Bluebird: High-performance SDN for Bare-metal Cloud Services

Manikandan Arumugam[1], Deepak Bansal[3], Navdeep Bhatia[1], James Boerner[3], Simon Capper[1], Changhoon Kim[2], Sarah McClure[3], Neeraj Motwani[3], Ranga Narasimhan[3], Urvish Panchal[1], Tommaso Pimpo[3], Ariff Premji[1], Pranjal Shrivastava[3], and Rishabh Tewari[3]

Arista[1], Intel[2], Microsoft[3]

Abstract

The bare-metal cloud service is a type of IaaS (Infrastructure as a Service) that offers dedicated server hardware to customers along with access to other shared infrastructure in the cloud, including network and storage.

This paper presents our experiences in designing, implementing, and deploying Bluebird, the high-performance network virtualization system for the bare-metal cloud service on Azure. Bluebird's data plane is built using high-performance programmable switch ASICs. This design allows us to ensure the high performance, scale, and custom forwarding capabilities necessary for network virtualization on Azure. Bluebird employs a few well-established technical principles in the control plane that ensure scalability and high availability, including route caching, device abstraction, and architectural decoupling of switch-local agents from a remote controller.

The Bluebird system has been running on Azure for more than two years. During this time, it has served thousands of bare-metal tenant nodes and delivered full line-rate NIC speed of bare-metal servers of up to 100Gb/s while ensuring less than 1µs of maximum latency at each Bluebird-enabled SDN switch. We share our experiences of running bare-metal services on Azure, along with the P4 data plane program used in the Bluebird-enabled switches.

1 Introduction

For some time now, Software Defined Networks (SDNs) have been foundational in enabling virtualized networks for customer workloads in multi-tenant clouds. Traditionally, the data plane of SDNs has been implemented in software as part of the end-host networking stack, typically leveraging virtual switches in hypervisors such as Open V-Switch (OVS) [16] or user-level networking libraries such as DPDK [17]. Given that scale and performance needs have grown over the years, the mechanisms available to offload such software-based packet processing have also evolved. These include solutions such as smartNICs [15], which leverage Switch-on-a-Chip (SoCs), ASICs, and FPGAs to perform packet processing at line rate without incurring significant overhead [58–60].

Today, cloud customers have even more demanding workloads in the cloud as they look to migrate their line-of-business applications and begin to phase out their own data centers. These workloads require complete control of the hardware, and in many cases, custom hardware to be hosted in the cloud. For example, workloads such as those for NetApp, Cray, SAP, and HPC [13, 53] require custom hardware. We refer to the cloud offering for supporting such workloads as bare-metal cloud services or hardware as a service (HWaaS).

Bare-metal workloads are not well-supported by traditional SDN stack implementations. In general, bare-metal servers do not offer the necessary opportunities for integration with the networking stack on the host or NIC, calling instead for a "bump-in-the-wire" approach that has no dependency on the host hardware. Since the Top-of-Rack (ToR) switches are the first network hop connected to these hosts, the ToR offers an excellent opportunity to implement this "bump in the wire."

In this paper, we introduce Bluebird, a ToR-based SDN solution that is broadly deployed in one of the largest public cloud infrastructures to enable bare-metal workloads. We also discuss the challenges, design, and operational experiences in building and designing such a solution.

Bluebird is based on programmable ASICs such as the now largely available Barefoot Tofino chipset [27] as well as upcoming merchant silicon offerings like Broadcom's SmartTOR ASIC [6]. The programmability of high-speed networking ASICs, along with the increase in scale, have made the ToR-based "bump-in-the-wire" approach feasible. Cloud providers must be able to evolve the SDN capabilities of the platform as customer requirements change. Without programmable chips, a cloud provider may have to wait for an 18-24 month technology cycle before changing their service offerings. Unless, of course, the cloud provider undertakes an off-cycle, expensive hardware replacement. Additionally, several SDN functions, such as load balancing, NAT, etc., require flow state tracking. ToR ASICs such as Barefoot Network's Tofino and Broadcom's SmartToR are now able to track millions of flows, which is critical to enable network virtualization in a ToR.

Bluebird is able to achieve line-rate throughput and deliver latencies of less than 1μs that are on par with non-virtualized environments. By leveraging route caching mechanisms, Bluebird can scale to the largest virtualized networks that exist in a public cloud. The rest of this paper is organized as follows: §2 reviews different SDN implementations; §3 describes goals and rationales behind Bluebird; §4 presents the design at the

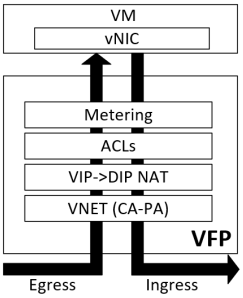

Figure 1: Virtual Filtering Platform (VFP) design.

base of our solution; §5 investigates performance; §6 explores operationalization and experience; §7 discusses related work; and §8 concludes this paper.

2 Background

In this section, we present an overview of the SDN stack implementations coexisting within Azure and how they compare with Bluebird to enable bare-metal services. Table 1 summarizes the comparison.

2.1 Host SDN

First, we consider end-host based software solutions. Figure 1 shows the Host SDN model, where the SDN software stack runs on the host machine hypervisor. Packets to and from the VMs are processed by a programmable virtual switch (vSwitch) operating within the hypervisor environment. The vSwitch's design is critical to the effectiveness of this solution since it is responsible for implementing the forwarding policy while still minimizing overhead. Since the host has already processed every packet reaching the physical forwarding layer, the physical networking equipment can be relatively simple. However, processing packets in software is expensive and competes with client software for host resources. This results in reduced revenue and has a negative impact on network efficiency as congestion increases CPU use [15].

Using Hyper-V as the hypervisor and the Virtual Filtering Platform (VFP) [14] as the vSwitch, Azure primarily employs the Host SDN model across its fleet (Figure 1). The VFP implements SDN policies by acting as a programmable platform accessible to the controllers running Azure's SDN. The VFP is organized in layers, which are stateful flow tables that enforce the controller's policy. Each layer implements a specific set of inbound and outbound rules that can filter and transform packets. A packet traverses the layers in order, matching one rule per layer by searching by rule priority.

Figure 1 shows a common layer configuration. Following the inbound order, the Virtual Network (VNET) layer provides tunneling for packets coming from Customer Address (CA) space to the Physical Address space (PA). Inbound packets are decapsulated while outbound packets are encapsulated. The next layer is the Ananta [61] NAT load balancer layer, which NATs inbound packets from a Virtual IP (VIP) to a Direct IP (DIP). The Access Control Lists (ACLs) layer is a stateful firewall, while the metering layer is for billing and is placed as the last layer between the VFP and the VM.

While the Host SDN model is used widely in Azure, it is not well-suited for bare-metal workloads. Such workloads are not Hyper-V based, leaving no environment to implement the VFP. Furthermore, the network performance is expected to be on par with deployments in non-virtualized environments, precluding a software-based solution.

2.2 SmartNIC-based SDN

A smartNIC is a programmable network interface card (NIC) that supports the network processing operations usually performed by the host CPU. SmartNICs can be configured for both control plane and data plane operations. They can make use of various technologies depending on the requirements: Application-specific Integrated Circuit (ASIC), System-on-chip (SoC), and Field-programmable Gate Arrays (FPGA).

Compared to VFPs, smartNICs offer lower latency and higher throughput while maintaining the same scalability and programmability level. This allows the host CPU to offload some costly network operations, resulting in reduced processing time and improved power efficiency. However, in some cases, only a subset of tasks is offloaded to the smartNIC, which in itself requires careful orchestration between the hypervisor and the smartNIC operating system.

While smartNICs have proven to be effective overall, they are not a good fit for the bare-metal model given the integration requirements. This is mostly due to the complexities that would be introduced at the hypervisor and network stack of the host. Additionally, the SDN stack implementation should be decoupled from any particular or specialized bare-metal appliance to ensure that a single, general approach will be compatible with all bare-metal services.

2.3 SDN on ToR

Traditionally, data centers are built using fixed-function switches. In such environments, the cloud intelligence resides in the host or hypervisor and not in the network. The host uses smartNICs and VFPs to define a clear separation between the control and data plane.

