Gearbox: A Hierarchical Packet Scheduler for Approximate Weighted Fair Queuing

Peixuan Gao
New York University

Anthony Dalleggio
New York University

Yang Xu *
Fudan University

H. Jonathan Chao
New York University

Abstract

Bandwidth allocation and performance isolation are crucial to achieving network virtualization and guaranteeing service quality in data centers as well as other network systems. Weighted Fair Queuing (WFQ) can achieve customized bandwidth allocation and flow isolation; however, its implementation in large-scale high-speed network systems is very challenging due to the high complexity of the scheduling and the large number of queues required.

This paper proposes Gearbox, a scheduler primitive for next-generation programmable switches and smart NICs that practically approximates WFQ. Gearbox consists of a logical hierarchy of queuing levels, which accommodate a wide range of packet departure times using a relatively small number of FIFOs. Gearbox's enqueue and dequeue operations have $O(1)$ time complexity, which makes it suitable to cope with high-speed line rates. Gearbox provides its simplicity and performance advantages by allowing slight discrepancies in packet departure time from strict WFQ. We show that Gearbox's normalized departure time discrepancy is bounded and has a negligible impact on bandwidth allocation and flow completion time (FCT).

We implement Gearbox in NS2 and in VHDL, targeted to a Xilinx Alveo U250 card with an XCVU13P FPGA. The NS2 evaluation results show that Gearbox closely approximates WFQ and achieves weighted max-min fairness in bandwidth allocation as well as flow isolation. Gearbox provides FCT performance comparable to ideal WFQ. The Gearbox FPGA prototype runs at 350MHz and achieves full line rate for 100GbE with packets larger than 123 bytes. Gearbox consumes less than 1% of the FPGA's logic resources and less than 4% of its internal block memory.

1 Introduction

Bandwidth allocation and isolation are key to network virtualization in data centers and the performance of network systems (e.g., FCT, packet tail latency, and QoS guarantee). WFQ is an ideal packet scheduling scheme that can achieve bandwidth guaranties and performance isolation, as well as other objectives, such as minimizing average FCT and reducing tail packet latency.

*Corresponding author

However, it is very challenging to implement a WFQ packet scheduler in large-scale high-speed network systems (e.g., systems with millions of active flows and link capacities of hundreds of Gbps). Although several hardware schedulers have been proposed in the past to sort or sequence packets based on their departure times, most of them do not scale well with system size [8] [9] [10] [4] [17], except for a few cutting-edge SoC chips [11] [5]. Push-in First-out (PIFO) [27] [28] is a viable solution but it does not scale easily due to the large number of parallel rank comparisons. More recent works such as AFQ [24] and PCQ [25] are based on the idea of calendar queues [7] [31] and have been implemented on emerging programmable switches. However, to provide the packet serving order of an ideal scheduler, AFQ requires a large number of queues while PCQ suffers from additional memory accesses due to frequent packet re-circulation and packet migration between queues.

We observe that by allowing a slight packet departure order skew in the WFQ, we have the opportunity to greatly simplify its implementation. We define departure time discrepancy (DTD) as the difference between the system time when a packet is served and its departure time assigned by WFQ and normalized DTD as DTD normalized to the expected delay [1]. In fact, packets with different expected delays in ideal WFQ scheduling have different tolerances to such DTD. Packets with longer expected delays can tolerate larger DTD than those with shorter expected delays. By taking advantage of this observation, we propose a new packet scheduler, called Gearbox, that approximates WFQ with a bounded normalized packet DTD. Gearbox adopts the idea of calendar queues [7] [31] and places packets with different expected delays in different logical levels of the calendar queues upon their arrival. When implemented physically, the queues are in fact individual FIFOs arranged into independent groupings. Gearbox can accommodate a very large range of departure times while using a relatively small number of FIFOs. The simplicity of Gearbox's approach makes it suitable for implementation on next-gen programmable switches as well as smart NICs which typically include FPGAs [15]. Note that by changing the 'departure time' in WFQ to other priority ranks calculated by other scheduling schemes (e.g., re-

[1] The departure time refers to the virtual departure time in WFQ [14] [22] [23]. The system time refers to the virtual system time in WFQ. When all the admitted flows are active, the virtual system time runs as fast as the real time.

maining flow size in pFabric [2]), Gearbox can approximate other scheduling schemes as a programmable scheduler. We summarize the contributions of this paper as follows:

- **Provide a close approximation of WFQ with bounded normalized DTD for packets.** Gearbox schedules packets with different expected delays using calendar queues with different granularities. Such tiered granularity allows Gearbox to closely approximate ideal WFQ with a relatively small number of FIFOs while guaranteeing a bounded normalized DTD.
- **Offer scalability with simple implementation.** Gearbox is implemented using queues (FIFOs) from a flat array that are arranged into logical levels. This non-hierarchical FIFO-based physical implementation allows Gearbox to admit a large number of packets and does not presume the existence of queue hierarchy in next-gen programmable switches [24] [25].
- **Implement Gearbox in NS2 with extensive packet-based evaluations.** We implement Gearbox in NS2 [21] and perform extensive packet-based evaluations. The results show that Gearbox closely approximates ideal WFQ and achieves a good weighted max-min fairness with per-flow isolation even in a short time scale. Our simulations based on a fat-tree topology show that Gearbox has an FCT performance closely matching the ideal WFQ using a PIFO.
- **Implement Gearbox in VHDL targeting an FPGA.** We implement a Gearbox VHDL prototype and target a Xilinx Alveo U250 FPGA card with a mid-speed grade XCVU13P FPGA. The Gearbox prototype runs at 350MHz and reaches full 100GbE line rate with packets larger than 123 bytes[2]. Based on the implementation report, Gearbox uses less than 1% of the target FPGA logic resources and less than 4% of its block random access memories (BRAM) as detailed further in Section 4.

The rest of the paper is organized as follows. Section 2 introduces the background and motivation of our work. Section 3 presents the detailed mechanism of the Gearbox scheduler and its extensions. Section 4 presents NS2 evaluation and the Gearbox hardware prototype. We discuss related works in Section 5 and conclude the paper in Section 6.

2 Background and Motivation

2.1 Weighted Fair Queuing

Bandwidth allocation and isolation are critical in data centers as well as other network systems [13] [18] [30] [3] [19]. These attributes are also essential to network performance such as max-min fairness, FCT, and tail latency [20].

WFQ [14] [22] [23] is the most effective algorithm to allocate bandwidth among flows and provide per-flow isolation. WFQ assigns a departure time to each packet and guarantees bandwidth allocation by scheduling packets in ascending order of their departure time[3]. For each packet scheduled by WFQ, its expected delay is its packet size divided by its assigned rate provided that the flow's traffic has been shaped. The expected delay is equal to the difference between the assigned departure time and the system time when the packet arrives.

2.2 Challenges of a WFQ Packet Scheduler

The major challenge of implementing WFQ on high-speed switches is the limited time to process each packet. For 64-byte packets, a scheduler for a 100GbE link has a processing time of only 6.72 ns[4]. In addition, sorting packets according to their departure time usually has a time complexity of $O(logN)$ [12], where N is the number of total packets or flows and could be on the order of several thousands. Maintaining a sorted list of packets in the limited packet processing time on high-speed links is very challenging.

An ASIC-based hardware sorter called Sequencer [8] [9] [10] inserts each arriving packet into a proper position in a queue according to its departure time. However, the Sequencer is not very scalable due to its high power consumption and chip area cost to support the parallel comparison of every arriving packet's departure time to all others' in the queue. The limited scalability of Sequencer precludes it from being used in a typical shared-memory switch in a data center [6] (e.g., with buffer size of $\sim$60K packets). The pipeline heap (P-heap) [4] [17] implements a WFQ scheduler with better scalability, but it is required for each output port, which results in significant chip area consumption [27], making a P-heap based WFQ scheduler on commodity switches less practical. Recent work on Pushed-in First-out (PIFO) [27] [28] performs parallel packet rank comparisons (similar to above-mentioned Scheduler), which limits its applicability in large high-speed switches.

A calendar queue, introduced by Randy Brown [7] and used in Timer Wheels [31] and used in Approximate Fair Queuing (AFQ) [24] and Programmable Calendar Queues (PCQ) [25], is scalable due to its implementation simplicity (only uses FIFOs). As a trade-off, calendar queues relax the packet sorting order compared to an ideal WFQ scheduler. Unlike in ideal WFQ, a calendar queue only sorts packets approximately in the ascending order of their departure times

[2]The VHDL design runs at 350 MHz in the target FPGA and performs enqueues and dequeues every 4 clock cycles, sustaining a packet rate of 87.5 Mpkts/sec. For Ethernet with a Preamble of 8 bytes and IFG of 12 bytes, the line bit rate for 123-byte packets is: $87.5M \times (8+123+12) \times 8 \approx 100Gb/s$.

