DeepQoS: Core-Stateless Hierarchical QoS in Programmable Switches

Ferenc Fejes, Szilveszter Nádas, Gergő Gombos, Sándor Laki, *Member, IEEE*

***Abstract*—Novel applications and network scenarios challenge existing traffic management strategies. Hierarchical Quality of Service (HQoS) provides a fine control of resource sharing and delay, but traditional HQoS solutions have challenging complexity that prevents their deployment in the traffic management engine of high-speed switches. Programmable switches have emerged to make the packet processing pipelines flexible and reconfigurable, but their traffic management capabilities still rely on fixed functions that cannot handle the complexity of traditional HQoS approaches. In this paper, we show how the extended programmability can help in addressing this challenge by the application of a fundamentally different algorithm. To emulate HQoS behavior we extend our core-stateless resource sharing framework called Per Packet Value (PPV) with a HQoS-packet marker architecture called DeepQoS. DeepQoS can be used to mark resource sharing policies of different layers simultaneously and effectively at a single point, e.g., ensuring the fair share of a household's traffic within an access aggregation network, while also controlling the shares of its subflows. In the PPV framework, bottleneck scheduling is very simple and is unaware of flows and policies, which are encoded to Packet Values. DeepQoS can use these existing simple PPV schedulers without any change. To demonstrate the deployability of the proposed method, we have created the DeepQoS marker implementation in DPDK while redesigned and implemented our packet scheduler called Virtual Dual Queue Core Stateless Active Queue Management (VDQ-CSAQM) on a P4-programmable switch. Using extensive measurements we demonstrate the unique capabilities of DeepQoS to realize rich and deep HQoS.**

***Index Terms*—Computer network management, Quality of service, Resource management, Software defined networking.**

I. Introduction

Good Quality of Service in novel applications such as AR/VR, cloud rendered gaming, HD or holographic video conferencing and remote presence requires high bandwidth, low latency or both. End users connect to the Internet with different subscriptions and access properties. As gigabit-speed access links became widespread, the possibility of temporal and even permanent overloads in the access aggregation network (AAN) has increased. These periods can be handled by over-provisioning, but it has a high price: high infrastructure cost and underutilization in most of the time. Hierarchical quality of service (HQoS) is a widely adopted solution to ensure complex resource sharing policies in AANs, where resource sharing is controlled within and among traffic aggregates (TAs), e.g., between virtual operators, network slices, users, or subflows of users. Nowadays, HQoS is typically enforced in the border gateway of the access network, since all traffic going towards or coming from the Internet flow through this node. Note that this solution puts high computational load on the gateway node and has further limitations.

Another area where HQoS has potential benefits is cloud networking where it can ensure complex QoS policies (including resource sharing and latency requirements) between tenants at the highest level as well as among flows of each tenant at bottom of the hierarchy. The access to the shared resources at tenant level is described in the Service Level Agreement, reflecting the per-tenant payment granularity applied by today's data centers. However, inside the tenant there may be various sub-services whose traffic shares the capacity allocated to the tenant and thus differentiation between the subflows can greatly improve efficiency and flexibility of tenant management and operation. Moreover, HQoS also has potential benefits in any other physical network infrastructures (e.g., private WAN, 5G/6G RAN, etc.) that are shared among virtual networks (e.g., slices, virtual operators). It can solve not only the isolation of virtual networks at the top level but the differentiation between traffic groups inside the virtual slices.

Implementing HQoS is challenging due to its high complexity. Traditional solutions require per-flow states and a complex queue hierarchy to be configured and managed [1]. Such complexity exceeds the traffic management (TM) capabilities of high-speed hardware switches including P4-programmable ones.

In this paper, we show how this challenge can be addressed with the extended programmability of P4-switches and our core-stateless resource sharing framework called Per Packet Value (PPV) [2]. Similarly to other core-stateless resource sharing proposals, the PPV framework consists of two key components: 1) A packet marker that tags each packet with a packet value according to the predefined QoS policy and 2) an Active Queue Management (AQM) algorithm or scheduler that solely uses the packet value carried by each packet to decide which packet to drop or mark with Explicit Congestion Notification (ECN) Congestion Encountered (CE) flag in case of network congestion. The original PPV framework assumes flat policy hierarchy and thus it only ensures flexible resource sharing between traffic aggregates (TAs) where a TA could be, e.g., a flow, the aggregated traffic of an application, a subscriber or a tenant. To support a multi-level policy

Manuscript submitted June 15, 2021. The research of S. Laki was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. S. Laki and G. Gombos also thank the support of the "Application Domain Specific Highly Reliable IT Solutions" project that has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme. *(The corresponding author: S. Laki, e-mail: lakis@elte.hu)*

F. Fejes and Sz. Nádas are with Ericsson Research, Budapest, Hungary.

G. Gombos and S. Laki are with ELTE Eötvös Loránd University, Budapest, Hungary.

hierarchy, we extend the packet marker component of PPV so that it can encode complex HQoS policies among the flows of each TA into packet values. The proposed new method called DeepQoS enables handling of policy hierarchies built from strict priority and weighted fair resource sharing components with arbitrary depth (as in traditional HQoS policies). The modified marker also ensures that the desired resource sharing between TAs still holds as in the original PPV framework. DeepQoS only requires changes in the packet value marking algorithm, letting the PPV-based scheduler untouched. In addition to HQoS policy encoding, we present the first realization of our Virtual Dual Queue Core-Stateless Active Queue Management (VDQ-CSAQM) [3], a PPV-based AQM method supporting both Classic and L4S (Low Loss, Low Latency and Scalable throughput) Internet services in programmable hardware switches.

In summary, the key contributions of this paper include:

- We propose DeepQoS, an extended PPV packet marker to implement complex HQoS policies (see Sec. V).
- We show and prove that DeepQoS keeps the TA-level resource sharing properties of the original PPV framework (see Sec. VII and Appendix A and B).
- We provide the data and control plane design for our PPV-based L4S scheduler called VDQ-CSAQM (see Sec. VI).
- We have implemented the prototypes of DeepQoS marker in DPDK and VDQ-CSAQM on an Intel/Barefoot Tofino-based switch that are used in the performance evaluation.
- We present thorough measurements in our high-speed testbed to demonstrate the viability of the proposed method under realistic traffic mixes (see Sec. VII).

II. Related Work

The area of sharing network resources have a rich literature with fundamentally different approaches. In this section, we aim to give an overview of the most important related work.

A. Stateful Resource Sharing

The majority of existing resource sharing solutions deployed in production networks rely on the paradigm of weighted fair queueing and strict priority. These methods use flow states at the bottleneck nodes for packet scheduling. Fair queueing methods like [4], [5], [6] propose to map flows to a number of queues. These solutions only work well if the number of queues is in the same order of the number of flows. This can be handled in software implementations easily where high performance is not a requirement but hardware solutions are too expensive and/or have performance constraints. High-speed hardware switches only have a limited number of queues per egress port; they were simply not designed for supporting per-flow queueing.

In addition, if we want to use a different resource sharing policy, it is hard to change since it require the reconfiguration of several network devices. One solution for this problem is the OpenQueue [7] that provides a language in which we can easily add or change policies and manage the buffers in run-time at a high abstraction level.

Some recent proposals aim to handle these issues for data center and WAN environments and go further by applying policies described by utility functions to control weighted queues of the bottleneck nodes or by enabling hierarchical resource sharing with flexible policy definition. For example, NUMFabric [8] aims at solving the network utility maximization problem by splitting it between end-hosts and switches, and introducing a weight exchange protocol between the two, promising fast convergence times in data center environments. Though NUMFabric provides very flexible means to describe a rich set of policies, it still applies traditional weighted fair queueing in the switches.

The BwE [9] is another approach. It uses rate limiting instead of weighted queues to manage the resource sharing for a globally-deployed private WAN. It introduces a bandwidth function to define flexible policies. BwE also shares the idea of the PPV framework [10] that routers cannot support the scale and complexity of enforcing per-flow policies. The difference between the two solutions is BwE uses a centralized control while PPV solves the throughput allocation in a fully distributed way. Note that bandwidth function policies of BwE can naturally be mapped to PPV policies and vice versa.

A virtual queue based per slice traffic management for P4 switches is proposed in [11]. While it extends the traffic management capabilities of the switch, it still requires traffic classification and per traffic aggregate states, therefore it cannot be extended for a complex HQoS hierarchy.

Flow Queue algorithms, like fq-codel [12] can solve equal resource sharing among a moderate number of flows. Extending that to more flows, to HQoS hierarchies and flexible policies beyond flow fairness is not trivial.

While [13] shows that implementing fair queueing for large number of flows is possible, it still requires flow identification and policy knowledge in every bottleneck.

B. Stateless resource sharing

Several resource sharing architectures have been proposed in the past decade that work without any per-flow states inside the network. Such architectures typically implement two mechanisms: 1) stateful packet tagging implemented at the edge of the network (i.e., Edge Nodes); 2) stateless packet scheduling implemented by all the nodes in the core of the network (i.e., Core (or Resource) Nodes) that handles (e.g., drop, ECN mark, prioritize, etc.) incoming packets solely based on their tags. According to this terminology, all the packet forwarding elements that could potentially be a bottleneck in the network are Core Nodes, including Edge Nodes as well.

The most widely known of such architectures is Diff-Serv [14] where markings identify a set of pre-defined policies called Per-Hop Behaviors (PHBs) to be applied by the routers. One of the key disadvantages of this approach is that Core Nodes have to be re-programmed whenever a new PHB (a new policy) is introduced, since Edge Nodes only assign packets to traffic classes and mark them accordingly, but the PHB logic is implemented by the Core Nodes.

Another such proposal is Core Stateless Fair Queuing (CSFQ) [15] that implements a single policy: proportional

fair bandwidth sharing and marks packets of each flow with its estimated rate at the edge. In contrast to DiffServ, the logic of the policy to be applied is implemented by Edge Nodes while Core Nodes solely use the packet markings during dropping and scheduling, resulting in a highly scalable bandwidth allocation approach. FairCloud [16] proposes the use of CSFQ for network isolation in data center networks but it does not deal with policy hierarchies. CSFQ has recently been extended to support two-level HQoS [1], however this extension requires flow states (of the higher hierarchy level) in the Core, thus it is less scalable both in depth of the HQoS and in the number of flows than the approach proposed in this paper.

Rainbow Fair Queueing (RFQ) [17] assigns a few drop precedence levels called colors to the packets while Core Nodes drop packets according to the assigned precedence level.

This concept is extended by the Per Packet Value (PPV) method [10] that uses scalar values as packet markings instead of a few colors. PPV method uses a Throughput-Value Function to express operator policies and solves the resource sharing problem by maximizing the total transmitted packet value. It aims at providing a practical, distributed approximate solution for the network utility maximization problem [18]. Different AQM algorithms [19], [20], [3] have also been proposed for the PPV concept.

The PPV framework is extended with the support of resource sharing policies across multiple layers of virtualization in [21] by introducing the HPPV remarker. Even though HPPV provides a general HQoS solution, the remarking algorithm is unnecessarily complex, especially if multiple hierarchy levels can be marked in a single point.

A recent core-stateless resource sharing proposal is ABC [22] that measures the activity level of flows and encodes activity information into packets that is solely used at forwarding nodes to enforce fair-bandwidth sharing among users by dropping packets with high activity values more likely. This solution is similar to the PPV method, but is less flexible in defining operator policies.

Core-stateless networking solutions used to be a widely examined research area in the early 2000s, however only a very few proposals [10], [19], [20], [3], [22], [1] have been published in the past few years. With the emerging trend of softwarization in computer networks and with the advent of programmable data planes, their applicability has become possible even in production environments. A very limited prototype of a HQoS capable PPV packet marker and of a P4 PPV scheduler was demonstrated in [23]. This paper extends the packet marker for general HQoS hierarchies and optimizes the P4 scheduler for higher bottleneck capacity.

Though the above resource sharing methods are closest to our work, there are fundamentally different approaches like ELMO [24], RDNA [25] and PolKa [26] handling QoS as a routing problem. These source routing-based solutions also share the core-stateless idea: packets are tagged at the source and tags are used in routing decisions of network nodes, enabling end-hosts/applications to select the route satisfying their requirements. They mostly focus on ensuring different end-to-end delay requirements and do not directly deal with congestion along the used paths. The method proposed in this paper solves a different problem but could be combined with these source routing approaches. Link-wise resource sharing and route selection are both important to satisfy complex QoS requirements in an arbitrary network.

III. System Model

In this section, we briefly overview our core-stateless resource sharing framework called Per Packet Value (PPV) and introduce the key definitions needed for the understanding of its HQoS extension. Similarly to other core-stateless proposals [14], [22], [17], [15], [1], the PPV framework consists of two key elements: 1) a packet marker which assigns values to each packet of a traffic aggregate (e.g., the total traffic of a subscriber in an access aggregation network or a tenant in a data center) according to a predefined resource sharing policy; and 2) a scheduler that solely uses the values carried by packets to make a decision on which packet to drop or mark with ECN CE flag if congestion is experienced (e.g., the buffer is full or its size exceeds a predefined threshold, etc.).

Accordingly, each traffic aggregate (TA) has its own marker instance that labels each packet entering the PPV networking domain with a drop precedence called Packet Value (PV). Though policy-based marking maintains states for TAs, each marker instance operates independently. Thus, packet markers can be implemented and deployed in a distributed way (e.g., running in a Kubernetes Cluster, Telecom/Edge Cloud).

The scheduler is implemented by all the nodes (e.g., routers, end-hosts) in the PPV domain since each of them could be a potential bottleneck. When the buffer is congested, these nodes drop (or mark with ECN CE) one or more packets – either from the buffer or the newly arrived packet – with the smallest PV(s), instead of tail dropping. To support the coexistence of flows with L4S and Classic congestion controls, a modified scheduler called VDQ-CSAQM has been proposed in [3]. It uses separate queues for the two congestion control classes to satisfy the different delay requirements but at the same time applies a coupled dropping strategy to ensure the desired resource share even between flows with incompatible congestion control.

