L3AF: Kernel Function as a Service

Table of Contents

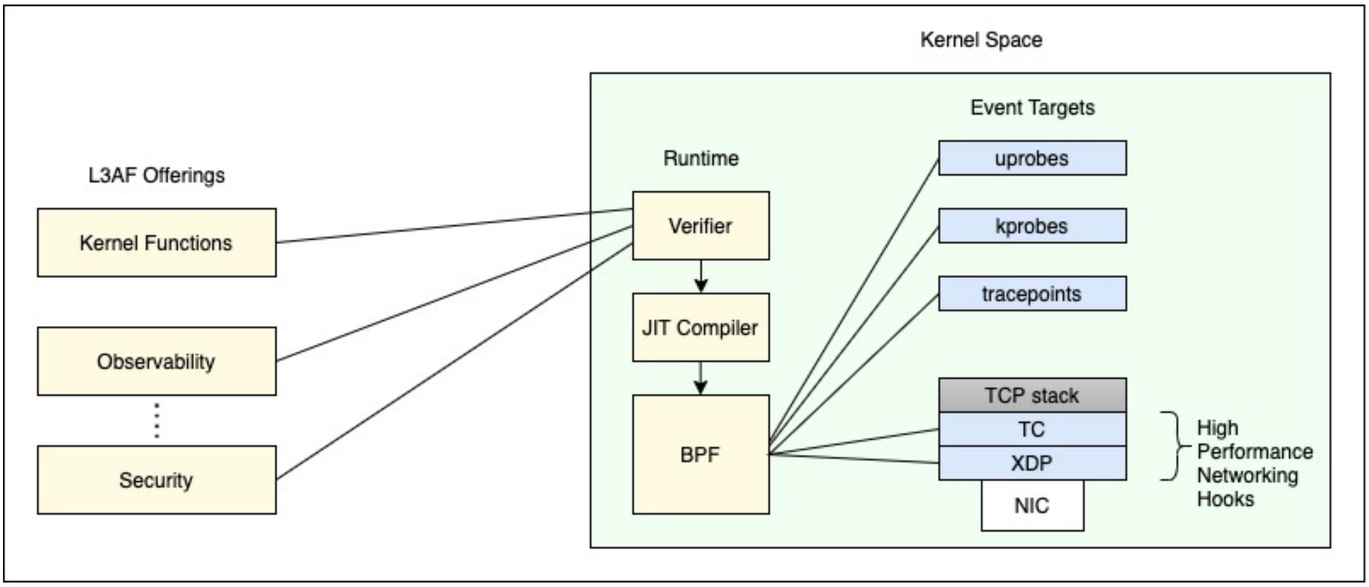
Getting started with eBPF

Traditionally, applications that are running in user space make system calls to access the kernel resources. But now, eBPF presents a new model that allows us to run custom sandboxed code in the kernel. With eBPF, kernel functions can be extended/customised through simple programs. These programs can be associated with desired kernel events, so they are executed whenever the event happens. To give an analogy, eBPF programs are to the kernel as to what plugins are to proxies or web servers.

Let us look at this in a little bit more detail to see how eBPF makes this possible. eBPF runs as a mini-VM inside the kernel. This mini-VM provides a sandboxed environment that has out-of-the-box integrations with low-level network hooks such as XDP/TC as well as probing mechanisms such as kprobes, uprobes, and tracepoints. With this architecture, it is now possible to write efficient eBPF programs and run them in the kernel. eBPF kernel programs are written in C and compiled to eBPF bytecode.

eBPF also provides a safe and secure way to do all of this inside the kernel. The verification step ensures that the eBPF program is safe to run. It validates that the program meets several conditions, for example, it makes sure that the program does not crash and that it always runs to completion (w/o sitting in infinite loops). The Just-in-Time (JIT) compilation step translates the generic bytecode of the program into the machine-specific instruction set to optimize the execution speed of the program. This makes eBPF programs run as efficiently as natively compiled kernel code.

For more details on eBPF, please refer https://ebpf.io/



eBPF Integration and Hooks

Introducing L3AF

Walmart Global Tech is developing some of the most cutting-edge products in the realm of eBPF under a project called L3AF. L3AF provides eBPF based networking and observability solutions with the help of an advanced control plane written in Go.

In the realm of networking, L3AF enables Kernel Function as a Service by providing complete lifecycle management of eBPF programs that instrument, inspect, and interdict traffic. These eBPF programs use low-level network hooks such as XDP and TC to give us an ultra-high performance programmable network data plane that executes prior to the higher and slower layers of the Linux networking stack.

On the observability side, L3AF provides a list of curated metrics by collecting and aggregating custom information generated at the source of the event in the kernel. These metrics provide detailed insight about cluster/node utilization and downstream/upstream network performance as well as traffic distribution across multiple clouds. L3AF provides deeper visibility into the system performance when compared with other programs, which rely on static counters and gauges exposed by the operating system(like /proc). L3AF also offers out-of-the-box integration with Prometheus by maintaining full compatibility, including support for Prom QL.

In this paper, we focus on L3AF's networking solutions, which enable Kernel Function as a Service (KFaaS)

Network Landscape, Challenges and Solutions

As enterprises adopt a hybrid cloud strategy and migrate workloads from private to public cloud, a few interesting opportunities become apparent:

- There is an immediate need to get the same level of network visibility in the public cloud as in the private cloud.

- A hybrid cloud environment requires greater control over traffic due to the limitations and costs associated with networks.
- It is essential to replace hardware-based packet processing solutions in private clouds with software-based solutions to achieve feature parity with public clouds as enterprises move towards symmetric deployment.

