Enabling Rank-Based P4 Programmable Schedulers: Requirements, Implementation, and Evaluation on BMv2 Switches

Mostafa Elbediwy, Bill Pontikakis, Jean-Pierre David, *Member, IEEE*, and Yvon Savaria, *Life Fellow, IEEE*

***Abstract*— Software-defined networking (SDN) has revolutionized network infrastructure, offering programmability to meet evolving network demands. However, the fixed-function nature of the packet scheduler in current network equipment impedes the exploration of scheduling policies within a programmable network environment. This paper proposes a novel methodology to implement rank-based programmable schedulers in programmable BMv2 switches expressed with the network-specific programming language (P4). A proposed custom networking environment facilitates the study and evaluation of various scheduling policies. This environment is used to implement 20 different scheduling and shaping policies to identify the required language constructs and components needed to express these policies with the P4 language efficiently. Our experiments reveal that specific scheduling policies do not seamlessly align with a previously proposed architecture for rank-based scheduling policies. Thus, we propose rank-based versions for five previously reported scheduling policies, making them efficiently implementable in any rank-based schedulers and programmable network equipment. The reported results confirm that the rank-based versions of these scheduling policies accurately replicate the behavior and performance of the original policies, with a maximum error of 0.5% in the resulting flow completion times (FCTs).**

***Index Terms*— Programmable data plane, scheduling algorithms, shaping algorithms, P4, BMv2.**

I. Introduction

QUALITY of Service (QoS) is critical when optimizing network traffic management and improving overall performance for high-speed networks. Various scheduling policies have been proposed to meet diverse QoS requirements, addressing factors such as latency, fairness, rate-limiting, and flow completion time. These policies differentiate services for network users and traffic types based on their specific QoS needs. As network conditions evolve, flexibility in selecting scheduling policies becomes essential for network operators.

A dedicated packet scheduler suitably embedded inside network switches is crucial to address this need. This scheduler should be sufficiently programmable to sort and schedule packets based on the selected policies. Recently, there have been proposals for programmable packet schedulers intended for integration into programmable hardware data planes. Many of these schedulers draw inspiration from the primitives offered by Push-In-First-Out (PIFO) [1], a mechanism that organizes packets based on their predefined ranks. In these rank-based schedulers, ranks can be computed in an earlier processing stage, such as a programmable ingress pipeline [1], [2], [3].

This work explores the compatibility between various scheduling policies, network programming languages (such as P4 [4]), and the aforementioned rank-based schedulers. We have implemented and will present pseudo-codes for over 20 distinct scheduling policies, demonstrating their compatibility with the previously proposed rank-based schedulers. While most of these policies naturally align with the programmable schedulers, a few lack a compatible rank-based version. Thus, we introduce rank-based versions for five policies that either replicate the behavior of the original algorithms or enhance it.

To implement the studied scheduling policies, it is essential to introduce a programmable scheduler into a programmable data plane. We propose a methodology for implementing a programmable rank-based scheduler in the BMv2 target switches, compatible with the P4 language. This method employs a generic layout, facilitating its adoption by various rank-based programmable schedulers, including the PIFO [1], the PIEO [2], and the DR-PIFO [5].

By utilizing a rank-based packet scheduler [5], we implement all the studied scheduling policies using a BMv2 target switch programmed with the P4 language. These implementations highlight the minimal requirements and components necessary to implement each policy. Moreover, the proposed rank-based versions undergo evaluation by comparing their behavior and performance against the corresponding original policies. This assessment uses two distinct testbeds: one featuring a single BMv2 switch, and the other consisting of a data-center topology comprising 13 BMv2 switches.

This work presents the following key contributions:

- **Characterization of scheduling schemes:** Various scheduling and shaping policies are analyzed to determine the minimum requirements for implementing them in P4 programmable switches.
- **Rank-based scheduling:** We propose rank-based versions of five scheduling policies: (P1-RB) rank-based DRR [6], (P2-RB) rank-based WRR [7], (P3-RB) rank-based WDRR [8], (P4-RB) Fair-Slytherin [9] and (P5-RB) rank-based SP-non-Work-Conserving, to be compatible with P4 switches comprising rank-based pro-

Received 1 December 2023; revised 8 September 2024; accepted 11 October 2024; approved by IEEE TRANSACTIONS ON NETWORKING Editor K. Park. Date of publication 29 October 2024; date of current version 14 February 2025. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada, in part by Prompt Quebec, in part by Intel, and in part by Noviflow. *(Corresponding author: Mostafa Elbediwy.)*

The authors are with the Department of Electrical Engineering, Polytechnique Montréal, Montreal, QC H3T 1J4, Canada (e-mail: mostafa.elbediwy@polymtl.ca).

Digital Object Identifier 10.1109/TNET.2024.3481152

grammable schedulers designed to support rank-based scheduling policies.

- **Implementations on BMv2 switches programmed with P4:** A generic method is proposed to seamlessly integrate any rank-based programmable scheduler with BMv2 software switches, programmed using the P4 language. Subsequently, this method is employed to implement all the studied policies.

II. Background

A. Scheduling Policies

Scheduling algorithms are crucial in network communication as they manage three primary constraints: bandwidth, promptness, and buffer space. These constraints govern the order of packet transmission, the specific transmission times, and their impact on latency and potential packet discarding, respectively [10].

1) Impact on Network Parameters: Within these constraints, bandwidth significantly influences the order of packet transmission, particularly in ordered queues. The promptness constraint dictates when packets are transmitted, while the buffer space constraint affects both latency and determining which packets may be discarded. The allocation of resources by scheduling algorithms directly influences critical network parameters such as 1- throughput, representing the rate of successful message delivery; 2- latency, the time delay between the initiation and completion of a data transfer; and 3- the loss rate, indicating the proportion of transmitted packets that do not reach their destination.

2) Classification of Scheduling Policies: Scheduling algorithms can be broadly classified as either work-conserving or non-work-conserving. Work-conserving scheduling ensures continuous device activity, offering 100% resource utilization and maximum throughput. On the other hand, non-work-conserving scheduling may lead to device idle periods, as each packet is assigned an eligibility time. This type of scheduling is primarily used for rate control and traffic shaping, potentially increasing average delays. Examples of work conserving algorithms include Delay Earliest-Due-Date (Delay-EDD) [11], Virtual Clock [12], Weighted Fair Queueing (WFQ) [13], and Worst-Case Weighted Fair Queueing (WF^2Q) [14]. Non-work conserving algorithms include Stop-and-Go [15], Jitter Earliest-Due-Date (Jitter-EDD) [16], Hierarchical Round Robin (HRR) [17], and Rate-Controlled Static Priority (RCSP) [18].

3) Universal Packet Scheduling (UPS) Exploration: In an attempt to find a universal packet scheduling (UPS) algorithm, authors in [19] aimed to represent all possible scheduling algorithms. A universal algorithm would satisfy multiple objectives, such as latency bounds and fairness, matching the performance of the best-known algorithm for a given objective. However, it was proven that a UPS algorithm does not exist for multi-hop networks with three or more nodes. Priority queues cover single nodes, and the Least Slack Time First (LSTF) algorithm is the closest to universality, meeting the stated objectives [19].

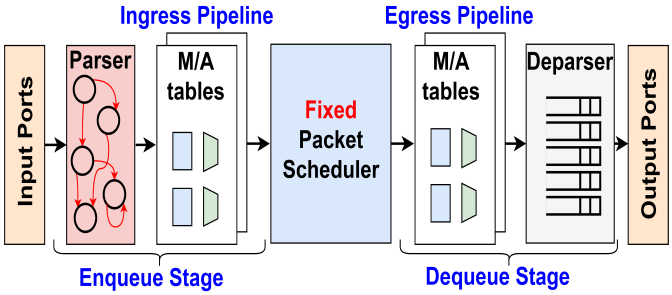

Fig. 1. V1model: P4 switch architecture [28].

