Dynamic Bucket Granularity Setting in Eiffel

Max Schwarz
University of Colorado Boulder
CSCI 5273
max.schwarz@colorado.edu

1 INTRODUCTION

Packet scheduling algorithms in networks are a core operation in the completion of flows and are subject to strict deadlines in order to meet specific QoS. Packet scheduling resides in the network interfaces of switches within datacenter networks. As packets are received, the location in the a queue in which the packet is placed is determined by the scheduler and dependent on network-set policies. Classical implementations exist as simple FIFO (first in first out) queues, and have gained complexity with the diversification of flow types and network requirements. State of the art approaches attempt to balance many factors, including efficiency and flexibility. Notable recent packet scheduling systems with differing priorities and performance include Carousel [4], PIFO (push-in first-out queue), and Eiffel. This project implements Eiffel, and implements a solution to an avenue for future work provided by the author of Eiffel, by providing a scheme to dynamically set non-uniform bucket granularities. The method hopefully is a step for the system presented in Eiffel in providing consistent performance without the manual fine tuning of parameters based on network and traffic characteristics. This project consisted of a study of packet scheduling and current approaches, modifying the Eiffel kernel module implementation to enable installation on my machines, the creation of a scheme for dynamic bucket granularity determination, and a comparison against the original Eiffel implementation. The findings from the comparison show the proposed method achieves similar results to the original implementation with a well-set granularity, and is able to modify an incorrectly set granularity to increase performance.

1.1 Motivation

A technical description of Eiffel and the presentation of the bucket granularities and their role in Eiffel is included in section 1.2. This section details the overall motivation for the project and the aspects other than the direct technical effort.

Packet scheduling as a network topic was chosen due to the timing and pacing of the course. During the time of project formulation/proposal, datacenter networking was the topic of discussion. After reading Carousel, and researching some of the papers that have cited it, the method used in Eiffel showed vast improvement in performance and cpu usage. Along with the improvements, Eiffel's implementation showed greater complexity and built off the work of other state of the art approaches to produce an interesting and effective packet scheduling method. The choice to aim the scope of the project at improving an existing method was motivated by two factors, an interest in implementing and testing a state of the art approach as well as a relative inexperience with networking topics and evaluation methodologies. Multiple avenues for future work are suggested in Eiffel, choosing dynamic selection of bucket granularities is appropriate for the project for a few reasons:

(1) An important practical implementation limitation or consideration of Eiffel is in choosing a bucket granularity that yields optimal results. In the existing implementation as of presenting of the paper, this choice is set upfront and unable to adapt to network changes. The granularity parameter is dependent on policy, and traffic characteristics which can change over time, making the chosen value sub-optimal.
(2) Modification of the setting of the bucket granularity requires understanding of the implementation of Eiffel as well as further research into methods of packet scheduling and related topics. Improving upon Eiffel also gives practical experience in how some of these state of the art techniques and systems are implemented on a real system and the methods used to collect data and produce an evaluation.
(3) Dynamic adjustment of parameters lends itself to many potential solution approaches, as well as referring to state of the art approaches to similiar problems in other works.

1.2 Background

1.2.1 Packet Scheduling. Packet schedulers are responsible for packet reception and transmission in network devices and controllers. Specifically, schedulers manage the queuing and dequeuing of packets based on a ranking according to a set policy. As packets are received, the scheduling algorithm will rank the packet according to the policy, and place it in a queue at a position reflecting the rank. On dequeue, the remaining packets in the queue, depending on implementation details, may be re-queued to reflect the changed state of the queue contents.

Both software and hardware approaches to packet scheduling exist, although recently software has become favored. This is due to the flexibility and programmability of software approaches in handling different scheduling policies. Hardware also has a much greater development cost and time than software, resulting in solutions that lag behind requirements. Lastly, hardware implementations are inherently stuck to that device, where software schedulers may be implemented at different layers across supported devices in the network. Due to this, software packet schedulers are more suited to the growing complexity of networks. Although flexible, these algorithms must operate on each packet, requiring significant cpu resources. At high line rates, scheduling can become a botteneck in the network, stemming from the cost of enqueing and dequeuing. Standard implementations of a scheduling queue employ a comparison-based approach incurring O(log n) operation costs. Recent works are dedicated to reducing this to achieve efficiency at high line rates.

