Approximate Fair Queueing: A Low Complexity Packet Scheduler for Embedded Networks

Arijit Ghosh

Tony Givargis

Technical Report CECS-09-02

April 14, 2009

Center for Embedded Computer Systems

University of California, Irvine

Irvine, CA 92697-3425, USA

(949) 824-8168

{arijitg, givargis}@cecs.uci.edu

Abstract

Fair queueing is a well-studied problem in modern computer networks. There are two classes of queueing algorithms. One focuses on bounded delay and good fairness properties. The other focuses on the performance of the scheduling algorithm itself. However in embedded networks working under real time constraints, equally important is the deadline imposed by the application. Modern queueing algorithms do not address that fact explicitly. In this paper we propose a scheduling algorithm, Approximate Fair queueing, that aims to bridge this gap. Approximate Fair Queueing schedules packets based on the packet's deadline. In doing so, it maintains fairness in allocating resources to all competing flows. The delay that a packet experiences due to the execution of the scheduling algorithm is also reduced by reducing the frequency of execution. We perform extensive evaluation by simulating a system with a single bottleneck node and up to 1000 concurrent flows.

Contents

List of Figures

Approximate Fair Queueing: A Low Complexity Packet Scheduler for Embedded Networks

Arijit Ghosh, Tony Givargis

Center for Embedded Computer Systems

University of California, Irvine

Irvine, CA 92697-3425, USA

{arijitg,givargis}@cecs.uci.edu

http://www.cecs.uci.edu

Abstract

Fair queueing is a well-studied problem in modern computer networks. There are two classes of queueing algorithms. One focuses on bounded delay and good fairness properties. The other focuses on the performance of the scheduling algorithm itself. However in embedded networks working under real time constraints, equally important is the deadline imposed by the application. Modern queueing algorithms do not address that fact explicitly. In this paper we propose a scheduling algorithm, Approximate Fair queueing, that aims to bridge this gap. Approximate Fair Queueing schedules packets based on the packet's deadline. In doing so, it maintains fairness in allocating resources to all competing flows. The delay that a packet experiences due to the execution of the scheduling algorithm is also reduced by reducing the frequency of execution. We perform extensive evaluation by simulating a system with a single bottleneck node and up to 1000 concurrent flows.

1 Introduction

Embedded network systems (EmNets), including sensor networks and distributed control applications, are becoming an important new computing class with wide ranging and novel applications. These applications

often have strict quality-of-service (QoS) requirements like throughput and latency. Satisfying the QoS requirements of competing flows, given a packet-switched network with decentralized control distributed over all the routers, is a long standing problem that has been the subject of considerable research [6, 10, 11, 15, 18, 19, 20, 33, 7] in the networking community.

An important component of the many QoS architectures proposed is the packet scheduling algorithm used by routers in the network. The packet scheduler determines the order in which packets of various independent flows are forwarded on a shared output link. Packet scheduling affects the queueing delay. The International Telecommunication Union Telecommunication Standardization Sector (ITU-T) G.114 recommendation, for example, specifies that for good voice quality, no more than 150 ms of one-way, end-to-end delay should occur. A poorly designed scheduling algorithm could add as much as two seconds of delay to a packet [12].

Generally speaking, packet scheduling algorithms can be divided into two categories. Timestamp-based (also called deadline-based) algorithms [23, 13, 4, 34] have provably good delay and fairness properties [16, 17, 22, 30, 31], but generally need to sort packet deadlines, and therefore suffer from complexity logarithmic in the number of flows N. Round-robin-based algorithms [29, 14, 9] have O(1) complexity, and while they support fair allocation of bandwidth, they fail to provide good delay bounds. EmNets typically have soft real time requirements and so the former kind is more preferable. However, to reduce the latency of a packet due to the scheduling algorithm, we need to make it simpler, which possibly opens the opportunity for a hardware implementation.

A packet scheduler should have the following properties.

