Intel® Reference Architecture for NFVI Forwarding Platform on 4th Gen Intel® Xeon® Scalable Processors on Red Hat* Enterprise Linux* with vAGF Workload

Intel Accelerated Solution

Authors

Sarita Maini

Padraig Connolly

Key Contributors

Ai Bee Lim

Andrew Duignan

1 Introduction

Intel® Accelerated Solutions are configurations of hardware and software that have been optimized for and accelerated by Intel technologies to minimize the challenges of evaluation and deployment. This document describes an Intel Accelerated Solution reference architecture that utilizes the 4th Gen Intel® Xeon® Scalable processor family.

When network operators, service providers, cloud service providers, or enterprise infrastructure companies choose a Network Functions Virtualization Infrastructure (NFVI) Forwarding Platform Reference Architecture from Intel, they should be able to deploy various virtualized forwarding plane applications more quickly, securely, and effortlessly.

The solution leverages the hardened hardware, firmware, and software which allows customers to integrate on top of this known platform configuration.

1.1 NFVI Forwarding Platform

This NFVI Forwarding Platform reference architecture is defined in collaboration with Communication Service Providers and ecosystem partners to expose the value of an I/O Balanced Architecture to maximize network I/O throughput with NUMA nodes. It is an enhanced NFVI solution for 4G or 5G core User Plane Functions (UPF), broadband use cases such as virtual Broadband Network Gateway (vBNG), virtual Access Gateway Function (vAGF), Network Services such as virtual Evolved Packet Core (vEPC), IPSEC Gateway application, and cable use cases such as virtual Cable Modem Termination System (vCMTS) that have a great demand for high performance and throughput.

This workload-optimized solution is designed to minimize the challenges of infrastructure deployment and optimization for the best performance with balanced IO across sockets for core-bound as well as IO-bound workloads. It defines the software and hardware stacks and has step-by-step instructions on virtualized Access Gateway Function (vAGF) deployment and optimization as well as the throughput that can be achieved with this solution.

1.2 Terminology

TERM	DESCRIPTION
AIC	Add-In Card
API	Application Program interface
AGF	Access Gateway Function
BIOS	Basic Input/Output System
BOM	Bill of Materials
BtG	Boot Guard Technology
CUPS	Control Plane and User Plane Separation
DC	Data Center
DIMM	Dual Inline Memory Module
DPDK	Data Plane Development Kit
DRAM	Dynamic Random Access Memory
DUT	Device Under Test
FN-RG	Fixed Network – Residential Gateway
GbE	Gigabit Ethernet
HQoS	Hierarchical Quality of Service
Intel® QAT	Intel® QuickAssist Technology
Intel® TXT	Intel® Trusted Execution Technology
Intel® UPI	Intel® Ultra Path Interconnect
Intel® VT	Intel® Virtualization Technology
NFVI	Network Function Virtualization Infrastructure
NIC	Network Interface Controller
NUMA	Non-Uniform Memory Access
NVMe*	Non-Volatile Memory Express*
OAM	Operation, Administration and Management
OCP	Open Compute Project
OEM	Original Equipment Manufacturer
PCIe*	Peripheral Component Interconnect express*
QinQ	A standard that allows multiple VLAN headers in an Ethernet frame
RAS	Reliability, Availability, and Serviceability
SR-IOV	Single Root Input/Output Virtualization
SSD	Solid State Drive
TPM	Trusted Platform Module
vAGF	virtualized Access Gateway Function
vBNG	virtual Broadband Network Gateway
vCMTS	virtual Cable Modem Termination System
VIM	Virtualization Infrastructure Management
VMX	Virtual Machine Extension
VNFM	Virtual Network Function Management

Table 1. Terminology

1.3 Reference Documents and Resources

DOCUMENT	DOCUMENT NUMBER/LOCATION
Intel® Select Solutions for Network Verification Scripts	639557
VBNG-VAGF.L.22.03.0-00072.tar.gz	764478
Red Hat's Certified Guest Operating System policy	https://access.redhat.com/articles/973163
Wireline Access Evolution and 5G Fixed-Mobile Convergence	https://builders.intel.com/docs/networkbuilders/wireline-access-evolution-and-5g-fixed-mobile-convergence-1639769220.pdf
Intel® Ethernet Controller E810 Dynamic Device Personalization (DDP) Technology Guide	617015
Intel® Ethernet Controller E810 Dynamic Device Personalization Package (DDP) for Telecommunications Technology Guide	618651
System Check for Speculative Execution Side Channel	614140
RFC 2544, Benchmarking Methodology for Network Interconnect Devices	https://tools.ietf.org/html/rfc2544
RFC 1242, Benchmarking Terminology for Network Interconnection Devices	https://tools.ietf.org/html/rfc1242
RFC 6201, Device Reset Characterization	https://tools.ietf.org/html/rfc6201
Intel® Select Solution for Network Function Virtualization Infrastructure (NFVI) v3 on Red Hat* Reference Design	639782

Table 2. Reference Documents and Resources

2 Solution Components

This solution consists of select hardware and various Intel® Xeon® processor technologies along with optimized software and firmware configurations.

2.1 Intel® Xeon® Processor Scalable Performance Family

Intel® Xeon® Scalable processors are designed to accelerate performance across the fastest-growing workloads. These processors have the most built-in accelerators of any CPU on the market to help maximize performance efficiency for emerging workloads, especially those powered by AI.

In addition to delivering outstanding general-purpose performance, Intel® Xeon® drives efficiency with built-in accelerators. Data center operators can leverage built-in AI, telemetry, and power management tools to intelligently control electricity usage.

Intel's innovative workload accelerators enable end users to do more with less reducing TCO by delivering performance, power, resource, and cost efficiency as well as providing advanced security technologies.

The 4th Gen Intel® Xeon® Scalable Processors (formerly code-named Sapphire Rapids) are the latest processors for Datacenter workloads that offer:

- **Enhanced Per Core Performance** with up to 60 cores in a standard socket
- **Enhanced Memory Performance** with support for up to 4800MT/s DIMMs (2 DPC)
- Increased Memory Capacity with up to 8 channels
- **Breakthrough System Memory & Storage** with Intel® Optane™ persistent memory 200 series
- **Built-in AI Acceleration** with enhanced performance of Intel® Deep Learning Boost
- **Faster UPI** with 3 Intel® Ultra Path Interconnect (Intel® UPI) at 11.2 GT/s
- **More, Faster I/O** with PCI Express 4 and up to 64 lanes (per socket) at 16 GT/s
- **New Hardware-Enhanced Security** delivering security technologies leadership with Intel® Software Guard Extensions (Intel® SGX), Intel® Total Memory Encryption (Intel® TME), Intel® Platform Firmware Resilience (Intel® PFR) etc.
- **Enhanced Intel® Speed Select Technology (Intel® SST)** with three capabilities supported on the majority of Gold CPUs

2.2 Intel® Ethernet 800 Series

Intel® Ethernet 800 Series offers:

- **Higher Bandwidth** as Intel's first NIC with PCIe* 4.0 and 50Gb PAM4 SerDes
- **Improved Application Efficiency** with Application Device Queues (ADQ), Dynamic Device Personalization (DDP)
- **Versatility** with Flexible speeds: 2x100/50/25/10GbE, 4x25/10GbE, or 8x10GbE
- **RDMA support** for both iWARP and RoCEv2 providing a choice in hyper-converged networks

2.2.1 Intel® Ethernet Network Adapter E810 Drivers: In-tree vs. Out-of-tree

Generally, the NFVI Forwarding Platform reference architecture recommends in-tree Intel® Ethernet Adapter E810 ice/iavf drivers and DDP components. However, the Ethernet Out-of-Tree (OOT) drivers often contain support for new features and fixes to known issues. For example, the features such as Rate limiting, ADQ and eDDP (enhanced Dynamic Device Personalization) are not presently supported in the in-tree driver for the E810 NICs.

The virtual Broadband Network Gateway being used as a workload in this reference architecture requires the OOT driver due to limitations in in-tree driver support. Intel continues to work with Red Hat* to add support for these types of functions in the Red Hat* Open Stack. Refer to https://access.redhat.com/articles/1067 which explains Red Hat* Support policy for Out of Tree (OOT) drivers.

The E810 Drivers can be found at the following locations:

DRIVER	OOT VERSION	LOCATION
ice	1.9.11	https://www.intel.com/content/www/us/en/download/19630/intel-network-adapter-driver-for-e810-series-devices-under-linux.html
NVM	3.0	CVL3.0 Sampling/NVMUpdatePackage https://www.intel.com/content/www/us/en/download/19626/non-volatile-memory-nvm-update-utility-for-intel-ethernet-network-adapters-e810-series-linux.html

2.2.2 Intel® Network Adapters with Data Plane Development Kit (DPDK)

Intel® Network Products deliver continuous innovation for high throughput and performance for networking infrastructure. The Intel® Network Adapter with Data Plane Development Kit (DPDK) provides highly optimized Network Virtualization and fast data path packet processing. DPDK offers many use cases that are hardened on this NFVI Forwarding Platform.

2.2.3 Intel® Ethernet 800 Series Dynamic Device Personalization (DDP)

Dynamic Device Personalization (DDP) usage to reconfigure network controllers for different network functions on-demand, without the need for migrating all VMs from the server, avoids unnecessary loss of compute for VMs during server cold restart. It also improves packet processing performance for applications/VMs by adding the capability to process new protocols in the network controller at run-time.

This kind of on-demand reconfiguration is offered in the Intel® Ethernet 800 Series NICs.

DDP describes the capability of Intel® Ethernet 800 Series devices to load an additional firmware profile on top of the device's default firmware image, enabling parsing and classification of additional specified packet types that can be distributed to specific queues on the NIC's host interface using standard filters. Software applies these custom profiles in a non-permanent, transaction-like mode so that the original network controller's configuration is restored after NIC reset or by rolling back profile changes by software. Using APIs provided by drivers, personality profiles can be applied by the DPDK. Support for kernel drivers and integration with higher level management/orchestration tools is in progress.

DDP can be used to optimize packet processing performance for different network functions, native or running in virtual environment. By applying a DDP profile to the network controller, the following use cases could be addressed.

A general purpose, OS-default DDP package is automatically installed with all supported Intel® Ethernet Controller 800 Series drivers on Microsoft* Windows*, ESX*, FreeBSD*, and Linux* operating systems. Additional DDP packages are available to address needs for specific market segments. For example, a telecommunications (Comms) DDP package is available to support certain market-specific protocols in addition to the protocols in the OS-default package.

- The OS-default DDP package supports the following:
  - MAC, EtherType, VLAN
  - IPv4, IPv6, TCP, ARP, UDP
  - SCTP, ICMP, ICMPv6, CTRL
  - LLDP, VXLAN-GPE, VXLAN (non-GPE), Geneve, GRE, NVGRE, RoCEv2
  - MPLS (up to 5 consecutive MPLS labels in the outermost Layer 2 header group)
- In addition to the previous list, the Comms DDP package also supports the following protocols:
  - GTP
  - PPPOE
  - L2TPv3
  - IPSec
  - PFCP

3 NFVI Forwarding Platform Reference Architecture Requirements

The primary focus of this reference architecture is to provide details of the customized NFVI Forwarding Platform configuration along with supporting the performance data for a high performance/throughput workload such as vAGF.

This chapter also focuses on the design requirements for this NFVI Forwarding Platform solution.

3.1 Reference Architecture Hardware Requirements

The checklist in the table below is a guide for assessing the conformance to the NFVI Forwarding Platform hardware platform requirement for the vAGF Configuration.

For the platform to conform, all requirements listed in the checklist below must be satisfied.

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED	QUANTITY PER SERVER
Processor	Option 1: 4th Gen Intel® Xeon® Platinum 8470N Processor at 1.7 GHz, 52C/104T, 300W Option 2: 4th Gen Intel® Xeon® Gold 6428N at 1.8 GHz, 32C/64T, 185WOption 3: 4th Gen Intel® Xeon® Gold 6438N at 2.0 GHz, 32C/64T, 205W	Required	2
Memory	Option 1: DRAM only configuration: 512 GB (16 x 32 GB DDR5, 4800 MHz)	Required	16
	Option 2: DRAM only configuration: 256 GB (16 x 16 GB DDR5, 4800 MHz)		16
Network	Intel® Ethernet Network Adapter E810-2CQDA2	Required	4
Storage (Boot Drive)	Intel® SATA Solid State Drive D3 S4510 or higher at 480 GB or larger boot drive	Required	1
LAN on Motherboard (LOM) or NIC	1/10 Gbps port for Internet access	Required	1

Table 3. NFVI Forwarding Platform - vAGF HW Configuration

3.2 Reference Architecture Software Requirements

The table below is a guide for assessing the conformance to the NFVI Forwarding Platform software requirements.

For the platform to conform, all requirements listed in the checklist below must be satisfied.

	INGREDIENT	SW VERSION DETAILS	
Firmware	BIOS MCU	EGSDCRB1.86B.8901.P01.2209200239 0xab0000c0	
	Intel® Ethernet Network Adapter E810-2CQDA2	ice driver: 4.18.0-372.16.1.el8_6.x86_64 firmware-version: 2.40 ice driver: 1.9.11 firmware-version: 3.00	
System Under Test	OS	Red Hat* Enterprise Linux* release 8.6 (Ootpa)	4.18.0-372.16.1.el8_6.x86_64
	APPs/Libraries	vAGF	22.03
		DPDK	20.11.5
		ICE COMMS Package version	DDP ICE COMMS 1.3.31.0

Table 4. NFVI Forwarding Platform – vAGF SW Configuration

Note: The software versions listed in the previous table are minimum requirements. It is recommended to use the latest version if available. This is a hardened software stack that has gone through verification.