A. Policy Encoding

As mentioned, the core essence of PPV is how packets of a traffic aggregate (TA) are tagged with PVs. To this end,

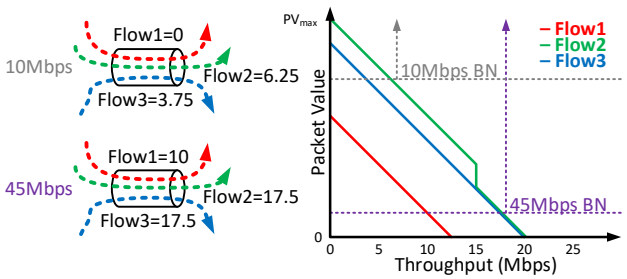


Fig. 1. Resource sharing with the PPV framework

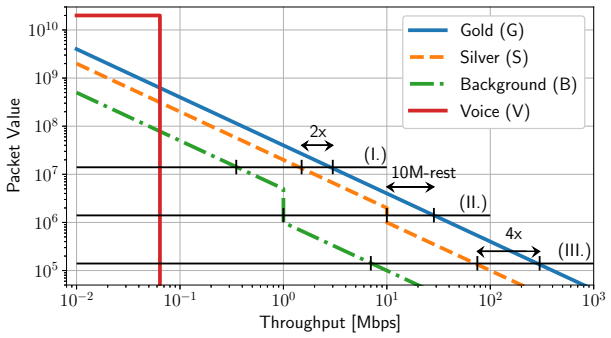


Fig. 2. Policy examples as Throughput-Value Functions

we use a Throughput-Value Function (TVF) that is defined as the derivative of a utility function: $V_a(x) = U'_a(x)$. Note that for each TA a the utility function $U_a(x)$ expresses the application gain (or the value realized by the network operator) if throughput x is assigned to it from the shared network-resources. The derivative of the utility function represents the extra value (e.g., the increase in profit for the operator) that can be generated if extra throughput is given to TA a. In this paper, we assume that TVFs are strictly monotone decreasing in the possible rate range of the given TA.

In the PPV method, TVFs are used to label packets where the packet value expresses the gain that is realized when the packet is delivered (marginal utility in other words). The TVF $V(.)$ defines the PV distribution of a TA for any sending rate R. Specifically, the throughput contribution of packets with PV at least $V(x)$ in the TA is x (for any $x : 0 \leq x \leq R$). A practical packet marker [27] of TA a continuously measures the aggregate's sending rate R_a, chooses x from range $[0, R_a]$ uniformly at random at packet arrival and assigns PV $p = V_a(x)$ to the given packet.

Fig. 1 illustrates how TVFs and PVs can be used to share the bottleneck capacity between various traffic aggregates. In the first case, the bottleneck capacity is 10Mbps shared between three flows. The red, blue and green curves on the right side represent the TVFs of Flow1, Flow2 and Flow3, resp. The gray dotted line illustrates the cutoff value that results in a resource allocation 0, 6.25 and 3.75Mbps for Flow1, Flow2 and Flow3, resp. This allocation is ensured by only transmitting packets with PV above the cutoff level. One can observe that Flow1 has no packet with PV above this threshold and thus it cannot even transmit a single packet. In the case of a 45Mbps bottleneck, the cutoff value is much smaller and thus all three flows have non-zero assigned throughput. The purple dotted line represents the cutoff value, leading to 10, 17.5 and 17.5Mbps throughput allocation for Flow1, Flow2 and Flow3, resp. In this case, only packets below the threshold, marked by the purple line, are dropped (or marked with ECN CE). One can observe that the inverse function of a TVF is basically a bandwidth function introduced in [9], thus bandwidth function policies can easily be mapped to TVFs in the PPV system model, and vice versa.

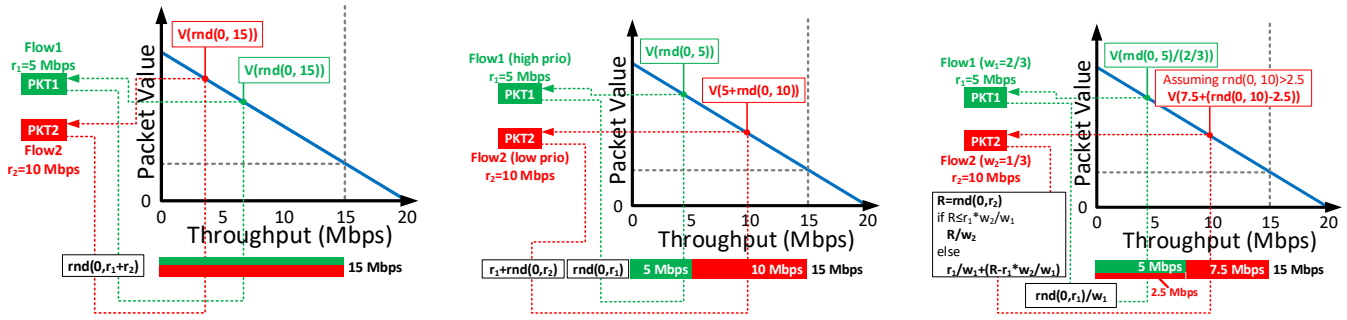
In general, at high congestion only packets with high PVs are transmitted, more precisely packets with PV above a certain cutoff value that we call Congestion Threshold Value (CTV). The CTV at a bottleneck represents the minimal PV that can successfully be transmitted via the congested link. The observed CTV reflects the actual congestion level, since at a congested link the total throughput of packets having PV at least the current CTV is exactly the bottleneck capacity. CTV is not a parameter of the proposed system, but an emergent property of the applied drop minimum-PV first AQM strategy. The amount of high and low PV packets determines the resource share between various flows, resulting in that at high congestion, aggregates with larger share of high PV packets get more throughput.

Fig. 2 depicts few example policies expressed as TVFs. Voice traffic is rate limited and has a guaranteed throughput need. It is defined by the red curve consisting of two segments: up to the rate limit PVs are represented by a slowly decreasing line close to the maximum value representing strict priority for voice packets, and above that the smallest value (0) is assigned to the packets, emulating a rate limiter policy. The blue, orange and green TVFs express weighted fairness between the different flow groups, where the weights rely on the congestion level (CTV). For example, at high congestion (I.) a Gold flow can get twice the throughput of a Silver flow; at medium congestion (II.) Silver flows get 10Mbps throughput and the rest can be used by Gold flows; at low congestion (III.) 1:4 resource share is defined between Silver and Gold flows.

B. Hierarchical Quality of Service

The key building blocks of HQoS include strict priority and weighted-fair scheduling. In traditional design, strict priority scheduling is implemented between queues with different priorities, ensuring that a packet from a given queue can only be served if there is no packet in higher priority queues. Weighted-fair scheduling assumes queues with weights and packets are served by weighted deficit round robin scheduling. In both cases, packets that needs to be handled similarly (belonging to the same flow, flow-group, subscriber, slice, etc.) are forwarded to the same queue. These blocks can be combined and organized in a multi-level hierarchy to create complex resource sharing policies. The traditional HQoS implementation requires flow classification and thus the configuration and maintenance of per-flow states and a hierarchy of queues in all the interior nodes implementing the scheduler (see Fig. 4). In contrast, our proposal realizes HQoS policies in the packet marker by reorganizing packet values among the packets of flows within a traffic aggregate (TA) so that the packet value distribution of the entire TA remains the same. Packets are tagged once at the entry point of the PPV network domain. The PPV scheduler implemented by all the nodes in the network solely does not require any change. It solely uses the packet tags and still works without complex queue management in a flow- and hierarchy-unaware way.

As mentioned previously, each packet of a TA is marked with a packet value that is chosen randomly according to the distribution defined by the total sending rate and the applied TVF. It has already been shown that this distribution ensures the desired resource sharing among various TAs. For TA a



(a) Without flow differentiation. (b) Emulated strict priority scheduling. (c) Emulated weighted-fair scheduling.

Fig. 3. Modeling HQoS policies by encoding them into packet values. We assume a single traffic aggregate with two flows a) without flow differentiation, b) with strict priority ordering and c) with weighted-fair scheduling.

with sending rate R_a and TVF $V_a(.)$, the packet value distribution generated by the marker algorithm in steady state can be described as $P[v > \delta] = V_a^{-1}(\delta)/R_a$ $(\delta \in [V_a(R_a), V_a(0)])$. This is a consequence of the PPV marking algorithm as if $x \in [0, R_a]$ is a uniform random variable, the corresponding packet value is described by a random variable $v = V_a(x)$.

Fig. 3 depicts more complex scenarios with two flows within the same TA a. The sending rate of the flows are r_1 = 5Mbps and r_2 = 10Mbps, resp., while TA a applies TVF $V_a(.)$ as its top-level resource sharing policy. Fig. 3a shows how packets of the two flows are marked according to the original PPV framework, handling them equally within the same aggregate. In this case, each flow in the TA has the same packet value distribution in steady state that is only affected by the actual throughput of the TA and the applied TVF. Since the marker does not differentiate between the flows, it results in the same conditional probability distribution for both flows $P[v > \delta|\text{Flow}_i] = V_a^{-1}(\delta)/R_a$ where $P[\text{Flow}_i] = r_i/R_a$. This is caused by the PV marking algorithm which chooses rate samples from $[0, R_a]$ according to a uniform distribution independently of which flow the packet belongs to. As a result, the rate samples x of each Flow_i $(i = 1, 2)$ follow the same conditional distribution $P[x < \delta \mid \text{Flow}_i] = \delta/R_a$ for any $\delta \in [0, R_a]$. Then the applied TVF transforms the rate sample distribution to a PV distribution. In this case, if the TA's throughput is cut at, e.g., 7.5Mbps, 50% of the packets of both flows are dropped since 50% of them have packet values less than V_a(7.5Mbps), resulting in the same drop ratio for the two flows.

The strict priority scheduling can be emulated in the packet marker by assigning packet values in order to the flow priorities as illustrated in Fig. 3b: the packets of Flow_1 always have higher priority than the ones of Flow_2. In this case, we split the packet value distribution of the TA a into two parts: Flow_1 gets packet values from range $[V_a(r_1), V_a(0)]$ while PVs in Flow_2 are chosen from the remaining range $[V_a(R_a), V_a(r_1)]$ (note: $R_a = r_1 + r_2$ in the example). This value assignment ensures that packets from Flow_1 can only be dropped if all the packets from Flow_2 were dropped previously. The figure depicts how this can be implemented in the PV marking: Flow_1 chooses $x \in [0, r_1]$ uniformly at random while Flow_2 from another range $[r_1, R_a]$ to calculate the packet value $v = V_a(x)$. The resulting packet value distribution of the TA remains the same since the joint rate distribution of the resulted per-flow conditional distributions remains uniform for any $\delta \in [0, R_a]$ (note: $P[\text{Flow}_i] = r_i/R_a$):

$$P[x < \delta \mid \text{Flow}_1] = \begin{cases} \frac{\delta}{r_1}, & \text{if } \delta \le r_1 \\ 1, & \text{otherwise} \end{cases}$$

$$P[x < \delta \mid \text{Flow}_2] = \begin{cases} 0, & \text{if } \delta \le r_1 \\ \frac{\delta - r_1}{r_2}, & \text{otherwise} \end{cases}$$

Encoding weighted fair resource allocation into packet values is more tricky. Fig. 3c depicts such a scenario for two flows. The weights of Flow_1 and Flow_2 are set according to a desired 2 : 1 share ($w_1 = 2/3$ and $w_2 = 1/3$). First, we compute the minimal throughput need ensuring that the flow's demand can fully be served according to its weight. For example, Flow_1 has a sending rate 5Mbps and a normalized weight 2/3. Dividing the sending rate with the flow's weight results in a projected capacity $C_1 = r_1/w_1 = 7.5Mbps$. If the outgoing link's capacity is at least C_1, the demand of Flow_1 ($w_1 C_1 = r_1$) could fully be satisfied. Note that if the bottleneck capacity is greater than C_1, the flow will use less resource from the bottleneck than its weighted share. Consequently, we first take the flow with the smallest projected capacity – Flow_1 in our case – since C_1 = 7.5Mbps and C_2 = 30Mbps. C_1 splits the rate range into two parts: range 0 to 7.5Mbps is shared among both flows, while the remaining range (7.5-15Mbps) is solely used by Flow_2. Similarly to the previous case, the packet value distribution is generated by applying different marking for the two flows: Flow_1 and Flow_2 choose $x \in [0, R_a]$ according to the following conditional probabilities and calculate the packet value $v = V_a(x)$ as previously ($\delta \in [0, R_a]$):

$$P[x < \delta \mid \text{Flow}_1] = \begin{cases} \frac{\delta}{C_1}, & \text{if } \delta \le C_1 \\ 1, & \text{otherwise} \end{cases}$$

$$P[x < \delta \mid \text{Flow}_2] = \begin{cases} \frac{\delta}{C_2}, & \text{if } \delta \le C_1 \\ \frac{\delta - r_1}{r_2}, & \text{otherwise} \end{cases}$$

Note that the joint distribution for the TA is uniform in the range $[0, R_a]$ and the resulting PV distribution is identical with the expected PV distribution of TA as shown in the first case. Since the expected PV distribution is derived from a uniform random rate distribution by transforming rate values

into PVs via the TVF, the differentiation between the flows can be implemented by the throughput transformations depicted in Fig. 3c. As mentioned previously, the first range is shared between the two flows according to their weights; in this example w_1C_1 from Flow_1's total throughput r_1 and w_2C_1 from Flow_2's throughput r_2 are transformed into the range $[0, C_1]$, covering it uniformly at random. Since $w_1C_1 = r_1$, Flow_1's demand is fully served in the first region, and $w_2C_1 \leq r_2$ from the total demand of Flow_2. The second region $[C_1, R_a]$ is solely used by Flow_2, namely the remaining $r_2 - w_2C_1$ from its total throughput need is mapped into this range uniformly at random.

IV. Hierarchical Marker Design

In the previous section, we have shown two basic examples on how strict priority (SP) and weighted fair (WF) resource sharing can be implemented in packet value marking by splitting the expected PV distribution into per-flow conditional components. A complex HQoS policy can be described as a multi-level tree built from SP and WF nodes as depicted in Fig. 4a. In this example, the TA consists of six flows: on the first level, Flow 2 and 3 share the resources according to a WF policy (denoted by $\text{WF}^{(1)}$), while there is also weighted fair scheduling between Flow 4, 5 and 6 (see $\text{WF}^{(2)}$). On the second level, we apply SP scheduling between Flow 1 and the flow mix of $\text{WF}^{(1)}$ (aggregated traffic of Flow 2 and 3). Finally, on the third level another WF scheduling ($\text{WF}^{(4)}$) is implemented between the flow mixes of $\text{SP}^{(3)}$ and $\text{WF}^{(2)}$.

As shown in the previous section, we assume strictly monotone decreasing TVFs and thus both SP and WF scheduling can be implemented by remapping input rate distributions to a uniform rate distribution over the aggregated throughput range of the inputs. It means that these policy components apply various rate transformations on per-flow rate samples to express the required resource sharing behavior between the flows.