Diverse policies exist with varying degrees of complexities. For example, policies generally fall under two categories, per-packet and per-flow, of which are not unilaterally supported by different schedulers. Per-flow differs from per-packet in that per-flow classifies packets into a flow based on a rule to determine packet source. Another policy feature that may not be supported by some schedulers is work conservation. Work and non-work conservation dictates the behavior of the controlled resource. Work conserving schedulers enforce the idling of the channel only when the queue is empty, or there are no packets waiting to be transmitted. The ranking and adjustment of the queue at enqueue as well as dequeue is also an implementation-specific supported policy designation. Flexibility of packet schedulers is determined by their ability to support many policies.

At the base of all packet schedulers is the queue data structure. Implementations need to balance complexity and flexibility with efficiency in the choice of a queuing method. Conventionally, packet schedulers use red-black trees and binary heaps to implement the priority queue. These result in O(log n) operation. More recent works use highly specialized data structures to drive down the overhead of the queuing.

In practice and in history, many approaches aim to balance the different considerations and metrics of packet schedulers. With increased complexity of policies, higher QoS standards, and a general increase in datacenter traffic, the creation of more advanced packet schedulers is an active topic of research and development.

Packet schedulers are aimed at maximizing network capacity while abiding by QoS standards and forwarding fairly. [6] This is done through improving network performance and efficiency by reducing resource requirements and utilization, while providing satisfactory fairness in packet forwarding. The use of memory in switches is dependent on the use of efficient queue data structures, and cpu utilization is determined by the efficiency and complexity of the scheduling algorithms. This efficiency can be defined in terms of minimizing system calls per scheduling action per packet to allow high processing rates. Generally, scheduling algorithms are designed with respect to a few principles: flexibility, efficiency/performance, security and protection, and overhead or resource consumption.

1.2.2 Eiffel. Eiffel is a state of the art programmable software packet scheduler with improved flexibility and efficiency over similar approaches [3]. It consists of a packet annotator, an enqueueing component, a queue implementation, and a dequeueing component. Packets recieved are assigned priorities by the annotator as mandated by a policy and sent to the enqueuing component. The actual enqueueing is determined by the policy, assigned priority, and queue structure. The queue stores packets waiting transmission and maintains the data structure. Using integer priority queues and a design coupled with awareness of minimizing system calls, Eiffel reduces overhead of insertion and removal to O(1) and thus a O(number of ranking policies) overhead for overall scheduling. The improvement to flexibility is realized through extending the work of [5], providing PIFO (push-in first-out queue) the ability to operate on per-flow policies, as well as ranking operations on dequeue. With these policy abstractions, Eiffel is shown to accommodate a wider range of rules and policies while reducing CPU utilization and operating at line rate.

Eiffel leverages integer priority queues dependent on the find first set (FFS) instruction, which determines the index of the least significant SET bit in a word with respect to the index of the least significant bit. Along with consideration and combination of methods of circular FFS-based queues and approximate gradient queues, Eiffel produces a method for O(1) operation overhead.
FFS enables O(1) priority lookups where each bit of a word represents a bucket in the queue. If the number of buckets is equal to or less than the size of the word, the FFS instruction gives the index of the lowest priority element. In the case of greater buckets than bits, a series of words can be used forcing the cost to O(number of words). At large numbers of words, processing in sequence becomes expensive, motivating the use of hierarchical bitmaps. Hierarchical bitmaps are trees with nodes taking the value of the occupancy of its children. Leaf nodes take the form of buckets, and determining the lowest or highest priority element is a conventional binary tree search assuming O(log number of buckets). Depending on the characteristics and the variance in priorities, FFS-based queues that employ hierarchical bitmaps can be sparse with a large number of consistently empty buckets. These structures also operate over a finite range of priorities.