3.3 BIOS Settings

To meet the performance requirements for the NFVI Forwarding Platform solution, the following BIOS settings in Table 5 provide guidance for optimized settings with 4th Gen Intel® Xeon® Scalable processors.

MENU (ADVANCED)	PATH TO BIOS SETTINGS	BIOS SETTINGS	REQUIRED SETTING FOR DETERMINISTIC PERFORMANCE
Socket Configuration	IIO Configuration -> Socket 0 Configuration ->IIO Pcie Ports 1 thru 7	IOU0 to IOU7	<x_x8x_x8>
Socket Configuration	IIO Configuration -> Socket 1 Configuration ->IIO Pcie Ports 1 thru 7	IOU0 to IOU7	<x_x8x_x8>
Socket Configuration	Advanced Power Management Configuration -> CPU P State Control	SpeedStep (Pstates)	Disable
Socket Configuration	Advanced Power Management Configuration -> CPU P State Control	Activate SST-BF	Disable
Socket Configuration	Advanced Power Management Configuration -> CPU P State Control	Energy Efficient Turbo	Disable
Socket Configuration	Advanced Power Management Configuration -> Hardware PM State Control	Hardware P-States	Native Mode
Socket Configuration	Advanced Power Management Configuration -> CPU C State Control	Enable Monitor MWait	Auto
Socket Configuration	Advanced Power Management Configuration -> CPU C State Control	CPU C1 auto demotion	Disable
Socket Configuration	Advanced Power Management Configuration -> CPU C State Control	CPU C1 auto undemotion	Disable
Socket Configuration	Advanced Power Management Configuration -> Package C State Control	Package C State	C0/C1 state
Socket Configuration	Advanced Power Management Configuration -> CPU - Advanced PM Tuning	Uncore Freq Scaling	Enable

Socket Configuration	Advanced Power Management Configuration -> CPU - Advanced PM Tuning	Uncore Freq RAPL	Disable
Socket Configuration	Advanced Power Management Configuration -> CPU - Advanced PM Tuning -> Energy Perf BIAS	Power Performance Tuning	BIOS Controls EPB
Socket Configuration	Advanced Power Management Configuration -> CPU - Advanced PM Tuning -> Energy Perf BIAS	ENERGY_PERF_BIAS_ CFG Mode	Performance
Socket Configuration	Advanced Power Management Configuration -> CPU - Advanced PM Tuning -> Energy Perf BIAS	Workload Configuration	I/O Sensitive

Table 5. Platform BIOS Settings

Note: Some servers may not provide the BIOS options as documented in the table above.

3.4 Platform Technology Requirements

This section lists the requirements for Intel's advanced platform technologies.

NFVI requires Intel® VT and Intel® Scalable I/O Virtualization (Intel® Scalable IOV) to be enabled to reap the benefits of hardware virtualization. Either Intel® Boot Guard or Intel® Trusted Execution Technology establishes the firmware verification, allowing for platform static root of trust.

PLATFORM TECHNOLOGIES		ENABLE/DISABLE	REQUIRED/RECOMMENDED
Intel® VT	Intel® CPU VMX Support	Enable	Required
	Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d)	Enable	Required
	Single Root I/O Virtualization (SR-IOV)	Enable	Required
Intel® Boot Guard	Intel® Boot Guard	Enable	Recommended
Intel® TXT	Intel® Trusted Execution Technology	Enable	Recommended

Table 6. Platform Technology Requirements

3.5 Platform Security

This NFVI solution must implement and enable Intel® Boot Guard Technology to ensure that the platform firmware is verified to be suitable in the boot phase.

In addition to protecting against the known attacks, Intel recommends installing the Trusted Platform Module (TPM). The TPM enables administrators to secure platforms for a trusted (measured) boot with known trustworthy (measured) firmware and OS, as well as enabling local and remote verification by third parties to advertise such known safe conditions for these platforms with the implementation of Intel® Trusted Execution Technology (Intel® TXT).

4 Virtual Access Gateway Function (vAGF)

4.1 vAGF Overview

The Access Gateway Function (AGF) is a function that provides connectivity from a wireline Access Network to the 5G Core Network. Access Gateway Function (AGF) is the access point for subscribers, through which they connect to the Internet and private networks. It provides critical subscriber management functions, such as authentication, IP address assignment, bandwidth allocation and accounting.

When a connection is established between the Customer Premises Equipment (CPE) and the AGF network function, the subscriber can access the broadband services provided by the telecom operator or Internet Service Provider (ISP). The role of the AGF is to aggregate traffic from various subscriber sessions from an access network and route it to the network of the service provider.

Since the subscriber directly connects to the edge router, vAGF effectively manages subscriber access and subscriber management functions such as:

- Authentication, authorization, and accounting (AAA) of subscriber sessions
- IP Address assignment
- Security
- Policy management
- Quality of Service (QoS) and Traffic Management

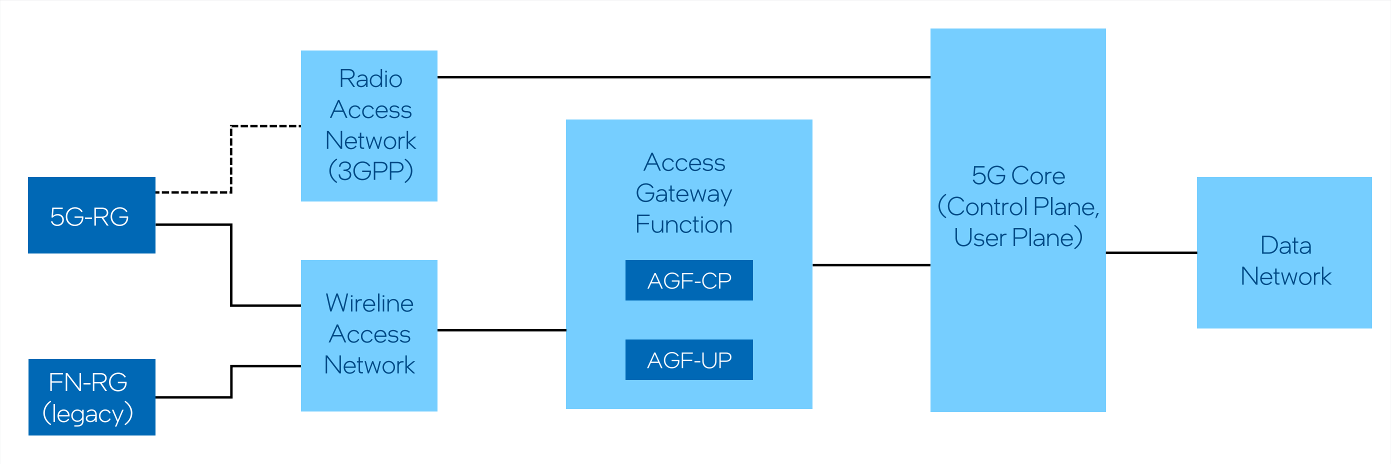


Figure 1. vAGF Overview

The vAGF is a virtualized software instantiation of what is typically a large ASIC-based fixed-function appliance usually located in a central office or metro Point of Presence (PoP). The vAGF is implemented as a set of Virtual Network Function (VNF) instances with each instance supporting a single Subscriber Service-Group, which typically contains hundreds of home routers/subscribers.

4.2 vAGF Pipeline

The vAGF Data Plane (DP), or User Plane (UP), is built around two packet processing pipelines - uplink (UL) and downlink (DL) - described below. The uplink data plane manages the flow of traffic from the end user's Customer Premises Equipment (CPE) to the core network. The downlink data plane handles the flow of traffic and data from the core network network to the end user. It manages and schedules traffic to users attached to the AGF.

The vAGF DP has been implemented using the FD.io VPP framework and run as a containerized application using Kubernetes as an orchestrator.

This reference architecture focuses on the data plane functions since the main goal is to show maximization of I/O for intense workloads like vAGF.

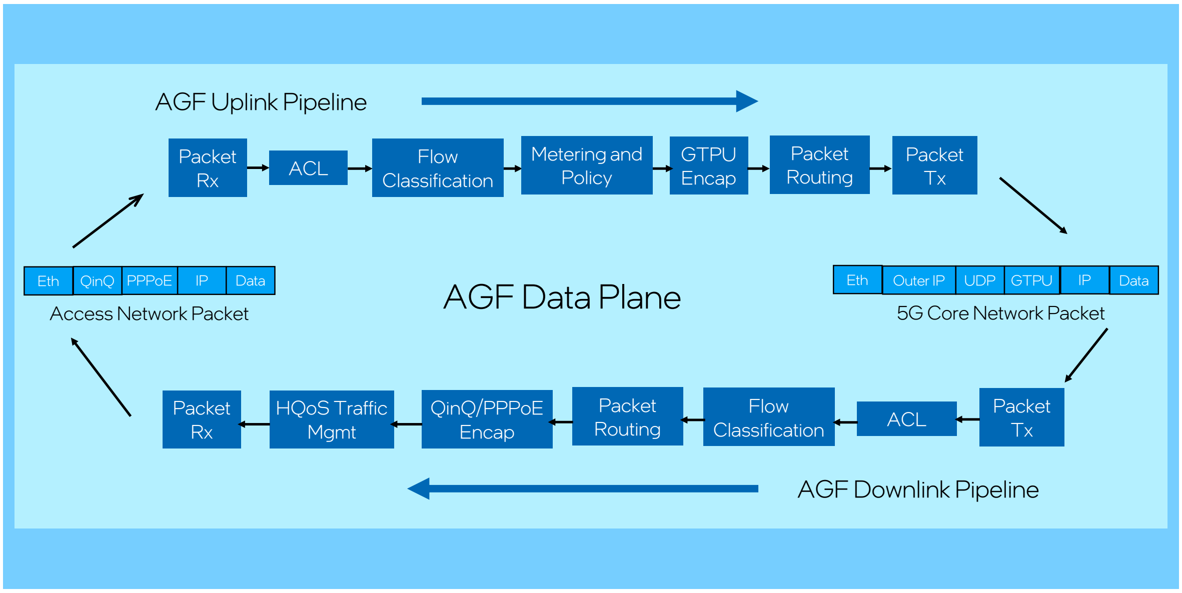


Figure 2. AGF Data Plane

4.2.1 Uplink Pipeline Overview

The reference implementation of the AGF uplink packet processing pipeline consists of the following functions:

Packet Rx (Receive): Packets from the wireline subscriber access network are received from the Network Interface Controller (NIC) ports using DPDK PMD drivers and sent to the next stage to begin packet processing.

Firewall/Access Control List (ACL): This stage employs an Access Control List (ACL) table to implement firewall policies (i.e., block rules) on the incoming traffic. This blocklist firewall has 150 block (random) rules. Table lookup operation is performed on each received packet, and in the case of rule match, the packet is dropped.

Flow Classification: This stage classifies each subscriber flow based on the source MAC address and double VLAN tags and strips the Q-in-Q header (and PPPoE header if PPPoE subscriber traffic is enabled).

Flow Metering and Policing: This function meters the subscriber traffic flows to determine the compliance with a service contract and applies traffic policing to enforce the contract. As a result, packets that conform to a specified rate are sent to the next stage of the pipeline while packets that violate the rate are dropped.

GTPu Encapsulation: At this stage, a GTP-U header is added to the IP packet

Routing: At this stage, an Ethernet header is added based on the route

Packet Tx (Transmit): Finally, the packets are sent out to the core network. With the help of DPDK poll mode drivers, packets are transmitted out of the system through the NIC ports connected to wireline core network.

4.2.2 Downlink Pipeline Overview

The reference implementation of the vAGF Downlink packet processing pipeline consists of the following functions. The incoming downlink packet typically consists of an Ethernet frame with IP/UDP header. The outbound traffic will be an Ethernet frame with encapsulated QinQ VLAN and IP/UDP headers.

Packet Rx (Receive): Packets from the core network are received from the Network Interface Controller (NIC) ports using DPDK PMD drivers and sent to the next stage to begin packet processing.

Firewall/Access Control List (ACL): This stage employs an access control list (ACL) table to implement firewall policies (i.e., allow rules) on the incoming traffic. Table lookup operation is performed on each received packet, and in the case of rule match, the packet is permitted to the next stage. The Allow list firewall has 4K allow (subscriber flow) rules.

Flow Classification: This stage performs exact-match classification on the 5-tuple header fields (inner IPv4 source and destination IP address, UDP source and destination ports and IP transport layer protocol ID) of the input packets to identify the session and stores the session info as packet metadata to be used later in the pipeline. In addition to this, access network encapsulations are stripped off the packets at this stage. It first strips the GTP-U header and then classifies each subscriber flow based on the 5-tuple header.

QinQ/Q-in-Q+PPPoE Encapsulation and Routing: At this stage, packets are encapsulated with a QinQ header added to the inner IPv4 packet based on the flow ID and (and PPPoE header if enabled) and routed to the access network via the correct network interface port.

Hierarchical QoS Traffic Management: Each packet runs through a hierarchical QoS (HQoS) scheduler to ensure that thousands of subscribers can get the desired broadband capacity as per the service contract. It implements a 4-level HQoS with one pipe per subscriber (configured to allow all traffic to pass)

Packet Tx (Transmit): With the help of DPDK poll mode drivers, packets are transmitted out of the system through the NIC ports connected to access network. Downlink Traffic Profile

4.3 vAGF Test Setup

The Intel® AGF Data Plane Package can be used to install multiple instances of a vAGF data plane reference application in a Linux Container environment on an Intel® Xeon® server.

