Kubernetes Load-balancing and related network functions using P4

Nupur Jain
Intel, CA USA,
nupur.jain@intel.com

Vinoth Kumar Chandra Mohan
Intel, CA USA,
vinoth.kumar.chandra.mohan@intel.com

Anjali, Singhai
Intel, Oregon USA,
anjali.singhai@intel.com

Debashis, Chatterjee
Intel, CA USA,
debashis.chatterjee@intel.com

Dan, Daly
Intel, CA USA,
dan.daly@intel.com

ABSTRACT

This paper highlights the use of the P4 language for the development of a Kubernetes load balancer and related network functions that address scale, security, and network performance requirements. Load balancers have multiple deployment scenarios from edge to data center clusters, including per-node application load distributions. A P4 data plane running on an Infrastructure Processing Unit (IPU) can serve as a highly performant, secure and flexible data plane for Container Network Interfaces (CNI) like Calico. Using P4, we can identify the packet headers and operator specific fields for load balancing with consistent service delivery across multi-cloud environments. Challenges like per flow monitoring, on-demand autoscaling and adding network policy ACLs (Access Control Lists) can be addressed with software and P4 data plane extensions on an IPU, eventually paving the path for modernized service mesh delivery.

CCS CONCEPTS

• Networks~Network services~Programmable networks

KEYWORDS

K8S, Programmable Networks, IPU, PNA

ANCS '21, December 13–16, 2021, Layfette, IN, USA

ACM ISBN 978-1-4503-9168-9/21/12…$15.00
https://doi.org/10.1145/3493425.3502768

ACM Reference Format:

Nupur Jain, Anjali Singhai Jain, Vinoth Kumar Chandra Mohan, Deb Chatterjee, Dan Daly. 2021. Kubernetes load-balancing and related network functions using P4. In Symposium on Architectures for Networking and Communications Systems (ANCS '21), December 13–16, 2021, Layfette, IN, USA. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3493425.3502768

1 INTRODUCTION

New generation of network load balancers[1] offer flexibility for dynamic scaling, route re-distribution, and service level connectivity for reliable delivery of service content to multiple requesting clients. So far, this has been offered through a model where packet processing is done either entirely in software or partly in software, and partly in hardware switches for performance gains.

Kubernetes proxy[2] is one such evolved form of load balancer that acts as the distributor sitting in front of a dynamic set of distributed pods or replicas that expose a service through a virtual IP (VIP) to external world. Any client connection to a service VIP is load balanced and undergoes Dynamic Network Address Translation (DNAT) to the pod IP address; it is also connection-tracked for fast path processing. These pods are ephemeral in nature, leading to a changing set of IP addresses, and requiring regular updates to service pool and forwarding rules by the control plane. Notifications from the hardware for flow timeouts are handled by the control plane for flow clean-up. Utilizing the fact that these are L4 transport connections distributed through consistent hashing and flow pinning by the load balancer, the functionality can be completely offloaded to a highly performant p4 programmable NIC to decrease network latencies and

effectively utilize bandwidth. Performance is especially important as every incoming packet is tracked and checked for connection policy or Access Control List (ACL) rules.

Recently, Intel unveiled a new class of processors called the Infrastructure Processing Unit (IPU). With an IPU-based architecture, CSPs can maximize data center revenue by offloading the entire infrastructure tasks from CPUs to IPUs, which frees 100% of the server CPU cycles for revenue-generating tasks. IPU also supports crypto engine, thus providing support for TLS (Transport Layer Security) and node to node crypto tunnel offload to hardware. We took advantage of P4's Portable NIC (network interface cards) Architecture (PNA)[3] to develop a K8s-lb.p4 file that is compiled using the IPU compiler to fit correctly within the hardware blocks. The compiler brings in a high level of hardware abstraction so that programmers need not understand the low-level details of the underlying hardware. This offers the control plane a simple logical view of hardware match action tables to program rules independently in each set of logical tables.

Data centers are evolving with the introduction of new stream multiplexing transport layer protocols. To adapt rapidly to these new protocols, P4 language offers user-defined parse graphs to parse the packet and extract the right header fields for fine grained load-balancing. This enables functionality upgrades with a minimal service downtime in real deployments. We believe the path forward is to make the whole networking platform programmable, giving the users the ability to observe packets and networking states, as well as the ability to enforce control plane policies and new packet forwarding behavior.

2 CONTROL PLANE

Our primary goal in experimentations with Kubernetes control plane is to introduce IPU ASIC as a p4 programmable data plane. The intention is to completely offload load balancer and related network function like ACLs (Access Control Lists) and routing as one P4 program optimized to run on hardware resources. The control plane components are also offloaded to IPU processor cores, thus freeing up host cores. Running control plane on IPU processor cores offers security domain separation from application software running on the host. The focus of this paper primarily is on offload of Kubernetes network service (Cluster IP service type). For our control plane on IPU, we use an existing Calico CNI plugin[4], an open-source networking solution for CNI. Calico is widely deployed and offers API interfaces for integrating different data plane types.