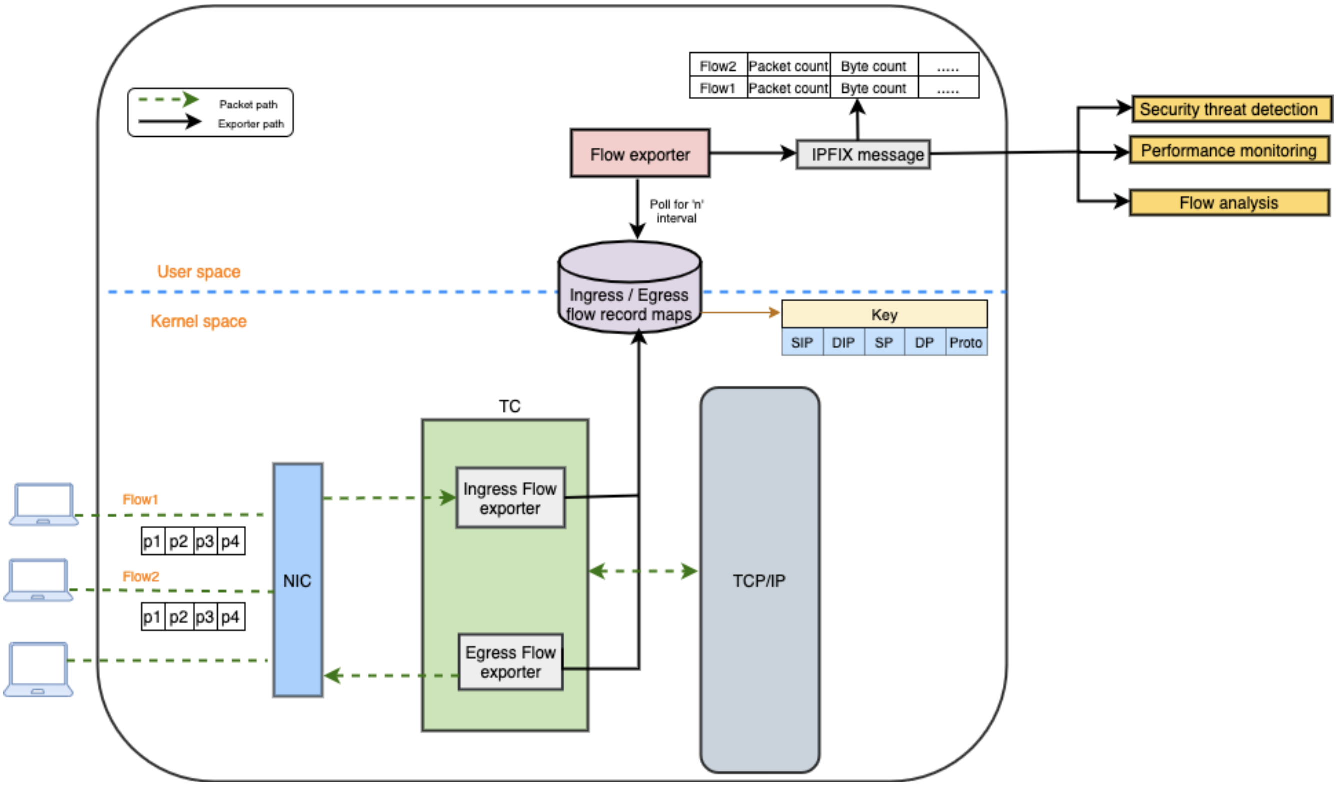
Let’s deep dive into some of these functionality gaps and their eBPF based solutions:

Traffic Flow Logs

As enterprises start serving live traffic out of public clouds, it is increasingly important for them to export Traffic Flow data to security solutions that provide advanced threat protection across the extended network and cloud.

Private clouds provide Traffic Flow data through dedicated network appliances (hardware-based solutions). However, tenants in public clouds do not enjoy a similar level of access/network visibility since the infrastructure layer is shared. Walmart considered options to address this, such as adding a network hop to process Traffic Flow data (using NetFlow protocol). Such a configuration increases traffic latency, and also adds another layer to manage in the traffic ingress stack.

As the best solution, Walmart’s L3AF project developed an eBPF program (Kernel Function), which extracts and exports flow metadata directly from linux based edge proxy servers.



eBPF based Flow Exporter Solution

As shown in the diagram, the eBPF Kernel Function (TC) retrieves flow attributes from every ingress and egress packet and updates them in the eBPF map as flow records. eBPF maps are generic key-value storage of various types that can share data between user and kernel space. Flow record maps contain a 5-tuple (sa, da, sp, dp, proto) as key, and other flow attributes like packet/byte counters, last seen, TCP flags etc. as values.

The flow exporter in user-space reads flow records periodically and calculates flow statistics since the last poll. The delta in counters between consecutive reads is calculated by preserving the flow state (after every read) in another last record map. This allows us to accurately track the active flows.

Once the necessary flow attributes are in place, an IPFIX message is created, as per RFC, and exported to any security threat detection and analysis tool.

Key Benefits are as follows:

- In addition to saving infrastructure costs, L3AF's solution significantly lowers the overall latency of traffic by eliminating the need to pass through an additional network hop.

- This solution uses a more flexible open-source IPFIX protocol instead of NetFlow. Unlike NetFlow, IPFIX allows the export of variable length custom fields (such as URL) that provides enhanced visibility.

Traffic Mirroring

The best way to succeed in a business is by providing an amazing customer experience. The quality of the overall experience is often what influences customers when they shop online. At Walmart, we want to have visibility into how our customers are interacting with our site.

Walmart has a few analytics solutions that can operate on the data streams and provide the needed analysis. But these solutions need the data of interest and that interest changes from time to time. There is an opportunity to save valuable time and money by automating the process of collecting this data.

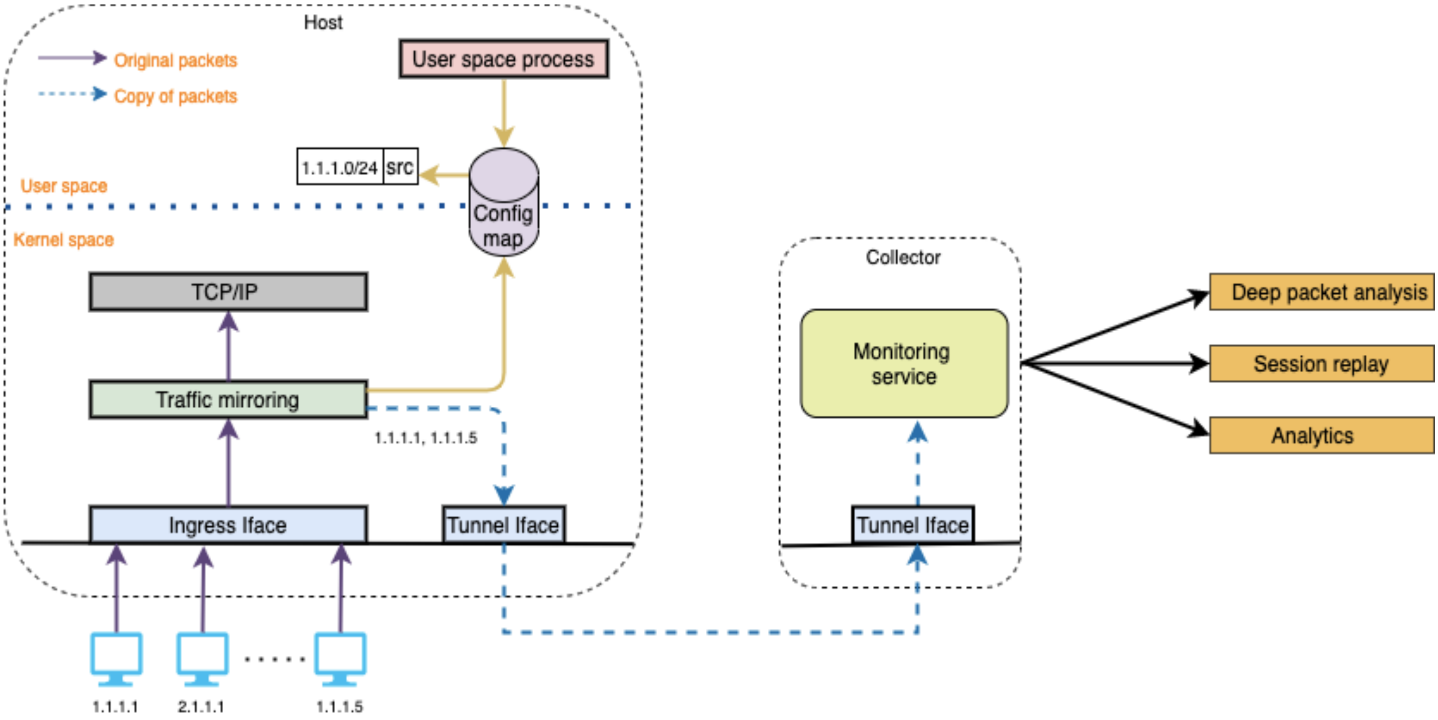
One of the most effective ways of collecting this data of interest in the public cloud is from the edge proxy servers. However, it is also a critical hop that handles all the ingress traffic to the site and is performance sensitive.