The application and environment can be used to evaluate the performance of a vAGF data plane on Intel® Xeon® based platforms. This is a POC evaluation application only and is neither intended nor is fully featured, hardened, or secured. Deploy in an isolated evaluation environment only.

Source code and build instructions are provided for the vAGF data plane application, and to set up an environment for traffic-generation and performance analysis of the application.

Figure 3 shows the test setup with multiple Physical Functions (PFs) split into Virtual Functions (VFs) that connect to each vAGF instances on both the sockets. The System Under Test (SUT) is connected to the Traffic generator. Ixia IxNetwork is used to generate the L2-3 traffic required to benchmark the vAGF. Each 100GbE port on Ixia is connected directly to a 100GbE port on the SUT. The vAGF container instances are NUMA-optimized, ensuring that the cores and VFs used in a specific vAGF instance are in a processor and a NIC in the same socket.

Ixia IxNetwork is configured for both Uplink and Downlink Traffic Flows in a single port.

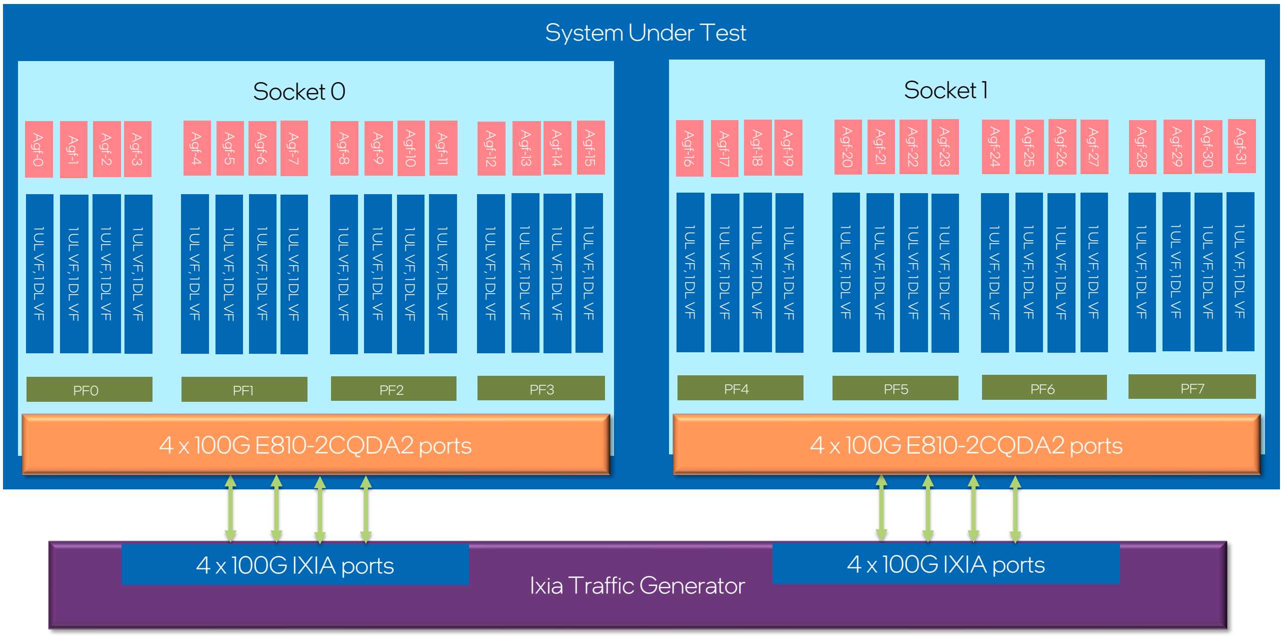
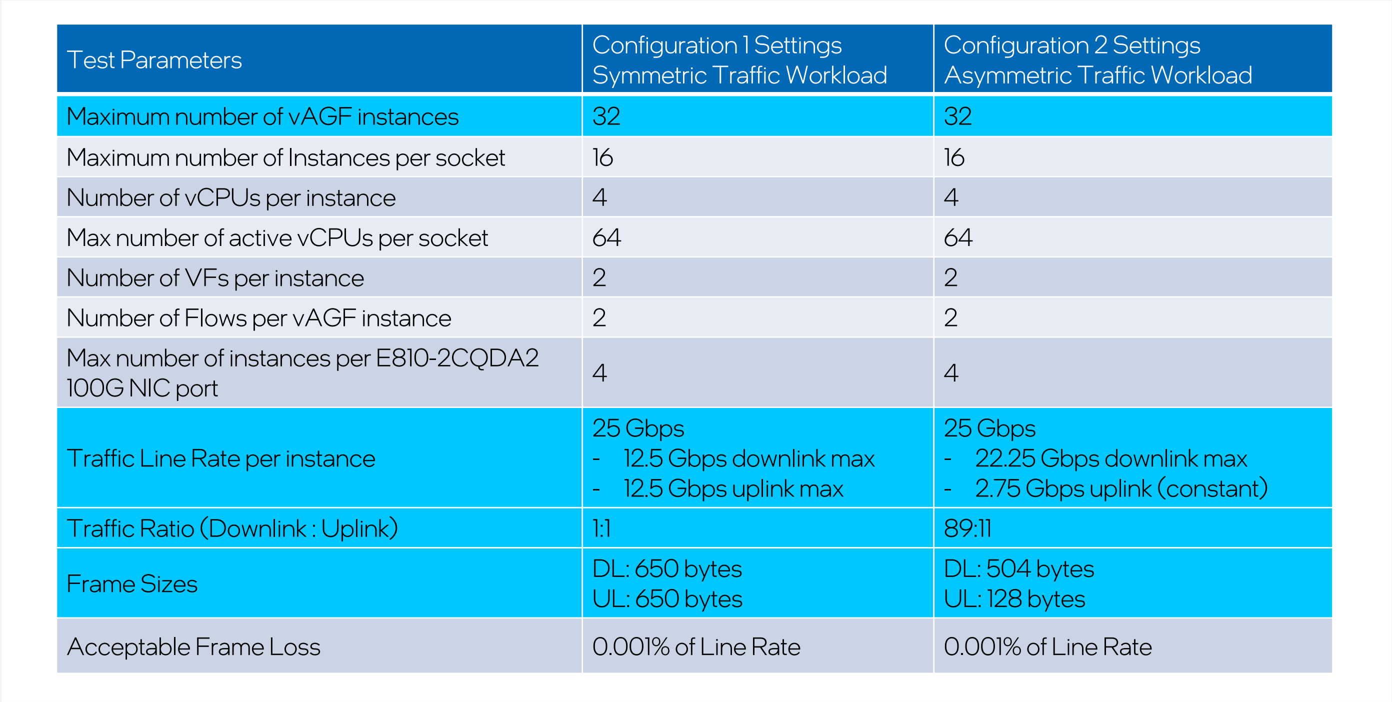


Figure 3. vAGF Test Setup

Detailed steps on configuring the Ixia traffic generator for benchmarking vAGF is described in Section 6 – Traffic Generator Configuration Guide.

The Intel® reference AGF application is benchmarked to understand how the architecture works in a real context. To get a more comprehensive view, two different configurations have been benchmarked - a symmetric traffic workload and an asymmetric traffic workload.

The following table provides more key information points about the benchmarking setup for these two test configurations.



Test Parameters	Configuration 1 Settings Symmetric Traffic Workload	Configuration 2 Settings Asymmetric Traffic Workload
Maximum number of vAGF instances	32	32
Maximum number of Instances per socket	16	16
Number of vCPUs per instance	4	4
Max number of active vCPUs per socket	64	64
Number of VFs per instance	2	2
Number of Flows per vAGF instance	2	2
Max number of instances per E810-2CQDA2 100G NIC port	4	4
Traffic Line Rate per instance	25 Gbps - 12.5 Gbps downlink max - 12.5 Gbps uplink max	25 Gbps - 22.25 Gbps downlink max - 2.75 Gbps uplink (constant)
Traffic Ratio (Downlink : Uplink)	1:1	89:11
Frame Sizes	DL: 650 bytes UL: 650 bytes	DL: 504 bytes UL: 128 bytes
Acceptable Frame Loss	0.001% of Line Rate	0.001% of Line Rate

Table 7. Symmetric vs Asymmetric Traffic Workloads

Each vAGF container instance has both a downlink and uplink pipeline. Each 100 Gbps Intel® Ethernet E810-2CQDA2 Port has 4 vAGF instances running on it, each served with 25 Gbps.

5 vAGF Installation Guide

5.1 vAGF Application Server Preparation

On the vAGF server, perform the following steps to prepare the server for the installation of vAGF.

5.2 System BIOS Settings

Configure the following within the system BIOS settings:

1. Enable Intel® Hyper-Threading Technology (Intel® HT Technology).
2. Enable Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d) (SR-IOV).
3. Set CPU Power and Performance profile to "Performance".
4. Disable Intel® Turbo Boost Technology.
5. Disable Energy Efficient Turbo.
6. Disable Enhanced Intel SpeedStep® Technology.

Refer to Table 5 for optimized BIOS settings for this workload

Note: Turbo-boost, P-States, and C-States can be controlled on a per core basis using the Intel® power-management tools available from https://github.com/intel/CommsPowerManagement.

5.3 RHEL* 8.6 Installation

1. Download the RHEL 8.6 Binary DVD from the Red Hat* Enterprise Linux* website.
2. Use the *.iso file to create bootable USB installation media and install the USB drive onto a USB port on the server.
3. Boot into the RHEL 8.6 installer. Select "Install Red Hat* Enterprise Linux*".
4. Select the appropriate options for Language, Keyboard layout, time zone and Network settings, SSD, partitioning, root password, create a user etc.

5.4 Install vAGF Dependencies

1. Install the following package dependencies:

```
# subscription-manager repos --enable=rhel-8-for-x86_64-appstream-rpms --enable=rhel-8-for-x86_64-baseos-rpms
# yum groupinstall “Development Tools”
# yum install -y kernel-tools numactl numactl-devel libvirt-devel socat python38 kernel-devel-$(uname -r)
# yum install -y podman-docker
# pip3 install fabric -U --force-reinstall
# pip3 install paramiko -U --force-reinstall
# pip3 install cryptography -U --force-reinstall
# yum install elfutils-libelf-devel
```

Note: For pip3, ensure that all_proxy and/or socks_proxy environment variables are not set. If either of these environment variables are set, pip will generate an error related to missing dependencies for SOCKS support.

2. Install the Intel® ice network adapter driver for PCIe:

   Download ice driver out-of-tree version 1.9.11

```
# tar -xvf ice-1.9.11.tar.gz
# cd ice-1.9.11/src/
# make
# make install
# modprobe -r ice; modprobe ice
```

3. Download the Comms DDP package that enables the NIC to parse extra header fields such as PPPoE and GTPu, from this link

```
# unzip '800 Series DDP Comms Package 1.3.31.0.zip'
# unzip ice_comms-1.3.31.0.zip -d ice-ddp-comms
# cd ice-ddp-comms/
# cp ice_comms-1.3.31.0.pkg /lib/firmware/updates/intel/ice/ddp/
# cp ice.pkg /lib/firmware/updates/intel/ice/ddp/
# modprobe -r ice
# modprobe  ice
```

   To confirm that the DDP ICE COMMS Package version 1.3.31.0 is successfully loaded:

```
# dmesg | grep COMMS
```

4. Configure the Linux* Kernel settings.

   The following should be configured in Linux* Kernel Grub settings:

   Set huge-page memory size and number of pages for DPDK app:

   (default_hugepagesz=2M hugepagesz=2M hugepages=30024)

   Enable SR-IOV (intel_iommu=on iommu=pt).

   Disable hardware control of P-states (intel_pstate=disable)

   Isolate vAGF dataplane cores from the Linux* kernel task scheduler.

   For example: `isolcpu = isolcpus=3-31,35-63,67-95,99-127`).

Note: The `isolcpus` settings will need to be adapted to the specific core layout of the CPU.

5. Run the DPDK utility to verify logical core layout for CPU socket 0 and socket 1

```
# $DPDK_ROOT_RELEASE/usertools/cpu_layout.py
```

6. Compile the GRUB configuration and reboot the server for these settings to take effect:

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
# reboot
```

5.5 vAGF Dataplane Application Environment Set-up

7. Copy Intel’s vBNG-vAGF Reference Architecture package to the root directory of the target vAGF server and extract files as root user:

```
# cp VBNG-VAGF.L.22.03.0-00072.tar.gz  /root
# cd /root
# tar -zxvf VBNG-VAGF.L.22.03.0-00072.tar.gz
```

The vAGF application and environment files are extracted to the /root/AGF directory

8. Start podman (Linux* container engine).

