Quality of Service Performance of Multi-Core Broadband Network Gateways

Rubens Figueiredo, Hagen Woesner
BISDN GmbH
Berlin, Germany
rubens.figueiredo@bisdn.de
hagen.woesner@bisdn.de

Andreas Kassler
Deggendorf Institute of Technology
Deggendorf, Germany
andreas.kassler@th-deg.de
Karlstads Universitet
Karlstad, Sweden
andreas.kassler@kau.se

Holger Karl
Hasso Plattner Institute
Potsdam, Germany
holger.karl@hpi.de

***Abstract*—Broadband network access is typically managed by Broadband Network Gateways (BNGs), which can be implemented as a Virtual Network Function (VNF). This paradigm shift is caused by network softwarization and allows the BNG to be deployed on commodity hardware, significantly reducing capital expenditure (CAPEX). But packet processing operations and complex Quality of Service (QoS) policies make it difficult to provide low and predictable latency at scale for a large number of subscribers. To improve performance, parallel queues at the Network Interface Card (NIC) and multiple dedicated CPU cores for packet processing are used, processing 50 million packets per second on commodity x86 hardware. How to guarantee latency, however, remains unclear. In this study, we conducted testbed-based experiments on a VPP/DPDK implementation of the BNG to benchmark its performance. Our findings reveal how latency and its variation increase with background traffic, and we analyze a parameter that contributes to a trade-off between throughput and latency. We also examine the ability of the multi-core architecture to guarantee latency, at a cost of reduced port utilization. These observations influence the design goal of isolating subscriber traffic and highlight the suitability of software BNG for guaranteeing performance.**

***Index Terms*—Performance measurements, benchmarking, Quality of Service, DPDK, VPP, BNG.**

I. Introduction

Residential and industrial internet access is typically delivered through fixed broadband access networks. To provide network connectivity, services such as Authentication, Authorization, and Accounting (AAA), packet routing and forwarding, and Quality of Service (QoS) enforcement are commonly implemented by a Broadband Network Gateway (BNG), a key component of an operator's network. Broadband operators accommodate a substantial number of subscribers, ranging from 5000 to 35000 subscribers, with traffic rates between 20 and 400 Gbps [1]. To support these rates, a conventional BNG implementation uses high-performance hardware, making it costly to implement and difficult to upgrade to support new functionality. Using the recent trend of softwarization, the BNG can also be implemented in software as a Virtual Network Function (VNF), making it deployable on commodity hardware, typically based on x86 or ARM CPU architectures.

978-3-903176-64-5

While the VNF implementation has significant benefits in flexibility, implementing it at scale, at low and predictable latency in software on commodity hardware is challenging due to the high computational load placed on the BNG [2].

This high computational load on the BNG derives from the QoS policies used to manage subscriber traffic. Concurrent access to network services such as Voice-over-IP (VoIP), internet gaming or video streaming degrades connection quality due to diverse and sometimes conflicting traffic requirements, including throughput and delay. Specifically, access networks enforce QoS policies to meet two requirements: isolation of traffic classes and sharing of unused capacity [3]. To isolate incoming traffic, packets are classified and put into different queues, typically between four and eight queues per subscriber [4]. Subsequently, a packet scheduling algorithm enforces that the traffic rate adheres to either a peak or guaranteed rate, according to specified service level agreements (SLAs) or network requirements. The scheduling algorithm further prioritizes between the different traffic classes, ensuring that critical applications receive preferential treatment when contending for network resources. This dual functionality allocates and manages network capacity to balance the needs of various traffic classes by providing throughput and latency guarantees.

A VNF implementation of this highly complex mechanism is challenging due to the large number of queues and the significant amount of state [2]. To address these challenges and enhance performance of the QoS algorithm, several approaches have been explored, including: 1) hardware acceleration on FPGAs [1] or Network Interface Cards (NICs) [5]; 2) adapting for multi-core environments [6]; 3) and software acceleration via DPDK [7]. For the pure software BNG, further performance scaling in commodity servers relies on using multiple cores in parallel to process workload.

An implementation of the software BNG is proposed in [8], leveraging VPP and DPDK for packet processing, and achieving approximately 122000 queues at 710 Gbps. But an in-depth performance evaluation of the system's capability to isolate traffic remains missing. We close this gap by characterizing latency and latency variation of a software BNG.

In particular, we answer the following question: *What is the impact of the Quality of Service algorithm of a software BNG on delay-critical traffic?* To address this question, we built a testbed and conducted a series of measurements to benchmark several BNG Key Performance Indicator (KPI).

The experiments conducted reveal the potential impact of background traffic on the latency of delay-critical traffic. Multiple instances of the algorithm can protect the delay of specific traffic classes at a cost of sub-optimal port utilization. This establishes a connection between traffic isolation and work conservation. Consequently, a balance needs to be struck between optimizing latency and efficiently utilizing the port's capacity, highlighting the need for further investigation and optimization in this area.

This paper is structured as follows. In Section II, we present the background on access network operator requirements and QoS models. Section III presents the evaluated target. Section IV shows the results of the experiments. Section V presents the answer to the research question. Section VI shows how our work relates to other research work. Section VII concludes our paper.

II. Background

In this section, we present the design of broadband access networks with a focus on the design of the BNG and the QoS model typically applied in these networks.

Several technologies need to work together in order to provide internet access to subscribers. Typically, a subscribers' home gateway establishes a connection to the core network via media like copper or fiber. The BNG terminates the access line, authenticating subscribers and forwarding their traffic to core and private networks [9]. The functionality and deployment of a BNG has been described in the BBF TR-178 [10]. More recently, and accompanying the trend of Software Defined Networking (SDN), the control and packet forwarding functions of the BNG were disaggregated. The BBF TR-459 [9] and the RFC 8772 [11] is a specification for the disaggregated BNG (DBNG), listing the necessary control interfaces between the two components.

We focus on the packet forwarding component of a BNG. Different operations are applied to the packets depending on the direction of the traffic, either *upstream* (from the subscribers to the core), or *downstream* (core to subscribers). Subscriber authentication is done by forwarding specific packets, such as PPP Discovery packets, to the control plane. After the subscriber is authenticated, and to enable the operator to identify each subscriber's traffic, header encapsulations such as VLAN or PPP are used, which need to be either added in the downstream or removed in the upstream. Other functions applied are Access Control Lists (ACL) for verifying incoming traffic is legitimate, accounting of subscriber usage of data plans, and IGMP/ MLD replication for IPTV Multicast. The BNG functionality is depicted in Figure 1.

