Core-Stateless Forwarding with QoS Revisited: Decoupling Delay and Bandwidth Requirements

Sándor Laki, *Member, IEEE,* Szilveszter Nádas, Gergő Gombos, Ferenc Fejes, Péter Hudoba, Zoltán Turányi, Zoltán Kiss, and Csaba Keszei

***Abstract*—Network QoS, fairness and resource sharing control are not completely solved problems. Available solutions lack scalability due to maintaining flow state, require re-tuning if traffic changes, focus on a limited set of networking scenarios or require complex, centralized controllers and feedback loops. In this paper, we propose a core-stateless solution for closed network domains like access, enterprise and data center networks that handles resource sharing and provides guarantees for per-hop latency, independently. The proposed method enables controlled resource sharing by encoding the utility function of flows to Packet Value markings. This allows expressing resource sharing policies for all possible congestion situations, while operation is completely flow unaware inside the network. In addition, it also satisfies per-hop delay requirements for traffic flows independently. The separation of the delay requirements of the packets from their importance has not generally been possible by existing methods so far. The performance of the proposed method has thoroughly been analyzed by large number of simulations covering both static and dynamic scenarios and was implemented in a cloud-native virtual router implementing all the policies needed for a Broadband Network Gateway, showing good performance and better scalability than existing weighted queuing-based solutions.**

***Index Terms*—Resource Sharing, QoS, Congestion Control, Core-Stateless.**

I. Introduction

QUALITY of Service is one of the most studied area in networking with a vast array of valuable knowledge, but the issues of network QoS and resource management are still not fully solved problems. With the advent of 5G, high speed access networks become general, making the appropriate overprovisioning of backhaul networks difficult and costly. This issue may be handled with robust and scalable Quality of Service solutions that have the potential to support congestion management and SLA enforcement/insurance among traffic aggregates at various levels (flow, user, application, etc.). However, similar problems also arise in multi-tenant data center and residential access networks.

There is a rough consensus that any QoS solutions that keep flow states inside the network will not scale well with the increasing demands. In addition, many novel applications cannot tolerate large delays and jitter thus a comprehensive QoS framework also needs mechanisms for ensuring controlled buffering latency. To overcome the scalability issues of traditional stateful QoS approaches, a number of core-stateless proposals have emerged in the past (see Sec. II), but none of them can be viewed as a complete QoS solution, focusing on either providing fair share among users or providing bounded delay. In core-stateless resource management, the operator policies are encoded into packet markings that are associated with each packet. Then the core of the network operates solely based on these markings with no flow-awareness.

Some QoS properties are trivial to provide in a core-stateless manner. Marking packets with simple priority levels at the edge and performing priority queueing inside the network results in strong differentiation of traffic. Another trivial approach would be to classify flows at the edge to a few distinct delay classes and to introduce different queue length limitations inside the network for these classes. This would allow to differentiate the delay experienced by various flows, but would directly bind resource sharing to delay requirements.

Smarter core-stateless solutions, however, provide more practical features. Core-Stateless Fair Queueing [1] allows the enforcement of equal fairness without relying on well-behaving endpoints, supporting a limited number of policies. Rainbow Fair Queuing [2] goes a step further by supporting weighted fairness among flows in a core-stateless fashion. ABC [3] 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. All of these proposals are limited to a certain kind of resource sharing policy and none of them provides a general way to define arbitrary policies, such as stateful solutions [4], [5].

Controlling delay is also an area, where existing solutions have shortcomings. The simplest network-side method to provide delay differentiation is the use of priority queueing. This, however, couples delay to bandwidth: low delay flows get high bandwidth at the expense of high delay flows. There are quite a few solutions on how to prevent starvation of the low priority flows, but they only provide rudimentary control. As

Manuscript submitted November 23, 2018. The research of G. Gombos was partially supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications). 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)*

S. Laki, G. Gombos, F. Fejes and P. Hudoba are with ELTE Eötvös Loránd University, Budapest, Hungary, and also with Faculty of Informatics, 3in Research Group, Martonvásár, Hungary.

Sz. Nádas, Z. Kiss and Z. Turányi are with Ericsson Research, Budapest, Hungary. At the time of this research, Cs. Keszei was with Ericsson, Santa Clara, USA.

an end-to-end solution to provide low queueing delay for flows L4S (Low Latency, Low Loss, Scalable Throughput) Internet service [6] has been emerged. It proposes DualQ AQM that maintains two queues for classic and L4S traffic, giving higher priority to scalable L4S flows.

In this paper, we introduce a core-stateless QoS mechanism based on our Per Packet Value concept [7], [8], [9] that can independently control resource sharing and delay differentiation. It offers a rich set of resource sharing policies for all the congestion situations, e.g., (at high congestion) all users share the bottleneck equally up to 100 Kbps representing a predefined minimal service, (at medium congestion) a selected user group (Gold) is allowed to get twice the throughput of normal users up to 500 Kbps, (at low congestion) normal users are limited at 250 Kbps, while Gold users are allowed to take the rest of the bottleneck capacity. When a source does not transmit to its allowance the unused capacity is momentarily split between the rest of the flows. Even in a congested bottleneck when a source ramps up its sending rate, it does not experience packet loss until it reaches its allowance.

The per-hop latency of packets can be governed via a small number of delay classes, which are independent of the resource sharing policies. According to a recent survey on techniques to reduce Internet latency [10] the proposed approach belongs to the group of methods aiming to control and reduce the per-hop queueing delay, providing strict guarantees on keeping the latency requirement of each packet.

At the edge of the network each packet is marked with two numbers: the packet value and the delay class. The packet value originally introduced in our previous paper [7] is calculated from the *Throughput-Value Function* (TVF) assigned to the traffic aggregate. TVF represents the resource sharing policy and is essentially the derivative of the utility function of the aggregate, thus the packet value of each packet expresses the contribution (or the importance) of the given packet in the total utility. Note that both packet value and delay class can be carried in newly created or existing packet header fields, such as in DSCP or in an MPLS label (in Sec. IV-D we show how packet values can logarithmically be encoded into 2 bytes).

Inside the network, bottlenecks solely operate on packet values and delay classes, being unaware of the flows/aggregates to be handled. They aim to maximize the total successfully transmitted value (sum of values in the forwarded packets) while fulfilling the queueing latency requirements of the associated delay classes. This simple local mechanism is used to satisfy the QoS requirements (both resource sharing and delay) of traffic aggregates.

We have implemented edge markers and bottleneck queues in NS-3 simulator as well as in a DPDK-based cloud-native virtual router implementing a Broadband Network Gateway (BNG) [11] for access networks. Since edge marker and bottleneck nodes operate independently, the hierarchical scheduling required by BNG nodes could be implemented in the proposed core-stateless framework in a distributed fashion, resulting in better scalability than existing solutions used in production network.

II. Related Work

This paper focuses on how bandwidth and per-hop delay requirements of traffic aggregates at various levels (flows/applications/subscribers) can be decoupled and satisfied in emerging WAN environments. Both delay-aware scheduling and bandwidth allocation have rich literature including utility functions [12], [13], host-based admission control [14], stateful [15], [5], [4], [16] and stateless [17], [1], [2] resource sharing, and priority-based packet scheduling [18], [19], [20], [21] that our work builds on. However, to the best of our knowledge, a flexible and scalable solution, that addresses both delay requirements and resource sharing, has not existed so far.

Stateful Resource Sharing. The majority of existing resource sharing solutions deployed in production networks rely on the common paradigm of weighted queueing. These methods are controlled in a centralized way and use flow states at the bottleneck nodes for packet scheduling. For example, a BNG node in an access network maintains separate queues for each user to ensure per-user QoS, applying Deficit Round Robin scheduling. This problem can be handled by specific hardware solutions consisting of hardware queues, however it results in scalability issues in software switches used in modern cloud-native networking solutions. The cause of the increased complexity is the less efficient software implementation of maintenance and scheduling of many queues. On the other hand, policies that can be expressed by weighted queueing (e.g., hierarchical QoS) are very limited or require complex feedback loops and the run-time modification of parameters to be applied. 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 [4] 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, only assuming a few 100s of active flows in the network. Another approach is BwE [5] that operates a centralized bandwidth broker to manage resource sharing for a globally-deployed WAN and introduces bandwidth functions to define flexible policies. BwE also shares the idea of this paper that routers cannot support the scale and complexity of enforcing per-flow policies. Accordingly, hosts rate limit their outgoing traffic and mark packets using the DSCP field. Routers solely use the DSCP marking to determine which path to use for a packet and which packets to drop when congested. To this end, BwE uses global knowledge of network topology and link utilization as input to bandwidth policers (e.g., running on the end-hosts). Though the goal of this paper is similar to the one of BwE, we propose a solution that, instead of centralized control, uses an open-loop, distributed operation and can react to changes in the traffic mix or available resources two-three orders of magnitude faster. Furthermore, it does not only solve

the resource sharing problem, but also takes into account the different delay requirements of traffic flows.

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 consists of two kind of nodes: 1) stateful Edge Nodes that are located close to the subscribers and responsible for packet marking; 2) stateless Core (or Resource) Nodes deployed in the core of the network that perform simple packet processing solely based on the markings of incoming packets. 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 DiffServ [17] 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 [1] 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. Rainbow Fair Queueing (RFQ) [2] 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 [7] that uses scalar values as packet markings instead of a few colors. PPV method we extend in this paper 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 [12]. Different AQM algorithms [8], [9] have also been proposed as extensions for the PPV concept to provide controlled delay with a single delay target in the past years, but these solutions do not take into account the different delay requirements of various applications and cannot guarantee bounded queueing delay for them. A recent core-stateless resource sharing proposal is ABC [3] 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.