```
# systemctl disable firewalld
# systemctl stop firewalld
# systemctl start podman
```

9. Build the vf-init podman image:

Note: VF-Init is used to program the E810 eSwitch using the new DCF PMD technology

a. Move to the VF-Init directory:

```
# cd AGF/vf-init
```

b. Build the VF-Init podman image

```
# podman build --tag vf-init:22.03 . --build-arg http_proxy=$http_proxy --build-arg https_
proxy=$https_proxy
```

10. Build the vAGF container:

a. Move to the AGF/vAGF directory (From VF-Init directory):

```
# cd ../vAGF
```

b. In AGF/vAGF/Dockerfile, add this line after line 41, to increase the number of logical cores supported to 512:

```
RUN sed -i '/-Db _ pie=true*/a \ \ \ \ \ \ \ \ -Dmax _ lcores=512 \\' /opt/vagf _ vpp/build/
external/packages/dpdk.mk
```

c. Using podman build the vAGF image:

```
# podman build --tag vagf:22.03 . --build-arg http_proxy=$http_proxy --build-arg https_
proxy=$https_proxy
```

11. Create VFs:

a. Check what available Intel® Ethernet Network Adapter E810-2CQDA2 resources you have on your vAGF server:

```
# lshw -c network -businfo | grep E810
```

For example, suppose your output was as follows:

```
pci@0000:86:00.0  ens13f0      network     Ethernet Controller E810-C for QSFP
pci@0000:86:00.1  ens13f1      network     Ethernet Controller E810-C for QSFP
pci@0000:d8:00.0  ens21f0      network     Ethernet Controller E810-C for QSFP
pci@0000:d8:00.1  ens21f1      network     Ethernet Controller E810-C for QSFP
```

You should use a PF from each Bus Pool (For E810-2CQDA2) thus to create your VFs it should be:

```
# echo 10 > /sys/class/net/ens13f0/device/sriov_numvfs
# echo 10 > /sys/class/net/ens21f0/device/sriov_numvfs
```

12. Bind VFs to DPDK:

a. Download igb_uio kmod:

```
# git clone http://dpdk.org/git/dpdk-kmods && cd dpdk-kmods/linux/igb_uio/
# make
# modprobe uio && insmod igb_uio.ko
```

b. Download and un-tar DPDK:

```
# cd
# wget https://fast.dpdk.org/rel/dpdk-20.11.5.tar.xz && tar -xvf dpdk-20.11.5.tar.xz
```

c. Bind the required VFs to DPDK:

```
# ./dpdk-stable-20.11.5/usertools/dpdk-devbind.py -b igb_uio $(lshw -c network -businfo | grep
-v "v1 " | grep "Virtual Function" | awk '{print $2}')
```

d. Enable trust mode on the VF 0 of each PF that you will be using

```
# ip link set ens13f0 vf 0 trust on
# ip link set ens21f0 vf 0 trust on
```

13. Make a note of the cpus to assign to your workloads later on:
Use cpu_layout.py to determine core enumeration and their hyper-threaded pair.

```
./dpdk-stable-20.11.5/usertools/cpu_layout.py
```

```
Example for a 32-core processor:
Core and Socket Information (as reported by '/sys/devices/system/cpu')
   ======================================================================

cores =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31]
sockets =  [0, 1]

             Socket 0            Socket 1
Core 0       [0, 64]             [32, 96]
Core 1       [1, 65]             [33, 97]
Core 2       [2, 66]             [34, 98]
Core 3       [3, 67]             [35, 99]
Core 4       [4, 68]             [36, 100]
Core 5       [5, 69]             [37, 101]
Core 6       [6, 70]             [38, 102]
Core 7       [7, 71]             [39, 103]
Core 8       [8, 72]             [40, 104]
Core 9       [9, 73]             [41, 105]
Core 10      [10, 74]            [42, 106]
Core 11      [11, 75]            [43, 107]
Core 12      [12, 76]            [44, 108]
Core 13      [13, 77]            [45, 109]
Core 14      [14, 78]            [46, 110]
Core 15      [15, 79]            [47, 111]
Core 16      [16, 80]            [48, 112]
Core 17      [17, 81]            [49, 113]
Core 18      [18, 82]            [50, 114]
Core 19      [19, 83]            [51, 115]
Core 20      [20, 84]            [52, 116]
Core 21      [21, 85]            [53, 117]
Core 22      [22, 86]            [54, 118]
Core 23      [23, 87]            [55, 119]
Core 24      [24, 88]            [56, 120]
Core 25      [25, 89]            [57, 121]
Core 26      [26, 90]            [58, 122]
Core 27      [27, 91]            [59, 123]
Core 28      [28, 92]            [60, 124]
Core 29      [29, 93]            [61, 125]
Core 30      [30, 94]            [62, 126]
Core 31      [31, 95]            [63, 127]
```

14. Run VF-Init Instance:

```
# podman run -d --network="host" --privileged=true -v /dev/hugepages:/dev/hugepages -v /lib/firmware/
updates/intel/ice/ddp/:/lib/firmware/updates/intel/ice/ddp/  --cpuset-cpus=0,1 -env PCIDEVICE _ INTEL _
COM _ INFRA _ DCF=0000:16:01.0,0000:19:01.0,0000:27:01.0,0000:2a:01.0,0000:a8:01.0,0000:ab:01.0,0000:b8:01.
0,0000:bb:01.0 --env MY _ POD _ IP=localhost --name=vf-init-0 -h vf-init-0 -it vf-init:22.03
```

a. Once the VF-Init instance is running, we now need to program the rules, this can be done using telnet by running the following:

```
# telnet localhost 8152
# vf-init>
# vf-init> set mac 00:00:00:01:WW:XX Y Z
```

Where:

- WW is replaced by instance ID
- XX is replaced by VF type (00 for Uplink and 01 for Downlink)
- Y is replaced by VF ID
- Z is replaced by Port ID

Examples:

Uplink 0000:86:01.0:

```
# vf-init> set mac 00:00:00:01:00:00 2 0
# vf-init> set mac 00:00:00:01:01:00 3 0
# vf-init> set mac 00:00:00:01:02:00 4 0
# vf-init> set mac 00:00:00:01:03:00 5 0
```

Downlink 0000:86:01.0:

```
# vf-init> set mac 00:00:00:01:00:01 6 0
# vf-init> set mac 00:00:00:01:01:01 7 0
# vf-init> set mac 00:00:00:01:02:01 8 0
# vf-init> set mac 00:00:00:01:03:01 9 0
```

Uplink 0000:d8:01.0:

```
# vf-init> set mac 00:00:00:01:04:00 2 1
# vf-init> set mac 00:00:00:01:05:00 3 1
# vf-init> set mac 00:00:00:01:06:00 4 1
# vf-init> set mac 00:00:00:01:07:00 5 1
```

Downlink 0000:d8:01.0

```
# vf-init> set mac 00:00:00:01:04:01 6 1
# vf-init> set mac 00:00:00:01:05:01 7 1
# vf-init> set mac 00:00:00:01:06:01 8 1
# vf-init> set mac 00:00:00:01:07:01 9 1
```

15. Run vAGF docker instances (Uplink and Downlink are in same container).

```
# podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--cpuset-cpus=<CLI-MGMT-CPU-ID>,<DL-CPU2-ID>,<UL-CPU-ID>,<DL-CPU-ID> \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:<UL-VF-PCI-ADDRESS> \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:<DL-VF-PCI-ADDRESS> \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=<SYSTEM-HOSTNAME> \
--env ETCD _ ENABLED=false --name=agf-<INST-ID> -h agf-<INST-ID> \
-it vagf:22.03
```

Where:

- `<CLI-MGMT-CPU-ID>` is replaced by an isolated core (Use step 7 above to figure this out)
- `<DL-CPU2-ID>` is replaced by an isolated vCPU (Use step 7 above to figure this out)
- `<UL-CPU-ID>` is replaced by an isolated vCPU (Use step 7 above to figure this out)
- `<DL-CPU-ID>` is replaced by an isolated vCPU (Use step 7 above to figure this out) (Make sure this is on the same Physical core as `<DL-CPU2-ID>` or else performance will degrade significantly)
- `<UL-VF-PCI-ADDRESS>` is replaced by a data plane VF (Make sure this aligns with the rules set in step 8)
- `<DL-VF-PCI-ADDRESS>` is replaced by a data plane VF (Make sure this aligns with the rules set in step 8)
- `<SYSTEM-HOSTNAME>` is replaced by the hostname of your vAGF server
- `<INST-ID> is replaced by the instance ID of the container`
  - Y is replaced by VF ID
  - Z is replaced by Port ID

Note: Make sure all VFs and CPU-IDs are on the same NUMA node or else performance will significantly degrade, you can check the NUMA Node of a VF using this command:

```
# cat /sys/bus/pci/devices/0000\:86\:01.2/numa _ node
```

Examples for a 52-core processor:

```
# podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=2,3,106,107 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:16:01.2 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:16:01.6 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-0 -h agf-0 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=4,5,108,109 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:16:01.3 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:16:01.7 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-1 -h agf-1 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=6,7,110,111 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:16:01.4 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:16:02.0 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-2 -h agf-2 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=8,9,112,113 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:16:01.5 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:16:02.1 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-3 -h agf-3 \
-it localhost/vagf:22.03
sleep 10
podman ps -a
podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=10,11,114,115 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:19:01.2 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:19:01.6 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-4 -h agf-4 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=12,13,116,117 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:19:01.3 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:19:01.7 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-5 -h agf-5 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=14,15,118,119 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:19:01.4 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:19:02.0 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-6 -h agf-6 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=16,17,120,121 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:19:01.5 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:19:02.1 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-7 -h agf-7 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=18,19,122,123 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:27:01.2 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:27:01.6 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-8 -h agf-8 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=20,21,124,125 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:27:01.3 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:27:01.7 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-9 -h agf-9 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=22,23,126,127 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:27:01.4 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:27:02.0 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-10 -h agf-10 \
-it localhost/vagf:22.03
sleep 10
podman ps -a
podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=24,25,128,129 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:27:01.5 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:27:02.1 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-11 -h agf-11 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=26,27,130,131 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:2a:01.2 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:2a:01.6 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-12 -h agf-12 \
-it localhost/:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=28,29,132,133 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ UL=0000:2a:01.3 \
--env PCIDEVICE _ INTEL _ COM _ APP _ AGF _ UP _ DL=0000:2a:01.7 \
--env MY _ POD _ IP=localhost --env MY _ NODE _ NAME=arch5-hddc301 \
--env ETCD _ ENABLED=false --name=agf-13 -h agf-13 \
-it localhost/vagf:22.03
sleep 10
podman ps -a
podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=30,31,134,135 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_UL=0000:2a:01.4 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_DL=0000:2a:02.0 \
--env MY_POD_IP=localhost --env MY_NODE_NAME=arch5-hddc301 \
--env ETCD_ENABLED=false --name=agf-14 -h agf-14 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=32,33,136,137 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_UL=0000:2a:01.5 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_DL=0000:2a:02.1 \
--env MY_POD_IP=localhost --env MY_NODE_NAME=arch5-hddc301 \
--env ETCD_ENABLED=false --name=agf-15 -h agf-15 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=54,55,158,159 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_UL=0000:a8:01.2 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_DL=0000:a8:01.6 \
--env MY_POD_IP=localhost --env MY_NODE_NAME=arch5-hddc301 \
--env ETCD_ENABLED=false --name=agf-16 -h agf-16 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=56,57,160,161 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_UL=0000:a8:01.3 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_DL=0000:a8:01.7 \
--env MY_POD_IP=localhost --env MY_NODE_NAME=arch5-hddc301 \
--env ETCD_ENABLED=false --name=agf-17 -h agf-17 \
-it localhost/vagf:22.03
sleep 10
podman ps -a

