QUANTONIUM

XDPswitch

A programmable, intelligent, SDN switch

Software Defined Networking, open source software and hardware, and network device programmability are reshaping the landscape of infrastructure networking. In this paper, we propose XDPswitch, a fully programmable SDN switch for telecom networking. XDPswitch is a combination of a Linux based networking OS, SDN management, a programmable data path based on eXpress Data Path and Berkeley Packet Filters, and reference a hardware design. The goal of XDPswitch is to reduce operator infrastructure cost and establish a platform for new innovation in 5G, IPv6, and other technologies.

Introduction

In the earliest days of the Internet, from about the mid-1970s through the 1980s, general-purpose minicomputers served as routers on the Internet. As the Internet grew, routers became highly specialized devices, and vendors like Cisco and Juniper were born. Hardware routers reigned supreme through the nineties and early 2000s as the Internet grew from a tiny research network to a critical world-wide infrastructure. There was a downside side though. Proprietary hardware and software is inflexible, expensive, and susceptible to vendor lock-in. While the solutions allowed the Internet to scale, they have since become impediments to innovation and a roadblock to moving the Internet forward.

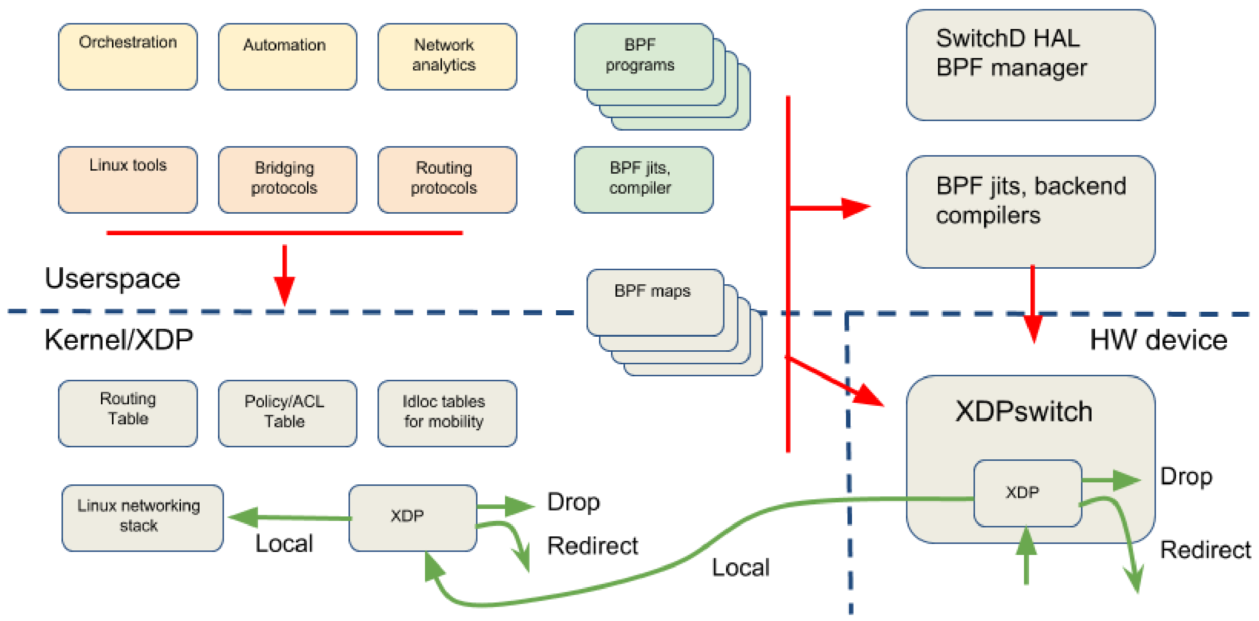
Fast forward to today. The pendulum is swinging back towards general purpose commodity solutions and programmability. There are five trends driving this:

1. *Software Defined Networking* (*SDN*). The concept is straightforward: disassociate the forwarding process (data plane) from the routing process (control plane). This breaks the constraining dependency between the data and control planes.
2. *Open source*. Open source is flexible and conceptually low cost. A number of open source projects, like Linux and OpenStack, have proven their mettle. The model of open source has extended to the realm of hardware in the Open Compute Project.
3. *Programmable network devices*. While the concept has been around a while, it is only recently that standard and portable programming models became feasible. P4 and eBPF are vast improvements over proprietary SDKs used in years past.
4. *IPv6*. Although IPv6 has been around for over twenty years, it is only now that it is seeing significant deployment in the Internet. IPv6 has a number of forward looking features that drastically improve security, privacy, network services, and mobility.
5. *5G and other new access technologies*. 5G is the next generation protocol being developed by the Third Generation Public Partnership (3GPP). It is a leap forward and promises to provide users high data rates, high reliability, super low latency, and scaling the number of users.

