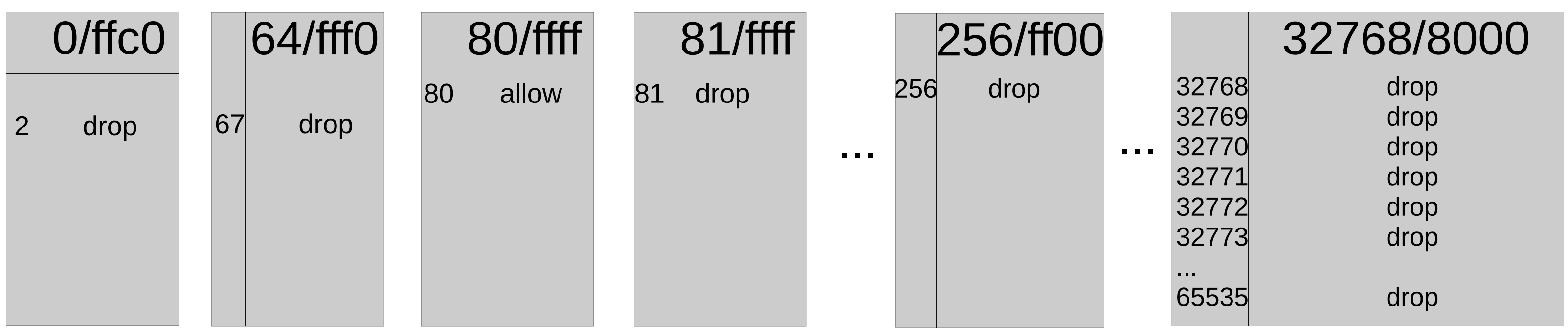
- matching on either **1)** or **2)** → 16*32 = **512 masks**

(TSE)

- **Goal**: blow up the tuple space
  - Spawn as many masks (and hashes) as possible
  - to make classification a costly linear search
  - One packet for each bucket
  - port=[0, 64, 80, 81, …, 32768] (16 packets)

MFC masks in OVS

TCP DST PORT	action
80	output:1
*	drop



(TSE)

- Without the flow table → **Difficult**
  - All possible packets seems fine
  - BUT: 2^k packets for a header of k bits!
    - too much effort
    - easily detectable (like a portscan, easily becomes volumetric)

- Can we just send *random* packets?

TSE w/ random packets

- Q: What are the chances that a random header spawns a new mask (and hash)?

dport=32769

	32768/8000
32768	drop
32769	drop
32770	drop
32771	drop
32772	drop
32773	drop
...	
65535	drop