podman run -d --privileged=true -v /dev/hugepages:/dev/hugepages \
--env CPUSET=58,59,162,163 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_UL=0000:a8:01.4 \
--env PCIDEVICE_INTEL_COM_APP_AGF_UP_DL=0000:a8:02.0 \
--env MY_POD_IP=localhost --env MY_NODE_NAME=arch5-hddc301 \
--env ETCD_ENABLED=false --name=agf-18 -h agf-18 \
-it localhost/vagf:22.03
```

6 Traffic Generator Configuration Guide

6.1 Ixia as Traffic Generator

1. Select Traffic and Add L2-3 Traffic Items.

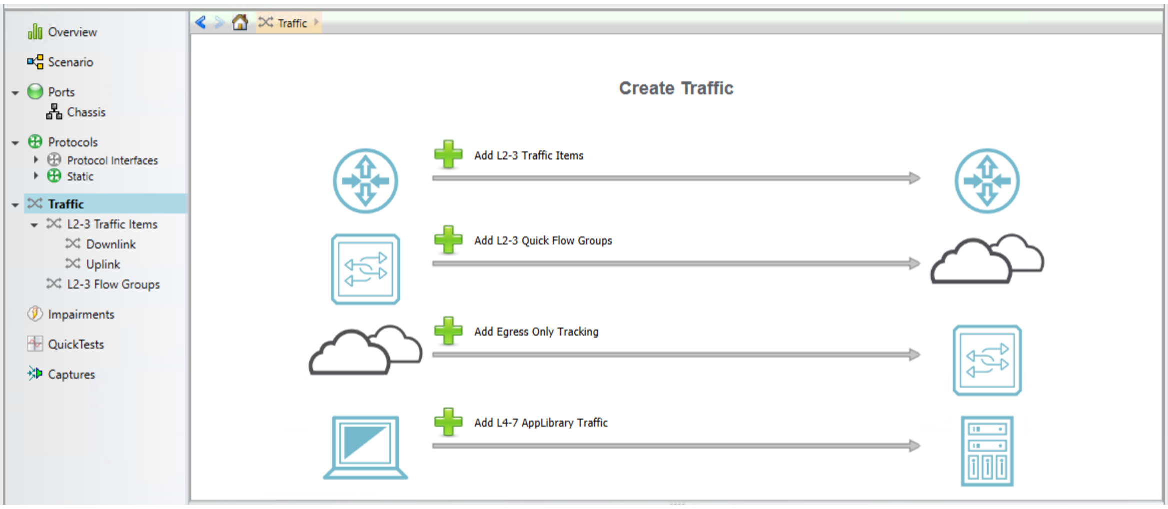
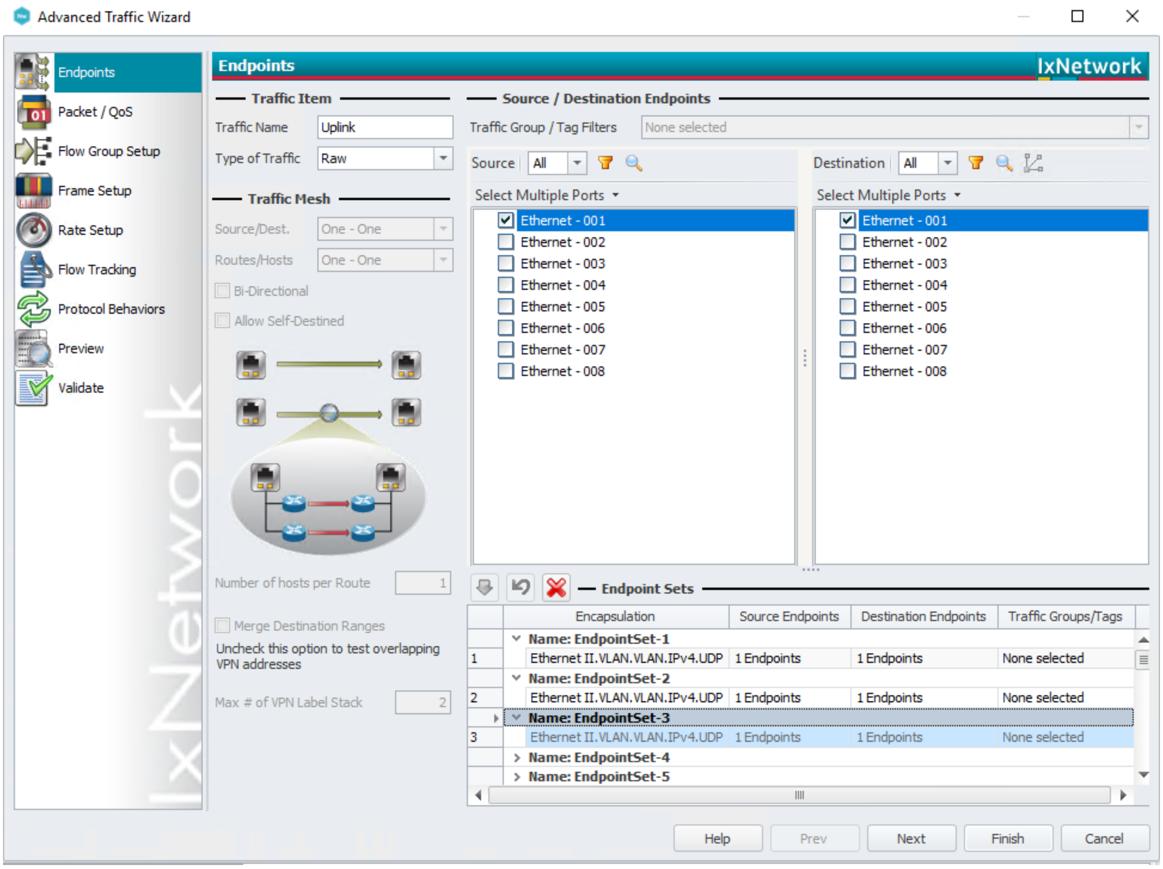


Figure 4. Add Traffic Items

2. Create each endpoint with one port as source and same port as destination. Repeat the same for all the required Traffic Items.



Traffic Item

Traffic Name: Uplink

Type of Traffic: Raw

Traffic Mesh

Source/Dest.: One - One

Routes/Hosts: One - One

□ Bi-Directional

□ Allow Self-Destined

Number of hosts per Route: 1

□ Merge Destination Ranges

Uncheck this option to test overlapping VPN addresses

Max # of VPN Label Stack: 2

Source / Destination Endpoints

Traffic Group / Tag Filters: None selected

Source: All

Select Multiple Ports

☑ Ethernet - 001
□ Ethernet - 002
□ Ethernet - 003
□ Ethernet - 004
□ Ethernet - 005
□ Ethernet - 006
□ Ethernet - 007
□ Ethernet - 008

Destination: All

Select Multiple Ports

☑ Ethernet - 001
□ Ethernet - 002
□ Ethernet - 003
□ Ethernet - 004
□ Ethernet - 005
□ Ethernet - 006
□ Ethernet - 007
□ Ethernet - 008

Endpoint Sets

	Encapsulation	Source Endpoints	Destination Endpoints	Traffic Groups/Tags
	Name: EndpointSet-1			
1	Ethernet II.VLAN.VLAN.IPv4.UDP	1 Endpoints	1 Endpoints	None selected
	Name: EndpointSet-2			
2	Ethernet II.VLAN.VLAN.IPv4.UDP	1 Endpoints	1 Endpoints	None selected
	Name: EndpointSet-3			
3	Ethernet II.VLAN.VLAN.IPv4.UDP	1 Endpoints	1 Endpoints	None selected
	Name: EndpointSet-4			
	Name: EndpointSet-5			

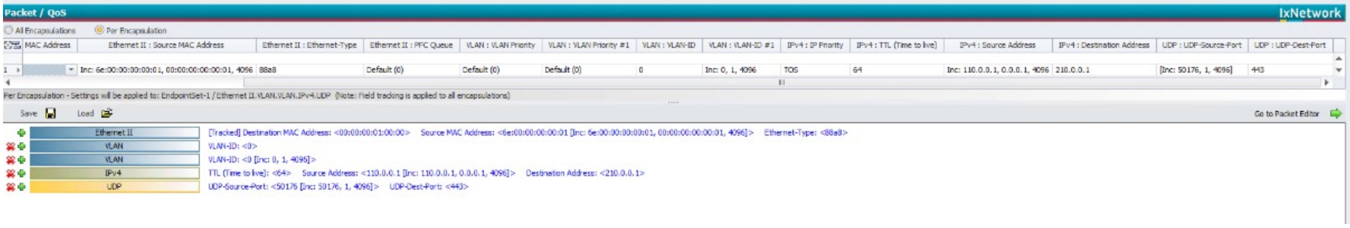
Figure 5. Create Required Endpoints

6.2 Configure Traffic Items Based on the Uplink or Downlink Configuration

6.2.1 Uplink Flow Groups

Configure the following settings that are common for all the Flow Groups. Add the VLAN followed by another VLAN to simulate QinQ followed by IPv4 and then UDP protocols to the frame.

- Destination MAC Address: Described below
- Source MAC Address:
  - Mode: Increment
  - Start: 6e:00:00:00:00:01
  - Step: 00:00:00:00:00:01
  - Count: 4096
- Ethernet Type: 0x88a8
- VLAN-ID: 0
- Ethernet Type: 0x8100
- VLAN-ID:
  - Mode: Increment
  - Start: 0
  - Step: 1
  - Count: 4096
- Ethernet Type: 0x8864
- PPPoE Session-ID:
  - Mode: Increment
  - Start: 0
  - Step: 1
  - Count: 4096
- Source IP Address:
  - Mode: Increment
  - Start: 110.0.0.1
  - Step: 0.0.0.1
  - Count: 4096
- Destination IP Address: 210.0.0.1
- UDP Source Port:
  - Mode: Increment
  - Start: 50176
  - Step: 1
  - Count: 4096
- UDP Destination Port: 443
- Payload: Increment Byte



	MAC Address	Ethernet II : Source MAC Address	Ethernet II : Ethernet-Type	Ethernet II : PFC Queue	VLAN : VLAN Priority	VLAN : VLAN Priority #1	VLAN : VLAN-ID	VLAN : VLAN-ID #1	IPv4 : IP Priority	IPv4 : TTL (Time to live)	IPv4 : Source Address	IPv4 : Destination Address	UDP : UDP-Source-Port	UDP : UDP-Dest-Port
1		Inc: 6e:00:00:00:00:01, 00:00:00:00:00:01, 4096	88a8	Default (0)	Default (0)	Default (0)	0	Inc: 0, 1, 4096	TOS	64	Inc: 110.0.0.1, 0.0.0.1, 4096	210.0.0.1	[Inc: 50176, 1, 4096]	443

Per Encapsulation - Settings will be applied to: EndpointSet-1 / Ethernet II.VLAN.VLAN.IPv4.UDP (Note: Field tracking is applied to all encapsulations)

Ethernet II: [Tracked] Destination MAC Address: <00:00:00:01:00:00> Source MAC Address: <6e:00:00:00:00:01 [Inc: 6e:00:00:00:00:01, 00:00:00:00:00:01, 4096]> Ethernet-Type: <88a8>

VLAN: VLAN-ID: <0>

VLAN: VLAN-ID: <0 [Inc: 0, 1, 4096]>

IPv4: TTL (Time to live): <64> Source Address: <110.0.0.1 [Inc: 110.0.0.1, 0.0.0.1, 4096]> Destination Address: <210.0.0.1>

UDP: UDP-Source-Port: <50176 [Inc: 50176, 1, 4096]> UDP-Dest-Port: <443>

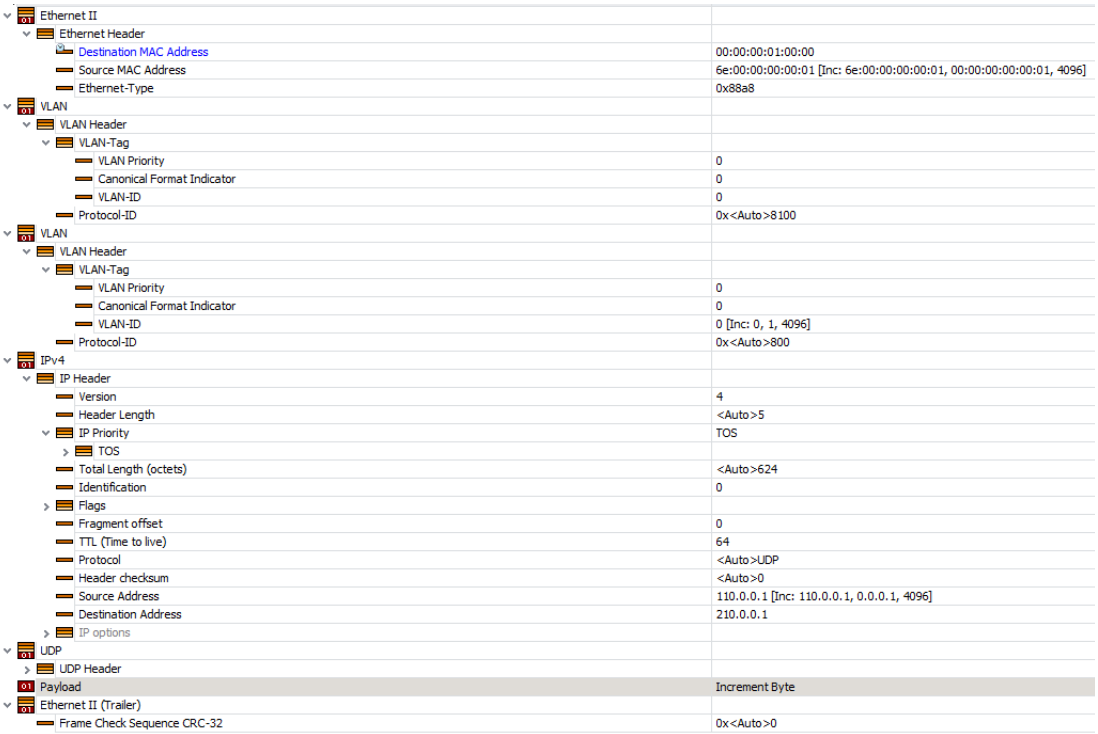
Figure 6. Add Protocols to the Frame in the Packet/QoS Editor

Table 8 shows example Destination MAC Address for the traffic items starting from 0 to 7, for a total of 8 uplink instances, in the order shown here. It can be extended to 32 instances by incrementing the 5th octet of the MAC address.

IXIA* 8x25GbE Traffic Profile - vAGF Uplink Traffic Item

TRAFFIC ITEM	TX PORT	RX PORT	DESTINATION MAC ADDRESS
Flow Group 1	Uplink Port 0	Uplink Port 0	00:00:00:01:00:00
Flow Group 2	Uplink Port 0	Uplink Port 0	00:00:00:01:01:00
Flow Group 3	Uplink Port 0	Uplink Port 0	00:00:00:01:02:00
Flow Group 4	Uplink Port 0	Uplink Port 0	00:00:00:01:03:00
Flow Group 5	Uplink Port 1	Uplink Port 1	00:00:00:01:04:00
Flow Group 6	Uplink Port 1	Uplink Port 1	00:00:00:01:05:00
Flow Group 7	Uplink Port 1	Uplink Port 1	00:00:00:01:06:00
Flow Group 8	Uplink Port 1	Uplink Port 1	00:00:00:01:07:00

Table 8. Uplink Flow Traffic Item Variable Values