In this paper, we introduce *XDPswitch*. XDPswitch combines an open source network OS based on Linux, the network management of SDN, and a programmable and portable datapath via BPF that can be offloaded to hardware. The result is a scalable, cost effective, interoperable networking platform that affords innovation in IPv6, 5G, and other forwarding looking technologies.

Architecture of XDPswitch

The diagram below illustrates the architecture of XDPswitch.

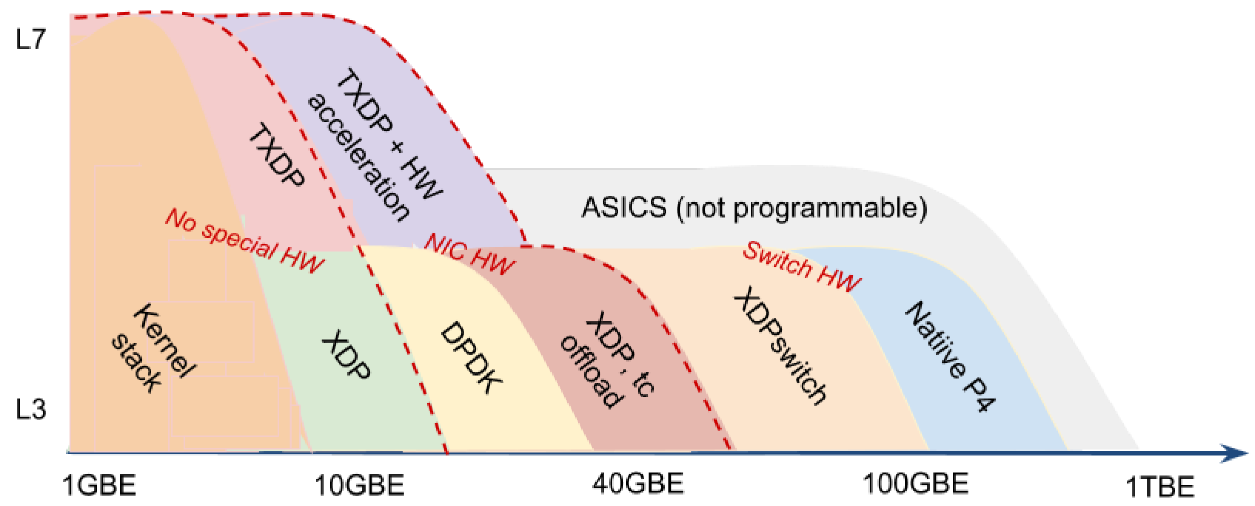


XDPswitch employs the Linux *switch-dev* model. This means that the switch is configured via Linux APIs and commands. A hardware abstraction interface, the switch-dev API, allows the system to program hardware data path. This allows switch functionality to be offloaded to a device. Routing, packet filtering, encapsulation, QoS and rate limiting, and other typical networking functions are configured using common Linux commands which in turn can be programmed in a hardware fast path. An XDPswitch is manage by a number of userspace daemons that that implement routing protocols like BGP that populate routes in the switch, speak to OpenFlow controllers, and perform other network orchestration to manage filtering, ACLs, QoS, mobility, and security. All of these use common Linux interface such as routes, iptables, and traffic control (tc) to configure the networking stack. In turn, the stack can offload functionality to the switch-dev device.

The switch-dev model also provides common interfaces and accounting for networking analytics, diagnostics, and accounting. A big advantage in retaining the Linux interface for such things is that network operations and diagnostics can be done consistently across all devices. For instance, the same tools to get packet and bytes counts can work on a server, rack switch, or core router if the consistent model is adopted. This makes the network more serviceable and reduces maintenance costs. It allows generic tools to be developed that don't need to be concerned with the idiosyncrasies of different implementations (one of the implicit goals of SDN).

Programmability of XDPswitch

The diagram below graphs the current landscape of network device programmability. The x-axis indicates the data rate of devices, and the y-axis indicates the complexity of features in terms of protocol layers going from layer 2 (link layer) to layer 4 (transport layer).



As the graph illustrates, the need for specialized hardware and solutions increases with both greater data rates and more complex functionality. The graph also shows the difference between stateless packet processing (traditional switch processing) and stateful flow processing (firewalls, proxies). XDPswitch provides programmable packet processing in the moderate to high range of link rates (40GBE-100GBE+). The foundation for programmability is eXpress Data Path (XDP) and extended Berkeley Packet Filters.