B. Rank-Based Packet Schedulers

Recently, there have been various proposals for programmable packet schedulers aimed at meeting the demand for expressing a wide variety of scheduling policies [1], [2], [3], [5], [20], [21], [22], [23], [24], [25]. These schedulers share a common concept of decomposing the scheduling problem into two stages: computing the rank value of each packet and then sorting packets based on their ranks. Thus, these packet schedulers have two key components: a programmable unit for rank computations and a fixed unit for scheduling packets. This is advantageous since the two units have distinct characteristics. Another component that exists only in the Dynamic Ranking Push-In-First-Out (DR-PIFO) [5], [26] is the re-ranking unit. This unit provides an optional adaptive ranking mechanism needed by some scheduling policies such as the pFabric [27].

The computational unit in packet schedulers must handle one packet at a time, allowing for relaxed timing constraints similar to other programmable components of the network switch. This processing can also be executed within the ingress pipeline [1], [3], [5]. On the other hand, the scheduling unit faces the challenge of ordering and scheduling all active flows and packets simultaneously. Therefore, implementing this unit with a fixed hardware architecture, following the ingress pipeline, is desirable to achieve the required timing performance. Scheduling and shaping policies should be expressed with suitable algorithms to fit this proposed rank-based scheduler model. Consequently, these algorithms should efficiently compute packet ranks, which are subsequently utilized by the scheduler to sort and schedule packets. However, specific policies, such as Deficit Round Robin (DRR) [6], lack rank-based versions compatible with these programmable rank-based schedulers.

C. Implementation on Programmable Data Planes

The current programmable data plane uses fixed-function schedulers, which consist of parallel limited-size strict priority queues. However, these schedulers impose constraints due to their lack of flexibility to adapt dynamically to changing network conditions. BMv2 target software switches, an example of a programmable data plane, are designed using P4-compatible architectures. Fig. 1 presents a simple P4 switch architecture, where most components of the target switch, except for the packet scheduler, can be programmed using P4 to control functionality efficiently.

Despite the programmability offered by P4 and the BMv2 model, neither provides an abstraction for a programmable

TABLE I
LIST OF THE 20 SCHEDULING POLICIES IMPLEMENTED AND CHARACTERIZED IN OUR STUDY

Name	Rank-based	Proposed Version
P1) Deficit Round Robin [6]	No	rank-based DRR **(P1-RB)**
P2) Weighted Round Robin [7]	No	rank-based WRR **(P2-RB)**
P3) Weighted Deficit Round Robin [8] [29]	No	rank-based WDRR **(P3-RB)**
P4) Slytherin [9]	Yes (approximated)	Fair-Slytherin **(P4-RB)**
P5) Rate-Limited-SP-non-Work-Conserving	No	rank-based SP-non-WC **(P5-RB)**
P6) Least Attained Service [30]	Yes	
P7) pFabric [27]	Yes	
P8) Weighted Fair Queuing [31]	Yes	
P9) Least Slack Time First [32]	Yes	
P10) Least Attained Recent Service [33]	Yes	
P11) Stop-and-Go [34]	Yes	
P12) Rate Controlled Service Disciplines [35]	Yes	
P13) Window Constrained Scheduling [36]	Yes	
P14) Approximate Fair Queueing [37]	Yes	
P15) Hierarchical (ex: FIFO-WFQ-SP)	Yes	
P16) Penalize Heavy Hitters [38]	Yes	
P17) WFQ - Queue Occupancy [39]	Yes	
P18) Rate-Limited-SP-Work-Conserving	Yes	
P19) Numfabric [40]	Yes	
P20) Independent Scheduling/Shaping Policies	Yes	

packet scheduler. Moreover, none of the previously proposed rank-based programmable schedulers has fully defined the key components necessary for implementing rank computations across diverse scheduling policies. Identifying these components is essential to understanding the expressiveness limits of scheduling models, network programming languages, and target switches.

III. CHARACTERIZATION OF SCHEDULING ALGORITHMS

A diverse range of scheduling policies previously proposed to manage the network traffic exists. Each policy aims to meet a specific Quality of Service (QoS) requirement, such as end-to-end latency, fairness, or flow completion time [40]. In this section, we illustrate the compatibility of various scheduling policies from the literature with rank-based schedulers. Our study comprehensively examines 20 scheduling policies, listed in Table I, to identify the necessary conditions for implementing each policy. These policies can be classified into two main categories: Work-Conserving Scheduling (WCS) and Non-Work-Conserving Scheduling (NWCS). Moreover, scheduling schemes can also involve hierarchical compositions of different policies.

As illustrated in Table I, four of the 20 selected scheduling policies do not have a naturally compatible rank-based algorithm. Additionally, the Slytherin scheduling policy only has an approximate rank-based version, which seems to provide an unfair service to its users. Accordingly, we propose a rank-based version for five scheduling policies: (P1) Deficit Round Robin (DRR), (P2) Weighted Round Robin (WRR), (P3) Weighted Deficit Round Robin (WDRR), (P4) Slytherin and (P5) Rate-limited Strict Priority non-Work-Conserving. These proposed rank-based versions either mimic the behavior of the original algorithms or improve their performance.

As illustrated in Fig. 1, the packet scheduler is situated between the ingress and egress pipelines, making them the packet enqueue and dequeue stages, respectively. The fixed packet scheduler shown in Fig. 1 should be replaced by a rank-based programmable scheduler, using the methodology proposed in Section IV. In the following algorithms, the computations performed during packet enqueue occur inside the ingress pipeline, while those required at packet dequeue are carried out inside the egress pipeline. Additionally, the results of these computations, such as the rank of each packet, are passed to the rank-based scheduler to sort and schedule network flow packets properly.

Later in this section, we provide pseudo-codes for the five proposed rank-based algorithms, which calculate three key packet attributes: rank, eligible dequeue time, and, if necessary, the updated rank of their flows. The scheduler uses these attributes to order packets and flows appropriately. Furthermore, we provide[1] pseudo-codes for the 20 selected scheduling algorithms, along with P4 programs implementing them with the aid of a rank-based scheduler [5] and BMv2 target switches. In all the following pseudo-codes, it is important to note that computations expressed outside enqueue or dequeue events should be performed only once at program compilation time and are not executed at runtime, unlike those triggered by enqueue or dequeue events. Therefore, these computations are not implemented in the ingress/egress pipelines within the data plane, as they are part of the P4 switch configuration or the P4 control plane.

A. Rank-Based Deficit Round Robin (P1-RB)

Scheduling policies for the Work-Conserving Scheduling (WCS) class schedule packets whenever the output link is available. These policies aim to achieve maximum throughput from the utilized switch. An example from this class is Deficit Round Robin (DRR) [6], which aims to achieve optimum fairness among multiple flows with various packet sizes. DRR gives each flow an equal opportunity to send up to one Quantum (Q) of bytes in each sending round. If a packet size exceeds the quantum, a deficit counter is incremented by Q for that round. The packet is sent once the deficit counter equals or exceeds the packet size. As a result, each sending round may involve multiple dequeue operations. This round-robin scheduling mechanism aims to send packets whenever the output link is idle. However, in rank-based schedulers, scheduling decisions must be made earlier to meet the hardware timing constraints of the scheduler [1]. Therefore, a new variant of the DRR policy is required, where the ranking of packets or flows is determined before the dequeue stage.

This work introduces a novel rank-based variant of the Deficit Round Robin (DRR) policy designed to align with

[1]The pseudo-codes, P4 programs, and implementations of the 20 selected scheduling and shaping policies are available at https://github.com/Elbediwy/DR-PIFO_20_policies

Algorithm 1 Rank-Based DRR (P1-RB)

Input: P.flow_id, P.length
Output: P.rank
Config. Parameters: N, Q, minimum_packet_size
1 /* "N": the number of the supported flows "srv_cntr": array to count the service received by each flow. "Q": "Quantum" the maximum number of bytes, that a flow can send every round. "deq_cycle": a number indicates the current dequeue cycle.*/
2 pkts_per_rnd = N * (Q / minimum_packet_size); // Maximum number of packets the scheduler can send during one round. **At Enqueue of each new packet (P):**
3 srv_cntr[P.flow_id] = **max**(srv_cntr[P.flow_id], deq_cycle * Q);
4 srv_cntr[P.flow_id] **+=** P.length;
5 virtual_round_id $= \lfloor (srv\_cntr[P.flow\_id] - 1)/Q \rfloor$;
6 P.rank = P.flow_id + (pkts_per_rnd * virtual_round_id);
7 P.round_id = virtual_round_id; **At Dequeue of a scheduled packet (P):**
8 deq_cycle = **max**(deq_cycle, P.round_id)

rank-based programmable schedulers. Unlike the original DRR algorithm, which determines the departure order of packets during each dequeue operation, the rank-based DRR approach assigns appropriate ranks to packets at the enqueue stage. The fixed scheduler then uses these ranks to determine the departure order of packets.