[3]In a WFQ scheduler, the packet with the smallest departure time leaves the scheduler first.

[4]It is possible to reduce the scheduling rate by using input queuing to combine small packets into a single larger packet. E.g., two 64-byte packets belonging to the same flow can be scheduled as a single 128-byte packet.

since its accuracy is limited by the granularity of the sorting buckets. Moreover, calendar queues also suffer from calendar range overflow when the available number of queues cannot accommodate the wide range of departure times. To address the range overflow problem, modifications to calendar queues have been proposed to accept packets with larger departure times instead of dropping them [7][25]. These modifications consist of recirculating out-of-range packets to the end of the queues to delay them until they reach their departure times. When implemented on high-speed switches, calendar queue packet schedulers with packet recirculation impose additional demands on memory bandwidth due to the accesses for both incoming and recirculated packets. We discuss this further in Section 2.4.

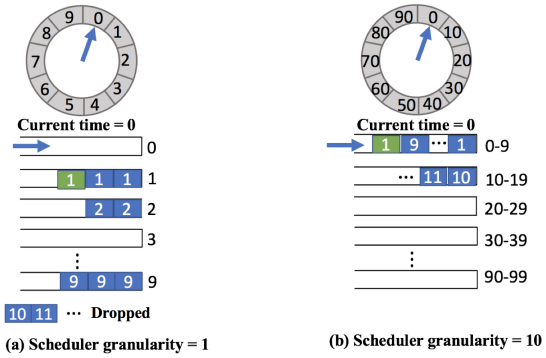


Figure 1: Different granularities

2.3 Granularity of the Scheduler and Departure Time Discrepancy

The calendar queue scheduler trades some packet serving order accuracy for simplicity and scalability. Rather than sorting packets perfectly in the ascending order of their departure times, the calendar queue scheduler only sorts packets approximately compared to ideal WFQ. This approximation is due to the FIFO-based structure of the calendar queue [7]: a calendar queue only sorts packets by placing them into different FIFOs or buckets. Each FIFO in a calendar queue cannot differentiate the departure times of the packets in it. Here we quantify the scheduling precision of a calendar queue scheduler as the 'granularity of the scheduler'.

Granularity of the scheduler: the minimal departure time difference that a scheduler can discriminate, noted as g.

Consider the example in Figure 1. For the scheduler in Figure 1(a), each queue represents one virtual time unit and the scheduler can discriminate between the departure times with a difference of 1, which means it has a granularity $g = 1$. The scheduler in Figure 1(b) applies a coarser granularity $g = 10$. This scheduler can only discriminate between packet departure times based on the tens digit of their departure time.

With tiered granularities, calendar queue schedulers sort packets into different approximate orders when compared with ideal WFQ. We formally define the concept of 'departure time discrepancy (DTD)' to quantify the difference between the approximate serving order and that of ideal WFQ.

Departure time discrepancy (DTD): the difference between the system time when a packet leaves the scheduler and its departure time as scheduled by WFQ. We denote the k^{th} packet in flow i as $P_{(i,k)}$ and its DTD as $d_{(i,k)}$.

$$d_{(i,k)} = \begin{cases} D_{(i,k)} - F_{(i,k)} & , D_{(i,k)} > F_{(i,k)} \\ 0 & , \text{otherwise} \end{cases} \quad (1)$$

Here $F_{(i,k)}$ is the ideal departure time as calculated by the WFQ scheduler for $P_{(i,k)}$ and $D_{(i,k)}$ is the actual departure time of packet $P_{(i,k)}$.

DTD shows the difference of the packet serving order between a calendar queue scheduler and an ideal WFQ. It indicates how well a calendar queue scheduler approximates an ideal WFQ from a packet's perspective. A small DTD indicates that the scheduler approximates an ideal WFQ closely and a large DTD, that a packet may experience a delay larger than its expected delay, which further leads to increased FCT or other consequences.

For a calendar-queue-based scheduler with a limited number of queues and fixed granularity [24] [25], different levels of granularity have pros and cons. Schedulers with finer granularity provide smaller DTDs. However, they only accommodate a narrow range of departure times and are therefore more prone to calendar range overflow. A rotating calendar scheduler consisting of M queues covers only $M * g$ future virtual time units. Packets with departure times beyond this bound will be dropped due to calendar range overflow, as shown in Figure 1(a). With a small number of available queues, we can consider the scheduler as having a shallow buffer, which is easy to overflow. For services requiring a certain amount of burstiness tolerance, this shallow buffer could lead to frequent packet drops. Furthermore, flows assigned relatively low bandwidths, which lead to large departure times, could experience excessive packet drops. This makes such fine-grained schedulers unsuitable to handle large variations in weighted bandwidth allocation. Schedulers with a coarser granularity alleviate the calendar range overflow issue by covering a wider range of departure times in the future. However, one obvious downside of using a coarse granularity is larger DTDs. Consequently, packets in the same queue may not be scheduled according to the order of their departure times. For example, in Figure 1(b) the green packet departs after packets with larger departure times in the coarse-grained scheduler.

As shown in the examples in Figure 1, scheduler granularity leads to a trade-off between fewer calendar range overflows and smaller DTDs.

2.4 Normalized DTD

How do we determine an appropriate granularity for the scheduler? One could argue that it is always better to have a scheduler with finer granularity. But is it necessary to achieve such a small DTD for all packets?

If we need to maintain a fine granularity while covering a wide departure time range, the total number of queues required would become very large. This design approach conflicts with the goal of a simple implementation. The time complexity of managing thousands of queues is very high and most switches may not have the required number of queues.

Another approach for providing fine scheduler granularity while preventing calendar range overflow is packet recirculation. The classic calendar queue [7] and Timer Wheel [31] place all packets into their queues. Packets scheduled too far in the future are simply enqueued in a queue roughly determined by the departure time modulo the total number of queues, and recirculated before their departure times become due. This will lead to additional memory bandwidth consumption, where memory bandwidth is very often the limiting factor in a networking equipment [32] [16]. Excessive packet recirculation can use up valuable shared memory bandwidth and lead to a significant drop in switch throughput. A recent work, PCQ [25], implements an alternative packet recirculation scheme. PCQ arranges calendar queues in multiple levels with different granularities. When PCQ finishes serving all the packets in the lower level, it recirculates and deposits all packets from a head queue in the higher level to appropriate queues in the lower level. Such a packet recirculation scheme can still lead to throughput reduction on high-speed switches.

Based on the above analysis, providing fine granularity with a wide range of packet departure times is resource intensive. But is it really necessary? We observe that packets have different tolerances to DTDs depending on their expected delays upon arrival as mentioned in Section 2.1. Packets with a smaller expected delay usually expect to be served shortly and are sensitive to small differences in the scheduling order. For these packets, a fine-grained scheduler is necessary to guarantee a small DTD. However, packets with larger departure times upon arrival can tolerate larger DTDs. These packets usually belong to flows with low assigned bandwidths: WFQ assigns these packets with large departure times to achieve bandwidth weighted max-min fairness in the long run. For these packets, their large expected delay makes it unnecessary to schedule them with fine granularity.

According to the above analysis, it is more appropriate to consider a packet's DTD based on its tolerance to it. We therefore normalize a packet's DTD to its expected delay.

Normalized DTD: the DTD normalized to the packet's expected delay, noted as $d_{n(i,k)}$.

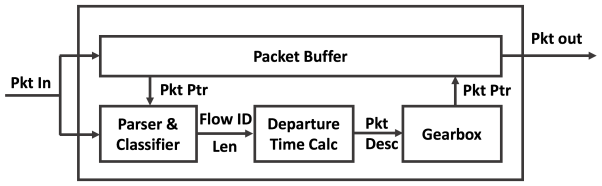


Figure 2: Gearbox System-level Application

$$d_{n(i,k)} = \begin{cases} \frac{D_{(i,k)} - F_{(i,k)}}{F_{(i,k)} - A_{(i,k)}} & , D_{(i,k)} > F_{(i,k)} \\ 0 & , \text{otherwise} \end{cases} \quad (2)$$

Here $A_{(i,k)}$ is the system time t_s when packet $P_{(i,k)}$ arrives at the scheduler, $F_{(i,k)}$ is the ideal departure time for packet $P_{(i,k)}$ determined by the WFQ scheduler, and $D_{(i,k)}$ is the actual departure time of packet $P_{(i,k)}$.

