Executive Summary

The ever-growing consumer demand for more bandwidth and services for less money has been driving service provider networks to their economic limit for some years now. In addition, communications service providers (CoSPs) need to support multiple access technology types (xDSL, PON, FWA, and DOCSIS) while making better use of existing fiber networks and increasing service delivery performance, all against a backdrop of declining revenues. With customer data traffic estimated to grow at a 26 percent rate (year over year) from 2017 to 2022, future networking solutions must show a path to solving tomorrow’s data compound-annual-growth-rate (CAGR) issue in a cost-effective and scalable way.

Pushing the boundary of performance requires the latest and greatest technology along with deploying this technology in a holistic and easy-to-use way. This paper proposes the following set of design principles for taking solutions based on Intel® architecture processors and network functions virtualization (NFV) to the next level of performance and network automation.

- Optimized server configuration
- Software “run-to-completion” model
- Intelligent I/O packet and flow distribution
- Independent scaling of the control and user planes
- Hierarchical quality of service considerations

Following these principles, Intel has demonstrated nearly 330 Gbps² of routing throughput performance for a virtual broadband network gateway (vBNG) running on a single server. This paper describes this effort and proposes a vBNG architecture for building network infrastructure and network functions to better take advantage of the underlying infrastructure.

Broadband Network Gateway

The BNG, aka broadband remote access server (BRAS), is the network edge aggregation point used by subscribers to access the Internet service provider (ISP) network. Through the BNG, subscribers connect to the ISP network to download Internet originating traffic and ISP services (e.g., web, voice, data, and video).

---

Figure 1. Example of a Network Connecting Clients to Internet and network services
The vBNG 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).

### Reference Pipeline

Each generation of Intel technologies (e.g., CPU, NIC SSD, FPGAs, and accelerators) brings new opportunities to improve performance and quality of experience (QoE) for users. Showing how to take advantage of these technologies, Intel builds reference pipelines, like the representative stages of vBNG control and data plane functions shown in Figure 2.

The control plane is responsible for subscriber authentication and management, including monthly usage service access and data plane configuration, based on subscriber profiles.

The upstream data plane manages the flow of traffic from users’ home routers to the ISP network. The average packet size for upstream traffic is generally smaller than for downstream traffic, and the amount of upstream traffic is normally five to eight times less than downstream traffic. While applications, like Instagram, Snapchat, and Periscope, have seen larger swaths of data being pushed onto the ISP network by users, broadband users are still overwhelmingly net consumers of data. In recent years, data and content creation has reduced the gap between upstream and downstream bandwidth usage.

### Upstream Processing Stages

The Intel reference pipeline implements the upstream processing stages shown in Figure 3 and described in the following:

- **Packet Rx (Receive):** The data plane development kit (DPDK) poll mode driver (PMD) is used to receive bursts of frames from the network interface controller (NIC) port and send them directly into an uplink thread to begin vBNG packet processing, described in the next stages.

- **Access Control Lists:** The DPDK Access Control List (ACL) library is used to apply an ordered list of filters (e.g., masks, ranges, etc.) to the frame. These comprise permit and deny filters, and all filters are evaluated per packet.

- **Flow Classification:** The DPDK Flow Classification Library is used to identify the session and classify the packet based on selected fields (e.g., 5 tuple).

- **Metering Policing:** The DPDK Traffic Metering and Policing API is used to apply a two-rate, three-color marking and policing scheme to the traffic.

- **DSCP Rewrite:** This stage supports the optional classification of the traffic type and rewrite of the IP differentiated services code point (DSCP) field to map the stream to a network supported class of service (CoS).

- **NAT:** Optionally, NAT 44 is performed to convert private addresses to public addresses.

- **Routing:** Access network encapsulations are stripped from data plane packets, and the packets are routed to the correct core network interface for transmission. Any core network encapsulations, such as MPLS, are applied either here or in the packet Tx block.

- **Packet Tx (Transmit):** The DPDK PMD is used to send bursts of frames to the NIC port.

The downstream data plane handles the flow of traffic and data from the Internet and ISP network to the end user. It manages and schedules traffic to users attached to the BNG. The downstream function optimizes bandwidth and resource usage to maximize users QoE, based on user tariff class and traffic priorities. The goal of the ISP is to ensure all their subscribers are receiving services to the highest standard while maximizing the utility of the network infrastructure. By 2022, global IP video traffic is forecast to grow four-fold from 2017 to 2022, a CAGR of 29 percent, and this trend will drive up the average packet size of the downstream link.
Downstream Processing Stages
The Intel reference pipeline implements the downstream processing stages shown in Figure 4 and described in the following:

- **Packet Rx**: The DPDK PMD receives frames from the NIC port and sends them directly into a downlink thread to begin vBNG packet processing, described in the next stages.

