Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

HSDBA: a hierarchical and scalable dynamic bandwidth allocation for programmable data planes*

Dengyu RAN[1,2], Xiao CHEN[1,2], Lei SONG‡[1,2]

[1]*National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China*

[2]*School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China*

E-mail: randy@dsp.ac.cn; xxchen@dsp.ac.cn; songl@dsp.ac.cn

Received Aug. 31, 2023; Revision accepted Mar. 10, 2024; Crosschecked Sept. 19, 2024

Abstract: Dynamic bandwidth allocation (DBA) is a fundamental challenge in the realm of networking. The rapid, accurate, and fair allocation of bandwidth is crucial for network service providers to fulfill service-level agreements, alleviate link congestion, and devise strategies to counter network attacks. However, existing bandwidth allocation algorithms operate mainly on the control plane of the software-defined networking paradigm, which can lead to considerable probing overhead and convergence latency. Moreover, contemporary network architectures necessitate a hierarchical bandwidth allocation system that addresses latency requirements. We introduce a fine-grained, hierarchical, and scalable DBA algorithm, i.e., the HSDBA algorithm, implemented on the programmable data plane. This algorithm reduces network overhead and latency between the data plane and the controller, and it is proficient in dynamically adding and removing network configurations. We investigate the practicality of HSDBA using protocol-oblivious forwarding switches. Experimental results show that HSDBA achieves fair bandwidth allocation and isolation guarantee within approximately 25 packets. It boasts a convergence speed 0.5 times higher than that of the most recent algorithm, namely, approximate hierarchical allocation of bandwidth (AHAB); meanwhile, it maintains a bandwidth enforcement accuracy of 98.1%.

Key words: Dynamic bandwidth allocation; Software-defined networking; Programmable data plane; Protocol-oblivious forwarding switch (POFSwitch)

https://doi.org/10.1631/FITEE.2300593 **CLC number:** TP393

1 Introduction

While the technology for dynamic bandwidth allocation (DBA) has considerably evolved in the domain of wired networks, the growing reliance on cloud services and the rising complexity of modern network architectures, including information-centric networking (ICN) (Xylomenos et al., 2014), polymorphic smart network (PINet) (Hu et al., 2020), and network slicing, have introduced new design requirements and challenges for data planes. Taking PINet as an example, the concept and architecture of PINet are introduced to tackle the challenges encountered in future heterogeneous networks. PINet is characterized by its openness, adaptability, and universality. The introduction of polymorphic nodes and a polymorphic routing mechanism enables the network to dynamically allocate and optimize resources to meet the requirements of diverse applications and services. In this scenario, the data plane needs to be programmable while offering isolation mechanisms to ensure efficient and equitable

‡ Corresponding author

* Project supported by the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDA031050100)

ORCID: Dengyu RAN, https://orcid.org/0000-0002-3866-2531; Lei SONG, https://orcid.org/0000-0002-8248-3194

allocation of bandwidth (Wang et al., 2022).

Bandwidth is a scarce resource in the network. Although congestion control mechanisms implemented on end-hosts contribute to managing bandwidth allocation, relying exclusively on them can result in an inequitable distribution of bandwidth. Malicious traffic or poorly designed applications can obtain a larger share of bandwidth by establishing multiple transmission control protocol (TCP) flows. Furthermore, certain link-capacity-based congestion control mechanisms, e.g., bottleneck bandwidth and round-trip propagation time (BBR) (Cardwell et al., 2017), have the potential to dominate the bandwidth resources over traditional pacing-based TCP connections (e.g., Tahoe, Reno, and Cubic). This affects the quality of service (QoS) for end-users or cloud tenants and ultimately results in the degradation of the service-level agreement (SLA). Consequently, researchers have proposed solutions to enable DBA in intermediate network devices, such as switches or routers (Kumar et al., 2015). Moreover, there are several well-established deployment scenarios, including software-driven wide-area network (SWAN) (Hong et al., 2013) and B4 (Jain et al., 2013).

Research conducted by Google has revealed that data centers typically exhibit a hierarchical structure with at least five levels of traffic aggregation (Jain et al., 2013; Noormohammadpour and Raghavendra, 2018). Similar patterns of bandwidth allocation have been observed in wide-area networks (WANs) (Kumar et al., 2015). Fig. 1 illustrates a prototypical example of fine-grained bandwidth allocation within the framework of PINet. Each layer in this hierarchy represents a distinct scope of resource allocation. The uppermost layer indicates the link capacity of the port, and the number of nodes at this layer corresponds to the number of switch panel ports. The subsequent layers symbolize the subdivisions of this bandwidth. Within the PINet framework, these layers are designated, in descending order, as port, polymorphic, subnet, user, and service. The DBA algorithm permits the configuration of a guaranteed rate (GR) for each node. In scenarios wherein the aggregate rate at the port surpasses the link capacity, the bandwidth is apportioned among nodes in accordance with the hierarchical max-min fairness (MMF) principle (Luangsomboon and Liebeherr, 2021). Consequently, each aggregation node

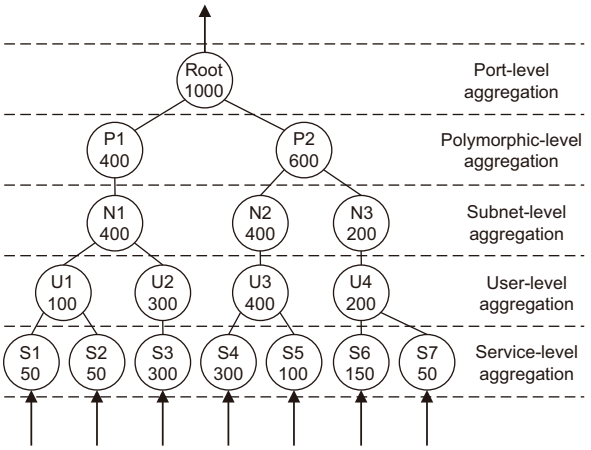

Fig. 1 Typical five-level bandwidth configuration for polymorphic smart network (PINet)

(AN) is allocated either the requested bandwidth or the fair share.

Programmable data planes have introduced flexibility and reconfigurability to data plane devices. However, their bandwidth management capabilities continue to rely on conventional queue scheduling algorithms (Bennett and Zhang, 1996, 1997; Guo, 2001; Ramabhadran and Pasquale, 2003; Saeed et al., 2019). Despite the effectiveness of these approaches, they are burdened by multiple implementation constraints. Incorporating such features into a programmable data plane is a complex process, starting with the hardware data plane. Among the popular choices are P4-based data planes, which can reach terabit-per-second-level throughput. However, such a hardware system is constrained by limited on-chip resources, typically supporting only 1–2 levels, each hosting 8–32 queues, and offering a finite selection of scheduling algorithm alternatives. In contrast, software data planes, such as Open vSwitch (OVS) (Pfaff et al., 2015), vector packet processor (VPP) (Barach et al., 2018), and protocol-oblivious forwarding switch (POFSwitch) (Yu JZ and Wang, 2014), do not impose restrictions on the number of queues. However, the extensive pre-allocated software queues can result in significant memory overhead and introduce additional computational burden to the central processing unit (CPU). This could potentially escalate the number of CPU cores in use and augment the queuing latency of packets.

Current research primarily concentrates on bandwidth sharing and traffic-scheduling problems in data center networks. These investigations account for the bandwidth constraints on network

pathways and aim to achieve equitable sharing of link resources by dynamically modulating bandwidth allocation strategies (Shieh et al., 2011; Chen et al., 2016; Xia et al., 2017). It is noteworthy that, in the context of data centers, there is no imperative for the data plane to preserve a hierarchical data structure. Additionally, bandwidth allocation is predominantly achieved via a centralized control plane (Al-Fares et al., 2010; Benson et al., 2011; Curtis et al., 2011), potentially leading to considerable convergence latency. Furthermore, several studies have used active queue management (AQM) mechanisms to discern network congestion and probabilistically drop packets (Sharma et al., 2018; Thapeta et al., 2021; Li et al., 2023). Yu ZL et al. (2021b) presented a hierarchical structure built upon the traditional core stateless fair queuing (CSFQ) strategy (Stoica et al., 1998), termed hierarchical CSFQ (HCSFQ), whereas approximate hierarchical allocation of bandwidth (AHAB) (MacDavid et al., 2023) focuses on bandwidth allocation challenges in network slicing. AHAB addresses the shortcomings of HCSFQ in terms of convergence speed and uses a sketch-based algorithm to estimate user rates, thereby reducing spatial occupancy. It is relevant to note that both HCSFQ and AHAB apply strategies similar to AQM in dealing with bandwidth constraints, and these are implemented within the pipelines of programmable devices. Both methods use the algorithms similar to the water-filling approach to heuristically approximate fair sharing.