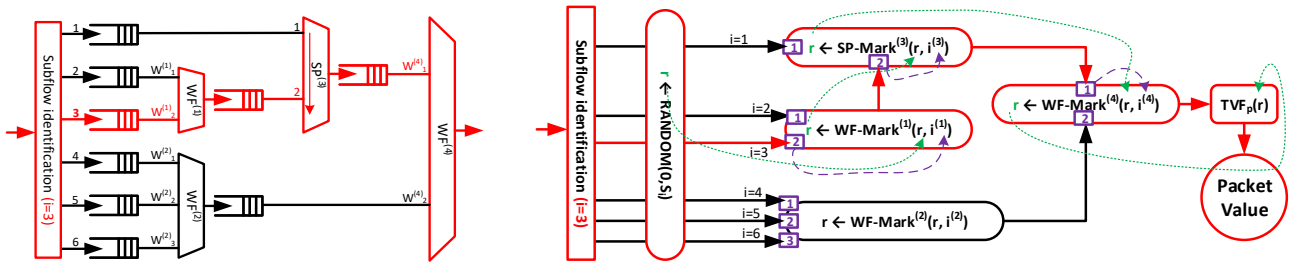
At the top level of the hierarchy, the applied TVF itself adds an additional hierarchy layer as described in Sec. III-A. This enables the distribution of packet marking. For example, the packet markers of each subscriber in Sec. VII operate independently from each other, including the DeepQoS marker of the selected household. Additional distribution of marking can be achieved by combining DeepQoS with the PV-remarking concept of HPPV [21]. HPPV enables to add a new hierarchy layer to an already marked packet TA. The HPPV marker does not need any other information than the incoming Packet Values and the new TVF to be applied.

A. SP Marking

The strict priority marking concept mentioned in the previous section can easily be extended for arbitrary number of flows. Assuming n flows $Flow_1$, ..., $Flow_n$ with flow rates r_1, ..., r_n and priorities $p_1 > p_2 > \cdots > p_n$, for the incoming packet stream SP marker generates a sequence of random samples from $[0, R]$ chosen uniformly at random, where $R = \sum_{i=1}^{n} r_i$. SP marker splits the throughput range into n disjoint sub-ranges and each flow chooses a sample from its sub-range uniformly at random. For the ith flow, x is chosen from $[\sum_{k=1}^{i-1} r_k, \sum_{k=1}^{i} r_k]$, ensuring that the samples of a flow will always be smaller than the samples of other flows with lower priorities.

B. WF Marking

Fig. 5 depicts a more complex example on how throughput samples of flows in a given traffic aggregate are remapped according to weighted fair resource sharing. The TA consists of three flows, i.e., Flow_1 (orange), Flow_2 (red) and Flow_3 (yellow) with arrival rates 6, 2, and 4 Mbps, resp. The weights of the three flows are 2, 1, 1, i.e., Flow_1 shall have twice as many resources as the others while Flow_2 and Flow_3 share the resources equally. Our goal is to determine the conditional rate distribution for each flows so that they jointly result in a uniform rate distribution in the range of 0 to 12 = 6+2+4 Mbps on which the operator policy (TVF) is applied. We first rank the flows according to their projected capacity. This can be calculated by dividing each flow rate by its normalized weight. According to the ranking, the second flow (red) is the first because it uses less than its weighted share from the bottleneck capacity. Then comes the first flow (orange), getting exactly as much as it deserves and finally there is the third flow (yellow) at the end that uses more than its weighted share. As seen in Sec. III, Flow_2 having the smallest projected capacity determines the first throughput range that is between 0 and 8 Mbps that is shared among all the three flows. Their contributions can be calculated by multiplying the normalized flow weights with the length of the range. Accordingly, Flow_1, Flow_2 and Flow_3 contribute $4 = 8 * 0.5$, $2 = 8 * 0.25$ and $2 = 8 * 0.25$, resp. One can observe that the demand of Flow_2 (red) is fully covered by the first range. In the second range, we continue with the other two flows whose remaining throughput needs are $6 - 4 = 2$ Mbps and $4 - 2 = 2$ Mbps (for Flow_1 and Flow_3, resp.). Since the second range is only shared by two flows with weights 2 and 1, the normalized weights need to be recalculated (2/3 and 1/3) and applied. The second flow according to the ordering is Flow_1 where the remaining throughput is 2 Mbps and its recomputed projected capacity is $2/(2/3) = 3$ Mbps that is the length of the second throughput range between 8 and 11 Mbps. The contributions of Flow_1 and Flow_3 in this range are $3 \times (2/3) = 2$ Mbps and $3 \times (1/3) = 1$ Mbps, resp. One can note that the throughput needs of Flow_1 are fully covered by the first two ranges. Thus, the third range is solely used by Flow_3. Its remaining throughput need is $4 - 2 - 1 = 1$ Mbps that is represented by the last range between 11 and 12 Mbps. One can observe in Fig. 5-(4.) that for each flow its contributions in various ranges determine the conditional probability distribution to be used for choosing rate samples for packet marking. For example, Flow_1 (orange) assigns 4 Mbps to the first range, 2 Mbps to the second, and nothing to the third from its total throughput need of 6 Mbps. Accordingly, $4/6 = 2/3$ of the packets gets a throughput sample from the first range while $2/6 = 1/3$ of them from the second range. The samples within the ranges are chosen uniformly at random.



(a) Traditional HQoS. (b) HQoS Packet Marker.

Fig. 4. In Traditional HQoS, packets are directed through a hierarchy of physical buffers. In HQoS marker, HQoS policies are encoded as a sequence of rate transformations. The marker translates the HQoS policy into a single packet value

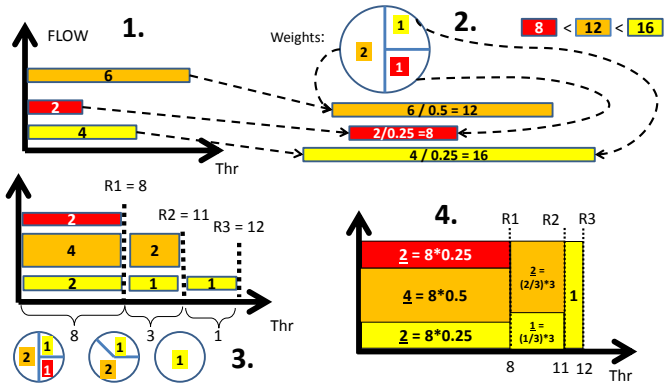


Fig. 5. WF marking example with three flows

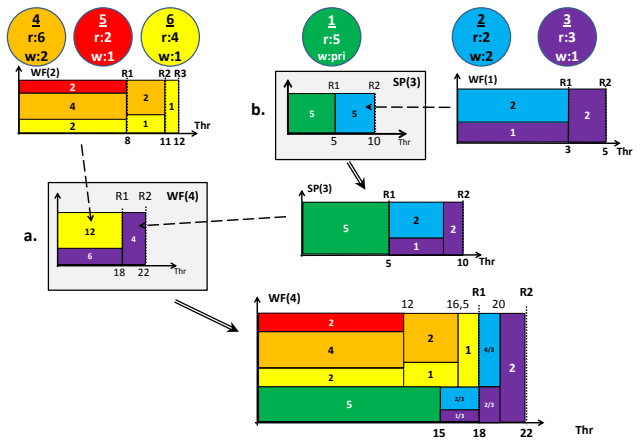


Fig. 6. HQoS graph example with six flows

C. HQoS graph

Fig. 6 depicts an example on how the complex HQoS policy shown in Fig. 4a is implemented by the PV marking algorithm. There are six flows in the given TA: the previously presented three flows (4-orange, 5-red and 6-yellow) with WF scheduling are depicted in the upper left corner (WF(2) as in Fig. 5) and there are three more flows (1-green, 2-blue and 3-purple) with sending rates of 5, 2, and 3 Mbps, resp. Flow 2 and 3 (blue and purple) are connected to a WF component with weights 2 and 1, resp. The calculated conditional distributions $P^{WF(1)}[x < \delta \mid \text{Flow}_i]$ $(i = 1, 2)$ are presented in the right upper corner (WF(1)). Note that the expected outgoing rate distribution of WF(1) is a uniform distribution in the range of 0 to 5Mbps. The output of this WF component is connected to an SP component. SP(3) defines a strict priority scheduling between the green flow and the aggregated traffic of blue and purple flows. The green flow with sending rate of 5Mbps has higher priority than the flow aggregate (blue and purple). The aggregated throughput of the blue and purple flows is also 5 Mbps. One can also observe in the right gray box (b.) that the range between 0 and 5Mbps is solely used by the green flow while the remaining range is shared between the blue and purple according to the per-flow distributions of WF(1). SP(3) also defines different conditional rate distributions for the two inputs according to the strict priority rule. However, its second input is a flow aggregate (WF(1)) of Flow 2 and 3 (blue and purple), and thus its conditional distribution $P^{SP(1)}[x < \delta \mid \forall\text{Flow} \in \text{WF}(1)]$ $(\delta \in [5, 10\text{Mbps}])$ can also be described as the joint distribution of conditional distributions of Flow 2 and 3 from the previous layer. Then WF(2) and SP(3) are used as inputs in WF(4) with weights 2 and 1, resp. This situation is the same as having 2 input flows with sending rates of 12 Mbps (red-orange-yellow) and 10 Mbps (blue-purple-green) as depicted in the left gray box (a.). Replacing these two areas with the previously computed distributions, we get the final distribution (WF(4)) on the bottom where each flow is represented. The subfigure shows the contribution of each flow in the different disjoint throughput ranges that are taken into account during the PV marking.

V. DeepQoS Marker Implementation

To express complex HQoS policies, DeepQoS Marker is built from WF and SP components connected as a directed acyclic graph as shown in Fig. 4. As shown previously, such a HQoS policy can be implemented by applying different per-flow conditional distributions during the PV marking. This section focuses on practical algorithms that satisfy the previously defined requirements on the resulted rate and PV distributions.

First, we define the building components (WF and SP) of the proposed DeepQoS method. Each of them is responsible for modeling the scheduling among multiple input flows (simply called as inputs). When a packet of input flow l arrives, we choose a rate sample r_{in} from 0 to the current arrival rate of input l uniformly at random. Then a DeepQoS component transforms r_{in} into the aggregated throughput range (sum of the input arrival rates) so that the transformed rate values r_{out}

follow the conditional distribution $P[r_{out} < \delta \mid \text{Flow}_l]$ while the joint output distribution for all inputs of the component results in a uniform distribution in the range from 0 to the sum of input arrival rates. This requirement is needed for that the PV distribution of the TA remains the same and thus the resource sharing properties of the original PPV concept still hold. The theoretical analysis of DeepQoS can be found in Appendix A and B.

A. WF Component

WF component implements a weighted fair resource sharing policy between a number of flows. It contains two key mechanisms: 1) An updating algorithm that periodically refreshes the per-flow rate measurements and the related variables (e.g., projected capacities, flow ranks and order) needed for the marking phase; and 2) A WF marking algorithm that transforms a rate sample $r_{in} \in [0, S_l]$ of input l into a rate value in the entire throughput range of the aggregate $[0, \sum_{k=0}^{n} S_k]$ so that it generates the desired PV distribution presented in the previous section.

WF-Updating algorithm: Algorithm 1 receives new per-flow rate measurements and updates the internal variables accordingly. It first determines a new indexing o_i such that the re-indexed input flows have increasing projected capacities (line 2-3). Then as it is shown in Fig. 5 the throughput range is split into several regions and the number of subflows sharing the same range is decreasing as moving away from the origin. The normalized weights $W_{j,i}$ in each region j are calculated in lines 7-9 by normalizing the weights for the flows o_i $(j \leq i)$ present in that region. Then the length of throughput ranges (δ_j) and the region boundaries for the aggregate (R_j) are calculated. For each flow o_i, we also split the flow's throughput range into smaller regions $R_{j,i}$ $(j < i)$ where the length of $R_{j,i}$ represents the throughput contribution of flow o_i in the aggregated range R_j (see line 13-14). Note that according to the above definition the last range ends at $R_n = \sum_{l=1}^{n} S_l$. Also the number of flows in each TVF region is decreasing, because, e.g., flow o_1 uses the least of its allocated fair share therefore its throughput need fully fits into the first range. In general, flow o_j can completely be represented in the first j regions.

Note that the updating algorithm needs to be executed periodically to get accurate rate sample distributions. Between two updates the method assumes a steady state. In our performance evaluation, we set the update period to 5 ms. The calculations can be done on a separate thread (e.g., in DPDK) or in the control plane, and thus it does not delay the packet processing on the fast path.

WF-Marking algorithm: Upon the arrival of a packet from input flow l, the marking algorithm first determines a rate sample r_{in} chosen uniformly at random from range $[0, S_l]$. Algorithm 2 then remaps the flow index into the ordered index i so that $o_i = l$ (line 2). For flow o_i, the $R_{j,i}$ $(0 < j \leq i)$ values split the throughput range $[0, S_{o_i}]$ into i ranges whose length defines the throughput contribution of flow o_i in the aggregated throughput range R_j. Note that $R_{i,i} = S_{o_i}$ by definition. Based on the relation of r_{in} and the $R_{j,i}$ values the relevant rate region (j) is determined (line 3). Transformed rate sample r_{out} is determined in line 4, by applying the normalized weight $W_{j,i}$ to the portion of the r_{in} represented in rate region j, i.e., the throughput contribution in range j which is $r_{in} - R_{j-1,i}$.

Algorithm 1: WF-Updating

Input : Rate measurements for each input: S_1, ..., S_n

1 **begin**
2 Calculate ordered-indexing: o_1, ..., o_n;
3 *Note:* $S_{o_i}/w_{o_i} \leq S_{o_j}/w_{o_j}$, $(i < j) \forall i, j \in [0, n]$;
4 $R_0 \leftarrow 0$;
5 **for** $i \leftarrow 1$ **to** n **do**
6 $R_{0,i} \leftarrow 0$;
7 **for** $j \leftarrow 1$ **to** n **do**
8 **for** $i \leftarrow j$ **to** n **do**
9 $W_{j,i} \leftarrow w_{o_i} / \sum_{k=j}^{n} w_{o_k}$;
10 **for** $j \leftarrow 1$ **to** n **do**
11 $\delta_j \leftarrow (S_{o_j} - R_{j-1,j}) / W_{j,j}$;
12 $R_j \leftarrow R_{j-1} + \delta_j$;
13 **for** $i \leftarrow j$ **to** n **do**
14 $R_{j,i} \leftarrow R_{j-1,i} + \delta_j \times W_{j,i}$;

Algorithm 2: WF-Marking

Input : Input index $l \in [1, n]$
Rate sample parameter $r_{in} \in [0, S_l]$
Output: Transformed rate $r_{out} \in [0, \sum_{k=1}^{n} S_k]$

1 **begin**
2 Find $i \in [1, n]$ so that $o_i = l$;
3 Find $j \in [1, i]$ so that $R_{j-1,i} < r_{in} \leq R_{j,i}$;
4 $r_{out} \leftarrow R_{j-1} + (r_{in} - R_{j-1,i}) / W_{j,i}$;
5 Return r_{out};

B. SP Component

Strict priority resource sharing policy between a number of flows can be implemented similarly to a WF Component. It also consists of an updating and a marking algorithm.

SP-Updating algorithm: Algorithm 3 is used to compute rate boundaries in the aggregate throughput range according to the flow priorities and the new rate measurements. As shown in Sec. IV-A, rate values of flow i are remapped to the range of $[L_{i-1}, L_i]$ where the interval length is exactly the flow rate S_i. L_i is used as an offset during the marking phase to accelerate the computations. These ranges are organized in increasing order of flow priorities.