```
Ethernet II
  Ethernet Header
    Destination MAC Address                  00:00:00:01:00:00
    Source MAC Address                       6e:00:00:00:00:01 [Inc: 6e:00:00:00:00:01, 00:00:00:00:00:01, 4096]
    Ethernet-Type                            0x88a8
VLAN
  VLAN Header
    VLAN-Tag
      VLAN Priority                          0
      Canonical Format Indicator             0
      VLAN-ID                                0
    Protocol-ID                              0x<Auto>8100
VLAN
  VLAN Header
    VLAN-Tag
      VLAN Priority                          0
      Canonical Format Indicator             0
      VLAN-ID                                0 [Inc: 0, 1, 4096]
    Protocol-ID                              0x<Auto>800
IPv4
  IP Header
    Version                                  4
    Header Length                            <Auto>5
    IP Priority                              TOS
      TOS
    Total Length (octets)                    <Auto>624
    Identification                           0
    Flags
    Fragment offset                          0
    TTL (Time to live)                       64
    Protocol                                 <Auto>UDP
    Header checksum                          <Auto>0
    Source Address                           110.0.0.1 [Inc: 110.0.0.1, 0.0.0.1, 4096]
    Destination Address                      210.0.0.1
    IP options
UDP
  UDP Header
Payload                                      Increment Byte
Ethernet II (Trailer)
  Frame Check Sequence CRC-32                0x<Auto>0
```

Figure 7. Add Protocols to the Frame in the Packet/QoS Editor

6.2.2 Downlink Flow Groups

Configure the following settings that are common for all the Flow Groups.

- Destination MAC Address: Described below
- Source MAC Address: 00:00:01:01:00:01
- Source IP Address: 210.0.0.1
- Destination IP Address:
  - Mode: Increment
  - Start: 110.0.0.1
  - Step: 0.0.0.1
  - Count: 4096

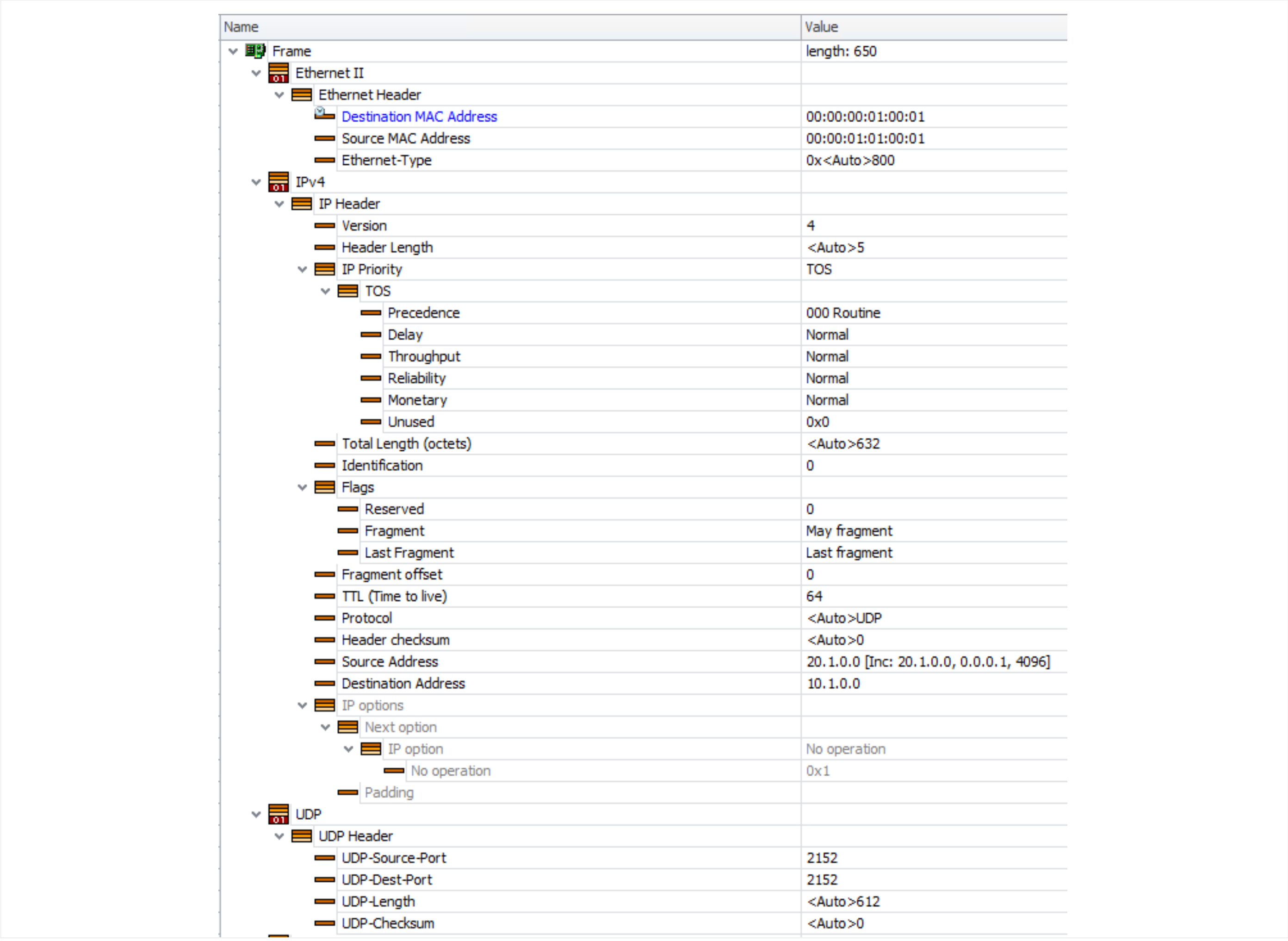
- UDP Source Port: 443
- UDP Destination Port:
  - Mode: Increment
  - Start: 50176
  - Step: 1
  - Count: 4096

Table 9 shows the Destination MAC Address for the instances starting from 0 to 7 for a total of 8 downlink instances in the order given next. It can be extended to 32 instances by incrementing the 5th octet of the MAC address.

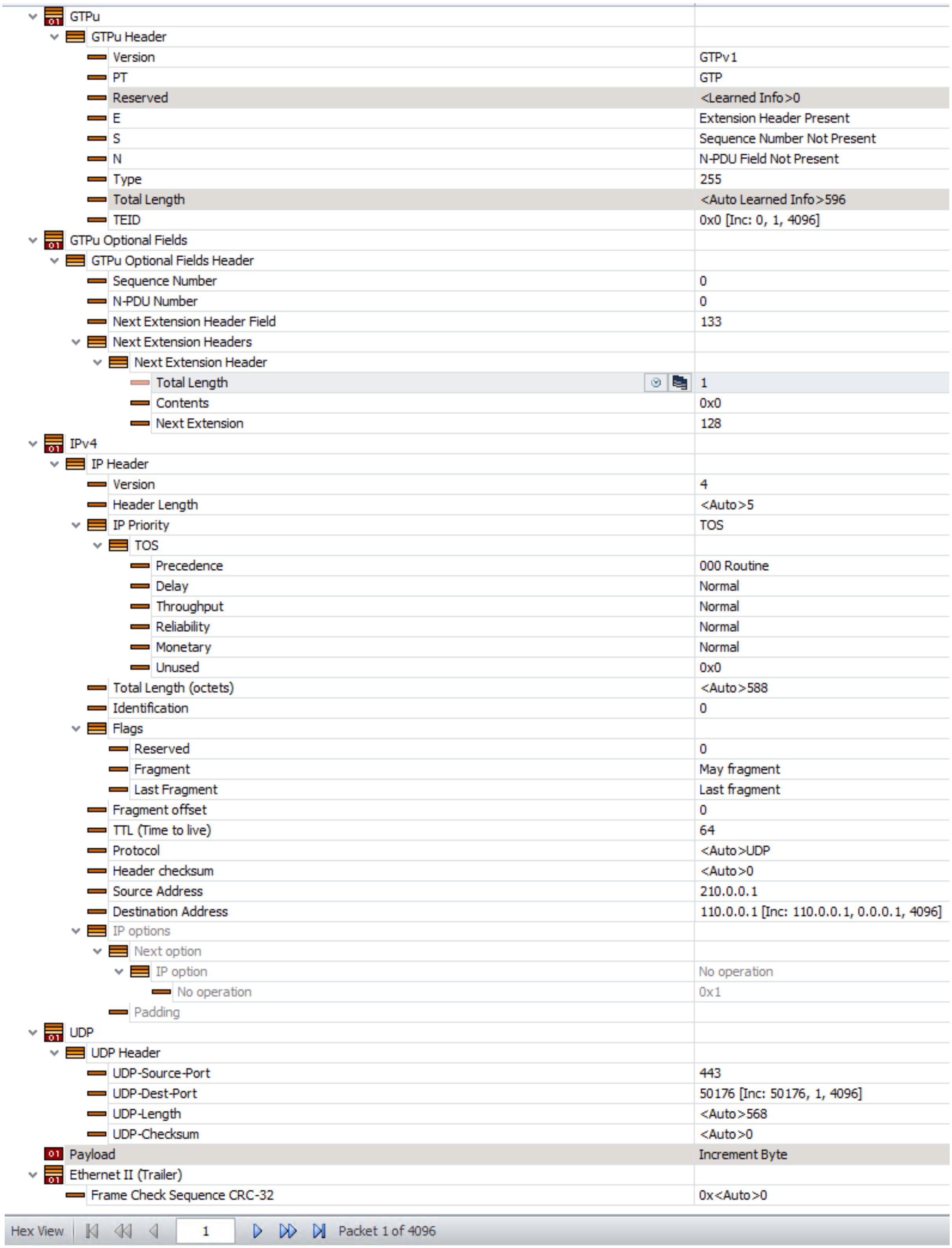
IXIA* 8x25GbE Traffic Profile - vAGF Downlink Traffic Item

TRAFFIC ITEM	TX PORT	RX PORT	DESTINATION MAC ADDRESS
Flow Group 1	Downlink Port 0	Downlink Port 0	00:00:00:01:00:01
Flow Group 2	Downlink Port 0	Downlink Port 0	00:00:00:01:01:01
Flow Group 3	Downlink Port 0	Downlink Port 0	00:00:00:01:02:01
Flow Group 4	Downlink Port 0	Downlink Port 0	00:00:00:01:03:01
Flow Group 5	Downlink Port 1	Downlink Port 1	00:00:00:01:04:01
Flow Group 6	Downlink Port 1	Downlink Port 1	00:00:00:01:05:01
Flow Group 7	Downlink Port 1	Downlink Port 1	00:00:00:01:06:01
Flow Group 8	Downlink Port 1	Downlink Port 1	00:00:00:01:07:01

Table 9. Downlink Flow Traffic Item Variable Values

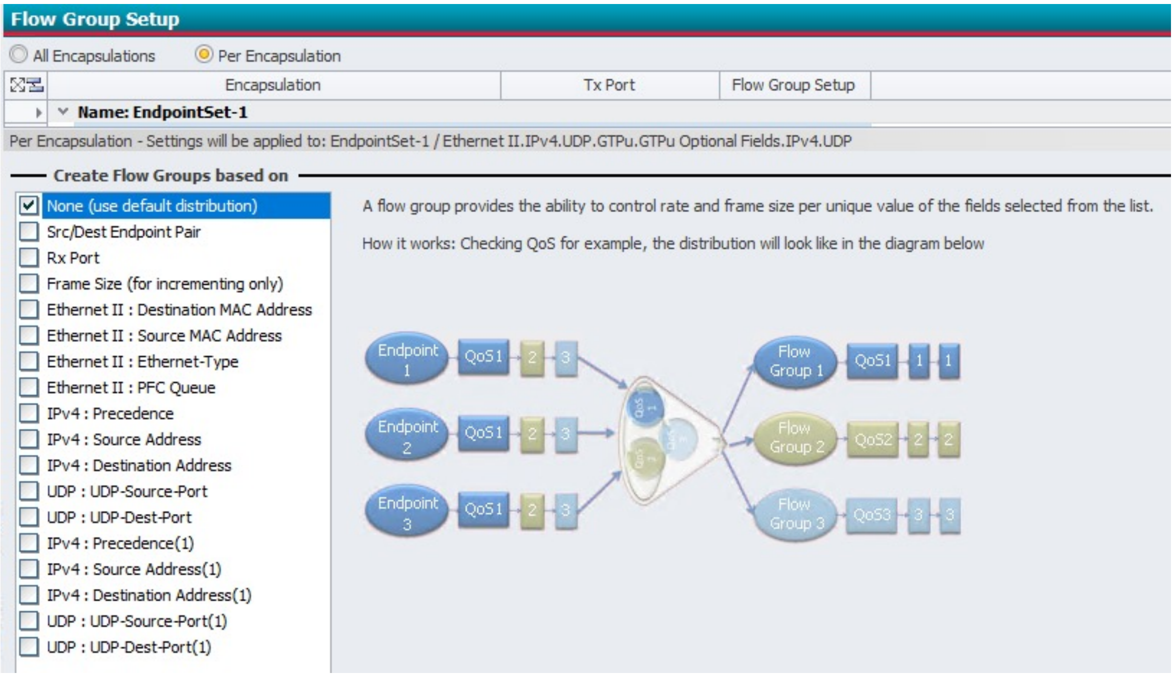


Name	Value
Frame	length: 650
Ethernet II	
Ethernet Header	
Destination MAC Address	00:00:00:01:00:01
Source MAC Address	00:00:01:01:00:01
Ethernet-Type	0x<Auto>800
IPv4	
IP Header	
Version	4
Header Length	<Auto>5
IP Priority	TOS
TOS	
Precedence	000 Routine
Delay	Normal
Throughput	Normal
Reliability	Normal
Monetary	Normal
Unused	0x0
Total Length (octets)	<Auto>632
Identification	0
Flags	
Reserved	0
Fragment	May fragment
Last Fragment	Last fragment
Fragment offset	0
TTL (Time to live)	64
Protocol	<Auto>UDP
Header checksum	<Auto>0
Source Address	20.1.0.0 [Inc: 20.1.0.0, 0.0.0.1, 4096]
Destination Address	10.1.0.0
IP options	
Next option	
IP option	No operation
No operation	0x1
Padding	
UDP	
UDP Header	
UDP-Source-Port	2152
UDP-Dest-Port	2152
UDP-Length	<Auto>612
UDP-Checksum	<Auto>0



GTPu	
GTPu Header	
Version	GTPv1
PT	GTP
Reserved	<Learned Info>0
E	Extension Header Present
S	Sequence Number Not Present
N	N-PDU Field Not Present
Type	255
Total Length	<Auto Learned Info>596
TEID	0x0 [Inc: 0, 1, 4096]
GTPu Optional Fields	
GTPu Optional Fields Header	
Sequence Number	0
N-PDU Number	0
Next Extension Header Field	133
Next Extension Headers	
Next Extension Header	
Total Length	1
Contents	0x0
Next Extension	128
IPv4	
IP Header	
Version	4
Header Length	<Auto>5
IP Priority	TOS
TOS	
Precedence	000 Routine
Delay	Normal
Throughput	Normal
Reliability	Normal
Monetary	Normal
Unused	0x0
Total Length (octets)	<Auto>588
Identification	0
Flags	
Reserved	0
Fragment	May fragment
Last Fragment	Last fragment
Fragment offset	0
TTL (Time to live)	64
Protocol	<Auto>UDP
Header checksum	<Auto>0
Source Address	210.0.0.1
Destination Address	110.0.0.1 [Inc: 110.0.0.1, 0.0.0.1, 4096]
IP options	
Next option	
IP option	No operation
No operation	0x1
Padding	
UDP	
UDP Header	
UDP-Source-Port	443
UDP-Dest-Port	50176 [Inc: 50176, 1, 4096]
UDP-Length	<Auto>568
UDP-Checksum	<Auto>0
Payload	Increment Byte
Ethernet II (Trailer)	
Frame Check Sequence CRC-32	0x<Auto>0

Figure 8. Example Downlink Packet /QoS for one Endpoint Set



Flow Group Setup

○ All Encapsulations ● Per Encapsulation