SDN on ToR refers to the use of programmable switches to execute policy on the ToR switch instead of the host. In order to use a programmable ToR, the target function(s) must be well-defined and limited in scope. A well-targeted SDN ToR application can reduce development time and complexity.

Most, if not all, SDN policies could be supported by programmable switches. However, we have taken an incremental approach where only a small subset of features is initially introduced. This set will be expanded once it has met certain

	End-host software stack per core (§2.1)	**SoC-based smart-NIC (§2.2)**	**ASIC-based smart-NIC (§2.2)**	**FPGA-based smart-NIC (§2.2)**	**Programmable ASIC-based ToR (Bluebird) (§2.3)**
Max Throughput	< 40Gbps & 10's of Mpps	Up to ~100Gbps & ~100Mpps	Up to 200Gbps & 100-200Mpps	Up to 200Gbps & 100-200Mpps	6.4-12.8Tbps & 5-7Bpps
Latency	< 100 μsec	> 1 μsec	> 1 μsec	> 1 μsec	< 1 μsec
Scale	GBs of DRAM + traditional cache hierarchy	8 GBs of DRAM + traditional cache hierarchy	Tens of MBs on chip cache + GBs of DRAM	Tens of MBs on chip BRAM + GBs of DRAM	12 stages of high-throughput pipeline and each pipeline has 24 TCAMs and 80 SRAM blocks
Cost per 100Gbps	Medium (including capex and opportunity cost)	Medium	Medium	Medium	Low
Power consumption per 100Gbps	~500-700W per server including the NIC	~500-700W per server including the NIC	~500-700W per server including the NIC	~500-700W per server including the NIC	~300W for the system that includes 64 ports of 100GbE (~5W/100GbE)

Table 1: Comparison of SDN stack implementations.

standards of quality and reliability. In Bluebird, the primary objective is for the ToR to maintain a high number of CA-to-PA mappings while ensuring hardware-like performance.

An SDN ToR gives us the ability to collapse multiple functions into one network element. In the Bluebird model, we collapse two key roles into a single device: 1) logical network isolation between customers via Virtual Routing and Forwarding (VRF) instances and 2) the association of one or more CA-to-PA mappings per VRF per customer. Implementing these two functions separately on different devices (such as routers implementing VRFs with tunnels to servers running software gateways) incurs the cost of routers and additional servers. By collapsing the routing and CA-to-PA mapping tasks onto a single device, we reduce the number of hops, encapsulations, and consequently latency for bare-metal workloads. Implementing these functionalities directly on the SDN switch results in increased performance and scalability.

To implement an SDN ToR for use with Bluebird, a single VRF is allocated per customer to guarantee logical isolation between customers. Since the goal of the SDN ToR is to connect a customer's bare-metal instance to their VNET, each VRF is programmed with CA-to-PA mappings in the form of VXLAN [48] static routes that associate the bare-metal host to its VNET. Customized P4 programming is used to perform the necessary encapsulation for packets destined to the VNET. This allows the communication between bare-metal (BM) and VMs and between BM and BM to happen at hardware speeds. Additionally, the number of programmable routes is extended through an onboard cache that increases the otherwise limited number of routes the switch ASIC on-chip memory can hold. The route-cache solution is discussed in more detail in §4.

2.4 SDN Servers

The bump-in-the-wire method could have also been approached by assigning dedicated ports on a custom DPDK-enabled SDN server attached to the bare-metal appliance, making use of DPDK-enabled smartNICs on this server to perform the bump-in-the-wire function. However, based on the power consumption data in the Table 1, one can deduce that dedicated servers that carry out bump-in-the-wire operations make the power overhead a non-starter. The performance and scale that an SDN ToR offers in a <500W power envelope makes a strong case for using a dedicated SDN ToR.

3 Design Goals and Rationale

In developing an SDN solution for bare-metal workloads in Azure, we had the following objectives:

1. *Programmability*
   SDN for bare-metal workloads needs to be able to evolve along with the rest of the SDN stack. The VFP [14] model enables many configurable virtual network features. As requirements and policies change over time, SDN for bare-metal should maintain interoperability with the existing stack. This is achieved through the ToRs' programmability which provides control at every stage of packet processing.
2. *Scalability*
   The most significant disadvantage of SDN on ToR compared to host implementations is the limited scale. Memory linearly scales with the number of hosts. Consequently, route capacity can quickly become a bottleneck in resource-limited ToRs. Accordingly, we developed a cache system that extends the hardware capacity of our ToRs and allows us to meet our scalability and performance requirements.
3. *Latency and throughput*
   Bare-metal workloads typically demand high bandwidth, low latency, and deterministic behavior. To meet these requirements, we have used programmable high-speed network ASICs since they offer consistent latency, high throughput, and sustained performance.
4. *High availability*
   To avoid customer impact due to hardware failure or maintenance, the SDN for bare-metal solutions must have high

availability. To support this requirement, we designed redundancy into Bluebird as described in §6.

5. *Multitenancy support*
   Azure supports a large number of customers and tenants who have the ability to create, modify, and delete virtual networks rapidly. When supporting multitenancy, isolation is critical for providing an experience indistinguishable from dedicated networks and servers.
6. *Minimal overhead on host resources*
   With the VFP model, VMs running on the host compete with the SDN stack for hardware resources. The introduction of AccelNet [15] and FPGA-based smartNICs has significantly reduced the overhead, but the VFP still stays on the host to process the first packet in the flow. With bare-metal workloads, customers expect the performance to be similar to that of direct access to the underlying hardware.
7. *Seamless integration*
   Bare-metal workloads run on a wide array of architectures and operating systems. Integrating a new workload in the Azure network should be possible without change to the bare-metal server. Bluebird decouples the workload architecture from the SDN stack and enables consistent virtualization of a diverse set of bare-metal workloads.
8. *External network access*
   Given that bare-metal hosts may require Internet or external network access, a form of Network Address Translation, directly available on the SDN ToR, should be supported.
9. *Interoperability*
   As we introduce programmable ToRs to support bare-metal workloads, these ToRs need to transparently operate with the existing SDN stack to ensure communication between the physical and virtual address space. Interaction with the VFP §4 is of primary importance to realize a heterogeneous system like the one proposed in this paper.

4 System Design

In network system design, there is often a trade-off between the cost of a device, its memory (or route capacity), and features intrinsic to the Network Processing Unit (NPU) or ASIC itself. Internet core routers are generally feature-rich, designed to support large route tables, able to move large amounts of network traffic (>30 Tbps), and usually quite expensive. Alternatively, the ToRs in data centers are cost-effective but have fewer features and support smaller routing tables. In our design, we needed an SDN ToR with a reasonably large VNET routing table but with the cost efficiencies of a typical data-center-class ToR. Bare-metal hosts would connect directly to such SDN ToRs and use a specialized route table to communicate with Azure VMs in a virtualized address space.

Before Bluebird was introduced, Azure supported on-prem bare-metal to VNET connectivity through a software-gateway model. In this model, traffic is encapsulated on a router and forwarded to one or more software-gateways. The software gateways, implemented on standard servers, hold large numbers of CA-to-PA mappings associating an on-prem customer to their VNET. However, with the introduction of the workloads for NetApp, it became clear that the gateway model would not meet the throughput and performance requirements. The NetApp bare-metal service required at least 240Gbps of throughput with a latency ceiling of no more than 4ms. With this in mind, we decided to adopt the SDN ToR model and program the CA-to-PA VNET routes directly onto the ToR, avoiding the software gateway altogether. This improves the throughput as it is now limited only by the throughput of the SDN ToR, which in Bluebird's case is 6.4Tbps.

A considerable effort was put into the planning of how on-chip resources had to be arranged in the pipeline to meet our needs. In particular, we had to decide how to allocate the on-chip switch memory to maximize the number of CA-to-PA mappings which are represented in the VXLAN Tunnel Endpoints (VTEP) resource table. Using Tofino's P4 programmable pipeline, we reduced the IPv4 and IPv6 unicast route table size and significantly increased the VXLAN VTEP table scale from 16K to 192K entries. The ability to support 192K CA-to-PA mappings offered greater flexibility in customer address choices since many specific routes (/32 IPv4 or /128 IPv6 routes) could be used to point to customer VNETs rather than limiting to a smaller number of aggregate routes. In order to make the design future proof, we gave ourselves the maximum allowable table space on the chip in the event that more specific routes were dominant.

