**AINFV: Analysis of Isolation (memory/packet) in Network Function Virtualization**

***Abstract:***

***Introduction:***

The starting days of networks, it was used to designed to send packets between two nodes. As the size of network increased, technology evolved and many network services were introduced from time to time i.e. routing, forwarding, security etc. Traditionally these network services were deployed using hardware middleboxes i.e. firewall, routers etc. These traditional hardware middleboxes are in the market for very long time and still serving their purposes. But there are disadvantages of using this approach, such as mentioned in [**Noval approach**] (a)inflexibility: unable to modify the network services, proprietary issue; (b) Non-Scalability: one needs to buy the new middlebox if the load of the network increases for certain period of time and the load stabilizes after some time, then the new middlebox is useless; and (c) Cost: expensive in terms of upgrading the network components by replacing old middleboxes with latest middleboxes to get the maximum throughput. These disadvantages encouraged the ETSI[**ref**] (European Telecommunication Standards Institute) in 2012 and the idea of NFV (Network Function Virtualization) was proposed. The idea was to replace the hardware middleboxes with software defined network services and deploy these network services as VM (Virtual Machine) on commodity servers. ETSI proposed that NFV will help the service providers as (a) swift deployment of network services; (b) comparatively cheap, by using the commodity servers; (c) more flexibility, upgrading of network service is in software. As mentioned in [**panda thesis**]NFV provide the blueprint of developing the network’s dataplane, that allows the developer to program every packet forwarding in the network. Same in SDN (Software Defined Networking), that provides the blueprint of managing the controlplane, i.e. allows developer to define the custom routing, managing network failures etc. NFV framework provides the following features[**panda thesis**]:

**-Multiplexing:** NFV framework should ensure that the NF (Network Functions) should be hardware independent, this helps in scaling of NF without changing the hardware.

**-Isolation:** NF deployed in virtualized share the under the underlying hardware, NFV framework should ensure the memory and packet isolation without affecting the performance

**-High Performance:** NF connected in series working as NF chains should have maximum throughput or equal to as of hardware middleboxes. NFV framework should ensure this throughput, as there is a major overhead of copying packets from one NF to other.

**-Efficiency:** Framework should ensure the minimal hardware utilization as the aim of NFV is to utilize the commodity servers in effective way.

**-Simplify NF Development:** Framework should ensure the simplicity in development of NF, by separating the tasks into two categories i.e. user defined functionality and preprocessing tasks. All of this should be automated.

**-Rapid Deployment:** Framework should ensure the rapid deployment by production ready NFs (i.e. NF testing and deployment in production environment on the go, to improve the performance) . This safes a lot of time.

***Problem statement***

NFV framework have many advantages but these frameworks are still long way from perfection in terms of development and deployment. For development part as addressed in [**Noval approach**] main issue is the performance trade-off due to low-level programming and optimization issues. Isolation. No standard model is defined, thus every vendor has its own programming model making NF operation complex to work in multi-tanent network environment. For deployment, current idea is to deploy NFs as VMs or Contianers to give isolation as it is main security concern. But at the cost of performance loss. Main idea is to deploy the NFs as a process instead of VMs or Containers.

***Available Solutions***

***Background***

***Requirements***

*As discussed earlier, NFV purposes is to simplify the development and deployment of NFs without changing the functionality and performance offered by traditional middleboxes. As mentioned in [***panda thesis***], there are some requirements that must be fulfiled:*

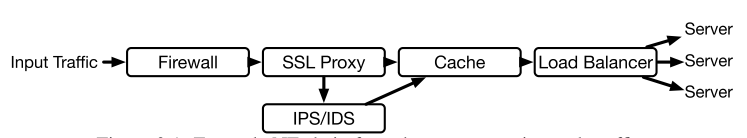
**-Performance:** Framework should not take more than 10 second of microsecond for processing packet. Single NF should be able to process 10-100Gbps of traffic. As the mentioned figures are equivalent to what we get with hardware middleboxes.

**-Efficiency:** Deployment should be done using single machine, because deployment across multiple machine will result in poor resource utilization and performance loss.