Encapsulation	Tx Port	Flow Group Setup
Name: EndpointSet-1		

Per Encapsulation - Settings will be applied to: EndpointSet-1 / Ethernet II.IPv4.UDP.GTPu.GTPu Optional Fields.IPv4.UDP

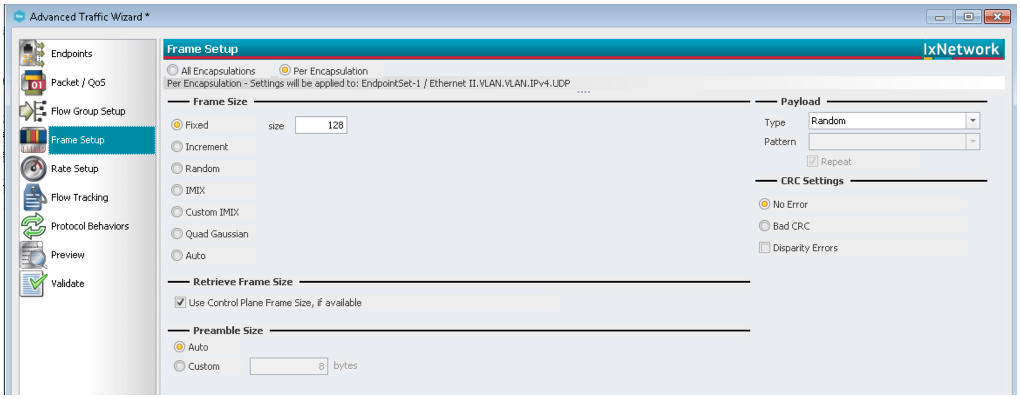
Create Flow Groups based on

- ☑ None (use default distribution)
- □ Src/Dest Endpoint Pair
- □ Rx Port
- □ Frame Size (for incrementing only)
- □ Ethernet II : Destination MAC Address
- □ Ethernet II : Source MAC Address
- □ Ethernet II : Ethernet-Type
- □ Ethernet II : PFC Queue
- □ IPv4 : Precedence
- □ IPv4 : Source Address
- □ IPv4 : Destination Address
- □ UDP : UDP-Source-Port
- □ UDP : UDP-Dest-Port
- □ IPv4 : Precedence(1)
- □ IPv4 : Source Address(1)
- □ IPv4 : Destination Address(1)
- □ UDP : UDP-Source-Port(1)
- □ UDP : UDP-Dest-Port(1)

A flow group provides the ability to control rate and frame size per unique value of the fields selected from the list.

How it works: Checking QoS for example, the distribution will look like in the diagram below

Figure 9. Flow Group Setup for one Endpoint Set



Advanced Traffic Wizard *

Frame Setup

○ All Encapsulations ● Per Encapsulation

Per Encapsulation - Settings will be applied to: EndpointSet-1 / Ethernet II.VLAN.VLAN.IPv4.UDP

Frame Size

- ● Fixed size 128
- ○ Increment
- ○ Random
- ○ IMIX
- ○ Custom IMIX
- ○ Quad Gaussian
- ○ Auto

Retrieve Frame Size

☑ Use Control Plane Frame Size, if available

Preamble Size

- ● Auto
- ○ Custom 8 bytes

Payload

Type Random

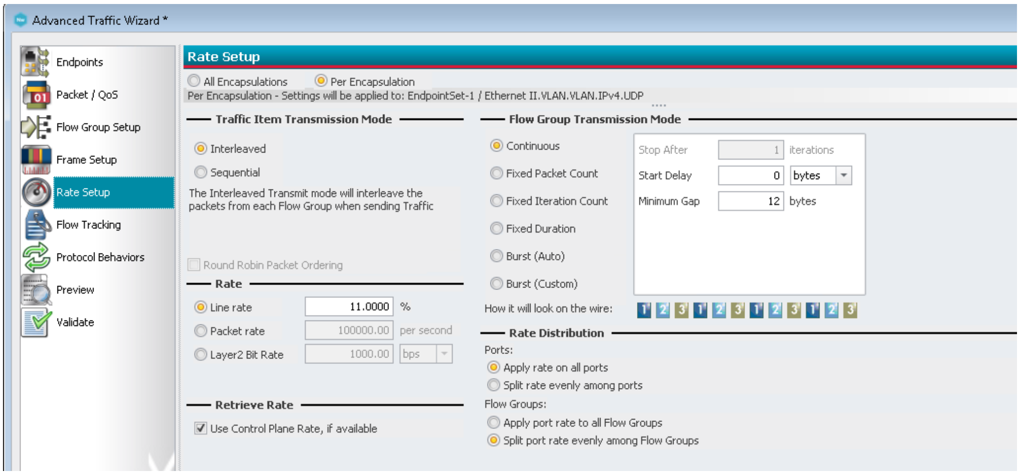
Pattern

☑ Repeat

CRC Settings

- ● No Error
- ○ Bad CRC
- □ Disparity Errors

Figure 10. Frame Setup with Packet Size of 128B for Uplink and 504B for Downlink



Advanced Traffic Wizard *

Rate Setup

○ All Encapsulations ● Per Encapsulation

Per Encapsulation - Settings will be applied to: EndpointSet-1 / Ethernet II.VLAN.VLAN.IPv4.UDP

Traffic Item Transmission Mode

- ● Interleaved
- ○ Sequential

The Interleaved Transmit mode will interleave the packets from each Flow Group when sending Traffic

□ Round Robin Packet Ordering

Rate

- ● Line rate 11.0000 %
- ○ Packet rate 100000.00 per second
- ○ Layer2 Bit Rate 1000.00 bps

Retrieve Rate

☑ Use Control Plane Rate, if available

Flow Group Transmission Mode

- ● Continuous
- ○ Fixed Packet Count
- ○ Fixed Iteration Count
- ○ Fixed Duration
- ○ Burst (Auto)
- ○ Burst (Custom)

Stop After 1 iterations

Start Delay 0 bytes

Minimum Gap 12 bytes

How it will look on the wire: 1 2 3 1 2 3 1 2 3 1 2 3

Rate Distribution

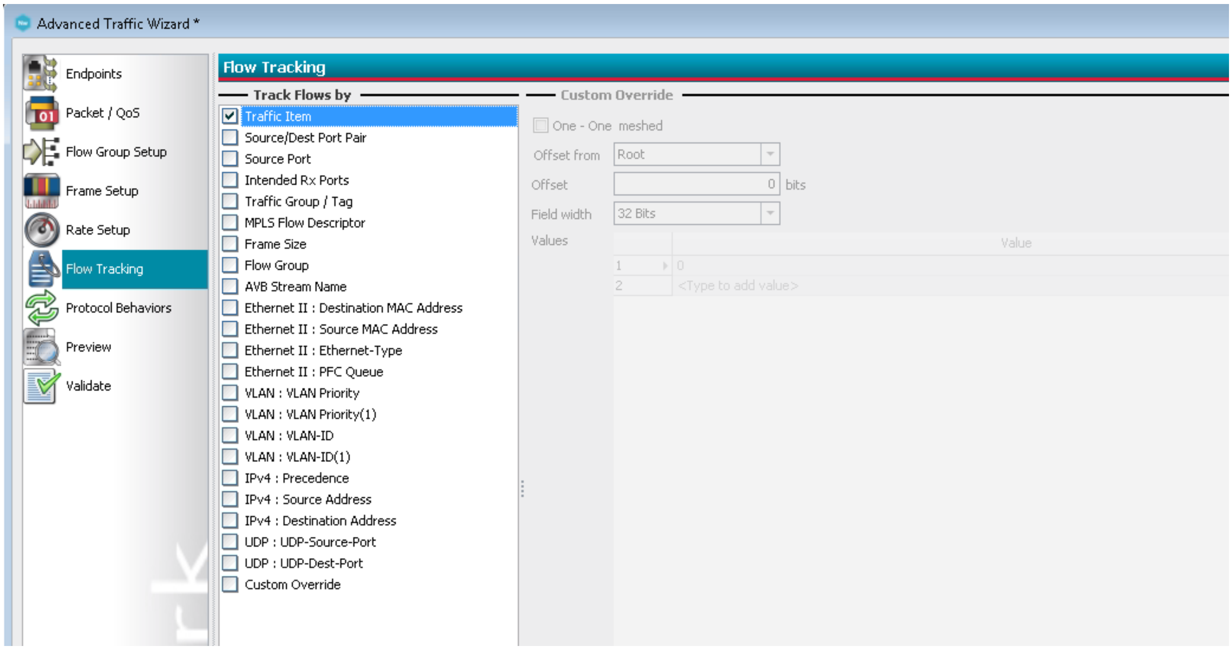
Ports:

- ● Apply rate on all ports
- ○ Split rate evenly among ports

Flow Groups:

- ○ Apply port rate to all Flow Groups
- ● Split port rate evenly among Flow Groups

Figure 11. Rate Setup of 11% Line Rate for Upstream and 89% Line Rate for Downstream



Track Flows by

- ☑ Traffic Item
- ☐ Source/Dest Port Pair
- ☐ Source Port
- ☐ Intended Rx Ports
- ☐ Traffic Group / Tag
- ☐ MPLS Flow Descriptor
- ☐ Frame Size
- ☐ Flow Group
- ☐ AVB Stream Name
- ☐ Ethernet II : Destination MAC Address
- ☐ Ethernet II : Source MAC Address
- ☐ Ethernet II : Ethernet-Type
- ☐ Ethernet II : PFC Queue
- ☐ VLAN : VLAN Priority
- ☐ VLAN : VLAN Priority(1)
- ☐ VLAN : VLAN-ID
- ☐ VLAN : VLAN-ID(1)
- ☐ IPv4 : Precedence
- ☐ IPv4 : Source Address
- ☐ IPv4 : Destination Address
- ☐ UDP : UDP-Source-Port
- ☐ UDP : UDP-Dest-Port
- ☐ Custom Override

Custom Override

☐ One - One meshed

Offset from: Root

Offset: 0 bits

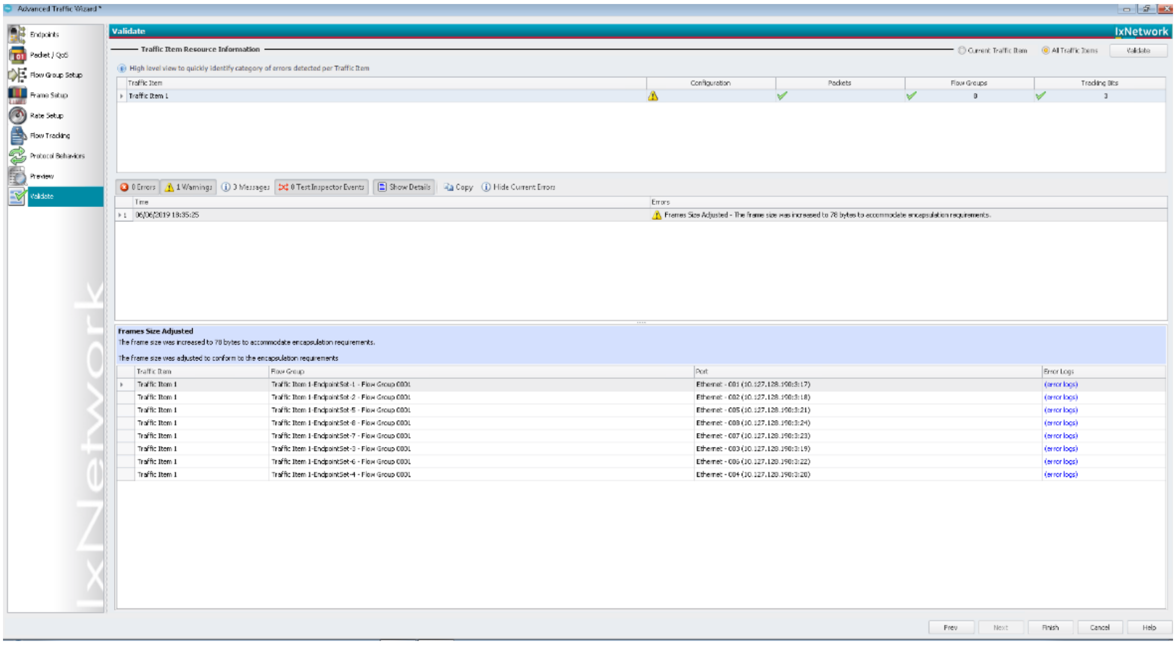
Field width: 32 Bits

Values

	Value
1	0
2	<Type to add value>

Figure 12. Flow Tracking Set to 'Traffic Item'

The last step is to validate the flow.



Traffic Item Resource Information

High level view to quickly identify category of errors detected per Traffic Item

Traffic Item	Configuration	Packets	Flow Groups	Tracking Bits
Traffic Item 1	⚠	✓	✓ 8	✓ 3

	Time	Errors
1	06/06/2019 18:35:25	⚠ Frames Size Adjusted - The frame size was increased to 78 bytes to accommodate encapsulation requirements.

Frames Size Adjusted

The frame size was increased to 78 bytes to accommodate encapsulation requirements.

The frame size was adjusted to conform to the encapsulation requirements

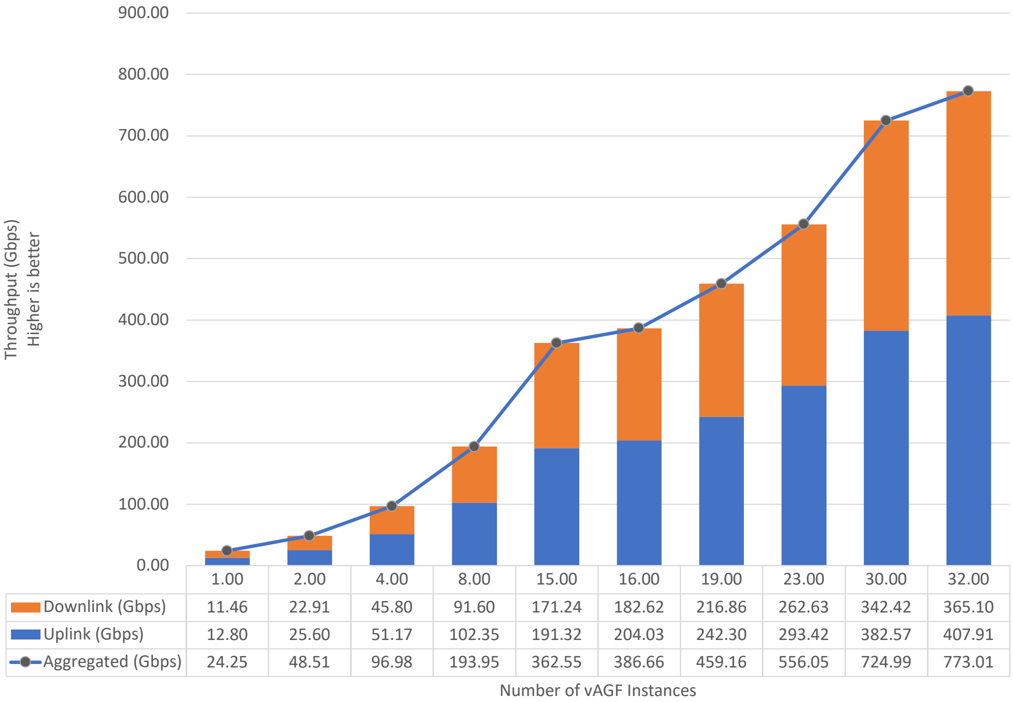
Traffic Item	Flow Group	Port	Error Logs
Traffic Item 1	Traffic Item 1-EndpointSet-1 - Flow Group 0001	Ethernet - 001 (10.127.128.190:3:17)	(error logs)
Traffic Item 1	Traffic Item 1-EndpointSet-2 - Flow Group 0001	Ethernet - 002 (10.127.128.190:3:18)	(error logs)
Traffic Item 1	Traffic Item 1-EndpointSet-5 - Flow Group 0001	Ethernet - 005 (10.127.128.190:3:21)	(error logs)
Traffic Item 1	Traffic Item 1-EndpointSet-8 - Flow Group 0001	Ethernet - 008 (10.127.128.190:3:24)	(error logs)
Traffic Item 1	Traffic Item 1-EndpointSet-7 - Flow Group 0001	Ethernet - 007 (10.127.128.190:3:23)	(error logs)
Traffic Item 1	Traffic Item 1-EndpointSet-3 - Flow Group 0001	Ethernet - 003 (10.127.128.190:3:19)	(error logs)
Traffic Item 1	Traffic Item 1-EndpointSet-6 - Flow Group 0001	Ethernet - 006 (10.127.128.190:3:22)	(error logs)
Traffic Item 1	Traffic Item 1-EndpointSet-4 - Flow Group 0001	Ethernet - 004 (10.127.128.190:3:20)	(error logs)