The reduction in IPv4 and IPv6 table space also allowed us to extend the VXLAN Network Identifier field space as well as add support for an IPv6 underlay. The custom P4 programmability proved to be extremely valuable in helping us achieve our scale objectives.

While the P4 profile we implemented to give the SDN ToR a large VXLAN VTEP table was adequate when we launched the service, we had to start planning for growth. The VXLAN VTEP route capacity of the ToR was further enhanced with the introduction of a route cache system. The route cache mechanism allowed the VXLAN VTEP table capacity to grow beyond the hardware limit of 192K entries. The caching solution is described later in this section (§4.4). P4 programming flexibility also allowed for other quick packet header manipulations that helped in the rapid development of this service, namely overwriting the inner Ethernet header with the destination VM's MAC address while modifying the VXLAN UDP source port to a custom value.

In the remainder of this section, we discuss the packet flow, related packet transformations, and the control plane design.

4.1 Packet Flow

In this section we provide an in-depth discussion on the packet flow. In order to achieve customer isolation at a logical

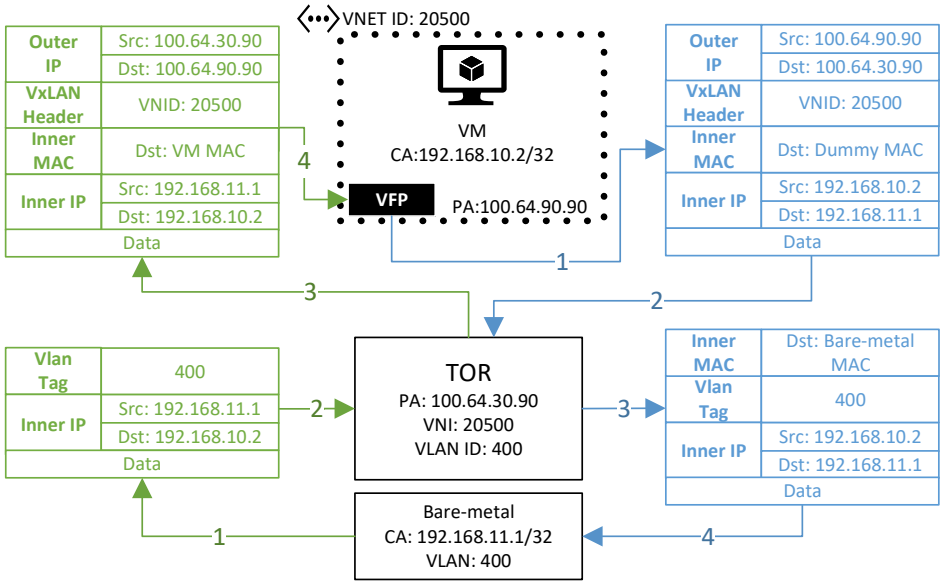

Figure 2: Packet flow between a VM and a bare-metal server.

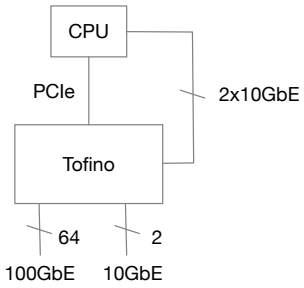

Figure 3: Front panel and CPU interfaces in an Arista 7170.

layer within a common fabric, we identify each customer's VNET by a Virtual Network Identifier (VNI) associated with a unique Virtual Routing and Forwarding (VRF) instance.

Bare-metal to VM. When a bare-metal server sends a packet to an Azure VM, a VLAN tag is added to the outbound packet. The ToR then receives the packet on the virtual interface specified by the VLAN tag. Each interface on the ToR has an associated VRF configured to route the packet to the customer VM. The routes in the VRF associate a VM's IP with the routable IP of the host containing the VM, the VNI of the VNET, and the MAC address of the VM. When the packet reaches the destination host, the VFP decapsulates the packet and uses the MAC to switch the packet to the correct VM. This flow is explained with an example below.

In the green flow shown in Figure 2 at point (1), the bare-metal server sends a packet to the VM through the VLAN 400 interface. The packet is then received on the associated interface on the server's ToR at (2). On the ToR, VLAN 400 is configured to be associated with the VRF 20500, which contains the routes programmed by Bluebird to route the packet to the VM. At (3), the ToR rewrites the inner destination MAC with the VM's MAC contained in the VRF. At (4), the ToR, configured to use the loopback interface as the VXLAN source interface, encapsulates the original frame in a VXLAN frame containing its own PA 100.64.30.90 (loopback IP address) as outer source IP and the VM's host PA 100.64.90.90 as the outer destination IP.

VM to Bare-metal. When an Azure VM sends a packet to a bare-metal server, a Bluebird-provisioned rule instructs the VFP on the host to encapsulate the Ethernet frame in a UDP datagram containing the customer VNI and the IP of the bare-metal's ToR as the destination VTEP IP. The SDN ToR decapsulates the packet and uses the VNI to identify the VRF. Within the VRF, a route lookup is performed to identify the next-hop to which the packet is subsequently be forwarded.

In the blue flow in Figure 2, a VM sends a packet to the bare-metal server where the inner source and destination IPs are respectively set to the VM's CA 192.168.10.2 and the server's CA 192.168.11.1. The destination MAC is set to a dummy value. In the VFP (1) the packet is encapsulated in a VXLAN frame containing the VM's VNI 20500, the outer source IP pointing to the host PA 100.64.90.90, and the PA 100.64.30.90 of bare-metal's ToR as destination IP. At (2), the bare-metal's ToR receives the packet and decapsulates it. The virtual network identifier is used to find the VRF associated with the customer virtual network. At (3), the switch learns the destination MAC through ARP and adds a VLAN tag pointing to the configured VLAN interface 400, and at (4), the packet is routed to the bare-metal server.

4.2 Platform Selection

Using a switch ASIC with a programmable P4 pipeline, we were able to quickly prototype packet formats that would interoperate with Azure's VFP. Additionally, based on the initial requirements, we needed support for at least 192K CA-to-PA mappings. A variety of silicon offerings could meet most of our requirements at the time, but the scale of CA-to-PA mappings pointed our investigation towards (Intel) Barefoot Network's Tofino-1 chipset.

The Tofino-1 is a 6.4Tbps single-chip solution with 12 programmable stages, 256x25/10G SerDes, and a software-defined P4 packet processing pipeline. On the Arista 7170 switch, the Tofino-1 is coupled with a Quad-core 2.2GHz Intel Pentium CPU with additional 2x10GbE ports from the switch ASIC wired directly into the CPU (as shown in Figure 3).

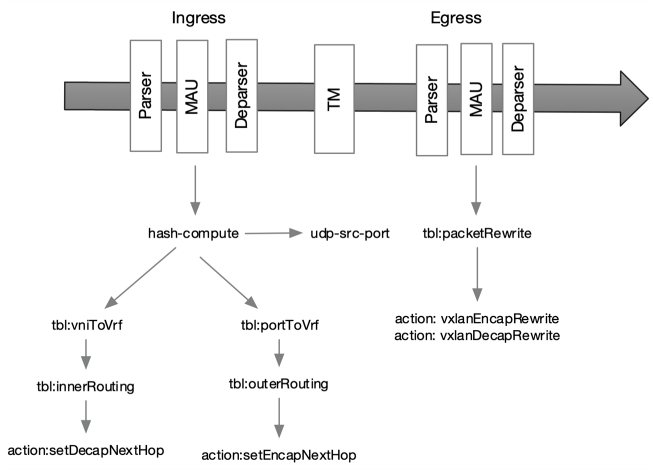

Figure 4: P4 programming pipeline.

These additional ASIC-to-CPU 10GbE ports provide a fast path for route-cached packets to be processed on the CPU.

4.3 P4 Pipeline Design

P4 facilitated rapid prototyping and quick iteration in finding a balance between desirable features and the available chip resources. For instance, while the P4 programming needed to create CA-PA mappings was easy to define, special considerations had to be given to whether the underlay would be IPv4 or IPv6. In a simplistic model, having an IPv6 underlay would significantly reduce the number of CA-to-PA mappings since the entries in the forwarding table would need additional resources to support IPv6 PA destinations. However, with our custom P4 pipeline, we were able to completely decouple the underlay route scale from the overlay route scale thus giving us the maximum number of CA-to-PA mappings independent of whether the underlay used is IPv4 or IPv6.