As explained, rank-based schedulers rank packets during the enqueue stage, whereas the actual sending rounds in the original DRR occur at the dequeue stage. The proposed rank-based DRR introduces the concept of virtual sending rounds. Each flow is allocated a transmission quota, denoted Q, divided into equal-sized virtual slots, with one slot assigned to each flow. During each virtual round, a flow can transmit several packets equal to Q divided by the packet size, using the minimum packet size as a reference. If a flow's service requirement exceeds the allocated quota, it will be prevented from sending more packets in that virtual round.

The pseudo-code for the rank-based DRR scheduling algorithm is outlined in Algorithm 1. The rank of each new packet is calculated according to multiple steps:

1) When a packet is enqueued from a specific flow, the "service counter" register of that flow is incremented by the packet size (line 4).
2) To prevent inactive flows from unfairly dominating the output link over active flows, the current service counter is determined based on the equation in line 3.
3) The appropriate virtual round ID is calculated based on the flow's service counter using the equation (line 5). To handle cases where the "service counter" is exactly equal to Q or multiples of Q, 1 byte is subtracted from the "service counter" to ensure correct rounding. For example, if the service counter is either 1500 or 3000 bytes, and Q is 3000, the virtual round ID should be equal to 0 in both cases.
4) Finally, the rank of each packet is determined using the equation stated in line 6. The multiplication in this equation determines the offset of the ranks within the specific round ID. Then, the product is added to the "Flow ID" since DRR prioritizes lower flow IDs within the same sending round.

It is important to note that packets originating from a specific virtual round will always have rank values lower than the ranks of packets from subsequent virtual rounds.

Afterward, the packet scheduler uses the rank of each new packet to sort the packets in ascending order. When a packet is scheduled, its round ID is then used to determine the current dequeue cycle (line 8), which provides feedback for the enqueue computations. Formerly inactive flows utilize this current dequeue cycle to calculate the virtual sending round ID correctly. The enqueue calculations can be implemented within the ingress pipelines of the data plane using a network programming language like P4. In contrast, the dequeue calculations can be executed within the egress pipelines.

B. Rank-Based Weighted Round Robin (P2-RB)

The Weighted Round-Robin (WRR) scheduling algorithm belongs to the round-robin scheduling class, similar to DRR. However, WRR differs by allocating the output link's bandwidth among active flows in proportion to their assigned weights [7]. These weights, set by the network operator, dictate the maximum number of packets that an active flow can send during a single sending round.

Here, we introduce a novel rank-based version of the Weighted Round-Robin (WRR) algorithm, designed to be efficiently expressed in the P4 language with the assistance of a rank-based scheduler. The pseudo-code for the rank-based WRR is outlined in Algorithm 2, presenting the necessary steps to execute the enqueue and dequeue operations of the WRR policy.

While the Deficit Round Robin (DRR) policy operates with a level of service measured in bytes, WRR deals with bulk service, representing the number of sent packets. This is a fundamental distinction between the rank-based versions of WRR and DRR. As a result, many steps in the WRR policy, presented in Algorithm 2, closely resemble those in the rank-based DRR illustrated in Algorithm 1.

In the rank-based WRR algorithm, the weight assigned to each flow determines the number of packets a flow can send during a single round. Thus, as demonstrated in Algorithm 2, the maximum possible number of transmitted packets per virtual round is the summation of the weights of all flows (line 2). The "round ID" of a flow is incremented each time the "service counter" reaches the weight of that flow (line 6). This mechanism ensures that flows with higher weights are prioritized in receiving service and allocated bandwidth proportionally based on their assigned weights.

Like the rank-based DRR approach, the current "dequeue cycle" is also updated during the dequeue operation in the rank-based WRR. This updated dequeue cycle serves as crucial feedback in the rank computation process, effectively preventing inactive flows from unfairly dominating the output link when they become active.

Algorithm 2 Rank-Based WRR (P2-RB)

Input: P.flow_id
Output: P.rank
Config. Parameters: N, W
/* "srv_cntr": array of the service received by each flow. "W": array of "Weights", maximum number of packets, that flows can send every round. */
pkts_per_round = $\sum_{i=0}^{i<N}$ W[n]; **At Enqueue of each new packet (P):**
round_id[P.flow_id] = **max**(round_id[P.flow_id], deq_cycle);
P.rank = P.flow_id + (pkts_per_round * round_id);
++ srv_cntr[P.flow_id];
if *srv_cntr[P.flow_id] == W[P.flow_id]* **then**
    srv_cntr[P.flow_id] = 0;
    ++ round_id[P.flow_id];
P.round_id = round_id[P.flow_id]; **At Dequeue of a scheduled packet (P):**
deq_cycle = **max**(deq_cycle, P.round_id)

Algorithm 3 Rank-Based WDRR (P3-RB)

Input: P.flow_id, P.length
Output: P.rank
Config. Parameters: N, Q, minimum_packet_size
pkts_per_rnd = $\sum_{i=0}^{i<N}$ Q[n] / minimum_packet_size;**At Enqueue of each new packet (P):**
srv_cntr[P.flow_id] = **max**(srv_cntr[P.flow_id], deq_cycle * Q[P.flow_id])
srv_cntr[P.flow_id] **+=** P.length;
virtual_round_id[P.flow_id] = $\lfloor (srv\_cntr[P.flow\_id] - 1)/Q[P.flow\_id] \rfloor$;
P.rank = P.flow_id + (pkts_per_rnd * virtual_round_id);
P.round_id = virtual_round_id; **At Dequeue of a scheduled packet (P):**
deq_cycle = **max**(deq_cycle, P.round_id)

C. Rank-Based Weighted Deficit Round Robin (P3-RB)

The Weighted Deficit Round Robin (WDRR) algorithm is an enhanced version of the WRR algorithm. Unlike conventional WRR, WDRR assigns different weights to flows and provides distinct quantum values for each queue or flow [8], [29]. These quantum values are utilized to distribute the output link's bandwidth among flows in proportion to their respective weights, enabling effective resource sharing. One distinguishing feature of WDRR compared to WRR is its ability to handle different packet sizes, as it operates in terms of bytes rather than packets.

We introduce a novel rank-based variant of the WDRR scheduling algorithm. Our approach involves adapting the proposed rank-based version of DRR to accommodate an array of distinct "Quantum" values, with each value assigned to a specific flow. This modification enables our rank-based WDRR to handle multiple flows with varying service requirements efficiently. To clearly understand our proposed method, we present the pseudo-code for the rank-based WDRR in Algorithm 3.

D. Fair Slytherin (P4-RB)

In [9], the authors introduced a novel packet scheduling scheme to prioritize packets that have encountered queueing delays in the previous network hops. This approach is known as the Slytherin scheduling algorithm, which leverages the Explicit Congestion Notification (ECN) header field. ECN is a 2-bit header field utilized to notify corresponding hosts of network congestion. When the queue occupancy surpasses a predetermined threshold, a network switch modifies a packet's ECN bits to "Congestion Experienced" (CE), represented by "11". In the Slytherin algorithm, packets marked with CE bits are directed to higher-priority queues for expedited processing. The Slytherin algorithm's implementation has significantly improved the Flow Completion Time (FCT), particularly benefiting short flows [9].

