NETRONOME

WHITE PAPER

Programmable Server Adapters: Key Ingredients for Success

IN THIS PAPER, WE DISCUSS ARCHITECTURE AND PRODUCT REQUIREMENTS RELATED TO PROGRAMMABLE SERVER ADAPTERS FORHOST-BASED SDN, AS WELL AS NFV (NETWORK FUNCTIONS VIRTUALIZATION AS DEFINED BY ETSI) APPLICATIONS THAT BEAR SIMILAR CHARACTERISTICS AS HOST-BASED SDN. THESE REQUIREMENTS CONSTITUTE KEY INGREDIENTS FOR SUCCESS OF SUCH SERVER ADAPTERS IN THE MARKET. COMPARISONS OF THESE PURPOSE-BUILT TECHNOLOGY AND ARCHITECTURE REQUIREMENTS VERSUS GENERAL-PURPOSE TECHNOLOGIES AND PRODUCTS (THAT TARGET THE SAME HOST-SDN USE CASES) ARE PRESENTED.

CONTENTS

INTRODUCTION

In a related whitepaper, "The Case for Programmable Server Adapters," we discussed the proliferation of host-based (Software-Defined Networking) SDN and the requirements for emerging server adapters that address the needs of host-based SDN. The scaling and efficiency challenges of host-based SDN are unique and, therefore, architectures and technology applied to server adapter products for host-based SDN must possess distinctive characteristics.

In this paper, we discuss architecture and product requirements related to programmable server adapters for host-based SDN, as well as NFV (Network Functions Virtualization as defined by ETSI) applications that bear similar characteristics as host-based SDN. These requirements constitute key ingredients for success of such server adapters in the market. Comparisons of these purpose-built technology and architecture requirements versus general-purpose technologies and products (that target the same host-SDN use cases) are presented.

THE UNIQUENESS OF DATA PLANE PROCESSING IN HOST-BASED SDN

Many instances of data plane processing have evolved in recent years, both in the open source community and in commercial deployments in the premises of data center operators. For example, the Open Virtual Switch (OVS) user space and kernel components have evolved to introduce more sophisticated and scalable tunneling and flow processing capabilities. The OVS kernel modules handle tunneling and switching. The kernel also implements a fast caching mechanism for non-overlapping or exact-match flows. More recently, support for tunnels, such as Virtual Extensible Local Area Networking (VXLAN) and wild carding match-action rules, have been added to the data path. In commercial deployments, such as Microsoft

Azure, the virtual switch supports network virtualization for tenants as well as sophisticated match-action processing for functions, such as load balancing and security.[1] In the Google Andromeda network, packet processing nodes provide match-action flow and data plane processing for firewall, security, rate limiting, routing and other functions.[2]
Data plane processing for host-based SDN has evolved in the host or server domain, implemented in software on general-purpose CPUs, such as the x86 instruction architecture. However, unlike traditional processing tasks executed on such general-purpose CPUs, data plane processing and, more specifically, tunnels encapsulation and decapsulation, and match-action based flow processing are unique in nature. This uniqueness results in significant inefficiencies when they are executed in general-purpose CPUs, specifically in terms of the number of CPU cores required to process such data plane functions. The inefficiencies exacerbate with higher scaling requirements, such as increased data rates, higher number of flows (which relates to number of users, tenants, applications, VMs, end points and other critical data center resources) and an increased level of sophistication of security and other policies applied to users, tenants and applications.

The Reasons for Inefficiencies

General-purpose CPUs have been optimized over the years to deliver higher-performance efficiencies for typical server applications, such as front-end web, middleware applications and back-end database and storage services. Such CPUs are optimized for highest single-threaded performance and assume that data access can be accelerated using warm local cache, significantly reducing the need to access memory. When memory access is needed — because the access latency to Random Access Memory (RAM) based on DDR3 or DDR4 technology is significantly higher than the clock speed at which higher-performance CPUs operate at — the CPU cycles executing application code can stall, waiting for the memory transactions to complete. As a result, the effective performance of such single-thread-optimized CPUs become significantly lower. The nature of data plane processing is exactly what causes such stalls to exacerbate because the critical element of such processing is match-action based flows; and flows, by nature, are diverse. Because data access related to flows lack the assumed locality of access suitable for caches, processing flows result in frequent cache misses and access to memory. The result is performance jitters and a large number of CPU cores being used up to process such flows at desired data rates, such 10Gbs or higher.

