The Design and Implementation of Open vSwitch

Ben Pfaff*
Justin Pettit*
Teemu Koponen*
Ethan J. Jackson*
Andy Zhou*
Jarno Rajahalme*
Jesse Gross*
Alex Wang*
Jonathan Stringer*
Pravin Shelar*
Keith Amidon†
Martin Casado*

*VMware
†Awake Networks
What is Open vSwitch?

From openvswitch.org:

Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag).
Where is Open vSwitch Used?

- **Broad support:**
  - Linux, FreeBSD, NetBSD, Windows, ESX
  - KVM, Xen, Docker, VirtualBox, Hyper-V, …
  - OpenStack, CloudStack, OpenNebula, …

- **Widely used:**
  - Most popular OpenStack networking backend
  - Default network stack in XenServer
  - 1,440 hits in Google Scholar
Open vSwitch Architecture

- **VMs**
  - VM 1
  - VM 2
  - VM n

- **Hypervisor**
  - ovsdb-server
  - ovs-vswitchd
  - kernel module

- **Controller**
  - OVSDB
  - OpenFlow

- **NICs**
Network Virtualization Use Case

- OpenFlow tables
  - Table 0
    - Flow 1
    - Flow 2
    - ...
  - Table 1
    - Flow 1
    - Flow 2
    - ...
  - Table 24
    - Flow 1
    - Flow 2
    - ...

- Packet ingress
- Physical to Logical
- L2 Lookup
- Logical to Physical
- Packet egress
Implications for Forwarding Performance

OpenFlow tables

- **Table 0**
  - Flow 1
  - Flow 2
  - ...

- **Table 1**
  - Flow 1
  - Flow 2
  - ...

- **Table 24**
  - Flow 1
  - Flow 2
  - ...

**Physical to Logical**

- $k_0$ hash lookups

**L2 Lookup**

- $k_1$ hash lookups

**Logical to Physical**

- $k_{24}$ hash lookups

100+ hash lookups per packet for tuple space search?
Non-solutions

- All of these helped:
  - Multithreading
  - Userspace RCU
  - Batching packet processing
  - Classifier optimizations
  - Microoptimizations
- None of it helped enough: % versus x.

Classification is expensive on general-purpose CPUs!
OVS Cache v1: Microflow Cache

Microflow:
- Complete set of packet headers and metadata
- Suitable for hash table
- Shaded data below:

OpenFlow Controller (in theory) → Microflow Cache → OpenFlow Tables

- hit
- miss
Speedup with Microflow Cache

From 100+ hash lookups per packet, to just 1!
Microflow Caching in Practice

- Tremendous speedup for most workloads
- Problematic traffic patterns:
  - Port scans
    - Malicious
    - Accidental (!)
  - Peer-to-peer rendezvous applications
  - Some kinds of network testing
- All of this traffic has lots of short-lived microflows
  - Fundamental caching problem: low hit rate
Simple Caching Theory

If \( k_c << k_0 + k_1 + \ldots + k_{24} \): benefit!
Naive Approach to Reducing Table Lookups

Combine tables 0...24 into one flow table. Easy! Usually, $k_c \ll k_0 + k_1 + \ldots + k_{24}$. But:

<table>
<thead>
<tr>
<th>Table 0</th>
<th>Table 1</th>
<th>Table 24</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ip_{src}=a, ip_{dst}=e$</td>
<td>$ip_{src}=a, ip_{dst}=f$</td>
<td>$ip_{src}=a, ip_{dst}=e, \ldots, eth_{dst}=i$</td>
</tr>
<tr>
<td>$ip_{src}=b$</td>
<td>$ip_{dst}=f$</td>
<td>$ip_{src}=a, ip_{dst}=e, \ldots, eth_{dst}=j$</td>
</tr>
<tr>
<td>$ip_{src}=c$</td>
<td>$ip_{dst}=g$</td>
<td>$ip_{src}=a, ip_{dst}=e, \ldots, eth_{dst}=k$</td>
</tr>
<tr>
<td>$ip_{src}=d$</td>
<td>$ip_{dst}=h$</td>
<td>$ip_{src}=a, ip_{dst}=e, \ldots, eth_{dst}=m$</td>
</tr>
</tbody>
</table>

$n_1$ flows $\times n_2$ flows $\times \ldots \times n_{24}$ flows

"Crossproduct Problem"
Lazy Approach to Reducing Table Lookups

Solution: Build cache of combined “megaflows” \textit{lazily} as packets arrive.

Table 0
\begin{align*}
  \text{ip} _ \text{src}= a \\
  \text{ip} _ \text{src}= b \\
  \text{ip} _ \text{src}= c \\
  \text{ip} _ \text{src}= d \\
\end{align*}
\begin{itemize}
  \item $n_1$ flows
\end{itemize}

Table 1
\begin{align*}
  \text{ip} _ \text{dst}= e \\
  \text{ip} _ \text{dst}= f \\
  \text{ip} _ \text{dst}= g \\
  \text{ip} _ \text{dst}= h \\
\end{align*}
\begin{itemize}
  \item $n_2$ flows
\end{itemize}

Table 24
\begin{align*}
  \text{eth} _ \text{dst}= i \\
  \text{eth} _ \text{dst}= j \\
  \text{eth} _ \text{dst}= k \\
  \text{eth} _ \text{dst}= m \\
\end{align*}
\begin{itemize}
  \item $n_{24}$ flows
\end{itemize}

Megaflow Cache
\begin{align*}
  \text{ip} _ \text{src}= a, \text{ip} _ \text{dst}= f, \ldots, \text{eth} _ \text{dst}= i \\
  \text{ip} _ \text{src}= c, \text{ip} _ \text{dst}= g, \ldots, \text{eth} _ \text{dst}= k \\
  \text{ip} _ \text{src}= d, \text{ip} _ \text{dst}= e, \ldots, \text{eth} _ \text{dst}= m \\
\end{align*}
\begin{itemize}
  \item populated \textit{dynamically} only from actually observed packets
\end{itemize}

Same (or better!) table lookups as naive approach. 
Traffic locality yields practical cache size.
OVS Cache v2: “Megaflow” Cache

- Hit
- Miss

Megaflow Cache

OpenFlow Tables

Kernel

Userspace
Generating Good Megaflows

• Goal: Less-specific megaflows.
  – Megaflows that match TCP ports are almost like microflows!
  – Naive approach tends to match every field that appears anywhere in flow tables

• Requirements:
  – online
  – fast

• Contribution: Megaflow generation improvements (section 5).
Megaflow vs. Microflow Cache Performance

- **Microflow cache:**
  - $k_0 + k_1 + \ldots + k_{24}$ lookups for first packet in microflow
  - 1 lookup for later packets in microflow

- **Megaflow cache:**
  - $k_c$ lookups for every packet

- $k_c > 1$ is normal, so megaflows perform worse in common case!

- Best of both worlds would be:
  - $k_c$ lookups for first packet in microflow
  - 1 lookup for later packets in microflow
OVS Cache v3: Dual Caches

- Microflow Cache
- Megaflow Cache
- OpenFlow Tables

Hit rates:
- $\mu$ hit
- M hit
- Miss

Diagram showing the relationships between the different cache layers and hit/miss rates.
Parting Thoughts

- Common tension: expressibility vs. performance
- OpenFlow is expressive but amenable to high performance
- An application-specific switch would not be built like OVS
  - And would likely be \textit{slower}
- Applications can freely evolve decoupled from performance
- Starting from a more general problem produced better results