Modern networks are increasingly relying on programmable data planes as foundational devices for packet forwarding, offering greater flexibility and scalability to meet the needs of various applications and workloads. This shift is evident across a broad spectrum of deployment scenarios (Dalton et al., 2018). As a consequence, we introduce the hierarchical and scalable dynamic bandwidth allocation (HSDBA) algorithm, designed for the deployment in programmable data planes. Distinguished by its rapid convergence and high efficiency in traffic isolation, HSDBA represents a significant advancement compared with existing DBA methodologies. Our contributions are delineated as follows:

1. Addressing the issue of fair bandwidth allocation, we propose a fine-grained hierarchical method that can support GR configurations.

2. The proposed method adopts a pipeline-based approach, eliminating any output queuing latency within the switch. The scheme is able to cope with the joining of new ANs and the departure of existing ones, ensuring the flexibility and scalability of the allocation.

3. The algorithm can be programmed by the existing protocol-oblivious forwarding (POF) instruction set. We implement a prototype of the POF software switch (x86 platform), and the experimental results show that HSDBA can achieve hierarchical bandwidth allocation and ensure isolation.

2 Related works

Bandwidth allocation and isolation are central to the performance of network systems (e.g., flow completion time, long-tail latency, and QoS guarantees) in data centers and WANs, ensuring that different virtual networks can meet their respective SLAs.

For the software data planes, Eiffel (Saeed et al., 2019) uses an integer priority queue to efficiently support a wide range of policies and ranking functions. It introduces novel programming abstractions to express scheduling policies. The QoS framework of DPDK (https://www.dpdk.org/) provides a five-level hierarchical scheduler that implements traffic shaping, strict priority, and weighted round robin at different levels. Since the priority of a traffic class (TC) is strictly defined, its scheduling is determined by the dequeue state machine which selects queues in ascending order of priority, and the lowest priority queue does not have a chance to be processed until the higher priority queues finish scheduling. Therefore, DPDK scheduling cannot satisfy the need for fair scheduling. Hierarchical link sharing (HLS) (Luangsomboon and Liebeherr, 2021) ensures hierarchy MMF of bandwidth. HLS supports minimum GRs and isolation between classes. HLS has overhead comparable to that of the other hierarchical scheduling algorithms for Linux kernels, such as credit-based shaper (CBS) (Floyd and Jacobson, 1995) and hierarchical token bucket (HTB) (Devera, 2003). C2QoS (Yang et al., 2021) is a CPU-cycle based network QoS strategy applied to vSwitch in public clouds. Considering the problem of vSwitch competing with virtual machines (VMs) for CPU resources, C2QoS allocates input/output-dedicated CPU resources to each VM to ensure bandwidth.

However, the context of C2QoS is not the same as the hierarchical network in this paper, wherein the software switches are deployed as bare metal in the WAN and each forwarding pipeline is bound to a specific logical core, so there is no competition for CPU and VM resources.

For the hardware data plane, Cascone et al. (2017) introduced FDPA, a method for bandwidth sharing predicated on the transmission rates among TCP senders, using byte counters in P4 switches to monitor each user's sending rate and directing incoming packets into distinct priority queues. Lee and Chan (2019) used multicolor marking in virtual networks to secure bandwidth guarantees, focusing on bandwidth isolation between virtual network services and enhancing the QoS. Cebinae (Yu LC et al., 2022) uses a token bucket to estimate the per-flow fair share rate, achieving fairness by "taxing" flows that exceed a threshold, but it takes several seconds to converge. Nimble (Thapeta et al., 2021) achieves precise TCP flow rate limiting by simulating queue draining in the data plane. However, it supports only fixed rates set by the control plane. To address this issue, Wu et al. (2023) proposed AQ, a scalable network abstraction method, defining a function named A-Gap to calculate the difference between different applications, ensuring that different congestion control algorithms can properly co-exist. Additionally, AHAB (MacDavid et al., 2023) and HCSFQ (Yu ZL et al., 2021b) use an approximate method for hierarchical bandwidth allocation, which is implemented on P4 switches. Another category of research focuses on programmable schedulers, supporting various scheduling tasks that include bandwidth allocation, e.g., PIFO (Sivaraman et al., 2016), PIEO (Shrivastav, 2019), SP-PIFO (Alcoz et al., 2020), PCQ (Sharma et al., 2020), and AIFO (Yu ZL et al., 2021a). This is a cutting-edge research direction in recent years.

It is evident that a majority of existing methods depend on queue mechanisms to facilitate bandwidth allocation. However, the finite quantity of queues available in switches, coupled with their inability to accommodate arbitrary hierarchical structures and to adjust dynamically, significantly hampers scalability. We introduce a novel DBA algorithm that transcends the limitations of queue-based scheduling. A comprehensive explanation of this approach will be elaborated upon in subsequent sections.

3 Problem specification

In this section, we give our problem definition and present the objectives on which we focus. Table 1 presents the main notations used in Sections 3 and 4.

Table 1 Summary of notations

Notation	Description
N	Any non-leaf aggregation node
child(N)	Set of child nodes of the aggregation node N
R_i	Bandwidth demand, where $i \in$ child(N)
α_N	Fair share rate of child(N)
B_N	Bandwidth obtained by node N
$\mathcal{S}, \mathcal{U}$	Satisfied, unsatisfied state
g_i	Guaranteed rate, where $i \in$ child(N)
w_i	Weight, where $i \in$ child(N)
l_i	Peak information rate, where $i \in$ child(N)
pkt	Packet
l(child)	Packet length
T_i, T_{i-1}	Current, previous packet timestamp

3.1 Weighted max-min fairness

Fairness has consistently held a paramount position in evaluating the effectiveness of resource allocation strategies. An ideal resource allocation mechanism should ensure fairness while optimizing resource utilization. MMF is widely acknowledged as a resource allocation strategy that achieves a balance between effectiveness and fairness, among the various approaches used to achieve fairness. MMF shields flows from starvation while simultaneously optimizing each flow's throughput. Furthermore, even in the presence of malicious flows, MMF can still provide basic guarantees for normal flows and effectively reuse the total bandwidth.

MMF categorizes flows into two groups based on a fair share value denoted as α_N. The first category encompasses flows with bandwidth requirements lower than or equal to α_N, and these flows are allocated a bandwidth resource value denoted as R_i that aligns with their respective demands. The second category comprises services with resource requirements surpassing α_N, and these services are allocated bandwidth equivalent to the fair resource value α_N.

Aligned with the issue outlined by MacDavid et al. (2023), our objective entails the progressive allocation of link bandwidth to individual ANs in a hierarchical fashion. When the total bandwidth demand of the child nodes exceeds the capacity B_N

of node N, the equitable allocation of bandwidth to node N, in accordance with MMF, can be mathematically depicted by the equation below:

$$\alpha_N = \arg\max_{\alpha_N} \sum_{i \in \text{child}(N)} \min\left(R_i, w_i \alpha_N\right). \quad (1)$$

This study offers support for GR for each AN. GR represents the minimum rate that a node can secure in the event of link congestion, serving as a safeguard against the manipulation of network resources by specific users or applications. Within the context of MMF, GR is attained through a well-structured weight configuration that fulfills the following conditions:

$$\frac{w_1}{g_1} = \frac{w_2}{g_2} = \ldots = \frac{w_n}{g_n}, \quad (2)$$

where n represents the number of child nodes of the AN.

In the context of a programmable switch with constrained switching capacity, there exists a unique allocation scheme that adheres to the principles of weighted MMF. Conventionally, the fair share computation is executed using the water-filling algorithm. However, this algorithm presents an inherent trade-off between convergence speed and accuracy. Notably, network traffic in real-world scenarios often demonstrates significant temporal fluctuations. If the convergence speed of the algorithm is low, bandwidth allocation might not effectively respond to traffic changes, thereby compromising fairness, particularly for shorter flows. Conversely, if the algorithm lacks adequate accuracy, it could result in unjust bandwidth allocation outcomes. Consequently, it becomes vital to develop an algorithm that can concurrently maintain high accuracy and demonstrate rapid convergence to facilitate efficient and equitable network resource allocation.