- **Access Control Lists (ACL)**: The DPDK Access Control List (ACL) library is used to apply an ordered list of filters (e.g., masks, ranges, etc.) to the frame and can be used for unicast reverse path forwarding implementation.

- **NAT**: Optionally, NAT 44 is performed to convert public addresses to private addresses.

- **Traffic Management**: Each packet runs through a hierarchical QoS (HQoS) block to ensure high priority packets are prioritized when transmitting packets to the access network. A four-level HQoS scheme is used with port, pipe, traffic class, and queue levels. Each pipe is assigned to a single subscriber.

- **Routing**: Core network encapsulations are stripped from data plane packets, and the packets are routed to the correct core network interface for transmission. Any access network encapsulations, such as VLAN, PPPoE etc., are applied either here or in the packet Tx block.

- **Packet Tx**: Using a DPDK polled mode driver (PMD), bursts of frames are transmitted to the NIC port.

BNG Design Considerations
In order to effectively deploy a BNG workload on a general-purpose server, the following architectural and implementation aspects should be considered:

Balancing I/O on the Server
With dual socket servers, internal connections such as Platform Controller Hubs (PCHs) and SATA controllers are commonly connected to CPU 0, as shown on the left side of Figure 5. This can result in an uneven distribution of PCIe* I/O bandwidth between the CPUs, with most of the bandwidth being connected to CPU 1. To balance the bandwidth, the Intel vBNG application runs control plane functions on CPU 0 and data plane functions on CPU 1, as shown on the right side of Figure 5.

When deploying a BNG on a general-purpose server, it is important to ensure there is enough I/O bandwidth to fully utilize the available CPU resources on the platform (i.e., the aim is to be CPU bound and not I/O bound). The advent of Control and User Plane Separation (CUPS)⁴ for BNG enables an entire server to be dedicated to running the BNG data plane. All data processing is localized to a single socket for performance efficiency, which necessitates an equal amount of I/O to be connected to each socket in order to achieve optimal performance. The provisioning of 2x16 or 4x8 lane PCIe slots on each socket provides a total I/O bandwidth of 400 Gbps on the server, equally balanced across the two sockets (see the Appendix for server configuration details).

Distributing Flows via a Network Interface Card (NIC)
A BNG has multiple worker threads processing subscriber traffic. Some form of distributor is required in order split out the subscriber flows among the CPU worker cores. This distributor task can be performed in software using dedicated cores, but there are a number of disadvantages to this approach. First, this distributor function can become a performance bottleneck as all flows must pass through this software function. Second, having one or many cores dedicated to performing distribution reduces the amount
of CPU cycles available to perform the actual BNG workload processing, thus reducing the number of BNG subscribers the server can handle.

These disadvantages can be overcome by distributing the flows in the NIC, which eliminates the software bottleneck, reduces latency through the system, and provides the BNG the CPU cores and cycles that otherwise would be used by the distributor task. The connection between the customer premises equipment (CPE) and the BNG is via a Point-to-Point Protocol over Ethernet (PPPoE) or Internet Protocol over Ethernet (IPoE) link with each subscriber having a unique IP or PPPoE session ID.

The Intel® Ethernet Controller XXV710 is a NIC that supports Dynamic Device Personalization (DDP),⁶ which allows dynamic reconfiguration of the packet processing pipeline within the NIC to meet specific use case protocol stack needs on demand. This can be done by adding new packet processing pipeline configuration profiles to a network adapter at run time without resetting or rebooting the server. For a BNG application, the DDP PPPoE profile enables the NIC to recognize these control plane packets and forward them separately to the control plane, as shown in Figure 8.

Implementing a Run-to-Completion Model
One of the key considerations when designing software-based BNG is ensuring performance scalability per this paper’s opening problem statement. The BNG should be assigned the minimal amount of resources needed to support the current number of active subscribers at any time of the day. This means the BNG must be able to scale both up and down based on the current workload.