Berkeley Packet Filters (*BPF*) was invented by Steven McCanne and Van Jacobson in 1992. It was originally used as the filtering language for tcpdump. BPF is a low level byte code that runs in a kernel virtual machine. In 2014, BPF was updated to extended BPF (eBPF). eBPF increased the capabilities, added a number of instructions and registers, and aligned the architecture to map closely to the 64 bit-x86 architecture. eBPF is the primary means of Linux kernel programmability and is used in several kernel sub-systems. In the networking stack, eBPF is de facto choice for any situation where user programmability is needed. These include use in traffic control (tc), iptables, Open vSwitch, and SO_REUSEPORT.

eXpress Data Path (*XDP*) was conceived in 2016 as a generic programmability model for fast low level packet processing in the LInux kernel. It is programmed using eBPF. A major part of XDP is the verifier. SInce XDP is running in the kernel, it’s vital to ensure that a program can’t crash the system or going into an infinite loop. An XDP verifier statically checks memory accesses and loops.

Comparing XDP/BPF to alternatives

eBPF is a form of this generic byte code. It can be compiled from a number of languages including a form of restricted C using LLVM and CLANG. P4 can also be be compiled into eBPF byte code. To this later point, eBPF could be considered as an underlying machine language for the P4 language. The table below compares eBPF and P4.

Dimension	BPF	P4
Model	Byte code that can be compiled from different languages	Domain specific language with native support in switches (e.g. Barefoot)
Functionality	Packet parsing, packet editing, key lookups, advanced function via helpers	Packet parsing, packet editing, advanced actions composed from primitives
Scope	Multi-purpose: kernel XDP, tc-BPF, switch HW and NIC offload, P4 to BPF compiler	Switch HW, emulation in VMs
Parsing	Programmed with tail calls for different layers (i.e. IPv4-IPv6)	Finite state machine extracts headers based on programmed parse graph
Meta data	BPF maps (<key,value>)	Table (<keys,actions>)
Performance	Dynamic compilation,flexible but lacks some optimizations	Static compilation slightly better performance, but less flexible
Extensibility	BPF helper functions	Build from primitives, out of core source

XDP and eBPF can also be compared to DPDK and VPP. DPDK, Data Plane Development Kit, is a kernel bypass technology that runs on various CPUs to access the packet datapath of a device directly in userspace. VPP, Vector Packet Processing, is a packet programming environment based on DPDK. The motivation for DPDK was that the kernel data path was too heavyweight and feature rich to get line rate packet processing performance. XDP has answered the performance gap and has a number of advantages over DPDK and VPP:

- XDP is integrated into Linux. It is supported by a large, well established developer community. DPDK and VPP are side channels that must be carefully isolated to not interfere with the networking stack.
- XDP programs use the same data structures and definitions as used in a sockets implementation.

- XDP can work directly in conjunction with the rest of the stack including TCP. There is no userspace TCP stack implementation that compares to that of the Linux stack.
- eBPF is portable and can be directly offloaded to devices with support. This includes the ability to compile P4 into eBPF.
- XDP and eBPF can run in kernel, userspace, hardware offload. The barrier of entry to develop in XDP/eBPF is much lower than DPDK or VPP

XDPswitch software

The software stack for XDPswitch is based on a Network OS (NOS). The NOS includes basic management and control daemons for networking, configuration, policy such as packet filtering, routing, and diagnostics. The NOS controls and programs low layers such as the host networking stack, XDP, iptables, tc, etc. Higher layer interfaces into the NOS allow integration of SDN and open networking frameworks.

The XDPswitch Network OS is provided by Cumulus Linux. Cumulus Linux is not SDN by itself, but enables SDN solutions through being a highly automatable operating system, using standards based approaches for interoperability and configuration.

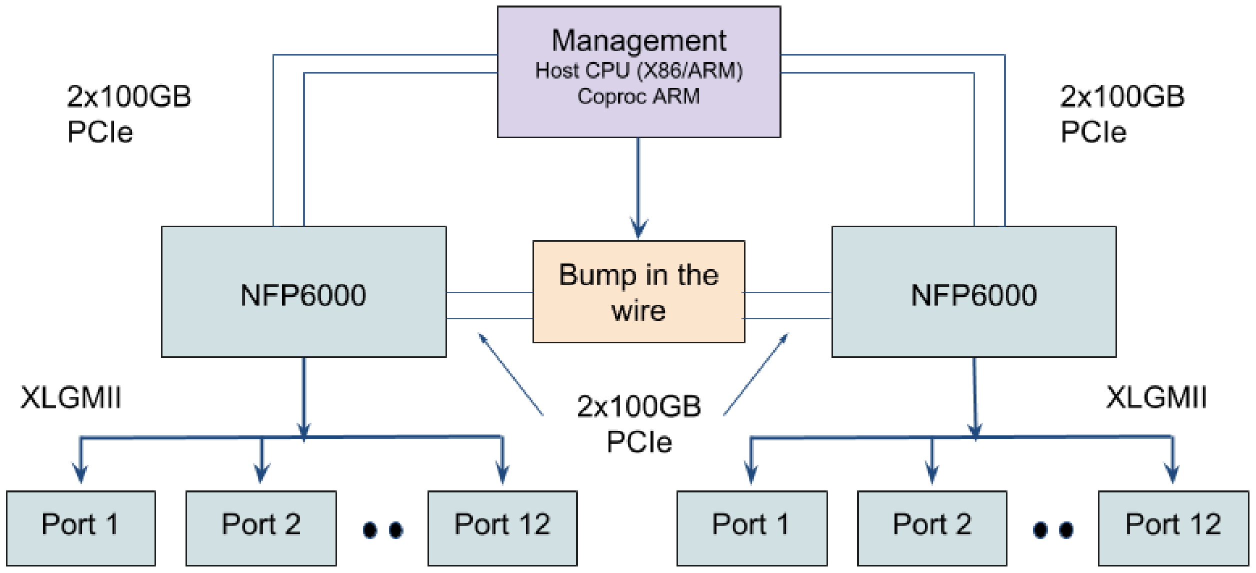
XDP programs can be written in C, P4, or other languages for which there is compiler support. XDP programs can be written in form of restricted C and compiled with LLVM and Clang. A P4 to eBPF compiler is available that first converts P4 to C, which can in turn be compiled into BPF. Optimization of compiling BPF from C is an ongoing effort.

