Shared Memory Enabled Service Plane Optimization

Ramanathan Subramanian
Enterprise Routing
Cisco Systems
San Jose, USA
ram.subramanian@cisco.com

Jeslin Puthenparambil
Enterprise Routing
Cisco Systems
San Jose, USA
jantony@cisco.com

***Abstract*—Applications and services that Edge technologies offer has made critical the efficient implementation and deployment of container-based network functions. The current SD-WAN applications make inefficient use of system resources, and will greatly benefit from rearchitecting the packet flow. We demonstrate our service plane optimization with the Snort application hosted as a container on Cisco Cloud Services Router (CSR).**

***Index Terms*—SD-WAN, NFV, Edge Computing, Snort, Service Plane**

I. Motivation

SD-WAN applications running as a container on networking hardware makes inefficient use of system resources. We propose and implement service plane optimizations to significantly improve CPU cores' utilization and per packet latency for the SD-WAN container applications (for e.g., open source Snort & AppQoE) hosted on Cisco devices. We will use Snort for discussion as it makes for a good use case. Snort essentially checks the unencrypted payload (includes the IP header fields) of the packet under inspection against numerous downloaded signatures to declare a verdict (forward/drop the packet).

II. Technical Contribution

Packets are received through polling. Threads will run continuously and poll the queue for fetching the packets. The DPDK poll mode driver DMA's the packets from the physical Ethernet interfaces to memory. In the existing approach, shown in the Fig. 1, there is actually post-processing of the packet back onto to the PP2 (Packet Processing) block even after say Snort's verdict was to drop the packet post re-insertion - notice the red dashed path of the packet leading it to being subsequently dropped. The post-processing on each packet translates to significant latency which consequently impacts the throughput. First, we do away with copying packets (making duplicate copies) meant for the punt path, by using our shared memory implementation *memif*. Note that copying packets is time consuming, and with small packets the copying action is quite frequent. The new approach saves one memory copy in each direction, and all kernel calls are saved too. The shared memory is only accessible between a data plane (which originates the data packets) and a service plane (associated with an application the data packets are being transmitted to). The data plane can store relevant information towards identifying those data packets, say in a ring buffer. Identification of the data packet associated with that relevant information becomes possible when the identifying information is provided with the verdict. The proposed approach exhibits a marked improvement over the existing approach with the rate of punting packets to the Snort running as a container e.g. 1M packet can be pushed up and down the Virtual Port Group (VPG). Moreover, the interface provides enhanced security by embracing an always point-to-point Master/Slave concept, and employing pointer free shared memory rings and descriptors.

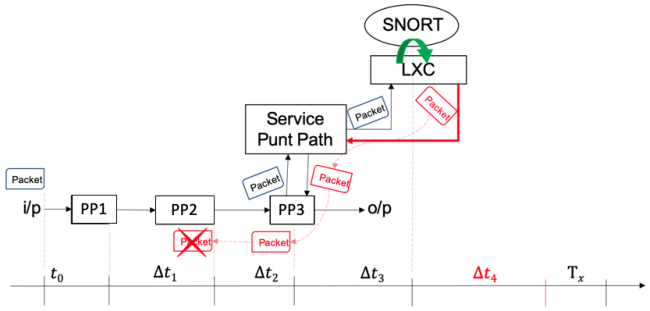


Fig. 1. Existing service punt/inject path.

TABLE I
Test Results on Cisco Cloud Services Router

Packet Size	Simple Loopback	Loopback with *memif*
362	PPS: 30,000	PPS: 320,000
100	PPS: 143,000	PPS: 831,000
1000	PPS: 121,000	PPS: 121,600

Fig. 2 succinctly captures how the per-packet latency is improved by the optimization proposed. The figure below indicates the marked improvement in the relevant metric, Packets Per Second (PPS), when *memif* is being used. Second, the packet need not be re-inserted. Instead, we will send only the verdict on the packet under inspection to PPE3 using the *memif* shared memory implementation. PPE3 will drop/forward the packet based on the verdict. The latency incurred in waiting for verdict through the *memif* interface