Figure 13. Validate Flow Group

7 vAGF Benchmarks

7.1 vAGF Symmetric Traffic Benchmarks

The results of the vAGF Symmetric benchmarks on 4th Gen Intel® Xeon® 8470N processors with 800 Gbps Line Rate (8 x 100G Intel® Ethernet Network Adapter E810-2CQDA2), 650B UL/DL Packet Size and a traffic ratio of DL/UL 1:1 and a Packet Loss of less than 0.001% with turbo disabled show a Maximum Receive Rate (MRR) of up to 773 Gbps Rx L2/L3 throughput which is 96.62% of the line rate and up to 795 Gbps Rx L1 throughput which is 99.37% of the line rate

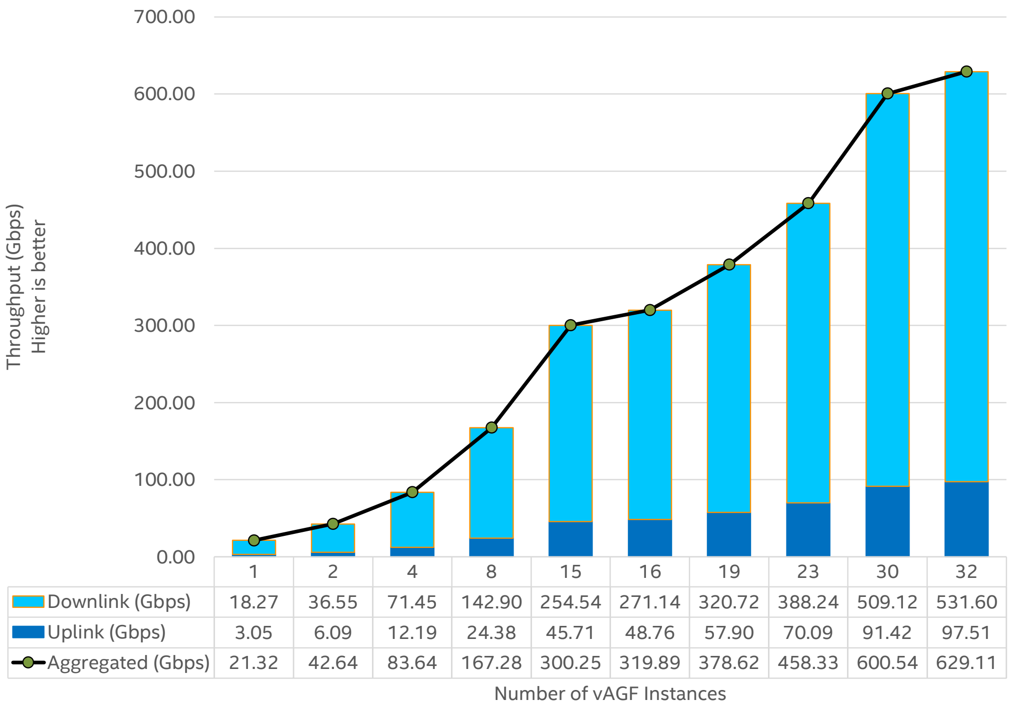


	1.00	2.00	4.00	8.00	15.00	16.00	19.00	23.00	30.00	32.00
🟧 Downlink (Gbps)	11.46	22.91	45.80	91.60	171.24	182.62	216.86	262.63	342.42	365.10
🟦 Uplink (Gbps)	12.80	25.60	51.17	102.35	191.32	204.03	242.30	293.42	382.57	407.91
● Aggregated (Gbps)	24.25	48.51	96.98	193.95	362.55	386.66	459.16	556.05	724.99	773.01

Figure 14. vAGF Symmetric Traffic Benchmarks[1,2]

7.2 vAGF Asymmetric Traffic Benchmarks

The results of the vAGF Asymmetric benchmarks on 4th Gen Intel® Xeon® 8470N processors with 800 Gbps Line Rate (8 x 100G Intel® Ethernet Network Adapter E810-2CQDA2), 504B DL and 128B UL Packet Size and a traffic ratio of DL/UL 89:11 and a Packet Loss of less than 0.001% with turbo disabled show a Maximum Receive Rate (MRR) of up to 629 Gbps Rx L2/L3 throughput which is 78.62% of the line rate and up to 663 Gbps Rx L1 throughput which is 82.87% of the line rate



	1	2	4	8	15	16	19	23	30	32
🟦 Downlink (Gbps)	18.27	36.55	71.45	142.90	254.54	271.14	320.72	388.24	509.12	531.60
🟦 Uplink (Gbps)	3.05	6.09	12.19	24.38	45.71	48.76	57.90	70.09	91.42	97.51
● Aggregated (Gbps)	21.32	42.64	83.64	167.28	300.25	319.89	378.62	458.33	600.54	629.11

Figure 15. vAGF Asymmetric Traffic Benchmarks[1,2]

8 Summary

The Intel virtualized Access Gateway Function benchmarks on 4th Gen Intel® Xeon® Platinum 8470N processors showed an impressive I/O throughput of up to 773 Gbps Maximum Receive Rate for L2/L3, which was 96.62% of the maximum line rate of 800 Gbps and up to 795 Gbps Maximum Receive Rate for L1, which was 99.37% of the maximum line rate. Only 32 cores out of a total 52 available cores were used per NUMA node, with both sockets populated. The use of the Dynamic Device Personalization (DDP feature in the Intel® Ethernet Adapter E810-2CQDA2 with the Intel COMMS DDP package contributed to the high performance.

The high core count of the 4th Gen Intel® Xeon® Scalable processors, combined with architectural improvements, feature enhancements, and high memory bandwidth, is a tremendous performance and scalability advantage over previous Intel® Xeon® processor generations, especially in today's NFVI environments. These processors are optimized for network, cloud native, wireline, and wireless core-intensive workloads, with up to 60 powerful cores and a wide range of frequency, features and power levels. The Intel® Xeon® Platinum 8470N processors with 52 cores at 1.7 GHz core frequency, 2.4 GHz uncore frequency, high DDR5 memory bandwidth and PCIe Gen 4 IO throughput has outstanding performance based on a balanced, efficient architecture that increases performance memory and I/O bandwidth to accelerate diverse workloads from the data center to the intelligent edge.

Table of Contents

1 Introduction 1
1.1 NFVI Forwarding Platform 1
1.2 Terminology 1
1.3 Reference Documents and Resources 3
2 Solution Components 3
2.1 Intel® Xeon® Processor Scalable Performance Family 3
2.2 Intel® Ethernet 800 Series 4
3 NFVI Forwarding Platform Reference Architecture Requirements 5
3.1 Reference Architecture Hardware Requirements 5
3.2 Reference Architecture Software Requirements 6
3.3 BIOS Settings 6
3.4 Platform Technology Requirements 7
3.5 Platform Security 7
4 Virtual Access Gateway Function (vAGF) 8
4.1 vAGF Overview 8
4.2 vAGF Pipeline 8
4.3 vAGF Test Setup 10
5 vAGF Installation Guide 11
5.1 vAGF Application Server Preparation 11
5.2 System BIOS Settings 11
5.3 RHEL* 8.6 Installation 11
5.4 Install vAGF Dependencies 12
5.5 vAGF Dataplane Application Environment Set-up 12
6 Traffic Generator Configuration Guide 20
6.1 Ixia as Traffic Generator 20
6.2 Configure Traffic Items Based on the Uplink or Downlink Configuration 21
7 vAGF Benchmarks 26
7.1 vAGF Symmetric Traffic Benchmarks 26
7.2 vAGF Asymmetric Traffic Benchmarks 27
8 Summary 28

[1] **Notices & Disclaimers**

Performance varies by use, configuration and other factors. Learn more on the Performance Index site. Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure. Your costs and results may vary. Intel technologies may require enabled hardware, software or service activation.

[2] **Configuration**

Test by Intel as of 11/05/2022. 1-node, 2x Intel® Xeon® Platinum 8470N, 52 cores, HT On, Turbo Off, Total Memory 512 GB (16 x 32GB DDR5-48000), BIOS EGSDCRB1.86B.8901.P01.2209200239, ucode 0xab0000c0, Red Hat Enterprise Linux 8.6 (Ootpa), kernel 4.18.0-348.el8.x86_64, gcc compiler 8.5.0, vAGF 22.03, DPDK 20.11.5, ice driver: 1.9.11, DDP ICE COMMS package version 1.3.31.0

0123/SM/H09/PDF ♻Please Recycle 763626-001US