The BNG also applies QoS functions to police and rate-limit the packets in both directions. The volume of traffic in access networks is asymmetric, with the downstream direction being the larger portion, corresponding to the different services accessed by the different users [12]. While some QoS is applied in both directions, the differences in volume and access patterns simplify the functionality applied in the upstream direction [1]. Traffic in the downstream direction is due to the services conventionally accessed by residential broadband subscribers, which include voice, video, and data services, collectively known as *triple play* [13]. Recently, service offerings have expanded to up to eight different services to include video game streaming or private Virtual Private Networks (VPN), as defined in [4].

These services require varying and sometimes conflicting QoS goals, such as throughput, latency, or packet loss. Beyond individual services, different traffic demands between subscribers may increase latency when sharing the access network [1]. Therefore, two distinct design goals for broadband access networks' QoS are identified: 1) **Traffic class isolation**: different traffic classes must be isolated from each other; 2) **Work conservation**: low-priority traffic classes may reuse *unused* capacity from higher-priority ones. To achieve these objectives, traffic from different subscribers and different services needs to be properly treated. To isolate traffic classes, the BNG classifies incoming traffic into different service classes and puts it into distinct queues. The mapping between service type and destination queue ID depends on packet header fields such as the VLAN PCP, IP TOS or DSCP, used by a classifier. A queue management algorithm, such as Random Early Drop (RED) or Proportional Integral controller Enhanced (PIE), monitors the queue's state and selectively drops packets to prevent congestion [14].

Hierarchical Quality-of-Service (HQoS) is a mechanism that maintains rate limits and transmission priorities for these queues. It organizes schedulers into levels, such as flow-group, subscriber, or network aggregation, applying QoS policies according to the level [2]. These levels consist of three types of nodes: leaf, inner, and root nodes, each with specified Committed Information Rate (CIR) and Peak Information Rate (PIR). Leaf schedulers directly interface with the queues, representing a specific traffic class, and the scheduling discipline is typically configured as strict priority (SP) mode or Weighted Round Robin (WRR). Higher scheduler levels represent traffic aggregates, e.g. subscriber or data-center tenant traffic. Finally, the root scheduler typically represents the physical output port. Packets are selected for transmission based on the token-bucket status for the different traffic classes, and srTCM [15] and drTCM [16] are used to enforce rate and burstiness limits. The structure of the HQoS mechanism is illustrated in Figure 2.

A scheduling algorithm is responsible to serve the queues according to the rates allocated to the different users and the service classes. A specific queue is selected and the scheduler determines whether packets are eligible for transmission. Each queue is fed to a scheduler in Level 0, and the scheduling discipline is typically configured as strict priority (SP) mode or Weighted Round Robin (WRR).

Research on hierarchical schedulers has resulted in a wide

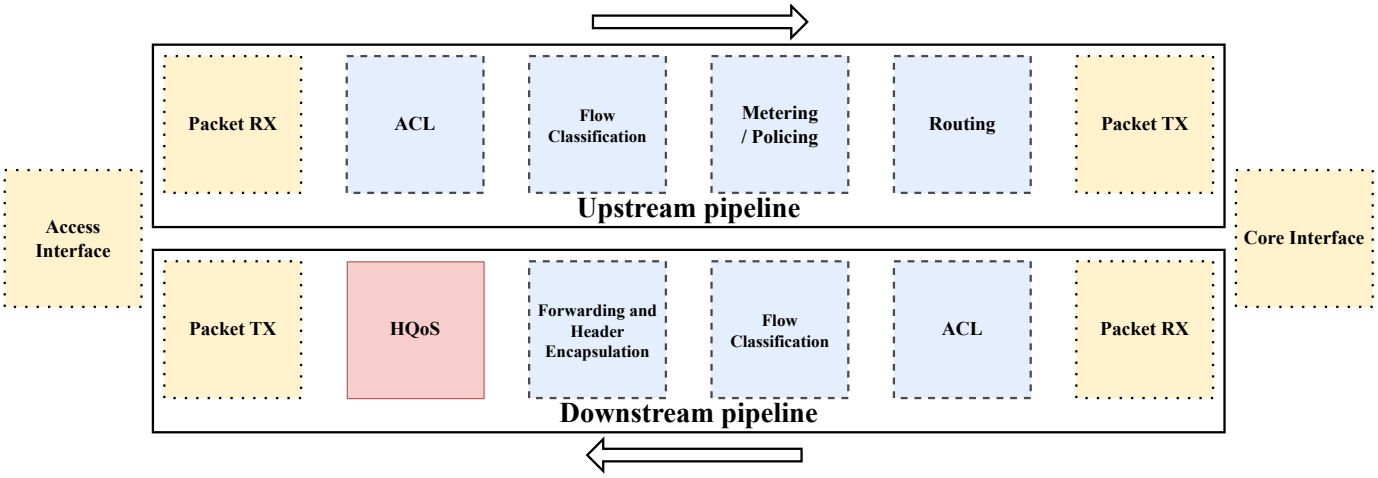


Fig. 1: BNG data plane upstream and downstream pipelines and functional blocks [8]. I/O and NIC-level functionality is represented in yellow; red represents QoS functions, and blue represents packet-header functions.

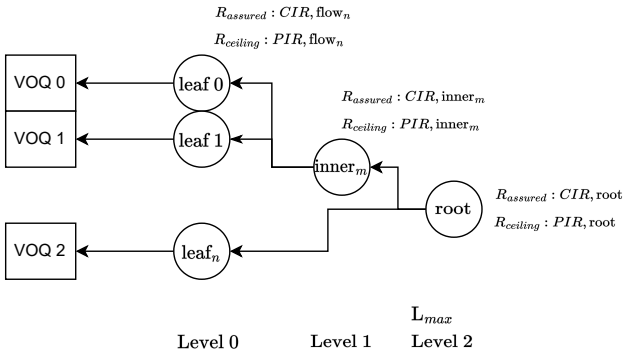


Fig. 2: Illustration of the HQoS mechanism.

array of different algorithms with several design goals and complexity tradeoffs. An ideal formulation of HQoS is realized by the Hierarchical Generalized Processor Sharing (H-GPS) [17]. H-GPS has been shown to not be implementable due to being based on a hypothetical fluid model, where in reality the scheduler can only support one session at a time [17]. Several real-world approximations of H-GPS have been proposed, including Hierarchical Packet Fair Queuing (H-PFQ) [17] or the Multiclass WRR [18]. In practice, these algorithms are typically evaluated by their implementation complexity, worst-case fairness, and delay bounds.