Cirular FFS-based queues respond to the issues with hierarchical FFS-based queue's space inefficiency and limited operational range. Generic circular queues, used in this context would result in improper ordering from range adjustmant. When out of range elements are added and the queue is mapped to a new range, existing elements are dequeued. After setting a new range, the new ordering of elements does not preserve the prior order. The proposed solution to this issue is adding an FFS queue as a buffer, containing priorities just greater than the first queue. When all elements are dequeued from the primary, the range is reset and is populated from the buffer queue. From these constructions the Circular Hierarchical FFS-based queue (cFFS) is introduced, which establishes two hierarchical FFS-based queues. This construction manages proper ordering and efficient operation through switching pointers to each respective queue based on the minimum element's position. Further optimization to achieve faster lookups results in the use of an approximate gradient queue. A gradient queue assigns a weight function to be applied to each bucket. It then defines a curvature function built from summing the weights of occupied buckets. Each produced curve is unique, and generalizes the occupancy of the queue. This formulation allows determination of the maximum index of an occupied bucket through the point at which the curvature function's derivative is 0. Eiffel's implementation chooses $2^{f(i)}(x-i)^2$ to be the weight function where i represents the index of the bucket and $f(i) = i/\alpha$ arises from approximation. From this choice, the point at which the derivative is zero is defined as $b/2a$. Ultimately, determining the minimum element is reduced to the problem of determining a and b where $a = \Sigma_i 2^{f(i)}, b = \Sigma_i i 2^{f(i)}$. The use of alpha in $f(i) = i/\alpha$ gives a and b more expressiveness, representing a greater range of i. This allows for finding the critical point, and therefore the minimum element in a single operation through the use of the function ceil(b/a). Using this approximation means the curvature function no longer guarantees finding the minimum element. To account for this, the conditions under which introduce error are characterized in terms of scheduling policies. Generation of uniform priorities is a trait of many policies, and will result in an error free determination of the critical point. For policies with tendencies to generate non uniform priorities, the guarantee does not hold and the function may output empty buckets. In this case, non-empty buckets will typically be close to the index, and a linear search is adequate. Together, the approximate gradient queue and and the circular FFS-based queue formulations allow Eiffel to perform lookups in one operation in the case of moving and static ranges as determined by policy priority characteristics.

To achieve the increased flexibility over PIFO, two scheduling primitives are added to the programming model and PIFO is expanded to support shaping. Support for per-flow ranking is added by a modification to PIFO to allocate a queue per flow to handle all packets of that flow. Each flow is then operated on and ranked rather than packets. On-dequeue scheduling is also added within the PIFO model, and allows for packets and flows to be re-ranked upon any dequeue taking place.

1.2.3 *Bucket Granularity.*

The bucket granularity is defined as the range of ranking values that map to the bucket. Higher bucket granularity results in more buckets to cover the required range of priorities. A higher granularity then means less packets per bucket, and finer grained control over the queuing. Preferably, the queue should contain as few empty buckets as possible, while minimizing the number of elements per bucket. This balance is determined by the bucket granularity, as too large of buckets will cause a high number of packets per bucket, and too small will introduce empty buckets. As seen in detailing the queuing structures used in the previous section, empty buckets cause complication to the lookup methods, resorting to a linear search to find elements. Along with increased overhead, empty buckets are a source of error, as the approximation method to drive down operations cause an inexact priority selection in the case of empty bucket selection. Another performance concern is the unnecessary allocation of memory for buckets that are never occupied. Although high granularity guarantees proper packet order in the queue, the accuracy and overhead are negatively impacted in practice. In the case of too low of bucket granularity, too many packet priorities are grouped together resulting in inconsistent and inaccurate priority selection. This outweighs the benefit from better throughput caused by a fewer number of buckets to search. This balance, and an indeterminate optimal configuration of packets per bucket, is an important parameter in determining the performance and efficiency of Eiffel as a whole.

The ranges of priorities that are generated in an application is dependent on traffic patterns and characteristics as well as the scheduling policies. Eiffel, in its implementation and evaluation, empirically chooses a bucket granularity on the setup, and sets it. This approach takes considerable effort in determining a granularity, does not guarantee long term performance, and is impractical in many applications, hurting the scalability of the packet scheduler.

2 SYSTEM DESIGN

2.1 Considerations

The effects of bucket granularity on the packet scheduler are described in section 1. The driving force of Eiffel's performance improvement is in the implementation of an optimized queue structure allowing extremely low overhead operations. Therefore, the design of a method for dynamic bucket granularity determination should incur as low overhead as possible, and be directly evaluated with respect to the aggregate cost of Eiffel's queue structure operations. This means that the proposed solution needs to be both memory and cpu efficient. Any change to the bucket granularity forces the queue to restructure, either resulting in penalties to QoS, or increased resource usage. Due to the need to be as selective as possible to reduce the costs associated with calculating and applying a change to bucket granularity, long term behavior should be prioritized over short term. Furthermore, a change should only be considered when the projected benefit is sufficient to justify the change. The principal driver in applying a granularity update should be long term behavior as it inherently filters short term instabilities and reduces the frequency of incurring change costs. Tracking the long term behavior of the queue results in significant dedication to storing information, therefore aggregations over windows are appropriate and allow storage of only enough information to make informed decisions.
Eiffel includes a limited study into the effects of bucket granularity on performance to be leveraged by a proposed solution. Rate of data transmission is shown to directly increase with fraction of non-empty buckets due to limiting the number of empty bucket hits. Error is also shown to increase superlinearly with ratio of empty buckets. For a given amount of empty buckets, it is shown that fewer packets per bucket also reduces error. Conversely, the queue's throughput or rate increases with increased packets per bucket, highlighting the balance between memory, cpu usage, priority selection error, and rate.