For example, say that packet $P_{(A,1)}$ and packet $P_{(B,1)}$ arrive at the scheduler when system time $t_s = 10$. Packet $P_{(A,1)}$ has an ideal departure time $F_{(A,1)} = 11$ and packet $P_{(B,1)}$ has an ideal departure time $F_{(B,1)} = 91$. Assume the scheduler applies a coarse granularity $g = 10$. Due to the coarse granularity, packet $P_{(A,1)}$ leaves the scheduler at $t_s = 19$ and packet $P_{(B,1)}$ leaves the scheduler at $t_s = 99$. Both of the packets have a DTD of 8. Although the two packets have the same DTD, they have different tolerances to it. The expected delay of packet $P_{(A,1)}$ and $P_{(B,1)}$ is $11-10=1$ and $91-10=81$ respectively. When we evaluate their normalized DTD, we have $d_{n(A,1)} = 8/1 = 8$ and $d_{n(B,1)} = 8/81 \approx 0.1$. This means packet $P_{(A,1)}$ is experiencing a delay that is 8 times its expected delay while packet $P_{(B,1)}$ has a delay only 1.1 times its expected delay. This indicates such granularity is appropriate for packet $P_{(B,1)}$ but is too coarse for packet $P_{(A,1)}$. This example shows that it is difficult to find an appropriate fixed granularity to schedule packets with different expected delays while satisfying DTD bounds.

3 Gearbox: Hierarchical Packet Scheduler

3.1 Basic Idea of Gearbox

Based on our analysis in Section 2.4, it is more appropriate to guarantee different DTD bounds for packets with different expected delays. In this case, we need a scheduler with flexible granularity to serve different packets. Thus, the scheduler must guarantee a low normalized DTD while keeping the implementation simple. We introduce Gearbox, a hierarchical packet scheduler that closely approximates WFQ and is simple to implement.

Figure 2 shows a simplified system-level application of Gearbox. Incoming packets are stored in the Packet Buffer. The packet header is sent to the packet Parser & Classifier to identify the flow (e.g., using 5-tuple classification) and determine the packet length. The Departure Time Calculator com-

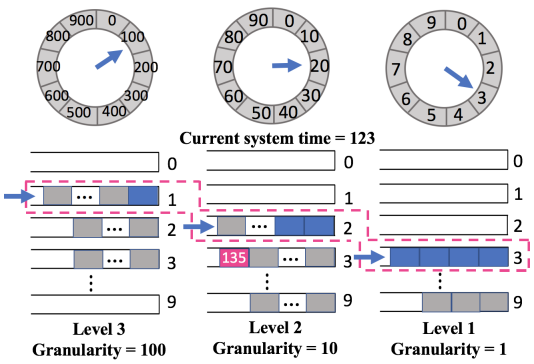


Figure 3: Gearbox: Hierarchical FIFO-based scheduler

Table 1: TERMS AND NOTATIONS

Notation	Description
$P_{(i,k)}$	k^{th} packet in flow i
$F_{(i,k)}$	Departure time of packet $P_{(i,k)}$ assigned by WFQ
$A_{(i,k)}$	System time when packet $P_{(i,k)}$ arrives at the scheduler
$D_{(i,k)}$	System time when packet $P_{(i,k)}$ leaves the scheduler
$d_{(i,k)}$	Departure time discrepancy (DTD) of packet $P_{(i,k)}$
$d_n(i,k)$	Normalized DTD of packet $P_{(i,k)}$
t_s	Current system virtual time
L	Total levels of the scheduler
M_l	Total number of FIFOs at level l
g_l	Granularity of level l
$Q_{(l,f)}$	FIFO f at level l

putes the packet's departure time and forwards it along with the packet descriptor (length, departure time, and enqueue Packet Pointer from Packet Buffer) to Gearbox to enqueue the packet. Upon request from a controller (not shown) Gearbox dequeues stored packet descriptors according to a calendar order described in detail below and forwards the descriptors to the Packet Buffer that outputs the packets. We further present the smart NIC use case and multi-pipeline use case of Gearbox in Appendix B.

Gearbox arranges FIFO-based calendar queues into 'logical levels' with different granularities, achieving the benefits of calendar queue schedulers with different granularities to serve packets with different expected delays. Lower logical levels with finer granularity provide smaller DTD for packets with departure times that are close to current system time upon arrival. Higher logical levels with coarser granularity serve packets with large departure times and cover a wider departure time range, reducing packet loss due to calendar range overflow. By providing tiered scheduling granularity, Gearbox guarantees a low normalized DTD for all packets with a relatively small number of queues. Moreover, Gearbox eliminates packet recirculation by directly serving packets at all levels based on our 'Compound FIFO' concept introduced in Section 3.2. This eliminates the additional memory accesses due to recirculation and allows Gearbox to achieve a high packet processing rate efficiently.

Figure 3 shows an example of the Gearbox hierarchical scheduler. Consider a scheduler with 30 available queues. Gearbox arranges them into 3 logical levels with different granularities, each logical level containing 10 queues. Level 1 has the finest granularity $g_1 = 1$, level 2 has a coarser granularity, $g_2 = 10$ and level 3 has the coarsest granularity, $g_3 = 100$. In this example, packets with a departure time within 10 virtual time units in the future upon arrival are enqueued in level 1 and are scheduled with the granularity of $g = 1$. Packets with a departure time larger than 10 but smaller than 100 virtual time units in the future upon arrival are enqueued in level 2. Other packets with departure time between 100 and 1000 virtual time units in the future upon arrival are enqueued in level 3.

Note that the 'levels' in Gearbox are logical concepts. When implemented physically, they are in fact individual queues (FIFOs) arranged into independent groupings, which means Gearbox only requires a single level of queues in the hardware. We'll further discuss this in Sections 3.2 and 3.3. As a result, Gearbox can be implemented on devices that do not support hierarchy.

In section 3.4, we prove that Gearbox guarantees a low normalized DTD for all packets. Our evaluation shows that Gearbox closely approximates ideal WFQ from a user's point of view. Gearbox can be thought of as a clock. The lower logical level with finer granularity is like the second hand, serving packets in units of seconds. The higher logical levels are like the minute hand or the hour hand, admitting more packets with a larger departure time and serving them with a coarser granularity. Each logical level of the scheduler cooperates to schedule packets with different granularities, which makes the scheduler just like a gearbox shifting between different gears.

To better illustrate the detailed schemes in Gearbox, we summarize the related concepts and notations in Table 1. We further generalize the multi-level architecture of the Gearbox scheduler. The scheduler contains L logical levels of calendar queues, each one with a different granularity g_l. Each logical level of the scheduler contains M_l FIFOs and covers a departure time range of $[t_s, t_s + (M_l * g_l))$.

3.2 Compound FIFO

Researchers have tried to arrange the calendar queues in hierarchical structures, however, their implementations are not suitable for the ultra-high-speed switches due to packet recirculation and the short packet processing time.

To eliminate packet re-circulation, Gearbox applies the concept of a 'Compound FIFO' to directly schedule packets in different levels efficiently. As Figure 3 shows, the compound FIFO consists of the current serving FIFO in each level. In the example, the compound FIFO consists of $FIFO1$ at level 3, $FIFO2$ at level 2 and $FIFO3$ at level 1,

representing the current system time $t_s = 123$. Note that all the FIFOs in the compound FIFO cover the current system time t_s, which means that the individual FIFOs that make up the compound FIFO possibly contain packets that need to be served at the current system time.

Gearbox serves packets in the compound FIFO using a factor that is inversely proportional to the granularity of the level that each queue belongs to. Gearbox serves packets from the lowest-level queue to the highest-level queue in the compound FIFO. Since the lowest-level has the finest granularity, the departure time of all the packets in this queue equals the current system time t_s. Gearbox serves all the packets in the queue in the lowest level. After draining the packets in the lowest-level queue, Gearbox starts to serve the queue at the next higher level. At each of the higher levels, we serve a number of bytes (rounded up to a whole number of packets) that is inversely proportional to the level's granularity g_l [5]. Gearbox finishes serving the compound FIFO when it finishes serving all the queues within it according to their levels. After serving the compound FIFO, Gearbox updates current system time t_s to the next non-empty compound FIFO.

As an illustration of serving packets from the different levels, the FIFOs in the pink box in Figure 3 are dequeued as follows. Gearbox first drains all the packets in $FIFO3$ at level 1. After that, since level 2 has the granularity of $g_2 = 10$ and $FIFO2$ at level 2 covers the time range from 120 to 129, Gearbox serves $1/10$ packets in this queue. Likewise, Gearbox serves $1/100$ of the packets in $FIFO1$ at level 3. After serving the correct proportion of packets in the queue at each level, Gearbox increases its current system time t_s by 1 and updates the compound FIFO based on the new system time $t_s = 124$.

The key contribution of the 'Compound FIFO' is freeing the scheduler from packet recirculation. By directly serving queues in different levels using a factor that inversely proportional to their granularity, Gearbox eliminates the need to recirculate packets. This means Gearbox's dequeue process is as simple as popping from a FIFO. This allows Gearbox to easily achieve a high packet processing rates on core switches. The trade-off is increased DTD in the higher levels.

