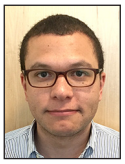
The Design and Implementation of Open vSwitch

BEN PFAFF, JUSTIN PETTIT, TEEMU KOPONEN, ETHAN J. JACKSON, ANDY ZHOU, JARNO RAJAHALME, JESSE GROSS, ALEX WANG, JONATHAN STRINGER, PRAVIN SHELAR, KEITH AMIDON, AND MARTIN CASADO

Ben Pfaff is a Lead Developer of the Open vSwitch project. He was a founding employee at Nicira and is currently at VMware. He received his PhD from Stanford University in 2007. blp@cs.stanford.edu

Justin Pettit is a Lead Developer on the Open vSwitch project. He was a founding employee at Nicira and previously worked at three successful startups focused on network security. He received his master's degree in computer science at Stanford University. jpettit@cs.stanford.edu

Teemu Koponen was the Chief Architect at Nicira before joining VMware. Teemu received his PhD from Helsinki University of Technology in 2008 and ever since has been indecisive enough to remain active within the network research community while working for the industry. He received the ACM SIGCOMM Rising Star Award 2012 for his contributions on network architectures. tkoponen@vmware.com



Ethan Jackson is a Staff Engineer at VMware and a researcher at UC Berkeley. His primary focus is on SDN, Network Function Virtualization, and high performance software switching. ejj@ej2.org

Before joining the Open vSwitch team, Andy Zhou worked on many networking and network security products using multicore NPUs. His other interests include embedded system, kernel, and computer architectures. He received his MSCS from Carnegie Mellon University. azhou@nicira.com

Open vSwitch is the most widely used virtual switch in cloud environments. Open vSwitch is a multi-layer, open source virtual switch for all major hypervisor platforms. It was designed de novo for networking in virtual environments, resulting in major design departures from traditional soft switching architectures. We detail the advanced flow classification and caching techniques that Open vSwitch uses to optimize its operations and conserve hypervisor resources. These implementation details will benefit anyone who uses Open vSwitch.

Virtualization has changed the way we do computing over the past 15 years; for instance, many datacenters are entirely virtualized to provide quick provisioning, spillover to the cloud, and improved availability during periods of disaster recovery. While virtualization has yet to reach all types of workloads, the number of virtual machines has already exceeded the number of servers and shows no signs of stopping [1].

The rise of server virtualization has brought with it a fundamental shift in datacenter networking. A new network access layer has emerged in which most network ports are virtual, not physical [2], and the first hop switch for workloads, therefore, increasingly resides within the hypervisor. In the early days, these hypervisor "vswitches" were primarily concerned with providing basic network connectivity. In effect, they simply mimicked their ToR (top-of-rack) cousins by extending physical L2 networks to resident virtual machines. As virtualized workloads proliferated, the limits of this approach became evident: reconfiguring and preparing a physical network for new workloads slows their provisioning, and coupling workloads with physical L2 segments severely limits their mobility and scalability to that of the underlying network.

These pressures resulted in the emergence of network virtualization [3]. In network virtualization, virtual switches become the primary provider of network services for VMs, leaving physical datacenter networks with transportation of IP tunneled packets between hypervisors. This approach allows the virtual networks to be decoupled from their underlying physical networks, and by leveraging the flexibility of general purpose processors, virtual switches can provide VMs, their tenants, and administrators with logical network abstractions, services, and tools identical to dedicated physical networks.

Network virtualization demands a capable virtual switch—forwarding functionality must be wired on a per-virtual-port basis to match logical network abstractions configured by administrators. Implementation of these abstractions, across hypervisors, also greatly benefits from fine-grained centralized coordination. This approach starkly contrasts with early virtual switches for which static, mostly hardcoded forwarding pipelines had been completely sufficient to provide virtual machines with L2 connectivity to physical networks.