Delay-aware scheduling. Similarly to core-stateless resource sharing solutions, pFabric [21] also assigns a priority-value to each packet and switches implement a very simple priority-based scheduling/dropping mechanism, requiring no flow state or complex rate calculations at the switches, no large switch buffers, and no sophisticated congestion control mechanisms at the end-hosts. It is shown in [21] that through different definitions of the priority-value pFabric can be used to support different objectives e.g., minimize flow completion time of web or data center workloads or maximize the number of packet deadlines met. However, its goal is not generic resource sharing, rather shortest-job-first data center networking, resulting in near theoretically optimal flow completion times, especially for short jobs. The PIFO programmable scheduler [18] demonstrates the feasibility of implementing programmable priority queues in modern switch hardware and thus how delay-aware scheduling can efficiently be implemented. Universal Packet Scheduling [19] attempts to realize any given packet schedule with a single scheduling algorithm in the switches (which coincidentally also requires a priority queue). Although promising, flexible packet scheduling alone cannot achieve arbitrary network-wide objectives defined such as the ones described in this paper. The key problem with existing delay-aware scheduling approaches based on packet priorities is that they strictly couple delay to bandwidth: low delay flows get high bandwidth at the expense of high delay flows and can also result in starvation issues.

Core-stateless networking solutions used to be a widely examined research area in the early 2000s, however only a very few proposals [7], [8], [9], [3] have been published in the past few years. With the emerging trend of softwarization in computer networks their usability has become possible even in production environments. In this paper we revisit core-stateless solutions and investigate how they can be extended to decouple bandwidth allocation and per-hop delay requirements.

III. Concept

In this section, we overview the proposed concept extending our resource sharing solution called Per Packet Value (PPV) with delay differentiation and sub-classes.

A. Utility Functions

One of the most important aspects of QoS is to govern how resources are shared between traffic aggregates traveling through shared bottlenecks. In the remaining part of the paper, we refer to a traffic aggregate as a group of packets to be handled similarly, e.g., the traffic generated by a given subscriber or end-user, but other kinds of grouping is also possible. One solution for this problem is the application of network utility functions [13], [12] that relate the state of applications and networks with the user satisfaction and can be used for rate control and resource allocation [22], [4]. Since the source rates are constrained by link capacities in the network, an optimization problem can be formulated to solve the resource sharing problem by maximizing the aggregated utility in the system. In the problem we aim to solve, operators can define resource sharing policies at various aggregation levels: per user, per application group, per individual application levels or as the combination of these.

For example, for each user a the utility function $U_a(x)$ can be defined to represent the user's satisfaction or with other terms the value realized by the operator if throughput x is assigned to user a from the shared resources of the network. In addition, network operators can incorporate their own preferences, by giving higher or lower value to more or

less important users or traffic, such as emergency services or background downloads, respectively.

Note that the utility function is an increasing, strictly concave and differentiable function of x over the range $x \geq 0$. According to network utility maximization [12], [13], by maximizing the aggregated utility value of end-users we can not only maximize the overall satisfaction of subscribers, but the total profit of the network operator.

B. System model

As with other core-stateless proposals [2], [1], [7], packets are labeled at the network edge where operators can express their resource sharing policies. In our system model, each packet is marked by two values: a Packet Value expressing the importance of the packet and a delay class that determines its per hop delay requirement. At packet arrival, the Delay Class determines where to insert the incoming packet in the buffer, while the Packet Value is used as a drop precedence if the associated delay requirement cannot be satisfied without dropping one or more packets from the buffer. Note that packet marking generally requires stateful information (e.g., per flow, per application or per user) to determine the policy to be applied. Policies can vary, for example, more important users may be assigned more resources; less and more important packets of an application can be differentiated, but it is also possible to, e.g., mark TCP SYN packets or DNS requests as of higher value and lower delay requirements to increase their chance of delivery and thus to avoid long initial timeouts caused by packet losses and long queueing.

Core Nodes (e.g., routers) in the network operates solely based on packet markings, and strive to selectively transmit packets to achieve maximum value transmission (while keeping packet ordering for each flow, if required). A simple realization uses a mid-drop queue similar to the one in [21]: when the queue becomes full, packets with the lowest importance are dropped (potentially from the middle of the buffer) to make room for a more valuable packet. This design has the advantage that overloaded Core Nodes remain stateless, working unaware of both the number of users and the policies to be applied. Though packet marking is stateful, this task can be performed independently for each traffic aggregate, distributing the computations among Edge Nodes, resulting in high scalability. Note that Edge Nodes also take part in packet forwarding and thus they are Core Nodes as well.

To calculate packet values for each packet of an aggregate we extend the concept of the Per Packet Values (PPV) method [7], supporting various resource sharing policies from strict priority scheduling to WFQ-like behaviors. Accordingly, each resource sharing policy is described by a Throughput-Value Function (TVF) defined as the derivative of a utility function: $V_a(x) = U'_a(x)$. This derivative function represents the extra value (e.g., the increase in profit for the operator) that can be generated if extra throughput is given to traffic aggregate/user a. Then TVFs are used to label packets with a Packet Value (PV) expressing the gain of the operator when the packet is delivered (marginal utility in other words). The goal is then to maximize the total aggregated PV delivered with respect to the available resources. It is important to note that the PVs of a traffic aggregate are not constant, but vary over time.

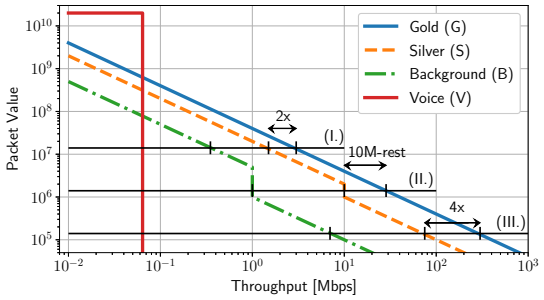


Fig. 1. TVF examples, log-log scale.

Operator policies are described by Throughput-Value Functions (TVFs) (marked by $V(.)$) that define the PV distribution of a traffic aggregate for any sending rate B. Specifically, the throughput contribution of packets with PV at least $V(b)$ in the traffic aggregate should be b (for any $b : 0 \leq b \leq B$). Accordingly, at high congestion only packets with high PVs are transmitted, more precisely packets with PV above a certain PV threshold 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 a natural property resulted by the applied drop minimum-PV first 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.

In a stationary congestion situation at a particular Core Node maximizing the transmitted value of packets is equivalent to transmitting all packets with PV above CTV and dropping the ones below. This is because transmitting any packet with PV below this threshold at the expense of one above it would decrease the total value delivered. In such a case, a source with TVF $V(x)$ will be allowed to transmit at rate R such that $V(R) = CTV$. If the source was allowed to transmit at a higher rate, it could only be via serving packets with value lower than the CTV or dropping packets with value higher than the CTV, respectively, leading to reduced total value transmitted. Thus *maximizing the transmitted value leads to maximum conformance to the policies*.

In a real network the CTV is not constant over time, but follows changes in available capacity, the amount of offered traffic and the composition of PVs in it. As congestion increases the CTV also rises (as more valuable packets get dropped). Compared to other metrics of congestion (e.g., utilization, packet loss/marking ratio) the CTV not only considers the amount of offered traffic in relation to resources, but it also takes the importance of the offered traffic into account.

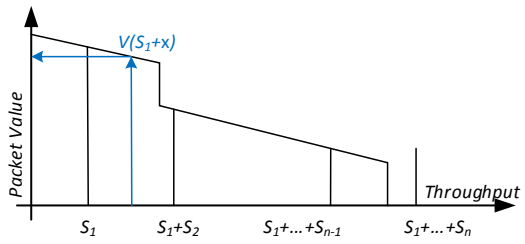


Fig. 2. Partitioning a TVF among its RS sub-classes

C. Encoding Resource Sharing Policies

Fig. 1 shows different operator policies expressed as TVFs that describe conditional weighted resource sharing between three Resource Sharing (RS) classes (Gold, Silver, and Background) while Voice has strict priority up to 64 Kbps. The intersections of the TVFs with horizontal lines representing different congestion levels (with different CTVs) define the desired throughput of the RS classes at the given congestion level. Until Silver users reach 10 Mbps, Gold ones get twice the throughput of Silvers (I.). When the throughput of Silver users is above 10 Mbps Gold sources get 4 times the throughput (III.). In between Silvers get 10 Mbps and Golds get the rest (which will be between 20 Mbps and 40 Mbps) (II.). For this range of congestion levels the Silver policy pretends the behavior of a rate limiter at 10 Mbps. The figure also shows a RS class called Background that has a small share of moderate PVs to keep connectivity going, and larger bandwidth is only allowed if there is little or no congestion. It can be seen how horizontal sections of TVFs describe strict priority; parallel lines weighted sharing; and vertical lines allow changing between weights depending on a throughput condition (like a rate limiter).

D. Resource Sharing Sub-Classes

The versatility of the concept allows differentiating traffic within traffic aggregates effectively achieving a simple hierarchical policy. For example, a BNG application requires differentiating 4 Resource Sharing (RS) sub-classes per traffic aggregate. A single TVF (as above) is still required to determine the total share of the user in which the sub-classes are served using strict priority. Thus the top priority sub-class of a user can potentially starve the lower priorities of that user, completely independently of how the RS sub-classes are served for other users. Fig. 2 shows how a TVF ($V(.)$) is partitioned among n RS sub-classes. The bandwidth momentarily available to the user (as defined by its TVF) is sub-divided among sub-classes using strict priority. Each sub-class i has a maximum bitrate ($R_i^{\max}$), thus packets above this rate shall be dropped or marked with zero PV. For each sub-class i, the contribution to the aggregated throughput is $S_i = \min(R_i, R_i^{\max})$, where R_i expresses the instantaneous rate of user's traffic in sub-class i. In contrast to the case when the whole TVF is used to determine the PV of a transmitted packet, if the packet belongs to sub-class i, the applied TVF domain is limited to range $(\sum_{j=1}^{i} S_j, \sum_{j=1}^{i+1} S_j]$ so that for any throughput value $0 < x <= S_i$: $V_i(x) = V(\sum_{j=1}^{i} S_j + x)$. The packet value distribution in the whole traffic aggregate is the same with and without the usage of sub-classes, but packets of high priority sub-classes are marked with higher values (using the first part of the TVF) than ones from low priority classes.