Multi-core and multi-threaded CPUs have evolved to help enable parallel processing across cores or threads to minimize the effects of stalling. However, because preserving network order requires synchronization across the cores or threads, resulting in significant programming complexity, most often, flows are pinned to cores or threads. This still does not solve the problem of cache misses, resulting in huge performance cliffs when datasets do not fit the cache sizes. Certain multi-core system on chip (SOC) solutions incorporate fixed-function hardware accelerators, such as for encryption or hashing to improve the situation. However, these approaches can, at best, be classified as “Band Aid” solutions to address the need for data plane processing using general-purpose CPUs. Since the solutions are not purpose-built to solve the specific challenges of data plane processing, they tend to be useful in limited use cases only, especially at lower performance and scale. Scaling requires use of a large number of CPU cores, which is not suitable from cost and power perspectives. Implementing data plane processing with efficiency and scale requires processing cores and product architectural elements that are purpose-built with special ingredients.

Key Ingredients for Success

1: Processor Multi-threading

Flow processing requires access to memory, such as DDR3- or DDR4-based RAM. To assist processing in CPU cores, hardware-based accelerators can be used for repetitive or specialized functions, such as cryptography or hashing. With single-threaded processing, as with general-purpose CPUs (like standard x86, MIPS and ARM cores), memory and accelerator access latencies result in wasted CPU cycles. For example, accessing DDR3 memory could take up to 750 cycles, and access to hardware accelerators can take even longer. Overall, total latency could be several times longer than the actual runtime instructions per packet. This could put the effective CPU utilization (or instructions per cycle) in the 10-20% range for common data plane processing tasks. Software custom coding techniques can be used to fill latency gaps; although, they are cumbersome and not easily portable. An ideal solution to the problem is processing cores that are highly multi-threaded. When processing cores are multi-threaded (for example eight threads per core), the processor pipeline is always executing useful instructions compared to single-threaded cores that would stall or execute idles. As a result, multi-threaded processing gain can be up to 800% over single threaded machines when memory or hardware accelerator access requirements are significant, as in the case with typical data plane processing requirements in host-based SDN and emerging NFV applications.

2: More Processor Cores Better than Faster Cores

General purpose CPUs are typically optimized for highest processor clock speeds at the expense of power and area. For example, big and complex pipelines with more than 15 stages, out-of-order execution and branch prediction tcapabilities are common in such CPUs. Large caches are also needed because of the lack of multi-threading, to reduce the effect of memory latency, as explained earlier. When such general-purpose CPU cores are packed into a single silicon die, as in MIPS or ARM-based multi-core SOC implementations, the effective performance gain is not as high as packing multi-threaded and a significantly larger number of smaller processing cores in the same silicon die. In other words, more optimized multi-threaded processing cores in silicon is better for data plane processing than fewer higher-performance, general-purpose CPU cores with no or lower number of threads and large caches. The use of large processor cores becomes a significant overhead when price and power constraints are placed in server adapter designs, as seen with general-purpose servers used in compute nodes.

3: Memory Multi-threading

Efficient access to memory is critical in data-intensive flow processing, and the challenge only exacerbates with larger number of flows and sophistication of flows (such as number of tuples used in matches and complexity of actions). Such requirements are bound to become pervasive in data centers with the growing need to support more users, tenants and applications, and the requirements for stringent policies related to security and service levels. While faster access to memory is important, multi-threaded access to memory is even more important. A multi-threaded memory subsystem with hardware accelerators can ensure that the processing cores do not stall. An example of such an efficient design is where multiple banks of Serial Random Access Memory (SRAM) are used with multiple crossbar inputs with

high bandwidth. Access to such SRAM banks is further accelerated using high-performance, tightly coupled dedicated hardware engines that perform critical functions, such as atomics, statistics, lookups and load balancing.