Fairness: The packet scheduler must provide some measure by which multiple flows receive a fair share of system resources, for example, the shared output link. In particular, each flow should get its fair share of the available bandwidth, and this share should not be affected by the presence and (mis)behavior of other flows.

Bounded delay: EmNet applications require the total delay experienced by a packet in the network to be bounded on an end-to-end basis. Packets arriving after the deadline are considered useless and are treated as lost; and the loss of a certain number of packets will seriously affect the quality of the application. The packet scheduler decides the order in which packets are sent on the output link, and therefore determines the

queuing delay experienced by a packet at each intermediate router in the network.

Low complexity: While often overlooked, it is critical that all packet processing tasks performed by routers, including output scheduling, be able to operate in nanosecond time frames. The time complexity of choosing the next packet to schedule should be small, and in particular, it is desirable that this complexity be a small constant, independent of the number of flows N. Implementing a hardware solution is preferable to produce desirable throughput. Indeed, activities like route lookup are also performed in hardware [24].

Designing a packet scheduler with all of these constraints is a big challenge that warrants attention. This paper presents a very simple queueing algorithm called Approximate Fair Queueing (AFQ). AFQ schedules the packet at the head of a flow depending on the deadline that is earliest. Once scheduled, consecutive packets in the same flow are scheduled as long as the allocated bandwidth constraints are respected. This notion is different from the existing timestamp schedulers. As will be shown later, AFQ is a better fit for embedded networks where flows work under soft real time constraints.

2 Related Work

There is a significant amount of prior work in finding scheduling disciplines that provide delay and fairness guarantees. Generalized Processor Sharing [27] (also called Fluid Fair Queuing) is considered the ideal scheduling discipline that achieves perfect fairness and isolation among competing flows. However, the fluid model assumed by GPS is not amenable to a practical implementation, as network communication takes place in the form of packets that must be transmitted atomically. Nevertheless, in terms of fairness and delay guarantees, GPS acts as a benchmark for other scheduling disciplines. Practical scheduling disciplines can be broadly classified as either timestamp schedulers or round-robin schedulers.

2.1 Timestamp schedulers

Timestamp schedulers [23, 15, 13, 21, 34] try to emulate the operation of GPS by computing a timestamp for each packet. Packets are then transmitted in increasing order of their timestamps. For example, the well-known Weighted Fair Queuing [13] algorithm uses this method by computing the timestamp of a packet as the time it would finish being serviced under a reference GPS server. WFQ exhibits some short-term unfairness which is addressed by the Worst-case Weighted Fair Queuing [23] algorithm. While WFQ sched-

ules the packet with the least timestamp among all packets, WF2Q only considers those packets that have started receiving service under the reference GPS server. As a result, WF2Q achieves "worst-case fairness", a notion defined in [23]. Although both WFQ and WF2Q have good delay bounds and fairness properties, the need to maintain a reference GPS server results in high complexity. Specifically, both algorithms have a time complexity of O(N), where N is the number of competing flows. It has subsequently been shown how to modify WF2Q so that it has a time complexity of $O(\log N)$ [5].

Self-Clocked Fair Queuing [21] and Virtual Clock [34] are timestamp schedulers that use computationally more efficient schemes to compute timestamps without maintaining a reference GPS server. As a result, timestamps can be computed quickly. However, it is still required to sort packets in ascending order of their timestamps. Consequently, they still have a time complexity of $O(\log N)$ per packet.

In general, although timestamp schedulers have good delay properties, they suffer from a sorting bottleneck that results in a time complexity of $O(\log N)$ per packet. The Leap Forward Virtual Clock [4] algorithm attempts to address this problem by coarsening the way in which timestamps are computed in Virtual Clock. This results in a reduced time complexity of $O(\log \log N)$ per packet. This is an interesting result in terms of showing that rough sorting is almost as good as exact sorting. However, the implementation requires a complicated data structure such as a *Van Emde Boas* tree that typically would have higher constants, and is not suited to a hardware implementation. Thus, the high computational costs associated with timestamp schedulers prevent them from being used in practice.