3.2 Objectives

3.2.1 Arrival rate measurement

Accurate estimation of flow rates plays a pivotal role in QoS management. Through accurate estimation of arrival rates, switch can allocate bandwidth and resources according to the SLAs or other prioritization policies. This practice ensures that critical applications and services receive sufficient bandwidth and priority, thereby optimizing the overall user experience.

3.2.2 Bandwidth isolation and fairness

The objective is to prevent monopolization of network bandwidth, and we want to ensure that all users or applications in the network obtain a fair share of the bandwidth through on-demand allocation. In this context, network resources are not fixedly allocated, but dynamically adjusted according to their weights and real-time arrival rates.

3.2.3 Bandwidth enforcement

This objective empowers network administrators to impose restrictions on the transmission rate of specific nodes. Imposing these limitations enables network administrators to have precise control over the distribution of network resources, ensuring that diverse applications and services conform to predetermined performance standards. This aspect is of paramount importance in upholding stability and reliability of the network, particularly in scenarios characterized by high network traffic or intense resource competition.

3.2.4 Scalability

This objective aims to equip switches with the flexibility to adjust to new bandwidth configuration needs and changes in user access. It necessitates that switches not only facilitate the joining of new ANs but also adeptly handle the exit of existing ones.

4 HSDBA approach

4.1 Design overview

The overall structure of the algorithm is shown in Fig. 2, and it consists mainly of three steps. The first step is rate measurement, which is used to obtain the actual rate of the current-level AN, i.e., the demand value of the node for bandwidth resources. This step provides the necessary input information to ensure an effective and fair allocation of bandwidth. The next step is bandwidth enforcement, whereby the switch implements a random packet drop strategy by calculating the drop probability of packets, thereby limiting the rate of the node within the fair share rate. For packets determined to be dropped, they are marked with drop_flag, and this step also involves maintaining the state values of the node. The final step is fair share update, in which the fair

share of each node is recalculated based on the measured rate and state transitions from the first two steps. This newly calculated fair share will be used as the basis for subsequent decisions on whether packets are dropped. The following subsections discuss in detail the specific design details of the three steps, combined with Algorithm 1.

Through these three steps, HSDBA can detect load changes on the data plane and automatically manage DBA. This automated process optimizes both network performance and service quality by ensuring efficient network utilization. The implementation of this method is crucial for managing network congestion and maintaining stable operation for mission-critical applications.

4.2 Rate measurement

Packet headers typically carry information regarding packet length. For instance, in the case of an incoming IPv6 packet, the device uses the payload_length field specified within the header to perform layer 2 (L_2) rate measurements. Similar to CSFQ (Stoica et al., 1998), this method uses the standard exponential averaging mechanism widely used in networking for rate estimation, as depicted by Eq. (3). The current rate is influenced by both current inputs and historical rates, undergoing adjustments via an exponential decay factor to reflect recent changes effectively. This exponential term, distinguished by the decay factor τ, dictates the degree of influence that the preceding rate exerts on the current rate. By iteratively applying this formula in a stepwise fashion to incoming packets, rate metrics are derived at various temporal points.

$$R_i = l(\text{pkt}) - R_{i-1}\mathrm{e}^{-\frac{T_i - T_{i-1}}{\tau}}. \tag{3}$$

Rate measurement is of paramount importance in various network use cases and serves as a fundamental aspect of network performance analysis and optimization. By facilitating real-time data collection, rate measurement enables the optimization of network performance, ensuring efficient resource utilization and maximized QoS. Moreover, it provides valuable insights into the dynamic behavior of network traffic, enabling active decision-making and proactive management of network resources.

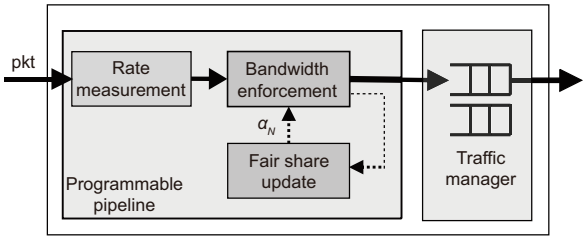

Fig. 2 Overview of hierarchical and scalable dynamic bandwidth allocation (HSDBA)

Algorithm 1 HSDBA

Require: pkt, the arrival packet
$B'_N \leftarrow$ link capacity
for each layer i **do**
  /* rate measurement */
  $\text{AN}_i \leftarrow$ table.lookup(i, pkt)
  AN_i.rate $\leftarrow$ RateMeasure($\text{AN}_i, l(\text{pkt})$)
  /* bandwidth enforcement */
  if $\alpha_i > \text{AN}_i$.rate **then**
    AN_i.state $\leftarrow \mathcal{S}$
    S.rate $\leftarrow$ RateMeasure($\text{AN}_{i-1}, l(\text{pkt})$)
  else if rand$(0,1) > \frac{\alpha_i w_i}{\text{AN}_i.\text{rate}}$ **then**
    AN_i.state $\leftarrow \mathcal{U}$
    S.rate $\leftarrow$ RateMeasure($\text{AN}_{i-1}, 0$)
    drop_flag $\leftarrow$ True
  else
    AN_i.state $\leftarrow \mathcal{U}$
    S.rate $\leftarrow$ RateMeasure($\text{AN}_{i-1}, 0$)
  end if
  U.weight $\leftarrow$ UpdateWeight(AN_{i-1}.state)
  /* fair share update */
  $\alpha_i \leftarrow (B_N - S.\text{rate})/U.\text{weight}$
  $B_N \leftarrow \min(\alpha_i w_i, \text{AN}_i.\text{rate})$
end for

4.3 Bandwidth enforcement

The enforcement of bandwidth is achieved through the execution of packet dropping, guided by a designated probability. The initial step involves comparing the value of R_i with $f_i = \min(l_i, w_i\alpha_N)$. In scenarios wherein $R_i \le f_i$, the packet loss probability is established as zero, indicating a state of satisfied for node i, denoted as $\mathcal{S}$. Conversely, if $R_i > f_i$, i is deemed unsatisfied (represented as $\mathcal{U}$). The formula for the probability of packet loss is as follows:

$$P_{\text{drop}} = \max\left(0, 1 - \frac{f_i}{R_i}\right) = 1 - \min\left(1, \frac{f_i}{R_i}\right). \tag{4}$$

For packets that reside in the unsatisfied state $\mathcal{U}$, a random number, rand, is generated by the switch. This number follows a uniform distribution in the range of 0 to 1. If the condition rand$>P_{\text{drop}}$ is fulfilled, the packet is consequently marked with drop_flag. The actual drop operation is then carried out after subsequent fair share updates have been performed.

The incorporation of bandwidth enforcement facilitates zero-queueing, achieved through the execution of packet drops during the pipelining stage. This ensures that the packets directed to the network interface card (NIC) do not exceed the link capacity, thereby eliminating the need for queue scheduling. The bandwidth enforcement approach offers several notable advantages. First, it reduces network latency by minimizing packet queuing latency, thereby enhancing the timeliness and responsiveness of packet transmission, which is particularly critical for latency-sensitive applications and services. Second, it simplifies the design and implementation of the network architecture, reducing the dependence on complex scheduling algorithms and streamlining the configuration process. Furthermore, it facilitates fault diagnosis by mitigating potential failures associated with queue scheduling, such as queue overflow, network congestion, and priority reversal. Applying pipeline processing minimizes the occurrence of these failures, simplifies troubleshooting procedures, and ultimately enhances the reliability and manageability of the network.

4.4 Fair share update

Based on the analysis conducted in Section 3.1, the concept of weighted max-min fairness (WMMF) involves classifying nodes into two groups based on the fair share. Nodes in the satisfied state $\mathcal{S}$ obtain bandwidth that matches their demand value. Conversely, nodes in the unsatisfied state $\mathcal{U}$ procure bandwidth equivalent to $w_i\alpha_N$. In the case where $\mathcal{U}=\varnothing$, the fair share α_N is determined as $\max(R_1,R_2,\ldots,R_n)$. However, if $\mathcal{U}\neq\varnothing$, which indicates congestion within the link, the WMMF principle asserts that the expression for α_N can be formulated as follows:

$$B_N=\sum_{i\in\text{child}(N)}\min(R_i,w_i\alpha_N)=\sum_{i\in\mathcal{S}}R_i+\sum_{i\in\mathcal{U}}w_i\alpha_N. \tag{5}$$

Thus, the fair share α_N can be expressed as

$$\alpha_N=\frac{B_N-\sum_{i\in\mathcal{S}}R_i}{\sum_{i\in\mathcal{U}}w_i}, \tag{6}$$

where $\sum_{i\in\mathcal{S}}R_i$ denotes the summation of rates for nodes in the satisfied state $\mathcal{S}$. The computation of this sum can be performed using the method presented in Section 4.2. Similarly, $\sum_{i\in\mathcal{U}}w_i$ stands for the sum of weights for the unsatisfied nodes $\mathcal{U}$, as detailed in Section 4.3. Upon determining the packet loss probability, the variable encapsulating the sum of weights is maintained in accordance with the current state and the previous state.