E. Decoupling Delay and Throughput Requirements

Providing bounded delay is important for a number of applications. In many cases the delay requirement of a traffic aggregate is independent of its bandwidth requirement or general importance. For example, real-time layered video (for gaming or conversations) could be transmitted with its base layer marked more important than its enhancement layers. This would ensure that enhancement layers get transmitted only when all base layers (and other more important traffic) can also be transmitted (including traffic with potentially less stringent delay requirements).

In general, many applications would benefit from a low-delay, elastic network service [23], especially since low-delay applications cannot rely on retransmissions and end-to-end control loops to govern transmission rates. The capability of sending excess in a layered fashion would be a good solution. Thus, in addition to PV the proposed method also associates a *Delay Class* expressing the acceptable per-hop delay of the aggregate, per bottleneck, to each packet. Note that for providing end-to-end delay guarantees additional mechanisms are needed that are outside the scope of this paper.

Fig. 3 provides an overview of the proposed delay-aware packet scheduler that aims at maximizing the total transmitted value, while keeping the delay requirements of packets. For the purpose of value calculation, we assume that the PV of a packet is zero if its queuing delay has exceeded the requirement of its delay class.

Assuming $n \geq 1$ delay classes and an egress port capacity C, the buffer at the Core Node is split into different non-overlapping regions $Q_1, Q_2, \ldots, Q_n$ defined by the per-hop delay requirements $D_1 < D_2 < \cdots < D_n$. The buffer starts with region Q_1 so that any packets in this region meet the queueing delay requirement D_1 with the assumption of outgoing rate C. Q_1 is followed by the region Q_2 and any packets in Q_1 and Q_2 can be served within delay requirement D_2, and so on. For delay class (DC) i, the size of corresponding buffer region denoted by $|Q_i|$ is calculated as: $|Q_i| = D_i \times C - \sum_{j=1}^{i-1} |Q_j|$. Similarly to other proposals like PIFO [18], incoming packets are inserted into the end of a queue region corresponding to their delay class. The main challenge is to maximize transmitted value and keep delay requirements at the same time. To achieve this, we propose five simple and local mechanisms:

Delay-based scheduling order: At packet arrival the packet with DC c is inserted into the end of buffer region Q_c ensuring that packet can be served within its delay requirement D_c.

Value-based dropping: Whenever a packet with DC c arrives into a full buffer region Q_c, if its PV is less than or equals to the minimum PV in the preceding buffer regions $Q_1, \ldots, Q_c$, it is dropped. Otherwise, one or more packets with the smallest packet value(s) from $Q_1, \ldots, Q_c$ are dropped until there is enough room for the new packet.

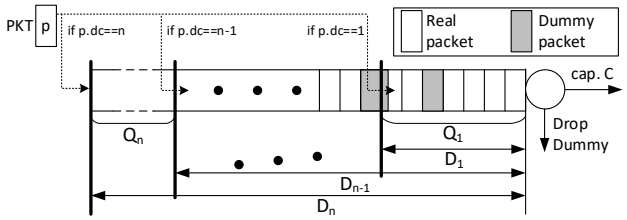


Fig. 3. Conceptual overview of the delay-aware Core Node.

Replacing dropped packets with dummy packets: If the total length of dropped packets exceeds the length of the packet to be enqueued, the last dropped packet (with the highest PV) is kept, but truncated to the length of the excess and is marked as *dummy*. These dummy packets keep their original PV and are handled the same way as normal packets in the buffer, but will not actually get transmitted, hence cause no real queuing delay to packets after them. They are kept to reserve space in the buffer and thereby avoid extensive packet drops due to small sized, low delay, high PV packets, as shown in Fig. 4. One can observe that Packets 1 and 2 (small, low delay of DC=1, high PV) arrive to Q_1, when there are three larger packets of lower value (A, B and C) already waiting there such that C is in Q_2 (Fig. 4a). Without dummy packets both B and C have to be dropped, since after the arrival of packet 1 (B is dropped) C shifts ahead to Q_1 and will be in the way when packet 2 arrives (Fig. 4b). Keeping parts of packet B as a dummy prevents packet C from shifting into Q_1, so it does not need to be dropped when packet 2 arrives (Fig. 4c).
Queue expansion by inserting zero-value dummy packets: A packet is always inserted to the end of its respective buffer region Q_c. The buffer space before the packet is filled with dummy packets of PV=0. This again helps to keep the arrived packet longer in its buffer region preventing unnecessary drops. Fig. 5a depicts an arrival situation when this is needed. Packets 1, 2 and 3 arrive in this order, 1 and 3 to Q_1 and 2 to Q_2. In Fig. 5b, dummy packets are not used, packet 2 is queued to region Q_1 and thus has to be dropped by packet 3 that has higher value and arrives later. In Fig. 5c, the inserted dummy packets prevent the unnecessary drop of packet 2. The dummy packets with PV=0 behave like many 1 byte long packets. However, we insert them as single placeholder packets and split them whenever needed, i.e., when they occupy the border of buffer regions and a packet is to be inserted there.
Dequeueing real packets and dropping dummies: Whenever a port is idle, the dummy packets from the front of the buffer are dropped (if any) and the first real packet is dequeued and sent out.

IV. Implementation

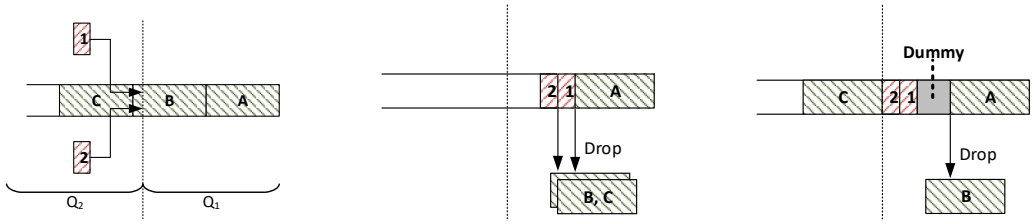
In this section we provide practical and scalable algorithms for Edge and Core Nodes in order to demonstrate the feasibility of the proposed concept.

A. Value Marking at the Edge

Edge nodes assign both a packet value and a delay class to each packet passing through, according to predefined operator policies. A practical packet marking method should satisfy three key requirements: 1) for stable traffic the observed TVF should follow the configured TVF; 2) the method should react quickly to changes in the arrival rate; and 3) should have modest computational and memory requirements.

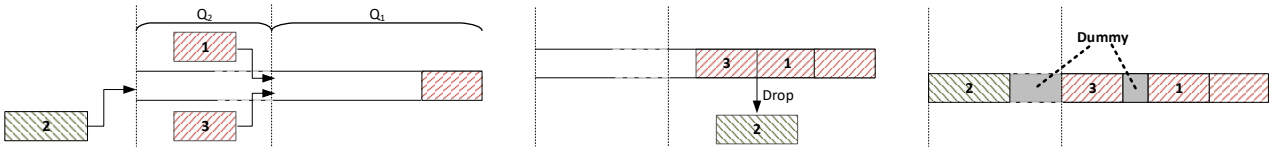
The authors of [7] propose hierarchical token buckets to assign packet values to each packet of a flow. The computational and storage complexities of their solution are much higher than our packet marking algorithm and are beyond practical application, requiring $O(N)$ states for distinguishing N different packet value levels.

In this section, we propose a probabilistic marking algorithm called *pMarker* that only requires three state variables while its computational complexity is also $O(1)$. The pMarker algorithm is based on measuring the arrival rate (R) of flows to be marked. Then the TVF $V(.)$ is applied to a throughput value chosen uniformly at random from the range 0 to the estimated rate ($v = V(\mathbf{rnd}(0, R])$). For stable incoming rate this results in the desired packet value distribution. To estimate the incoming rate, we have experimented with various methods including Exponentially Weighted Moving Average (EWMA) also used by RFQ [2] and a simple time-slotted moving average. Both proved to work well for stable traffic loads, but they reacted slowly in transient cases. To remedy these issues our marker algorithm uses a modified, variable-rate token bucket of size *TBS* (set to a few times the MTU, 6 kBytes in our experiments). The TVF to apply is denoted by $V(.)$. We also introduce an averaging time constant δ. Note that δ is the time-scale at which the throughput changes in the aggregated traffic can be captured (set to the RTT in our experiments).
Initialization: The token bucket rate and the token level are both set to zero: $R = 0$ and $tokens = 0$.
Updating token bucket: On packet arrival the bucket is updated and the new token level is calculated as follows: $tokens_{\text{new}} = tokens_{\text{old}} + R \times \Delta t - p.size$, where Δt is the time elapsed since the last packet is received and $p.size$ is the size of the incoming packet. Based on the new token level $token_{\text{new}}$, three different cases are distinguished: stable period, rate under and overestimation.
Stable period: If the new token level is between zero and the bucket threshold *TBS*, the current rate estimate R is considered stable and therefore there is no need for further adjustment. In this case, the new token level is accepted and the packet is marked to $v = V(x)$ where $x = \mathbf{rnd}((0, R])$.
Rate underestimation: If $tokens_{\text{new}}$ is less than zero, the rate R is considered underestimated and thus is updated as $R_{\text{new}} = R_{\text{old}} + (|tokens_{\text{new}}| + \beta)/\delta$. To compensate for rate underestimation we assign $v = V(x)$, where $x = \mathbf{rnd}((R_{\text{old}}, R_{\text{new}}])$ chosen uniformly at random. This emulates the behavior of HTB used in [7] when all the buckets below R_{old} are depleted. To avoid increasing R and choosing from this new interval too often, the increase in the rate estimate is accelerated with β (1500 byte). The token level is also set to β after the rate increase overwriting the negative $tokens_{\text{new}}$.
Rate overestimation: If $tokens_{\text{new}}$ is above *TBS*, the rate estimate should be reduced. The new rate is calculated as $R_{\text{new}} = R_{\text{old}} - (tokens_{\text{new}} - TBS)/\delta$. If this new estimate is below $p.size/\delta$, the rate represented by this single packet in a δ



(a) Initial state (b) Without dummy packets (c) With dummy packets

Fig. 4. The role of dummy packets with PV>0 when high value packets arrive at a full buffer region.