- key finding is the number of wildcarded bits (k) for header length h

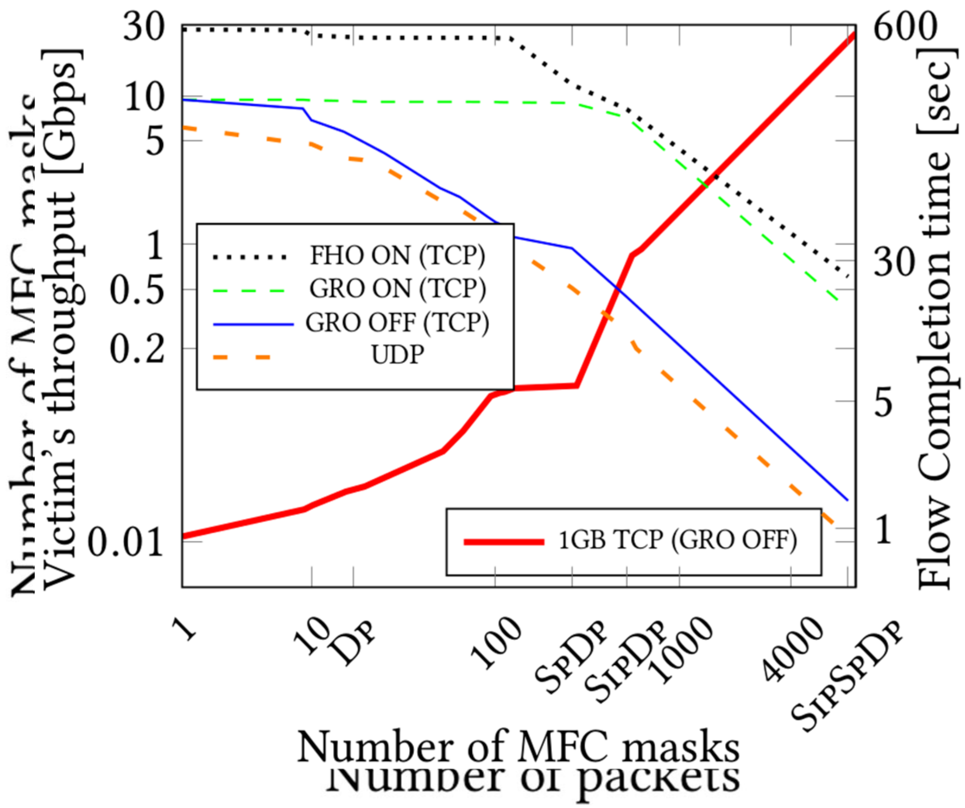
$$p_k(MFC) = \frac{2^k}{2^h}$$

- 1*** **** **** **** (32768) ~ **50%**
- 0000 0000 01** **** (64) ~ **0.1%**

	64/fff0
64	drop
65	drop
66	drop
67	drop
68	drop
69	drop
...	...
79	drop

TSE w/ random packets

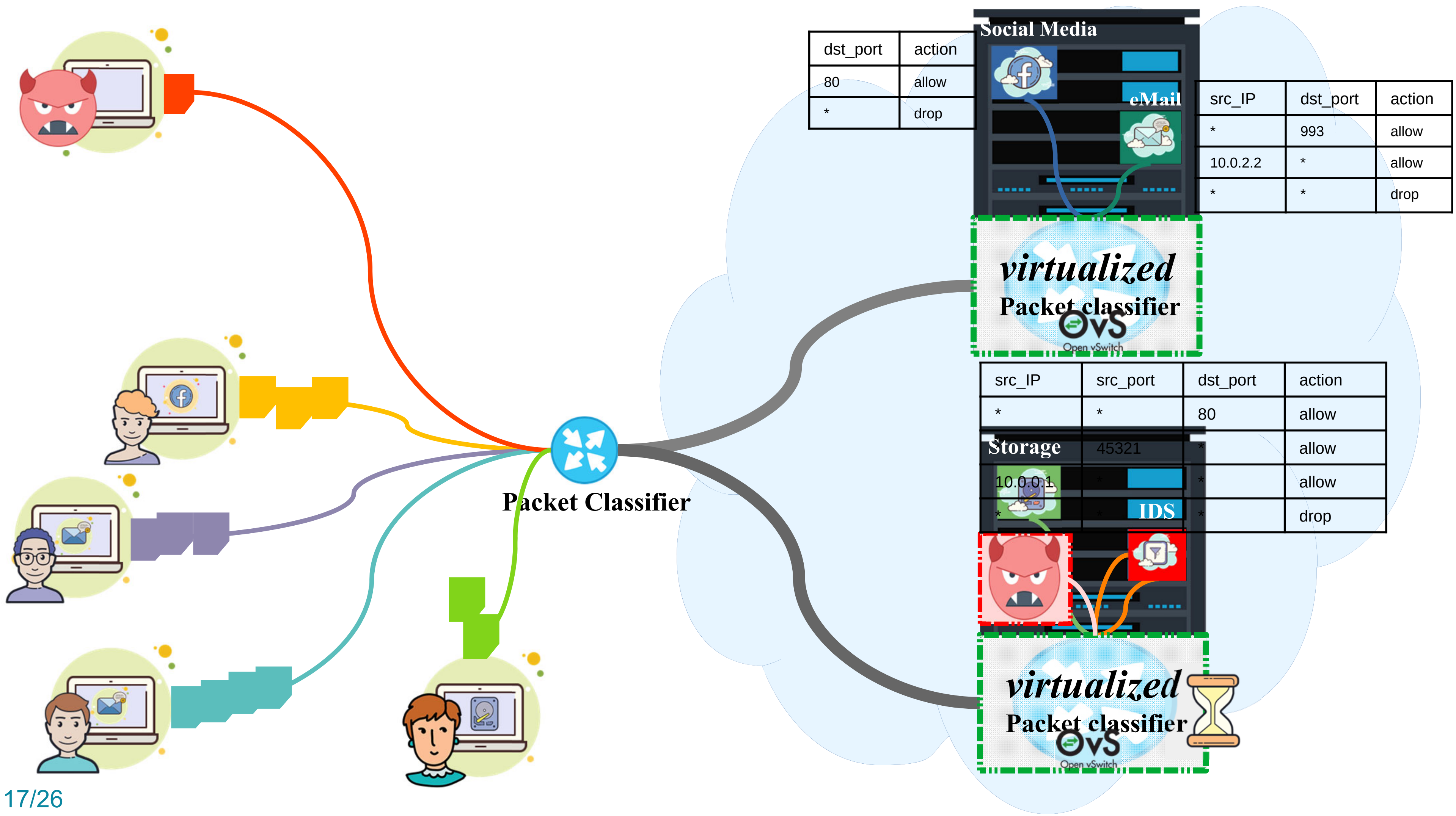
- **(M)**easured and **(E)**xpected numbers for different ACLs assumed to be installed by the victim
  drop to 10%
  - Dp
    - dst_port only
  - SipDp
    - src_IP + dst_port
  - SpDp
    - src_port + dst_port
  - SipSpDp (full-blown)
    - src_IP+ src_port + dst_port



Denial-of-Service

- Success rate of randomly generated packets
  - **672 kbps (!) attack traffic → 88% performance drop**
  - 1,000 pps → reduce from 10 Gbps to 1,2 Gbps