However, in practice, it is not possible to use Eq. (6) to compute α_N directly because the value of α_N depends on the contents of the sets $\mathcal{S}$ and $\mathcal{U}$, which in turn depend on the value of α_N. Nonetheless, it is still possible to compute α_N heuristically using Eq. (6). This is done by first initializing α_N to zero. Subsequently, in each iteration, the current value of α_N is used to determine the sets $\mathcal{S}$ and $\mathcal{U}$, and then Eq. (6) is used to compute the next value of α_N. Once α_N remains constant over successive iterations, the calculation can be stopped. This method, although indirect, is effective in practice for estimating the fair share rate.

The entire bandwidth convergence process is driven by packets within the data plane. Upon each packet's arrival at the switch, we harness the state forwarding capabilities of the programmable switch to update the state of its associated node and initiate corresponding actions based on state transitions. To illustrate this process, we construct a two-stage finite-state machine (FSM), as depicted in Fig. 3 and Table 2. Within the flow table entries, we use a 1-bit variable to maintain this state, initializing all ANs to the $\mathcal{U}$ state. If the rate of an AN falls below the fair share, its state transitions to $\mathcal{S}$. Then, the packet contributes to the computation of $\sum_{i\in\mathcal{S}}R_i$ and facilitates the updating of $\sum_{i\in\mathcal{U}}w_i$, further triggering the recalculation of α_N. For other scenarios involving state transitions, please refer to Table 2.

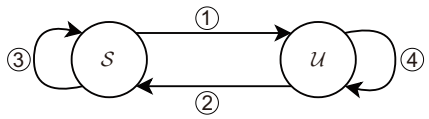

Fig. 3 A two-stage finite-state machine for aggregation nodes

Table 2 Operations corresponding to the two-stage finite-state machine

State	Update weight	Update satisfied AN rate	Update fair share
①	✓	✗	✓
②	✓	✓	✓
③	✗	✓	✓
④	✗	✗	✓

AN: aggregation node

To provide a concrete example, consider the scenario where three ANs, namely, A_1, A_2, and A_3, possess an equal weight and share a link capacity of $B_N = 50$. Given the measured rates of $R_1 = 10$, $R_2 = 20$, and $R_3 = 30$, we initialize $\alpha_N = 0$ and $\mathcal{S} = \varnothing$. In the first iteration, $\alpha_N = \frac{50-0}{3} = 16.7$, resulting in $\mathcal{S} = \{A_1\}$. Moving to the second iteration, $\alpha_N = \frac{50-10}{2} = 20$, resulting in $\mathcal{S} = \{A_1, A_2\}$. In the third iteration, $\alpha_N = \frac{50-(10+20)}{1} = 20$, leading to $\mathcal{S} = \{A_1, A_2\}$. At this point, the iteration is complete and the converged fair share is 20.

4.5 Joining and pruning of aggregation nodes

This subsection details the workflow when an AN joins or exits the network and the corresponding data structure. Nodes residing on the same hierarchical level have their matching fields and states uniformly stored within a singular flow table, as illustrated in Fig. 4. This figure provides a visual representation of the storage structure for the top three layers as exemplified in the PINet configuration (shown in Fig. 1). In real-world applications, the dimensions of the flow table escalate significantly with an increase in the number of hierarchical layers. The size of the first table is equivalent to the number of ports. Conversely, the subnet-level aggregation within the third table generally encompasses thousands of service entries. By leveraging the pipeline-based bandwidth allocation strategy, nodes can be dynamically installed or removed at runtime with ease. This approach distinctly provides superior scalability compared to the queue-based approach. The subsequent paragraphs provide a comprehensive explanation of the processes involved when nodes join or are pruned.

1. Joining. When a packet associated with AN arrives at the switch, its treatment depends on whether it explicitly matches an entry in the table provided by the controller. If there is a direct match, state maintenance and fair share update are executed as outlined in Sections 4.2–4.4. However, if the packet fails to match any entry, the default action of the entry is executed, which involves registration with the control plane via packet_in. This mechanism enables the controller to acquire the entry information of unregistered ANs and manage them effectively. The states of the newly installed AN entries are designated as $\mathcal{U}$, and the state values related to the ANs are updated accordingly. This approach ensures the effective management and service provisioning of nodes across varying hierarchical layers.

2. Pruning. In some cases, the control plane may actively delete nodes in one or more branches. This requires the controller to trigger the deletion of all entries in the flow table that belong to that branch and maintain the state values. To further complicate the situation, since the whole fair share update process is packet-driven, when the traffic of an AN suddenly stops, there will be no subsequent packets to maintain the state, resulting in zombie entries that keep occupying space and state values. To solve this problem, one straightforward approach is to poll all entries and states through the control plane to find and deal with these entries. This approach quickly detects changes in bandwidth requirements, but increases the overhead of the switch's local control plane. Another approach is to set an idle timeout for entries, so that if no packet is matched within the idle survival time, the contents of the entry are cleared. This approach reduces the control plane overhead, but may increase the bandwidth convergence time. Each approach possesses advantages and disadvantages. Our adopted strategy entails using polling for internal nodes since they request greater bandwidth and necessitate timely responses to avoid resource wastage. Conversely, for a larger number of leaf nodes, we use the timeout approach to maintain the state and asynchronously trigger the deletion of entries.

Match	Action	
Flow key	Measured rate	State
Root	1000	$\mathcal{S}$

(a)

Match	Action	
Flow key	Measured rate	State
P1	400	$\mathcal{S}$
P2	600	$\mathcal{S}$

(b)

Match	Action	
Flow key	Measured rate	State
N1	400	$\mathcal{S}$
N2	400	$\mathcal{S}$
N3	200	$\mathcal{S}$

(c)

Fig. 4 Example of the flow table structure: (a) port-level aggregation; (b) polymorphic-level aggregation; (c) subnet-level aggregation

3. Complexity analysis. According to the design described in this subsection, nodes at the same layer are collectively stored in a single flow table. Given this data structure, it is necessary to update and maintain the status of nodes at each layer within the tree-like bandwidth configuration. This further implies that a matching operation must be performed for each layer, and the process for each packet received by the switch is identical. Based on this, the time complexity of this algorithm can be summarized as $O(d)$, where d represents the depth of the tree. Regarding the space complexity, the algorithm requires storage of a number of table entries equal to the total number of nodes n across d flow tables, thereby defining the space complexity of the algorithm as $O(n)$.

5 Prototype implementation

This section describes the implementation of HSDBA in a programmable data plane. Specifically, we implement HSDBA-POF in the software switch developed on the basis of the existing POFSwitch (Yu JZ and Wang, 2014) and POF forwarding instruction set (POF-FIS). Based on this prototype, the state domain is divided into the global state, table state, flow state, and packet state using the method proposed by Jing et al. (2022). Data types and data locations are used to describe the data in different state domains. This allows compatibility with different types of data processing without affecting the forwarding performance (Jing et al., 2021).

The POF-FIS approach exhibits considerable flexibility in its application, necessitating only {offset, length} as domain operands to facilitate the computation of corresponding data. Consequently, we apply the existing POF-FIS to implement the three steps outlined in Sections 4.2–4.4, each corresponding to a callable instruction block. The first is the rate_estimate instruction block, which maintains a sliding window in the flow state area, accumulates the total packet length during the window time, and divides this value by the window length to derive the instantaneous rate. This rate is then written back to the packet state for subsequent flow table processing. If the window time is set to $<$1 s, the total number of bytes counted within this second equals the rate value in bytes per second. If the window value is set to less than one second, additional scaling calculations (i.e., left-shifting or dividing) are performed to obtain the instantaneous rate. The second is the bandwidth_enforce instruction block, using which we execute the random packet drop calculation. The current state of the AN, represented by a single bit, can be determined using data transfer instructions (e.g., set_field and rand), logical instructions (e.g., compare), and arithmetic operations (e.g., div_field). Lastly, the fairshare_update instruction block, based on the state transition determined in the second step, uses add_field, sub_field, and div_field operations to complete the fair share update. It is important to note that the fair share value is stored in the global data area. This setup allows for its update by different entries, while enabling subsequent packets to directly read the updated fair share value. This value then serves as a reference for the processing of subsequent packet drops.