SP-Marking algorithm: Algorithm 4 describes the marking algorithm modeling a strict priority policy. Upon arrival of a packet from flow l, it simply offsets input rate sample $r_{in} \in [0, S_l]$ into the $[L_{i-1}, L_i]$ range (line 2).

C. DeepQoS Marking Graph

The previously introduced WF and SP marker components can be organized into a hierarchy. They hold the property

Algorithm 3: SP-Updating

Input : Rate measurements for each input: S_1, ..., S_n

```
1 begin
2     Note: indexes reflect the priority order;
3     L_0 ← 0;
4     for i ← 1 to n − 1 do
5         L_i ← L_{i−1} + S_i;
```

Algorithm 4: SP-Marking

Input : Input index $i \in [1, n]$ (priority order)
Rate sample parameter $r_{in} \in [0, S_i]$
Output: Transformed rate $r_{out} \in [0, \sum_{k=1}^{n} S_k]$

```
1 begin
2     r_out ← L_{i−1} + r_in;
3     Return r_out;
```

that for a traffic mix with n inputs where $P[\text{Flow}_i]$ is the probability that the packet to be marked belongs to input i and the input rate sample r_{in} for each input i is chosen uniformly at random from $[0, S_i]$, the output rates of both WF or SP components follow a uniform distribution over the aggregated throughput range $[0, \sum_{k=1}^{n} S_k]$. Note that an input can either be a single flow or a flow aggregate (traffic of multiple flows in the TA) joined according to a previous WF or SP component in the hierarchy.

In our model, HQoS policies can be described as a marker-graph. Let the marker-graph be a directed acyclic graph $G = (V, E)$ where V consists of marker (rate transformation) nodes (WF or SP), a designated starting and ending nodes.

Fig. 4 depicts an example for a traditional HQoS scheduler and its analogous HQoS marker graph. When comparing the two, it is visible that the connections in the traditional HQoS scheduler are represented by the connections in the marker graph. The red components and arrows in both figures show the path of a packet belonging to Flow 3 as an example. DeepQoS packet marking starts with identifying the flow (i) the packet belongs to. Then a rate sample $r \in [0, S_i]$ is chosen uniformly at random. This rate sample is transformed (one or more times) by routing through a sequence of WF and SP marker nodes in the graph until the TVF node is reached and the packet value is determined. Note that the top level policy in the hierarchy is expressed by the TVF and ensures the resource sharing between TAs. Green arrows show how r is repeatedly transformed for Flow 3, and purple arrows illustrate how the local index $i^{(x)}$ is determined according to the flow mix of the local inputs.

Algorithm 5 formalizes this description. $m, i^{(m)}$ indicate the mth node of the graph, reached in the $i^{(m)}$th input port. *adj* function determines the next node in the graph (i.e., where the outgoing arrow points to). In line 4, the algorithm chooses the ith outgoing edge from the starter node, while in line 7 the adjacent node of each marker node is unique. *apply* represents the application of the given WF or SP component's marking algorithm. Finally, $V(.)$ is a TVF implementing the top-level resource sharing policy between the TAs.

Algorithm 5: HQoS packet marking algorithm.

Input : Flow index i in the TA
Marker-graph $G(V, E)$
Operator policy of the TA as TVF $V(.)$
Output: Packet value v

```
1 begin
2     Let s ∈ V be the starting node;
3     r ← random(0, S_i);
4     m, i^(m) ← adj(s, i);
5     while m is not the ending node do
6         r ← apply(m, i^(m), r);
7         m, i^(m) ← adj(m);
8     v = V(r);
9     Return v;
```

VI. Scheduler - P4 Implementation of VDQ-CSAQM

DeepQoS marking could be used with any PPV-aware scheduler since the PV distribution of each TA remains the same as with the original PPV marking algorithm. The applied HQoS hierarchy is fully represented by the DeepQoS marking. The scheduler itself is fully unaware of the used HQoS policies, the number of hierarchy levels, the number of flows and the traffic aggregates. To demonstrate the simplicity and versatility of the system we adapted the L4S-capable VDQ-CSAQM [3] scheduling algorithm to a Tofino-based P4 programmable switch. As an orthogonal requirement to controlling resource sharing, this scheduler can differentiate between L4S and Classic Internet services, providing ultra-low queueing delay for L4S and higher delay for Classic traffic classes in the same system. Note that the VDQ-CSAQM algorithm was originally implemented in DPDK and performed computations on the fast path that proved to be too complex for today's programmable switches. To find a good trade-off between performance and small data plane complexity, we redesigned the VDQ-CSAQM algorithm by distributing computations between data and control planes as depicted in Fig. 7.

A. Data Plane (DP) Pipeline

The main goal of our architecture design is to simplify the per-packet operations in the P4 pipeline as much as possible. As in the original design of VDQ-CSAQM [3], for each egress port two physical queues are configured with strict priority scheduling between them. The L4S queue (Queue 0) has higher priority than the Classic one (Queue 1). Non-ECT packets with packet value less than CTV_i (Congestion Threshold Value) are dropped in the ingress pipeline before queueing, while ECN CE marking happens at the egress side, based on the same CTV_i values. We have replaced Virtual Queues (VQs) of the original algorithm with two groups

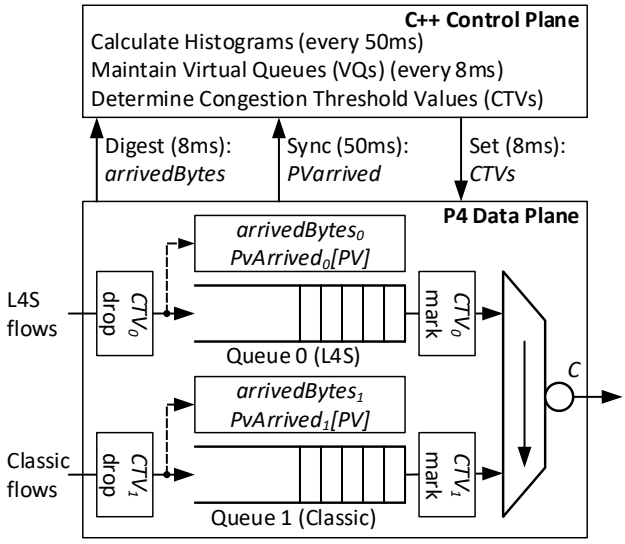


Fig. 7. Data and control plane design of VDQ-CSAQM

of counters (counter overflow is allowed): $arrivedBytes_i$ (4 bytes) counts the amount of bytes arrived at Queue i, while $PvArrived_i[PV]$ (4 bytes/PV) counts the bytes carried in the packets with a given PV that arrived at Queue i. All counters are defined for each queue of each PPV-enabled egress port, data aggregation happens in the CP. The PV is encoded into 1024 different values (i.e., 10 bits) logarithmically which corresponds to a resource sharing precision of 3% in the range of 10 Kbps to 1 Tbps. Each queue shares the same buffer area of the switch. Note that today's programmable switches have been designed for use cases where deep buffers are not required, posing additional challenges to keep the control at AQM algorithms instead of experiencing random packet drops. At the typical capacity of 10 Gbps we use in the evaluation, this means 17.6 ms buffer space in total shared among all the egress ports. Note that programmable switches for other business cases (e.g., AANs) requiring deeper buffers could potentially be developed in the future.

Though all the necessary steps of the algorithm could be implemented in the ingress pipeline, for practical considerations we have split it into ingress and egress parts. With this design the pipeline requires only 3 stages at both ingress and egress parts, allowing the coexistence of the scheduler with other functions like switching, routing or even complex 5G User Plane Function. At ingress side, the CTV-based packet dropping is implemented by a match-action table with entries for each PPV-enabled egress port and traffic class (either L4S or Classic). This table is quite small, requiring only two entries per physical port. Then, the queue identifier is set according to the traffic class the packet belongs to. Finally, $arrivedBytes_i$s are also counted at the ingress block, implemented by registers since their values need to be sent to the control plane in digest messages. Packet marking with ECN-CE has been moved to the egress block, implemented by a similar lookup table as packet dropping at ingress. The packet value histograms $PvArrived_i$ are realized as counter arrays indexed by packet values (0-1023). Note that for each PPV-enabled port two such histograms need to be maintained which results in ≈ 8KBytes SRAM usage per port. In most traditional networking use cases the majority of packet processing logic is implemented at the ingress pipeline. The proposed pipeline also requires some computations at the ingress part but the majority of the logic is implemented at the egress, allowing the seamless co-existence of PPV-based scheduling with other packet processing pipelines (e.g., routing, 5G User Plane Functions, etc.). The CTV values in both ingress and egress parts are periodically updated by the control plane.

B. Control Plane (CP) process

To keep the data plane (DP) implementation lightweight, the complex computations of VDQ-CSAQM have been moved to a local control plane (CP) process running on the CPU of the switch. The CP written in C++ relies on the native Barefoot Runtime API that enables fast interaction with the data plane objects. It has three key roles: 1) maintaining VQ states, 2) updating PV histograms and 3) calculating CTV values to be applied in the DP.

DP sends the current values of registers $arrivedBytes_i$ as a digest message to the CP in every 8 ms. From the digest message and the values received in the previous iteration, the CP calculates the bytes received within the update interval. Then the state of each VQ is updated according to the calculated incoming bytes, its virtual service rate and the elapsed time since the last update. The virtual service rates of the VQs are set relative to the link capacity (C): $C_{v0} = 0.9C$ and $C_{v1} = 0.98C$. Coupling of the two VQs is implemented in the CP by considering the incoming bytes of both queues for Queue 1 (i.e., the low priority queue used for Classic traffic). Thereby we emulate coupling behavior of [3], where packets arriving to Queue 0 are also taken into account in VQ_1.

The CP syncs $PvArrived_i$ counters from the DP in every 50 ms that are used to calculate the Packet Value histograms for both queues. Since we use circular counters, the histogram can be computed by taking the difference of the new and previous values for each counter index. Similarly to VQs, the histogram of Classic traffic class is coupled, representing the PV distribution of both L4S and Classic packets. Note that though $arrivedBytes_i$ can be calculated from $PvArrived_i$, we maintain it separately to optimize the DP-CP communication. To get stable and more representative PV histograms, a longer measurement period is needed while faster control can only be ensured by updating VQ states (and thus CTVs) more frequently.

The desired packet marking/dropping probability (q_i) is then calculated from the VQ lengths and predefined target queue sizes. At this point, we deviates from the original design of [3], since programmable switches only have a shallow buffer area. It simply makes the large VQ target of 20 ms (Classic) impossible and the temporal packet bursts in transient states lead to unexpected packet losses (not the ones with low PVs) in the traffic management engine of the switch, taking control from VDQ-CSAQM algorithm in the background. We carried out number of experiments with the original algorithm and smaller targets, but it led to unstable behavior. Instead,

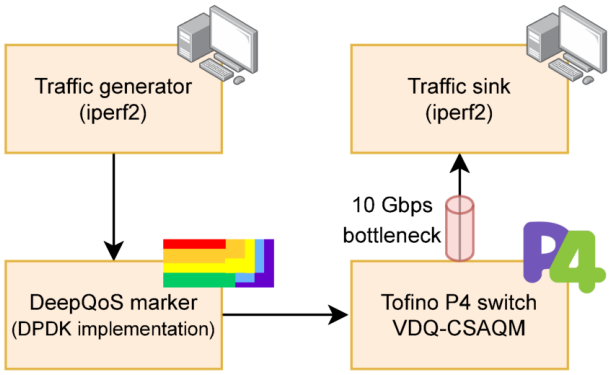


Fig. 8. Testbed

we extended the original algorithm with a RED-like [28] mechanism to calculate the dropping/marking probabilities q_i for each VQ, based on its instantaneous VQ length. We calculate $q_i = \max\left(0, \min\left(0.75, \frac{VQ_i - vqTh_i}{6 \cdot vqTh_i}\right)\right)$, where the VQ thresholds are set to $vqTh_0 = 0.5$ ms and $vqTh_1 = 1$ ms. The packet value thresholds CTV_i are then calculated to define the q_i percentile least important bytes in the PV histogram H_i: $CTV_i = \text{argmin}_v \sum_{j=1}^{v} H_i[j] \geq q_i \times \sum_{k=0}^{1023} H_i[k]$. Finally, the CP updates the match-action tables implementing CTV-based dropping and ECN-marking with the new CTV values.

a) Notes on CTV update times.: The Classic and L4S CTVs are used to generate congestion signals to the end-hosts. Congestion control is acknowledgment-clocked and thus the CTV needs to be controlled at the same timescale as the RTT. The CTV update is triggered by the DP. It encapsulates the VQ states into a digest message, and sends it to the CP in every 8ms. The CP recalculates the CTVs and updates the appropriate table entries in the DP. The time between generating the digest and finishing the table entry updates is approx. 2ms, giving enough time between two digest generation to see the effect of the modified settings. The PV histograms are collected in counter arrays that are synched in every 50ms. Reading and processing counter values take approx. 25ms in our CP implementation. We decided to keep it at 50ms to have stable packet value distributions needed for deriving the CTVs. Though these settings work well in the scenarios presented in Sec. VII, environments with small RTTs and much smaller target delays may require a faster control loop, resulting in additional challenges for future programmable data planes.

VII. Evaluation

Our performance evaluation to be presented in this section relies on the P4 implementation of VDQ-CSAQM run on a Delta AG9064v1 P4-programmable hardware switch and the DPDK-based implementation of the proposed DeepQoS marker deployed in a designated server machine. These nodes are connected in a chain topology between a traffic generator and a sink machine as shown in Fig. 8. The configuration of the three servers is identical (Xeon E5-1660 v4@3.2GHz CPU, with 64Gb RAM and Intel XL710 40GbE Ethernet NIC). The servers' operating system is Ubuntu Server 18.04 with the TCP Prague linux kernel patch (version 5.4.0-rc3) [29]. The accurate ECN [30] feature needed for the correct operation of TCP Prague has also been enabled. We have to note that the RTT Scaling mechanism of TCP Prague has been disabled during our measurements since it proved to be incompatible with shallow buffers of programmable switches, leading to significant underutilization of the bottleneck link.

The test traffic (both TCP and UDP) was generated by the iperf2 tool where the applied Congestion Control Algorithm (CCA) was varied, using TCP Cubic as classic and TCP Prague as scalable CCA. Different propagation RTTs were emulated by delaying the TCP acknowledgments at the sink side, using the tc netem tool. The bottleneck between the switch and sink nodes was created by rate limiting the appropriate egress port of the P4 switch. In most scenarios, the bottleneck capacity was set to 10 Gbps. In all the evaluation measurements, the same VDQ-CSAQM and marker parameters were used. The averaging timescale in the rate measurements of the packet marker was set to 40ms (as described in [27], Sec. IV-A) and the internal states of DeepQoS marking components were updated in every 5ms.