Recent lower bounds [32] suggest that the $O(\log N)$ sorting overhead is fundamental to achieving good delay bounds. In particular, any scheduler that has a complexity of below $O(\log N)$ must incur a GPS-relative delay proportional to N, the number of flows.

2.2 Round robin schedulers

Round-robin schedulers [29, 9, 26] are the other broad class of work-conserving schedulers. These schedulers typically assign time slots to flows in some sort of round-robin fashion. By eliminating the sorting bottleneck associated with timestamp schedulers, they achieve an O(1) time packet processing complexity. As a result, they tend to have poor delay bounds and output burstiness.

Deficit Round Robin (DRR) [29] is a well-known example of a round-robin scheme. DRR assigns a

quantum size to each flow that is proportional to the weight of the flow. Each flow has a deficit counter that measures the current unused portion of the allocated bandwidth. Packets of backlogged flows are transmitted in rounds, and in each round, each backlogged flow can transmit up to an amount of data equal to the sum of its quantum and deficit counter. The unused portion of this amount is carried over to the next round as the value of the deficit counter. Once a flow is serviced, irrespective of its weight, it must wait for N-1 other flows to be serviced until it is serviced again. Also, during each round, a flow transmits its entire quantum at once. As a result, DRR has poor delay and burstiness properties. However, due to its extreme simplicity, DRR (or some variant) is the scheduling discipline typically implemented in high speed routers such as the Cisco GSR [3].

Summarizing, timestamp schedulers have good fairness and delay properties but high complexity, while round-robin schedulers are simple to implement but have poor delay bounds and show output burstiness. More recently proposed schemes [9, 26, 8] have attempted to achieve the best of both worlds by combining the fairness and delay properties of timestamp schedulers with the low complexity of round robin schedulers. This is typically done by evolving a round robin scheme like DRR and incorporating some elements of a timestamp scheduler.

The Smoothed Round Robin [9] discipline addresses the output burstiness problem of DRR. This is done by spreading the quantum allocated to a flow over an entire round using a Weight Spread Sequence. Although SRR also results in better delay bounds than DRR, the worst case delay experienced by a packet is still proportional to N, the number of flows.

Aliquem [26] is an evolution of DRR that permits scaling down the quantum assigned to a flow in each round. However, since the quantum may be less than the maximum packet size, a flow may not be able to transmit any data in each round. Therefore a mechanism is required to keep track of the round in which a flow has accumulated enough credit to transmit a packet. Their mechanism, called Active List Management, can be implemented using a priority encoder, similar to Stratified Round Robin. The scaling down of the quanta results in better delay and burstiness properties.

Bin Sort Fair Queuing [8] uses an approximate bin sorting mechanism to schedule packets. Each packet is assigned a deadline similar to a timestamp scheduler. Packets with close deadlines are assigned to the same bin. Within a bin, there is no sorting of packets based on deadlines. Therefore, packets are transmitted

in approximately the same order as their deadlines.

Although both Aliquem and BSFQ significantly improve the delay bounds of DRR, it appears that the worst case delay of even a single packet is still proportional to N, the number of flows.

Stratified round robin [28] is an attempt to bridge the gap between the simplicity of round robin schedulers and the fairness of timestamp schedulers. SRR groups flows of roughly similar bandwidth requirements into a single flow class. Within a flow class, a weighted round robin scheme is employed. However, deadline based scheduling is employed over flow classes. Although the number of slows to be sorted is reduced by this method, it still is partially dependent on the efficiency of the sorting algorithm.

AFQ depends on sorting. However, the sorting is executed fewer times which provides a substantial gain in packet latencies, especially when the network traffic gets heavy.