POFSwitch uses a notable optimization strategy for idle timeouts to facilitate efficient HSDBA. Specifically, upon the triggering of an idle timeout, the switch refrains from immediately executing a delete operation. Instead, it designates the entry as inactive while preserving the state value, which represents U.weight in Algorithm 1. This strategy is underpinned by the observation that in a majority of cases, subsequent incoming streams may reactivate these entries. To minimize the registration delay of a node, the system swiftly reactivates an inactive entry when a subsequent packet matches it, eliminating the need for a full registration process at the switch's local control plane. Upon reaching the end of the node's intended lifecycle, the control plane explicitly removes the corresponding flow table entry. This streamlined approach optimizes efficiency and reduces system complexity.

6 Evaluation

6.1 Experimental setup

Fig. 5 shows the testbed for this case study. The HSDBA-POF prototype has been implemented on top of POFSwitch and DPDK 19.11.3. The hardware setup includes an Intel Xeon Silver 4216 CPU operating at 2.10 GHz, complemented by 256 GB of RAM, and an X710-DA4 NIC. The system runs on the CentOS 7.9.2009 operating system. Both

HSDBA-POF and DPDK are compiled using "-O3" and AVX512 optimizations. The controller works on another server with the same hardware configuration as POFSwitch, controlling the switch through the corresponding south bound interface. The Keysight Ixia XGS12 network test platform is set up as a traffic generator and analyzer that could simulate thousands of flows connected to the switch. Ixia establishes 4×10 Gb/s links with HSDBA-POF.

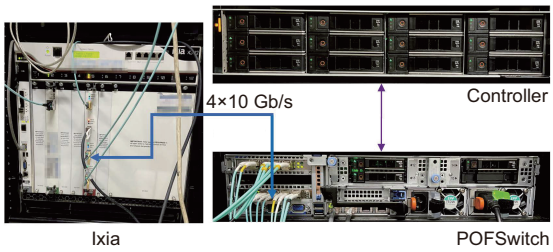

Fig. 5 Testbed topology for the experiments

To enable efficient testing, we use distinct bit intervals of the IPv4 destination address as the matching key for each flow table, which is subsequently installed by the control plane. In our setup, the packet length is fixed at 1024 bytes. By sequentially enabling and disabling L_2 and L_3 flow groups (FGs) in Ixia, we can measure the throughput curve of the flow using the receive rate display. This measurement provides valuable insights into the switch's bandwidth allocation across different flows.

6.2 Isolation of the data plane

6.2.1 Single layer

We evaluate the DBA capabilities of the algorithm in a single-layer scenario with equal or different weight configurations. The experimental setup is similar to that described in other works (Alcoz et al., 2020; Luangsomboon and Liebeherr, 2021; Yu ZL et al., 2021b). By controlling the traffic generator, we generate a total of six user datagram protocol (UDP) flows originating from different ports, with each flow transmitting at a rate of 3 Gb/s. These flows are all forwarded to a single port, with each flow being continuously sent over a period of 60 s. The experiment is initially conducted in a single-layer flat mode, in which all ANs are subordinate to the root node. The configuration details for this experiment are presented in Table 3.

Fig. 6 presents the results of bandwidth allocation under a single-layer configuration. As depicted in Fig. 6a, when all nodes possess equal weights, their scheduling results should align with those of the tail-drop first-in first-out (FIFO) queue. During the congestion interval (between 30 s and 80 s), each node secures a fair share of bandwidth. A distinctive feature of HSDBA is its capacity to flexibly allocate bandwidth based on the assigned weights of nodes. As demonstrated in Fig. 6b, during periods of link congestion, nodes A_3 and A_6 are able to proportionally allocate more bandwidth.

Table 3 Experimental configuration for verifying single-layer bandwidth isolation

Class	Interval (s)	Rate (Gb/s)	Equal weight in Fig. 6a	Different weights in Fig. 6b
A_1	[0, 60]	3	1	1
A_2	[10, 70]	3	1	1
A_3	[20, 80]	3	1	2
A_4	[30, 90]	3	1	1
A_5	[40, 100]	3	1	1
A_6	[50, 110]	3	1	2

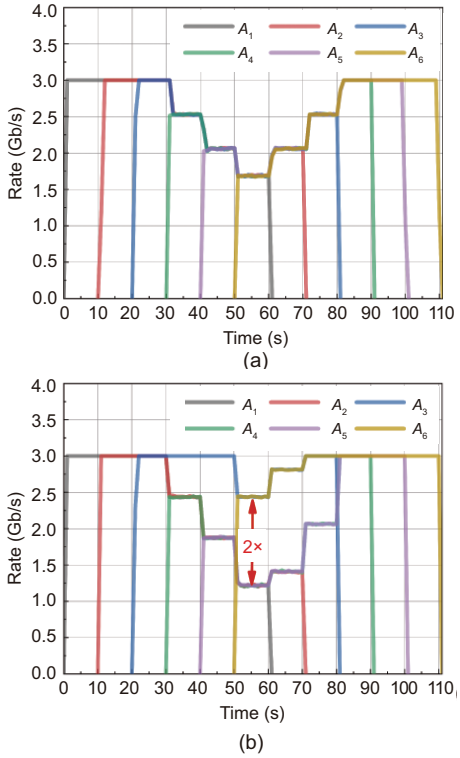

Fig. 6 Bandwidth allocation results in single-layer flat mode with a link congestion interval of 30–80 s: (a) equal weight; (b) different weights (References to color refer to the online version of this figure)

6.2.2 Multiple layers

Subsequently, we present the experimental results of hierarchical bandwidth allocation under diverse weights in a multilayer configuration. We assess the bandwidth convergence in a two-layer scenario and allocate weights to the nodes across both layers. The duration and traffic of each flow are kept consistent with those in the single-layer test. The specific configuration is depicted in Fig. 7.

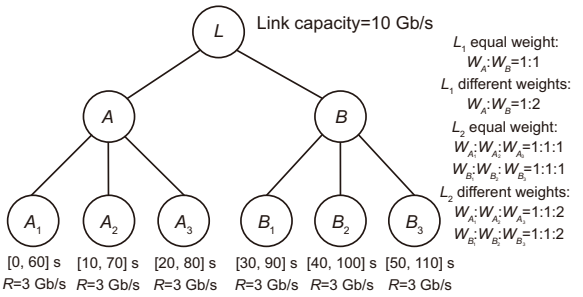

Fig. 7 Bandwidth allocation configuration in the multilayer mode

Fig. 8 presents the results of bandwidth allocation under a multilayer configuration. The scheduling results differ from those of the single-layer configuration, primarily due to the initial requirement for bandwidth allocation to ensure isolation between nodes A and B at layer 1 (L_1). Consider the equal weight scenario depicted in Fig. 8a: during periods of link congestion (e.g., between 30 s and 40 s), the bandwidth request for node A at L_1 is 9 Gb/s, while that of node B is 3 Gb/s. In line with the principle of fairness, the bandwidth demand of node B_1 can be fully met. However, nodes A_1–A_3 can only distribute a fair bandwidth of 7 Gb/s obtained by node A, resulting in approximately 2.3 Gb/s each. Figs. 8b–8d illustrate more complex scenarios involving varying weight configurations across layers. The results demonstrate that HSDBA is capable of distributing bandwidth equitably among all nodes, adhering to the hierarchical weight MMF allocation principle. This weight-sensitive bandwidth allocation strategy