Table I shows the labels of the six flow types (flows with different CCA and RTT parameters) we use in the evaluation. In most evaluation scenarios two flow types are considered in the same system while the total number of flows from each type are varied from 2 to 200. Either Gold or Silver research sharing policies are applied for the flows with different types. We use the TVFs as defined in Fig. 2. To show the resource sharing accuracy of DeepQoS marking we often select a designated TA called household ("HH" label in the figures) and depict how its flows share the available resources according to the defined HQoS policy.

TABLE I
Flow types used in the evaluation

Flow parameters		Flow type labels	
CCA	RTT	Gold TVF	Silver TVF
Cubic	5 ms	"G5"	"S5"
Cubic	40 ms	"G40"	"S40"
Cubic	0.3 ms	"G.3"	"S.3"
Cubic	0.7 ms	"G.7"	"S.7"
Prague	5 ms	"Gp"	"Sp"
Prague	40 ms	"Gp40"	"Sp40"

A. Dynamic scenarios without HQoS

The first set of evaluation scenarios investigates the performance of our modified VDQ-CSAQM method implemented in P4. To this end, in these simple use cases, we only assume a number of subscribers having the same or different resource sharing policies expressed as TVFs (Gold or Silver) and show how the desired weighted fairness is guaranteed among them. Note that in these scenarios uses the original PPV packet marking algorithm [27] without its DeepQoS extension.

1) Data center environment with different RTTs: This evaluation scenario is similar to the one presented by Yu et al. in their recent paper on HCSFQ (see Fig. 11 in [1]). We use the same settings except that the bottleneck capacity is 10Gbps (instead of 40Gbps). There are four TCP Cubic flows in the system, arriving one after another as depicted

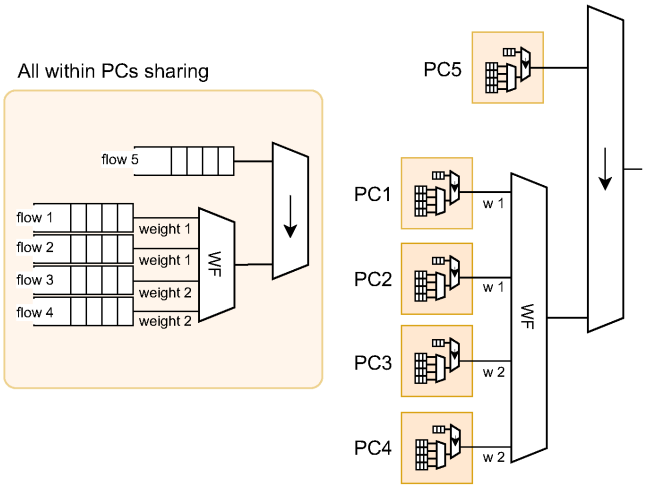


Fig. 9. HQoS WFQ example configuration

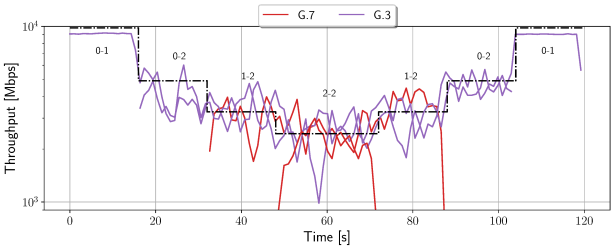


Fig. 10. FIFO, Data center env. with mixed RTTs

in Fig. 10 and 11. The first two flows entering at 0sec and 15s have 0.3ms RTT, while the RTT for the other two flows starting at 30s and 45s is 0.7ms. As shown in Fig. 10, high bandwidth fluctuation of the flows can be observed without VDQ-CSAQM (considering a simple FIFO with tail dropping) and a stable fair resource sharing is not achieved. However, in Fig. 11 VDQ-CSAQM results in much better fairness and more stable resource sharing with quicker convergence to the desired rate. Note that VDQ-CSAQM applies the same policy (Gold TVF) to all flows, representing a fair throughput allocation. The observed efficiency and behavior is similar as for HCSFQ (see Fig. 11b in [1]).

2) Fairness between Classic and Scalable flows: To verify how our VDQ-CSAQM implementation works with L4S traffic, we have mixed classic (Cubic) and scalable (Prague) TCP flows with various traffic intensities, but with the same 5ms RTT. Fig. 12 depicts the observed throughput of various

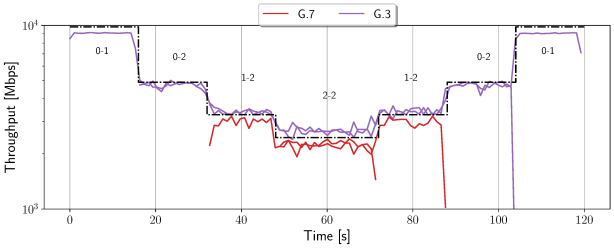


Fig. 11. VDQ-CSAQM, Data center env. with mixed RTTs

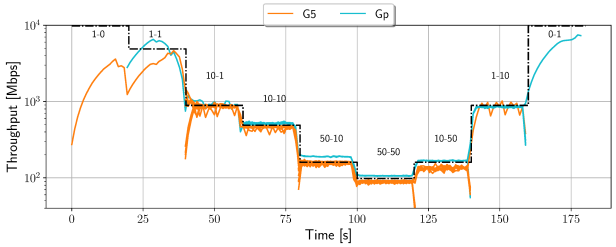


Fig. 12. Mixed classic and scalable TCP flows, 5 ms RTT

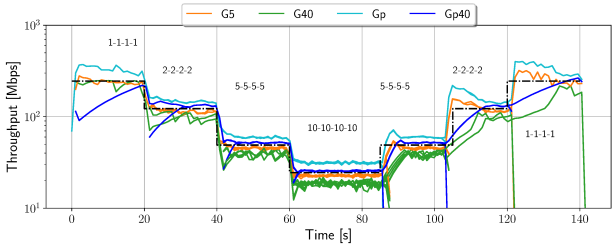


Fig. 13. Mix of classic and scalable TCP flows, 5 and 40 ms RTTs

number of Cubic (orange) and Prague (blue) flows that coexist in the same network and use the same (Gold) policy. The scenario starts with a single classic flow, then other flows enter the system and/or terminate in every 20s. The figure depicts the number of classic and scalable flows in the stable periods (in this order). One can observe that a single classic flow (0-20s) has a slow convergence time and cannot fully utilize the available capacity. As the number of flows becomes larger, the fairness even between flows with different CCA is maintained well and the per-flow throughput is close to the ideal (dashed curve). It even holds when the difference between the number of classic and scalable flows is large. One can also observe that the throughput values of TCP Prague flows are slightly above the calculated ideal and the curve of classic ones. Scalable CCAs are more aggressive than loss-based classic methods, and though VDQ-CSAQM provides in-network support for classic flows to increase their sending rates faster, the classic TCP behavior remains the same and the unused capacity is immediately occupied by the scalable flows.

3) Classic and Scalable flows with different RTTs: In this scenario, we examine how the resource sharing is affected by heterogeneous RTTs which can happen over the Internet. We consider both L4S and Classic flows with both 5ms and 40ms RTTs. In contrast to other evaluation scenarios, the bottleneck capacity is set to 1Gbps since with 10Gbps bottleneck speed the utilization and adaptation speed of a small number of flows with 40ms RTT have especially been affected by the tiny buffers of the programmable switch. We believe that such unfairness is acceptable at high speeds in systems with limited flow counts. As shown later in this section, for larger number of flows, even with 40ms RTT the fairness and convergence time become much better.

Fig. 13 shows the scenario with flows from four different flow types: 1 flow from each type are launched at the beginning, at 20s 1 additional flow from each type joins, at 40s 3-3-3-3, and at 60s 5-5-5-5 more flows start. The flows terminate

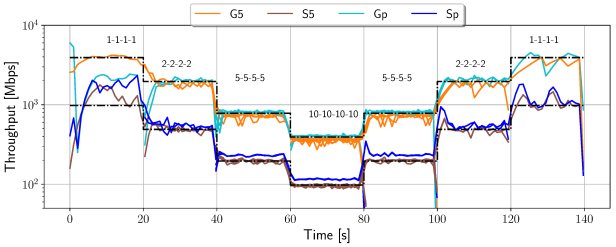


Fig. 14. Mix of classic and scalable TCP, gold and silver policies

in reverse order at 80s, 100s and 120s. Though the 40ms flows converge somewhat slower (esp. for 1-1-1-1 and 2-2-2-2 cases), the obtained fairness still remains good. The throughput curves are close to the ideal regardless of the number of flows. Similarly to the previous case, L4S flows get somewhat more throughput than Classic ones. In addition, RTT unfairness is also visible, giving slightly higher throughput to flows with 5ms RTT in each CCA class. The P4 implementation of VDQ-CSAQM results in similar behavior as the original algorithm of [3].

4) Resource sharing policy enforcement: In addition to equal sharing, PPV framework can express rich resource sharing policies. In this scenario, one half of both classic and scalable flows apply Silver policy, while the other half of them are marked according to the Gold TVF, defining an expected 4:1 (G:S) resource share. The RTT is 5 ms for all flows and the bottleneck capacity is 10Gbps. One can observe in Fig. 14 that the desired resource share is realized between the Gold and Silver flows regardless of their CCA (classic or scalable). Similarly to the previous cases, there are larger deviances from the ideal if the number of flows are small, but as their counts increase the throughput curves run close to the ideal, giving slightly higher share to L4S flows within the policy classes.

B. Dynamic scenarios with HQoS

In this section, we focus on the performance of the proposed DeepQoS marker. To this end, in each scenario we select a designated user whose traffic aggregate consists of multiple flows with a predefined HQoS policy among them. In addition to aggregated throughput of end users, detailed measurement results are separately presented for the designated user called household (HH) traffic.

1) Dynamic Household traffic: Fig. 15 shows our first DeepQoS scenario in which there are 25 classic and 25 scalable TCP flows as a static background traffic (each is considered as a separate TA). The traffic of a household as depicted in Fig. 9 is marked by DeepQoS. Each TA applies a Gold policy (TVF) and the RTT is 5ms. New flows in the household traffic start in every 10s. Since all the TAs apply the same policy, the expected fair share is about 200 Mbps for each TA. The aggregated throughput of the household ("HH") is shown separately on the top, while the flow shares within the household traffic is depicted on the bottom. First, flow PC1-3 starts (orange), when it is alone it utilizes the whole share of the HH. As other flows join, they share the dedicated capacity of the HH according to the HQoS policy shown in Fig. 9. The

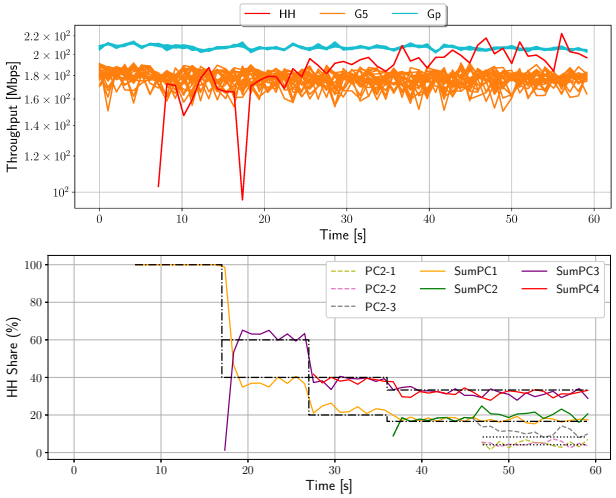


Fig. 15. Dynamic household traffic

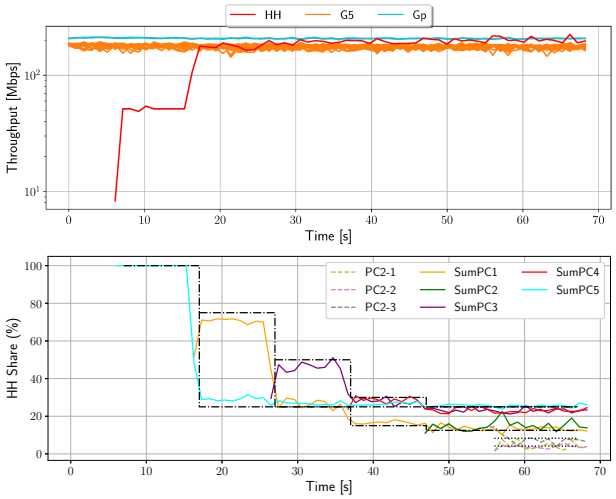


Fig. 16. Dynamic household traffic where PC5 served with strict priority

ideal flow shares calculated from the HQoS policy are marked by dashed/dotted lines. One can observed that for PC2 both the aggregated throughput and the share of its subflows are close to the ideal curves. The aggregated throughput of the HH (red curve in the top figure) shows similar behavior as the background flows (orange and blue). Though HH flows use classic Cubic CCA, as the number of subflows increases it results in more stable rate control and better utilization similar to L4S background flows. This scenario illustrates that the resource sharing targets are met on all hierarchy levels: on the top level between the TAs as defined by the TVF, between the PCs and between subflows of the same PC as defined by the DeepQoS marking. We emphasize here that VDQ-CSAQM is unaware of the HQoS hierarchy, it just maximizes the transmitted Packet Values while it also differentiates the queuing delay between classic and scalable traffic.

2) Dynamic Household traffic with a priority PC: Fig. 16 shows the previous scenario with a slight modification: PC5 receives gaming traffic (PC5-1) which is modeled by an unresponsive UDP flow with a constant sending rate of 50Mbps.

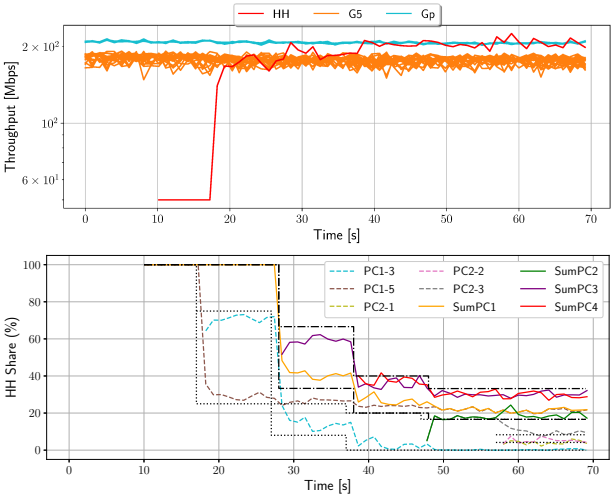


Fig. 17. Dynamic household traffic where flow 5 of PC1 served with strict priority

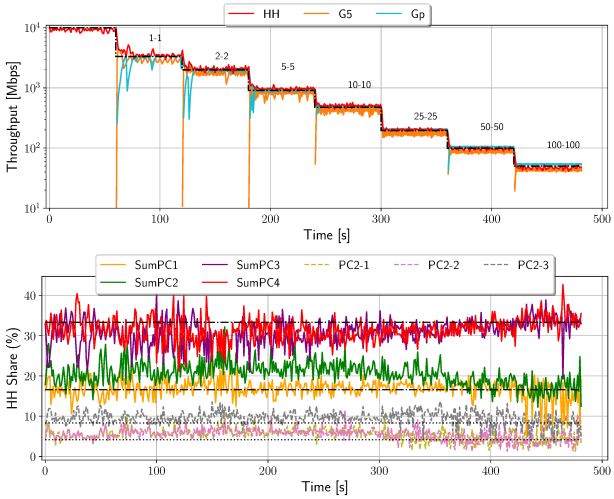


Fig. 18. Static household traffic without strict priority sources

PC5 is served with strict priority in the HH therefore it can consume the 50Mbps without any restriction over the entire measurement interval. PC1-PC4 share the remaining 150 Mbps bandwidth with the same ratio as in the previous scenario. The gaming traffic experiences 0% packet loss even though there is about 0.5% packet loss in the classic queue. This is the natural consequence of the PPV concept: if some flow's throughput ends up under its fair share, the bottleneck does not drop any of its packets. The total throughput of the HH is also in pair with the curves of the background flows as in the previous scenario.