The Intel reference pipeline uses a run to completion model to process the uplink and downlink pipelines. As a result, all pipeline functions executed on a packet are run on the same core. This has advantages in that packets do not have to move between cores, thereby minimizing cache misses and overall latency. A direct result of this design pattern is that an individual vBNG instance cannot scale out beyond a single core. Scaling beyond a single core is done by creating a new vBNG instance that runs on a different core. The NIC is programmed (i.e., DDP) on the fly to direct specific subscribers to each new individual vBNG instance.
The combination of a run-to-completion model and a single core running multiple vBNG instances eliminates the need for the orchestrator to understand the internal operation of the vBNG application in order to scale. The orchestrator can scale capacity up or down by increasing or decreasing the number of CPU cores assigned to the vBNG application, enabling linear scalability across a given server. The orchestrator can optimize resource utilization when it is furnished with information regarding the number of subscribers (for a known traffic profile) that a single core instance can support, which may vary by CPU SKU.

### Separating Uplink and Downlink Processing

CPU resource usage by the BNG uplink and downlink pipelines are not symmetric since the downlink normally requires more cycles per packet due to inherently larger packet sizes. In order to effectively schedule a BNG, the Intel reference pipeline splits the uplink and downlink into two separate applications that can be instantiated and then scheduled separately. This separation provides greater flexibility in scheduling and CPU resource usage. For example, a downlink pipeline can be assigned a full physical core (two sibling hyper-threaded cores) while a uplink pipeline might only require half a physical core (one hyper-thread core). Figure 9 shows how the CPU resources of a dual socket server could be partitioned when running eight vBNG instances per socket. Each pipeline can report telemetry individually, and the telemetry database can be used to collate all relevant usage statistics.

### Assigning a Single I/O Connection per Pipeline

The Intel reference pipeline should be run on a BNG data plane server connected to a basic leaf switch that can route both access and core network traffic. With this setup, the switch routes uplink traffic coming from the access network ports to the BNG ports for processing and routes returning packets from BNG uplink pipelines to the core network ports. The flow is reversed for downlink traffic.

As mentioned previously, the amount of uplink traffic is increasing over time, but it is generally only an eighth of the downlink traffic in a wireline network. Therefore, a BNG that uses separate, dedicated physical ports for access and core network port connections is likely to underutilize the available I/O bandwidth of the uplink ports. Instead, sharing physical NICs between upstream and downstream traffic allows I/O bandwidth to be fully utilized.

### Figure 9. CPU Core Allocation Example for eight vBNGs Instances Running per Socket

The routing of subscriber traffic to a vBNG instance is done via a dedicated Single Root Input/Output Virtualization (SR-IOV) connection that can send arriving packets to the vBNG in accordance to its capacity. SR-IOV allows a single physical NIC to be split and shared among multiple pipeline instances, each with its own I/O port. SR-IOV also provides flexibility in the use of physical NICs, such as dedicating a physical NIC to downlink traffic only or sharing a NIC between uplink and downlink traffic. As NIC speeds increase towards 100 gigabits-per-second (Gbps), it is expected that downlink and uplink traffic will share the same physical NIC.

### Figure 10. BNG Downlink Container Using A Single Virtual Function for Both Rx and Tx
Hierarchical QoS (HQoS) is a function that is implemented on the vBNG downlink pipeline. It ensures traffic priority is preserved when traffic coming from the core network is scheduled for transmission on the reduced bandwidth access network pipe to a subscriber, and the available bandwidth on a given port is used efficiently across all users. HQoS must be implemented at the physical port level in order to effectively schedule traffic for transmission on that port. For example, HQoS can help avoid a load balancer having all the ingress traffic pass through a single software block, which could create a bottleneck in the system and decrease flexibility.

As discussed previously, each downlink pipeline has a single virtual function connection for I/O. The following sections describe three models for implementing HQoS:

**Software-Only Model**
A software-only model fully implements HQoS in software. Each vBNG downlink pipeline is allocated part of the port’s overall bandwidth and shapes its traffic per its bandwidth allocation. The advantage of this method is no hardware support is needed, and HQoS processing is shared among the active vBNG instances. The disadvantage of this method is unused bandwidth in one vBNG instance cannot be shared with another instance, which may lead to sub-optimal use of the port’s bandwidth.

**Hardware/Software Hybrid Model**
A hardware/software hybrid model can be used when the NIC has some HQoS offload capability, as shown in Figure 12. The hybrid model is much the same as the software-only model in that each vBNG process runs a software HQoS processing block. The difference is the last level of HQoS is offloaded to the underlying NIC. With the NIC performing last-level arbitration, any unused bandwidth from one vBNG instance can be assigned to a vBNG instance that requires more bandwidth.

**Full HQoS Offload Model**
A full HQoS offload model requires a NIC that can support full offload of HQoS processing from all vBNG instances, as shown in Figure 13. A major advantage of this model is the CPU does not play a role in HQoS, freeing up CPU cycles for other pipeline blocks.