4: High-performance Distributed Mesh Fabric

The multi-threaded processing cores, hardware accelerators and multi-bank memory units previously described need to work well-synchronized, at high performance and avoiding stalls in processing. This can be enabled using an efficient high-performance fabric that connects these critical processing and data set handling elements together. Such a distributed mesh fabric must avoid contention and bus saturation issues that are common with shared bus architectures seen with general-purpose CPU-based SOC solutions. Data plane processing programs, as with tunnels and match-action processing, are inherently small and, therefore, can fit in level 1 or local memory. This is because data sets for packet processing are very small (for example, flow tables and statistics). In such scenarios, coherency of data can be handled much more efficiently by using hardware locks. As such, large instruction and data caches with multiple levels and related expensive coherency support are not relevant for data plane processing. Without coherent caches, there is no need for a coherent fabric and, therefore, the fabric can be replaced with a fatter, lower-latency-distributed mesh to avoid bus saturation.Another key advantage of a high-performance distributed mesh fabric connecting the processing, memory and hardware accelerator elements is the ability to create a modular architecture that can be scaled up and down in terms of number of processing cores available in a given product, with software compatibility preserved. For example, a single silicon design may be scaled up to support multi-port applications (such as service node or middle box applications) where the traffic is predominantly network port-to-port and bandwidth requirements could be north of 100Gbps. On the lower end, the same silicon design can be scaled down to fit the price and power profile of a traditional compute node server adapter application with single or dual port, with maximum bandwidth of 40Gbps or 50Gbps.

5: Multi-threading Optimized Programming Tools

While general-purpose CPU cores are easier to program (for example, by using standard C-based programming tools), they do not support development of optimized data plane processing applications. When bandwidth requirements and CPU efficiency requirements are relaxed, such tools, when used with general-purpose CPUs, may enable significantly easier code portability and faster time-to-market, especially when the complexity and number of flows is lower. The previous sections established the need for processor and memory multi-threading, and the importance of many smaller processing cores (instead of fewer higher-performance processing cores) when scale and efficiency becomes important. When programming such multi-threaded processing cores, it is critical that robust, easy-to-use, C-based programming tools and environments are available that provide thread-level visibility during programming and allow creation of data plane processing programs that are optimized for multi-threaded operation. It is also important that a run-to-completion programming model must be supported, avoiding scenarios where processing of complex packets or flows does not hold up processing of simple packets, and also avoid the need to “recirculate” packets. The programming tools must allow for invoking accelerators, as needed, via instructions (e.g., hash, LPM [longest prefix match] lookup, ACL [access control lists], statistics, meters, etc.)

6: Hitting Compute Node Economics

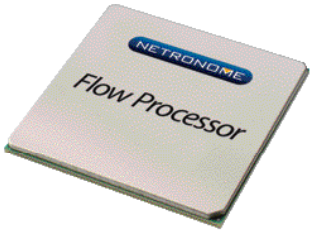
Programmable server adapters are evolving in a natural way, starting with niche, lower-volume applications, with the promise to grow into high-volume, mainstream deployments. Based on multicore SOC silicon, such adapters have found their home in appliances or purpose-built servers that are sometimes called service nodes or network nodes. There are also instances of use of Network Processing Units (NPUs) and Field Programmable Gate Arrays (FPGAs) in such applications. Because the volume of deployment for such servers has not been large in the past, data center operators have been willing to pay premium pricing for programmable server adapters capable of data plane processing. Also, the need to fit within the 25 Watts PCI Express power envelope (as seen in most general-purpose servers) has not been mandatory. Because the operation of service nodes or network nodes is not directly in the path of the traffic to and from the compute nodes, sometimes lower-performing adapters have also been found acceptable; and data center operators have resorted to using them to keep costs and power consumption down. With host-based SDN and NFV technologies moving toward mainstream adoption, the need for programmable server adapters in high-volume compute nodes is expected to rise significantly. This will require programmable server adapters to achieve typical performance requirements in compute node servers, namely 10GbE moving to 25, 40 and 50GbE in the next few years. Such adapters will have to operate at line-rate and within the PCI express power envelope of 25 Watts in most servers deployed today. And most importantly, they will have to be reasonably priced to support the volume economics of compute node servers. This means that silicon technology and data plane processing architectures used in programmable server adapters must allow for performance, scale and economics. Key ingredients 1 through 5 previously highlighted are required to enable needed data plane processing requirements at 25, 40 and 50GbE bandwidths while hitting compute node economics expected by data center operators.