So, we started exploring some of the commercial solutions, a few of which are listed here:

- Running a stand-alone agent that would mirror 100% of traffic on the proxy VMs. However, this would incur:
  - Significant traffic expenses as we would mirror 100% data.
  - Additional licensing cost ($ / NIC that we decide to mirror).
  - Overhead on the resources of host.

- Using traffic mirroring services that are offered natively by the public cloud. However, this isn't a consistent solution as many flavours of the public cloud either do not offer this solution or do not offer the necessary capability to filter the data of interest.

To overcome these limitations, the L3AF project developed an eBPF based software solution that encapsulates the filtering and mirroring functionalities together. This solution supports one or many custom filters in 5-tuple (sa, da, sp, dp, proto) and also allows us to capture only header data (if required), thereby limiting the bandwidth utilization.



eBPF based Traffic Mirroring Solution

Additionally, given that eBPF is very light weight, highly performant and safe, this solution has been implemented at the source (i.e., on the edge proxy). So, on the edge proxy, we attach the mirroring function to the primary NIC that processes the actual traffic. This solution examines every incoming/outgoing packet using the TC hook and matches it against the filter (5-tuple based). If the match is successful, it clones the packet and redirects to a secondary NIC that forwards the traffic to the analytics systems on a GUE tunnel.

Key Benefits are as follows:

- Unlike the hardware-based solutions (taps) primarily used in the private cloud, we achieved the use-case through a software-based solution. This software-based solution runs directly on the host thus, eliminating an additional point of hardware failure and saving on infrastructure cost.

- Traditional cloud solutions mirror 100% of traffic to an aggregation layer that sits in a centralized cloud. So, there is an additional traffic cost involved in sending the traffic and receiving optimized traffic back into our cloud subscription. The customized filtering capability allows us to mirror only the traffic that we care about to the analytics systems.

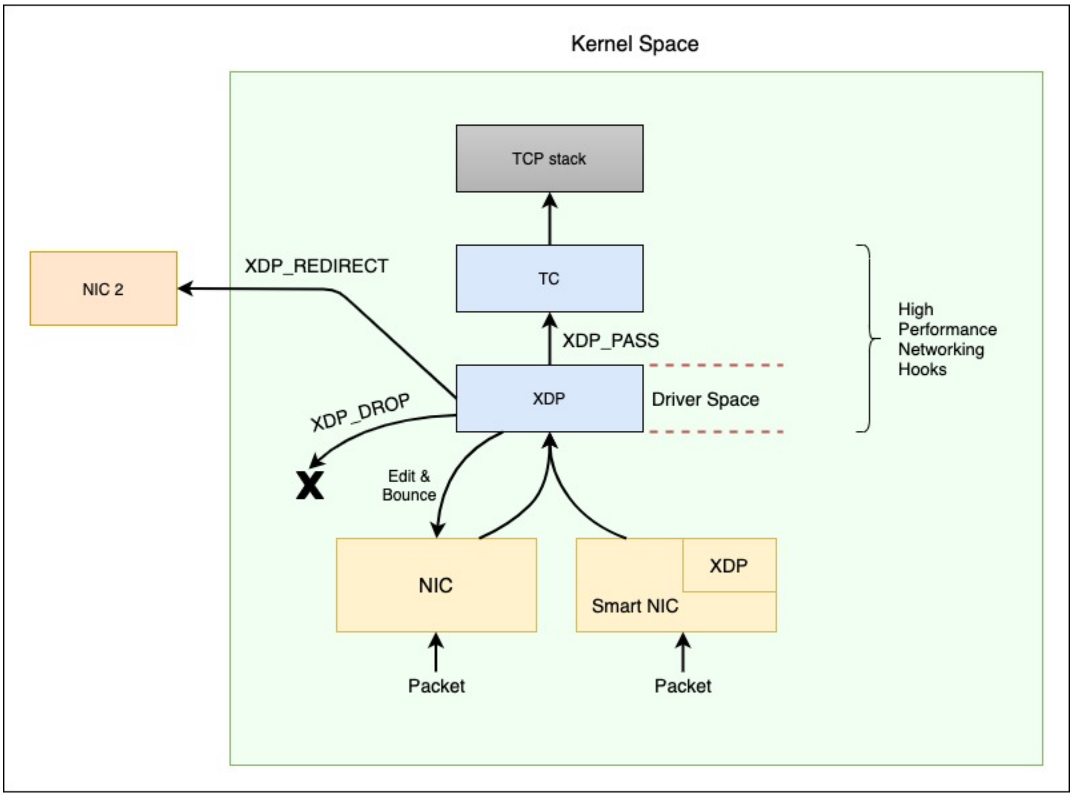
- Since eBPF solutions can be dynamically programmed to attach/detach, they provide the flexibility to selectively choose the applications/domains that we want visibility into, on the fly, in a seamless manner.

We have reviewed a few network visibility use-cases so far. In the next section, we will focus on how L3AF uses XDP to provide high-performance networking solutions at scale.