3 Generalized Processor Sharing

A GPS server is considered the ideal scheduling discipline providing perfect fairness and isolation among competing flows. In this section, we review the basic idea behind it, discuss its advantages and highlight why a policy based on it is not the most suitable for embedded networks.

3.1 Model

A Generalized Processor Sharing (GPS) server [27] is work conserving and operates at a fixed rate T. By work conserving, we mean that the server must be busy if there are packets waiting in the system. It is characterized by positive real numbers $\phi_1, \phi_2, \ldots, \phi_N$. Let $S_i(\tau,t)$ be the amount of session i traffic is served in an interval $(\tau,]$. A session is backlogged at time t if a positive amount of that session's traffic is queued at time t. Then, a GPS server is defined as one for which

$$\frac{S_i(\tau,t)}{S_j(\tau,t)} \geq \frac{\phi_i}{\phi_j}, j \neq i, j = 1,2,\ldots N$$

for any session i that is continuously backlogged in the interval $(\tau, t]$.

Summing over all sessions j:

$$S_i(\tau,t)\sum_j \frac{1}{\phi_j} \geq (t-\tau)r\phi_i$$

and session i is guaranteed a rate of

$$g_i = \frac{\phi_i}{\sum_j \phi_j} r$$

3.2 Advantages

GPS is an attractive multiplexing scheme for a number of reasons:

- If r_i is the average rate of session i, then, as long as $r_i \leq g_i$, the session can be guaranteed a throughput of ρ_i independent of the demands of the other sessions. In addition to this throughput guarantee, a session i backlog will always be cleared at a rate $\geq g_i$.
- The delay of an arriving session i bit can be bounded as a function of the session i queue length, independent of the queues and arrivals of the other sessions.
- By varying the ϕ_i's, we have the flexibility of treating the sessions in a variety of different ways as long as the combined average rate of the sessions is less than r.
- It is possible to make worst-case network queueing delay guarantees when the sources are constrained by leaky buckets.

3.3 Scope for improvement in EmNets

As already explained, GPS-based scheduling algorithms emulate the bit-by-bit round robin principle on a reference GPS server and schedules the packet with the earliest deadline. It is completely ignorant of the deadline imposed by the application. Consider three backlogged flows f_0, f_1 and f_2 with packet sizes of 1500, 500 and 500 bytes respectively. For simplicity, let us assume that all flows are intended for different applications running on the same host. A GPS scheduler will schedule the flows f_1 and f_2 before f_0. At the application level, it is quite possible that f_0 has a much tighter deadline and needs to be scheduled first. However, a GPS scheduler will not be able to address this.

A GPS scheduler calculates the timestamp of departure for *every* packet. In a real-time data flow, there is a relative timing information that holds for consecutive packets. For example, imagine a high definition video stream in a video sensor network. Once the stream starts playing at the destination, then every subsequent packet has to arrive within a particular time. If this insight can be exploited, then it might

be possible to reduce the number of packets for which the deadline is calculated. While this does not reduce the time complexity, it does reduce the constant factor (or the number of iterations), which depending on bandwidth and the length of the flow, can only improve the performance of a GPS scheduler.

We exploit precisely these two opportunities in a very simple scheduling algorithm which we call Approximate Fair Queueing (AFQ). AFQ schedules flows. We assume that every packet has a deadline specified by the sender. Based on that, a node computes the deadline of only the first packet in the flow. The scheduler schedules a flow with the earliest deadline. Once a flow is scheduled, it sends the maximum number of packets allowed depending on its share of the available bandwidth. We now provide a detailed explanation of our approach.

4 Approximate Fair queueing

4.1 Model

There are N backlogged flows f_1 to f_N. A flow is considered backlogged if it has at least one packet to send. The flows share an output link with bandwidth B. Flow f_i has a requested bandwidth of r_i. The actual allocated bandwidth b_i, is the fraction of the total bandwidth allocated to it, normalized with respect to the total of all requested bandwidths.

$$b_i = \frac{r_i}{r_1 + r_2 + \cdots + r_N} \cdot B$$

The above ensure that

$$\sum_{i=1}^{i=N} b_i \leq B$$

For every flow f_i, $b_i < B$. Otherwise, scheduling is trivial as there is only one flow that is allocated the entire bandwidth.