7: Ready Software Ecosystem for Mainstream Adoption

Besides meeting performance, feature, price and power requirements, mainstream adoption of programmable server adapters will require an adequately supportive software ecosystem. Specifically, the server operating system kernel, user space and virtual switch networking software stacks must support installation and operation of such server adapters that can offload data plane processing applications, such as virtual network tunneling and match-action related flow processing. For example, features implemented in commercial and open source operating system environments must support data plane processing offload features, such as learning about and manipulating flows populated in hardware-based flow tables (implemented in programmable server adapters) by using a set of flow table APIs. The offload decision component in the software stack may choose to offload flows into the server adapter or handle them using the kernel space datapath (for example, in the open virtual switch or OVS kernel) or user space datapath (utilizing Data Plane Development Kit [DPDK] support for OVS). The flow table API set, for example, will need to expose and allow the discovery of hardware flow tables. For example, the API set allows discovery to occur using APIs, such as Get Tables, Get Headers, Get Actions, Get Header Graph and Get Table Graph commands. The API set also allows tables to be created and deleted using Create Table and Delete Table commands. The API set also allows flows manipulation using Get Flows, Set Flows, Delete Flows and Modify Flows commands. The kernel may provide such an offload mechanism in the form of the flow table API set, while policy decisions on discovering hardware offload capabilities and decisions on whether to offload may be delegated to

user space in the operating system. Such a flow table API set must allow for flexible match-action capabilities. For example, the API set could allow for tables with different matching schemes, such as exact match tables, tables that allow masking of match fields, those with Open Flow style actions, OVS datapath tables, as well as more advanced arbitrary pipelines of tables, including using some tables for OVS offload and others that may come earlier or later in the pipeline. Such features, when implemented in the open source community, must be up-streamed into the appropriate open source mainline trees (such as in www.kernel.org or www.openvswitch.org). Commercial operation systems and hypervisor distributions from well-known operating system vendors must include such datapath offload capabilities like the aforementioned. Operating system vendors must support qualification of programmable server adapter vendor-supplied device drivers and associated software with their distributions to enable seamless operation of such adapters and their mainstream adoption by data center operators.

Mainstream Market-ready Programmable Server Adapters



Host-based SDN and NFV deployments are expected to drive mainstream adoption of programmable server adapters. Such adapters have to be purpose-built and support key ingredients for success. Netronome's innovative C-programmable flow processors that utilize a unique small-core highly multi-threaded architecture are built to specifically solve the scaling and efficiency challenges of host-based SDN. They are unique in that they possess the key ingredients for success and are in lock step with important developments occurring in the open source, as well as commercial operating system and hypervisor ecosystems.

1 Source: Presentation by Albert Greenberg at Open Networking Summit 2014

2 Source: Presentation by Amin Vahdat at Open Networking Summit 2014

NETRONOME

Netronome Systems, Inc.
2903 Bunker Hill Lane, Suite 150 Santa Clara, CA 95054
Tel: 408.496.0022 | Fax: 408.586.0002
www.netronome.com

WP-NFP-6000-1/2016