XDP based processing at scale

With the advent of XDP and eBPF, it is now possible to achieve high performance packet processing in the kernel data path. XDP allows us to attach an eBPF program to a low-level hook inside the kernel. This XDP hook implemented by the network driver provides a programmable layer before the Linux networking stack.

When a packet arrives, the network driver executes the eBPF program in the main XDP hook. This framework allows us to execute custom eBPF programs at the earliest possible point after the packet is received from the hardware, thereby ensuring ultra-high performance. Some of the smart NICs also support offloaded XDP, which lets the program run on the NIC without using the host CPU in any way.



XDP in action

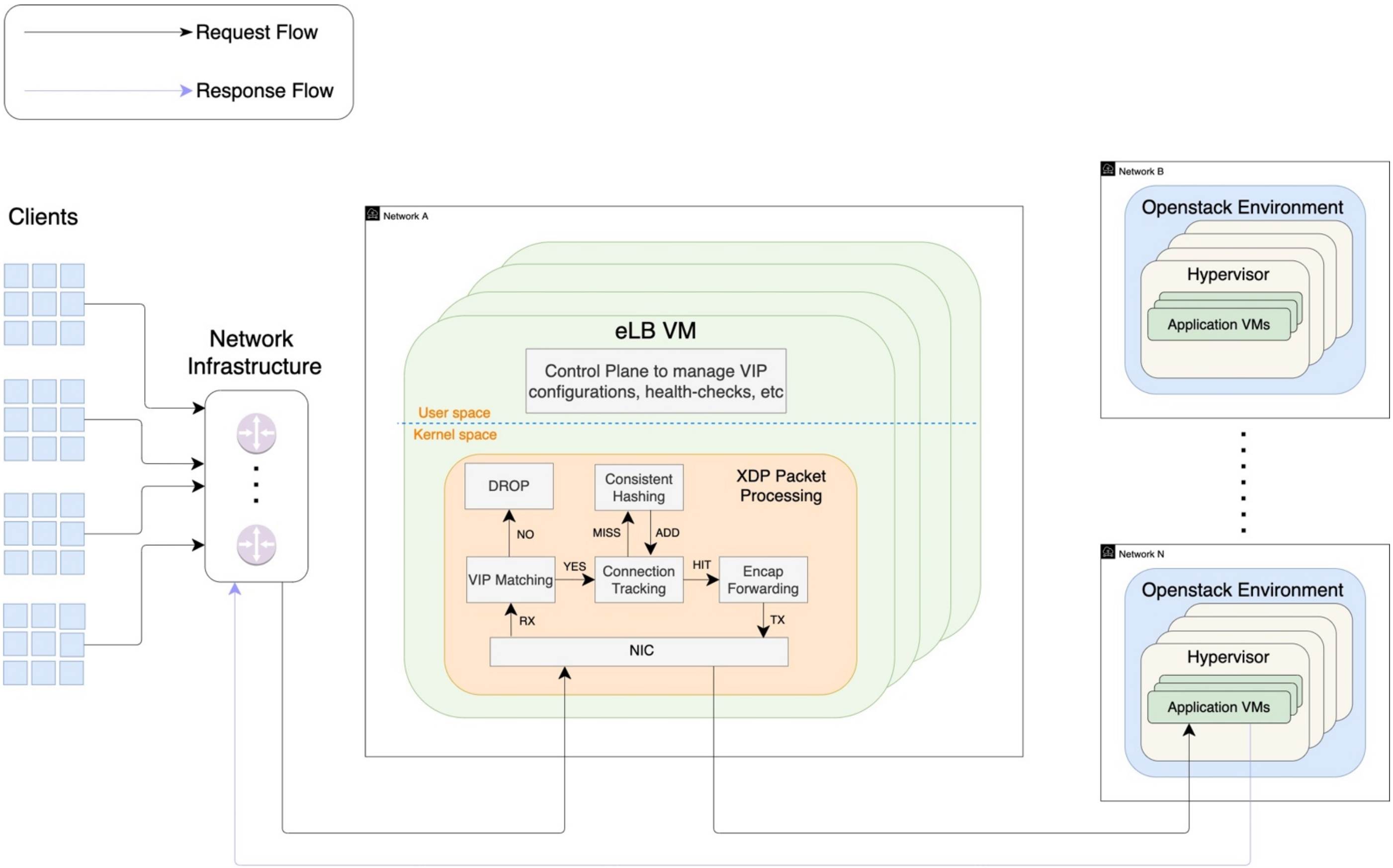
In the next couple of sections, we will discuss a few functionality gaps that can be addressed by leveraging XDP/eBPF to take direct action in the traffic path.

eBPF Load-Balancer (L4)

Several internet companies serve millions of requests every second out of their edge network. The Layer 4 Load Balancer which is located at critical paths in the traffic flow takes an extremely high volume of traffic. Since the L4 LB must process every incoming packet, the solution needs to be highly performant. And also, must provide the necessary flexibility and scalability required in production environments.

Traditionally, L4 load balancers have been hardware-based primarily to suit the high-performance requirement. However, taking a hardware-centric approach limits the system's flexibility and introduces limitations such as lack of agility, scalability, and elasticity. As applications increase in number, complexity, and importance, it is vital that the infrastructure layer is app-focused and not limited by the confines of hardware configurations.

All the performance needs can be met with software solutions themselves using eBPF. The L3AF project has leveraged Katran to develop an eBPF based load balancer offering (eLB) that is implemented using XDP in a hair-pin model on a single NIC. This load balancer redirects traffic to the backend at the device driver level using XDP_TX.