3.3 Enqueue and Dequeue Processes

Enqueue Process The Gearbox packet enqueue process is straightforward. To enqueue a packet, Gearbox needs to determine the correct level and the destination FIFO for the arriving packet. Gearbox first finds the level to enqueue using the difference between the current system time t_s and the departure time of the packet $F_{(i,k)}$. The packet will enqueue the lowest possible level that covers this interval. After finding the enqueue level, Gearbox simply divides the interval mentioned above by granularity g_l at this level to find the corresponding FIFO to enqueue.

Figure 3 shows an example of Gearbox's enqueue process. In the example, Gearbox has 3 levels covering the departure time range $[t_s, t_s + 9]$, $[t_s, t_s + 99]$, $[t_s, t_s + 999]$, respectively. The pink packet with a departure time of 135, arrives at the scheduler when $t_s = 123$. Upon the packet's arrival, the scheduler calculates the interval $F_{(i,k)} - t_s = 12$. This interval falls into the virtual time range covered by level 2. When we divide the interval 12 by the granularity g_2, we have $\lfloor 12/10 \rfloor = 1$, which indicates that this packet is enqueued in the FIFO next to the current serving FIFO in this level. The enqueue process is summarized in Algorithm 1.

Algorithm 1 Enqueue Process

```
1: function ENQUEUE PACKET(P_(i,k))
2:     for level l from 1 to L do
3:         if ⌈(F_(i,k) − t_s)/g_l⌉ ≤ M_l then
4:             f = ⌊(F_(i,k) − t_s)/g_l⌋
5:             P_(i,k) enqueue f^th FIFO following the
               current serving FIFO
6:             return
7:     Drop packet P_(i,k)
```

Dequeue Process As we have introduced the compound FIFO in *Section 3.2*, the dequeue process of Gearbox was described above as serving the compound FIFO. We generalize Gearbox's dequeue process with the pseudo-code in Algorithm 2, where f_l is the FIFO of level l in the compound FIFO

Algorithm 2 Dequeue Process

```
1: function DEQUEUE PROCESS
2:     for level l from 1 to L do
3:         if Q_(l,f_l) is not empty then
4:             dequeue Q_(l,f_l) up to Size(Q_(l,f_l))/g_l
```

3.4 Normalized DTD Analysis

We previously defined the normalized DTD $d_{n(i,k)}$ as the DTD normalized to the packet's expected delay in equation (2). Now we need to determine its bound in Gearbox. Based on the above expression, the maximum value of $d_{n(i,k)}$ occurs when DTD is at the maximum value and the expected delay is at the minimum value. For a packet at level l, the maximum DTD equals the level's granularity g_l. Then we have:

$$\max\{D_{(i,k)} - F_{(i,k)}\} = g_l \tag{3}$$

[5] In the VHDL implementation the granularity g_l of each level is a power of 2, to implement the inverse proportional calculation using bit shifting.

We need to find the minimal expected packet delay possible at level l. Based on the enqueue process described in section 3.3, packets enqueue at different levels based on their expected delay. Only packets with an expected delay between g_l and $g_l * M_l$ will enqueue at level l. Therefore, we have minimal expected delay:

$$\min\{F_{(i,k)} - A_{(i,k)}\} = g_l \tag{4}$$

From equation (3) and (4), we have maximum delay:

$$\max\{D_{(i,k)} - A_{(i,k)}\} = 2g_l \tag{5}$$

Then from (4) and (5) we have maximum normalized DTD:

$$\max\{d_{n(i,k)}\} = \max\{\frac{D_{(i,k)} - F_{(i,k)}}{F_{(i,k)} - A_{(i,k)}}\} = \frac{2g_l - g_l}{g_l} = 1 \tag{6}$$

The maximum normalized DTD has an upper bound of 1, which means, in the worst case, that a packet scheduled by Gearbox will have a maximum delay that is twice its expected delay in WFQ. In other words, a packet that expects t microseconds delay in a WFQ scheduler will experience at most $2t$ microseconds delay in Gearbox in the worst case[6]. We evaluate the normalized DTD in actual time using simulations and provide our evaluation results in *Section 4*.

3.5 The Packet Out-of-order Issue

Packets in the hierarchical calendar queue scheduler may experience a packet out-of-order issue. According to the architecture of Gearbox, FIFOs at different levels might cover an overlapping virtual time range. As a result, a packet $P_{(i,k)}$ with a departure time $F_{(i,k)}$ may enqueue at any level of the scheduler, depending on its arrival time $A_{(i,k)}$. Similarly, a subsequent packet $P_{(i,k+1)}$ from the same flow may enqueue at a different level. When two packets from the same flow enqueue in different levels in the scheduler, the relative order of the packets is not deterministic. It is possible that packet $P_{(i,k)}$ leave later than the subsequent packet $P_{(i,k+1)}$ if it is enqueued at a different level in the scheduler, causing the packets to be dequeued out of order.

Figure 4 shows an example of the packet out-of-order issue. In the figure, packets $A1$ and $A2$ arrive at the scheduler when $t_s = 0$. Since both packets have a departure time exceeding $t_s + 9$, they cannot be enqueued at level 1 and are thus enqueued in $FIFO1$ at level 2. Later, the system time t_s updates to $t_s = 10$. A new packet $A3$ from the same flow arrives at the scheduler with a departure time of $F_{(A,3)} = 12$. At this moment, $F_{(A,3)} - t_s = 2 < 10$ and the scheduler places $A3$ in $FIFO2$ at level 1. At this point, packets from flow A

[6]The delay discrepancy may be smaller than 2x if all the flows are not active and the packet has an earlier opportunity to be dequeued

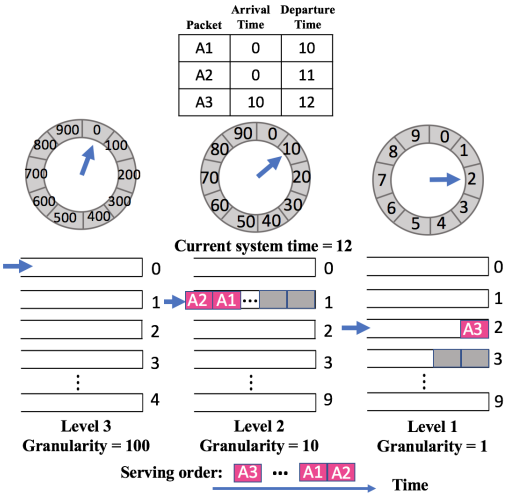


Figure 4: The packet out-of-order issue

are placed at different levels. As the scheduler starts to serve packet $A3$ when $t_s = 12$, packets $A1$ and $A2$ are still queued up at the tail of $FIFO1$ at level 2. In this case, packet $A3$ leaves before packets $A1$ and $A2$.

3.6 Solution to the Packet Out-of-order Issue

We introduce two modifications to eliminate the uncertainty in the packet serving order. First, we do not 'wrap around' FIFOs in the same level. This means that after the scheduler drains a FIFO and starts to serve the next one, the drained FIFO does not enqueue any packets until the scheduler finishes serving all the packets in the entire level. Second, we track the 'last packet enqueued level' for each flow, noted as L_i, where i is the flow id. When a new packet $P_{(i,k)}$ arrives at the scheduler, it is enqueued in a level equal to or higher than level L_i.

Let's consider the same example in Figure 4 after applying the solution. When packet $A1$ and $A2$ enqueue level 2 of the scheduler, Gearbox marks flow A's last packet enqueue level as $L_A = 2$. When packet $A3$ arrives at the scheduler, although $F_{(A,3)} - t_s = 2 < 10$, the scheduler will only place packet $A3$ into a level equal or larger than $L_A = 2$. In this case, Gearbox places packet $A3$ in $FIFO1$ at level 2 right after packet $A1$ and $A2$. As a result, it is scheduled after its preceding packets $A1$ and $A2$. Consequently, these modifications eliminate the packet out-of-order issue.

The modifications mentioned above may lead to side effects that could potentially increase the DTD of packets. The flows with input rates higher than their allocated rates will enqueue and stay in a higher level, where they will suffer from a coarser granularity and higher DTD. We further introduce the 'Step-down FIFO', an extension of Gearbox to solve these side effects. We present the details of this extension in Appendix A.