- **What if the adversary has more knowledge/resources?**



dst_port	action
80	allow
*	drop

src_IP	dst_port	action
*	993	allow
10.0.2.2	*	allow
*	*	drop

src_IP	src_port	dst_port	action
*	*	80	allow
[illegible]	45321	*	allow
10.0.0.1	*	*	allow
*	*	*	drop

Co-located TSE attack

- Adversary leases resources in the cloud
- Configures its own ACL
- Sends only the required number of packets
  - one packet for each mask (and hash)

- *More significant service degradation* – much *less packets*
  - **1000 pps → thousands of masks → close to 0% (full DoS)**
- However:
  - Attack is against the infrastructure *not* a specific target
  - **DoS against the co-located services "*only*"**

Effects in a broader scale

- In a cloud, an attacker can easily exploit this!
- Several public cloud deployments are affected
  - Docker/OVN (based on OVS)
  - ✓ **Kubernetes/OVN (based on OVS)**
  - Contiv/VPP Kubernetes (based on VPP)
  - ✓ **OpenStack/Neutron/OVN (based on OVS)**
  - OpenStack/Neutro-VPP (based on VPP)

Countermeasures

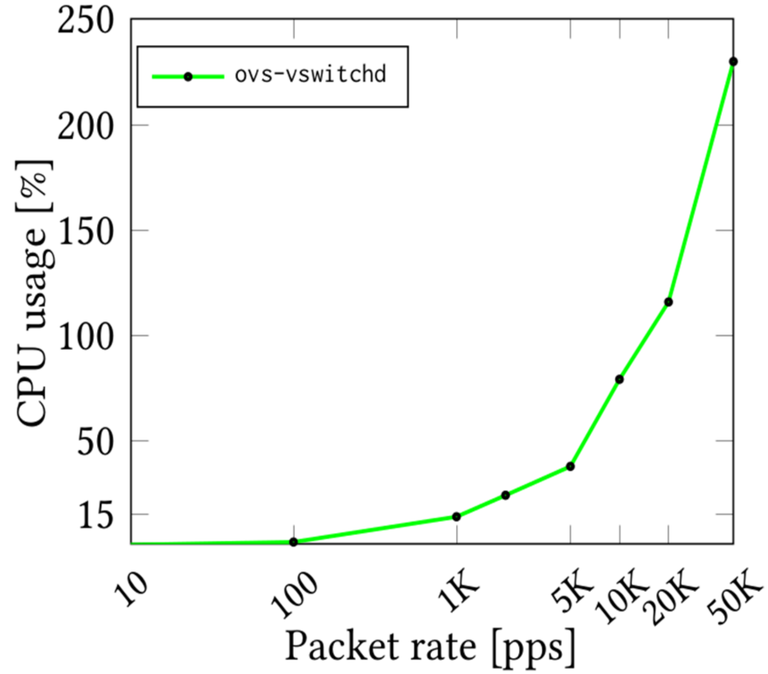
- **Filtering out** the attack traffic is **hard**
  - legitimate traffic
  - no attack signature (random packets w/ random headers)
  - low-attack rate (thousands of packets per second)

- A long term solution
  - Different classifiers:
  - Hierarchical trees, HyperCuts, HaRP, etc.

MFC Guard (MFCg) in action

MFC Guard (MFCg)

- When MFC is cleaned the victim's performance goes back to its baseline
  - attack packets → slow path
- CPU overhead?
  - **1 *kpps* attack rate = 15% CPU usage**
  - **10 *kpps* attack rate = 80% CPU usage**



General TSE

- Random packets
  - Probability that from *n* random packets there will be at least 1 packet that sparks an MFC entry for a given *k* is:

$$p_{(k,n)}(MFC) = (1 - (1 - p_k(MFC))^n) * C_k$$

  - C_k is the number entries for a given *k* (e.g., k=0, C_k = 2
- Expected value can be formalized by:

$$\mathbb{E}_{(k,n)}(MFC) = \sum_{k=0}^{h} p_{(k,n)}(MFC)$$

Countermeasures

- Immediate yet impractical remedies
  - offload ACL implementation to a different switch
    - others might suffer from the same attack
  - high performance gateway appliance
    - cannot help against an attack within the cloud
  - switch MFC completely OFF
    - biggest performance improvement so far

Tuple Space Search

TCP DST PORT	action
80	output:1
*	drop

- entries matching on the same header are collected into a hash
  - masked packet headers can be found fast
- Masks and associated hashes are searched sequentially
- in

Can be a costly linear search in case of lots of mas

- PKT_IN → APPLY_MASK → LookUp → Repeat until found

dport=80

dport=32777

	0/ffc0
1	drop
2	drop
3	drop
4	drop
5	drop
6	drop
...	...
63	drop

	64/fff0
64	drop
65	drop
66	drop
67	drop
68	drop
69	drop
...	...
79	drop

	80/ffff
80	allow

	81/ffff
81	drop

...

	256/ff00
256	drop
257	drop
258	drop
259	drop
260	drop
261	drop
...	...
511	drop

...

	32768/8000
32768	drop
32769	drop
32770	drop
32771	drop
32772	drop
32773	drop
...	
65535	drop