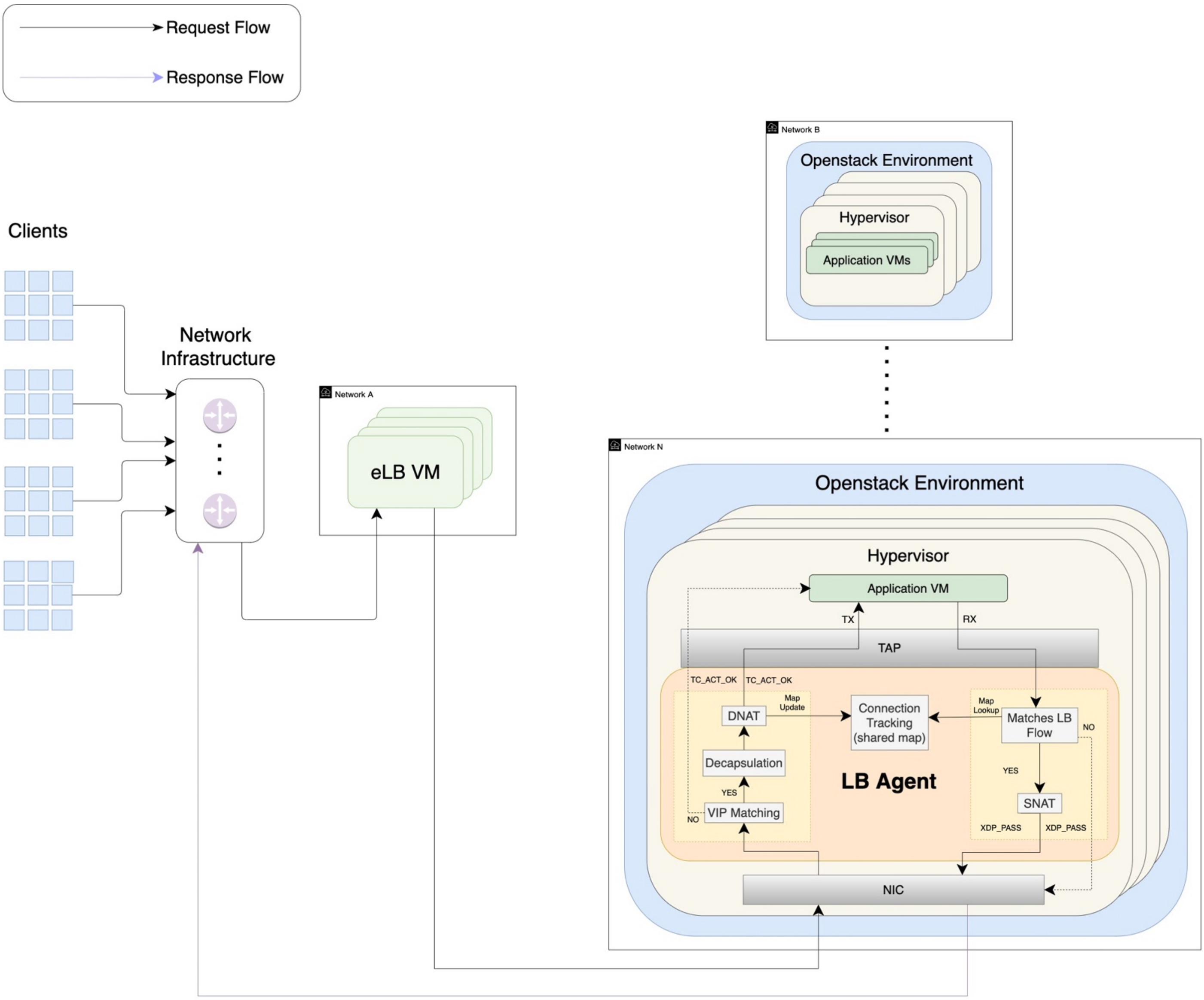


eLB - XDP based L4 Load-Balancer

One of the key features that we wanted to enable in Walmart’s production environment is DSR (Direct Server Return) so that we can send responses directly to the client. This ensures that the eLB does not need to handle return packets, which are typically larger in size.

To implement DSR, we developed an LB Agent that runs on our fleet of hypervisors in the private cloud. For other environment types, we run the agent on VMs (Virtual Machines), bare metals, etc. depending on the use case. The agent can run on any commodity hardware that is based on Linux.

When a client requests an application service, the router receives a VIP packet. The router then forwards the packet to one of the eLB nodes in the cluster through ECMP. Since all the eLB nodes announce the VIP with the same cost, they are BGP peered with the router (we run goBGP on LB nodes). When eLB receives the packet, it runs the Maglev algorithm, selects an endpoint from the set of service endpoints associated with the VIP, and encapsulates the packet using Generic UDP Encapsulation (GUE) with the outer IP header destined to the service endpoint. In Walmart's private cloud environment, encapsulation is necessary as we need to route packets to the backends that are in a different L2 domain as of the corresponding eLB node.



LB Agent to enable DSR

Application nodes receive encapsulated traffic forwarded by eLB. The lifecycle of most of our virtualized app workloads is directly managed by the individual app teams. To have a solution that is transparent to the applications, the agent is designed to run on the tap devices corresponding to the VMs. The agent decapsulates the incoming traffic and forwards it to the application. And in the return path, the agent SNATs the outgoing traffic to the VIP using connection tracking – all of which is done using eBPF (XDP and TC hooks).

The LB agent is provisioned dynamically by our control plane and, the solution is resilient to any changes at the infrastructure layer (scaling, migration, etc.)

Key Benefits are as follows:

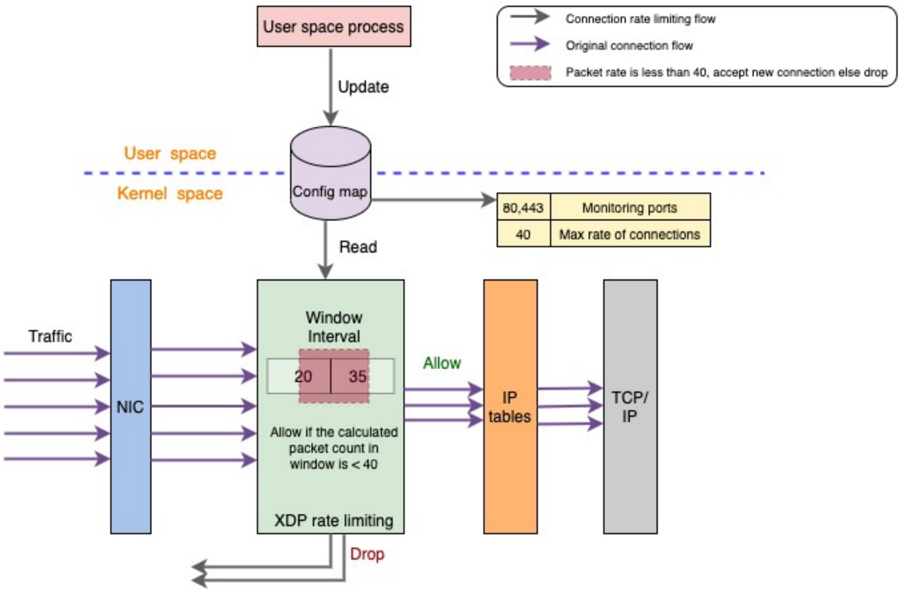
- eLB is helping us replace hardware-based solutions that limit the system's flexibility with a modern software-based solution. Enabling DSR not only eliminates centralized choke points in our network but also helps us improve the overall site latency.
- eLB leverages XDP which gives much better performance when compared to other software technologies such as DPDK and LVS. This coupled with DSR enables us to significantly reduce our L4 LB infrastructure footprint.