With the HQoS mechanism being adopted in the access network to fulfill the tasks of traffic shaping and rate limiting, ensuring system performance is critical. This is challenging due to the rate demands of up to 400 Gbps and large number of queues for all subscribers. These factors impose requirements in packet processing rate, available memory, and memory bandwidth. To execute this functionality on commodity hardware, a solution is necessary that addresses these requirements.

III. Software-Accelerated BNG

In this section, we describe the software implementation of the BNG, originally proposed in [8]. We present the design of the system, highlighting the design choices that make the framework suitable for fast packet processing. Additionally we discuss the mechanisms used for scaling the system to multiple cores.

A. BNG Software Architecture

The software BNG comprises two distinct components: 1) a header-processing component, which handles packet matching, forwarding, and classification, and 2) an HQoS block, responsible for queuing, shaping, and scheduling the packets. The downstream pipeline is notably more complex than the upstream pipeline, primarily since HQoS computation involves writing and accessing multiple data structures. Keeping in line with asymmetric functional pipelines, one CPU core is exclusively dedicated to the upstream packet processing tasks, while the downstream path is split across two CPUs handling packet processing and HQoS separately. We refer to the threads as PP_{US}, PP_{DS} and $HQoS$, which together form a *BNG instance*. Separating these processes into distinct CPU cores ensures isolated access to Layer 1 and Layer 2 (if not running on the same physical core) cache memories. This architectural decision is key in improving throughput and latency by keeping the instruction and data caches warm with data relevant to each process [19].

The threads are implemented using VPP (Vector Packet Processing) [20], a high-speed packet processing framework. It achieves high speed by combining DPDK and vector processing of packets. Vector processing aggregates packets into a "vector" and applies a set of processing tasks to that vector, thereby improving instruction cache efficiency [20]. Additionally, VPP prefetches data to reduce the impact of data cache misses. Beyond these throughput-improvement features, VPP also provides a comprehensive networking stack that can

be extended to support functionalities such as PPPoE header processing and policers through the development of plugins.

Packet I/O is based on SR-IOV, which allows the creation of Virtual Functions (VF) on top of a PCIe Physical Function (PF) to parallelize access to a device. The PP threads receive packets from the RxQ by polling and applying the respective header-processing functions. After packet processing in the PP_{DS} pipeline, outgoing packets are classified with HQoS-relevant metadata and placed on a ring buffer. The classifier reads the packet header and writes metadata necessary for the $HQoS$ thread. This metadata refers to the placement of each packet in the scheduler. The functionality and design of this structure is represented in Figure 3.

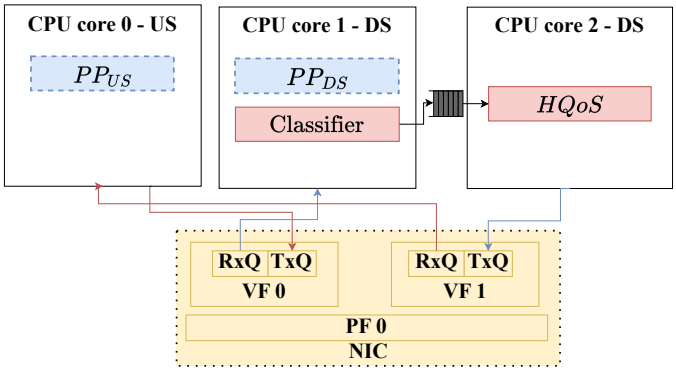


Fig. 3: Functional diagram of the software BNG [8].

The $HQoS$ thread retrieves a batch of packets from the ring buffer and the DPDK QoS framework is used to execute the HQoS process [7]. A batch of packets is prepared to enqueue packets in the scheduler, with each packet directed to the destination queue identified by the metadata written to the packet header. This metadata refers to the IDs of the physical port, subport, traffic class and queue within the traffic class [7]. Reading these data structures requires accessing the DPDK mbuf. For performance reasons, the packet data and the queue position must be present in the CPU Layer 1 and Layer 2 cache, so prefetching is used to populate the cache. As prefetching operations incur some latency, other instructions are executed in parallel in a pipeline. This prefetching pipeline enhances the enqueueing process cache efficiency. If the queue has enough available capacity, packets are appended to the queue, and a bitmap is written to notify the dequeue process to read from those queues.

The subsequent dequeue stage uses a parallel compute engine, which concurrently schedules multiple queues while ensuring that the necessary data structures are present in the higher cache levels. Packets are selected for transmission if the token buckets of the different levels have sufficient credits. Note that only the pointer to the packet is present in the previously described operations, while the packet data itself is kept in Layer 3 cache. Despite these optimizations, the system still behaves as a bottleneck due to the various data accesses and required computations. To mitigate this bottleneck, one strategy is to horizontally scale the processing workload across multiple CPU cores, which we explore in the following section.

TABLE I: DPDK port HQoS scheduler model [7].

Level	Name	L_{max}
3	Port	1
2	Subport	8
1	Pipe	4096
0	Traffic class	13
	Queue	16

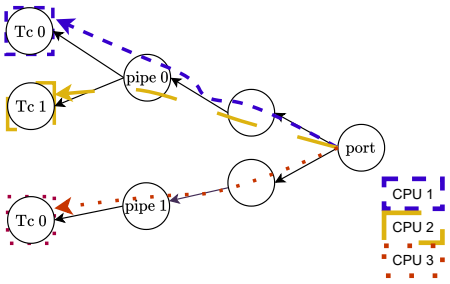
B. Multi-Core Scaling

Supporting e.g. 35000 subscribers, each with 8 traffic classes, would require 280000 queues. This would result in a very large memory footprint that is not feasible to achieve in one x86 CPU core. To address this limitation, the process is scaled horizontally to other CPU cores by introducing additional parallel instances of the BNG [8].