In [9], the proposed scheduling scheme was simplified and approximated to develop the Slytherin algorithm in an implementable form on the available network switches. However, with the advent of rank-based programmable schedulers, which are capable of precise implementation, we recognize the potential for a more accurate approach.

Thus, we propose an enhanced version of the Slytherin algorithm, called fair-Slytherin, outlined in Algorithm 4. Fair-Slytherin employs a novel approach to determine the rank of each enqueued packet. For packets entering the first network hop in their path to the destination host, their ranks are equal to their arrival times (line 3). By contrast, for the other packets, the ranks are calculated based on their queueing delays in the previous network hops (line 5). A packet's queueing delay is computed when it is dequeued from the current hop (line 8). If the packet experiences a higher delay at the current hop compared to previous hops, this computed delay is attached to the dequeued packet (line 10) to be used by the next network hop.

By integrating this refined approach into the scheduling process, fair-Slytherin offers more accurate packet classification and prioritizes packets based on the actual delays experienced throughout their network journey. This, in turn, is anticipated to enhance overall network performance, particularly in scenarios where reduced latency and fair distribution of resources are critical factors.

E. Rank-Based Strict Priority Non-Work-Conserving (P5-RB)

The Strict Priority (SP) scheduling scheme ensures that packets with higher priority are scheduled before those with lower priority, which aligns with the principles of the rank-based programmable schedulers [1]. However, a drawback of this scheduling scheme is the potential starvation of lower-priority flows, which may not receive adequate service. To address this issue, a well-established solution involves limiting the scheduling rate of higher-priority flows.

Algorithm 4 Fair Slytherin (P4-RB)

Input: P.arrival_time, P.previous_hop_delay
Output: P.rank
Config. Parameters: Maximum_possible_delay_value
1 **At Enqueue of each new packet (P):**
2 **if** *P.previous_hop_delay == 0* **then**
3 P.rank = P.arrival_time;
4 **else**
5 P.rank = Maximum_possible_delay_value -
6 P.previous_hop_delay;
7 **At Dequeue of each scheduled packet (P):**
8 hop_delay = (P.departure_time - P.arrival_time);
9 **if** *hop_delay > P.previous_hop_delay* **then**
10 P.previous_hop_delay = hop_delay;

Algorithm 5 Rank-Based SP-Non-WC (P5-RB)

Input: P.flow_id, P.length, P.ToS
Output: P.rank, P.eligible_time
Config. Parameters: max_bytes, window_period
1 window_ID = **array**[N]; // the current window ID of each flow.
2 max_bytes = **array**[N]; // array contains the maximum number of bytes that each flow can send during a single time window. **At Enqueue of each new packet (P):**
3 P.rank = P.ToS;
4 received_bytes[P.flow_id] += P.length;
5 P.eligible_time = window_ID[P.flow_id] * window_period;
6 **if** *(received_bytes[P.flow_ID] >= max_bytes[P.flow_ID])* **then**
7 receive_bytes[P.flow_ID] = 0;
8 ++ window_ID[P.flow_id];

This rate-limiting solution is applied using the Non-Work-Conserving Scheduling (NWCS) primitive alongside SP. Policies belonging to the NWCS category schedule packets or flows as soon as they become eligible. Unlike work-conserving scheduling, the output link may remain idle even if packets wait in the queues, but none of them are eligible yet. Combining these techniques prevents rate-limited flows from surpassing their assigned bandwidth, even if the output link is idle. This approach balances between ensuring a priority-based service for high-priority packets and mitigating the risk of neglecting lower-priority flows.

We propose a rank-based variant of the SP-Non-Work-Conserving scheduling scheme, achieved by introducing an assigned "eligible time" for each packet. The pseudo-code for this rank-based version is provided in Algorithm 5. This algorithm incorporates information about the desired bandwidth of each flow and the duration of the time window to determine the maximum number of bytes a flow can be transmitted in a single time window. These predetermined eligible byte values are stored in the "max bytes" array (line 2).

The rank of each packet is determined based on its assigned Type of Service (ToS) (line 3). During runtime, when a new packet is enqueued, the "received bytes" counter of its corresponding flow is incremented by the length of the packet (line 4). Subsequently, the eligible time for the packet is calculated based on the current "window id" of its flow (line 5). The algorithm then compares the "received bytes" with the eligible bytes (line 6). If the received bytes reach or exceed the eligible bytes, they are reset (line 7), and the "window id" is incremented (line 8). The packet is then stored in the implemented rank-based scheduler to ensure proper scheduling. The scheduler guarantees that all packets are sorted based on their ranks, and no packet is scheduled before its eligible time.

IV. Implementation in Programmable Data Plane

This section outlines the necessary steps to introduce a rank-based programmable packet scheduler as an extern to the P4 language. Since they are architecture-specific, extern objects have their behavior implemented in a target switch and are manipulated by the P4 language through well-defined interfaces. The Behavioral Model version 2 (BMv2) [41] is an open-source software switch specifically designed to serve as a target for P4 programs. BMv2 supports introducing new software externs [42], making it the ideal model for our implementation.

As discussed in Section II-B, a rank-based programmable scheduler typically consists of two main components: the rank computation unit and the scheduling unit. We introduce the scheduling unit as an extern, which we will refer to as the "scheduling extern". On the other hand, the rank computation process is expressed as a standard P4 program, utilizing the available P4 components and incorporating some newly introduced arithmetic operations. To be compatible with the current P4 language, these new arithmetic operations are also introduced as externs, and we will refer to them as "computational externs". Our approach for introducing new externs aligns with the methodology proposed in [42]. For both the scheduling and the computational externs, we have designed interfaces that allow the packet scheduler to interact with other components in the switch's architecture seamlessly. Moreover, these interfaces can be leveraged by any other rank-based programmable scheduler.

A. Implementation of the Scheduling Unit (Scheduling Extern)

The source code for the P4 compiler and the BMv2 are written in C++. To ensure optimal compatibility, the software model of the target scheduler should be expressed in the C++ language too. For the sake of the experiment, we have implemented a software extern of a packet scheduler called DR-PIFO [5] using the extern template presented in [42]. However, this implementation process is not restricted to the DR-PIFO scheduler, since any other rank-based scheduler can be deployed instead.

Due to the nature of the P4 language, an extern must be called from inside one of the standard entities, such as the

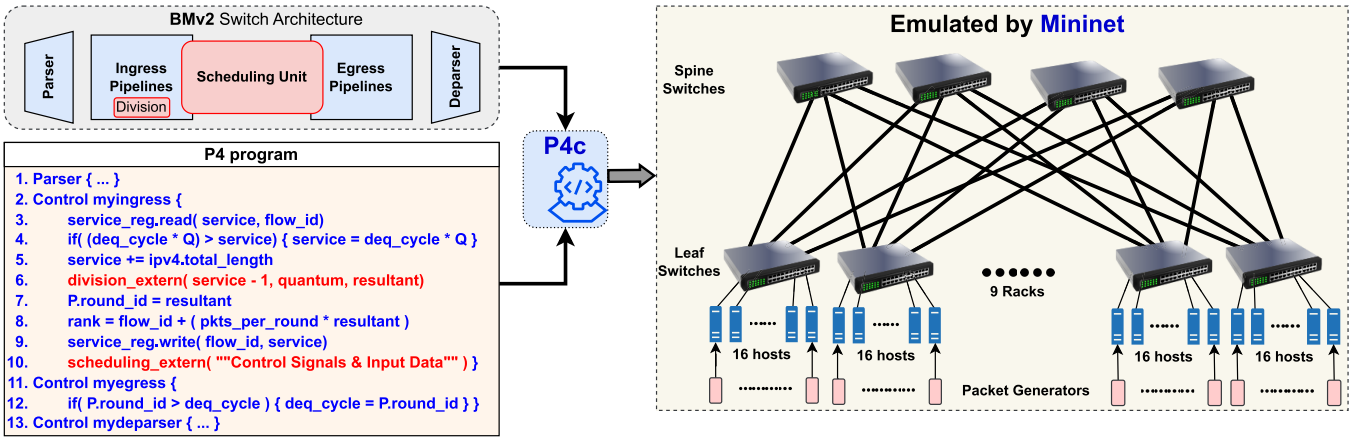

Fig. 2. Illustrative implementation of the rank-based DRR (P1-RB) Algorithm 1, using a BMv2 switch and a P4 program in conjunction with a rank-based packet scheduler. The P4 compiler (P4c) is used to program the network switches in an emulated environment by Mininet. The red color indicates newly introduced components in P4 and BMv2.

ingress pipeline. In our implementation, the scheduling extern is called at the end of the ingress pipeline to mimic the Portable Switch Architecture (PSA) [28]. Then, the scheduling extern is now recognized by the P4 language, enabling the P4 programmers to exploit it in any P4 program. As shown in Fig. 2, in the P4 program (line 10), the "scheduling extern" is called by every new packet processed by the ingress pipeline.