The proposed vBNG architecture supports all three of these models, depending on the capabilities of the underlying hardware.

**Virtualization of the BNG**
The vBNG architecture proposed in this paper does not limit how a vBNG instance is virtualized. For example, either full virtual machine (VM) virtualization or Linux® containers can be used. When a vBNG comprises two individual pipelines, deploying each in a separate container may be better than deploying them in VMs, which is a heavier weight virtualization option. This means in order to deploy a vBNG instance, a pair of containers is launched. A follow on paper will address the deployment and management of vBNG instance containers with Kubernetes®.
By combining all of the architecture considerations previously discussed, it is possible to build a scalable, orchestratable, and CUPS-enabled BNG solution that efficiently uses the I/O and compute resources of an Intel® processor-based server, as shown in Figure 14. This solution can help CoSPs address the need to deliver ever-increasing bandwidth at lower cost, as outlined at the beginning of the paper.

**Performance Benchmarking**

Performance measurements on the blue pipeline blocks shown in Figure 15 were taken on a single socket server with Intel® Hyper-Threading Technology (Intel® HT Technology) enabled. The number of vBNG instances were scaled from one to eight using the 8:1 downlink:uplink traffic ratio. The same traffic profile was applied to all instances (4,000 flows, downlink packet size = 512B, uplink packet size = 128B), and the cumulative throughput (downlink+uplink) across all instances was measured. The optional grey blocks were not enabled.

Figure 16 shows the throughput of an Intel® Xeon® processor-based server (dual socket) running one through eight vBNG instances. The throughput scales almost linearly, and with eight instances deployed, 334 Gbps of traffic is processed. This is achieved using 12 data processing cores (1.5 cores per instance for eight instances).

Figure 17 shows the throughput in packets per second (PPS) in each direction. The uplink processes nearly 45 percent the number of downlink packets despite processing one-eight the downlink bandwidth because the uplink traffic has a smaller average packet size than the downlink traffic. Having separate pipelines for uplink and downlink allows the provisioning to be changed independently in a vBNG if the uplink/downlink traffic ratio or profile changes over time.

Figure 18 shows the breakdown of the cycles per packet spent in each pipeline stage for a single instance. There is a fixed overhead associated with running a pipeline block that is amortized across the number of packets handled in a burst, therefore the number of cycles per packet will increase if a pipeline is less busy.

Since the uplink pipeline does not require a full core to process the traffic, two uplink pipelines can be run on the same physical core, each pipeline using a separate Intel HT core.

For the downlink, the HQoS processing takes half of the total number of cycles of the total pipeline. Due to the packets-per-second requirement, the best performance in this scenario is to allocate a full core to each downlink pipeline, run HQoS on one Intel HT core, and run the forwarding and routing blocks on the other Intel HT core.
Figure 16. Intel® Xeon® Processor-Based Server (Dual Socket) Throughput Running Various vBNG Instances

Figure 17. Intel® Xeon® Processor-Based Server (Dual Socket) Packet Throughput Running Various vBNG Instances

Figure 18. Cycles per Packet For Each vBNG Pipeline Block
**Summary**

The future viability of NFV-based networking equipment running on general-purpose servers hinges on the ability to service ever-increasing traffic volume in a cost-effective manner. This paper presents architectural considerations and benchmarking data that demonstrate the huge potential for NFV-based packet processing. In addition, CoSPs can deliver network connectivity and new services from the edge of the network using a combination of BNG and service-edge solutions.

By rethinking how virtualized network functions are created and deployed, new possibilities arise, such as:

1. Redefining the unit of performance from the number of VMs or containers to the number of cores that deliver a nearly linear increase in uplink and downlink throughput.

2. Creating a model that is virtual network function (VNF) architecture agnostic (i.e., VM, container, or bare metal).

3. Generating a deterministic price per home model that reliably accounts for the impact of data CAGR over the deployment lifecycle.

4. Increasing network availability by converting the large monolith of systems to distributed systems, which enables CoSPs to better manage fault domains and contain affected areas, thus minimizing connection storms and outage times.

5. Creating a multifunction edge infrastructure that can address both NFV and services.

Raw BNG performance is not a definitive metric due to other factors such as network location, subscriber density, average bandwidth per subscriber, and traffic CAGR over the deployment lifecycle. The vBNG architecture presented in this paper allows CoSPs to model uplink and downlink throughput and scale control and user plane function independently on general-purpose servers in a predictable and reliable way.

Future Intel papers will demonstrate specific technologies and software schemes running on Intel processors in ways that help scale performance, address multi-tenancy, and implement intelligent power policy (e.g., time of day scaling), Kubernetes orchestration, and path-to-cloud native applications.