To describe the packet transformations used in our flows, we used the P4 programmable pipeline on Tofino, which consists of an ingress and egress block (see Figure 4), with each block comprising a sequence of sub-blocks: Parser, Match-Action-Unit (MAU), and Deparser. The Parser block parses the packet and extracts the relevant headers. After that, the MAU performs table lookups and manipulates the packet. The Deparser then reassembles and sends the packet to the Traffic Manager (TM). Finally, packet queuing, replication, and scheduling are done by the TM. A part of the P4 program [1], which illustrates one of the inner-MAC rewrite transformations implemented in our pipeline, is open-sourced.

In the sample P4 program shared [1], the parser excludes regular IP packets and keeps the VXLAN encapsulated packets. After each packet is decapsulated, a route lookup is done on the inner destination IP which determines the action to be taken and the data to be rewritten. In the egress logic, several fields, such as the inner-MAC address, are rewritten based on the bridged metadata received from the ingress pipeline.

```
ip route vrf VNET-A
192.168.10.2/32 vtep 10.100.2.4 vni 20500
router-mac-address 00:12:23:54:A2:9F
```

Figure 5: Static VXLAN route configuration on ToR

While the P4 example provided gives the reader a high-level view of the flexibility available in packet manipulation, the actual P4 code used for this service is more intricate and optimized, allowing for 192K mappings and additional features.

The flexibility that P4 provides allowed us to give the ToR a 'personality' based on the application. A ToR can be preloaded with a custom P4 profile, making the ToR suitable for a given application. For example, we use a 'bare-metal' profile when the ToR is used for Bluebird workloads and a 'NAT-profile' when source NAT is required. Each profile results in a different P4 program getting activated in hardware.

The rest of the packet transformations are presented in detail in the remainder of this section.

Inner Destination MAC Rewrite. When a BM sends traffic to a VM, the receiving hypervisor's VFP uses the inner destination MAC (DMAC) to forward the packet to the destination VM. If the DMAC is unknown, the VFP drops the packet. For this reason, the CA-to-PA routes have the form of static routes extended with additional fields. An extended route, defined on the SDN ToR, contains the VTEP as the PA address, the VNI for the VNET, and the destination MAC of the VM. When a packet matches a route, the ToR overwrites the DMAC with the MAC of the VM. For example, the route in Figure 5 points to a VNET VM with MAC 00:12:23:45:A2:9F at address 192.168.10.2 residing on a VXLAN with VNI 20500 and reachable host (PA) at 10.100.2.4.

Limiting the Range of the VXLAN Source Port. In a traditional VXLAN, the ToR imposes a VXLAN UDP source port value derived from the incoming packet's entropy. The source port is calculated per packet. This is done to help with hash-based ECMP load-balancing schemes employed by network chipsets, ensuring that VXLAN packets are properly load-balanced across the network. To aid the VFP on the VM host in identifying BM-sourced VXLAN packets, we limit the range of source port values usable by the SDN ToR. Limits in imposable ports can be set with a simple CLI command.

Inner Destination MAC Masking. We also needed customization on the SDN ToR to ignore the inner destination MAC address arriving in the VXLAN packet. This is in the direction of the VNET to the SDN ToR, where the inner destination MAC address is usually resolved by an ARP exchange between two VXLAN hosts. Specifically, an ARP request/reply exchange would have to take place, ensuring that the end-hosts are aware of each other's MAC addresses. The entire ARP resolution step can be skipped if the SDN ToR absorbs all packets regardless of the DMAC value, as described in packet flow in §4.1. The VFP transmitter on the VNET simply writes a bogus inner DMAC value in the

packet destined to the SDN ToR. The SDN ToR is made to ignore this bogus MAC and proceeds to route all the received frames regardless. The SDN ToR accomplishes this task using a wildcard bitmask applied using a CLI command.

4.4 Route Cache

Although we were able to make room for additional mappings using a custom P4 pipeline, we were faced with the challenge of increasing the scale beyond what the chip hardware could support. At this point, we had a working P4 pipeline that was programmed to fit 192K CA-PA mappings, support an IPv6 underlay, and offer 1:1 static NAT to BM hosts.

As the number of bare-metal customers grew, we realized that the 192K CA-PA upper-limit of the Tofino would soon become a bottleneck. We considered a few alternatives, one of which was to quickly onboard the next-generation Tofino (Tofino-2), which was known to support up to 1.5M CA-PA mappings in hardware. However, we ended up pursuing the route cache feature since it would be valuable regardless of the scale of the underlying Tofino ASIC. Route caching gives us a five-fold increase over the original 192K mappings. The route cache feature is an implementation of the familiar statistical multiplexing model whereby a significantly higher number of customers can share a finite resource, as long as not all customers are active at the same time. In other words, if we know that our customers are not always using all available hardware entries, we can reassign those unused entries to other active users. A primary enabler for the route cache concept is the way Bluebird provisions SDN entries. Bluebird does not preconfigure the SDN ToR but instead dynamically provisions mappings as needed allowing the route cache logic to continuously determine which entries are to remain in hardware or moved to software.

Before describing the route cache feature, it is important first to understand the Software Forwarding Engine (SFE). The SFE is a DPDK-enabled packet processing function provided by the ToR's CPU which has a packet forwarding rate of 200K pps. CPU bound packets use the 2x10GbE interfaces connecting the Tofino to the CPU to reach the SFE. A software agent on the ToR monitors hardware entries on the switch ASIC and moves inactive entries to the SFE. SFE resources are CPU and memory bound. With 16GB of memory, up to 600K mappings can be stored in the SFE, whereas 1.5M mappings can be stored with at least 32GB of memory. The number of mappings compared to the total memory is low because the memory also supports the switch operating system, i.e., EOS (Extensible Operating System). The portion of memory used by the SFE is relatively small: about 3GB out of the available 32GB total memory is used to store mappings. The rest of the memory is used for running the operating system and other agents such as the routing agent, platform agent, etc.

Bluebird configures static VXLAN routes/mappings and specifies whether the route/mapping is cacheable or not. The mapping itself is programmed by Bluebird using a JSON RPC call as described later in this section. A mapping is identified as active if there was a packet that recently used the prefix programmed in the VNET route. We will use the concept of a 'hitbit' to note when a route entry is touched by a packet, either in software or hardware. EOS then maintains this hitbit for all hardware and software (SFE) entries. The hardware hitbit is triggered when mappings are used for hardware-based forwarding. A SFE hitbit is triggered when a packet takes the software DPDK path, indicating that this software mapping now needs to be upgraded from the SFE to hardware.

Threshold level	Utilization	Idle time
low	85%	1100s
medium	90%	300s
high	95%	100s

Table 2: Default caching thresholds.

To select the mappings that are evicted from the hardware, we use a Least Recently Used (LRU) eviction algorithm. LRU entries are found by polling the hitbit property maintained in the hardware for each prefix. The flows going through the SDN ToR are monitored using an age-based idle timer. No packet state is maintained, nor are the packets examined. We considered other hardware eviction algorithms but ultimately rejected them. These included 1) tracking TCP flows and 2) tracking flows that carried the most traffic volume.

These options were rejected because tracking TCP flows is expensive from two perspectives: 1) the need to send packets containing TCP flags S, F, R to an agent on the switch for tracking purposes, and 2) flows are expensive to store as the access key would comprise of the source IP/port and destination IP/port. For routing purposes, the switch only needs the destination IP and the added flow state becomes unnecessary overhead.

We decided to implement the idle-timer based approach since it was simpler and it met our requirements. The idle timer based approach is the simplest and most efficient because it does not require tracking individual flows or state. In the future, other algorithms may be considered as we learn more about customer traffic patterns.

In the CLI, we can specify CA-PA mapping entries as cache candidates. All candidate mappings become a part of the route cache, which means that these prefixes can be downgraded to software if they become inactive and can be upgraded to hardware if they become active. The aging time of hardware routes and how many of these entries remain in hardware can be configured as a percentage of total hardware capacity. Finally, the default threshold values are listed in Table 2.