It was this context—the increasing complexity of virtual networking, emergence of network virtualization, and the limitations of existing virtual switches—that allowed Open vSwitch to quickly gain popularity. Today, on Linux, its original platform, Open vSwitch works with most hypervisors and container systems, including Xen, KVM, and Docker. Open vSwitch

Jarno Rajahalme is part of the Open vSwitch team at VMware and has specialized in the OVS flow classifier algorithms. He received his doctor of science in technology degree from Aalto University in 2012, and is the author or co-author of tens of patents and several conference and journal papers. jrajahalme@nicira.com

Jesse Gross works on the Open vSwitch team at VMware where he has led the development of several protocols used for network virtualization. Jesse was also the original maintainer of the kernel components of Open vSwitch in Linux. He holds a degree in computer science from Stanford. jgross@vmware.com

Alex Wang is a developer on Open vSwitch. He received his master's degree in electrical engineering from UC San Diego. ee07b291@gmail.com

Jonathan Stringer hails from New Zealand, where he studied computer science specializing in networks. He's previously been involved in SDN deployments in New Zealand and now actively works on the Open vSwitch team at VMware. joe@wand.net.nz

Pravin Shelar is an Open vSwitch developer. He is currently the OVS kernel module maintainer. His most recent focus has been on tunneling. pshelar@nicira.com

Keith Amidon has spent 20+ years building high performance networks and networking software for forwarding and security. He managed the Open vSwitch development team at Nicira/VMware and recently co-founded a stealth-mode network security startup. keith@awakenetworks.com

Martin Casado is a Fellow and the SVP and GM of the Networking & Security Business Unit at VMware. He was the co-founder and CTO of Nicira Networks. He received his PhD from Stanford University where he remains a Consulting Assistant Professor. mcasado@vmware.com

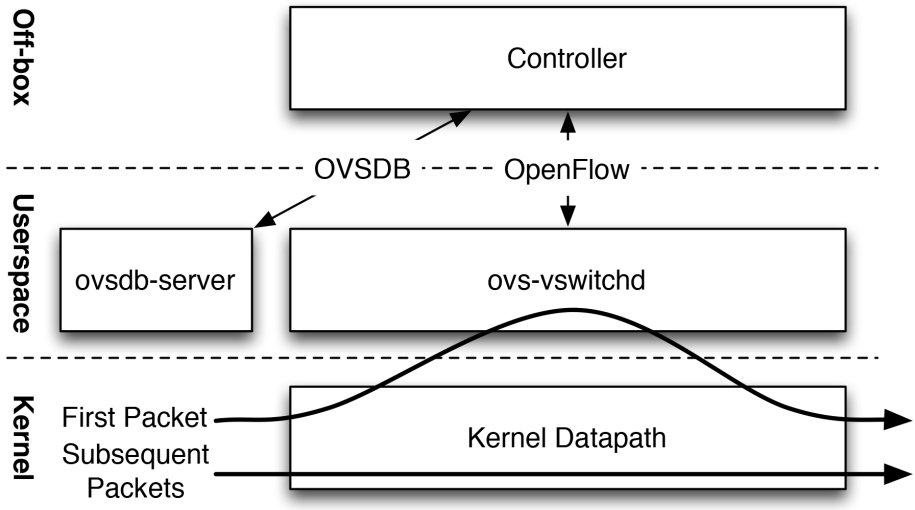


Figure 1: The components and interfaces of Open vSwitch. The first packet of a flow results in a miss, and the kernel module directs the packet to the userspace component, which caches the forwarding decision for subsequent packets into the kernel.

also works "out of the box" on the FreeBSD and NetBSD operating systems, and ports to the VMware ESXi and Microsoft Hyper-V hypervisors are underway.

In this article, we give a brief overview of the design and implementation of Open vSwitch [4]. The key elements of its design revolve around the performance required by the production environments in which Open vSwitch is commonly deployed, and the programmability demanded by network virtualization. Unlike traditional network appliances, whether software or hardware, which achieve high performance through specialization, Open vSwitch is designed for flexibility and general-purpose use. It must achieve high performance without the luxury of specialization, adapting to differences in platforms supported, all while sharing resources with the hypervisor and its workloads. For a more complete description of Open vSwitch and its performance evaluation, see our upcoming paper in the proceedings of the USENIX NSDI '15 conference [6].