**Appendix**

<table>
<thead>
<tr>
<th>vBNG Server</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Platform</strong></td>
<td>Supermicro® X11DPG-QT</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Intel® Xeon® Gold 6152 Processor, 2.1 GHz, 22 Cores</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>12x16 GB DDR4</td>
</tr>
<tr>
<td><strong>Hard Drive</strong></td>
<td>Intel® SSD DC S3520 Series (480G)</td>
</tr>
<tr>
<td><strong>Network interface Card</strong></td>
<td>8 x Intel® Ethernet Controller XXV710 (16 x 25GbE Ports)</td>
</tr>
<tr>
<td><strong>Software</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Host OS</strong></td>
<td>Ubuntu* 18.04 Linux* Kernel 4.15.0-42-generic</td>
</tr>
<tr>
<td><strong>vBNG</strong></td>
<td>VBNG.L.18.011.0-00040</td>
</tr>
<tr>
<td><strong>Linux Container</strong></td>
<td>Docker version 18.06.1-ce, build e68fc7a</td>
</tr>
<tr>
<td><strong>DPDK</strong></td>
<td>DPDK-v18.11</td>
</tr>
<tr>
<td><strong>BIOS Settings</strong></td>
<td>P-state Disabled, HT ON, C-States Disabled, Turbo Boost Disabled, SRIOV and Vtd enabled</td>
</tr>
</tbody>
</table>
## Application Configuration per Instance

<table>
<thead>
<tr>
<th>Uplink</th>
<th>Downlink</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frame Size:</strong> 128B*; % of Overall Traffic: 11; Subscribers: 4K/Instance; 1x vCPU per Instance</td>
<td><strong>Frame Size:</strong> 512B*; % of Overall Traffic: 89; Subscribers: 4K/Instance; 2x vCPU per Instance</td>
</tr>
<tr>
<td><strong>ACL</strong></td>
<td><strong>ACL</strong></td>
</tr>
<tr>
<td>Blacklist with 150 Rules</td>
<td>Reverse Path forwarding – One Rule per Subscriber (4k)</td>
</tr>
<tr>
<td><strong>Flow Classification</strong></td>
<td><strong>HQoS</strong></td>
</tr>
<tr>
<td>Flows Classified on VLAN Tag Pair</td>
<td>4 Level HQOS – Port, Pipe, Traffic Class, and Queue</td>
</tr>
<tr>
<td><strong>Policer/Metering</strong></td>
<td><strong>Routing</strong></td>
</tr>
<tr>
<td>Two Rate Three Colour Marker</td>
<td>One Route per Subscriber (4k)</td>
</tr>
<tr>
<td><strong>Routing</strong></td>
<td></td>
</tr>
<tr>
<td>Single Forwarding Rule</td>
<td></td>
</tr>
</tbody>
</table>

*Frame size quoted is max size of frame at any point in processing. (e.g. uplink 128Byte =120byte +(2x4Byte access vlan tags))

## Test Environment Configuration Information and Relevant Variables

<table>
<thead>
<tr>
<th>Traffic Generator</th>
<th>Connection Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ixia* Novus* 100GE8Q28 + Fan + 25G</td>
<td>Ixia Ports and DUT Ports Connected Back-to-Back (Sixteen Connections)</td>
</tr>
</tbody>
</table>

---

2. Benchmark and performance tests (“benchmarks”) measure different aspects of processor and/or system performance. While no single numerical measurement can completely describe the performance of a complex device like a microprocessor or a personal computer, benchmarks can be useful tools for comparing different components and systems. The only totally accurate way to measure the performance of your system, however, is to test the software applications you use on your computer system. Benchmark results published by Intel are measured on specific systems or components using specific hardware and software configurations, and any differences between those configurations (including software) and your configuration may very well make those results inapplicable to your component or system.
7. Benchmark and performance tests (“benchmarks”) measure different aspects of processor and/or system performance. While no single numerical measurement can completely describe the performance of a complex device like a microprocessor or a personal computer, benchmarks can be useful tools for comparing different components and systems. The only totally accurate way to measure the performance of your system, however, is to test the software applications you use on your computer system. Benchmark results published by Intel are measured on specific systems or components using specific hardware and software configurations, and any differences between those configurations (including software) and your configuration may very well make those results inapplicable to your component or system.

© 2019 Intel Corporation. All rights reserved. Intel, the Intel logo, and Xeon are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.
Printed in USA 0419/BB/ICMCSW/PDF ☎ Please Recycle 340031US