Although the P4 language recognizes the scheduling extern, it cannot function correctly in the switch since it needs to hold and reorder packets, which is not naturally allowed by the P4 externs or the switch architecture. Thus, as depicted in Fig. 2, the scheduling unit is introduced as a module inside the BMv2 switch architecture. We modified the switch's C++ source code to introduce the scheduler software model. In this implementation, the scheduling extern acts as the interface of the scheduler module with the P4 program. Then, the scheduler module in BMv2 receives the appropriate inputs with the authority to schedule packets accordingly in the underlying target BMv2 switch. As a result. The scheduler software module is responsible for storing and ranking packets and is exclusively accessible for insertion via the ingress pipeline.

B. Implementation of the Priority Calculations (Computational Externs)

In addition to the scheduling extern, the rank computation process should be expressed in the P4 language to properly apply the desired scheduling algorithms as illustrated in section III. While most functions and components required for rank computation are available in P4 and BMv2 switches, two arithmetic operations are missing from the P4 language. These operations include a run-time division operation, which some scheduling algorithms need. Another missing operation is the multiplication of an integer by a fractional value, which is not supported in P4 due to the absence of fractional value representation. To address this issue, we introduce these two operations as software computational externs in the P4 language, enabling their use in the rank computation of scheduling policies, if required. In Fig. 2, an example of implementing the rank-based DRR algorithm is presented where a division extern is used in the rank computation of packets as outlined in the P4 program (line 6). On the other hand, the P4 language provides the other needed arithmetic operations in the same example, such as addition, or other required components, such as registers.

C. The Software Networking Environment

To integrate the proposed P4 programs, a modified BMv2 model comprising the externs of the utilized rank-based scheduler into a programmable switch was developed. This model is used later as a software environment that emulates the behavior of a programmable network of switches. The components of this environment are illustrated in the DRR case study presented in Fig. 2.

Several steps are required to implement the packet scheduler and validate the various scheduling policies discussed so far. First, we modify the BMv2 switch architecture to incorporate the utilized scheduler responsible for storing and scheduling packets. This modification is a one-time process independent of the chosen scheduling policy. Second, based on the applied scheduling policy, a P4 program is necessary to express the rank computation of packets and transmit the appropriate control signals and input data to the scheduling extern. The P4 program is then compiled with the modified BMv2 model using the P4 compiler (P4c). The resulting configuration files are utilized by the Mininet tool to emulate the behavior of the programmed switches. To emulate a network of multiple hosts and switches, within the Mininet tool, the network topology and the required hosts can be generated and configured to match the desired testbed setup. In our experiments, the workload generated by the packet generators varies depending on the selected case study and is transmitted to the respective hosts.

V. Evaluation of Proposed Rank-Based Scheduling Policies

In this section, we evaluate the accuracy and performance of the proposed rank-based algorithms compared to their

original policies. This evaluation is conducted using two distinct testbeds and realistic data center workloads derived from experiments detailed in [5] and [26]. Additionally, this section identifies the minimum prerequisites for implementing each scheduling policy individually through various experiments involving 20 policies on the BMv2 software model and the P4 programming language.

A. A Single BMv2 Switch Testbed

The first testbed is based on a single BMv2 target switch programmed by P4. In this switch, we implement a rank-based packet scheduler, the DR-PIFO [5], using the steps outlined in Section IV. We use one input port to pass the adopted workload to the switch, and all packets are routed to a single output port, which we use to monitor the output dequeue sequence of packets.

In this testbed, the evaluation metric is Bandwidth Utilization (BU), which is the ratio between the bandwidth assigned by the packet scheduler to a specific flow and the nominal bandwidth that should have been assigned to the same flow. We calculate the nominal bandwidth based on the expected ideal behavior of the adopted scheduling scheme in each case study. Thus, in our experiments, the selected scheduling algorithm determines the BU assigned to each flow (mainly based on the expected output order of this flow's packets), and each flow is assigned a bandwidth that should not exceed what it can use as explained in [5]. Consequently, the ideal value of BU is 1, and in general, the objective of a programmable scheduler is to minimize deviations of the BU value from this ideal value. We refer readers to the work in [5] for more details about this adopted testbed.

B. Testbed With a Leaf-Spine Topology of BMv2 Switches

The performance evaluation of the proposed rank-based algorithms extends to a data center topology composed of BMv2 switches, all programmed using P4. The topology comprises 9 leaf switches, each connected to 16 hosts that function as senders or receivers. Specifically, 72 hosts are designated as senders, while the other 72 serve as receivers. The leaf switches are connected by 4 spine switches, which route packets among leaf switches. Further details about this experimental setup can be found in [5].

We implemented the DR-PIFO scheduler as an extern across all switches using the BMv2 switch architecture and the P4 program, following the steps outlined in Section IV. Furthermore, each switch's rank computation is embedded in the deployed P4 program. It is worth noting that the P4 program is recompiled for each case study to adjust the rank calculations. In this testbed, the timing performance of the proposed algorithms is compared to the performance of the original policies. Thus, the evaluation metrics adopted are the Flow Completion Time (FCT) and the end-to-end delay, serving as end-to-end performance indicators [43], [44]. While the size or the duration of a flow is considered in the FCT, the end-to-end delay does not consider the flow's size. Our evaluation is based on the relative error of the FCT and the end-to-end delay, rather than relying on exact timing values [5], [45], [46]. This approach helps us assess the accuracy of the proposed rank-based versions compared to the original policies.

C. Evaluation of the Proposed Rank-Based Algorithms

We have successfully coded, compiled, and executed P4 programs for all the 20 studied scheduling algorithms. As mentioned in Section III, we introduced novel rank-based versions for 5 out of the 20 implemented scheduling policies: (P1-RB) rank-based DRR, (P2-RB) rank-based WRR, (P3-RB) rank-based WDRR, (P4-RB) Fair-Slytherin, and (P5-RB) rank-based SP-non-WC. The remaining 15 scheduling policies already have rank-based algorithms compatible with the rank-based schedulers. The P4 programs, provided in Section III, which implement these 15 scheduling policies, are visually tested. In these tests, the P4 programs exhibit the same behavior as theoretically expected from the policies. Thus, in this section, we focus on evaluating the performance of the newly proposed rank-based versions compared to the behavior of their original algorithms.

In a single-switch testbed, the rank-based versions of the first four algorithms produce identical packet departure sequences as their original counterparts. As a result, the Bandwidth Utilization (BU) values for all four algorithms consistently achieve the optimal value of 1. This shows that the proposed rank-based versions provide the same level of fairness as the original algorithms while handling various network flows and users. In the case study involving Slytherin, performance evaluation is conducted exclusively in the topology testbed, as Slytherin requires multiple switches along the destination path.

For the testbed configured as a datacenter leaf-spine topology, the relative errors on Flow Completion Time (FCT) values for the initial four algorithms are depicted in Fig. 3a. These relative errors are computed by comparing the FCT values of the rank-based version against those of the original version of the same algorithm. The results demonstrate that the proposed rank-based variants closely emulate the behavior of the original algorithms, exhibiting a maximum deviation of only 0.5%. Furthermore, the relative errors in the end-to-end delay of different flow sizes are presented in Fig. 3b. These relative errors confirm the observation made from the FCT results, with a maximum error of merely 0.25% in the end-to-end delays.