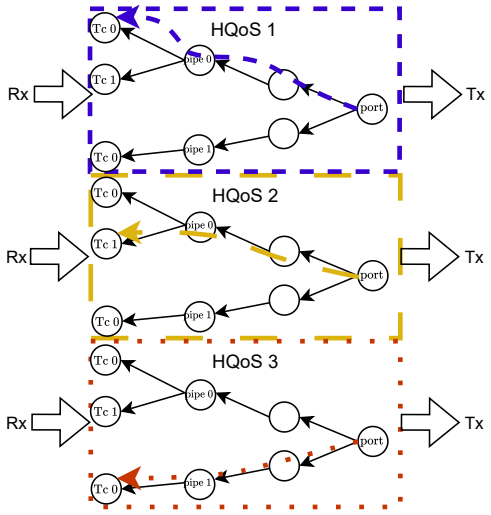
In order to scale packet I/O in servers equipped with commodity hardware, two solutions are used, SR-IOV and multi-queue NICs. Multi-queue NICs can create multiple hardware receive (Rx) and transmit (Tx) queues, improving throughput by processing packets across parallel CPU cores [21]. These techniques are often combined to create multi-queue VFs. Incoming packets are distributed to the different VFs and queues by the NIC by using a packet forwarding technique such as pure MAC switching, Receive Side Scaling (RSS), or advanced hardware switch offloads.

Each polling thread retrieves packets from the queues, handles packet headers, and directs packets to their designated HQoS instance. Each HQoS instance independently manages the enqueueing and dequeueing of packets, operating in isolation from other instances. The PF capacity is statically allocated to each instance, as the state of the HQoS process is not shared between threads. This would require implementing concurrent access to the queue and scheduler structures, via locking mechanisms or utilizing lock-free data structures, both of which adversely impact performance [7]. This challenge is illustrated in Figure 4a, where multiple threads, represented by multiple paths, are accessing the queuing structure, active bitmaps and token bucket data [7]. The lack of shared state between the HQoS threads highlights the primary difficulty with the current solution: while scaling to more CPUs can improve throughput, the lack of synchronization between the threads will lead to sub-optimal use of the port's data rate [8].

The evaluation present in reference [8] offers insights into the achievable system performance by measuring the maximum forwarding rate that the system can sustain with a Packet Drop Rate (PDR) of 0.001%. But this PDR test uses an idealized workload that does not accurately challenge the system's capability to enforce QoS. Specifically, the workload distributes traffic equally across instances and queues; failing to capture scenarios where the port capacity is not optimally utilized [8]. Hence, an in-depth performance evaluation of the system's capability to accurately perform Quality of Service (QoS) remains missing.



(a) Shared access to the HQoS tree [22]. In this design, the different CPUs, represented by the different colors, access the same inner nodes on their way to the leaf.



(b) Independent HQoS execution model. This paper evaluates an implementation of this model.

Fig. 4: Design of the multi-core HQoS model. The figures represent a set of processors accessing a hierarchical scheduler, indicated by the dotted lines and different colors.

IV. Evaluation

Our experiments show the following:

- **Impact of background traffic**: background traffic impacts the latency of delay-critical traffic (Section IV-B);
- **Suboptimal use of port capacity**: uneven load of the BNG instances leads to unused capacity (Section IV-C1).

A. Experimental Setup

Physical Testbed The testbed comprises one x86 server as the Device Under Test (DUT) using Ubuntu 22.04, which is equipped with 2x Intel Xeon Gold 6238R CPUs clocked at 2.2 GHz, 8x32 GB DDR4 RAM operating at 2.6 GHz, and an Intel E810-CQDA2 2x100 Gbps NIC attached to the same NUMA core where the respective BNG instances run, avoiding cross-NUMA performance issues. We tuned BIOS settings [23], including CPU core isolation for running VPP/DPDK applications. Note that in the related work [8], the DUT was equipped with 2x Intel Xeon Gold 6438N CPUs (for differences in CPU see Table II [1]). We manually configured CPU power states using power.py, setting the scaling governor to performance and enabling a turbo frequency of 3 GHz, the maximum the used system allows.

TABLE II: CPUs used for evaluation in this paper and the original one.

Specification	6238R – DUT	6438N – [8]
Number of cores	28	32
Max. turbo frequency	4.00 GHz	3.60 GHz
L3 Cache memory	38.5 MB	60 MB

We use an IXIA Ixload hardware traffic generator connected to the NIC of the DUT with a 100 Gbps fiber cable, providing latency measurements with nanosecond time precision. We measured IXIA to DUT delay of 24 μs which was subtracted from the results present in the following sections.

Key Performance Indicators: We measure the maximum sustained throughput, latency and jitter. Latency is defined as the difference between the packet transmit time from the packet arrival time, measured with a a cut-through method [24], and the minimum, average and maximum latency values are reported. Jitter refers to the latency variation is obtained by the difference between the latency of two consecutive packets.

Application configuration: In our measurements, we evaluate the performance of an IPoE based BNG, where a VPP Layer 3 forwarding function forwards the packets, executed in the PP_DS threads. We used VPP version v23.10 and DPDK v23.07. DPDK's rte_sched library was based on the version used in [8], which adds one scheduling hierarchy level when compared to the version in upstream DPDK. We have created eight VFs from one PF, each with a Rx/Tx queue pair. One BNG instance, consisting of one PP_{DS} and one $HQoS$ thread, is attached to two VFs, with a total of four BNG instances created. One 100 Gbps PF was split into four different VFs, resulting in a shaper configuration of 25 Gbps per VF. The 25 Gbps capacity is equally divided equally by all pipes and traffic classes, resulting in a peak shaping rate of 6.1 Mbps for a pipe and 1.5 Mbps for a TC. Contrary to the original implementation we did not consider the upstream packet processing thread, as we are focused on benchmarking the performance of the system while applying QoS for downstream packets. Moreover, to study the application in isolation, the BNG instances are run inside a VPP process, as opposed to creating multiple containers. To improve throughput between NIC and PP_{DS}, the maximum of 4096 descriptors was configured in the NIC. To reduce latency for each individual traffic class, the queue sizes for each individual traffic class is 8 packets long. Finally, the queue size between the processes is 512 packets long, corresponding to two times the maximum size of the VPP vector size [20].