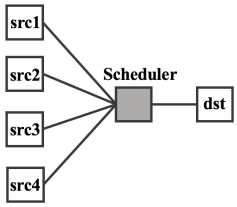


Figure 5: Single-node topology

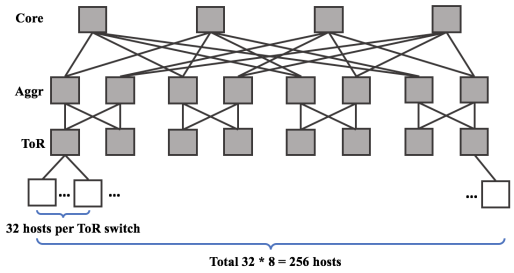


Figure 6: Three-level fat-tree topology

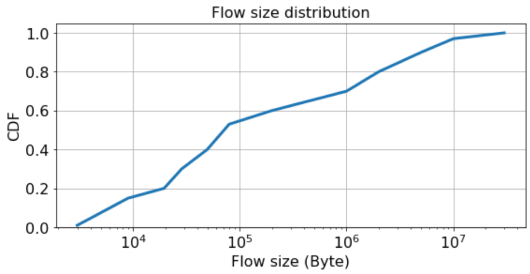


Figure 7: Web-search flow size distribution

4 Implementation and Evaluation

In this section, we describe the design and implementation of Gearbox in NS2, a packet-based simulator [21], and in VHDL as a hardware prototype (targeted to Xilinx's Alveo U250 card[34]), along with the extensive simulations to evaluate the performance of Gearbox. The two implementations and their associated verification environments enabled us to explore different aspects of Gearbox including its performance in networks, the performance of the hardware prototype, and the hardware resource utilization.

4.1 Packet-based Evaluation

To evaluate Gearbox's performance in a large-scale network topology with real-world data traffic, we implemented Gearbox in NS2 [21] and conducted extensive packet-based simulations.

4.1.1 Evaluation Setup

Network topology We set up two different network topologies in NS2: (1) a single-node star topology for bandwidth allocation and fairness evaluation, and (2) a fat-tree topology to evaluate FCT and normalized DTD.

For the single-node topology, we connect five servers to a switch as shown in Figure 5. All the links have equal bandwidth of 10 Gbps and a delay of $3\mu s$. We later use this simple star topology to form a classic incast traffic pattern to observe the bandwidth share of individual flows and evaluate the fairness of the scheduler.

We built a three-level fat-tree topology for large-scale simulation. As shown in Figure 6, there are 4 Core switches, 8 Aggregation switches, 8 Top-of-Rack (ToR) switches and 256 servers. The links between servers and ToR switches are 10Gbps with 10 ns delay. Other links have a bandwidth of 40Gbps and a $1\mu s$ propagation delay. We apply Gearbox and other scheduler schemes on every ToR/Aggregation/Core switch to evaluate the FCT performance.

Traffic loads We generate empirical traffic workloads based on the datacenter flow size distribution from an operational datacenter that supports web-search service [2]. The traffic follows a heavy tail distribution as Figure 7 shows. The flows arriving in a Poisson process with different arrival intervals result in different traffic loads. Each flow randomly selects the source and the destination hosts in the topology in a uniform distribution.

Alternative approaches We compare Gearbox with single-level calendar queues [24]. All the schedulers in our evaluation have 56 FIFOs and are evaluated with different granularities as discussed in *section 2.3*. Table 2 provides details of the Gearbox and calendar queue schedulers. We also compared Gearbox with an ideal PIFO-based WFQ scheduler as well as a simple drop tail queue. In our packet-based simulation, all packets have the same size of 1,500 bytes [7] and the number of bytes per virtual time unit is set to 750 bytes (the finest supported granularity).

4.1.2 Single-node Microbenchmark

Gearbox can reach good max-min fairness We observe that Gearbox can reach satisfactory max-min fairness in bandwidth allocation, close to that of PIFO-based WFQ. In

[7] In NS2 packet based simulations, packet size is not a factor that affects switch performance.

Table 2: PACKET SCHEDULER SET UP

Packet Scheduler	Granularity
Gearbox	$g_1 = 1, g_2 = 8, g_3 = 64$
CQ-1	$g = 1$
CQ-10	$g = 10$
CQ-100	$g = 100$

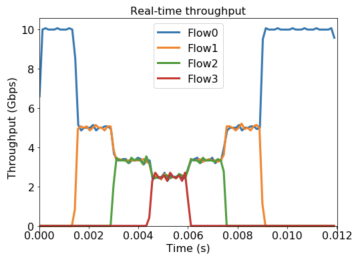


Figure 8: Real-time throughput of Gearbox

our single-node microbenchmark evaluation, we used four TCP flows, each sending traffic to one destination node with different starting times and ending times. Based on the results shown in Figure 8, Gearbox allows the TCP flows to reach a bandwidth max-min fairness with good flow isolation. When a new flow joins, Gearbox quickly adjusts the bandwidth allocated to all the active flows and reaches the max-min fairness. After a flow ends, Gearbox also allows the remaining flows to ramp up quickly.

Normalized Fairness Metric To better evaluate fairness in bandwidth allocation, we use the Normalized Fairness Metric (NFM). Shreedhar and Varghese first measured fairness in bandwidth allocation using the Fairness Metric (FM) [26]. FM reflects the maximum difference in the number of served bytes between two flows during a time period. By definition, FM's values vary significantly based on the bandwidth assigned to each flow and the total shared bandwidth.

To normalize the influence of this factor, Brent Stephens introduced a better metric: the Normalized Fairness Metric (NFM) [29]. NFM normalizes the Fairness Metric according to the number of bytes that each flow should serve. In a scenario where 4 flows are sharing a link of 4 Mbps with equal weights, each flow is assigned a bandwidth of 1 Mbps. During 1 second, each flow should have 128 kbytes of data served. If the measured Fairness Metric $FM(1sec) = 32kbytes$, then the Normalized Fairness Metric $NFM(1sec) = 32kbytes/128kbytes = 0.25$.

Gearbox has a good NFM even for short time scales. By definition, a lower NFM indicates better fairness in bandwidth allocation. In our evaluation, we observed the NFM of flows with 4 different weight sets (shown in Table 3) in different time scales. From the results shown in Figure 9, Gearbox outperforms coarse-grained calendar queues with different bandwidth allocations. When flows are assigned with weights that differ significantly, the fine-grained calendar queues suffer from packet loss while Gearbox maintains good fairness performance. From the perspective of max-min fairness, Gearbox's performance matches closely that of PIFO-based WFQ.

Table 3: Flow Weight Sets

Weight Set 1	1 : 1 : 1 : 1
Weight Set 2	2 : 2 : 1 : 1
Weight Set 3	50 : 50 : 1 : 1
Weight Set 4	100 : 100 : 1 : 1

4.1.3 Large-scale Simulation

We extend our simulation to a three-level fat-tree topology with more servers and higher link rates as described in section 4.1.1. We focus on the normalized FCT performance of different size flows. [8]

Figure 10(a) shows the average normalized FCT across different traffic loads and Figures 10(b) and 10(c) show the average normalized FCT of different size flow groups under 70% and 90% load, respectively. The 95^{th} percentile normalized FCT under various traffic loads is shown in Figure 10(d). Figures 10(e) and 10(f) show the 95^{th} percentile normalized FCT broken down per flow size under 70% and 90% traffic load, respectively.

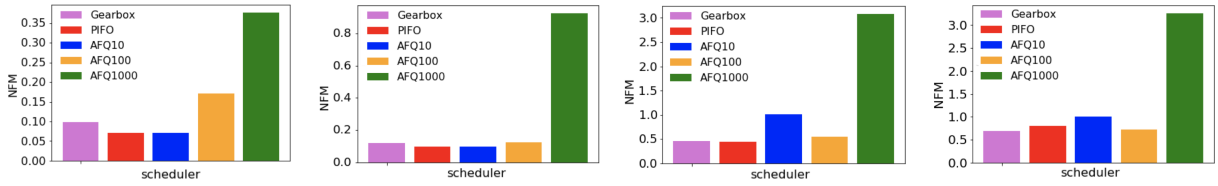
Short flows benefit from low DTD Gearbox closely approximates WFQ and provides per-flow isolation, which results in low DTD, a key factor for short flow FCT performance. With WFQ, different flows are isolated from each other and packets from large flows will not block packets from small flows. As we discussed in *section 2.3*, the lower level of Gearbox provides a fine scheduling granularity, guaranteeing that packets from short flows depart according to their departure times without a large discrepancy. Figures 10(b), 10(c), 10(e) and 10(f) show that short flows that are less than 80 kbytes have a small normalized FCT. According to Figure 10, Gearbox can achieve a low normalized FCT close to ideal PIFO-based WFQ and the calendar queue with the finest granularity. On the other hand, packets from short flows would suffer a large DTD in the coarse-grained calendar queues and the drop tail queues.

Short flows consist only of a few packets. Therefore, DTD can cause delays that has a negative effect on their FCT. To further observe the delay of short flows under different scheduler schemes, we measure the average end-to-end delay. As shown in Figure 11, the extra delay in the coarse-grained calendar queues leads to a large normalized FCT for short flows. In contrast, Gearbox has a low delay that is close to that of PIFO-based WFQ.