We also compared the original Slytherin algorithm to our proposed Fair-Slytherin under equivalent network conditions. We directly compared the FCT and the end-to-end delay values between the two algorithms. This comparison aligns with the assertions made in [9], which emphasize Slytherin's efficiency in enhancing the timing performance, particularly by reducing FCT values for shorter flows. The ratio between the FCTs and the delays of our Fair-Slytherin and the original Slytherin are reported in Fig. 3c and Fig. 3d, respectively. The outcomes show that our Fair-Slytherin consistently reduces the FCT and the delay values across most flow sizes, with a significant reduction for shorter flows.

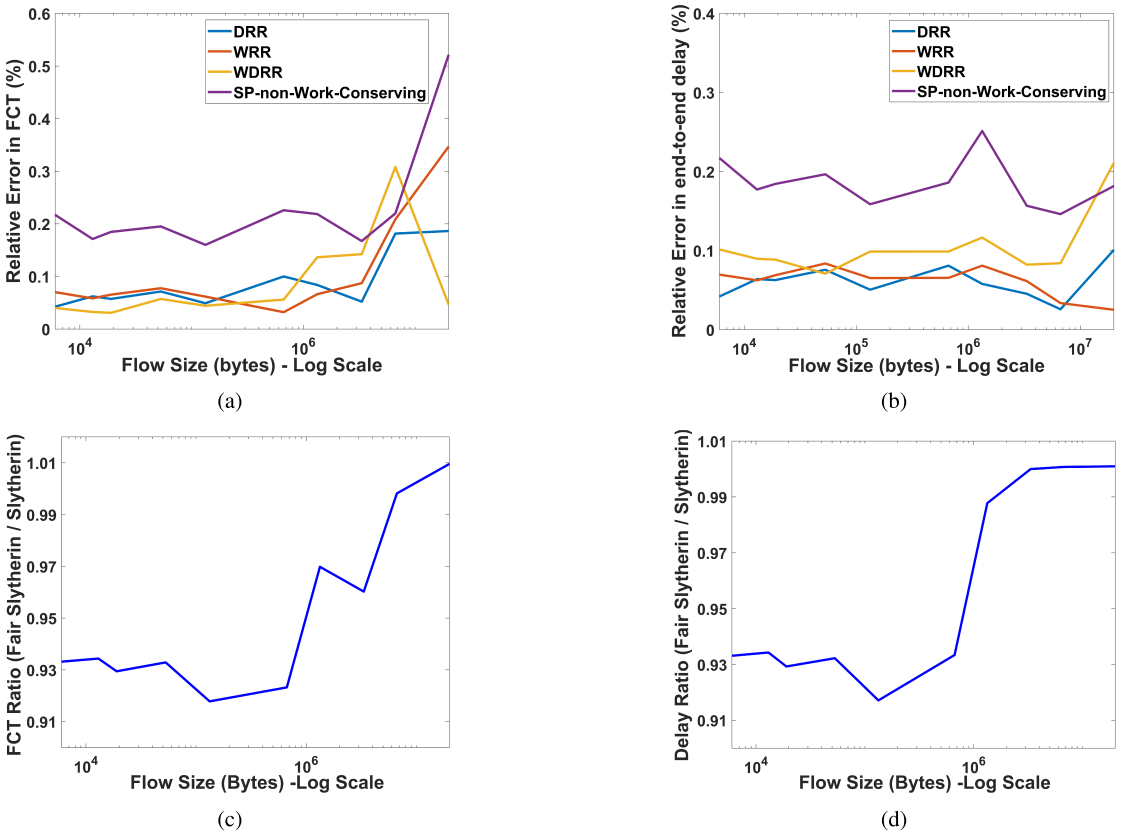

Fig. 3. The results of evaluating the rank-based algorithms. (a) The relative FCT error for the proposed novel rank-based versions. (b) The relative error in the end-to-end delays for the same policies (c) The ratio between the FCTs of our Fair-Slytherin and those of the original Slytherin. (d) The ratio for the end-to-end delay in the Slytherin case study.

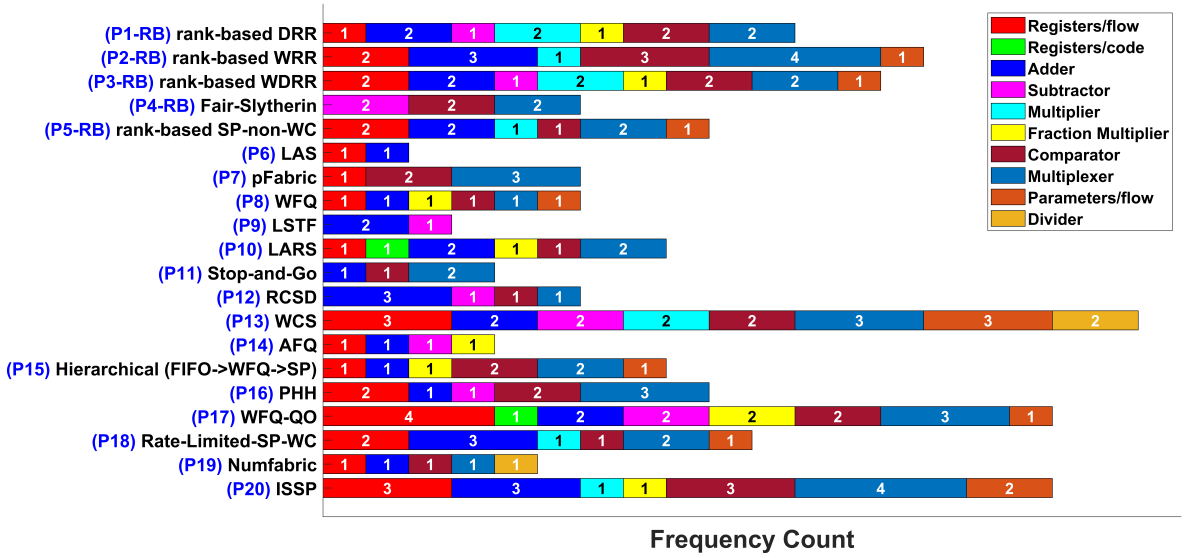

Fig. 4. Minimum requirements for implementing various scheduling and shaping algorithms.

D. Reports of the Required Resources

The implemented prototypes have identified the type and number of components required to express the scheduling policies. These components are listed in Fig. 4, showing the minimum necessary objects for each type to implement each scheduling policy individually. The pseudo-codes for these scheduling policies can be found in Section III. To accommodate multiple scheduling algorithms in the same program, the minimum requirements of each algorithm should be considered. It is important to note that the prototypes are not optimized, and optimization could potentially reduce the requirements and allow for resource sharing among different computation aspects.

All the listed components in Fig. 4, except the divider and the fraction multiplier, are available in the P4 language. The divider is used to perform the division of two integers at run time, and the fraction multiplier is used to multiply an integer by a fraction. A floor division is utilized in the prototypes

since the results must be integers to be acknowledged by the P4 language. For the same reason, the result of the fraction multiplier is constrained to be an integer for compatibility. These two components are implemented as externs, but we believe they should be fully integrated into the P4 language. However, the computation of scheduling algorithms might need to be optimized or approximated to avoid using these components. The availability of the existing components may be limited by the deployed switch architecture, and the number of required components can impact the switch's ability to support a larger number of flows. Additionally, the bit width of variables and parameters used in the scheduling algorithms can change based on various specifications of the target switch. We suggest that future research explores opportunities for optimizing these algorithms.