The *weight* w_i of flow f_i is defined as its allocated bandwidth normalized with respect to the total bandwidth of the output link. Thus,

$$w_i = \frac{b_i}{B}$$

It follows from the above that

$$\sum_{i=1}^{i=N} w_i \leq 1$$

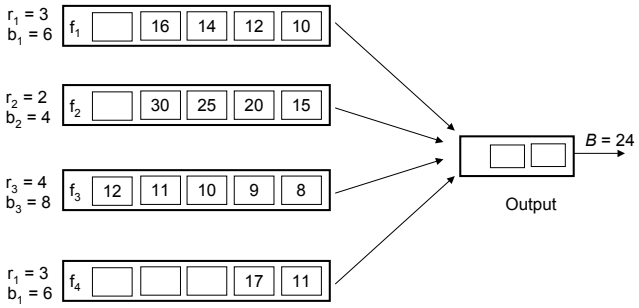

Figure 1: There are 4 flows sharing an output link of bandwidth B=24. Flow f_1 has a requested bandwidth of 3. It is allocated a bandwidth = $\frac{4}{3+2+4+3} \cdot 24 = 6$. The allocated bandwidths of other flows are computed similarly.

Figure 1 shows an example output link being shared by four flows. The output link has a bandwidth $B = 24$. The requested bandwidth r_i and the allocated bandwidth a_i of flow f_i are shown. The packets are numbered by their deadline. So for example, the 1^{st} packet in f_4 has a deadline of 11, the next one 13 and so on.

4.2 Algorithm

In a manner similar to GPS algorithms, AFQ computes a deadline and schedules the one with the earliest deadline. However, in order to give priority to the flows with an earlier application deadline and to reduce the number of computations, it does things slightly differently.

Unlike a GPS scheduler, the AFQ scheduler schedules flows. Each flow has a deadline associated with

Table 1: Flow specifications

Name	Description	Rate
HDTV	A single channel of High Definition resolution MPEG2 encoded video	20 Mbps [2]
SDTV	A PAL or NTSC-equivalent Standard Definition video	3 Mbps
Stereo	A multichannel DolbyDigital 'AC-3' audio with a maximum 13.1 channels	6 Mbps [1]
Standard Audio	An audio channel encoded with Advanced Audio Coding format	128 Kbps

it. The deadline of a flow is the deadline of the packet at the head of its queue. AFQ simply schedules the flow with the earliest deadline. When a flow is scheduled, it is given a credit which is proportional to the flow's allocated bandwidth. The credit decides the number of bytes that the flow is actually allowed to send. The credit scheme is similar to deficit round robin. In deficit round robin, each flow is assigned a fixed credit which is larger than the smallest size packet. In contrast, we assign a credit which is proportional to its weight.

$$c_i \propto b_i$$

Let a scheduled flow f_i have p_1, p_2, ..., p_n be n queued packets of fixed size p. Then the number of packets, num_i that will be sent by f_i is given by

$$num_i = \lfloor \frac{c_i}{p} \rfloor$$

Figures 2, 3, and 4 show an example. Assume each packet requires two units. The first flow to be scheduled is f_3. Since the assigned bandwidth $b_3 = 8$, f_3 gets to send 4 packets once scheduled. The deadline of f_3 is now set to 12. The next flow to be scheduled is f_1. As can be seen, AFQ, like GPS, is work conserving.

5 Evaluation

In this section we present a thorough evaluation of AFQ by comparing it to Wf2q which is a very common implementation of a GPS scheduler. We simulate a single node system similar to the one shown in Figure 1. There are N flows that share the output link. We vary N from 100 to 1000. Each flow sends 100,000 packets.