As stated in *section 3.4*, Gearbox guarantees a low normalized DTD for all packets. We evaluated the normalized DTD of Gearbox [9] as shown in Figure 12. The normalized delay shown in the figure is the actual delay normalized to

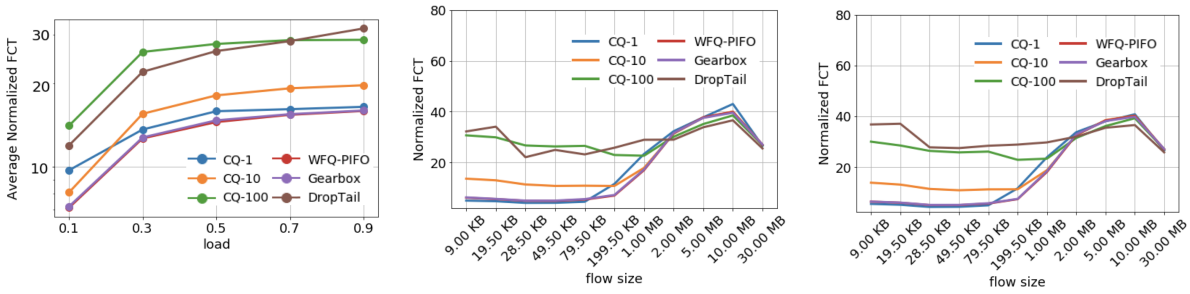
[8] "Normalized FCT" means a flow's actual FCT normalized to its ideal FCT when no other flows are active in the network.

[9] Normalized delay was measured in actual (not virtual) time

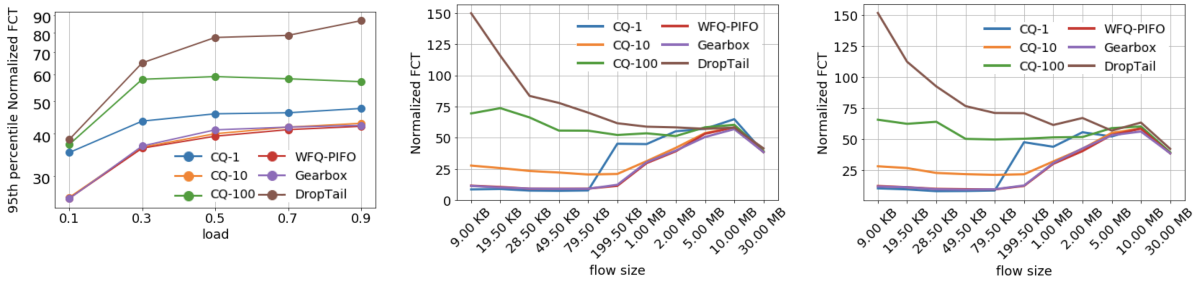


(a) NFM, 48 μs, weight set 1 (b) NFM, 48 μs, weight set 2 (c) NFM, 120 μs, weight set 3 (d) NFM, 120 μs, weight set 4

Figure 9: Normalized Fairness Metric



(a) Average normalized FCT (b) Breakdown normalized FCT, 70% load (c) Breakdown normalized FCT, 90% load



(d) 95^{th} percentile normalized FCT (e) $95th$ percentile normalized FCT - per flow size, 70% load (f) $95th$ percentile normalized FCT - per flow size, 90% load

Figure 10: Normalized FCT in fat-tree topology

the delay of the ideal WFQ using a PIFO. In Figure 12, the red dashed line represents the normalized delay with a value of 1, which indicates that the delay is equal to that of ideal WFQ. The simulation results show that Gearbox has a satisfactory normalized delay that is close to 1 for flows with different sizes, indicating that Gearbox has a delay performance closely matching that of ideal WFQ.

Large flows benefit from low packet loss rate Gearbox not only provides a small normalized FCT for short flows, but it also reduces packet loss and re-transmission for large flows. Thanks to its higher levels, Gearbox can schedule packets with large departure times with a very low drop rate.

Figure 10 shows Gearbox can achieve good normalized FCT performance for mid-sized flows around 200 kbytes or larger. We further measured the average packet loss of different flow groups. As shown in Figure 13, the single-level calendar queues with fine granularity drop packets frequently for the mid-sized and large flows, triggering a large number of re-transmissions and leading to a high normalized FCT. However, Gearbox and other coarse-grained calendar queues have low packet loss rates that are close to zero. Gearbox reduces packet loss related to calendar range overflows and guarantees a low normalized FCT for mid-sized and large flows.

As shown in the above simulation results, Gearbox combines the benefits of both fine and coarse calendar queue granularities. Consequently, Gearbox provides satisfactory normalized FCT performance for flows with different sizes as discussed in *Section 3*. The evaluation results show that Gearbox provides performance comparable to PIFO-based WFQ.

4.2 Hardware Prototype Design

Overview We implemented Gearbox in VHDL with multiple parameters (generics) for easy scalability including:

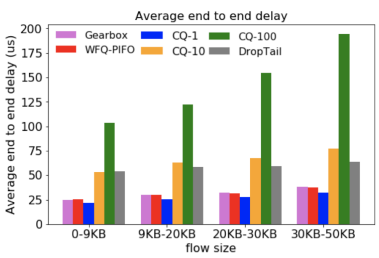


Figure 11: Average end-to-end delay of short flows

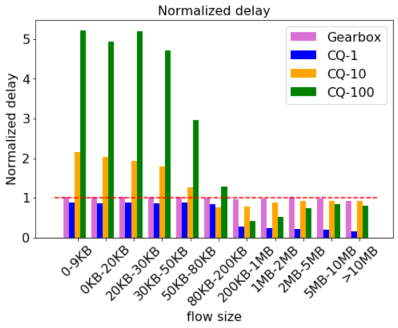


Figure 12: 99^{th} percentile normalized delay

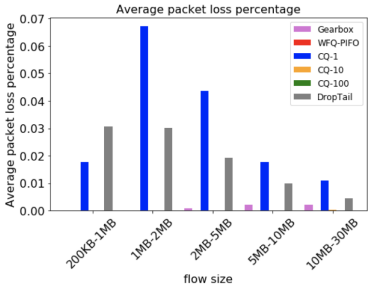


Figure 13: Average packet loss percentage

- Number of levels
- Number of FIFOs per level
- Number of flows
- Other sizing parameters for memories and logic

The VHDL code, test bench, and FPGA implementation files are available at `https://github.com/Gearbox-NSDI/Gearbox_NSDI`

Hardware prototype architecture A high-level block diagram of the VHDL implementation of Gearbox is shown in Figure 14.

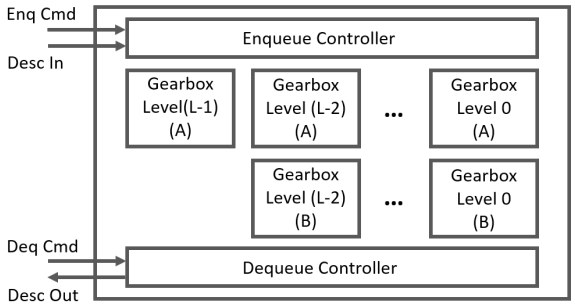


Figure 14: Gearbox Block Diagram - VHDL Implementation

At the top level, Gearbox consists of a parameterized number of instances of the Gearbox Level sub-block along with the Enqueue and Dequeue Controller blocks. Each level except the highest consists of two Gearbox Level sub-blocks denoted (A) and (B). The A and B sub-blocks form a ping-pong scheme to guarantee access to a full set of FIFOs (corresponding to virtual time units) while maintaining packet order. The highest level consists only of a single Gearbox Level sub-block because a wraparound of the FIFO index does not cause out-of-order packets.

The Gearbox Level sub-block shown in Figure 15 consists of a parameterized number of FIFOs along with Enqueue and Dequeue logic blocks.

The enqueue and dequeue operations are described below. The packet descriptor is formatted as follows:

Packet Pointer (15)	Address of packet in packet buffer
Packet Length (11)	Packet length in bytes
Packet Time (20)	Packet transmission time, i.e., the time it takes to transmit the packet at the given flow rate
Flow ID (10)	Flow identification number
Packet ID (16)	Packet identification number (used only to detect out of order events)

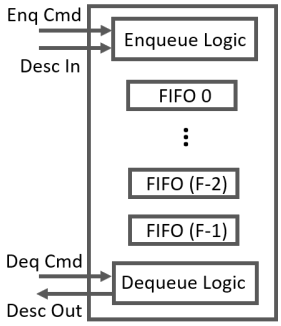


Figure 15: Gearbox Level - VHDL Implementation

The width of each descriptor subfield is parameterized. The numbers in parentheses denote example widths for a given configuration.

Enqueue Operation Upon receipt of an enqueue command, Gearbox performs the following steps:

1. Calculate the packet's departure time based on the packet transmission time and the system time t_s
2. Determine the enqueue level and enqueue FIFO within that level
3. Store the packet descriptor in the calculated level and FIFO

Gearbox completes an enqueue operation in three clock cycles.