(a) Initial state (b) Without dummy packets (c) With dummy packets

Fig. 5. Inserting dummy packets with PV=0 as placeholders to prevent unnecessary drops.

interval, the traffic had a long idle period. In this case we reset the system as if a packet arrived to a freshly initialized buffer: $R_{\text{new}} = p.size/\delta$ and $tokens_{\text{new}} = 0$. In any case the incoming packet is tagged with $v = V(x)$, where $x = \mathbf{rnd}((0, R_{\text{new}}])$.

B. Resource Sharing Sub-Classes within a Traffic Aggregate

In some use cases (e.g.,BNG), RS sub-classes within the traffic aggregate of a user have to be differentiated to provide precedence among various services such as voice (VoIP), IPTV-video and best-effort data. As described in Sec. III the bandwidth momentarily available to the user (with TVF $V(.)$) shall be sub-divided among RS sub-classes using strict priority. In the marker we implemented this policy by applying n instances of the pMarker rate estimator to calculate per-class rates R_is (see Fig. 2). Similarly to the pMarker method, for each incoming packet of sub-class i we first chose a throughput value x form range $(0, R_i]$ uniformly at random. If $x < R_i^{\max}$, the assigned PV is $v = V(\sum_{j=1}^{i-1} S_j + x)$, otherwise $v = 0$. We periodically (e.g., in every 10 ms) update all rate estimators, in order to have accurate estimations for RS sub-classes.

C. Delay class marking

In addition to PVs each packet carries its delay class (DC). DCs can be completely independent of the resource sharing policy, thus any part of an aggregate (or RS sub-class) can be marked with any DC. Note that if packets requires in-order delivery, they should be marked with the same or increasing DCs.

D. Core Node

A practical implementation of the proposed delay-aware Core Node (see Section III-E) for k DCs is shown in Fig. 6. For each DC $i \in [1, n]$, a standard FIFO packet buffer Q_i is introduced, which represents the corresponding buffer region shown in Fig. 3. We code PV logarithmically to 2 octets, v denotes the unsigned integer representing the coded PV while the zero PV is coded to $v = 0$.

To avoid costly searches in the queue and applying mid-queue dropping, we emulate this behavior by 1) maintaining the number of bytes to be dropped from each coded PV (v), and 2) dropping it only when a packet with that PV v reaches the head of the buffer. To achieve that, two histograms are introduced for each Q_i: 1) $H_{\text{IN,i}}$ for representing how many bytes of each v is stored in the queue for forwarding; 2) $H_{\text{DROP,i}}$ for keeping track of how many bytes of each v is to be dropped. Then instead of directly dropping b bytes of coded PV v we simply move b bytes with coded PV v from the input histogram to the drop histogram by applying $H_{\text{IN,i}}(v) \mathrel{-}= b$ and $H_{\text{DROP,i}}(v) \mathrel{+}= b$. The corresponding packet(s) will remain in Q_i until they reach the head of that FIFO, when it is finally dropped. This also implements the *Replacing dropped packets with dummy packets* behavior as described in Sec. III-E. This is because $H_{\text{DROP,i}}$ maintains bytes to be dropped, not packets, and dropping a few bytes from a PV leaves the rest bytes of a long packet in $H_{\text{IN,i}}$. Note that the histograms are implemented as simple counter arrays indexed by the coded PVs.

Let p be the packet arrived; c its delay class, v its coded PV and $p.size$ its size in bytes. We calculate the actual queue length of Q_i as $q_i = \sum_j H_{\text{IN,i}}(j)$, i.e., only considering bytes not to be dropped at the queue head. The free space in $Q_1..Q_i$ is calculated as $s_i = C \times D_i - \sum_{j=1}^{i} q_j$. (Note that it can be temporarily negative.) The operation of a Core Node can be split into the following steps:

Enqueue: Upon packet arrival, if there is enough space in Q_c, i.e., $s_c \geq p.size$, the packet is *inserted* to the end of Q_c. If $s_c > p.size$, a dummy packet with $v = 0$ and size $s_c - p.size$ is *inserted* before the actual packet. If there is not enough space we *mark for dropping* bytes with the smallest coded PVs (w) and with $w < v$ from $Q_1 \ldots Q_c$ until either $s_c = p.size$ or we cannot drop more (as all $H_{\text{IN,j}}(w) = 0$, where $j = 1 \ldots c$ and $w = 0 \ldots v - 1$). If we succeed, then p is *inserted* to the end of Q_c, otherwise it is dropped upon arrival. Finally, *Queue*

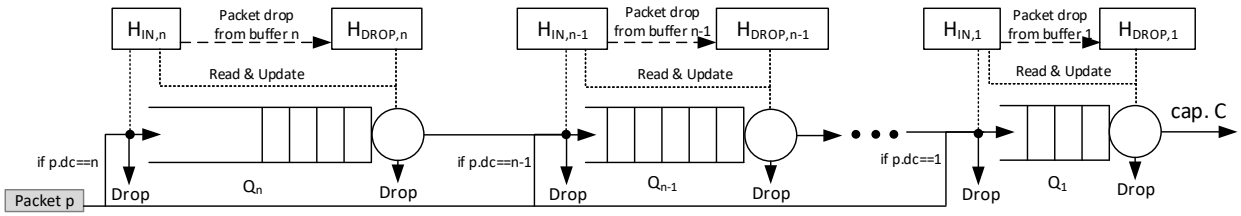


Fig. 6. Implementation of the delay-aware Core Node. The different buffer regions are emulated by a sequence of FIFO queues and packet value histograms.

maintenance is performed for $Q_1 \dots Q_c$.

Insert: When a packet p is inserted to Q_i, the packet (or packet pointer) is stored in the FIFO queue and $H_{\text{IN,i}}(v)$ += $p.size$.

Marking for dropping: When b bytes of coded PV v is to be dropped from Q_i, it is checked whether $H_{\text{IN,i}}(v) \geq b$ and if it is, then that amount of bytes are marked for dropping by $H_{\text{DROP,i}}(v)$ += b and $H_{\text{IN,i}}(v)$ −= b.

Queue maintenance: This step compresses all packets towards the first FIFO (Q_1). Starting from Q_1, queues are filled with packets from queues with higher DCs. A queue Q_i is considered *filled* if $s_i \leq 0$. If it is not yet filled the first packet p in Q_j, where $j > i$ is examined.

(1) Dummy packets with PV>0: If $H_{\text{DROP,j}}(v) > 0$ and $v > 0$ then p is *dropped*.

(2) Dummy packets with PV=0: If $v = 0$ then its size is decreased ps −= $\min(H_{\text{DROP,j}}(0), p.size)$. Then if $p.size = 0$ it is dropped. Otherwise we determine the amount of free space (s_i) in the buffer being filled (Q_i). If $s_i < p.size$ then we create a dummy packet with PV=0 and size s_i, we decrease the size of the original packet $p.size$ −= s_i and also update $H_{\text{IN,j}}$ −= s_i and insert the newly created packet into Q_i.

(3) Packets not to be dropped: If $H_{\text{DROP,j}}(v) = 0$ we *dequeue* p from Q_j and *insert* it to Q_i.

Queue maintenance stops when all packets are advanced to the first possible place, i.e., the first m FIFOs are *filled* and the rest of the packets are in Q_{m+1}. The cost of queue maintenance varies widely: in the best case, every packet has the same size and if a packet is dequeued, a single packet pointer is moved from Q_i to Q_{i-1} for each $i > 1$ indices, thus the computational complexity is proportional to the number of DCs; in the worst case, a maximum sized packet (MTU) is dequeued and all the queues are filled with minimum sized packets ($minps$), resulting in $O(nMTU/minps)$ computational costs since $\lceil MTU/minps \rceil$ packets are moved from Q_i to Q_{i-1} for each $i > 1$ indices. Note that the number of operations in the worst case can reduce the achievable performance (esp. when packet dropping is considered). According to[24], [25], the average packet size in the Internet is between 709 and 891 bytes. It indicates that in the average case 2-3 packets shall be moved from Q_i to Q_{i-1} (for each $i > 1$), resulting in 2-3 times higher computational costs than what we can get in the best case. However, it is still much better than the worst case.

Packet drop: When packet p from Q_j is dropped, it is removed from Q_j, and $H_{\text{DROP,j}}(v)$ −= $p.size$ is updated. Then if $H_{\text{DROP,j}}(v) < 0$, $H_{\text{IN,j}}(v)$ += $H_{\text{DROP,j}}(v)$ and then $H_{\text{DROP,j}}(v) = 0$. Note that dropping a packet has generally a high cost because of memory management operations.

Dequeue: If the outgoing link is idle and there is at least one real packet in the buffer, the dequeue process is performed: 1) the first packet p from the first non-empty queue, Q_j ($j = 1 \dots n$) is *dropped* while $H_{\text{DROP,j}}(v) = 0$; 2) then if a packet p remains, that packet is dequeued, $H_{\text{IN,j}}(v)$ −= $p.size$, and 3) *Queue Maintenance* for all queues is performed to compress the queue.

V. Evaluation in a Virtual Router

To show the practical usability of the proposed approach we have implemented and deployed the proposed packet marking and scheduling algorithms in a cloud-native virtual router implementing a BNG [11] network function. Our prototype is a drop-in replacement of the existing Deficit Round Robin-based Traffic Management (TM) component of the router. Since the devices in the access network are assumed to be unable implementing complex policies, BNG nodes emulate the queueing in the entire access network to provide quite rich policies and ensure QoS among thousands of subscribers.