IPU Manager, an offload entity, runs on IPU processors. It uses P4Runtime client to program logical tables in hardware through Intel P4 Software Development Environment (SDE). The IPU Manager initializes the pipeline with a load-balancer p4 program and exposes the logical table view to the control plane for rule programming. All service event API calls from the CNI are terminated and handled by the IPU Manager. Each API call is translated and mapped to a set of fixed logical tables. This enables the control plane to add a new service VIP (Virtual IP), assign endpoint IPs to VIP pool, and provision forwarding rules for endpoint connectivity. The ability to add rules dynamically allows for adjusting to traffic patterns by either scaling more endpoints or removing some. With the launch of every new pod, Calico CNI runs a state machine and requests IPU Manager to add a hardware device interface to the container pod. IPU Manager also programs the pipeline with corresponding forwarding rules and policies prior to the pod activation.

3 DATA PLANE

The Intel IPU data plane comprises of a P4 based programmable ASIC pipeline, Crypto Accelerators, Inline Crypto Engine, NVMe and RDMA (Remote Direct Memory Access) Engines that make it suitable for hyperscale deployments of containerized data center applications with hardware acceleration. For load balancer network function offload, the pipeline flow is represented in Figure 1.

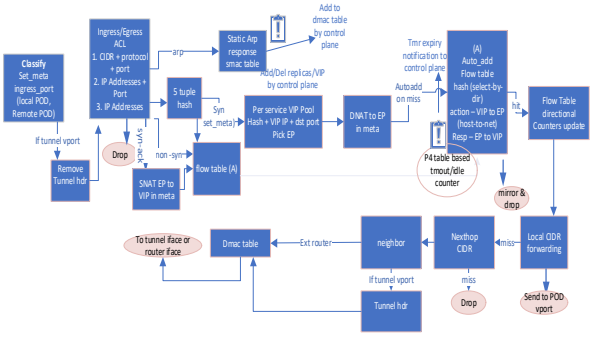

Figure 1: The load-balancer implementation in p4 programmable pipeline

On receiving a TCP SYN packet from a client, a user defined hash algorithm is used to generate a 5-tuple

hash. This hash, along with exact match on VIP and destination port, determines the worker endpoint selection from the VIP's endpoint pool. The IPU automatically adds a flow cache rule for the first packet, with an action to change the destination IP address to a worker endpoint (DNAT). The packet is forwarded to the destination pod on the same node or a remote node depending on the forwarding or routing rules defined. In case of remote node, the tunnel header is added and CRC re-generated. The use of a consistent hash in flow pinning ensures that all the subsequent packets of that transport connection are sent to the same endpoint. The consistent hash also guarantees minimal disruption to existing connections when an endpoint leaves or joins the pool. On receiving a response from a tunnel port, the outer tunnel header is removed. The packet fields are reversed and hashed to find the entry in the flow table to increment the hit counter. Flow table also does a reverse NAT to get back to the VIP address from endpoint IP address. The packet is forwarded to a local interface based on lookup. Tunnel implementation for VXLAN (Virtual Extensible LAN protocol) is already supported in PNA. PNA also offers extern function for handling Crypto; hence supports IPSEC tunnel offloads. This will help with future implementation of node-to-node IPSEC tunnels.

IPU offers two types of interfaces as resources for K8s control plane to pods requesting for IPU acceleration. The first one is Intel® Scalable IOV (IO Virtualization) ADI (Assignable Device Interfaces)) that offers scale and DMA domain isolation. The second one is Container Device Queues (CDQ) that offer port-based splitting, popularized by Linux subdev architecture. Even though CDQ does not provide memory isolation, it is ideal for device resource isolation for rate-limiting per pod interface.

4 CONCLUSION

Preliminary performance figures for node local traffic indicate reduction in CPU utilization because of the dataplane offload. IPU is designed to handle millions of flows at line rate. However, for node local traffic, we expect a slight degradation or comparable performance because of PCIe (PCI Express) hops compared to a software only solution. The future prototyping efforts will quantify the performance benefits that come from reduced packet path latencies and offloading all related network function in one optimized P4 program to the ASIC.

REFERENCES

[1] Eisenbud, Daniel E., 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), pp. 523-535. 2016

[2] Springer Caban W. (2019) Load Balancers. In: Architecting and Operating OpenShift Clusters. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-4985-7_5

[3] Brebner, Gordon. "Extending the range of P4 programmability." In Keynote in the First European P4 workshop (P4EU). 2018

[4] Kumar R., Trivedi M.C. (2021) Networking Analysis and Performance Comparison of Kubernetes CNI Plugins. In: Bhatia S.K., Tiwari S., Ruidan S., Trivedi M.C., Mishra K.K. (eds) Advances in Computer, Communication and Computational Sciences. Advances in Intelligent Systems and Computing, vol 1158. Springer, Singapore. https://doi.org/10.1007/978-981-15-4409-5_9

[5] SHANTI SWAROOP MOHARANA , RAJADEEPAN D. RAMESH & DIGAMBER POWAR, 2013. ANALYSIS OF LOAD BALANCERS IN CLOUD COMPUTING, International Journal of Computer Science and Engineering (IJCSE) ISSN 2278-9960 Vol. 2, Issue 2, May 2013, 101-108.https://www.researchgate.net/profile/Shanti-Moharana/publication/236521813_Analysis_of_Load_Balancers_in_Cloud_Computing/links/00b7d517fd8f65d3d1000000/Analysis-of-Load-Balancers-in-Cloud-Computing.pdf

[6] PNA https://github.com/p4lang/pna