3) Dynamic Household traffic with a priority flow: Fig. 17 depicts a similar case to the previous scenario, but instead of giving strict priority to the total traffic of PC5 within the HH traffic, we only prioritize a single gaming flow (PC1-5) of PC1. Note that the flows of PC1 have no high priority in the aggregated traffic of the HH. In this case, the gaming traffic does not experience any packet loss until its constant sending rate of 50Mbps does not exceed the fair share of PC1. When PC4 starts the transmission at 40 sec, PC1's fair share drops below 50 Mbps, leading to packet losses in the gaming traffic. As a consequence, flow PC1-3 (the other flow of PC1) is slowly suffocated by starvation since the gaming traffic (PC1-5) consumes all the available bandwidth of PC1. This scenario demonstrates the capability of DeepQoS to provide loss-less delivery for the gaming flow, if there are enough resources for the given PC in the HH (at least the capacity for serving the demand of the strict priority flow).

4) Static HH with dynamic background: In the following scenario depicted in Fig. 18, we investigate the robustness of DeepQoS policy enforcement in the presence of dynamic background traffic. In the first phase, a static HH traffic is generated from the beginning to the end of the scenario, consisting of flows with classic CCA. The traffic mix of the HH and the applied HQoS policy are the same as in the last of phase of Fig. 15, having hierarchical WF scheduling among the PCs and their flows. At 60s, two background flows (1-1 classic and scalable) as separate TAs arrive in the system.

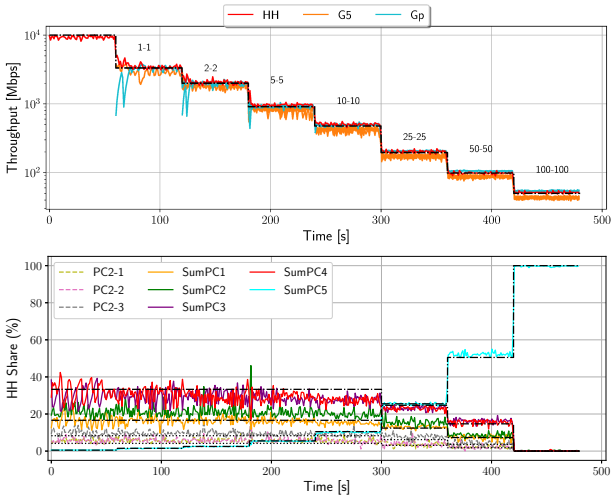


Fig. 19. Static household traffic where PC5 served with strict priority

Then, in every 60s, we launch additional background flows (1-1, 2-2, 6-6, 10-10, 30-30, 50-50 and 100-100) where each of them represents a TA. The red curve in the top figure represents the aggregated throughput of the HH, indicating fair resource share among all the TAs in the system. In addition, the throughput ratio of flows in the HH traffic is not affected by the increasing traffic load and thus remains the same during the entire measurement period. It demonstrates that DeepQoS with VDQ-CSAQM is robust and reacts well to dynamic background traffic.

5) Static HH with a priority PC and dynamic background: In this scenario the static HH traffic includes a strict priority gaming PC5 as the last state in Sec. VII-B2. The background traffic is the same as in the previous scenario. In contrast to the previous cases, the gaming traffic is considered as an L4S flow, thus it is directed into the high priority queue of the VDQ-CSAQM scheduler. The rest of the HH's traffic consists of classic TCP flows. Fig. 19 shows that while the aggregated throughput of the HH follows its ideal, the 50 Mbps sending rate of gaming flow (PC5-1) contributes more and more to the

aggregate as the background load increases. In addition to per-flow throughput, we also monitored the queueing delay in the L4S queue of the scheduler. VDQ-CSAQM kept the delay low for L4S traffic including the gaming flow: the min/max/average delay was 0.04/0.31/0.094ms. At the same time, the observed resource share between the flows of the HH follows the defined HQoS policy well.

The bottom figure clearly shows how PC5's share becomes larger and larger in the HH traffic, while the other flows share the remaining bandwidth as defined by the HQoS policy. In the last stage (after 210 sec) of the scenario the allocated (ideal) throughput of the HH drops to 47.5Mbps and, as a result, the system can only serve the gaming traffic with dropping about 6% of its packets. Since PC5 has strict priority in the HH, flows of every other PC are completely starved.

C. Static scenarios

In this section, we analyze the performance of the proposed method in a large number of static measurement scenarios with various network conditions. The aggregated figures show the calculated throughput deviation for each flow type. The throughput deviation for a given throughput sample x_i of flow/TA i is defined as $\frac{x_i}{\text{ideal}_i} - 1$ where ideal_i is the ideal throughput of flow/TA i according to the applied TVF and HQoS policy. For each flow we calculated throughput samples in 1s time windows. The average, the 10th and the 90th percentile of the throughput deviation samples are depicted for each flow type. Most scenarios mix flows from two types while the total number of flows is varied from 2 to 200. We examine symmetric traffic mixes where the number of flows from the two types is the same, and asymmetric ones where the first flow type contributes 10% or 90% of the flows (labeled with "-.1" or "-.9", resp.). In some cases, a household (HH) user is also added to the traffic mix to demonstrate the performance of HQoS policy enforcement. Each measurement scenario lasts at least 20s, resulting in 20 or more throughput samples for each flow (the transients are disregarded).

Fig. 20 depicts the scenarios where all flows apply a Gold policy, and there is no designated household traffic in the system. We consider the following flow type settings: classic Cubic flows only with 5ms RTT for reference ("5"), classic Cubic flows with different RTTs (5ms-40ms: "5-40" and 0.3ms-0.7ms: ".3-.7"), and finally a mix of either classic Cubic or L4S Prague flows with 5ms RTT ("5-p"). In general the fairness is quite good in all cases, significant deviations can only be observed when the number of flows is small (2 or 4), and the fairness improves as the number of flows in the traffic mix grows. In scenario "5", the average fairness is perfect by definition and the deviation decreases as the number of flows increases. In the "5-40" cases, flows with 5ms RTT experience somewhat larger throughput, but the difference is much smaller than expected from the RTT difference. In the asymmetric cases of "5-40-.1" and "5-40-.9" the deviation from the ideal is somewhat larger, especially when only 10% of the flows are with 5ms RTT (5-40-.1) and they have more room to compete with other flows having 40ms RTT. In the ".3-.7" scenarios, the deviation becomes larger as the number of flows increases. One can observe that flows with larger RTT (0.7ms) experience higher throughput. This is due to that VDQ-CSAQM updates cut-off values (CTVs) in every 8ms which is much larger than the typical RTT. It means that a calculated CTV is valid for tens of RTTs which likely hurts the flows with lower RTT more significantly. Note that this issue can be fixed by shorter update periods which requires some changes of the P4 implementation design. The fairness is still quite good with small deviation from ideal. The "5-p" scenarios contain flows with both Cubic and Prague CCAs while the RTT is 5ms for each flow. One can observe that scalable Prague flows obtain larger throughput. First, the rate reduction of a Prague flow in case of congestion is much smaller than with Cubic CCA, meaning that they can better utilize their fair share. Second, Prague flows are ECN capable and their packets marked with ECN-CE are not dropped, in some cases, especially when the number of flows is high, the ratio of ECN marking reaches 10-15%, which means that they experience a similar throughput boost. The results are similar in the "5-p-.9" scenarios. In case of "5-p-.1" scenarios, Cubic flows experience higher share than Prague ones. This is due to the VDQ-CSAQM design detail (Sec. VI), that 8% of the bottleneck capacity is reserved for cubic flows (the difference between the serving rates of the two virual queues.

As a reference, Fig. 21 depicts the same scenarios with a simple FIFO/tail-drop scheduler where the buffer size is 1ms. For easy comparison we use the same limits on the y axis as on the original figure. We excluded the "5-p" scenarios since we had no reference L4S scheduler in P4. By comparing the two figures it is easy to see the huge improvement in fairness with VDQ-CSAQM in all cases. Even for the simplest scenario "5", the deviation is significantly decreased by the use of VDQ-CSAQM. The fairness degrades most for the scenarios "5-40", some measurement samples do not even fit to the [-0.5, 1] range. These outliers get more than twice or less than half of the ideal throughput. In the ".3-.7" scenarios, both the average fairness and the deviation degraded significantly compared to VDQ-CSAQM.

Fig. 22 depicts the same scenarios as Fig. 20 with the traffic of a single household (HH) added to each case. The HH is considered as a single TA consisting of multiple subflows. The ideal throughput of the total household is the same as that of any other flow type in the scenarios. The HH usually experiences a slightly higher throughput than its fair share, but considering that this TA consists of 6 parallel TCP flows the resulted resource share is still good. Adding the HH as a new TA does not significantly change how the rest of the flows share the system capacity (compared to Fig. 22). In the "5-40" and "5-40-.1" scenarios, when there is a high number of flows with 40ms RTT, the deviation of the HH's aggregated throughput becomes higher. The flows with 40ms RTT utilize their fair share harder because of their slower convergence. Considering that both the number of TCP flows per user (6 for HH, 1 for all other), and the RTT difference (5 vs. 40ms) work against fair sharing, the method results in a quite good performance. For the "5-p-0.1" case for large number of flows the HH obtains a higher share. In these scenarios, the system is less stable, because of the aforementioned 8% difference

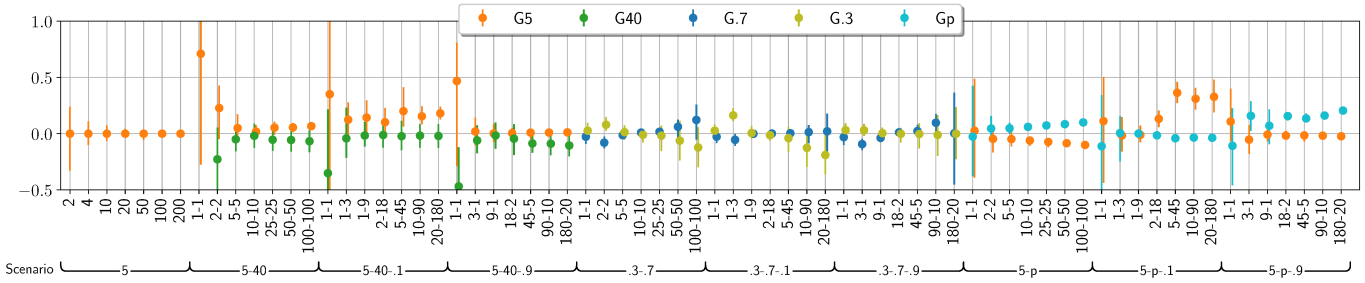


Fig. 20. Resource sharing for static traffic mix

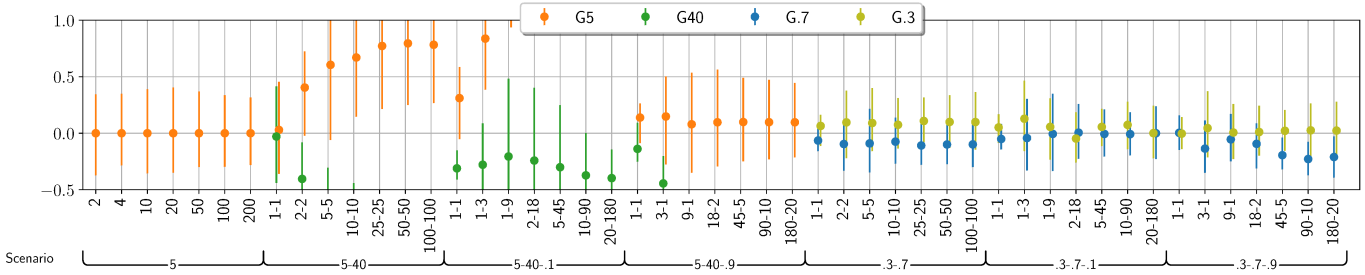


Fig. 21. FIFO resource sharing for static traffic mix

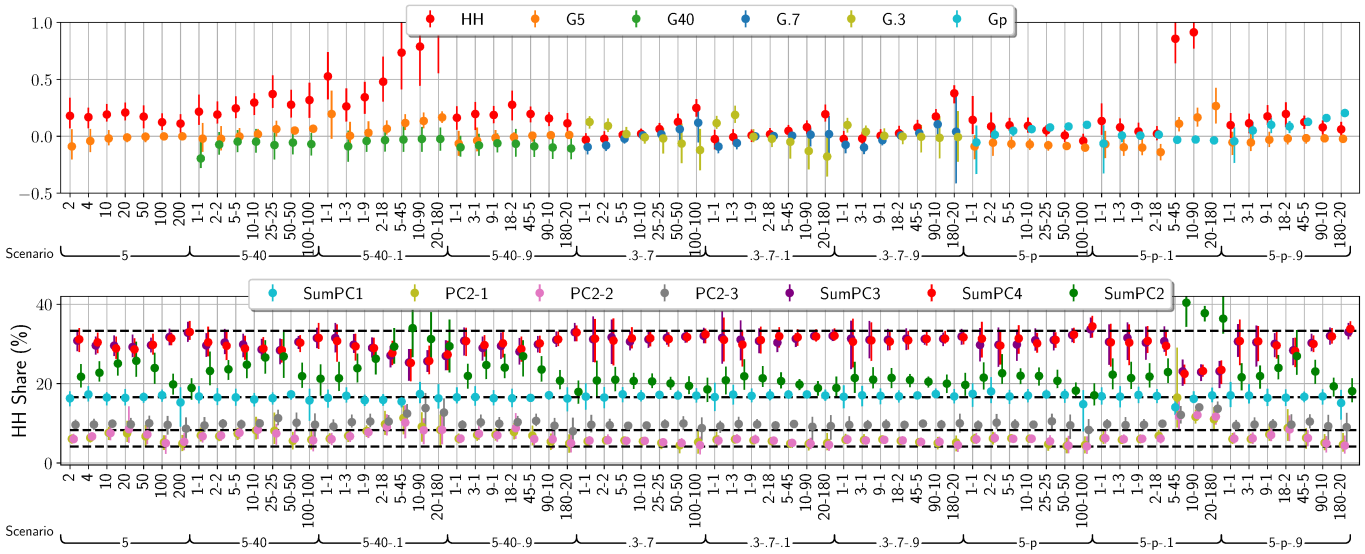


Fig. 22. HQoS resource sharing for static traffic mix with the traffic of a household included

in the service rates of virtual queues. In the rest of the "5-p" scenarios, the experienced throughput values are close to the ideal. The bottom figure depicts the resource share of flows within the HH according to the applied HQoS policy. In most scenarios, the throughput values of both flows and PCs are close to the ideal. One can also observe that the highest deviation from the ideal throughput ratios are in those "5-p-.1" scenarios that contains much more Prague flows than Cubic ones (5-45, 10-90 and 20-180). In these cases, the aggressiveness of more TCP flows per traffic aggregate (either the PC aggregates or the HH aggregate) matters more, explaining the larger deviation. Note that this level of deviations can only be seen in the presence of large number of L4S flows and when the composition of the TAs fundamentally differs (i.e., each background TA is represented by a single TCP flow while the HH consists of 6 flows that leads to better overall utilization).