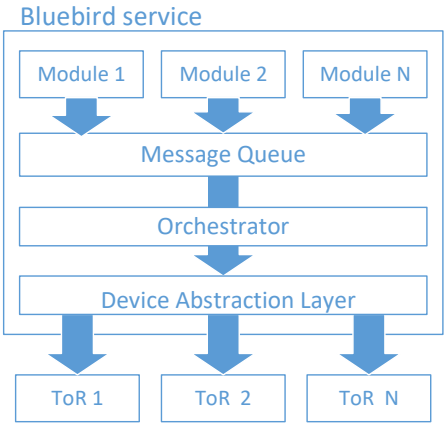

Figure 6: Bluebird service structure.

4.5 Control Plane and Policy Provisioning

In the Azure host SDN, a controller running on each host programs the VFP. However, since this is no longer an option for the bare-metal workloads, two alternatives for hosting SDN logic were considered; an agent on the ToR or programming the ToR via an external service. We rejected the idea of having an agent on the ToR since it did not meet our performance goals. The agent would compete for resources with latency-sensitive operations running on the ToR, and resource consumption could increase as requirements changed over time. On the other hand, an external service does not need immediate proximity to the ToR because configuration on a ToR has less stringent latency requirements than typical data plane operations. Moreover, a separate service also has advantages in terms of fault tolerance and deployment time. As a result, we introduced the Bluebird Service (BBS) (Figure 6).

Provisioning SDN Policy on ToR

BBS is a lightweight, multi-tenant, stateless microservice that configures the ToRs with virtual network policies for each customer (Figure 6). Each policy is represented as a JSON object that specifies a ToR's desired state, called the goal-state. Azure SDN services send their goal-states to BBS' message queue leading to the orchestrator module which parses and consolidates the goal-states into ToR configurations. Configurations are then transferred to the device abstraction layer (DAL), which keeps the SDN business logic independent from ToR implementations. In the DAL, they undergo a conversion process that results in a sequence of commands for the targeted ToR. The list of allowed commands is strictly limited to the operations needed for bare-metal provisioning, reducing the possibility of interference with other automation systems responsible for software upgrading or traffic shifting. In the case of the Arista 7170, BBS uses the JSON-RPC 2.0 protocol over HTTPS. Each JSON payload contains an ordered list of commands using Arista EOS CLI syntax.

BBS performs a sync-check on all ToR configurations at defined intervals. At every sync, it calculates the delta between a ToR's configuration and its target configuration and performs a reconciliation in case of differences. Each configuration request is atomic, and configurations are versioned to avoid inconsistent states due to out-of-order execution. BBS also ensures state consistency between multiple ToRs if they are part of a high availability network configuration (see §6.1) in which they are seen as one logical entity.

To prevent resource exhaustion due to extremely large VNETs generating a high number of routes, BBS limits the number of programmable routes per VNET. The default limit is maintained as a function of the ToR's capacity. When more routes are needed, the limit can be raised to match the requirements and, in some cases, customers are migrated to dedicated ToRs.

Azure regions protect from failures through increased redundancy. Each region is subdivided into several distinct physical locations called availability zones (AZs). Each zone is made up of one or more data centers (DCs) equipped with independent power, cooling, and networking. BBS is deployed per AZ within an Azure Service Fabric [33] ring organized as a series of active and inactive instances. Additionally, BBS's scope is not limited to a single AZ and can target any AZ within the same region. However, this is limited only to scenarios in which another AZ is severely impacted and the local BBS is unable to function.

5 Performance

Over the past two years, Azure has been deploying bare-metal services on SDN-ToRs in over 42 data centers. In addition, Azure has successfully onboarded several HWaaS native applications such as Cray ClusterStor, and NetApp Files [13,53]. As of today, we have powered several thousands of bare-metal servers and serve thousands of terabytes of traffic per day. Although a significant number of customers are adopting these bare-metal services, the number of routes tied to these workloads has yet to grow to the point of exceeding the route cache threshold of 85% capacity which would trigger the use of route cache. We estimate that the threshold will be exceeded within a year and believe that the route cache feature will play an important role in the future of the bare-metal service offering as the number of provisioned VNET routes outpaces the growth in hardware capacity of SDN ToRs.

We have compared the performance of bare-metal servers with VMs using Azure accelerated networking, both running in an Azure data center on Intel Xeon E5-2673 v4 (Broadwell at 2.3 GHz) CPUs with 40Gbps NICs and Windows Server 2019. We measured throughput, CPU overhead, and latency, with both solutions performing similarly and with no appreciable difference.

This section presents a performance analysis specifically focusing on route cache that is carried out through the use of synthetic testing tools and production data where available.

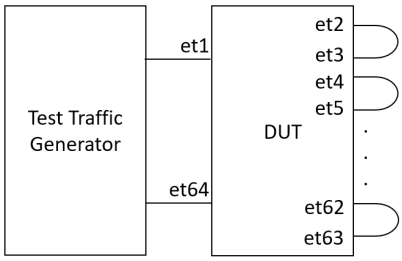

Figure 7: Test topology.

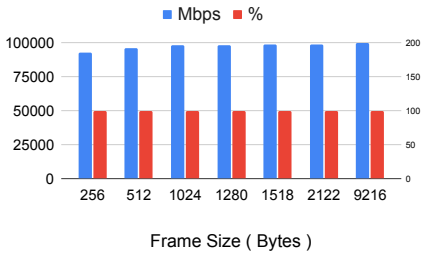

Figure 8: Throughput in Mbps on the left y-axis and in % on the right y-axis per frame size.

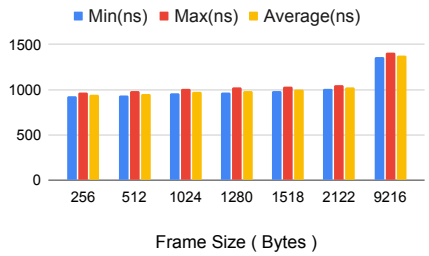

Figure 9: Latency measurements in nanoseconds (y-axis) per frame size (x-axis).

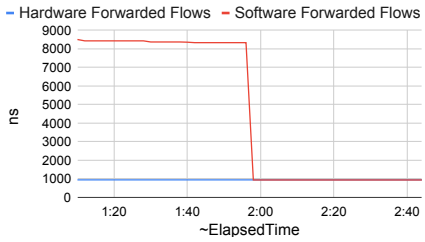

Figure 10: Latency comparison between software forwarded and hardware forwarded flows in nanoseconds per frame.

5.1 Hardware Performance

To measure the port-to-port latency within a SDN ToR, we connected a traffic generator directly to the Device Under Test (DUT). Figure 8 shows that the Intel Tofino ASIC performs at wire-speed with a consistent port-to-port latency of <1μs. The Tofino chip demonstrated deterministic performance across all cases of packet manipulation required for Bluebird. Additionally, even during heavy traffic load, minimal differences were observed between the min and max throughput values. Goodput was tested by passing L2 frames of various size through a DUT in a snake topology (Figure 7).

In Figure 8, 100% goodput is sustained across all the tested packet sizes up to 9216 bytes, with the only expected exception for packets of 256 bytes. This is due to the smaller relative size difference between header and payload. This performance is in line with our requirements to support bare-metal workloads that are bandwidth and latency-sensitive.

From a power utilization perspective, although the ToR behaves like a bump in the wire, it is not adding any power draw over a regular data center design. The typical/max power draw of BlueBird ToRs is 271W/571W. This is in the same range as other ToRs used in Azure with the same bandwidth (64x100G) which have a typical/max power draw of 314W/616W.

5.2 Performance Impact of Route Caching

The route cache feature is responsible for moving route entries from the SFE to the hardware and vice versa while ensuring the process is transparent to the customer. The CPU on the ToR provides a DPDK-enabled packet processing function, helping to meet our stringent latency requirements. As mentioned earlier, we use 2x10G high-bandwidth links between the CPU and the ASIC. Since the traffic consists exclusively of TCP flows, our system has enough time to move the entries from the SFE to hardware by the time the TCP 3-way handshake is complete.

There are two primary factors that contribute to the observed latency when route-caching is performed; 1) the latency experienced while packets are being forwarded in software by the SFE and 2) the time taken to move a route entry from the SFE to the hardware.