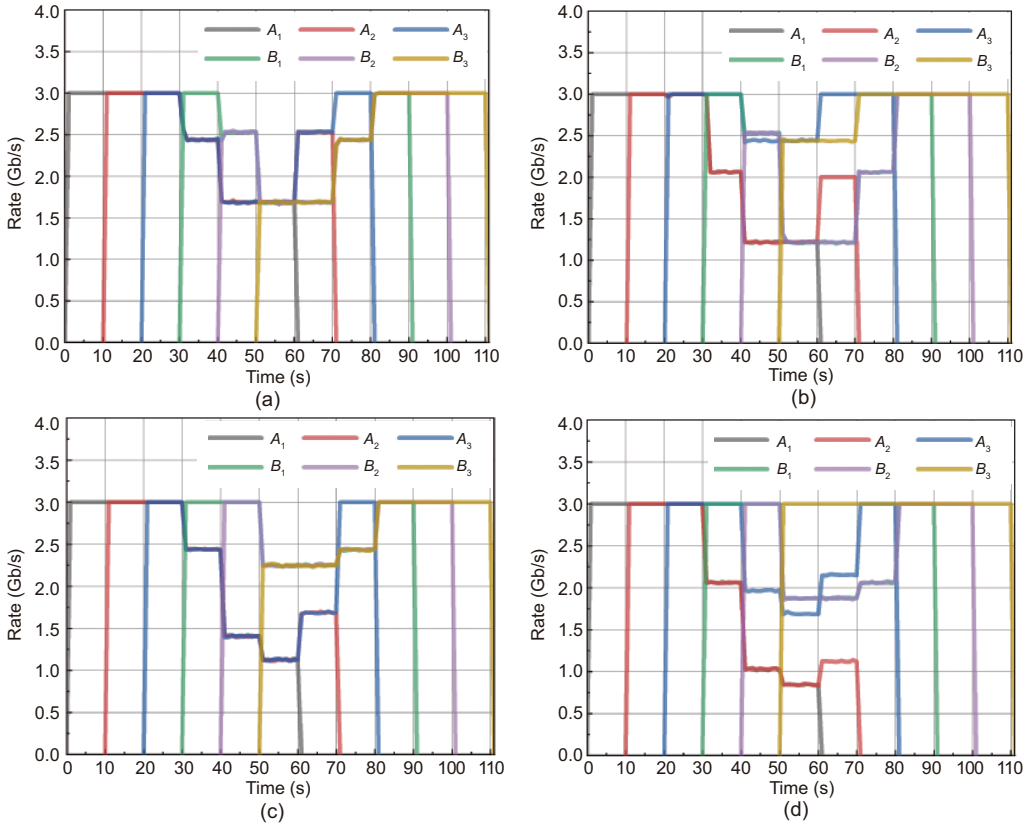

Fig. 8 Bandwidth allocation results in a multilayer mode with a link congestion interval of 30–80 s: (a) L_1 equal weight and L_2 equal weight; (b) L_1 equal weight and L_2 different weights; (c) L_1 different weights and L_2 equal weight; (d) L_1 different weights and L_2 different weights (References to color refer to the online version of this figure)

is crucial for ensuring the performance in critical applications and enhancing the overall network efficiency in various scenarios.

6.3 Bandwidth allocation combined with congestion control algorithms

In this experiment, we explore the cooperative effect between HSDBA and host-side congestion control mechanisms. Without active bandwidth enforcement by the switch, different congestion control algorithms lead to varying bandwidth convergence results. For this investigation, we set up an experiment where FG1 and FG2 each comprise 20 flows. FG1 lasts from 0 to 40 s, while FG2 runs from 20 to 60 s. The two FGs use distinct congestion control algorithms. We then evaluate the average throughput achieved per flow in both FG1 and FG2, comparing the results with or without the application of HSDBA.

As depicted in Fig. 9a, when both FGs use Linux's default Cubic algorithm, their convergence results are strikingly similar, with each flow essentially receiving an equal share of bandwidth. This can be largely attributed to the inherent fair bandwidth allocation mechanism of the TCP.

However, in real-world scenarios, it is improbable that each flow traversing the switch will adopt the same transport layer protocol. To simulate this situation, we configure FG1 to use the Cubic algorithm and FG2 to use the BBR algorithm (BBR is an aggressive algorithm that may lead to fairness issues for TCP connections). Furthermore, we maintain the setting of FG1 and set FG2 to UDP. As illustrated in Figs. 9b and 9c, without the application of HSDBA, the presence of FG2 compresses the traditional TCP connection and secures more bandwidth, no matter whether it is TCP-BBR or UDP. This results in a significant reduction in fairness. Conversely, HSDBA can provide fairness guarantees such that TCP connections using different congestion control algorithms receive nearly equal transmission rates.

Since this research method uses a packet loss policy for bandwidth limitation, a small amount of jitter may occur for congestion control algorithms based on packet loss detection. Nonetheless, the fairness issue is significantly improved compared to the scenario wherein HSDBA is not applied. In summary, as network service providers cannot control all end-hosts to adopt the same congestion control protocol, this experiment demonstrates that the application of HSDBA can achieve fair bandwidth allocation in the presence of a diverse set of congestion control protocols.

6.4 Bandwidth allocation convergence

In our testbed environment, due to the minimum limit of a 1-s data-sampling interval imposed by the Ixia tester, we are unable to directly observe the bandwidth convergence speed (the convergence time is typically in a millisecond level) from the time dimension. Consequently, we opt to use the number of packets driving bandwidth convergence as a measure of bandwidth convergence speed. For the experimental setup, we adhere to the same pattern as HCSFQ and configure four flows in single-layer mode with a fixed link capacity of 100 Mb/s. These four flows maintain a stable sending rate of 100 Mb/s, but each has different start and end time.

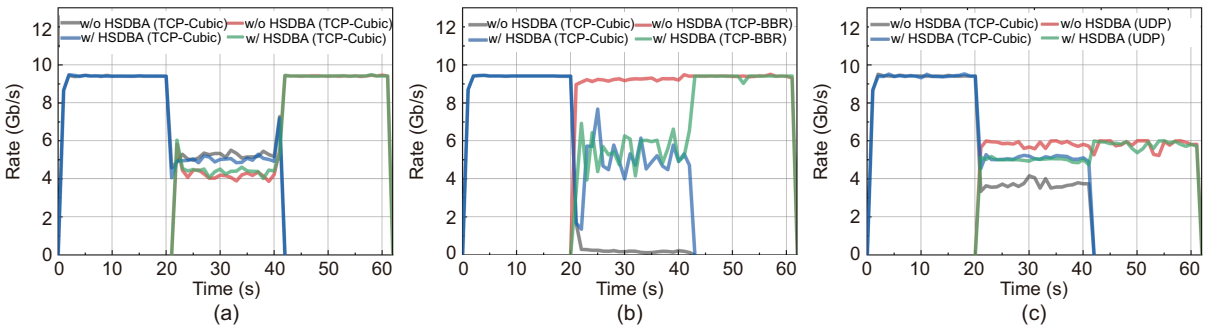

Fig. 9 Bandwidth allocation results combined with congestion control algorithms: (a) TCP-Cubic+TCP-Cubic; (b) TCP-Cubic+TCP-BBR; (c) TCP-Cubic+UDP (w/: with; w/o: without. References to color refer to the online version of this figure)

Compared to HCSFQ and AHAB, our approach uses a more direct and efficient strategy for calculating the fair bandwidth. Specifically, instead of iteratively approximating the fair bandwidth using a heuristic algorithm akin to the water-filling approach, we directly compute the updated fair share in a one-step manner through state transition. Consequently, in our approach, as soon as a change in the flow rate is detected, the switch is capable of using the current packets to drive the fair share update. This implies that the speed of bandwidth convergence is solely influenced by the number of packets that can drive the measured rate update. With this computational strategy, our method outperforms both AHAB and HCSFQ in terms of convergence speed. As shown in Table 4, our method is approximately 19.6 and 0.5 times faster when compared to HCSFQ and AHAB, respectively, in terms of convergence speed. These results demonstrate that by using the strategies of state shifting and on-the-fly updating, our method can significantly enhance the efficiency of bandwidth convergence. This is crucial for network systems that require real-time agility.

Table 4 A comparison of the number of packets used for the convergence of various bandwidth allocation algorithms

Algorithm	Number of packets
HCSFQ	516
AHAB	37
HSDBA	25

6.5 Bandwidth enforcement precision

We evaluate the precision of bandwidth enforcement by setting rate limits that range from 100 kb/s to 1 Gb/s. As depicted in Fig. 10, HSDBA exhibits an average deviation of 1.9% when enforcing any prescribed rate limit, resulting in 98.1% accuracy. Based on the representation of Eq. (4), it can be inferred that to achieve accurate bandwidth-limiting results, one must obtain precise packet loss probabilities, which in turn are determined by accurate bandwidth measurements. This result underscores the accuracy of both the rate measurement and drop probability calculations, demonstrating that HSDBA is capable of accommodating future growth in port line speeds and handling any level of rate-limiting scenarios.