XDP supports both extensibility and a flexible shared data model. BPF helpers can be used to extend functionality. BPF helpers are functions implemented natively in backend software or hardware that can be called by XDP programs. A wide variety of helpers have been implemented. BPF maps contain data that is shared by the XDP program and the control application. A BPF map for instance could contain the routing table for a switch.

Note that nearly all of the core infrastructure software for XDPswitch is based on open source. This core software creates a platform upon which feature customization and productization are possible. Because the platform is generic and extensible, the software that implements protocol features can move between environments. For instance, protocol processing for switching a new protocol can be developed on a plain host system and then run on a high performance hardware device in production.

Example Reference Hardware

The figure below illustrates a hardware design for a twenty-four port 40GBE switch. The switch chip in this design is the Netromone NFP6000.



Netronome builds NICs that natively support eBPF. While their initial focus has been on NICs, some of the higher end models lend themselves well for use in a switch solution. In the diagram above, a dual switch configuration is illustrated. This would employ two NFP6000 chips each of with provide twelve 40GBE ports. Each switch connects to the host via 2x100GB PCIe, and the switches themselves can be interconnected use 2x100GB PCIe.

Beside basic eBPF, the NFP6000 has some compelling features that could be exposed to an XDP program:

- IPsec and TLS crypto
- Fast table lookups
- Atomic memory lookups
- Up to 24G onboard DDR SDRAM

Going beyond a simple switch

As mentioned previously, XDPswitch is a platform for innovation that goes way beyond just providing traditional switch capabilities. In the mobile world there is increasing interest in service mapping, identifier locator protocols, addressing privacy, and scaling. XDPswitch provides the programmable and scalable datapath for those. In another trend, there is increased demand for higher protocol layer functionality in the network. For instance, in security, IPsec and TLS as network services are popping up. Such things entail stateful functionality and techniques of deep packet inspection that are not normally thought of as switch functionality.

IPv6 and 5G are motivation to change to leverage the forward looking features of these technologies. Making full use of these has both opportunities and challenges. As an example, consider Firewall and Service Tickets (FAST). The basic idea of FAST is that tickets are attached to packets by application, and they describe the precise services that should be applied to a packet. FAST has been proposed as a means to leverage and monetize network service features, like network slices and Network Function Virtualization (NFV), that have been incorporated into 5G. To make effective use of FAST an efficient and high speed, but programmable, datapath is needed.

As previously stated, the pendulum is swinging back towards commodity general purpose computers being able to serve as high speed routers. Amongst other effects, this starts to blur the line between servers and switches. Devices may take on a dual role for both. This becomes evident in the space of edge computing and access methods in IoT and mobile networks. The same device may act as both an uplink router and also a content cache for local users. The distinction in functionality is uninteresting if this arrangement ultimately achieves the goal of providing improved end user experience. XDPswitch evolved from server-side technologies and integrates seamlessly in a devices with this dual role.

Conclusion

In this paper we presented XDPswitch, a programmable SDN switch that provides a flexible platform for innovation in networking. While there are competing approaches to the problem, we believe that XDPswitch is unique and disruptive in that: 1) it is the first instance of a native XDP/BPF switch, 2) it seamlessly integrates into Linux to provide a common low cost solution, and 3) it is a path to more advanced accelerations and features that heretofore to be considered to be more in the domain of host functionality. XDPswitch serves well as a cost effective solution that fits in well with various SDN initiatives, but also provides an avenue to future innovation and product features beyond what has already been envisioned for the Internet.