Connection and Connection Rate Limiting

As enterprises increase their digital footprint, it is vital to have safeguards put against sudden bursts of traffic or cyber-attacks. Having a connection limiting and connection rate limiting feature allows us to limit the concurrent number of TCP connections and the rate at which new TCP connections are established respectively. We also want this limit to be tuneable based on adequate benchmarking to ensure that the upstream systems are not overwhelmed.

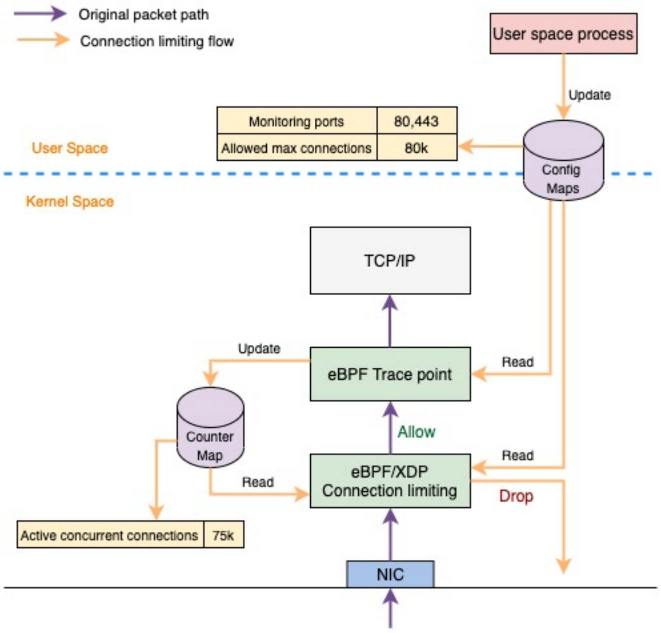
Walmart has developed the eBPF/XDP programs that can perform connection and connection rate limiting.

- Connection rate limiting uses a sliding window approach for managing the connections, which is much easier to use and understand when compared to token bucket/leaky bucket algorithms. The program only expects the "traffic rate" as input, while some of the other algorithms also need "traffic burst" as input, which is difficult to determine in complex systems. The program essentially uses the traffic patterns that it sees on the system to do the needed calculations internally.



Connection Rate Limiting Solution

- Max connection limit uses tracepoints/kprobes to track the number of concurrent connections and gives the feedback (using BPF maps) to an XDP function that drops/resets the connections if the number exceeds the max limit configured.



Connection Limiting Solution

Key Benefits are as follows:

- Adding the connection/rate limiting functionality to our edge proxies and load-balancers protects our compute resources from getting overwhelmed when there is a sudden burst of traffic that is beyond what our resources are capable of handling.
- By using XDP, we are able to drop connections at much higher rates compared to other solutions.

Both the above XDP functions and XDP based eLB, can be run in a chained fashion to work cooperatively with each other. This can ensure that all the illegitimate traffic is dropped by the connection and connection rate limit functions and eLB doesn't have any undesired affects even under adverse conditions.

In the next section, we discuss how to orchestrate kernel functions in the desired sequence (For example : rate-limit->max-limit->eLB).

Control Plane, Chaining of eBPF programs, and Open-Source plans

The popularity of eBPF is rapidly growing. There are more and more eBPF programs being written to solve a wide variety of problems. Several start-ups are building technologies around eBPF, and large technology companies like Facebook, Netflix, and even Microsoft are embracing eBPF to solve large-scale problems. At Walmart, we too are embracing eBPF and using it to solve similar problems.

A challenge we faced when first adopting eBPF was how to manage and orchestrate multiple eBPF programs on a large scale. We require to run numerous eBPF programs on a given node and, we have thousands of nodes across many DCs in a hybrid cloud environment using multiple cloud providers. Due to the lack of an enterprise-ready solution, we decided to develop our own control plane. This control plane orchestrates and composes independent eBPF programs across our network infrastructure to solve crucial business problems. Our control plane is a vital component of L3AF.

Enabling Kernel Function as a Service

The popularity of eBPF is rapidly growing. There are more and more eBPF programs being written to solve a wide variety of problems. Several startups are building technologies around eBPF, and large technology companies like Facebook, Netflix, and even Microsoft are embracing eBPF to solve large scale problems. At Walmart, we too are embracing eBPF and using it to solve similar problems.

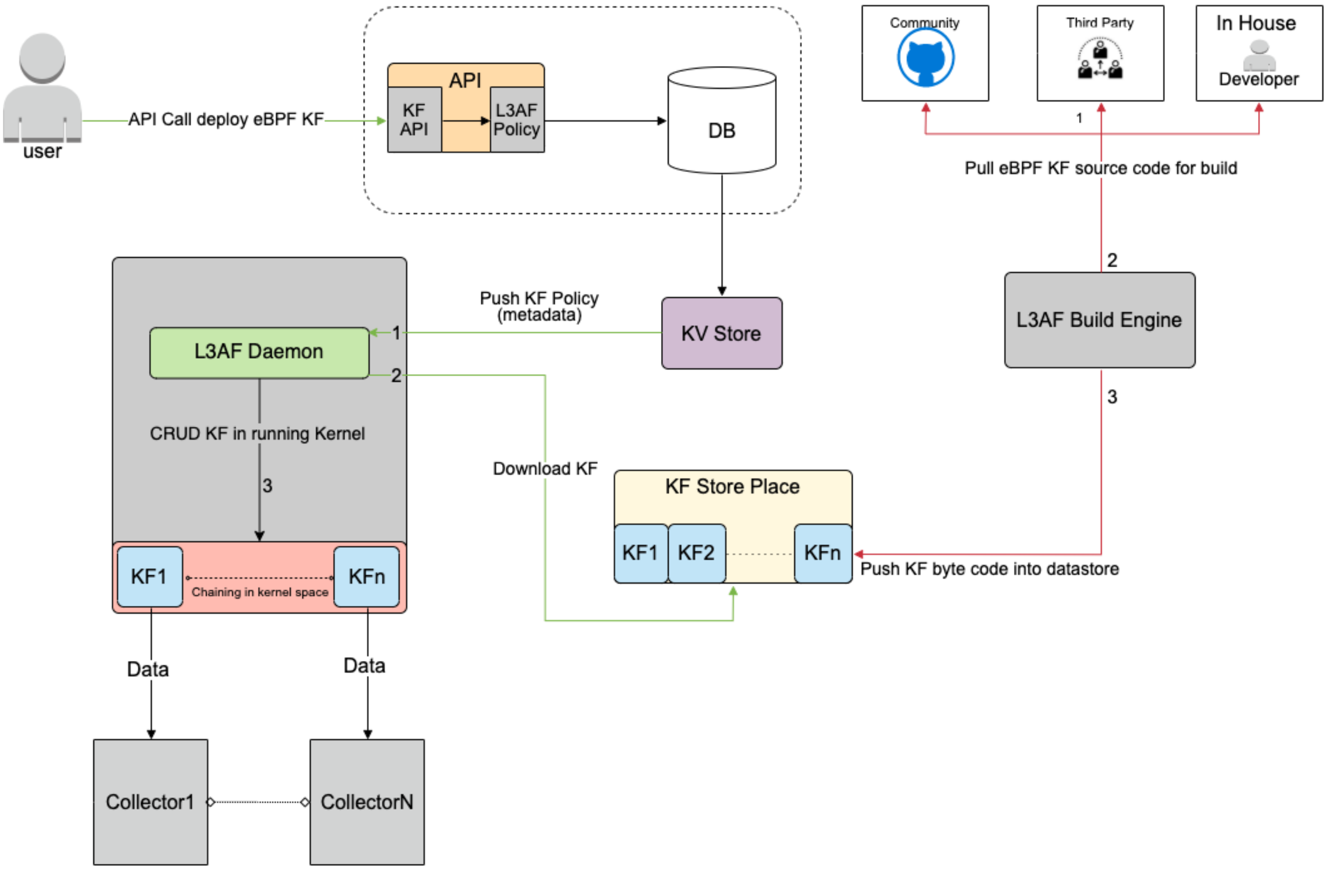
A challenge we faced when first adopting eBPF was how to manage and orchestrate multiple eBPF programs on a large scale. At Walmart, we require to run numerous eBPF programs on a given node and, we have thousands of nodes across many DCs in a hybrid cloud environment using multiple cloud providers. Due to the lack of an enterprise-ready solution, we decided to develop our own control plane. This control plane orchestrates and composes independent eBPF programs across our network infrastructure to solve crucial business problems. Our control plane is a vital component of L3AF.