Fig. 23 depicts measurements with the same traffic mixes but instead of applying the same Gold policy for all the TAs, one of the flow types has Silver policy in every scenario. According to the TVFs, it means an expected 4:1 resource share between Gold and Silver flows, resp. In the top figure the first flow type in the labels is Gold (G) and the second is Silver (S), while in the bottom figure it is the other way around. For the "G5-S5" scenarios for larger number of flows the 4:1 weighted fairness can be realized accurately. For fewer flows, Silvers experience slightly more throughput than their ideal

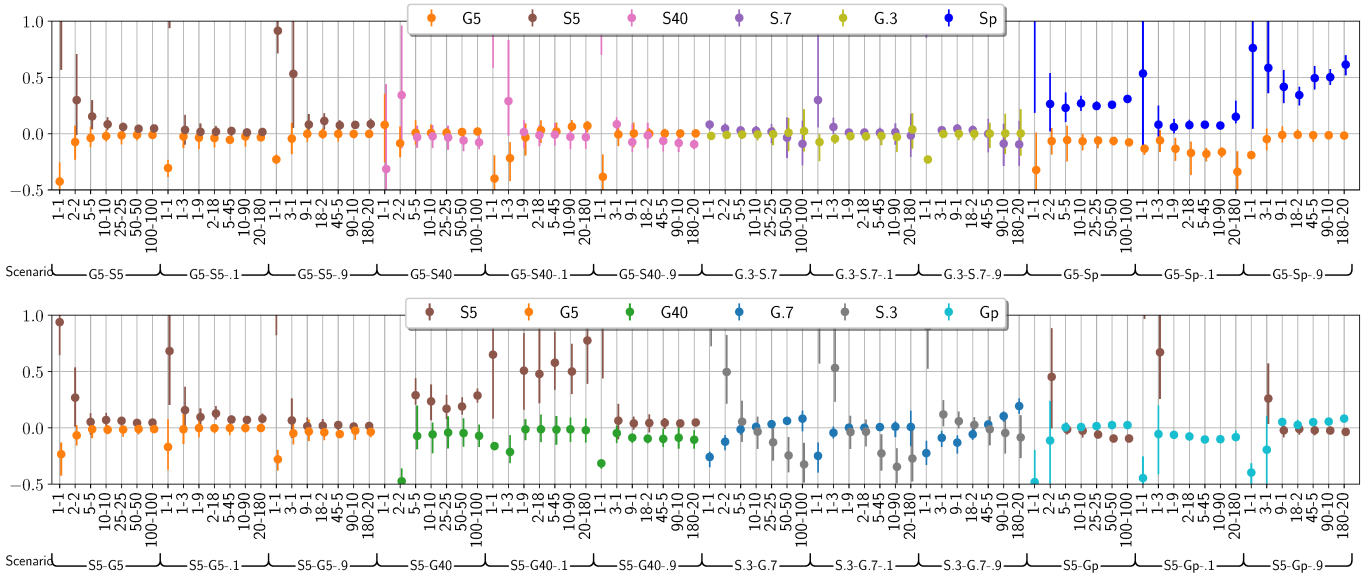


Fig. 23. Resource sharing for static traffic mix with Gold ans Silver policies

allocation. This is not surprising since Silver and Gold flows increase their congestion window in the same way with the same clocking (RTT is the same), but the ideal rate of Silver flows is 1/4th of the Gold ones which can be reached faster, and Silvers have the potential to utilize the available unused resources till Golds reach their fair share. (Note that the "G5-S5" scenario has the same traffic as the "S5-G5", "G5-S5-.1" as "S5-G5-.9", and "G5-S5-.9" as "S5-G5-.1". The scenarios are kept for easy comparison of the figures). For the "G5-S40" scenarios, the experienced fairness is good, except when the number of flows is limited (e.g., 2 or 4). In contrast, the "S5-G40" cases show much higher throughput deviations, since Gold flows with large RTT can receive higher throughput (4 times more than Silvers) but they can increase their sending rate much slower than the Silver flows, leading to slower convergence and underutilization. This is especially visible in the "S5-G40-.1" case, when only 10% of the flows are Silver with 5ms RTT, thus these flows have a good chance to grab more capacity than their fair share by occupying the unused resources of Gold flows. Note that even in this imbalanced case the overall fairness is acceptable and much better than with FIFO queues. The throughput of each individual Gold flow is very close to the ideal, and thus the higher sending rates of Silver flows do not have a significant impact on the Gold ones. For the "G.3-S.7" scenarios with data center-like RTTs, the fairness is close to perfect. Interestingly, slight unfairness can be observed when the number of flows is large, the reasons being similar as for Fig. 20. In the opposite "S.3-G.7" cases high deviations can be seen for each setting. Though flows with 0.3ms RTT get somewhat less throughput than their ideal, the resulted fairness is still in an acceptable range. The observed deviations originate in the too large update period (8ms) of the current VDQ-CSAQM implementation and the effect of heterogeneous RTTs. The fairness of "S5-Gp" scenarios is even better than the results of "5-p" in Fig. 20. It is not surprising and caused by the different CCA behaviors. The Cubic CCA reduces the sending rate more than the Prague CCA and its flows are marked with the Silver policy, resulting in lower desired resource share. An interesting case is "S5-Gp-.1", where the 8% difference in the service rates of applied virtual queues makes the desired resource sharing impossible, leading to higher relative deviance than 1.0 (out of the range shown in the figure). Despite this phenomenon, Gold Prague flows only get slightly less throughput than what they could ideally have. For the "G5-Sp" scenarios the fairness is somewhat worse than for "5-p" where the same policy is applied for all TAs. In this case, the different behaviors of CCAs work against reaching the desired resource share. Note that similarly to previous cases, the throughput of each individual Gold TA is only slightly below its ideal and the higher sending rates of Slivers do not significantly degrade the performance of Gold TAs.

D. Limitations

As shown previously, the flow types we mix in different scenarios result in significant unfairness in a FIFO-based bottleneck. We do not expect perfect resource shares with VDQ-CSAQM either, rather a good enough fairness, for the following reasons.

I) The Tofino-based P4 switch has been designed for use cases where large packet buffers are not needed (e.g., data center networks). Achieving full utilization with different TCP CCAs and different number of flows requires right-sized buffers [31] (e.g., proportional to the bandwidth-delay product), especially in transient cases. With the current settings the small buffers limit the performance of flows with large RTT (40ms) in some cases, but even for smaller RTT it might be a drawback for many CCAs. This also results in that few TCP flows (1 to 4) usually cannot utilize the bottleneck perfectly, in that case the fairness is also less perfect, because the throughput is very dependent on the actual CCA used. Programmable switching ASICs appeared on the market few

years ago, targeting data center networks. However, it is very likely that ASICs for different business use cases (e.g., AANs) will also be designed in the future, potentially having larger packet buffers and thus remedying the issues mentioned in this paragraph.

II) TCP CCA controls the bitrate by two mechanisms: 1) changing the congestion window based on congestion detection and 2) self clocking. Self clocking works much better when each flow has it own buffer, which is true for FQ schedulers [12], but not for VDQ-CSAQM.

III) Different flow aggregates have different aggressiveness. Prague CCA on one hand reduces the sending rate less in case of congestion than Cubic CCA. On the other hand, Prague CCA increases its sending rate slower in lack of congestion than Cubic, as it mimics a NewReno-like behavior. A flow aggregate with several TCP flows (e.g., a household) both decreases less for congestion and increases its sending rate faster. An unresponsive UDP flow can be especially aggressive, and harder to control in a shared buffer.

IV) For flows with very small RTT (0.3ms and 0.7ms) the update period of 8ms in the P4 implementation of our VDQ-CSAQM scheduler might be too long, raising further issues on how the control and data plane interaction and thus the feedback loop in programmable switches could be accelerated for supporting real-time control in AQMs or other algorithms.

V) Finally, the service rates of the virtual queues limit the bottleneck utilization to 98% (or to 90% if only L4S flows are present). This slight underutilization is compensated by the resulting lower queue delay.

VIII. Conclusion

High-speed hardware switches has not been designed to support complex Hierarchical Quality of Service (HQoS). We proposed DeepQoS, a HQoS capable core-stateless packet marker which can be used to mark resource sharing policies of different layers of HQoS simultaneously and effectively in a single point. It extends the existing core stateless Per Packet Value marking, and similarly to that different aggregates can be marked independently from each other. Using DeepQoS a HQoS hierarchy of arbitrary depth can be realized with a very simple scheduler. DeepQoS can freely be combined with the existing remarking-based HPPV solution, which is more appropriate at higher aggregation. DeepQoS marking can be used with any Core Stateless Per Packet Value Scheduler. These schedulers do not need to be HQoS aware and can be used as is. To demonstrate the simplicity of scheduling we implemented our Virtual Dual Queue Core Stateless Active Queue Management (VDQ-CSAQM) proposal on programmable switches. Our evaluation demonstrates that the constrained Traffic Management engine of the programmable switches can be extended using DeepQoS to indeed realize a HQoS hierarchy of arbitrary depth. We argue that by combining DeepQoS with the PV-remarking concept of HPPV [21], complex HQoS trees required to control resource sharing in access aggregation networks and in cloud networks can be realized on programmable switches.

Appendix A
Rate samples generated by DeepQoS components

Lemma 1 (SP component)**.** *Assuming n input flows so that the probability of packet arrival from input i is* $P[Input_i] = S_i/\sum_{k=1}^{n} S_k$ *and for each input i:* $r_{in}^{(i)} \in [0, S_i]$ *input samples are chosen uniformly at random, SP-Marking algorithm generates* $r_{out} \in [0, \sum_{k=1}^{n} S_k]$ *samples that follow a uniform probability distribution in the aggregated throughput range.*

Proof. Let assume that r_{out} is a random variable representing the output of SP-Marking algorithm. For any $Y \in [0, \sum_{k=1}^{n} S_k]$:

$$P[r_{out} < Y] = \sum_{i=1}^{n} P[\text{Input}_i] P[\sum_{k=1}^{i-1} S_k + r_{in}^{(i)} < Y \mid \text{Input}_i] =$$

$$\sum_{i=1}^{n} \frac{S_i}{\sum_{k=1}^{n} S_k} P[r_{in}^{(i)} < Y - \sum_{k=1}^{i-1} S_k \mid \text{Input}_i]$$

Let us assume that $\sum_{k=1}^{j-1} S_k \leq Y \leq \sum_{k=1}^{j} S_k$. Then the above probability can be calculated as

$$\sum_{i=1}^{n} \frac{S_i}{\sum_{k=1}^{n} S_k} P[r_{in}^{(i)} < Y - \sum_{k=1}^{i-1} S_k \mid \text{Input}_i] =$$

$$\sum_{i=1}^{j-1} \frac{S_i}{\sum_{k=1}^{n} S_k} \times 1 + \frac{S_j}{\sum_{k=1}^{n} S_k} \frac{Y - \sum_{k=1}^{j-1} S_k}{S_j} = \frac{Y}{\sum_{k=1}^{n} S_k}.$$

□

Lemma 2 (WF component)**.** *Assuming n input flows so that the probability of packet arrival from input i is* $P[Input_i] = S_i/\sum_{k=1}^{n} S_k$ *and for each input i :* $r_{in}^{(i)} \in [0, S_i]$ *input samples are chosen uniformly at random, WF-Marking algorithm generates* $r_{out} \in [0, \sum_{k=1}^{n} S_k]$ *samples that follow a uniform probability distribution in the aggregated throughput range.*

Proof. We assume that the indexes of the n input flows are in increasing order according to their projected capacity. Let r_{out} be a random variable representing the output of WF-Marking algorithm. For each input i, the rate transformation function that maps $r_{in}^{(i)} \in [0, S_i]$ samples into $[0, \sum_{k=1}^{n} S_k]$ is denoted by $f_i(.)$. For any $Y \in [0, \sum_{k=1}^{n} S_k]$:

$$P[r_{out} < Y] = \sum_{i=1}^{n} P[\text{Input}_i] P[f_i(r_{in}^{(i)}) < Y \mid \text{Input}_i] =$$

$$= \sum_{i=1}^{n} \frac{f_i^{-1}(Y)}{\sum_{k=1}^{n} S_k} =$$

Assuming that $R_{j-1} < Y \leq R_j$, the inverse transformation can be written as $f_i^{-1}(Y) = (Y - R_{j-1})W_{j,i} + R_{j-1,i}$.

$$= \sum_{i=1}^{j-1} \frac{S_i}{\sum_{k=1}^{n} S_k} + \sum_{i=j}^{n} \frac{(Y - R_{j-1})W_{j,i} + R_{j-1,i}}{\sum_{k=1}^{n} S_k} =$$

$$= \frac{1}{\sum_{k=1}^{n} S_k} (\sum_{i=1}^{j-1} S_i + \sum_{i=j}^{n} ((Y - R_{j-1})W_{j,i} + R_{j-1,i})) =$$

$$= \frac{1}{\sum_{k=1}^{n} S_k} (\sum_{i=1}^{j-1} S_i + \sum_{i=j}^{n} (R_{j-1,i} - R_{j-1})W_{j,i}) + \sum_{i=j}^{n} (YW_{i,j})) =$$

$$= \frac{Y}{\sum_{k=1}^{n} S_k} + \frac{1}{\sum_{k=1}^{n} S_k}\left(\sum_{i=1}^{j-1} S_i + \sum_{i=j}^{n}(R_{j-1,i} - R_{j-1}W_{j,i})\right).$$

For the proof, we only need to show that the last parenthesis is zero. After rearranging the equation to be proved is:

$$\sum_{i=j}^{n}(R_{j-1,i} - R_{j-1}W_{j,i}) = \sum_{i=1}^{j-1} S_i. \quad (1)$$

This equation can be proved by induction starting with $j = n$ as the base case:

$$R_{n-1,n} - R_{n-1}W_{n,n} = (R_{n,n} - \delta_n) - (R_n - \delta_n) \times 1 = \\ = S_n - \sum_{k=1}^{n} S_k = -\sum_{k=1}^{n-1} S_k.$$

Then let us assume that the statement holds for $j = m + 1$. Then we can easily show that it holds for $j = m$ as well:

$$\sum_{i=m}^{n}(R_{m-1,i} - R_{m-1}W_{m,i}) = \sum_{i=m}^{n}(R_{m-1,i} - R_m W_{m,i} + \delta_m W_{m,i}) = \\ = \sum_{i=m}^{n}(R_{m,i} - R_m W_{m,i}) = S_m + \sum_{i=m+1}^{n} R_{m,i} - R_m \sum_{i=m}^{n} W_{m,i} = \\ = S_m + \sum_{i=m+1}^{n}(R_{m,i} - R_m W_{m+1,i}) = S_m - \sum_{k=1}^{m} S_k = -\sum_{k=1}^{m-1} S_k,$$

where the equation in the last line comes from the induction hypothesis. □

A natural consequence of the above lemmas that the DeepQoS Marking algorithm with arbitrary marking graph results in the same packet value distribution for a traffic aggregate as the original PPV marker without HQoS since the resulting rate samples follow a uniform distribution in the throughput range of the traffic aggregate.