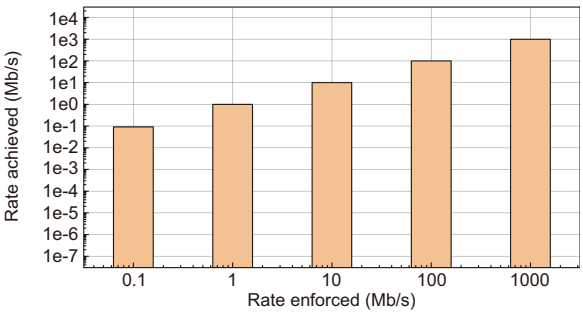

Fig. 10 HSDBA rate enforcement accuracy (100 kb/s to 1 Gb/s)

6.6 Forwarding latency

Packets are subject to three distinct delays in the switch: processing delay, control plane delay, and queuing delay. Processing delay refers to the time taken for each packet to classify and execute the instruction block. Control plane delay encompasses the time required for the packet to be transmitted to the local/global control plane for processing before being sent back to the switch as a packet_out. Queuing delay is the latency incurred when the link is busy transmitting other packets, causing newly arriving packets to wait in the output buffer.

In our experimental setup, we set the link capacity to 1 Gb/s and generate four UDP flows, each capable of transmitting at a rate of 300 Mb/s. Within POFSwitch, we implement a queue-based deficit weighted round robin (DWRR) scheduler that maps the four flows to four queues. This configuration is compared against HSDBA-POF in multilayer mode to measure the packet forwarding latency. Additionally, we establish the latency of the switch operating at 1 Gb/s with zero packet loss as a baseline for comparison.

The results are depicted in Fig. 11. Since HSDBA completes the bandwidth enforcement in the pipeline, the packets are forwarded to the NIC at a rate that does not exceed the link capacity, which reduces the latency of the packets waiting in the software queues. In comparison, the HSDBA-POF method significantly reduces latency compared to the queue-based isolation approach, achieving an average latency reduction of 80.1%. If the long-tail effect of the CPU system is not taken into account, then theoretically the packets for each flow can be forwarded in a constant time in any case. In the context of the control plane latency of the joining packet, it takes about 3.6 ms to send the packet_in to the

open network operating system (ONOS) controller and complete the registration, and about 4.53 ms to update the new rules in the data plane.

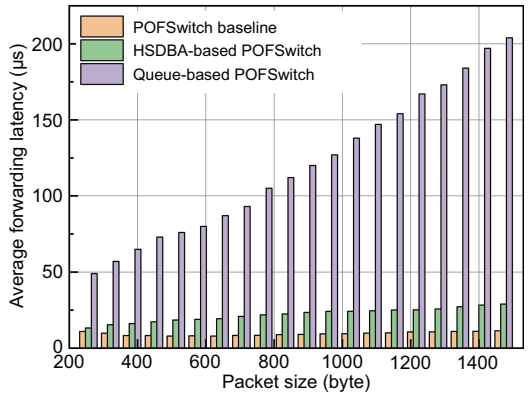

Fig. 11 HSDBA average forwarding latency compared to queue-based POFSwitch (References to color refer to the online version of this figure)

6.7 Scalability

In this experiment, we demonstrate the scalability of HSDBA by increasing the number of configured layers and nodes. When the number of nodes to be supported increases, the aggregate rates of the nodes may differ by orders of magnitude. Moreover, the presence of measurement errors may potentially lead to a decrease in flow fairness. Similarly, when the number of supported layers increases, multiple rate measurements are needed, which may cause the calculation errors at each layer to accumulate and eventually become unacceptable.

We configure the bandwidth allocation as a six-layer complete binary tree, which includes 32 leaf nodes. We refer to the high "H" scenario configuration in the work of Luangsomboon and Liebeherr (2021), setting nodes 1–12 to generate 100 Mb/s UDP flows and nodes 13–32 to generate 500 Mb/s UDP flows, with all nodes set to an equal weight. We measure their convergent bandwidth.

The experimental results are shown in Fig. 12. When only an FIFO method is used as the output, the presence of large flows greatly reduces the bandwidth obtained by small flows, and the rate of some small flows may even drop to near-zero. In contrast, HSDBA can provide hierarchical bandwidth isolation guarantees, and all small flows (flows 1–12) can obtain the bandwidth they need. Flows 13–16, which belong to the left subtree of the binary tree, can obtain bandwidth equal to their demand values in higher-level bandwidth allocation, while flows 17–32, which all belong to the right subtree, can only fairly share 5 Gb/s of bandwidth.

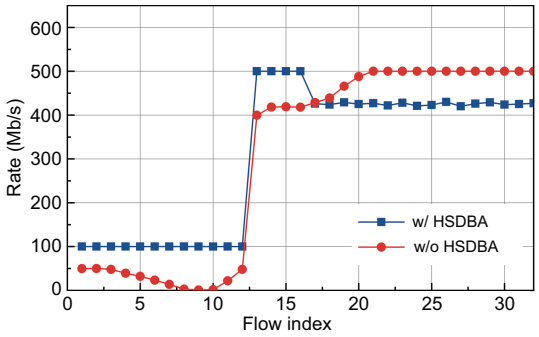

Fig. 12 HSDBA fair bandwidth allocation in a five-level binary tree (w/: with; w/o: without)

For the hardware resources occupied by HSDBA, our method consumes mainly the storage resources in the switch. The 128-bit flow metadata space can meet the state programming requirements of HSDBA. For a flow table with a table item size of 4096, the huge page memory occupied by a single-layer flow table is approximately 2.68 MB, which is acceptable in the software data plane.

7 Conclusions

This paper introduces HSDBA, an approach to hierarchical and fair DBA in software-defined networking frameworks. Diverging from traditional methodologies, HSDBA is implemented entirely in the data plane and does not rely on queue schedulers. The bandwidth allocation process is executed in three packet-driven steps: rate measurement, bandwidth enforcement, and fair share update. This process ensures that HSDBA's bandwidth allocation mechanism is both granular and responsive.

Our comprehensive evaluations reveal that HSDBA outperforms current solutions in achieving hierarchical MMF, showcasing a convergence rate up to 19.6 and 0.5 times higher compared with those of HCSFQ and AHAB, respectively. Regarding latency, HSDBA minimizes packet-queuing delays by preemptively dropping packets in the pipeline, yielding an average latency reduction of 80.1% when compared to queue-based scheduling algorithms. Additionally, HSDBA maintains a bandwidth

enforcement accuracy of 98.1%. Moreover, HSDBA surpasses these traditional algorithms in terms of scalability and flexibility, theoretically accommodating any number of bandwidth configurations across any network layer. This marks progress in meeting and managing the growing demand for network bandwidth, providing network service providers and data center operators with a powerful tool to ensure high-quality service and efficient resource utilization.

Future work will extend the applicability of HSDBA to P4-enabled switches, addressing the challenges posed by hardware resource and computing constraints. Future tests on hybrid hardware and software data planes will further determine HSDBA's efficiency and flexibility in various network environments.

Contributors

Dengyu RAN and Lei SONG designed the research. Dengyu RAN processed the data and drafted the paper. Xiao CHEN helped organize the paper. Xiao CHEN and Lei SONG revised and finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

Alcoz AG, Dietmüller A, Vanbever L, 2020. SP-PIFO: approximating push-in first-out behaviors using strict-priority queues. Proc 17th USENIX Conf on Networked Systems Design and Implementation, p.59-76.

Al-Fares M, Radhakrishnan S, Raghavan B, et al., 2010. Hedera: dynamic flow scheduling for data center networks. Proc 7th USENIX Conf on Networked Systems Design and Implementation, Article 19.