Dequeue Operation Upon receipt of a dequeue command, Gearbox performs the following steps:

1. Find first non-empty level and FIFO and update the system time t_s
2. Calculate number of bytes to serve from each level
3. Dequeue from each level a number of descriptors to reach or exceed the number of bytes to serve

Gearbox completes a dequeue operation in four clocks when performing steps 1 through 3 and in two clock cycles if performing only step 3 (i.e., dequeuing from each non-empty level once steps 1 and 2 are completed).

Math Operations The VHDL implementation of Gearbox is scalable using generics that are powers of 2, notably for the number of levels, granularity of each level, and the number of FIFOs within each level. This enables calculations that require division operations to be implemented using bit shifting or truncation, which are much more efficiently implemented in logic gates.

Targeting to an FPGA We targeted the Gearbox VHDL design configured with 4 levels and 16 FIFOs per level, 256 locations (packet descriptors) per FIFO, and 1K flows to a Xilinx Alveo U250 board [34], which uses an UltraScale+ VU13P FPGA with mid-speed grade. Using Vivado 2020.2 [33], we obtained the utilization and performance shown in Table 4.

Table 4: FPGA prototype utilization and performance

	Frequency	LUTs	FFs	BRAM
Units	350 MHz	12331	9953	96
Device Util Pct		0.71%	0.29%	3.6%

With an enqueue and a dequeue every four clocks, the design sustains a packet rate of $350 \div 4 = 87.5$ Mpkts/sec, which is equivalent to a line rate of 100 Gigabit Ethernet for 123-byte packets or larger, taking into account a Preamble of 8 bytes and an IFG of 12 bytes.

5 Related Work

After the proposal of numerous bandwidth allocation algorithms such as WFQ, PGPS, and SCFQ, academia and industry have worked to implement packet schedulers supporting these algorithms. This trend first begins with the ASIC design. In the 1990s, the Sequencer [8] [9] [10] was an ASIC-based hardware packet scheduler that sorts packets into ascending order based on their departure time with limited scalability. In the 2000s, a specialized data structure: pipeline heap (P-heap) [4] [17] provided fine-grained priority queues in hardware, which improves the scalability but is required for each egress port [27] [28] and therefore uses significant chip area on high-speed switches. The recent work PIFO [27] [28] provides a programmable packet scheduler, which has a very small chip area overhead and is relatively easy to implement. However, it requires special hardware support (such as TCAM) and has limited scalability.

The limitations of the ASIC-based hardware packet schedulers we mentioned in section 2 motivated research in approximate schedulers based on strict-priority queues. Among them are the Approximate Fair Queuing (AFQ) [24] and Programmable Calendar Queues (PCQ) [25]. AFQ and PCQ perform well in bandwidth allocation with the same weights. However, when flow weights vary over a wide range, AFQ and PCQ's fixed granularity leads to low scheduling precision or packet drops. The authors of PCQ discuss a hierarchical architecture in their paper, but its dequeuing scheme might lead to starvation of flows in the lower level. SP-PIFO [1] is another recent work that uses a unique algorithm to adjust the priority between FIFOs to minimize scheduling errors. However, SP-PIFO may cause misordering of packets within a single flow, which would lead to problems for TCP-based flows.

6 Conclusion

In this paper, we propose Gearbox, a hierarchical packet scheduler that practically approximates WFQ. Gearbox is a FIFO-based packet scheduler targeted to next-gen programmable switches and smart NICs. Its tiered granularity allows Gearbox to achieve an adequate normalized DTD and FCT performance with a relatively small number of queues. Gearbox eliminates packet recirculation and has a streamlined operation that allows it to achieve high packet processing speed. From our evaluation, Gearbox achieves weighted max-min fairness in bandwidth allocation and FCT performance comparable to that of ideal WFQ. We implement Gearbox in an NS2 simulator and a VHDL-based hardware prototype, targeting a Xilinx ALVEO U250 FPGA card. Our Gearbox hardware prototype runs at 350 MHz, which is equivalent to a maximum throughput of 58.8 Gbps with 64-byte packets over 100GbE and full line rate 100GbE with packets larger than 123 bytes.

Acknowledgements

We thank the anonymous reviewers for their valuable comments and advice. We also acknowledge Xilinx for supporting our hardware prototype implementation.

References

[1] ALCOZ, A. G., DIETMÜLLER, A., AND VANBEVER, L. Sp-pifo: Approximating push-in first-out behaviors using strict-priority queues. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (2020), pp. 59–76.

[2] ALIZADEH, M., YANG, S., SHARIF, M., KATTI, S., MCKEOWN, N., PRABHAKAR, B., AND SHENKER, S. pfabric: Minimal near-optimal datacenter transport. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 435–446.

[3] ALJAEDI, A., CHOW, C. E., ELGZIL, A., ALAMRI, N., AND BAHKALI, I. Network virtualization with openflow for large-scale datacenter networks. *IJCSNS 17*, 9 (2017), 10.

[4] BHAGWAN, R., AND LIN, B. Fast and scalable priority queue architecture for high-speed network switches. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)* (2000), vol. 2, IEEE, pp. 538–547.

[5] BROADCOM. Broadcom StrataDNX™ BCM88480 Traffic Management Architecture. `https://docs.broadcom.com/doc/88480-DG1-PUB`, 2021.

[6] BROADCOM. High Capacity StrataXGS®Trident II Ethernet Switch Series. `http://www:broadcom:com/products/Switching/Data-Center/BCM56850-Series.`, 2021.

[7] BROWN, R. Calendar queues: a fast 0 (1) priority queue implementation for the simulation event set problem. *Communications of the ACM 31*, 10 (1988), 1220–1227.

[8] CHAO, H. J. Architecture design for regulating and scheduling user's traffic in atm networks. In *ACM SIGCOMM Computer Communication Review* (1992), vol. 22, ACM, pp. 77–87.

[9] CHAO, H. J., CHENG, H., JENQ, Y.-R., AND JEONG, D. Design of a generalized priority queue manager for atm switches. *IEEE Journal on Selected Areas in Communications 15*, 5 (1997), 867–880.

[10] CHAO, H. J., JENQ, Y.-R., GUO, X., AND LAM, C.-H. Design of packet-fair queuing schedulers using a ram-based searching engine. *IEEE Journal on Selected Areas in Communications 17*, 6 (1999), 1105–1126.

[11] CISCO. Cisco Silicon One Product Family White Paper. `https://www.cisco.com/c/en/us/solutions/silicon-one.html`, 2021.

[12] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. *Introduction to algorithms*. MIT press, 2009.

[13] DALTON, M., SCHULTZ, D., ADRIAENS, J., AREFIN, A., GUPTA, A., FAHS, B., RUBINSTEIN, D., ZERMENO, E. C., RUBOW, E., DOCAUER, J. A., ET AL. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (2018), pp. 373–387.

[14] DEMERS, A., KESHAV, S., AND SHENKER, S. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM Computer Communication Review* (1989), vol. 19, ACM, pp. 1–12.

[15] FIRESTONE, D., PUTNAM, A., MUNDKUR, S., CHIOU, D., DABAGH, A., ANDREWARTHA, M., ANGEPAT, H., BHANU, V., CAULFIELD, A., CHUNG, E., ET AL. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (2018), pp. 51–66.

[16] HASAN, J., CHANDRA, S., AND VIJAYKUMAR, T. Efficient use of memory bandwidth to improve network processor throughput. *ACM SIGARCH Computer Architecture News 31*, 2 (2003), 300–313.

[17] IOANNOU, A., AND KATEVENIS, M. G. Pipelined heap (priority queue) management for advanced scheduling in high-speed networks. *IEEE/ACM Transactions on Networking (ToN) 15*, 2 (2007), 450–461.

[18] KIM, D., YU, T., LIU, H. H., ZHU, Y., PADHYE, J., RAINDEL, S., GUO, C., SEKAR, V., AND SESHAN, S. Freeflow: Software-based virtual rdma networking for containerized clouds. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)* (2019), pp. 113–126.

[19] MEDEIROS, B., SIMPLICIO, M. A., AND ANDRADE, E. R. Designing and assessing multi-tenant isolation strategies for cloud networks. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)* (2019), IEEE, pp. 214–221.

[20] NAGARAJ, K., BHARADIA, D., MAO, H., CHINCHALI, S., ALIZADEH, M., AND KATTI, S. Numfabric: Fast and flexible bandwidth allocation in datacenters. In *Proceedings of the 2016 ACM SIGCOMM Conference* (2016), ACM, pp. 188–201.

[21] NETWORK SIMULATOR DEVELOPMENT GROUP, T. The network simulator 2. In *https://www.isi.edu/nsnam/ns/*. 2000, 2000, p. 1.

[22] PAREKH, A. K., AND GALLAGER, R. G. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM transactions on networking*, 3 (1993), 344–357.