Appendix B
Resource sharing with DeepQoS components

A mentioned previously, in the PPV framework we always drop packets with the minimum packet value. For each congestion level and traffic mix, this mechanism leads to a packet value threshold called CTV that controls the resource share among the traffic aggregates. Packets with values at least the CTV are forwarded, while below that they are dropped or marked with ECN CE. However, the CTV can also be mapped to a rate value (r_{th}) expressing the throughput share of the traffic aggregate at the bottleneck by applying the inverse function of the TVF used for packet marking. In this section, we focus on how the cutoff rate r_{th} can be transformed to cutoff rates $r_{th}^{(i)}$ for each input flow i of DeepQoS components. Then we also discuss the resource sharing properties of the different components.

Lemma 3 (Cutoff rate backpropagation in the SP component)**.** *Assuming n input flows with sending rates S_i ($i = 1, \ldots, n$), let $r_{th} \in [0, \sum_{k=0}^{n} S_k]$ be a cutoff rate. For each input i, there exists an $r_{th}^{(i)} \in [0, S_i]$ rate threshold so that SP-Marking algorithm maps any $r \in [0, r_{th}^{(i)}]$ to range $[0, r_{th}]$, while any $r \in [r_{th}^{(i)}, S_i]$ to $[r_{th}, \sum_{i=0}^{n} S_i]$, where*

$$r_{th}^{(i)} = \begin{cases} S_i, & \text{if } r_{th} \le \sum_{k=1}^{i-1} S_i \\ r_{th} - \sum_{k=1}^{i-1} S_i, & \text{if } \sum_{k=1}^{i-1} S_k < r_{th} \le \sum_{k=1}^{i} S_k \\ 0, & \text{otherwise} \end{cases}$$

Proof. Let $r_i \in [0, S_i]$ be a rate value of input i. For r_i, SP-Marking results in a transformed rate $\hat{r} = \sum_{k=1}^{i-1} S_k + r_i$. $\hat{r} < r_{th}$ iff $r_i < r_{th} - \sum_{k=1}^{i-1} S_k$. □

This lemma indicates that SP-Marking algorithm implements a strict priority scheduling among the input flows since if $\sum_{k=1}^{i-1} S_k < r_{th} \le \sum_{k=1}^{i} S_k$, the input i is rate limited but for any input $j < i : r_{th}^{(j)} = S_j$ and each input $j > i$ is fully starved by cutting their throughput to zero.

Lemma 4 (Cutoff rate backpropagation in WF component)**.** *Assuming n input flows with sending rates S_i ($i = 1, \ldots, n$), let $r_{th} \in [0, \sum_{k=0}^{n} S_k]$ be a cutoff rate. We also assume that input indexes are ordered according to their projected capacity. For each input i, there exists an $r_{th}^{(i)} \in [0, S_i]$ rate threshold so that WF-Marking algorithm maps any $r \in [0, r_{th}^{(i)}]$ to range $[0, r_{th}]$, while any $r \in [r_{th}^{(i)}, S_i]$ to $[r_{th}, \sum_{i=0}^{n} S_i]$, where*

$$r_{th}^{(i)} = \begin{cases} r_{th} W_{1,i}, & \text{if } 0 \le r_{th} \le R_1 \\ (r_{th} - R_1)W_{2,i} + R_{1,i}, & \text{if } R_1 < r_{th} \le R_2 \\ \ldots, & \ldots \\ (r_{th} - R_{i-1})W_{i,i} + R_{i-1,i}, & \text{if } R_{i-1} < r_{th} \le R_i \\ S_i, & \text{otherwise} \end{cases}$$

Proof. Let $r_i \in [0, S_i]$ be a rate value of input i. If $r_i \in [R_{j-1,i}, R_{j,i}]$ for any j, WF-Marking results in a transformed rate $\hat{r} = R_{j-1} + (r_i - R_{j-1,i})/W_{j,i}$. $\hat{r} < r_{th}$ iff $r_i < (r_{th} - R_{j-1,i})/W_{j,i}$. □

This lemma defines the cutoff rates for each input i, but for the analysis of its resource sharing property one can calculate the ratio $r_{th}^{(i)}/r_{th}$. If $0 \le r_{th} \le R_1$, the resulted ratio is

$$\frac{r_{th}^{(i)}}{r_{th}} = \frac{r_{th} W_{1,i}}{r_{th}} = W_{1,i} = w_i.$$

One can see that the cutoff rates results in the expected weighted fair allocation since the first throughput range is fully used by all the inputs. In the second range when $R_1 < r_{th} \le R_2$, it is a slightly different, since the demand of input 1 is less than its weighted share and the remaining capacity is split between the remaining input flows ($i \ge 2$):

$$\frac{r_{th}^{(i)}}{r_{th}} = \frac{(r_{th} - R_1)W_{2,i} + R_{1,i}}{r_{th}} = \frac{(r_{th} - R_1)W_{2,i} + w_i \frac{S_1}{w_1}}{r_{th}}.$$

Since the demand of input 1 is less than its weighted share if the cutoff (bottleneck) capacity is r_{th}, each input shares the throughput range of 0 to the projected capacity (S_1/w_1) of input 1 proportionally to its weight ($R_{1,i} = w_i \times S_1/w_1$) while the remaining capacity between R_1 and r_{th} is shared between

the remaining inputs $(2, 3, \ldots, n)$, leading to a throughput contribution of $(r_{th}-R_1)W_{2,i}$ (recap that $W_{2,i} = w_i/\sum_{k=2}^{n} w_k$). The same explanation holds for the general case ($R_{j-1} < r_{th} \leq R_j$, $1 < j \leq i$), the projected capacity of input $j-1$ is shared proportionally, leading to $R_{j-1,i}$ (this is the throughput of input i whose transformed rate is R_{j-1}), while the remaining capacity $r_{th} - Rj - 1$ is split between inputs $j, j+1, \ldots, n$ according to their normalized weights.

Note that the above to lemmas can also be applied recursively on a DeepQoS marking graph.

REFERENCES

[1] Z. Yu, J. Wu, and B. Vladimir, "Twenty years after: Hierarchical core-stateless fair queueing," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/yu

[2] S. Nádas, Z. R. Turányi, and S. Rácz, "Per packet value: A practical concept for network resource sharing," in *IEEE Globecom 2016*, 2016.

[3] S. Nádas, G. Gombos, F. Fejes, and S. Laki, "A congestion control independent l4s scheduler," in *Proceedings of the Applied Networking Research Workshop*, 2020, pp. 45–51.

[4] P. E. McKenney, "Stochastic fairness queueing," in *IEEE INFOCOM'90*. IEEE Computer Society, 1990, pp. 733–734.

[5] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on networking*, vol. 4, no. 3, pp. 375–385, 1996.

[6] J. Nagle, "On packet switches with infinite storage," *IEEE transactions on communications*, vol. 35, no. 4, pp. 435–438, 1987.

[7] K. Kogan, D. Menikkumbura, G. Petri, Y. Noh, S. I. Nikolenko, A. Sirotkin, and P. Eugster, "Towards software-defined buffer management," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2337–2349, 2020.

[8] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti, "Numfabric: Fast and flexible bandwidth allocation in datacenters," in *Proceedings of ACM Sigcomm*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 188–201. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934890

[9] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat, "Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing," in *Proceedings of ACM Sigcomm*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787478

[10] S. Nádas, Z. R. Turányi, and S. Rácz, "Per packet value: A practical concept for network resource sharing," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–7.

[11] H. Harkous, C. Papagianni, K. De Schepper, M. Jarschel, M. Dimolianis, and R. Preis, "Virtual queues for p4: A poor man's programmable traffic manager," *IEEE Transactions on Network and Service Management*, 2021.

[12] T. Høiland-Jørgensen, P. McKenney, dave.taht@gmail.com, J. Gettys, and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," RFC 8290, Jan. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8290.txt

[13] A. Kortebi, L. Muscariello, S. Oueslati, J. Roberts *et al.*, "On the scalability of fair queueing," in *ACM HotNets-III*, 2004.

[14] M. Carlson, W. Weiss, S. Blake, Z. Wang, D. Black, and E. Davies, "An architecture for differentiated services," Internet Requests for Comments, RFC Editor, RFC 2475, December 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2475.txt

[15] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 33–46, Feb. 2003. [Online]. Available: http://dx.doi.org/10.1109/TNET.2002.808414

[16] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: Sharing the network in cloud computing," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, p. 187–198, aug 2012. [Online]. Available: https://doi.org/10.1145/2377677.2377717

[17] Z. Cao, E. Zegura, and Z. Wang, "Rainbow fair queueing: theory and applications," *Computer Networks*, vol. 47, no. 3, pp. 367 – 392, 2005. [Online]. Available: //www.sciencedirect.com/science/article/pii/S1389128604002361

[18] F. Kelly, "Charging and rate control for elastic traffic," *European transactions on Telecommunications*, vol. 8, no. 1, pp. 33–37, 1997.

[19] S. Laki, G. Gombos, S. Nádas, and Z. Turányi, "Take your own share of the pie," in *Proceedings of the Applied Networking Research Workshop*. ACM, 2017, pp. 27–32.

[20] S. Nádas, G. Gombos, P. Hudoba, and S. Laki, "Towards a congestion control-independent core-stateless aqm," in *Proceedings of the Applied Networking Research Workshop*. ACM, 2018, pp. 84–90.

[21] S. Nádas, Z. Turányi, G. Gombos, and S. Laki, "Stateless resource sharing in networks with multi-layer virtualization," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[22] M. Menth and N. Zeitler, "Fair resource sharing for stateless-core packet-switched networks with prioritization," *IEEE Access*, vol. 6, pp. 42 702–42 720, 2018.

[23] F. Fejes, S. Nádas, G. Gombos, and S. Laki, "A core-stateless l4s scheduler for p4-enabled hardware switches with emulated hqos," in *In IEEE Infocom (Demo)*, 2021.

[24] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, and M. Hira, "Elmo: Source routed multicast for public clouds," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 458–471.

[25] A. Liberato, M. Martinello, R. L. Gomes, A. F. Beldachi, E. Salas, R. Villaca, M. R. N. Ribeiro, K. Kondepu, G. Kanellos, R. Nejabati, A. Gorodnik, and D. Simeonidou, "Rdna: Residue-defined networking architecture enabling ultra-reliable low-latency datacenters," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1473–1487, 2018.

[26] C. Dominicini, R. Guimarães, D. Mafioletti, M. Martinello, M. R. N. Ribeiro, R. Villaça, F. Loui, J. Ortiz, F. Slyne, M. Ruffini, and E. Kenny, "Deploying polka source routing in p4 switches : (invited paper)," in *2021 International Conference on Optical Network Design and Modeling (ONDM)*, 2021, pp. 1–3.

[27] S. Laki, S. Nádas, G. Gombos, F. Fejes, P. Hudoba, Z. Turányi, Z. Kiss, and C. Keszei, "Core-stateless forwarding with qos revisited: Decoupling delay and bandwidth requirements," *IEEE/ACM Transactions on Networking*, 2020.

[28] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on networking*, vol. 1, no. 4, pp. 397–413, 1993.

[29] L. T. L. fork, "L4S testing kernel," 2021, [Available: https://github.com/L4STeam/linux/commit/6633163. Online; accessed June-2021].

[30] B. Briscoe, M. Kühlewind, and R. Scheffenegger, "More accurate ecn feedback in tcp," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-tcpm-accurate-ecn-14, February 2021, available: https://www.ietf.org/archive/id/draft-ietf-tcpm-accurate-ecn-14.txt.

[31] F. Fejes, G. Gombos, S. Laki, and S. Nádas, "Who will save the internet from the congestion control revolution?" in *Proceedings of the 2019 Workshop on Buffer Sizing*, 2019, pp. 1–6.

Ferenc Fejes achieved his MSc degree at University of Debrecen in 2018. Then he started his PhD studies at the Eötvös Loránd University. Since 2021, Ferenc works as a researcher at Ericsson Research, Budapest, Hungary. His research focusing on the principles and practical applications of core stateless and collaborative resource sharing with the per-packet value approach. He also has experimental knowledge with congestion controls, multipath transport and AQMs.

Szilveszter Nádas received the M.Sc. degree in electrical engineering from the Budapest University of Technology and Economics in 2000. He is with Ericsson Research, Hungary since then. He has been working with Traffic Management for 20 years, and his main interest is controlling Resource Sharing. He is also interested in the interaction of different mechanisms of Traffic Management (e.g., AQM and Congestion Control) and he believes that more coordination among the mechanisms is necessary.

Gergő Gombos was graduated (MSc) in 2012 at Eötvös Loránd University (ELTE) in Computer Science. He defended his PhD in 2018. The topic of his thesis is the Semantic Web and the distributed computing in Hadoop environment. Since 2018 he works as an assistant professor at the Department of Information Systems of Eötvös Loránd University. His research interest includes computer networks, Hadoop and Spark environments, big data architectures and NoSQL databases.

Sándor Laki received the M.Sc. and Ph.D. degrees in computer science from Eötvös Loránd University in 2007 and 2015, respectively. He is currently an Assistant Professor with the Department of Information Systems, Eötvös Loránd University. He has authored over 40 peer-reviewed papers and demo papers, including publications at JSAC, INFOCOM, ICC, and SIGCOMM. His research interests include active and passive network measurement, traffic analytics, programmable data planes, and their application for new networking solutions.