Barach D, Linguaglossa L, Marion D, et al., 2018. High-speed software data plane via vectorized packet processing. *IEEE Commun Mag*, 56(12):97-103. https://doi.org/10.1109/MCOM.2018.1800069

Bennett JCR, Zhang H, 1996. WF/sup 2/Q: worst-case fair weighted fair queueing. Proc IEEE INFOCOM, p.120-128. https://doi.org/10.1109/INFCOM.1996.497885

Bennett JCR, Zhang H, 1997. Hierarchical packet fair queueing algorithms. *IEEE/ACM Trans Netw*, 5(5):675-689. https://doi.org/10.1109/90.649568

Benson T, Anand A, Akella A, et al., 2011. MicroTE: fine grained traffic engineering for data centers. Proc 7th Conf on Emerging Networking Experiments and Technologies, Article 8. https://doi.org/10.1145/2079296.2079304

Cardwell N, Cheng Y, Gunn CS, et al., 2017. BBR: congestion-based congestion control. *Commun ACM*, 60(2):58-66. https://doi.org/10.1145/3009824

Cascone C, Bonelli N, Bianchi L, et al., 2017. Towards approximate fair bandwidth sharing via dynamic priority queuing. IEEE Int Symp on Local and Metropolitan Area Networks, p.1-6. https://doi.org/10.1109/LANMAN.2017.7972168

Chen L, Li BC, Li B, 2016. Barrier-aware max-min fair bandwidth sharing and path selection in datacenter networks. IEEE Int Conf on Cloud Engineering, p.151-160. https://doi.org/10.1109/IC2E.2016.35

Curtis AR, Kim W, Yalagandula P, 2011. Mahout: low-overhead datacenter traffic management using end-host-based elephant detection. Proc IEEE INFOCOM, p.1629-1637. https://doi.org/10.1109/INFCOM.2011.5934956

Dalton M, Schultz D, Adriaens J, et al., 2018. Andromeda: performance, isolation, and velocity at scale in cloud network virtualization. Proc 15th USENIX Conf on Networked Systems Design and Implementation, p.373-387.

Devera M, 2003. Linux Hierarchical Token Bucket. `http://luxik.cdi.cz/~devik/qos/htb/` [Accessed on Aug. 31, 2023].

Floyd S, Jacobson V, 1995. Link-sharing and resource management models for packet networks. *IEEE/ACM Trans Netw*, 3(4):365-386. https://doi.org/10.1109/90.413212

Guo CX, 2001. SRR: an $O(1)$ time complexity packet scheduler for flows in multi-service packet networks. *ACM SIGCOMM Comput Commun Rev*, 31(4):211-222. https://doi.org/10.1145/964723.383076

Hong CY, Kandula S, Mahajan R, et al., 2013. Achieving high utilization with software-driven WAN. Proc ACM SIGCOMM, p.15-26. https://doi.org/10.1145/2534169.2486012

Hu YX, Li D, Sun PH, et al., 2020. Polymorphic smart network: an open, flexible and universal architecture for future heterogeneous networks. *IEEE Trans Netw Sci Eng*, 7(4):2515-2525. https://doi.org/10.1109/tnse.2020.3006249

Jain S, Kumar A, Mandal S, et al., 2013. B4: experience with a globally-deployed software defined WAN. *ACM SIGCOMM Comput Commun Rev*, 43(4):3-14. https://doi.org/10.1145/2534169.2486019

Jing LN, Chen X, Wang JL, 2021. Design and implementation of programmable data plane supporting multiple data types. *Electronics*, 10(21):2639. https://doi.org/10.3390/electronics10212639

Jing LN, Wang JL, Chen X, 2022. MSSA: constant time state search through multi-scope state area. *Appl Sci*, 12(2):559. https://doi.org/10.3390/app12020559

Kumar A, Jain S, Naik U, et al., 2015. BwE: flexible, hierarchical bandwidth allocation for WAN distributed computing. Proc ACM Conf on Special Interest Group on Data Communication, p.1-14. https://doi.org/10.1145/2785956.2787478

Lee SSW, Chan KY, 2019. A traffic meter based on a multicolor marker for bandwidth guarantee and priority differentiation in SDN virtual networks. *IEEE Trans Netw Serv Manag*, 16(3):1046-1058. https://doi.org/10.1109/TNSM.2019.2923110

Li ZY, Hu YX, Tian L, et al., 2023. Packet rank-aware active queue management for programmable flow scheduling. *Comput Netw*, 225:109632. https://doi.org/10.1016/j.comnet.2023.109632

Luangsomboon N, Liebeherr J, 2021. HLS: a packet scheduler for hierarchical fairness. IEEE 29th Int Conf on Network Protocols, p.1-11. https://doi.org/10.1109/ICNP52444.2021.9651972

MacDavid R, Chen XQ, Rexford J, 2023. Scalable real-time bandwidth fairness in switches. IEEE Conf on Computer Communications, p.1-10. https://doi.org/10.1109/INFOCOM53939.2023.10228997

Noormohammadpour M, Raghavendra CS, 2018. Datacenter traffic control: understanding techniques and tradeoffs. *IEEE Commun Surv Tutor*, 20(2):1492-1525. https://doi.org/10.1109/COMST.2017.2782753

Pfaff B, Pettit J, Koponen T, et al., 2015. The design and implementation of Open vSwitch. Proc 12th USENIX Conf on Networked Systems Design and Implementation, p.117-130.

Ramabhadran S, Pasquale J, 2003. Stratified round robin: a low complexity packet scheduler with bandwidth fairness and bounded delay. Proc Conf on Applications, Technologies, Architectures, and Protocols for Computer Communications, p.239-250. https://doi.org/10.1145/863955.863983

Saeed A, Zhao YM, Dukkipati N, et al., 2019. Eiffel: efficient and flexible software packet scheduling. Proc 16th USENIX Conf on Networked Systems Design and Implementation, p.17-32.

Sharma NK, Liu M, Atreya K, et al., 2018. Approximating fair queueing on reconfigurable switches. Proc 15th USENIX Conf on Networked Systems Design and Implementation, p.1-16.

Sharma NK, Zhao CXY, Liu M, et al., 2020. Programmable calendar queues for high-speed packet scheduling. Proc 17th USENIX Conf on Networked Systems Design and Implementation, p.685-699.

Shieh A, Kandula S, Greenberg A, et al., 2011. Sharing the data center network. Proc 8th USENIX Conf on Networked Systems Design and Implementation, p.309-322.

Shrivastav V, 2019. Fast, scalable, and programmable packet scheduler in hardware. Proc ACM Special Interest Group on Data Communication, p.367-379. https://doi.org/10.1145/3341302.3342090

Sivaraman A, Subramanian S, Alizadeh M, et al., 2016. Programmable packet scheduling at line rate. Proc ACM SIGCOMM, p.44-57. https://doi.org/10.1145/2934872.2934899

Stoica I, Shenker S, Zhang H, 1998. Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high speed networks. Proc ACM SIGCOMM, p.118-130. https://doi.org/10.1145/285237.285273

Thapeta VS, Shinde K, Malekpourshahraki M, et al., 2021. Nimble: scalable TCP-friendly programmable in-network rate-limiting. Proc ACM SIGCOMM Symp on SDN Research, p.27-40. https://doi.org/10.1145/3482898.3483361

Wang JL, Jing LN, Chen X, et al., 2022. Programmable data processing method and system design for polymorphic network. *J Commun*, 43(4):14-25. https://doi.org/10.11959/j.issn.1000-436x.2022070

Wu XY, Wang Z, Wang WT, et al., 2023. Augmented queue: a scalable in-network abstraction for data center network sharing. Proc ACM SIGCOMM, p.305-318. https://doi.org/10.1145/3603269.3604858

Xia WF, Zhao P, Wen YG, et al., 2017. A survey on data center networking (DCN): infrastructure and operations. *IEEE Commun Surv Tutor*, 19(1):640-656. https://doi.org/10.1109/COMST.2016.2626784

Xylomenos G, Ververidis CN, Siris VA, et al., 2014. A survey of information-centric networking research. *IEEE Commun Surv Tutor*, 16(2):1024-1049. https://doi.org/10.1109/SURV.2013.070813.00063

Yang Y, Jiang HY, Wu YL, et al., 2021. C2QoS: CPU-cycle based network QoS strategy in vSwitch of public cloud. IFIP/IEEE Int Symp on Integrated Network Management, p.438-444.

Yu JZ, Wang XZ, 2014. POFSwitch v1.0. https://github.com/ProtocolObliviousForwarding/pofswitch [Accessed on Aug. 31, 2023].

Yu LC, Sonchack J, Liu V, 2022. Cebinae: scalable in-network fairness augmentation. Proc ACM SIGCOMM, p.219-232. https://doi.org/10.1145/3544216.3544240

Yu ZL, Hu CH, Wu JF, et al., 2021a. Programmable packet scheduling with a single queue. Proc ACM SIGCOMM, p.179-193. https://doi.org/10.1145/3452296.3472887

Yu ZL, Wu JF, Braverman V, et al., 2021b. Twenty years after: hierarchical core-stateless fair queueing. Proc 18th USENIX Symp on Networked Systems Design and Implementation, p.29-45.