All the policies supported by the original TM engine and needed for the BNG network function can also be supported by the proposed core-stateless solution. The prototype performance has been analyzed under different realistic workloads. The performance evaluation focused on the delay scheduler component, which reached 5.94 MPPS processing speed when it was not programmed to be a bandwidth bottleneck (no-BN use case), while it reached 3.6 MPPS in case queues were full due to emulated network bottleneck (BN-high case). Processing speed maximum was measured by increasing packet rate until the point where measurement results still exhibited an algorithmically correct behavior. Note that this performance is very close to what production ready virtual BNG (vBNG) nodes provide today and our prototype can be tuned to further improve the performance.

Original DRR-based vBNG node. The original TM engine is based on DPDK's [26] rte_sched [27] and its pipelining model and contains further optimization in order to reach 7 MPPS single-core forwarding speed in production under similar conditions to what is used in our evaluation. It had a stateful scheduler, which results in poor (non-linear) scalability with the number of users.

Measurement setup. All evaluation scenarios described in this section were performed on a compute node (Intel(R) Xeon(R) E5-2699 v4 @ 2.20GHz CPU) running a VM connected to network tester nodes via 10 Gbps links. Existing vBNG products generally solve the resource allocation problem by emulating the entire access network, bringing the dropping

TABLE I
MEASUREMENT SETTINGS AND RESULTS

Source type	Voice	IPTV	BE	Total
Packet size [byte]	128	256	512	
Delay requirement [ms]	2	4	50	
RS sub-class priority	1	2	3	
Sending rate/flow [Kbps]	500	1500	7500	9500
Total sending rate [Mbps]	500	1500	7500	9500
Rate limit ($r_{\max,i}$) [Kbps]	540	1260	9000	9000
no-BN Total R_s [Mbps]	500	1258	7231	8988
no-BN avg/max del. [ms]	.14/.41	.14/.4	.14/.41	
BN-low Total R_s [Mbps]	500	1257	5238	6995
BN-low R_s/flow [Kbps]	500	1255±45	5225±175	
BN-low a/m delay [ms]	1.6/2.0	3.6/4.0	48.1/50.0	
BN-high Total R_s [Mbps]	500	1247	5248	6995
BN-high R_s/flow [Kbps]	500	1240±60	5215±195	
BN-high a/m delay [ms]	2.5/7.0	4.6/9.0	47.6/54.4	

and scheduling decisions to this single location. In our case, the vBNG node has implemented both our packet markers and core schedulers. Note that our implementation can also be used in general software routers, not restricted to BNG nodes.

Markers were run in the context of vBNG workers distributed among different CPU cores as marking can be done independently for each user. The delay scheduler was run on a dedicated IO core where the same core had to deal with the regular IO tasks in addition. When the IO core starts being overloaded these software queues start to grow. When they become full the excess traffic is tail-dropped outside the delay scheduler. We call the maximum performance the processing rate possible without such software queue tail drops. In this section we show single-core IO performance results.

In the evaluation we emulated the traffic load of 1000 users where the same policy (TVF) was applied to mark the packets. Table I details the measurement settings and the throughput results. Note that the bottleneck buffer is sized according to the largest delay requirement (50 ms in our case), but the histogram-based dropping mechanism introduced in Sec. IV requires additional buffer space (e.g., 50% more in our evaluation) for emulating mid-drop behavior. In total 9.5 Mbps downlink (download) traffic was generated by each user. Each aggregate consisted of a Voice, IPTV and Best Effort Resource Sharing sub-classes where Voice had the highest priority access to the network resource (see Section IV-B). The configured per user TM policy contains per RS class and total rate-limits as described in the 'Rate limit' rows of Table I. The offered Voice traffic did not exceed the rate limit for the Voice class, but the offered IPTV and the total traffic was higher than their respective limits.

Measurement results. The observed packet forwarding rates in the scheduler (R_s) are shown in Table I for the different traffic items and the 3 scenarios below.

1) **no-BN** First, the egress port rate is 10 Gbps. In this case the scheduler is not a traffic bottleneck since the total offered traffic is only 9.5 Gbps. Only the Marker may limit the sub-flows by policing to $r_{max,i}$s. In row "**no-BN**" one can show that, as intended, Voice traffic is not limited; Video traffic is limited to the rate limit of its class; and the total flow throughput is limited to 9 Mbps (dropping more from the best-effort traffic). These results verify that the markers work as intended and that the scheduler can support this 3 MPPS load without experiencing CPU overload issues.

TABLE II
PV RANGE AND FORWARDING PERFORMANCE IN MPPS

Packet size	512 PVs	1024 PVs	4096 PVs	16384 PVs
64B	11.3	10.5	8.3	5.1
800B	**5.6**	5.4	3.7	1.7
1500B	**2.9**	**2.9**	2.7	1.0

2) **BN-low** The port rate is now limited to 7 Gbps, thus it becomes a bottleneck. When comparing Total R_s to the previous case, it can be seen that only BE traffic is dropped more (as intended by the policies). Also the rate variance among flows is small, indicating good flow fairness. In this case, the node is also able to handle the incoming traffic load without any CPU performance limitation.

3) **BN-high** To check the CPU limitation of the prototype implementation, we started increasing the arrival rate of packets by decreasing the size of best effort packets (while keeping the throughput), and the CPU bottleneck was found around 370 bytes. It can be seen in Table I that the throughput performance is similar to the previous case.

Traffic of the various RS sub-classes are also bound to different per-hop delay requirements. Table I also contains the measurement of the maximum and average delays experienced by the packets of the various sub-classes. One can observe that the pre-configured values are kept even in case of increased packet rate. The observed maximal per-hop latency values are slightly above the required delays in BN-high scenario when the arrival rate is increased to 3.6 MPPS and the node has started reaching its CPU limitation. Additional queueing delay is caused by software queues outside of the TM engine.

Limitations and further optimization. Our DPDK-based prototype runs on a standard x86 architecture where the performance is affected by many factors like memory access, cache misses or number of CPU cores. We have analyzed our implementation and identified two performance bottlenecks: 1) We use 2 bytes to encode PVs that generate a wide range of PVs, supporting very accurate resource sharing control. However, PV histograms used in our implementation require significant amount of memory, leading to a number of cache misses during operation. This can be remedied by reducing the applied PV range. Using packet size of 800 Bytes (the Internet average) and 512 different PVs, the singe-core packet processing performance of our DPDK-based prototype was 5.6 MPPS (line rate at 37 Gbps) with unresponsive traffic (generated by DPDK PktGen tool). In this case, PVs were encoded into 9 bits instead of 16. The performance for other packet sizes and PV ranges is summarized in Table II. The line rate performance with different packet sizes is marked by boldface. Note that smaller PV ranges lead to lower resource sharing accuracy (5-6% relative error for 512 PVs). 2) Our Core Node runs on a single CPU core. The use of multiple CPU cores could potentially improve the performance, but it requires thread-safe PV histograms (e.g., a granular bin-locking mechanism).

VI. Evaluation using simulations

In addition to the DPDK-based prototype, we have also implemented all the algorithms detailed in Sec. IV in NS-3 [28], [29]. Sources include responsive TCP, unresponsive and aggressive UDP, and Voice (low-rate unresponsive UDP). The applied operator policies are described by the TVFs depicted in Fig. 1. Each TCP source generates 5 parallel TCP flows to emulate a user running multiple TCP-based application at the same time. The TCP CUBIC [30] implementation of Linux Kernel 4.1 through NS-3 Direct Code Execution [31] was used as congestion control algorithm. UDP sources represent very aggressive users. Each UDP traffic aggregate consists of a non-congestion controlled UDP flow, having a much higher sending rate than its allowed throughput share (1.5 times the ideal bottleneck capacity share), using 1500 byte packets. Voice sources illustrate VoIP traffic consisting of a single UDP flow transmitted at a constant rate of 70 Kbps with packet size of 320 bytes. We also assume that there are no bottlenecks on the uplink. Each traffic source is connected to a separate packet marker placed before the Core Node, constituting a single bottleneck dumbbell topology often used for AQM evaluation [32]. In addition to the PV, packet markers also assign a delay class, either DC 1, 2 or 3 to each packet. Note that our bottleneck implementation requires a buffer sized according to the largest delay requirement (DC-3) and additional (50% more) buffer space for emulating the histogram-based mid-drop behavior. The ideal share of traffic aggregates is denoted by black dotted lines in the graphs.

A. Static scenarios

In these scenarios, our main objective is two-fold: 1) showing how the resource sharing performance is affected by various network settings, and 2) how the delay requirements of different traffic classes are satisfied under stable traffic conditions. To this end, we assume long running flows generated by variable number of Gold and Silver subscribers during the 15 sec long simulation under different network settings. Each flow is created in the first second of the simulation and ends in the last one. The number of Gold and Silver subscribers are varied in a wide range: 1-1,1-10,10-1,10-10, 10-50, 50-10 and 50-50, while 20 Voice traffic aggregates are present all along as permanent background traffic. The protocols used for data transmission covers responsive TCP, aggressive unresponsive UDP and low-rate UDP-based unresponsive voice traffic mixes, as described previously. The RTT is set to 10, 40 and 100 ms, while the bottleneck capacity is mostly chosen from 100, 200, 500, 1000 and 10000 Mbps. Though simulating a large number of TCP sources on a 10 Gbps bottleneck proved to be impossible due to the limitations of the NS-3 DCE, we include measurement results (Fig. 8) carried out by the DPDK-based prototype (see Section V). The measurement setup is similar as in the simulations, but the number of subscribers is scaled up to 10020 (5000 Silver-5000 Gold-20 Voice). Three delay classes (DC 1, 2 and 3) are used with delay requirements D_1=2 ms, D_2=5 ms and D_3=10 ms for Voice, Silver and Gold flows, resp. The Voice, Gold and Silver TVFs applied to mark the packets of the specific RS classes are depicted in Fig. 1.