To accurately measure the latency experienced by packets forwarded in software, we used instrumented packets that were generated by a traffic analyzer and forcefully forwarded them via the SFE. This was done by defining 5000 routes on the SDN ToR and sending traffic to each one of these entries while deliberately preventing the entries from being programmed into the hardware ASIC. To ensure that these software entries would not get programmed in the hardware, we temporarily disabled the route cache feature after the route entries were activated in the SFE. This approach ensured that the route entries in the SFE remained in software while we recorded latency measurements. In this state, where the routes were present only in the SFE, we found that packets experienced an increased latency of about 8μs when compared to the latency experienced by packets using hardware entries.

Under normal circumstances, and assuming the route-entry used is in the SFE, we expect the packets to be software-routed for only a short time. The first packet that triggers a hitbit recording immediately kicks off the hardware programming for the SFE route. While this transition is hard to measure using generic traffic analyzers due to their lack of precision, we were able to inspect system logs on the SDN ToR to

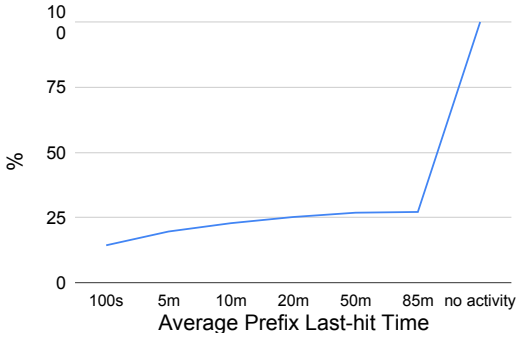

Figure 11: Prefix utilization based on last-hit time.

learn that this transition time is <2ms. To measure this, we compared the logged timestamps between when the packet hit the SFE and when the route appeared in hardware.

5.3 Validating Route Cache

The premise of designing and implementing the route cache model was based on a theoretical assumption that not all bare-metal customers would require hardware table entries at the same time. In order to prove that route caching would work, we had to find a way to record the age, or the last-time a given hardware entry was used. In other words, if an entry had aged and it's recorded "last time used" was old, then that entry could be demoted and moved to the SFE. Furthermore, we created the following time-buckets for age categories; 0s-100s, 101s-5m, 5m-10m, 10m-20m, 20m-50m, 50m-85m and a 'no activity' bucket for all prefixes that have not been touched, in hardware, for over 85 minutes. Grouping the prefix counts by last-time-used gives us further flexibility in tuning how aggressively we want to move entries to the SFE.

After weeks of data gathering from production SDN ToRs upgraded with the route cache feature, the results were inline with our expectations. Figure 11 shows the hardware table utilization across the route-entry last-hit time bins for the entire SDN ToR fleet. We see that 10% of all the prefixes in the hardware were utilized or 'hit' in the last 100 seconds.

Bluebird does implement an inherent artificial hardware provisioning constraint, to mostly protect against complete hardware table exhaustion on the SDN ToR. This protection was put in place to ensure that the 192K entries are never consumed. However, what we learned was that the hardware table utilization is in fact only 50% utilized and no more than 20-25% of the prefixes are active at any given time (Figure 11). This means that that a majority of the prefixes (75-80%) that are in hardware can in fact be moved to the SFE.

Armed with this data, we can now remove the Bluebird provisioning constraints and allow for provisioning of more than 192K entries knowing that 75% of the prefixes will most likely reside in the SFE.

Based on Figure 11, we can conclude that entries categorized under 'no activity' are good candidates to migrate to the SFE leaving newly-vacated hardware entries available to other customers. Since only 20-25% of the entries are used at any given time, route caching allows us to increase our scale by 4-5x.

6 Operationalization and Experiences

Bluebird has now been deployed at scale in multiple data centers for various bare-metal workloads. The service has brought together high-throughput and low-latency bare-metal offerings to existing cloud customers without compromising scale or reliability. Bluebird has accomplished all the goals that we set out in §3. In order to make Bluebird successful, we adopted well-known operational models including continuous integration for both service delivery and feature development, ensuring redundancy in all aspects, planned failure for maintenance purposes, and incorporating monitoring and alerting.

During the feature development phase we used a P4 emulator [2] to simulate the entire pipeline in software which gave us a glimpse into the complete lifecycle of a packet. Having this flexibility removed the need for hardware at every stage of the development cycle. The software tools helped implement all the SDN ToR features, simulated the hardware, and allowed for rapid prototyping without any of the usual and costly hardware resource dependencies. Furthermore, P4 provided the flexibility of software with hardware-level performance. For example, routing look-up decisions would generally occur at a certain, fixed point in an ASIC's pipeline. In the case of P4, we had the flexibility of doing a routing look-up after the parser stage or in the MAU, giving us the luxury of normalizing the contents of an incoming packet and acting on any portion of the inner or outer IP header (see Figure 4). It was this flexibility in P4 that also allowed us to limit the UDP source-port values described earlier.

For workloads where data-plane redundancy was required, we paired two SDN ToRs using Multi-chassis Link Aggregation Group (MLAG) (Figure 12). While MLAG seemed like a natural choice for first-hop redundancy at layer-2, we did not want BBS to be concerned with the details of MLAG itself or make MLAG a design requirement moving forward. Hence, we created a common anycast loopback IP to represent both members within a redundant pair of SDN ToRs. BBS configures each member like any other SDN-ToR using a shared anycast loopback IP address representing the SDN ToR's physical address. If traffic were to arrive on a member of the SDN ToR pair with failed links to the bare-metal server, MLAG would locally switch the packets to the neighboring SDN ToR with active links to the bare-metal host.

While the P4 emulator tools helped with the development of the software features running on the SDN ToR, a different software emulator was used to help operationalize the SDN ToR in the network. A Docker container emulating a complete SDN ToR was used to speed up the management plane bring-up between Bluebird and the SDN ToR. Since the

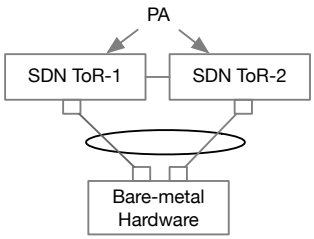

Figure 12: Bare-metal redundancy.

container version of the SDN ToR had all the properties of the hardware based SDN ToR (minus the hardware), the Bluebird management plane was developed and tested against this container. The Docker container was also useful when testing new customer scenarios before taking them to production.

For monitoring and alerting, we built our system to gather numerous metrics per ToR, BM server, and BBS. These metrics are collected in a centralized monitoring system, combined, and further transformed to create alerts. The actions based on the collected metrics differ based on the conditions or events configured. The metrics and alerts are also frequently added and updated as more experience is gained from production. Additionally, they are aggregated in customizable dashboards used to drive decision-making.

6.1 Lessons Learned

These are some of the lessons learned since the release of Bluebird:

- **Data-plane packet mirroring for debugging:** Having the ability to inspect data plane traffic during troubleshooting proved helpful. All data plane packets can be mirrored to the ToR CPU to inspect traffic entering or leaving the SDN ToR. This provides an additional layer of visibility to all the interface and protocol-level counters that are available on the switch. We did not expect this feature would be used as often as it has for debugging issues in production.
- **ASIC with re-configurable programming pipeline:** The Tofino ASIC, with its re-configurable pipeline, allowed us to develop features like route cache even after Bluebird was deployed. The VXLAN source port offset feature, described in §4.3, was possible because of the malleability of the pipeline. Making the decision to use a P4 ASIC proved to be useful as it allowed us to develop all the features that would otherwise have not been possible.
- **ASIC emulators can be used to speed up software development on ToR:** We used a P4 emulator during the ToR packet pipeline development process. Because of this, the development team did not have to wait for the hardware to be available. Moreover, the end-to-end packet flow could be tested by generating actual data-plane packets. This enabled us to test smaller parts of the P4 code without having an end-to-end pipeline ready.
- **Virtualized ToR image for control-plane testing:** Since the availability of the hardware ToR switch for lab testing and development is limited, having a VM image of the ToR was extremely useful to test the route programming service along with the control-plane interaction.
- **Need for 64-bit OS:** With a 32-bit OS, only 4GB of virtual memory could be addressed giving us 600k route cache entries. Hence, moving to a 64-bit OS and increasing the amount of RAM on the SDN-ToR was required to support 1.5M route-cache entries.
- **Limited control-plane vocabulary:** We limited the scope of commands that BBS was allowed to execute on the SDN ToR. The commands issued by BBS are strictly related to adding/deleting customer VRFs and mappings. This limits the damage that can be caused by bad actors and avoids interference with the rest of the automation framework. All other provisioning, maintenance, and monitoring functions are performed by the larger automation framework.
- **Software coordination at scale:** BBS runs on a server-class machine that is far more capable than the comparatively underpowered ToR's hardware. This difference is also reflected in the number of concurrent connections possible, which, if left unchecked, can impact programming time due to BBS exceeding the ToR connection limit. Consequently, requests from BBS are sent to a queue and batched to keep the number of connections within the limits of the switch.
- **Customer traffic should be agnostic of ToR availability:** MLAG helped us abstract out whether a ToR was in maintenance or not. This made the BBS less disruptive to deploy and maintain for redundant workloads.
- **Reconciliation is necessary:** Restoring an outdated ToR configuration can lead to failures and programming conflicts. Reconciliation is necessary to ensure that errors introduced by outdated configurations are repaired. A reconciliation process running after a configuration is restored guarantees eventual consistency and transient conflicts. For instance, CA-to-PA mappings received from upstream are compared with the ToR's current configuration, and stale mappings are removed in the process. Reconciliation is also performed against BBS' internal cache and state. Restoring a state after a service fail-over is also a source of incoherent configurations. Missed notifications during downtime are, in fact, a common occurrence in a live service.
- **Artificial limits can cause overheads:** During analysis of scale requirements from production data on the size of VNETs, we concluded that we had to limit the per-customer mappings until the route cache feature was enabled. As a result, an upper limit was put to stop one customer from monopolizing an entire ToR. This artificial limit soon became an operational overhead since we had to increase it per customer on an on-demand basis and in many cases the new limit was barely above the imposed value.
- **State to reduce reconciliation time:** BBS initially was