Our control plane consists of multiple components that work together to orchestrate eBPF programs:

- L3AF Daemon (L3AFD), which runs on each node where KF runs. L3AFD reads configuration data and manages the execution and monitoring of KFs running on the node. L3AFD is written in Golang.

- Deployment APIs, which a user to generate configuration data. This configuration data includes which KFs will run, their execution order and the configuration arguments for each KF.
- A database and KV store that stores the configuration data.
- A datastore that stores the KF byte code.

The control plane is shown graphically here:



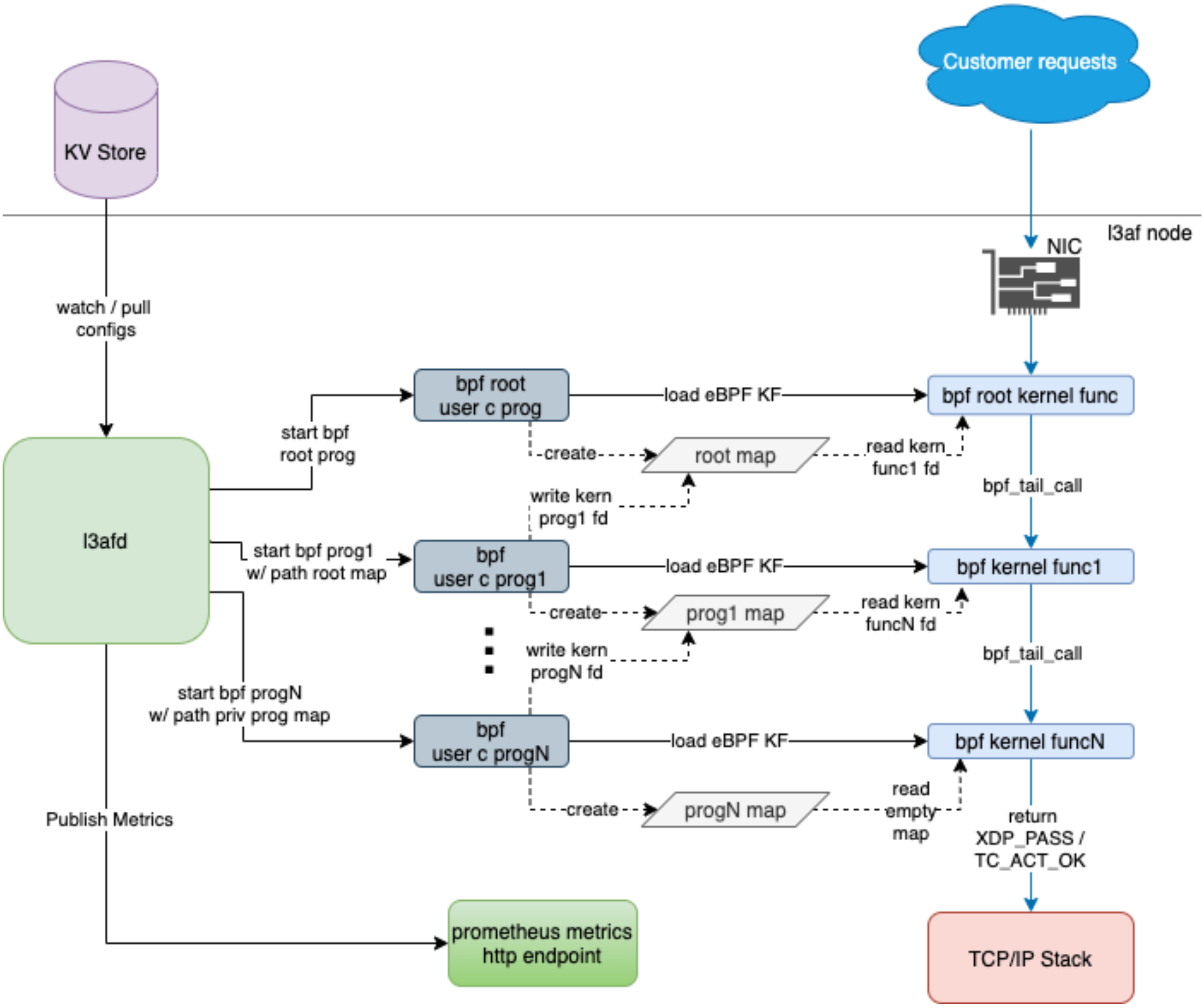
L3AF Ecosystem

As seen in the above diagram, eBPF KFs can be community developed, third-party vendor ones, or the ones available from L3AF. The L3AF build engine pulls the Kernel Function source code, compiles the source code against different kernel versions, and pushes the bytecode to an artifact management solution.