VI. Discussion

To our knowledge, no previous work has comprehensively studied the list of existing scheduling policies, provided in Table I, in a common programmable network environment. A possible reason behind this is the absence of a programmable abstraction of a scheduler in the current programmable data planes. In this work, integrating a rank-based scheduler, such as the DR-PIFO, into a programmable network environment enables us to study diverse networking scenarios. Accordingly, we implement many scheduling and shaping policies individually, encompassing 20 distinct policies, to determine their minimum requirements and necessary components. Furthermore, we introduce rank-based versions for 5 of these 20 policies, while the remaining policies naturally align with the rank-based scheduling model. The evaluation of the newly proposed rank-based versions illustrates that they closely replicate the original behavior of the respective policies, with no error in the single switch testbed, and a maximum FCT error of 0.5% in the data-center topology testbed.

While the DR-PIFO was chosen as the rank-based scheduler in our experiments, we also explored alternative packet schedulers, such as the PIFO [1]. The PIFO can express most of the studied scheduling policies with identical performance. The exception arises when scheduling policies necessitating dynamic ranking, such as the pFabric [27] and window-constrained scheduling [35]. In a previous work [5], the state-of-the-art DR-PIFO scheduler exhibited superior accuracy in expressing these scheduling algorithms compared to other rank-based schedulers.

A. Programmable Scheduler Integration Vs. Extern Implementation

Providing the functionality of a programmable packet scheduler as an external component, as seen in the case of our proposed networking environment described in Section IV, offers several potential advantages. First, it enhances extensibility, allowing upgrades without requiring modifications to the PSA architecture. Second, it promotes a separation of concerns, reducing complexity. With this approach, the PSA architecture can concentrate on core switch functionality, supported by P4, while the packet scheduler addresses scheduling policies. This isolation can facilitate the specialization of the scheduler to suit specific network requirements.

However, implementing the packet scheduler's functionality as an external component may introduce certain drawbacks. It can lead to performance degradation due to increased communication latency and complicate debugging. By contrast, a native scheduler module integrated within the PSA architecture and supported by the P4 language could streamline debugging and enhance interoperability within the larger SDN ecosystem. Such an approach would empower network operators to define QoS Objectives without needing intricate knowledge of the implementation details.

B. Event-Driven Packet Scheduling Model

Our model offers additional flexibility to the fixed packet schedulers in today's programmable data plane. However, the utilized programmable scheduler still operates within the boundaries of traditional fixed-interval designs where the processing and scheduling of packets occur synchronously at regular clock-based intervals, regardless of dynamic network conditions. A more flexible and responsive approach would involve an event-driven design.

The work presented in [47] lays a solid foundation for event-driven packet processing in P4-based switch architectures. Tailoring such a design to a rank-based scheduler like the DR-PIFO is feasible. Custom events, including flow arrival and departure, queue occupancy updates, dynamic weight adjustments, and congestion detection events, must be defined. Logic to support this event-driven design would involve calculating initial weights or queue occupancy upon a flow's arrival and triggering events to update queue occupancy and flow-specific states. The shared register extern suggested in [47] can be utilized to maintain shared state information. This state could include flow-specific buffer occupancy counters, flow weights, or congestion status indicators.

Event-driven designs have the potential to reduce packet latency and improve scalability by reacting to events as they occur. They allow real-time adjustments to scheduling parameters based on flow behavior. However, such a design may result in increased resource utilization and introduce hardware design and programming complexity.

VII. Conclusion

In conclusion, this paper extensively investigated the compatibility of 20 network scheduling policies with state-of-the-art rank-based programmable schedulers. The majority of these algorithms align with rank-based scheduling. We proposed novel rank-based versions for five policies not originally formulated as rank-based. To seamlessly integrate any programmable rank-based scheduler into the programmable data plane, we introduced a generic methodology, incorporated within our customized programmable network environment. Leveraging this environment, we successfully implemented all the scheduling policies that were considered. These implementations were validated with BMv2 programmable switches

programmed using the P4 network programming language. These implementations facilitated the evaluation of both the accuracy and performance of the proposed rank-based versions across various testbeds and workloads. The results indicated that the novel rank-based versions closely replicate the behavior of the original policies, with a maximum error of 0.5%.

Furthermore, we identified the minimum requirements for implementing 20 considered scheduling policies. While most of these required components are readily available in contemporary data planes, the widely used P4 network programming language does not provide some inherently. Consequently, the authors believe that a promising avenue for future research is to enrich P4 with the reported components required to implement the set of 20 considered rank-based scheduling policies without losing generality for applicability to any chosen scheduling policy.

ACKNOWLEDGMENT

The authors would like to thank André Béliveau from NoviFlow for his valuable advice and technical guidance.

REFERENCES

[1] A. Sivaraman et al., "Programmable packet scheduling at line rate," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 44–57.

[2] V. Shrivastav, "Fast, scalable, and programmable packet scheduler in hardware," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 367–379.

[3] N. K. Sharma et al., "Programmable calendar queues for high-speed packet scheduling," in *Proc. 17th {USENIX} Symp. Netw. Syst. Design Implement. ({NSDI})*, 2020, pp. 685–699.

[4] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[5] M. Elbediwy, B. Pontikakis, A. Ghaffari, J.-P. David, and Y. Savaria, "DR-PIFO: A dynamic ranking packet scheduler using a push-in-first-out queue," *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 1, pp. 355–371, Feb. 2024.

[6] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Trans. Netw.*, vol. 4, no. 3, pp. 375–385, Jun. 1996.

[7] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 8, pp. 1265–1279, Oct. 1991.

[8] J. Lakkakorpi, A. Sayenko, and J. Moilanen, "Comparison of different scheduling algorithms for Wimax base station: Deficit round-robin vs. Proportional fair vs. weighted deficit round-robin," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Mar. 2008, pp. 1991–1996.

[9] H. Rezaei, M. Malekpourshahraki, and B. Vamanan, "Slytherin: Dynamic, network-assisted prioritization of tail packets in datacenter networks," in *Proc. 27th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2018, pp. 1–9.

[10] S. Laki et al., "Core-stateless forwarding with QoS revisited: Decoupling delay and bandwidth requirements," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 503–516, Apr. 2021.

[11] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishement in wide-area networks," *IEEE J. Sel. Areas Commun.*, vol. 8, no. 3, pp. 368–379, Apr. 1990.

[12] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," in *Proc. ACM Symp. Commun. Archit. Protocols*, 1990, pp. 19–29.

[13] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, Jun. 1993.

[14] H. Zhang and B. Jon, "WF^{2}Q: Worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, vol. 96, Mar. 1996, pp. 120–128.

[15] S. J. Golestani, "A stop-and-go queueing framework for congestion management," in *Proc. ACM Symp. Commun. Archit. Protocols*, 1990, pp. 8–18.

[16] D. Verma, "Guaranteeing delay jitter bounds in packet switcing networks," in *Proc. IEEE Tricom*, Apr. 1991, pp. 35–46.

[17] R. Kalmanek, Charles, K. Hemant, and S. Keshav, "Rate controlled servers for very high-speed networks," in *Proc. IEEE Global Telecommun. Conf. Exhibit. (GLOBECOM)*, Dec. 1990, pp. 12–20.

[18] H. Zhang and D. Ferrari, "Rate-controlled static-priority queueing," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Mar. 1993, pp. 227–236.

[19] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal packet scheduling," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2016, pp. 501–521.

[20] A. G. Alcoz, A. Dietmüller, and L. Vanbever, "SP-PIFO: Approximating push-in first-out behaviors using strict-priority queues," in *Proc. 17th {USENIX} Symp. Netw. Syst. Design Implement. (NSDI)*, 2020, pp. 59–76.

[21] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 6, pp. 1030–1039, Jun. 1999.

[22] C. Zhang et al., "PIPO: Efficient programmable scheduling for time sensitive networking," in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2021, pp. 1–11.

[23] Z. Yu et al., "Programmable packet scheduling with a single queue," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 179–193.

[24] M.-Y. Zhu, K.-F. Chen, Z. Chen, and N. Lv, "FAIFO: UAV-assisted IoT programmable packet scheduling considering freshness," *Ad Hoc Netw.*, vol. 134, Sep. 2022, Art. no. 102912.

[25] B. Vass, C. Sarkadi, and G. Rétvári, "Programmable packet scheduling with SP-PIFO: Theory, algorithms and evaluation," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2022, pp. 1–6.