Workload parameters: To evaluate the impact of background traffic on latency and throughput, we conducted ex-

[1] https://ark.intel.com/content/www/us/en/ark/compare.html?productIds=232397,199345

periments with background and VoIP-like traffic. All packets generated are IP/UDP, with the used packet sizes and packet rates being specified in the following sections. Each PP_{DS} instance has a MAC address, and we use the destination IP address to identify a single user. Background traffic is created with 4094 users, and VoIP-like traffic is directed to 200 users. We use the first two IP addresses for the network and the VPP instance IP address, hence we do not create traffic destined to the maximum of 4096 users per instance. Furthermore, all users have up to four traffic classes, identified based on the TOS field.

Table III describes the chosen configuration of the VPP/ DPDK parameters.

TABLE III: VPP/ DPDK scheduler configuration

Parameter	Setting	Value
Scheduler	Port Line rate	100 Gbps
	Subport	25 Gbps
	Subport Tc	25 Gbps
	Pipe PIR	6.1 Mbps
	Tc PIR	1.5 Mbps
Profiles	# Subport profiles	1
	# Pipe profiles	4096
	# Tc p/ pipe	4
Queue sizes	Tc queue size	8 Packets
	Ring buffer	512 packets
	Rx/Tx descriptors	4096 packets

Calibration We measured the maximum Non-Drop Rate (NDR) to be 12 Mpps for 576 B packets on a single core.

Dataset The dataset with the experiments is available on github [2].

B. Single-core performance

In this section, we present the results of the experiments done with a BNG configuration consisting of one PP_{DS} and one $HQoS$ thread.

1) Comparison to the state of the art: We use the evaluation present in [1] to provide a point of comparison. We aim to characterize the performance of latency-critical traffic under various amounts of traffic load. For this test we disabled batching of packets in the $HQoS$ thread. As in [1], 10.000 packets of size 178 Byte marked with TOS=0 are created at a rate of 2.000 packets/s, representing the latency-critical traffic with VoIP-like packets. The background traffic is modeled with IMIX packets (64 B, weight: 7; 594 B: weight 4; 1500 B: weight 1), and the target background traffic data rates of 0, 1 and 9 Gbps were created, using the TOS 1 and 2 fields of the header. Two main factors distinguish our evaluation from the one in [1]. First, we are testing an IPoE-based BNG, while Kundel et al. experimented with a PPPoE-based BNG. The added packet processing and header fields in the packet would add a constant load to all packets in the PP_{DS} thread, but $HQoS$ processing remains the same. Second, as the DPDK QoS scheduler is optimized for many queues [7], the performance of a single queue, as measured in [1] would not result in proper values. With this in mind, and in order to apply the same rate to the BNG as used in [1], we chose to measure the latency of 200 queues and average the overall delay.

Table IV presents the results of this experiment. For all scenarios, no packet loss is visible. As for the latency, in the best-case scenario with no background load, the minimum latency of the traffic class is higher than the latency of the hardware PPPoE BNG, but the delay variation is relatively low. As expected for the software IPoE, with increased background load, all latency metrics increase. When the background load increases to 9 Gbps, a noticeable increase of all metrics is visible. The increase in latency due to the increase in background load is consistent with the reported results from FlowValve [25]. The increased latency in the DPDK scheduler is due to the increased number of parallel queues, increasing the number of destination queues and token structures that must be loaded into memory.

Takeaway 1: Increasing background traffic affects the latency and latency variation of delay-critical traffic.

2) Batch size: With this experiment, we aim to characterize the latency of the system with different batch sizes feeding the $HQoS$ process. With the batch configuration of the previous experiment, increasing the load beyond 4 Mpps to the $HQoS$ will result in packet drops. To ensure that the batch process remains constant, we model the background traffic with fix-sized 512 Byte packets.

Figure 5 shows the result of the experiment. For emphasis, we plot the average and minimum latency in Figure 5a and the box-and-whiskers plot of the maximum latency variation in Figure 5b. With the increase of batch size from 0, there is a small decrease in the average latency, but maximum latency variation increases.

Unlike the adaptive batch parameter in the PP_{DS} thread, the batch parameter we consider in this experiment is not adaptive to the incoming traffic load. While providing a simple mechanism to increase throughput, this experiment shows the necessity of tuning this parameter for the QoS characteristics.

Takeaway 2: The HQoS may be tuned for throughput, latency or latency variation using the batch size parameter.

C. Multi-core performance

In this section, we assess the performance of multiple instances of BNG. We create four instances using in total four PP_{DS} and four $HQoS$ threads.

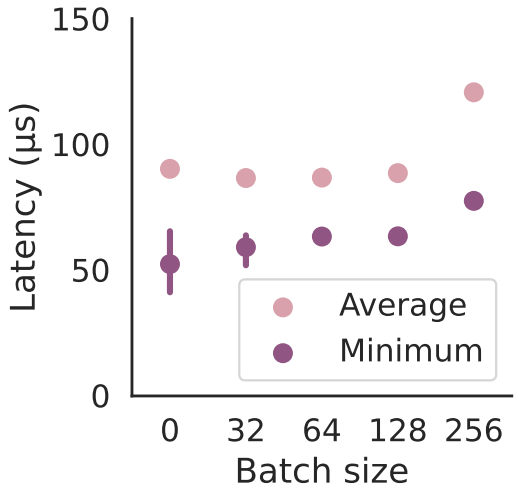
1) Use of port capacity: In this experiment, we illustrate the sub-optimal utilization of port capacity in a multi-core $HQoS$ setup. The goal of this experiment is to demonstrate the impact of non-uniform traffic distribution on the utilization of port capacity. To achieve this, we examine two scenarios. 1) **4 Threads:** All threads are equally utilized. 2) **2 Threads:** We only generate traffic towards two threads, while the other remain unutilized. We use throughput as performance metric.