When users want to deploy a KF, they can call the L3AFD API with appropriate parameters. This request would generate a new config (KV pair) that will be saved in a database and distributed across all the hosts using a config distribution mechanism.

Once L3AFD reads this new config, it orchestrates kernel functions on the Linux host as per the defined parameters. If the user gives a set of kernel functions, then L3AFD can orchestrate all of them in the sequence that the user wanted.

Executing eBPF programs in a sequence is called "chaining," and it is quite complex (at least for the kernel versions we use in production). Let us do a deep dive into how L3AFD makes this possible. Below is a diagram followed with some explanation:



L3AFD Orchestration

At a high level, what this diagram shows is that L3AFD chains eBPF KFs by leveraging eBPF maps. Chaining is achieved by having the next program's file descriptor (fd) stored in a map created by the previous program. A given eBPF kernel program then calls the next eBPF kernel program by using the bpf_tail_call kernel functionality. This happens repeatedly until the end of the chain is reached.

In actuality, for each chain, only the first eBPF kernel program is attached to the network interface. Subsequent eBPF programs in the chain are essentially called on behalf of this first program. Due to this, there is a requirement for a "root" passthrough program. The root program allows chains to rebuild without detaching or reattaching the eBPF program to the interface.

Below is the list of steps involved in creating the chain:

- The config that L3AFD receives includes the network interface and program type (XDP, TC ingress, and TC egress). It also includes a sequence number that indicates the position of a KF in the chain. Based on the information that is available in the config, L3AFD downloads artifacts from the KF datastore onto the node.

- If the sequence number is 1 (a KF is the first eBPF program), then L3AFD will perform the following:
    - Start the appropriate type of root program (i.e., XDP or TC) depending on the program type. This root program will use libbpf API's or TC hooks to attach the root kernel bytecode to the Network Interface. <Write something>
    - Start the "user prog1" of the first KF using APIs with start arguments and this "user prog1" loads the bytecode of "kernel func1" and updates func1 fd into the root map using eBPF APIs.

- If the sequence number is X (somewhere in the middle of the chain), L3AFD will perform the following:
    - Start the KF and update the next KF's program fd (X+1) to the progX map.
    - Update the progX map into the previous KF's program map (X-1) like an insertion in the linked list using Cilium's eBPF library APIs.

- If the sequence number is Z (last KF in the chain), then L3AFD will perform the following:
    - Start the KF and update the progZ map into the previous KF's program map (Z-1).

L3AF adheres to the "build once, deploy everywhere" philosophy, wherein we would build the deployment package once for any environment (i.e., multiple kernel versions) and set configuration at deploy time.

L3AFD has other duties, too. It monitors KF health, gathers configurable KF metrics, and manages KF resource utilization. This health and metrics data gets exported in a format compatible with PromQL.

L3AFD provides an API for configuration, so users may use their existing systems for configuration distribution to L3AFD nodes. For example, users may use etcd, consul, or a custom in-house solution to distribute configuration data to the L3AF nodes. Then, a small service can be implemented to convert the configuration data to L3AFD API calls. The L3AF team is interested in various projects that are aiming to standardize configuration distribution for hybrid cloud environments, and future versions of L3AF may move in that direction.

Speaking of the future, this is just the beginning for L3AF. We have many exciting ideas for the future.

Future Plans

eBPF is a cutting-edge technology. eBPF features are frequently added in new kernel versions. Also, new userspace tools and libraries are emerging, which weren't available when we started developing L3AF a couple of years ago.

At present, as shown in the diagram above, L3AFD executes separate C userspace programs, which load corresponding eBPF Kernel Functions. However, a pure Go eBPF userspace library has been created (by Cilium). We plan to leverage that library to port all of our C userspace code to Go, which should simplify our userspace code and expand the capabilities of what the userspace eBPF programs can do. We plan to implement the new Go userspace programs as RPC-based plugins for improved process control and communication.

As mentioned, KF chaining is quite complex. However, it can be simplified in Linux kernel versions 5.10 and higher. With these kernel versions, libxdp and its XDP Dispatcher functionality make things much more straightforward (but we will keep the support for chaining with older kernels). TC chaining can also be simplified using TC's userspace tools.

Another crucial part of our L3AF project is to establish a "Kernel Function Marketplace," where KF developers and users can share their own signed KFs and download KFs from others. L3AF can then be used to orchestrate and compose selected KFs from the marketplace to several business needs.

This concludes our paper on eBPF networking solutions at Walmart. We've discussed network visibility, XDP packet-processing, and developing our own control plane. Together, these parts have come together to form our L3AF model that we're using to solve difficult problems in innovative ways. Before we end, we'd like to give an overview of the benefits of our L3AF model.

Benefits of L3AF Model

Many commercial solutions can aggregate and route traffic to relevant monitoring and analysis tools. However, these solutions are proprietary which limits their offerings to the features and functions that have been developed by their engineers. The idea behind L3AF is to provide an open and extensible platform that offers certain Kernel Functions out of the box. And also enables users to add offerings dynamically to our KF ecosystem as per their use-cases and requirements.

Additionally, L3AF can support use-cases where action needs to be taken in the direct path of traffic. A few examples of such use-cases are packet tagging, rate limiting, load-balancing, and traffic direction. Such use-cases are not possible to achieve with traditional agent-based solutions, as most of them are still running their programs in the TCP/IP stack. L3AF leverages eBPF, which allows us to run these in the kernel with ultra-low overhead. And, also offers capabilities that other commercially available tools will not be able to unless they go through major design and architecture level changes. All of these give L3AF the first-mover advantage in this space.

To summarize, below are the key benefits of the L3AF model:

Technical Benefits are as follows:

- One-Stop-Shop for all Kernel Functions, thereby avoiding vendor and cloud lock-ins.
- Distributes Kernel Functions across hosts eliminating appliance and centralized choke points in the network (and acts as close to the source as possible).
- Supports Kernel Function chaining to achieve desired workflows.

- Leverages eBPF for data path, so gives an ultra-high performance.
- Reduces any additional hops that may be otherwise necessary to perform the Kernel Functions managed by L3AFD.

Business Benefits are as follows:

- Flexible platform to configure, customize and monitor the Kernel Functions according to the user requirements.
- Reduce licensing expense and overhead of managing Kernel Functions across numerous vendor products and making them work together.
- Reduce integration costs of implementing new Kernel Functions across platforms and avoid being limited by the lowest common denominator across platforms.
- Reduce network hops avoiding additional public cloud traffic costs and bottlenecks.
- Offers Cutting edge technology solutions with capabilities that are yet to be available in commercial tools.

Thanks for tuning in!