[23] PAREKH, A. K., AND GALLAGER, R. G. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM transactions on networking 2*, 2 (1994), 137–150.

[24] SHARMA, N. K., LIU, M., ATREYA, K., AND KRISHNAMURTHY, A. Approximating fair queueing on reconfigurable switches. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (2018), pp. 1–16.

[25] SHARMA, N. K., ZHAO, C., LIU, M., KANNAN, P. G., KIM, C., KRISHNAMURTHY, A., AND SIVARAMAN, A. Programmable calendar queues for high-speed packet scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (2020), pp. 685–699.

[26] SHREEDHAR, M., AND VARGHESE, G. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on networking*, 3 (1996), 375–385.

[27] SIVARAMAN, A., SUBRAMANIAN, S., AGRAWAL, A., CHOLE, S., CHUANG, S.-T., EDSALL, T., ALIZADEH, M., KATTI, S., MCKEOWN, N., AND BALAKRISHNAN, H. Towards programmable packet scheduling. In *Proceedings of the 14th ACM workshop on hot topics in networks* (2015), ACM, p. 23.

[28] SIVARAMAN, A., SUBRAMANIAN, S., ALIZADEH, M., CHOLE, S., CHUANG, S.-T., AGRAWAL, A., BALAKRISHNAN, H., EDSALL, T., KATTI, S., AND MCKEOWN, N. Programmable packet scheduling at line rate. In *Proceedings of the 2016 ACM SIGCOMM Conference* (2016), ACM, pp. 44–57.

[29] STEPHENS, B., SINGHVI, A., AKELLA, A., AND SWIFT, M. Titan: Fair packet scheduling for commodity multiqueue nics. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)* (2017), pp. 431–444.

[30] THIMMARAJU, K., RÉTVÁRI, G., AND SCHMID, S. Virtual network isolation: Are we there yet? In *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges* (2018), pp. 1–7.

[31] VARGHESE, G., AND LAUCK, A. Hashed and hierarchical timing wheels: efficient data structures for implementing a timer facility. *IEEE/ACM transactions on networking 5*, 6 (1997), 824–834.

[32] WANG, Z., HUANG, H., ZHANG, J., AND ALONSO, G. Shuhai: Benchmarking high bandwidth memory on fpgas. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2020), IEEE, pp. 111–119.

[33] XILINX. Vivado Design Suite, Integrated Design Environment. `https://www.xilinx.com/products/design-tools/vivado.html`, 2021.

[34] XILINX. Xilinx Alveo U250 Data Center Accelerator Card. `https://www.xilinx.com/products/boards-and-kits/alveo/u250.html`, 2021.

Appendix

A Step-down FIFO

The solution to the packet out-of-order issue in Section 3.6 may increase the DTD of specific flows. Normally, packets will enqueue at the appropriate level with a granularity that is appropriate for their expected delays. As long as the packets from a flow arrive at the allocated rate, its packets will always enqueue at the appropriate level with a guaranteed low DTD. However, when a flow's input data rate exceeds its allocated bandwidth, its packets may queue up at higher levels. As the scheme in Section 3.6 maintains the 'Last packet enqueued level' L_i, such flows would only enqueue their subsequent packets in the higher levels from that point on. As previously discussed, when packets that belong to a lower level enqueue at a higher level, they suffer from a coarser granularity and higher DTD. This leads to larger packet delays and increases FCT. In this case, flows need to step back to the lower level in which they are supposed to enqueue so they can restore the lower DTD. Is it possible for a flow that queues up to a higher level to step down to a lower level when its arrival rate decreases to its admitted bandwidth?

The answer is yes: we introduce the 'Step-down FIFO', which allows flows at a higher level to go to a lower level. To understand the design of the Step-down FIFO, we must first understand why we need to enqueue packets into a higher level. According to Section 3.4, we can secure the serving order of packets as long as we serve them with the same granularity. In other words, if we can serve packets at the higher level with the same granularity at the lower level, it would be safe to enqueue the subsequent packets into the lower level without causing packet misordering. At this point, 'Step-down FIFO' serves as a special FIFO at a higher level that provides the same granularity of a lower level. A 'Step-down FIFO' expands a FIFO at the higher level into multiple queues with finer granularity, preserving the departure

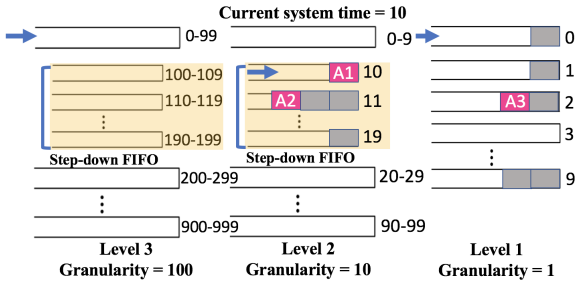


Figure 16: Step-down FIFO

time difference between the packets. When a flow's latest packet enqueues into a 'Step-down FIFO' at level l, Gearbox schedules it with the granularity of level $(l-1)$. Thus, we can mark the 'Last packet enqueued level' $L_i = (l-1)$. As a result, we can enqueue the subsequent packets from the same flow into level $(l-1)$.

Figure 16 illustrates how a 'Step-down FIFO' works. FIFOs marked in yellow at level 2 and 3 are 'Step-down FIFOs', which maintain the same granularity as the next lower level. In this example, Gearbox uses 10 FIFOs to achieve a Step-down FIFO with the finer granularity. Packets $A1$ and $A2$ enqueue the 'Step-down FIFO' and preserve their departure times $F_{(A,1)} = 10$ and $F_{(A,2)} = 11$ with the granularity of $g_1 = 1$. By doing so, Gearbox will schedule packets $A1$ and $A2$ with the same granularity at level 1 and the packets will leave the scheduler at $t_s = 10$ and 11. In this case, it is safe to place packet $A3$ at level 1 without causing packet out-of-order issues. In this example, 'Step-down FIFO' makes it possible for flow A to step down from level 2 to level 1.

With a 'Step-down-FIFO', the flows that follow their allocated rate will eventually get back to the level they belong to and restore their DTD. Assume flow i belongs to level l, where the departure time of its packets increases by g_l. Due to prior burstiness, packets from flow i queue up at a higher level l' and need to step back down to level l. Since flow i now follows its admitted rate r_i, its packets arrive with a departure time interval of g_l without accumulation. Based on the basic idea of Gearbox in Section 3.1, each FIFO at level l' (including the 'Step-down FIFO') covers a departure time range of $g_{l'}$ and $g_{l'} >> g_l$. Since $F_{(i,k)}$ increases by g_l, eventually there will always be a packet that enqueues in the 'Step-down FIFO' at level l'. As the 'Step-down FIFO' at level l' schedules packets with the granularity of g_l, Gearbox will mark the last enqueue level of flow i as $L_i = l$ and the subsequent packets from flow i will get back to level l. In this way, the Step-down-FIFO enables the flows that follow their allocated rates to eventually get back to the lower level to which they belong.

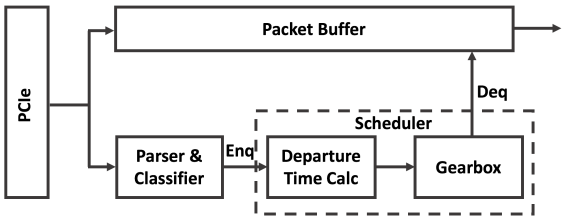


Figure 17: Gearbox's Application in a NIC (egress only shown)

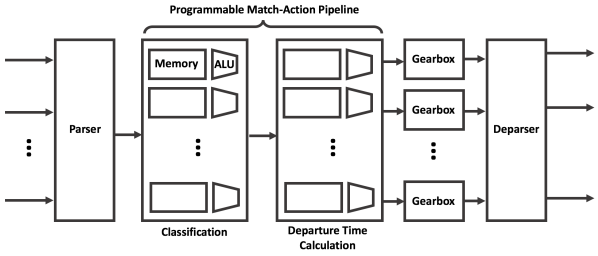


Figure 18: Gearbox's Application in Multi-pipeline Systems

B Gearbox's Application in NIC and Multi-pipeline Systems

Figure 17 shows a Gearbox application in a NIC with only the egress path shown. Packets from the server arrive over PCIe and are stored in the Packet Buffer. The packet headers are parsed and classified in the Parser & Classifier block to extract the flow id and the packet length. The Departure Time Calculator computes the departure time using the packet length and forwards it to Gearbox for enqueue. Gearbox dequeues descriptors and forwards the packet pointers to the Packet Buffer to output the packets.

Figure 18 shows a Gearbox application in a multi-pipeline switch. After the parsing stage, the classification (flow identification) is done in the first stage. The second stage computes the departure times, which are fed to multiple Gearboxes for enqueue. Dequeued packets are output by the deparser.