To quantify the performance of the proposed core-stateless QoS framework, two key performance indicators (KPIs) have been calculated: 1) The *relative flow rate* calculated as the ratio of the average throughput of a given flow and the ideal (optimal) throughput of a single Silver flow expresses the throughput 'fairness' obtained for the different network settings and configurations. Note that the ideal relative flow rate of a Silver flow is represented by 1.0. In the top graphs of Figures 7-9 we present the average, 1st and 99th percentile of the observed relative rates for Gold, Silver and Voice traffic. 2) The observed packet-level *queueing delay* is collected for each delay classes to show if delay requirements are satisfied. In the bottom graphs of Fig. 7-9 the average, minimum and maximum of the observed queueing delays are displayed for the different delay classes. The transient periods, i.e., the first and the last seconds of the simulations, when the flows start and end, have been excluded from the calculation of performance indicators.

1) Unresponsive and aggressive UDP sources: The proposed delay scheduler is first evaluated with unresponsive traffic mixes generated by aggressive UDP sources. The traffic mix consists of 20 Voice (DC 1) and variable number of Silver (DC 2) and Gold (DC 3) UDP sources. We assume a single bottleneck in the network whose capacity is varied between 100, 200, 500, 1000 and 10000 Mbps. Fig. 7 depicts the average, 1st and 99th percentiles of relative flow rates (top graph) and the average, minimum and maximum of queueing delay (bottom graph) obtained for the different settings (105 different simulation scenarios in total). For each scenario, the numbers of Silver (x) and Gold (y) subscribers are shown by the 'Sx-Gy' labels along the X-axis.

The top graph of Fig. 7 shows a perfect match between the observed and the calculated optimal resource usage. The optimal relative rates are depicted by black dotted lines for Gold, Silver and Voice flows. The ideal Gold ratio jumps between 2 and 4, depending on the bottleneck capacity and the number of Gold and Silver users in the traffic mix (i.e., on whether Silver flows can reach 10 Mbps).

One can also observe on the bottom of this figure that all the delay requirements are met (as guaranteed by design of the proposed scheduling algorithm).

2) Responsive TCP sources: Our second set of scenarios focus on the handling of responsive flows when both Gold and Silver subscribers generate CUBIC [30] TCP flows (5 long running connections per subscriber) with DC 3 and 2, respectively. As in the previous case, 20 permanent Voice flows with delay class 1 are also present in the scenarios. We assume a single bottleneck in the network whose capacity is varied (100, 200, 500, 1000 Mbps and 10 Gbps). Because of the limitations of NS-3 DCE, the 10 Gbps case has been evaluated in our testbed, using the same general setting as in the simulations. Fig. 8 depicts the average, 1st and 99th percentiles of relative flow rates (top graph) and the average, minimum and maximum of queueing delay (bottom graph) for the different settings (72 + 21 scenarios).

One can observe in the top graph of Fig. 8 that the deviations from the ideal relative rates are slightly larger than for the previous scenario with unresponsive flows. The highest

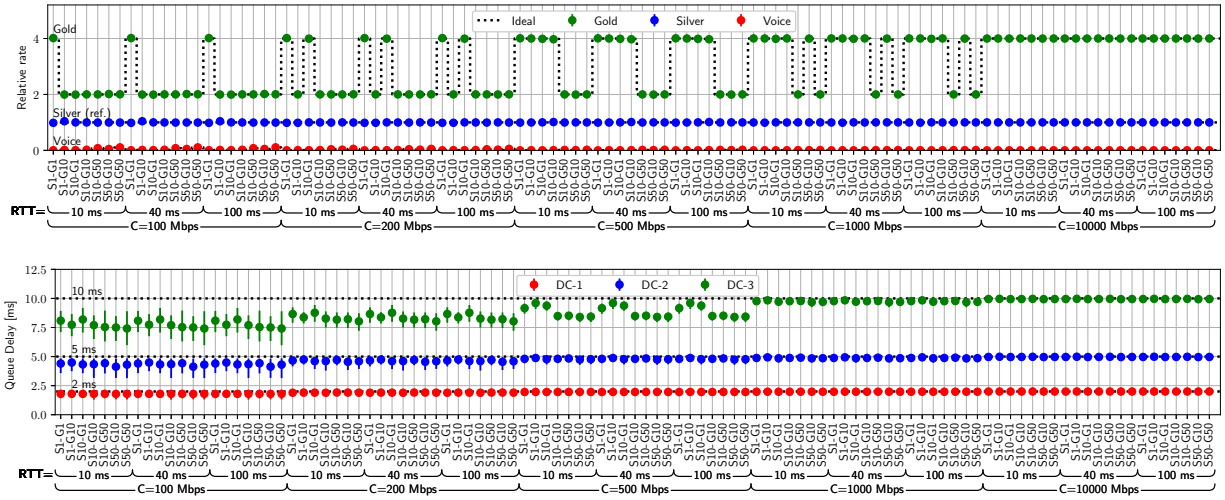


Fig. 7. Variable number of aggressive Gold (G) and Silver (S) UDP sources and 20 UDP-based Voice (V) flows with 10, 5 and 2 ms delay requirements, resp. The bottleneck capacity and the RTT in the system are both changed.

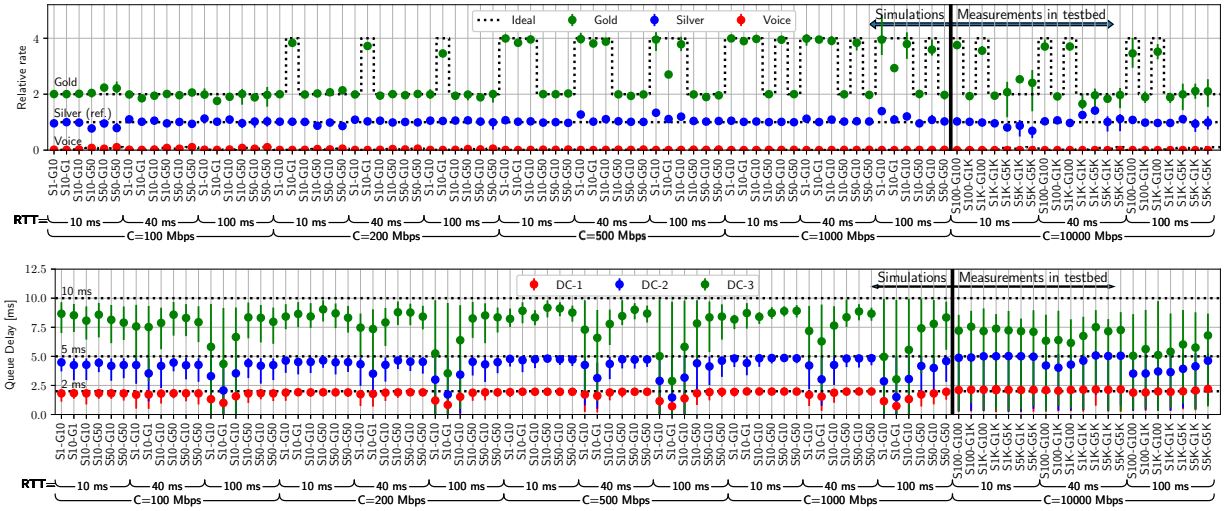


Fig. 8. Variable number of Gold (G) and Silver (S) TCP sources and 20 UDP-based Voice (V) flows with 10, 5 and 2 ms delay requirements, resp. The bottleneck capacity and the RTT in the system are both changed.

deviations can be seen for cases when the number of Gold and Silver users is inbalanced, such as S10-G1, S1K-G100 or S5K-G1K. This phenomenon is caused by the TCP congestion control. In congestion situations, the proposed scheduler starts dropping packets with PV below the actual CTV, resulting in that both Gold and Silver TCP flows experience packet losses when their sending rates go beyond a limit. However, Gold TCPs reach this limit at a higher throughput (2 or 4 times higher) than Silver flows, but it also means that the relative decrease is much higher for them than for Silver flows. Silver flows increase their relative rates much higher than the Gold ones and thus for small temporal periods they obtain a bit more throughput than their ideal. If there are 10 Silver flows and a single Gold one this extra throughput is shared among the Silver subscribers resulting in negligible overall increase and deviation from the Silver ideal as seen for cases: C $\geq$ 200 Mbps; RTT=100 ms; S10-G1. This behavior becomes more significant when the RTT is increasing (e.g., most visible for cases where RTT=100 ms and S100-G100 or S1K-G100 at 10 Gbps). If we have a sufficient number of subscribers in both traffic classes, an almost perfect match between the experienced and the ideal relative rates can be observed.