[26] M. Elbediwy, B. Pontikakis, J.-P. David, and Y. Savaria, "A hardware architecture of a dynamic ranking packet scheduler for programmable network devices," *IEEE Access*, vol. 11, pp. 61422–61436, 2023.

[27] M. Alizadeh et al., "pFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM*, 2013, pp. 435–446.

[28] F. Hauser et al., "A survey on data plane programming with p4: Fundamentals, advances, and applied research," 2021, *arXiv:2101.10632*.

[29] R. Sharma, S. S. Sehra, and S. K. Sehra, "Review of different queuing disciplines in VOIP, video conferencing and file transfer," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 4, no. 3, pp. 264–267, Mar. 2015.

[30] I. A. Rai, E. W. Biersack, and G. Urvoy-Keller, "Size-based scheduling to improve the performance of short TCP flows," *IEEE Netw.*, vol. 19, no. 1, pp. 12–17, Jan. 2005.

[31] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 4, pp. 1–12, Aug. 1989.

[32] J. Y.-T. Leung, "A new algorithm for scheduling periodic, real-time tasks," *Algorithmica*, vol. 4, nos. 1–4, p. 209, 1989.

[33] M. Heusse, G. Urvoy-Keller, A. Duda, and T. X. Brown, "Least attained recent service for packet scheduling over wireless LANs," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, vol. 2, Jun. 2010, pp. 1–9.

[34] H. Zhang and D. Ferrari, "Rate-controlled service disciplines," *J. High Speed Netw.*, vol. 3, no. 4, pp. 389–412, 1994.

[35] Y. Zhang and R. West, "End-to-end window-constrained scheduling for real-time communication," in *Proc. 10th Int. Conf. Real-Time Embedded Comput. Syst. Appl. (RTCSA)*, 2004, pp. 143–152.

[36] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*. Renton, WA, USA: USENIX Association, Apr. 2018, pp. 1–16.

[37] L. Yang, B. Ng, and W. K. G. Seah, "Heavy hitter detection and identification in software defined networking," in *Proc. 25th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2016, pp. 1–10.

[38] C.-C. Li, S.-L. Tsao, M. Cheng Chen, Y. Sun, and Y.-M. Huang, "Proportional delay differentiation service based on weighted fair queuing," in *Proc. 9th Int. Conf. Comput. Commun. Netw.*, Oct. 2000, pp. 418–423.

[39] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti, "NUMFabric: Fast and flexible bandwidth allocation in datacenters," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 188–201.

[40] P. Goyal, H. M. Vin, and H. Cheng, "Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 690–704, Oct. 1997.

[41] T. P. L. Consortium. (2016). *Behavioral Model (BMV2)*. [Online]. Available: https://github.com/p4lang/behavioral-model

[42] J. S. da Silva, F.-R. Boyer, L.-O. Chiquette, and J. M. P. Langlois, "Extern objects in p4: An ROHC header compression scheme case study," in *Proc. 4th IEEE Conf. Netw. Softwarization Workshops (NetSoft)*, Jun. 2018, pp. 517–522.

[43] B. Yan, Q. Liu, J. Shen, and D. Liang, "Flowlet-level multipath routing based on graph neural network in OpenFlow-based SDN," *Future Gener. Comput. Syst.*, vol. 134, pp. 140–153, Sep. 2022.

[44] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, 2006.

[45] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: Probabilistic in-band network telemetry," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, New York, NY, USA, 2020, pp. 662–680.

[46] A. Ingolfsson, E. Akhmetshina, S. Budge, Y. Li, and X. Wu, "A survey and experimental comparison of service-level-approximation methods for nonstationaryM(t)/M/s(t)queueing systems with exhaustive discipline," *INFORMS J. Comput.*, vol. 19, no. 2, pp. 201–214, May 2007.

[47] S. Ibanez, G. Antichi, G. Brebner, and N. McKeown, "Event-driven packet processing," in *Proc. 18th ACM Workshop Hot Topics Netw.*, Nov. 2019, pp. 133–140.

Mostafa Elbediwy received the B.Sc. degree in nanotechnology and nanoelectronics engineering from Zewail City of Science and Technology, Egypt, in 2018, and the Ph.D. degree from the Electrical Engineering Department, Polytechnique Montréal. His Ph.D. research focuses on designing hierarchical, scalable, and programmable traffic management architectures that can operate at line rate and are also efficient enough to be implemented in today's programmable software-defined networks. His research interests include network traffic management, software-defined networking, computer architecture, digital systems design, high-level synthesis, and reconfigurable computing.

Bill Pontikakis received the B.Eng. degree in computer engineering and the M.A.Sc. degree in electrical and computer engineering from Concordia University, Montreal, QC, Canada, in 2002 and 2003, respectively, and the Ph.D. degree in electrical engineering from Polytechnique Montréal, Montreal. He is currently a Research Associate with the Canada NSERC Industrial Research Chair for High-Speed and Programmable Packet Processors, Department of Electrical Engineering, Polytechnique Montréal. His contributions to the research chair are vital to its success, as he actively participates in the chair's scientific, operational, budgeting, public relations, and research vision aspects. His current research interests include programmable network data planes and the P4 language. He presented his first paper at the IEEE ISCAS conference during his undergraduate studies, which earned him the ReSMIQ undergraduate research award in microelectronics in Montreal.

Jean-Pierre David (Member, IEEE) received the B.Ing. degree in electrical engineering from the University of Liège, Liège, Belgium, in 1995, and the Ph.D. degree from Universitè Catholique de Louvain, Belgium, in 2002. He was an Assistant Professor with Universitè de Montrèal, Montreal, QC, Canada, in 2002, and moved to Polytechnique Montrèal, Montreal, in 2006, where he is currently a Professor with the Department of Electrical Engineering. His research interests include digital system design, reconfigurable computing, high-level synthesis, high-speed communications, neural networks, and their applications.

Yvon Savaria (Life Fellow, IEEE) received the B.Ing. and M.Sc.A. degrees in electrical engineering from École Polytechnique Montréal, Canada, in 1980 and 1982, respectively, and the Ph.D. degree in electrical engineering from McGill University, in 1985. Since 1985, he has been with Polytechnique Montréal, where he is currently a Professor with the Department of Electrical Engineering. He is currently involved in several projects related to embedded systems in aircraft, wireless sensor networks, virtual networks, software-defined networks, machine learning (ML), embedded ML, computational efficiency, and application-specific architecture design. He has carried out work in several areas related to microelectronic circuits and microsystems, such as testing, verification, validation, clocking methods, defect and fault tolerance, effects of radiation on electronics, high-speed interconnects and circuit design techniques, CAD methods, reconfigurable computing and applications of microelectronics to telecommunications, aerospace, image processing, video processing, radar signal processing, and the acceleration of digital signal processing. He holds 16 patents, has published 190 journal articles and 485 conference papers, and was the thesis advisor of 190 graduate students who completed their studies. He is a Fellow of the Canadian Academy of Engineering. He is also a member of the executive committee of the Regroupement Stratégique en Microélectronique du Québec (RESMIQ) and a member of the Ordre des Ingénieurs du Québec (OIQ). In 2001, he was awarded a Tier 1 Canada Research Chair (www.chairs.gc.ca) on the design and architecture of advanced microelectronic systems, which he held until June 2015. He also received a Synergy Award from Canada's Natural Sciences and Engineering Research Council in 2006. Since June 2019, he has been the NSERC-Kaloom-Intel-Noviflow (KIN) Chair Professor. He has been working as a Consultant or was sponsored for carrying out research with Bombardier, Buspass, CNRC, Design Workshop, Dolphin, DREO, Ericsson, Genesis, Gennum, Huawei, Hyperchip, Intel, ISR, Kaloom, LTRIM, Miranda, MiroTech, Nortel, Octasic, PMC-Sierra, Space Codesign, Technocap, Thales, Tundra, and Wavelite. He was the Program Co-Chairperson of NEWCAS 2018 and the General Chairperson of NEWCAS 2020.