Design

In Open vSwitch, two major components direct packet forwarding. The first, and larger, component is `ovs-vswitchd`, a userspace daemon that is essentially the same from one operating system and operating environment to another. The other major component, a *datapath kernel module*, is usually written specially for the host operating system for performance.

Figure 1 depicts how the two main OVS components work together to forward packets. It is the kernel datapath module that receives the packets first, from a physical NIC or a VM's virtual NIC. There are then two possibilities: either `ovs-vswitchd` has given the datapath instructions on how to handle packets of this type or it has not. In the former case, the datapath module simply follows the instructions, called *actions*, which list physical ports or tunnels on which to transmit the packet. Actions may also specify packet modifications, packet sampling, or instructions to drop the packet. In the other case, where the datapath has not been told what to do with the packet, it delivers it to `ovs-vswitchd`. In userspace, `ovs-vswitchd` determines how the packet should be handled, then it passes the packet back to the datapath with the desired handling. Usually, `ovs-vswitchd` also tells the datapath to cache the actions, for handling similar future packets.

Open vSwitch is commonly used as an SDN switch, and the main way to control forwarding is OpenFlow [5]. It is the responsibility of `ovs-vswitchd` to receive OpenFlow flow tables from an SDN controller, match any packets received from the datapath module against these OpenFlow tables, gather the actions applied, and finally cache the result in the kernel datapath. This allows the datapath module to remain unaware of the particulars of the OpenFlow wire protocol, further simplifying it. From the OpenFlow controller's point of view, the caching and separation into user and kernel components are invisible implementation details; in the controller's view, each packet visits a series of OpenFlow flow tables, and the switch finds the highest-priority flow whose conditions are satisfied by the packet and executes its OpenFlow actions.

Flow Cache Design

Algorithmic packet classification is expensive on general purpose processors, and packet classification in the context of OpenFlow is especially costly because of the generality of the form of the match, which may test any combination of Ethernet addresses, IPv4 and IPv6 addresses, TCP and UDP ports, and many other fields, including packet metadata such as the switch ingress port. This cost is amplified by the large number of flow tables used by sophisticated SDN controllers: for example, VMware NSX [3] uses about 15 classifications per packet at minimum.

Open vSwitch uses two strategies to maximize performance in the face of expensive packet classification. The first strategy is to optimize the classification itself, by refining the classification algorithms and our implementations of them over time. The second strategy is to perform fewer classifications through effective use of caching. This section introduces the flow cache design, and the following section delves into the details.

Open vSwitch's kernel datapath initially cached *microflows*, that is, each cache entry had to match on all of the fields supported by OpenFlow. Microflow caching is very fine-grained: each cache entry matches, roughly, one stream of packets in a single transport connection. A microflow cache can be implemented as a hash table, which allows the kernel module to be very simple.

Microflow caching is effective with the most common network traffic patterns, but it seriously degrades when faced with large numbers of short-lived connections. In such cases, many packets miss the cache and must not only cross the kernel-userspace boundary, but also execute a long series of expensive packet classifications. In production, this kind of traffic can be caused by port scans, network management tools, P2P applications, malware, and other sources. None of these is common, but they happen often enough that customers notice the issue.

To improve performance under those traffic patterns, we augmented the microflow cache with a *megaflow cache*. The megaflow cache is a single flow lookup table that supports generic matching, i.e., it supports caching forwarding decisions for larger aggregates of traffic than connections through wildcarding. The megaflow cache somewhat resembles a general-purpose OpenFlow table, but it is simpler in two ways: it does not have priorities, which speeds up packet classification because any match is a "best match," and there is only one megaflow table, instead of a pipeline of them, so any packet needs only one classification rather than a series. In the common case, a megaflow lookup remains more expensive than a microflow cache lookup, so we retained the microflow cache as a first-level "exact-match cache," consulted before the megaflow cache. This reduces the cost of megaflows from per-packet to per-microflow.