2.2 Implementation

With clearly defined considerations regarding the balance between too large and too small of bucket granularity, the development of a solution is subjected to the following principles:

- An optimal solution would provide convergence on an average bucket occupancy of 1.
- With respect to the optimization efforts of Eiffel, reducing the cpu and memory overhead of dynamic granularity setting must be prioritized over marginal performance gains.
- Due to large timescales of operation of deployed packet schedulers, any action repeated over the lifetime of the scheduler becomes costly.

The design of the solution can be broken into three steps.

(1) Obtain measure of empty buckets
(2) Determine if adjustment is necessary and the degree of adjustment
(3) Apply change to the queue discretely

To determine the ratio of empty buckets present, random sampling is used to obtain a subset of the circular gradient queue (cgq). This random subset is used to reflect the state of the whole queue while reducing the overhead required in analyzing the entire contents of the structure. This set samples from the bitmap representation in the cgq, containing a set bit for non-empty buckets, and 0 for empty buckets. This bitmap is maintained as part of cgq, and the pointer to both the primary and buffer queues is readily available as cgq uses it to switch between active queues. The instruction POPCNT is used to accelerate the counting of set bits in the bitmap subset. From this count, a representative ratio of empty buckets is obtained.

A decision of whether to adjust, and how large of an adjustment to apply is established using the ratio of empty buckets. Eiffel suggests reevaluation of the bucket granularity at 30% empty buckets. This is used as a benchmark threshold to determine when a vote is cast for a bucket size change. A sliding window with a horizon of α is implemented as a bitmap to store votes. The rightmost bit is set, representing a vote, if the ratio of empty buckets exceeds 30%. The value of α is highly dependent on the overall frequency of the routine being called. Effectively, it defines how many bucket occupancy checks to include in the decision to change. The number of votes is then counted using POPCNT each call to the algorithm. The number of votes is then thresholded according to a parameter β to determine if an update to granularity is triggered. β can be set high to reduce how reactive the algorithm is perturbations or instability in priority assignments, or low to increase to reactivity. If not enough votes are in the current window for a change, the sliding window is shifted left by one. In the event of enough votes for a change, the voting array is reset to 0s, and the magnitude of change is determined. An important principle to consider at this stage is achieving an optimal configuration by converging on an average bucket occupancy of 1. Checking for empty buckets is done with low overhead due to simplicity of the help of POPCNT. Determining the average bucket occupancy requires many more operations and complexity as a single bit per bucket is not enough information for average occupancy determinations. In the circular approximate gradient queue used in Eiffel, this information is available, but would require extra computation. In a cFFS implementation, this information would not be readily available and would require even more overhead to determine. To enable the algorithm to support both reduction and increase of bucket granularity without having to calculate the average occupancy, a 5% empty bucket is the objective. This allows the use of the empty bucket ratio to to be used for both increasing and decreasing the granularity without sacrificing too much performance by not achieving 1 packet per bucket. The granularity is then adjusted by an amount proportional to the difference between the empty bucket ratio and 5%. The constant of proportionality used is 1% and determined empirically by using a few different values. This value is the number of buckets to add or remove from the queue as determined by the sign.

The application of the change to the queue as implemented is specific to the cgq implementation. Both the primary and buffer queues without packets are created using the new bucket size and the same minimum priority value supported. This is done by using the creation that is already implemented in Eiffel, and takes as input the bucket size and minimum supported priority. When the primary queue becomes empty, and the pointer is switched to the buffer queue, the pointer is changed to point to the new queue. When the pointer is switched back to the primary queue, the buffer queue pointer is changed. During queue switching, a flag set when enough votes are cast for an update is checked to invoke the queue replacement.