978-1-7281-3187-0/20/$31.00 ©2020 IEEE

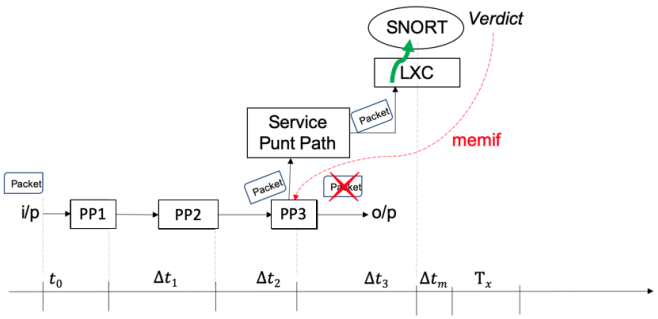


Fig. 2. *memif* enabled service punt/inject path.

is negligible as compared to the latency tolerated from the re-insertion of the packet in the existing approach. A combination of packet fields is usually used to identify the flow a packet belongs to e.g. NetFlow methodology. Let us say the first packet was identified to belong to a malicious flow. This observation can be used to avoid punting every subsequent packet belonging to the same flow (identified by the first packet in that flow that reached Snort). It precludes even the need to wait on a verdict from Snort.

III. Use Case

The data streams that have high confidence levels allow for savings in time and computing resources by having some number of subsequent data packets passed through without further evaluation. This savings can provide an improvement in per packet latency and throughput within the network. The time/compute resources can also be re-allocated to perform evaluations for those data content streams that have lower confidence levels and therefore necessitate more frequent evaluations say to prevent malicious content from being used and transmitted within the network. This approach will allow for tremendous overall savings in computing and memory resources. With this feature implemented, more apps can be deployed, higher throughput licenses can be supported bringing in more revenue. Essentially, customers using any networking hardware that allows for "bump in the wire" (typically all the Cisco routers and switches) will benefit from the proposed approach. Applications like AppQoE, SDWAN optimization, and Snort running on such devices as Intrusion Prevention System (IPS) will benefit from the proposed optimization. For companies selling networking gear in large volumes, these optimizations will bring in significant cost savings and additional sales.

IV. Technology Demonstration

The demo will showcase the working model of the shared memory implementation with CSR. CSR is an inexpensive hardware (also has a virtual VM instance), and makes it easier to spin up and establish a proof of concept. We will demonstrate the three scenarios as depicted in Fig. 3, and in particular the performance in the third "With App" scenario with and without *memif* will be established (refer Table I). We will demo the enhancement in performance by passing

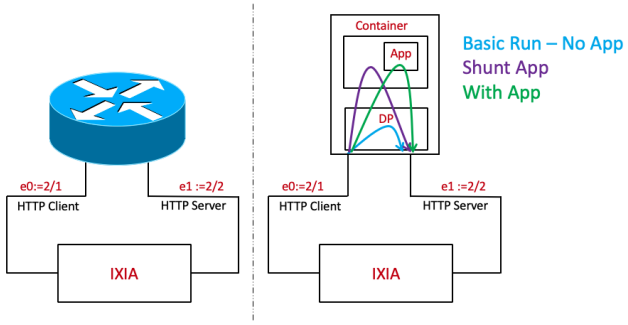


Fig. 3. Performance testing on CSR with IXIA traffic.

traffic with changing flows, observing CPU cores' utilization, per packet latency, back pressure stats, and no drop rate.

References

[1] Blake Anderson and David McGrew, "Identifying Encrypted Malware Traffic with Contextual Flow Data" in Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec '16). ACM, New York, NY, USA, 35-46, 2016.
[2] Roesch, Martin, "Snort: Lightweight intrusion detection for networks" in Lisa, vol. 99, no. 1, pp. 229-238. 1999.
[3] Fortinet Security Processors with Hardware-based Acceleration.
[4] AppWall - Web Application Firewall (WAF) from Radware.
[5] Andre Camillo, "The Current and the Future States of DNS security (2019)".
[6] Cisco Umbrella - Cloud Security Platform.
[7] X. Long, et al.,"Autonomic Networking: Architecture Design and Standardization" in IEEE Internet Computing, vol. 21, no. 05, pp. 48-53, 2017.