Caching-Aware Packet Classification

Open vSwitch uses a *tuple space search* classifier [7] for all of its packet classifications, both kernel and userspace. To understand how tuple space search works, imagine that all the flows in an Open vSwitch flow table matched on the same fields in the same way: for example, all flows match the source and destination Ethernet address but no other fields. A tuple search classifier implements such a flow table as a single hash table. If the controller then adds new flows with a different form of match, the classifier creates a second hash table that hashes on the fields matched in those flows. With two hash tables, a search must look in both hash tables. If there are no matches, the flow table doesn't contain a match; if there is a match in one hash table, that flow is the result; if there is a match in both, then the result is the flow with the higher priority. As the controller continues to add more flows with new forms of match, the classifier similarly expands to include a hash table for each unique match, and a search of the classifier must look in every hash table.

As Open vSwitch userspace processes a packet through its OpenFlow tables, it tracks the packet field bits that were consulted as part of the forwarding decision. This technique constructs an effective megaflow cache from simple OpenFlow flow tables. For example, if the OpenFlow table only looks at Ethernet addresses (as would a flow table based on L2 MAC learning), then its megaflows will also look only at Ethernet addresses. On the other hand, if even one flow entry in the table matches on the TCP destination port, tuple space search examines TCP destination port of every packet, so that every packet in, for example, a port scan must go to userspace, and performance drops.

However, in the latter case, a more sophisticated classifier may be able to notice cases where the match on TCP destination can be omitted. Thus, after introduction of megaflows, much of our performance work on Open vSwitch has centered around making userspace generate megaflows that match on fewer fields. The following sections describe improvements of this type that we have integrated into Open vSwitch.

Tuple Priority Sorting

Lookup in a tuple space search classifier ordinarily requires searching every tuple. Even if a search of an early tuple finds a match, the search must still look in the other tuples because one of them might contain a matching flow with a higher priority. We improved on this by searching the hash tables from largest to smallest maximum priority. Then a successful search can often terminate early because the current match is known to be higher-priority than any possible later match.

Staged Lookup

Even if a tuple includes many fields, a single field might be enough to tell that a search must fail: for example, if all the flows match on a destination IP that is different from the one in the packet we are looking up, then it suffices to just examine the destination IP field. The staged lookup optimization makes use of this observation by adding to a generated megaflow only match fields actually needed to determine that the tuple's flows did not match.

The tuple implementation as a hash table over all its fields made such an optimization difficult. One cannot search a hash table on a subset of its key. We considered other data structures, such as tries or per-field hash tables, but these increased search time or space requirements unacceptably.

The solution we implemented statically divides fields into four groups, in decreasing order of traffic granularity: metadata (e.g., the switch ingress port), L2, L3, and L4. We changed each tuple from a single hash table to an array of four hash tables, called *stages*: one over metadata fields only; one over metadata and L2 fields; one over metadata, L2, and L3 fields; and one over all fields. A lookup in a tuple searches each of its stages in order. If any search turns up no match, then the overall search of the tuple also fails, and only the fields included in the stage last searched must be added to the megaflow match.

Prefix Tracking

Flows in OpenFlow often match IPv4 and IPv6 subnets to implement routing. When all the flows that match on such a field use the same subnet size, for example, all match /16 subnets, this works out fine for constructing megaflows. If, on the other hand, different flows match different subnet sizes, like any standard routing table does, the constructed megaflows match the longest subnet prefix: for example, any host route (/32) forces all the megaflows to match full addresses. Suppose, for example, Open vSwitch is constructing a megaflow for a packet addressed to 10.5.6.7. If flows match subnet 10/8 and host 10.1.2.3/32, one could safely install a megaflow for 10.5/16 (because 10.5/16 is completely inside 10/8 and does not include 10.1.2.3), but without additional optimization Open vSwitch installs 10.5.6.7/32.

We implemented optimization of prefixes for IPv4 and IPv6 fields using a trie structure. If a flow table matches over an IP address, the classifier executes an LPM lookup for any such field *before* the tuple space search, both to determine the maximum megaflow prefix length required, as well as to determine which tuples can be skipped entirely without affecting correctness.