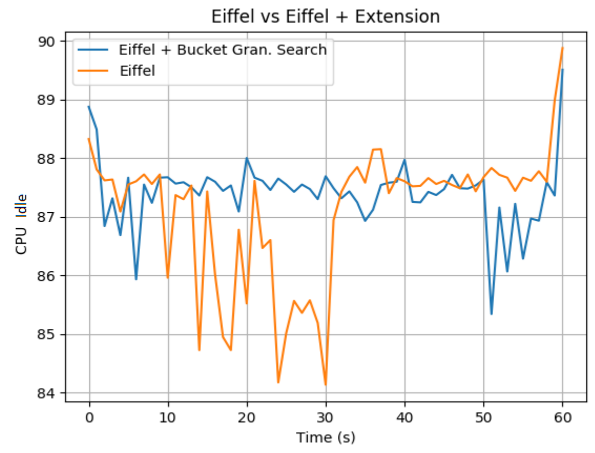

Figure 1: Idle CPU for Eiffel vs Eiffel + Granularity Updates

Eiffel is implemented as a qdisc kernel module, modifying the enqueue and dequeue functions. The Linux Kernel containing the Eiffel qdisc module available is based on Linux Kernel 4.10. This project modifies the Eiffel qdisc module, defined in /net/sched of the kernel source, in particular sch_gq.c. For the implementation of dynamic granularity setting, $\alpha = 10$ (32 x 10 bits) and $\beta = .9$. For testing purposes, The function do_gettimeofday is used to prevent the routine from being called more frequently than once every 6 seconds [1]. This results in the sliding window covering one minute. This is a temporary method to allow testing despite an unsolved problem of how to schedule a check of whether the granularity should be changed. This can not be every time enqueue or dequeue is called due to the overhead, and remains an unsolved issue.

Along with the issue of frequency, a problem encountered was building and installing the Linux Kernel provided. Just to get the Kernel to build required research and application of multiple patches. The installation was also a struggle as I had to fight with unsupported nvidia drivers, and other issues resulting from installing an older Kernel. To use neper [2], I had to install the Kernel on multiple machines, both of which I encountered different errors that took some time to resolve.

3 FINDINGS

With respect to the higher level goal of implementing Eiffel, modifying, and using the method on a workload, I was able to build and install the modified kernel with the updated Eiffel qdisc. Figure 1 shows the average system idle CPU of Eiffel vs Eiffel with the granularity update method over tests of 60 seconds. These averages are over 5 workload generations using different numbers of flows, a fixed rate, and 1 thread. Over the tests, the average CPU usage of Eiffel with the granularity update is .30% higher than without. A test to determine the ability of the method to adjust the bucket granularity ran an identically configured workload generation over 5 minutes. The starting granularity was adjusted in each test. The average number of buckets resulting from the 5 tests was 8214, and by the end of the tests, the number of buckets in every test was within +/- 150 of 8214. This shows the method was successful in achieving convergence when given varied starting granularities. The evaluation of the quality of this number of buckets with respect to rate, cpu usage, and selection accuracy was planned by is left to possible future work.

4 REVIEW OF TEAM MEMBER WORK

This project was not completed as part of a group.

5 CONCLUSION

Overall, I was not able to complete all the parts of the project I wanted to. Components that I did not have a chance to complete were how to incorporate non-uniform bucket granularity assignment, a more thorough evaluation of the method used including more sophisticated/targeted workload generation as well as metrics such as selection accuracy and rate, and a more formal method to classify the balance that exists between selection error, rate, and overhead. The project was successful in terms of my goal of implementing and running a state of the art approach to a networking topic. It exposed me to some of the testing software that's available, the considerations of resource use and efficiency, as well as a more comprehensive study into packet scheduling than what was done in the course.

REFERENCES

[1] [n. d.]. Components of Linux Traffic Control. ([n. d.]). https://tldp.org/

[2] [n. d.]. neper: a Linux networking performance tool. ([n. d.]). https://github.com/google/neper

[3] Ahmed Saeed, Nandita Dukkipati, Valas Valancius, Terry Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable Traffic Shaping at End-Hosts. In *ACM SIGCOMM 2017*.

[4] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable Traffic Shaping at End Hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 404–417. https://doi.org/10.1145/3098822.3098852

[5] Ahmed Saeed, Yimeng Zhao, Nandita Dukkipati, Mostafa Ammar, Ellen Zegura, Khaled Harras, and Amin Vahdat. 2018. Eiffel: Efficient and Flexible Software Packet Scheduling. (10 2018).

[6] Tsung-Yu Tsai, Yao-Liang Chung, and Zsehong Tsai. 2010. Introduction to Packet Scheduling Algorithms for Communication Networks. In *Communications and Networking*, Jun Peng (Ed.). IntechOpen, Rijeka, Chapter 13. https://doi.org/10.5772/10167