Figure 6 shows the result of the experiment. As anticipated, the 4 threads scenario demonstrates performance close to line rate, with degradation starting when the maximum transmission capacity is reached. In the 2 threads scenario,

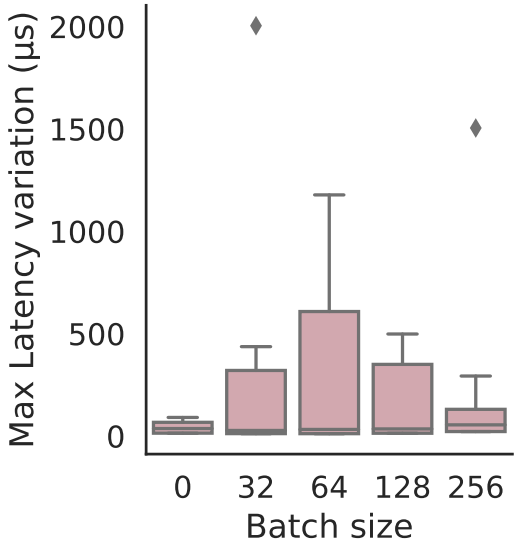
[2]https://github.com/rubensfig/research_experiments/tree/tma_paper

TABLE IV: Performance of the VPP/ DPDK BNG for 178 Byte probe packets. For reference we included the results from the related work [1].

Background Load	BNG	# Packets	Loss (%)	Minimum latency (μs)	Maximum latency (μs)	Average latency (μs)	Average latency variation (μs)
0 Gbps	Software IPoE	10000	0	7.45	25.76	8.15	0.38
	Hardware PPPoE	10000	0	2.02	12.85	7.24	3.00
1 Gbps	Software IPoE	10000	0	7.35	24.85	10.48	1.82
	Hardware PPPoE	10000	0	2.01	12.76	7.39	3.00
9 Gbps	Software IPoE	10000	0	20.03	105.5	31.09	59.06
	Hardware PPPoE	10000	0	2.01	14.47	7.34	3.00



(a) Effect of batch size in the latency of the VoIP traffic.



(b) Effect of batch size in the maximum latency variation of the VoIP traffic. We plot the box-and-whiskers value of the five experimental runs, showing the median and percentiles of the results.

Fig. 5: Effect of batch sizes in latency and latency variation of the VoIP traffic.

two effects can be observed. First, the unused capacity from the first thread is not distributed to the other threads processing traffic, leading to an effective maximum port capacity of 50 Gbps. This is caused by the lack of shaper synchronization, limiting each thread to the pre-configured maximum share with no possibility to temporarily borrow credits from the non-utilized threads. Second, increasing the offered load to the system beyond 50 Gbps will drive the instance into an overload situation, causing packet drops and degrading overall performance.

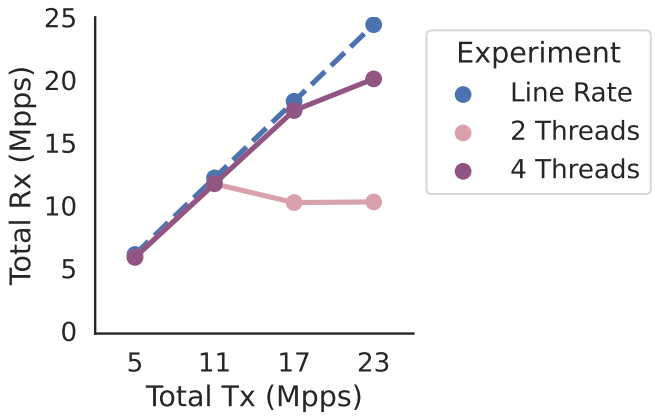


Fig. 6: Sub-optimal utilization of port bandwidth. We plot the 95% CI for all experiments, but the bar is too narrow to be visible.

Takeaway 3: The lack of a global scheduling tree results in sub-optimal utilization of port capacity, breaking work conservation.

2) Traffic isolation between instances: In this section, we show how scaling BNGs instances keeps the latency of the different instances isolated. We demonstrate the impact of non-uniform traffic distribution for different instances of the BNG on the latency. We create a 'uniform' and 'non-uniform' scenarios for workload. In this case, a 50 Gbps rate was fixed and distributed to each instance. In the 'uniform' scenario, all were transmitting 12.5 Gbps, and for the 'non-uniform' distribution, the first instance sends 8 Gbps and the last sends 16 Gbps.

Figure 7 shows the result of the experiment. We demonstrate that sending high-demand flows to a specific instance does not affect the delay of the other instances. Figures 7b and 7c plot

the minimum, average and maximum latency. The extremely high values of maximum latency is attributed to packet loss in the DUT of about 1.2% for the uniform case, and 1.8% for the non-uniform case in the PP_{DS} threads. The difference in packet loss must be further investigated, as for the two experiments is not explained, as the packet loss was not observed in the previous experiments.

Takeaway 4: Pinning high-volume traffic on specific instances is a viable strategy to protect the latency of traffic on other instances.

V. Discussion

We are now able to answer our original question: *What is the impact of the Quality of Service algorithm of a software BNG on delay-critical traffic?* This impact is that the latency of delay-critical traffic increases with background traffic, which violates the BNG design requirement of traffic isolation.

This observation impacts the functionality of the system to provide performance guarantees. First, due to the impact of background traffic in the latency of delay-critical traffic, handling traffic with different requirements from multiple users in the same instance might violate the latency guarantees of the different users. Second, the batching mechanism used for improving throughput has a negative impact on the latency and latency variation, which are among the most commonly cited QoS metrics for VoIP [26]. An HQoS process that is handling delay-critical traffic may be optimized for a throughput or latency scenario, but a solution that addresses all use cases is missing.

The parallel instances of the BNG execute the $HQoS$ algorithm independently, and higher traffic demands in one instance does not affect the others. Hence, these observations motivate a mechanism that distributes traffic according to their QoS requirements. A traffic distribution method must ensure load balancing between instances, as the independent calculation of the $HQoS$ potentially disrupts work conservation, resulting in decreased use of the port capacity. Moreover, the multi-queue NIC must also manage its queues, and the effect of the NIC-level transmission scheduling functionality in the BNG KPI is still unclear.

Therefore, we propose the following enhancements to the software BNG as future work:

- Implementing an adaptive batching mechanism for the $HQoS$ process;
- Investigating the NIC-level QoS functionality;
- Developing a QoS aware flow distribution method;
- Extending the DPDK QoS framework to facilitate multi-threaded execution.