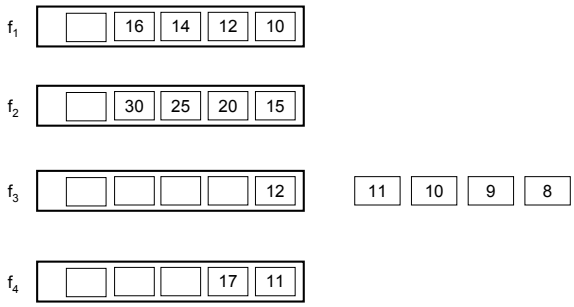

Figure 2: Since f_3 has the earliest deadline, it is scheduled. Since the allocated bandwidth of f_3 is 4, it can send 4 packets.

For our benchmark, we use multimedia flows. Specifically, a flow is randomly chosen from one of four types which are described in Table 1. Packets are generated according to the specified rate. Every packet is assigned a deadline of 5 seconds from the time it is produced at the source. The delay experienced by a packet is the sum of its processing delay in executing the scheduling algorithm and the queueing delay. In real life, a packet will be subject to propagation and transmission delays too. But since they are independent of the queueing algorithm, we ignore them in this paper. We assign a fixed delay of 1 ms (for one execution) to both algorithms irrespective of the number of flows. Recall that both algorithms require sorting which has a complexity of $O(N)$. This we thought was fair since we do not reduce the complexity of the algorithm *per se*. Our benefit comes from the fact that AFQ is executed fewer times, a fact that makes a telling impact when the number of flows increase. We will compare the latency, throughput and fairness characteristics of the two algorithms.

Fairness is a very general notion and might mean different things in different contexts. Here we compare fairness in terms of two well known measures.

Jain's fairness index is a measure of fairness that is independent of the population size, is independent of the scale, is bounded and is continuous [25]. Jain's fairness index is a measure of the fairness in resource allocation. In our system, the resource is bandwidth. However, bandwidth directly affects throughput. Since

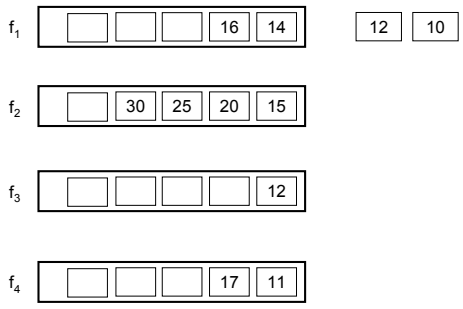

Figure 3: Of the remaining flows, f_1 has the earliest deadline. Based on its allocated bandwidth, it will send 2 packets.

the latter is what matters ultimately, we compute fairness in terms of the throughput. If t_i is the throughput of flow f_i, then Jain's fairness is defined as:

$$Fairness = \frac{(\sum t_i)^2}{(—f— \cdot \sum t_i^2)}$$

Figure 5 shows how the two algorithms compare as the number of flows are increased. At lower numbers, their performance is similar. However, as the number of flows increase, it appears that AFQ gets fairer. This claim needs further clarification. W2fq performs equally well, and sometimes better, as compared to AFQ in terms of how fairly raw bandwidth is allocated. But AFQ, as we will see later, has a much higher throughput because of lower processing time. This become more acute as the number of flows increase and hence AFQ performs better.