The bottom graph of Fig. 8 presents that all the delay requirements are met, but much higher delay variations can be experienced, especially for simulation cases with 100 ms RTT and limited number of Silver or Gold users, and for the testbed measurements in general. This is because the effect of a TCP congestion window decrease is most significant if there is only a few subscribes and they have different TVFs. The larger the RTT is, the smaller the average queueing delay we can observe. The large RTTs result in slower feedback loop in rate control, leading to much longer ramping up period after instant packet losses and generally high oscillations in the congestion window. For cases with high rate, high RTT and

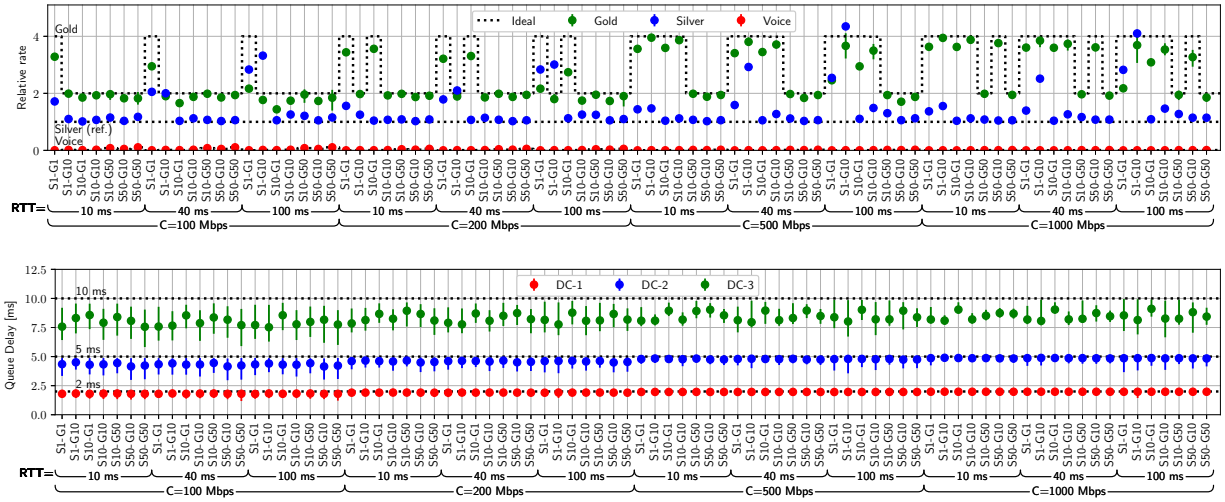


Fig. 9. Variable number of Gold (G) TCP and aggressive Silver (S) UDP sources and 20 UDP-based Voice (V) flows with 10, 5 and 2 ms delay requirements, resp. The bottleneck capacity and the RTT in the system are both changed.

small number of flows high per flow bandwidth-delay product can be experienced and the effect of congestion window reduction on the total buffer length becomes significant.

3) Coexistence of TCP and unresponsive UDP sources: In addition to the previous cases, we also conducted simulations with coexisting responsive TCP, and unresponsive and aggressive UDP sources. In this set of scenarios, we consider 20 permanent Voice flows with delay class 1 as unresponsive but low-rate background traffic, the number of Gold TCP (DC 3) and unresponsive Silver UDP (DC 2) is varied, similarly to the previous scenarios. We assume a single bottleneck in the network whose capacity is varied between 100, 200, 500 and 1000 Mbps. Fig. 9 depicts the average, 1st and 99th percentiles of relative flow rates (top graph) and the average, minimum and maximum of queueing delay (bottom graph) obtained for the different settings (84 simulation scenarios).

The top graph of Fig. 9 shows much larger deviations from the ideal relative rates than in the previous two static scenarios. The highest deviations can be seen for cases when a single subscriber is present in one of the RS classes, such as S1-G1 or S10-G1. Similarly to the previous case, this phenomenon is caused by the conservative and unscalable behavior of traditional TCP. As TCP reduces its rate, the freed resources are immediately occupied by the unresponsive UDP sources, while TCP flows need more time for increasing their throughput. As a result, UDP flows get more bandwidth. This is more significant if the number of TCP sources is limited.

The bottom graph of Fig. 9 shows low variance in the observed queueing delay. The buffer region of DC 2 is filled by the aggressive UDP sources, resulting in an average queueing delay close to the delay requirement. The only fluctuations can be seen for DC 3 (the delay class of TCP sources). However, the delay requirements are met for all the cases.

B. Dynamic Scenario

Fig. 10 demonstrates the performance of the proposed QoS framework under varying traffic intensity. The simulation time

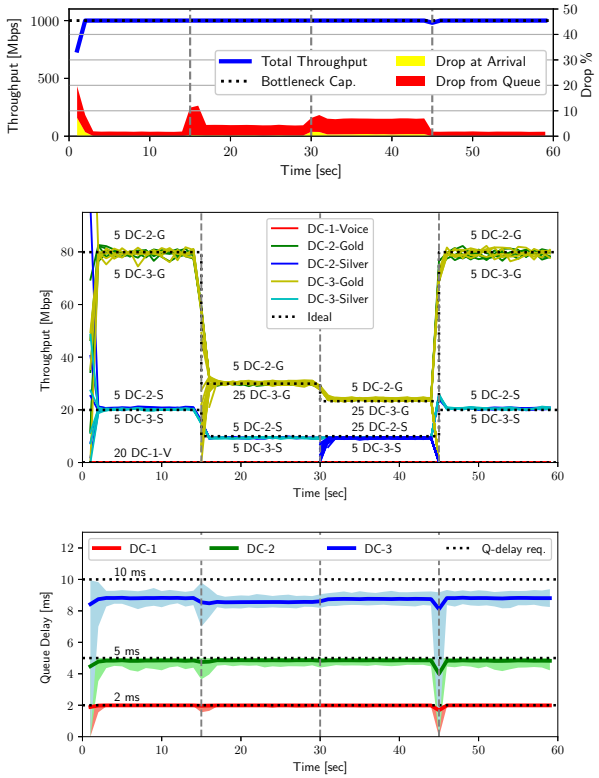


Fig. 10. Variable number of Gold (G) and Silver (S) TCP sources with variable delay requirements and 20 Voice background flows with DC-1=2 ms delay requirement. The bottleneck is 1 Gbps, while the RTT is 10 ms.

is split into four 15 sec long sessions: 1) The 20 Voice sources with delay class 1 start transmitting. 5-5 Gold TCP sources with delay classes 2 and 3, resp., and 5-5 Silver sources with DC 2 and 3 have also been launched. 2) 20 new Gold sources with DC 3 enter the system. 3) 20 new Silver sources with DC 2 arrive. 4) 20 Gold (DC 3) and 20 Silver (DC 2) sources stop the data transmission.

The middle figure shows the resource share among the different traffic sources. The ideal throughput values for the RS classes are denoted by dotted lines. An almost perfect match can be seen between the experienced and the calculated ideal curves while the convergence to the new ideal throughput in transient intervals is fast (the traffic mix changes every 15 s). The bottleneck link utilization and the drop rates are depicted in the top figure, resulting in full utilization and moderate drop rates. The bottom figure shows the observed queueing delay values for the different delay classes; the average values are marked by the solid lines, while the areas around each line represent the range between the observed minimum and maximum, using one second time resolution. One can observe that even if the number of sources significantly changes, small and temporal delay fluctuations can only be experienced.

C. Convergence time

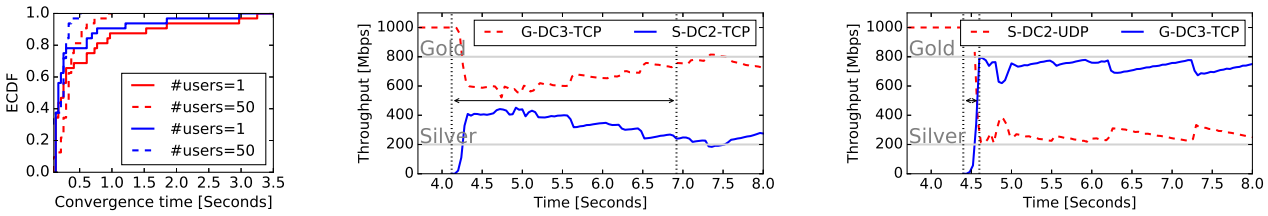
In this section, we analyze how fast our solution can enforce throughput for a newly arrived user. To this end, the convergence time as an additional performance indicator is introduced, quantifying the time elapsed from seeing the first packet of the new aggregate at the egress port of the Core Node to approaching the flow's ideal throughput, as shown in Fig. 11b-11c. In this scenario, we assume 1 or 50 permanent users sharing an 1 Gbps link. In a single simulation setting, the permanent flows are identical, being marked by the same operator policy (Silver or Gold TVFs), using the same transport protocol (UDP or TCP) and having the same delay class (DC2 or DC3). Note that all the possible combinations of these simulation settings are examined. Then a new aggregate consisting of 5 TCP connections arrives at the Core Node (its RS class (Gold, Silver) and delay class (DC2, DC3) are also varied). The rtt=10 ms and D_1=2 ms, D_2=5 ms and D_3=10 ms.

Let the convergence time with X% convergence range be the time elapsed from seeing the first packet of the new aggregate at the egress port of the Core Node to reaching the X% range around its ideal throughput. Fig. 11a shows the observed convergence times (64 different cases) for 5% and 10% convergence ranges, represented by Red and Blue curves, resp. One can observe that in the worst cases only a single permanent user is present in the system. With 10% range the convergence is always done within 2.7 seconds and for 90% of the cases the ideal range is reached in less than 1 second that is reasonable for production environments since large bandwidth traffic aggregates are mostly generated by data transfers. This scenario is depicted in Fig. 11b. The initial overshoot of the Silver is caused by the high initial PVs marked by the packet marker described in Sec. IV. This triggers congestion response from the Gold TCP source, taking time to ramp up its throughput again. In contrast, Fig.11c illustrates how the newly arrived TCP source can reach its ideal throughput in the presence of an unresponsive permanent traffic.

In addition, we have also carried out measurements in our testbed with 10 Gbps bottleneck capacity and 5-100ms emulated RTT. In contrast to the simulations, TCP sources are only investigated in two setups, with 100 and 1000 permanent TCP sources with the same RS class (Gold or Silver). In both cases a new TCP source (Gold or Silver) is launched 15 seconds after the start of the background sources. The throughput is only monitored in every 100 ms due to the limitation of the traffic generator tool. In all cases with 5ms RTT, the observed convergence time with 10% range of the newcomer traffic aggregate was less than or equals to 100 ms. When the RTT was 100 ms and 1000 permanent users were present, the convergence time was less than 300-400 ms. However, with 100 background users the ideal throughput of each aggregate was larger and thus the longer convergence time was needed. The longest convergence times were experienced when both the background and the new users had the same RS class: for Gold and Silver users the convergence time was around 10 and 5 seconds, resp. This increase was caused by the TCP behavior in environments with large RTT.