We also adopted prefix tracking for L4 transport port numbers. This prevents high-priority ACLs that match specific ports from forcing all megaflows to match the entire port field.

Cache Invalidation

The flip side of caching is the complexity of managing the cache. Ideally, Open vSwitch could precisely identify the megaflows that need to change in response to some event. For some kinds of events, this is straightforward, but the generality of the OpenFlow model makes precise identification difficult in other cases. One example is adding a new flow to an OpenFlow table. Any megaflow that matches a flow in that OpenFlow table whose priority is less than the new flow's priority should potentially now exhibit different behavior, but we do not know how to efficiently (in time and space) identify precisely those flows. The problem is worsened by long sequences of OpenFlow flow table lookups. We concluded that precision is not practical in the general case.

To revalidate the cached flows, Open vSwitch has to examine every datapath flow for possible changes. Each flow has to be passed through the OpenFlow flow table in the same way as it was originally constructed so that the generated actions can be compared against the ones currently installed in the datapath. This is time-consuming if there are many datapath flows or if the OpenFlow flow tables are complicated. Older versions of Open vSwitch were single-threaded, which meant that the time spent reexamining all of the datapath flows blocked setting up new flows for arriving packets that did not match any existing datapath flow. This added high latency to flow setup for those packets, greatly increased the overall variability of flow setup latency, and limited the overall flow setup rate. Therefore, Open vSwitch had to limit the maximum number of cached flows installed in the datapath to around 1,000. When Open vSwitch 2.1 introduced multiple dedicated threads for cache revalidation, we were able to scale the revalidation performance to match the flow setup performance, as well as greatly increase the maximum kernel cache size, to about 200,000 entries.

Open vSwitch userspace obtains datapath cache statistics by periodically (about once per second) polling the kernel module for every flow's packet and byte counters. The core use of datapath flow statistics is to determine which datapath flows are useful and should remain installed in the kernel and which ones are not processing a significant number of packets and should be evicted. Short of the table's maximum size, flows remain in the datapath until they have been idle for a configurable amount

of time, which now defaults to 10 seconds. Above the maximum size, Open vSwitch drops this idle time to force the table to shrink. The threads that periodically poll the kernel for per-flow statistics also use those statistics to implement OpenFlow's per-flow packet and byte count statistics and flow idle timeout features. This means that OpenFlow statistics are themselves only periodically updated.

The above description covers the invalidation strategy of the megaflow cache. The invalidation of the first-level microflow cache (discussed in the Flow Cache Design section) is much simpler. The kernel only opportunistically invalidates microflow entries: when a microflow cache results in a miss and the megaflow cache is about to insert a new microflow entry, an existing microflow entry is replaced if the entry hashes to a hash table bucket already in use.

Conclusion

We described the design and implementation of Open vSwitch, an open source, multi-platform OpenFlow virtual switch. Open vSwitch has simple origins, but its performance has been gradually optimized to match the requirements of multi-tenant datacenter workloads, which has necessitated a more complex design. Given its operating environment, we anticipate no change of course but expect its design only to become more distinct from traditional network appliances over time.

References

[1] T. J. Bittman, G. J. Weiss, M. A. Margevicius, and P. Dawson, "Magic Quadrant for x86 Server Virtualization Infrastructure," Gartner, June 2013.

[2] Crehan Research Inc. and VMware Estimate, March 2013.

[3] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang, "Network Virtualization in Multi-Tenant Datacenters," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)*, Seattle, WA, April 2014.

[4] Open vSwitch: http://www.openvswitch.org, September 2014.

[5] OpenFlow: https://www.opennetworking.org/sdn-resources/openflow, January 2014.

[6] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*, Oakland, CA, May 2015.

[7] V. Srinivasan, S. Suri, and G. Varghese, "Packet Classification Using Tuple Space Search," in *Proceedings of the ACM SIGCOMM '99 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 1999.