We acknowledge the threats to validity of our study. The comparative study of software DUT and the Tofino/FPGA [1] was conducted with a different header modification functionality. Specifically, our BNG processes IPoE traffic, while the Tofino/FPGA is based on PPPoE. The complexity of header operations differs between these two scenarios, where the PPPoE is the more complex workflow. We expect that this difference would increase the latency equally for all packets, but the change of performance was not studied in this paper. While the change in performance for the Intel Tofino should be minimal, the change in performance for the x86-based software DUT is unclear. While we still find that our study was valid, any comparative analysis between the two systems must be analysed with this information.

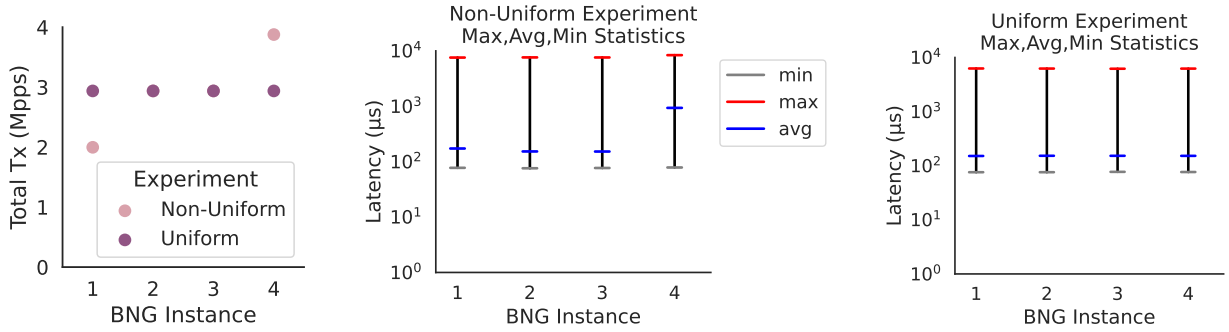
Another limitation of our study is the unexplored variations in scheduler rate configuration, WRR schedulers or AQM methods. These unexplored aspects could have implications for the system performance, and should be investigated in future studies.

VI. Related Work

Many academic and industry studies have been exploring the design of BNG and their execution models. In this section, we analyze the existing literature for performance evaluation methodologies of the BNG, and for proposals to improve the $HQoS$ performance.

OpenBNG [1] executes the BNG in an accelerated environment using programmable hardware accelerators. This approach splits the functionality in a heterogeneous execution environment, where packet processing is executed in P4 targets while an FPGA runs the HQoS scheduler. The FPGA is necessary to address the shallow buffers of the Tofino ASIC, which limits the space available for implementing queues. They presented a comprehensive latency and jitter characterization of their system. We adopted their evaluation methodology to evaluate the latency of VoIP streams under increasing background load. The hardware support of the OpenBNG allows for stable latency for all tested scenarios. Another notable advantage of OpenBNG's hardware-based approach is its scalability, with the FPGA providing ample space for all queues, facilitating scalable deployment of the BNG.

Fejes et al. [2] propose a hierarchical QoS system utilizing a Per Packet Value (PPV) marker. This approach assigns values to each packet based on their header fields, thereby grouping packets into traffic aggregates according to a Throughput-Value Function. A scheduler utilizes this information to map the traffic distribution of a traffic class into an aggregate distribution of all the input traffic, thus achieving a multi-level split of the capacity. Unlike traditional HQoS systems, which require complex queueing and hierarchy management, this PPV-based system simplifies the implementation of complex HQoS scheduling. They presented an in-depth evaluation of the HQoS with dynamic and static scenarios. The dynamic scenarios show the transient performance of the system to accurately perform the prioritization and shaping of the different traffic classes. The static scenarios evaluate the throughput deviation of the system, defined as $\frac{x_i}{\text{ideal}_i} - 1$, with different mixes of background traffic. They show the PPV concept is able to implement the HQoS concept without resorting to managing large numbers of queues and maintaining the complex scheduling structure. Our analysis diverges from this study, as we focus on the latency performance of the system. Integration of this HQoS into the BNG as not yet been done.



(a) Uniform and non-uniform workload used in this experiment.

(b) Latency of the different $HQoS$ instances under non-uniform workloads.

(c) Latency of the different $HQoS$ instances under uniform workloads.

Fig. 7: Latency traffic isolation and non-uniform workload application.

In [3], one proposal to execute HQoS in a multi-core manner is described, aiming to improve the performance of the Linux Kernels' HTB. The approach involves separating the enqueue and dequeue operations to independent cores, with scheduling eligibility determined during dequeue, thereby avoiding the need for locking mechanisms. The systems' throughput and number of queues was evaluated through experiments, assessing the capability to shape to two types of services. While our work differs in its focus on user space networking, the proposal of decoupling the enqueue and dequeue mechanism might be applicable to the DPDK scheduler, potentially improving performance at the expense of CPU cores.

Lemeshko et al. [27] proposed a two-level hierarchical QoS system running in a multi-core environment. This system manages capacity by macro-queues and sub-queues. Sub-queue calculation function is kept internal to each CPU thread, while the upper-level functions for macro-queues are overseen by a coordinator thread. Although the original study solely presented a numerical analysis of the system, the results indicated a scalable algorithm. This approach holds promise to alleviate for addressing challenges related to separate instances and preserving work conservation.

VII. Conclusions

Implementing complex packet-processing functions on readily available hardware save costs for operators. In this paper, we have investigated the packet-forwarding performance of a software BNG, a crucial piece of infrastructure for network connectivity in broadband access networks. Through real testbed experiments, we benchmarked the latency resulting from a software BNG implementation. Our experiments showed the negative impact of background traffic on latency and latency variation of delay-critical traffic. We demonstrated how batch configuration and multi-core scaling may be used to counteract the latency increase, with some limitations. First, the static batch parameter is not adaptive to the traffic load of the system, potentially increasing the latency in low-load scenarios. Second, the non-uniform distribution of traffic to instances in the multi-core scaling scenario results in sub-optimal utilization of port capacity. These observations impact the ability of the software BNG to provide performance guarantees.