developed as a fully stateless service. After each restart, the switch configuration would simply be reconstructed through data received from upstream and downstream components. However, this significantly increased the rehydration and reconciliation time. Eventually, we moved to a stateful model to reduce the time consuming interactions with the other components. In the new model, BBS maintains a versioned representation of the switch configuration and communication is established only when strictly necessary.

- **For bug fixes, prefer a new image over a patch:** During the initial deployment stages, for quick bug fixes, we deployed bug-fix patches on top of the ToR OS image. But as the number of ToRs grew, it became cumbersome to deploy patches throughout the fleet of devices and keep a track of them. So, we decided to have bug fixes in new images only. Now we upgrade the devices more often but in a manner that is easier to track. This decision has helped us improve the quality of the code overall.
- **Unmodified Linux kernel:** The ToR OS uses an unmodified Linux kernel. Because of this, we were able to use open source tools like tcpdump, iperf, etc. for debugging without any issues. Also, we are able to run Docker containers on top of the ToR OS for SSH user certificate rotation.

7 Related Work

A practical implementation of Bluebird relies on the ability to enable custom and dynamic SDN policies in a ToR, enabled by recent work in programmable switches [4, 5]. As discussed in §2, many other forms of hardware can be used to implement the SDN stack including smartNICs [58–60] and servers running any one of a variety of software network processing systems such as [17, 25, 41, 62]. While there is a vast array of prior work in this space, the state-of-the-art software solutions are not able to meet the throughput of a programmable switch and require far more power. Work in network function virtualization [20,43,49,54,66,68,70] shows that these software-based approaches can be feasible at scale, though they do not meet our stringent requirements. Similarly, smartNICs have been used to offload various custom network operations [15, 44]. However, the bare-metal model precludes these options as they reside at the host. One may also adopt a hybrid approach, leveraging commodity switches and software [3, 18], but again the power consumption of a server is higher than a ToR switch.

Programmable switches have been used for a wide variety of other applications such as caching [32, 45, 47], telemetry [24, 57], consensus [11, 31], machine learning [63], and various network functions [37,39,52]. Despite the well-known and strict resource constraints in programmable switches [65], these systems demonstrate that non-trivial computations can be done in the network at line rate on these devices.

Bluebird leverages this speed while overcoming the state limitations of Tofino switches by using the switch CPU and memory for cached flows. As the scale of the necessary state continues to grow, upgrading to the Tofino-2 [28] or adopting a switch memory extension may be helpful [40]. With increased traffic engineering and the rise of SDN, the limits of in-switch memory have become a noticeable issue prompting investigations into the practicality of caching for traditional routes [38, 46] and flow policies [9, 36]. We do not have to implement the complex dependency logic of [36] since the CA-PA mappings are non-overlapping. Similarly, [38, 46] and [9] capitalize on relationships between entries in a FIB or openflow table [51] which are not present in Bluebird's in-switch data. Other SDN rule distribution techniques [34, 35, 55] do not apply to the Bluebird design as there is only one switch on route to implement the necessarily policies.

Since the early and largely academic SDN designs [7,19,22, 23, 42, 51], hyperscale cloud networks have adopted SDN to virtualize and isolate tenant networks [10,12,14,21,30,56,61], implementing at various layers of the stack. Regardless of multi-tenancy, SDN is used to operate large single-tenant networks with high-level intent and to implement arbitrary traffic engineering [8, 26, 29, 64, 67, 69, 71]. Bluebird is a new addition to our SDN deployment accommodating the unique requirements of baremetal customers: the servers must be connected to virtual networks, but we cannot enforce any SDN policies at the host in an approach such as [50]. This encourages new in-ToR support so that the additional costs of running the necessary network functions on a server similarly to [50] can be avoided.

8 Conclusions and Future Work

We have presented our experiences designing, implementing, and deploying Bluebird, a high-performance network virtualization system for bare-metal cloud services on Azure. Bluebird has been running in Azure data centers for more than two years and has served demanding workloads like those for Netapp, Cray, and SAP.

The abstraction layer in Bluebird's control plane allows us to handle different switches with minimal change. By using high-performance programmable ASICs, we rearranged ToR's resources to increase route capacity. On top of that, we have implemented a cache system that extends the capacity even further while incurring a negligible performance penalty. The support for additional routes has allowed us to improve performance by removing software gateways. Lastly, our design decouples the SDN stack from the bare-metal services and facilitates the introduction of new and diverse workloads.

In the future, as we learn more about customer traffic, we will explore ways to improve the cache system, such as by considering a different eviction algorithm.

References

[1] https://github.com/aristanetworks/p4-vxlanencapdecap/blob/main/switch-vxlan.p4.

[2] P4 behavioral model, 2021. https://github.com/p4lang/behavioral-model.

[3] Mina Tahmasbi Arashloo, Pavel Shirshov, Rohan Gandhi, Guohan Lu, Lihua Yuan, and Jennifer Rexford. A scalable VPN gateway for multi-tenant cloud services. *SIGCOMM Comput. Commun. Rev.*, 48(1):49–55, April 2018.

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[5] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 99–110. Association for Computing Machinery, 2013.

[6] Broadcom. Trident SmartToR, 2021. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/smarttor.

[7] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, page 1–12. Association for Computing Machinery, 2007.

[8] Sean Choi, Boris Burkov, Alex Eckert, Tian Fang, Saman Kazemkhani, Rob Sherwood, Ying Zhang, and Hongyi Zeng. FBOSS: Building switch software at scale. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 342–356. Association for Computing Machinery, 2018.

[9] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. DevoFlow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 254–265. Association for Computing Machinery, 2011.

[10] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 373–387, Renton, WA, April 2018. USENIX Association.

[11] Huynh Tu Dang, Pietro Bressana, Han Wang, Ki Suh Lee, Noa Zilberman, Hakim Weatherspoon, Marco Canini, Fernando Pedone, and Robert Soulé. P4xos: Consensus as a network service. *IEEE/ACM Trans. Netw.*, 28(4):1726–1738, August 2020.

[12] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 523–535, Santa Clara, CA, March 2016. USENIX Association.

[13] Hewlett Packard Enterprise. Cray clusterstor e1000 storage systems, 2021. https://buy.hpe.com/us/en/enterprise-solutions/storage-solutions/cray-clusterstor-storage-systems/cray-clusterstor-e1000-storage-systems/cray-clusterstor-e1000-storage-systems/p/1012842049.

[14] Daniel Firestone. VFP: A virtual switch platform for host SDN in the public cloud. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 315–328, Boston, MA, March 2017. USENIX Association.

[15] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: SmartNICs in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, Renton, WA, April 2018. USENIX Association.