**-Chaining:** Framework should be able to combine multiple NF called chaining i.e. *NF1→NF2→....→NFn*. Packet processing starts from *NF1 to Nfn.* ***Fig1*** show the NF chain for processing web traffic.

**Multi-vendor:** NFV framework should support the multi-vendor NF to exists in a network, with security measures i.e. isolation.

**Multi-tenant:** In cloud environment , multiple tenants exists sharing the virtual resource provided by the service provider. NF should be deployed in such a way that the deployment for one tenant should not affect the operation of other tenant.



***fig1***

Mentioned requirements help in building well structured NFs and deployment ensuring the isolation. To get the further insights of NFV framework, NFV is divided into two parts, one part deals with the development model and second part deals with the execution model.

**Development model:**

Throughput and latency are two major metrics affecting the network performance. Throughput is packet processing in a given time where as latency is time between sending and receiving of packet. These two metrics depend on number of things [**noval approach**]i.e. context switching and copying, network card to cache copying, TLB (Translation Lookaside Buffer) misses and memory allocation. There are many libraries available for fast packet processing namely DPDK[**ref**] and netmap[**ref**] etc. As mentioned in [**panda thesis**] DPDK (Data Plane Development Kit) libraries provide fast packet processing mechanism by: (a) Using PMD (Poll-mode Driver) instead of depending on the CPU interrupts for acknowledgment of received packet; (b) assigning NIC (Network Interface Card) to single process instead of relying on kernel for NIC multiplexing; (c) provides the interface for connecting NIC directly to NF, instead of using intermediate elements (i.e. vSwitch) that required additional computation for packet movement. These libraries helps in improving performance and developers to focus on optimization (i.e. how the packets should be batched). The use of vectorization, as proposed in [**ref],** VPP (Vector Packet Processing) allows the processing on vectors of packets (i.e. up to 256 packets can be read at once.)

As dicussed in [**panda thesis**] The Click modular router [**ref**] based on DPDK libraries, provides the abstraction to develop new NF in such a way by combining multiple packet processing elements. But does not define the how packets flow between different elements. Click provides the limited functionality for customization. Hence for every new NF, developers have to re-write the those elements from scratch, a lot of time is spent on optimizing the elements. Development model should be modular, some module with fixed functionality and common for all NFs, whereas other modules should be user-defined for specific functionality. Developer is responsible for optimizing the user-defined modules only.

**Execution Model:**

Current practice is to deploy NFs in VMs or Containers and for communication vSwitch is used. VMs and Containers ensures the memory isolation (i.e. operation on one NF will not affect the other NF in network). vSwitch allows the NFs to periodically use the NIC for sending and receiving of packets in networks or between NFs. But all this processing of packets is just copying of packets in network and every NF has its own copy of packets that violates the packet isolation(i.e. at any point in time, only one NF should have access to that particular packet) and considerable hard to achieve. Above mentioned technologies have greater influence on performance degradation. As mentioned in [**model paper**], comparing the single process with dedicated NIC, per-core throughput drops by 3x when processing 64B packets using Containers and up to 7x while using VMs. This performance degraded further more when NFs are chained, Containers are 7x slower compared to NF chained in single process and VMs are up to 11x slower. Furthermore, NF chained single process is 6x faster than NF chained Containers or Vms, where each NF having its dedicated core.

Main reason of this performance gap is that. Firstly during packet processing, packets tends to cross the memory isolation barrier. Secondly the use of context switch that ensures that packet should cross core boundries [**model paper**].

***VNF***

***NF Chains***

***Programming VNF***

***Isolation***

***Memory Isolation: Vms/Conatainers***

***Packet Isolation: High performance I/0 libraries for packet processing zero copy Isolation***

***Proposed Framework***

***Overview***

***Main Components***

***Framework***

***Development Model***

***Execution Model***

***Testing***

***Analysis Tool***

***Test-bed***

***Achieved Results***

***Overheads***

***Framework Evaluation***

***Performance analysis of framework based different Nfs***

***Throughput/MPPS***

***Security Analysis***

***NF vs NF Isolation***

***Comparison of Frameworks***

***Based on Packet/memory Isolation***

***Discuss new & old Framework introduced***

***Conclusion drawn***

***Future Work***