Acknowledgment

Parts of this work have been funded by the Bavarian State Ministry of Education and Culture, Science and Art through the High-Tech Agenda (HTA) and the Knowledge Foundation of Sweden (KKS) through project DRIVE. We thank the help of the team at BISDN and the anonymous reviewers for their feedback.

Ethical Considerations

This work does not raise any ethical issues.

References

[1] R. Kundel, L. Nobach, J. Blendin, W. Maas, A. Zimber, H.-J. Kolbe, G. Schyguda, V. Gurevich, R. Hark, B. Koldehofe, and others, "OpenBNG: Central office network functions on programmable data plane hardware," *International Journal of Network Management*, vol. 31, no. 1, p. e2134, 2021. Publisher: Wiley Online Library.

[2] F. Fejes, S. Nadas, G. Gombos, and S. Laki, "DeepQoS: Core-Stateless Hierarchical QoS in Programmable Switches," *IEEE Transactions on Network and Service Management*, vol. 19, pp. 1842–1861, June 2022.

[3] Z. Li, N. Yu, and Z. Hao, "A Novel Parallel Traffic Control Mechanism for Cloud Computing," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, (Indianapolis, IN, USA), pp. 376–382, IEEE, Nov. 2010.

[4] A. Nowicki, "AGF Functional Requirements," July 2023.

[5] Y. Kuperman, M. Mikityanskiy, and R. Efraim, "Hierarchical QoS Hardware Offload (HTB)," https://netdevconf.info/0x14/pub/papers/44/0x14-paper44-talk-paper.pdf.

[6] C. Jia, Z. Fu, X. Hu, S. Cao, L. Wang, and J. Li, "Multi-core HTB for bandwidth sharing," in *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, (Ithaca New York), pp. 169–171, ACM, July 2018.

[7] "56. Quality of Service (QoS) Framework — Data Plane Development Kit 23.11.0 documentation." https://doc.dpdk.org/guides/prog_guide/qos_framework.html.

[8] J. Pažej, A. Duignan, and J. Singh, "Power Efficiency of a Cloud-Native Broadband Network Gateway (BNG) Using 4th Gen Intel® Xeon® Scalable Processors." https://www.intel.com/content/www/us/en/content-details/786305/enhancing-performance-and-power-efficiency-of-a-cloud-native-broadband-network-gateway-bng-using-4th-gen-intel-xeon-scalable-processors.html.

[9] "TR-459: Control and User Plane Separation for a disaggregated BNG," Technical Report TR-459, Broadband Forum, 2020. https://www.broadband-forum.org/technical/download/TR-459.pdf.

[10] "TR-178: Multi-service Broadband Network Architecture and Nodal Requirements," Technical Report TR-178, Broadband Forum, Sept. 2014. https://www.broadband-forum.org/technical/download/TR-178_Issue-1.pdf.

[11] S. Hu, D. E. Eastlake 3rd, F. Qin, T. M. Chua, and D. Huang, "The China Mobile, Huawei, and ZTE Broadband Network Gateway (BNG) Simple Control and User Plane Separation Protocol (S-CUSP)," Request for Comments RFC 8772, Internet Engineering Task Force, May 2020. Num Pages: 124.

[12] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On dominant characteristics of residential broadband internet traffic," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, (Chicago Illinois USA), pp. 90–102, ACM, Nov. 2009.

[13] N. Solihah and M. I. Nashiruddin, "Performance Evaluation of the 10 Gigabit Symmetric PON for Triple-Play Services," in *2020 IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*, (Batam, Indonesia), pp. 136–143, IEEE, Dec. 2020.

[14] G. Hong, J. Martin, and J. M. Westall, "On fairness and application performance of active queue management in broadband cable networks," *Computer Networks*, vol. 91, pp. 390–406, Nov. 2015.

[15] J. Heinanen and R. Guerin, "A Single Rate Three Color Marker," Request for Comments RFC 2697, Internet Engineering Task Force, Sept. 1999. Num Pages: 6.

[16] J. Heinanen and R. Guerin, "A Two Rate Three Color Marker," Request for Comments RFC 2698, Internet Engineering Task Force, Sept. 1999. Num Pages: 5.

[17] J. C. R. Bennett and H. Zhang, "Hierarchical Packet Fair Queueing Algorithms," vol. 5, no. 5, 1997.

[18] H. Chaskar and U. Madhow, "Fair scheduling with tunable latency: A round-robin approach," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 592–601, Aug. 2003.

[19] P. Zheng, A. Narayanan, and Z.-L. Zhang, "A Closer Look at NFV Execution Models," in *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*, (Beijing China), pp. 85–91, ACM, Aug. 2019.

[20] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi, "High-Speed Software Data Plane via Vectorized Packet Processing," *IEEE Communications Magazine*, vol. 56, pp. 97–103, Dec. 2018. Conference Name: IEEE Communications Magazine.

[21] G. Kim and W. Lee, "Network Policy Enforcement With Commodity Multiqueue NICs for Multitenant Data Centers," *IEEE Internet of Things Journal*, vol. 9, pp. 6252–6263, Apr. 2022.

[22] A. Chandra and P. Shenoy, "Hierarchical Scheduling for Symmetric Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, pp. 418–431, Mar. 2008.

[23] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "How Low Can You Go? A Limbo Dance for Low-Latency Network Functions," *Journal of Network and Systems Management*, vol. 31, p. 20, Jan. 2023.

[24] L. Avramov and jhrapp@gmail.com, "Data Center Benchmarking Terminology," Request for Comments RFC 8238, Internet Engineering Task Force, Aug. 2017. Num Pages: 20.

[25] S. Xi, F. Li, and X. Wang, "FlowValve: Packet Scheduling Offloaded on NP-based SmartNICs," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pp. 347–358, July 2022. ISSN: 2575-8411.

[26] B. Adhilaksono and B. Setiawan, "A study of Voice-over-Internet Protocol quality metrics," *Procedia Computer Science*, vol. 197, pp. 377–384, 2022.

[27] O. Lemeshko, O. Yeremenko, L. Titarenko, and A. Barkalov, "Hierarchical Queue Management Priority and Balancing Based Method under the Interaction Prediction Principle," *Electronics*, vol. 12, p. 675, Jan. 2023.