[16] Linux Foundation. Open vSwitch, 2016. https://www.nvidia.com/en-us/networking/ethernet/connectx-5/.

[17] Linux Foundation. Data plane development kit (DPDK), 2021. http://www.dpdk.org.

[18] Rohan Gandhi, Hongqiang Harry Liu, Y. Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. Duet: Cloud scale load balancing with hardware and software. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 27–38. Association for Computing Machinery, 2014.

[19] Yashar Ganjali and Amin Tootoonchian. HyperFlow: A distributed control plane for OpenFlow. In *2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN 10)*, San Jose, CA, April 2010. USENIX Association.

[20] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. OpenNF: Enabling innovation in network function control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 163–174. Association for Computing Machinery, 2014.

[21] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A scalable and flexible data center network. *Commun. ACM*, 54(3):95–104, March 2011.

[22] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4D approach to network control and management. 35(5):41–54, October 2005.

[23] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[24] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 357–371. Association for Computing Machinery, 2018.

[25] Sangjin Han, Scott Marshall, Byung-Gon Chun, and Sylvia Ratnasamy. MegaPipe: A new programming interface for scalable network I/O. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, page 135–148, USA, 2012. USENIX Association.

[26] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 74–87. Association for Computing Machinery, 2018.

[27] Intel. Intel Tofino, 2021. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.

[28] Intel. Intel Tofino 2, 2021. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html.

[29] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 3–14. Association for Computing Machinery, 2013.

[30] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, Albert Greenberg, and Changhoon Kim. EyeQ: Practical network performance isolation at the edge. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 297–311, Lombard, IL, April 2013. USENIX Association.

[31] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. NetChain: Scale-free sub-RTT coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 35–49, Renton, WA, April 2018. USENIX Association.

[32] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. NetCache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 121–136. Association for Computing Machinery, 2017.

[33] Gopal Kakivaya, Lu Xun, Richard Hasha, Shegufta Bakht Ahsan, Todd Pfleiger, Rishi Sinha, Anurag Gupta, Mihail Tarta, Mark Fussell, Vipul Modi, Mansoor Mohsin, Ray Kong, Anmol Ahuja, Oana Platon, Alex Wun, Matthew Snider, Chacko Daniel, Dan Mastrian, Yang Li, Aprameya Rao, Vaishnav Kidambi, Randy Wang, Abhishek Ram, Sumukh Shivaprakash, Rajeet Nair, Alan Warwick, Bharat S. Narasimman, Meng Lin, Jeffrey Chen, Abhay Balkrishna Mhatre, Preetha Subbarayalu, Mert Coskun, and Indranil Gupta. Service fabric: A distributed platform for building microservices in the cloud. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18. Association for Computing Machinery, 2018.

[34] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. Optimizing the one big switch abstraction in software-defined networks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, page 13–24. Association for Computing Machinery, 2013.

[35] Y. Kanizo, D. Hay, and I. Keslassy. Palette: Distributing tables in software-defined networks. In *2013 Proceedings IEEE INFOCOM*, pages 545–549, 2013.

[36] Naga Katta, Omid Alipourfard, Jennifer Rexford, and David Walker. CacheFlow: Dependency-aware rule-caching for software-defined networks. In *Proceedings of the Symposium on SDN Research*, SOSR '16. Association for Computing Machinery, 2016.

[37] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. HULA: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '16. Association for Computing Machinery, 2016.

[38] Changhoon Kim, Matthew Caesar, Alexandre Gerber, and Jennifer Rexford. Revisiting route caching: The world should be flat. In *Proceedings of the 10th International Conference on Passive and Active Network Measurement*, PAM '09, page 3–12, Berlin, Heidelberg, 2009. Springer-Verlag.

[39] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. TEA: Enabling state-intensive network functions on programmable switches. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 90–106. Association for Computing Machinery, 2020.

[40] Daehyeok Kim, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, and Srinivasan Seshan. Generic external memory for switch data planes. In *Proceedings of the 17th ACM Workshop on Hot*

Topics in Networks, HotNets '18, page 1–7. Association for Computing Machinery, 2018.

[41] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, August 2000.

[42] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, page 351–364, USA, 2010. USENIX Association.

[43] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. Embark: Securely outsourcing middleboxes to the cloud. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 255–273, Santa Clara, CA, March 2016. USENIX Association.

[44] Bojie Li, Kun Tan, Layong (Larry) Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. ClickNP: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 1–14. Association for Computing Machinery, 2016.

[45] Ming Liu, Liang Luo, Jacob Nelson, Luis Ceze, Arvind Krishnamurthy, and Kishore Atreya. Incbricks: Toward in-network computation with an in-network cache. *SIGARCH Comput. Archit. News*, 45(1):795–809, April 2017.

[46] Yaoqing Liu, Syed Obaid Amin, and Lan Wang. Efficient FIB caching using minimal non-overlapping prefixes. *SIGCOMM Comput. Commun. Rev.*, 43(1):14–21, January 2013.

[47] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. DistCache: Provable load balancing for large-scale storage systems with distributed caching. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 143–157, Boston, MA, February 2019. USENIX Association.

[48] Mallik Mahalingam, Dinesh Dutt, Kenneth Duda, Puneet Agarwal, Larry Kreeger, T. Sridhar, Mike Bursell, and Chris Wright. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348, August 2014.

[49] Joao Martins, Mohamed Ahmed, Costin Raiciu, and Felipe Huici. Enabling fast, dynamic network processing with ClickOS. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, page 67–72. Association for Computing Machinery, 2013.

[50] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. Snap: A microkernel approach to host networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 399–413. Association for Computing Machinery, 2019.

[51] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[52] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 15–28. Association for Computing Machinery, 2017.

[53] Microsoft. Azure NetApp files, 2021. https://azure.microsoft.com/en-us/services/netapp/#overview.

[54] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262, 2016.

[55] Masoud Moshref, Minlan Yu, Abhishek Sharma, and Ramesh Govindan. Scalable rule management for data centers. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 157–170, Lombard, IL, April 2013. USENIX Association.

[56] Jayaram Mudigonda, Praveen Yalagandula, Jeff Mogul, Bryan Stiekes, and Yanick Pouffary. NetLord: A scalable multi-tenant network architecture for virtualized datacenters. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 62–73. Association for Computing Machinery, 2011.

[57] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 85–98. Association for Computing Machinery, 2017.

[58] NVIDIA. Connectx-5, 2021. https://www.nvidia.com/en-us/networking/ethernet/connectx-5/.

[59] NVIDIA. Data processing units, 2021. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[60] NVIDIA. Innova-2 flex, 2021. https://www.nvidia.com/en-us/networking/ethernet/innova-2-flex/.

[61] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A. Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, Changhoon Kim, and Naveen Karri. Ananta: Cloud scale load balancing. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 207–218. Association for Computing Machinery, 2013.

[62] Luigi Rizzo and Matteo Landi. Netmap: Memory mapped access to network devices. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 422–423. Association for Computing Machinery, 2011.

[63] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan R. K. Ports, and Peter Richtárik. Scaling

distributed machine learning with in-network aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, April 2021.

[64] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 418–431. Association for Computing Machinery, 2017.

[65] Naveen Kr. Sharma, Antoine Kaufmann, Thomas Anderson, Arvind Krishnamurthy, Jacob Nelson, and Simon Peter. Evaluating the power of flexible packet processing for network resource allocation. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 67–82, Boston, MA, March 2017. USENIX Association.

[66] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. *SIGCOMM Comput. Commun. Rev.*, 42(4):13–24, August 2012.

[67] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Hong Liu, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network. *Commun. ACM*, 59(9):88–97, August 2016.

[68] Arjun Singhvi, Junaid Khalid, Aditya Akella, and Sujata Banerjee. SNF: Serverless network functions. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, SoCC '20, page 296–310. Association for Computing Machinery, 2020.

[69] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky H.Y. Wong, and Hongyi Zeng. Robotron: Top-down network management at Facebook scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 426–439. Association for Computing Machinery, 2016.

[70] Richard Wang, Dana Butnariu, and Jennifer Rexford. OpenFlow-based server load balancing gone wild. In *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'11, page 12, USA, 2011. USENIX Association.

[71] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. Taking the edge off with Espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 432–445. Association for Computing Machinery, 2017.