D. Web traffic

To examine the performance of the proposed delay-aware scheduler under more realistic traffic load, we have implemented a web traffic model in the NS-3 simulator similar to the one presented in [4]. The size of object to be downloaded is selected uniformly at random from four values with equal probabilities. Accordingly, we distinguish S, M, L and XL flows: 10 kB (S), 100 kB (M), 1 MB (L) and 10 MB (XL). The applied TVF is Silver or Gold with equal probability. Flows arrive according to a Poisson process with 5 ms average inter-arrival time. In addition, 50 long Silver TCP downloads assigned to delay class 2 are also in the system through the entire simulation, representing permanent background traffic. In all the scenarios the RTT is set to 40 ms. The web traffic scenario focuses on the following cases: 1) a Reference case with a Core Node using simple FIFO tail-drop queues (not aware of PVs or DCs) with 40 ms long buffer; 2) using the proposed delay-aware scheduler while web traffic is assigned to DC 1; 3) also using the proposed delay-aware scheduler while web traffic is assigned to DC 2. Simulations are run for 120 s, the bottleneck capacity is C=1 Gbps, while the delay requirements for DC 1 and 2 are D_1=5 ms and D_2=40 ms. Fig. 12 depicts the observed web traffic download times (also known as Flow Completition Time) for the different cases. The boxplots show the min, max, average (marked by triangle), median (horizontal black line) and interquartile ranges (box). The scaling of Y axis on the left (for S and M flows) and on the right (for L and XL flows) are different. The results for the case 1) are shown on both figures as "Ref". For case 2) (top fig.) one can observe that download times are always smaller and more consistent with the proposed method due to controlled resource sharing. The figure also shows that differentiation between Gold and Silver web users is only visible when the files are large enough, otherwise the download time is dominated by the RTT. For case 3) (bottom fig.), due to the higher queueing delay (the long flows maintain long queues), the web flows remain in slow start longer, therefore the difference between Gold and Silver users disappears for S, M and L flows. However, XL Gold flows are long enough to exploit the larger throughput they get from the shared resource. In general, Gold flows also experience much smaller delay



(a) Convergence times. (b) Slow convergence. (c) Fast convergence.

Fig. 11. Convergence after the arrival of a new traffic aggregate.

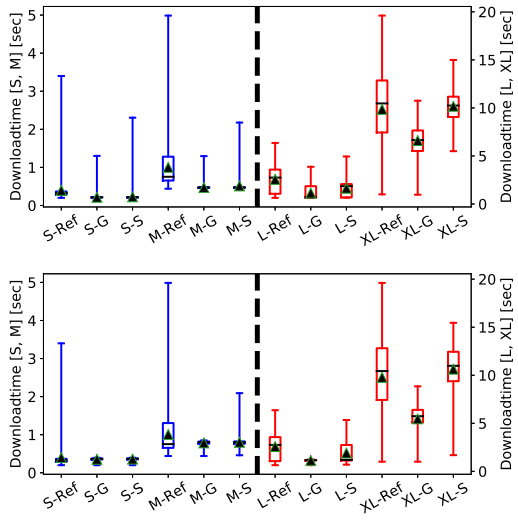


Fig. 12. Web download times. 2) top, 3) bottom.

variance than Silver ones. The download times, especially for smaller files, also increases due to the larger queueing delay. It is long known that short TCP transfers benefit from low RTT (to quickly open the window), while long downloads do better with larger buffer sizes (to keep traffic flowing in congestion avoidance). These conflicting requirements were not possible to be handled until now. The proposed scheme can provide different queueing delay for short and long TCP connections, while governing resource sharing between them.

VII. Conclusion

In this paper, we have presented a core-stateless QoS solution that handles resource sharing among traffic aggregates and also provides guarantees for per-hop delays. In contrast to prior work, the proposed method decouples delay and resource sharing requirements of traffic aggregates, enabling the definition of a rich-set of QoS policies. Policies are encoded into packet markings (packet value and delay class) assigned to each packet of the traffic aggregate. Core Nodes (e.g., router) solely use these two markings to decide how to handle packets to be forwarded. The method also facilitates the differentiation among sub-flows within a traffic aggregate, e.g., providing priorities to sub-classes and ensures bounded per-hop delay for packets. Its performance has been investigated considering both static and dynamic scenarios in throughout simulations as well as measurements with a high-performance DPDK-based prototype. Finally, we also demonstrated that proposed marker and scheduler can be used as a highly-scalable alternative to hierarchical WFQ in an existing vBNG product.

References

[1] I. Stoica *et al.*, "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.

[2] Z. Cao *et al.*, "Rainbow fair queueing: theory and applications," *Computer Networks*, vol. 47, no. 3, pp. 367 – 392, 2005.

[3] 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.

[4] K. Nagaraj *et al.*, "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.

[5] A. Kumar *et al.*, "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.

[6] O. Bondarenko *et al.*, "Ultra-low delay for all: Live experience, live analysis," in *Proceedings of the 7th International Conference on Multimedia Systems*, ser. MMSys '16. New York, NY, USA: ACM, 2016, pp. 33:1–33:4.

[7] S. Nádas *et al.*, "Per packet value: A practical concept for network resource sharing," in *IEEE Globecom 2016*, 2016.

[8] S. Laki *et al.*, "Take your own share of the pie," in *Proceedings of the Applied Networking Research Workshop*. ACM, 2017, pp. 27–32.

[9] S. Nádas *et al.*, "Towards a congestion control-independent core-stateless aqm," in *Proceedings of the Applied Networking Research Workshop*. ACM, 2018, pp. 84–90.

[10] B. Briscoe *et al.*, "Reducing internet latency: A survey of techniques and their merits," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2149–2196, 2016.

[11] J. S. M. Vallejo and C. E. Rothenberg, "Broadband network gateway implementation using a programmable data plane processor," Tech. Rep., 2017.

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

[13] Y. Yi and M. Chiang, "Stochastic network utility maximisation – a tribute to kelly's paper published in this journal a decade ago." *European Transactions on Telecommunications*, vol. 19, no. 4, pp. 421–442, 2008.

[14] B. K. Choi and R. Bettati, "Endpoint admission control: network based approach," in *Proceedings 21st International Conference on Distributed Computing Systems*, Apr 2001, pp. 227–235.

[15] E. Danna *et al.*, "A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 846–854.

[16] M. Chowdhury *et al.*, "Efficient coflow scheduling with varys," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 443–454.

[17] M. Carlson *et al.*, "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

[18] A. Sivaraman *et al.*, "Programmable packet scheduling at line rate," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 44–57.

[19] R. Mittal *et al.*, "Universal packet scheduling," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIV. New York, NY, USA: ACM, 2015, pp. 24:1–24:7.
[20] B. Zhang *et al.*, "A nearly optimal packet scheduling algorithm for input queued switches with deadline guarantees," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1548–1563, June 2015.
[21] M. Alizadeh *et al.*, "pfabric: Minimal near-optimal datacenter transport," in *Proceedings of ACM Sigcomm*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 435–446.
[22] R. J. La and V. Anantharam, "Utility-based rate control in the internet for elastic traffic," *IEEE/ACM Transactions On Networking*, vol. 10, no. 2, pp. 272–286, 2002.
[23] D. Papadimitriou *et al.*, "Open research issues in internet congestion control," Internet Requests for Comments, RFC Editor, RFC 6077, February 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6077.txt
[24] A. Espinal *et al.*, "Traffic model using a novel sniffer that ensures the user data privacy," in *MATEC Web of Conferences*, vol. 292. EDP Sciences, 2019, p. 03002.
[25] "The caida ucsd statistical information for the caida anonymized internet traces," Available: https://www.caida.org/data/passive/passive_trace_statistics.xml, 2020.
[26] "Data plane developement kit," Available: https://dpdk.org, 2018.
[27] "Dpdk, quality of service (qos) framework," Available: https://dpdk.org/doc/guides/prog_guide/qos_framework.html, 2018.
[28] "The ns3 simulator," Available: http://www.nsnam.org/, 2016.
[29] T. R. Henderson *et al.*, "Network simulations with the ns-3 simulator," in *In Sigcomm (Demo*, vol. 14, 2008.
[30] S. Ha *et al.*, "Cubic: A new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.
[31] H. Tazaki *et al.*, "Direct code execution: Revisiting library os architecture for reproducible network experiments," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 217–228.
[32] N. Kuhn *et al.*, "Characterization guidelines for active queue management (aqm)," Internet Requests for Comments, RFC Editor, RFC 7928, July 2016. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7928.txt

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.

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.

Ferenc Fejes achieved his MSc degree at University of Debrecen in 2018. Then he started his PhD studies at the Eötvös Loránd University in collaboration with the Ericsson Research, 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.

Péter Hudoba received the M.Sc. degree in computer science from Eötvös Loránd University, Budapest, Hungary in 2015, where he is currently pursuing the Ph.D. degree in security at the Department of Computer Algebra. He is currently a research assistant at Eötvös Loránd University and works on projects related to security, programmable data planes and complex system testing.

Zoltán Turányi received his M.Sc. in computer science from the Budapest University of Technology and Economics in 1996. He is currently an Expert at Ericsson Research. His background includes IP networking and QoS, mobility and mobile core networks, and SDN whereas his current research focus is on Cloud networking, execution environments, FaaS and programming models.

Zoltán Kiss received his M.Sc. degree in electrical engineering from the Budapest University of Technology and Economics in 2008, and joined Ericsson Research in 2016 as a Senior Researcher. His research interests include high performance networking, realtime systems and operating system network stacks.

Csaba Keszei graduated from the Technical University of Budapest in 2000. Right after it he joined Ericsson Research Hungary, worked on IP mobility, mobile core network and high performance routing data plane implementations. By 2014 he re-located to San Jose, California to develop advanced traffic engineering solutions for virtualized data plane. Today he works at Arrcus Inc., a group of networking industry veterans providing software-powered network transformation for the interconnected world.