The fairness measure of **Bennet-Zhang** (also called worst-case fairness) is a more refined notion of fairness. Rather than comparing the relative measure among flows, it compares the service received by a single flow f_i to the service it would receive in the ideal case. The ideal case is defined as that when f_i has exclusive access to an output link of bandwidth r_i. Note that this case is identical to the GPS model. Service, in this context, is measured by the throughput. We present the results for 100, 500 and 1000 flows

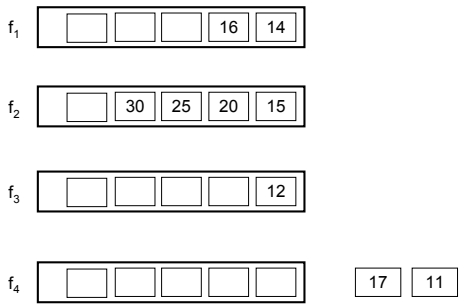

Figure 4: Now f_4 gets scheduled. Again, its allocated bandwidth allows it to send 2 packets.

in Figures 6, 7 and 8 respectively. As can be seen, AFQ outperforms Wf2q across the spectrum. AFQ is cognizant of the packet deadline and allows the important ones to go first, resulting in a vast majority of flows to experience a very high fairness measure. Wf2q is generally varies within a much smaller range than AFQ indicating that the former is more predictable than the latter.

We next measure what we define as the **startup latency**. A multimedia stream has a strict deadline within which successive packets must be received. Packets arriving after the deadline are considered useless and are treated as lost; and the loss of a certain number of packets will seriously affect the quality of the application. Startup latency is a measure of how long the destination has to wait before it can start playing the stream without losing a single packet due to jitter violations. More specifically, for every flow, let p packets be generated at a the rate of r packets/second starting at time t_s. Let t_f be the time that the final packet is received. Then,

$$startup\ latency = t_f - (p * r) - t_s$$

According to the above, the lower value of startup latency, the better the scheduling algorithm. It also follows that it is possible for startup latency to have a negative value. In this case, it means that all packets arrived at the destination before the specified deadline (5 seconds in our experiments). We present the results for 100, 500 and 1000 flows in Figures 9, 10 and 11 respectively. AFQ shows two very desirable properties:

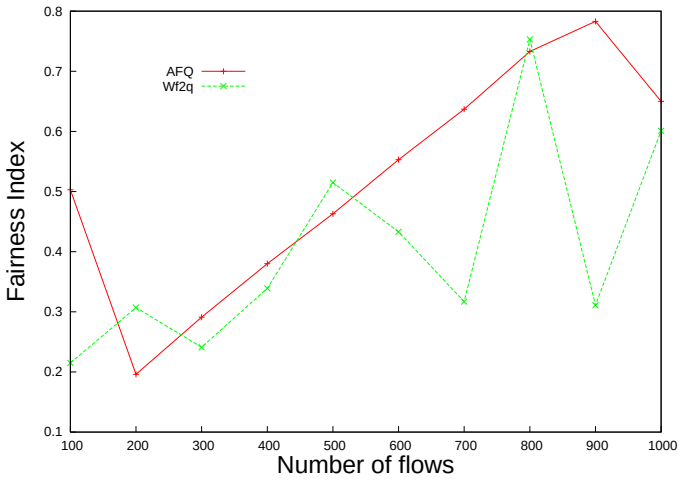

Figure 5: Jain's fairness

- AFQ consistently outperforms Wf2q. The margin increase with the increase in the number of flows.
- The performance of AFQ remains relatively stable even with an increase in the number of flows.

Figures 12 and 13 show the CDF of the end-to-end packet latencies of all packets for all flows.

We finally turn our attention to the **throughput** which in this paper is measured by the total number of bits that were delivered from the time the first packet was generated at the source to the time when the last packet was received at the destination. As before, we present the results for 100, 500 and 1000 flows in Figures 14, 15 and 16 respectively. AFQ shows an order of magnitude better throughput than Wf2q. Part of this is due to the more intelligent scheduling. But the biggest impact comes from the reduction in the number of times that the sorting algorithm is executed. Recall, that in a GPS algorithm, every packet encounters a $O(\log N)$ delay while being inserted into a sorted queue. In AFQ, in contrast, sorting is done once for P packets where P depends upon the allocated bandwidth and the size of the individual packets. This reduction in the expensive sorting algorithm makes a more pronounced impact as the number of flows increase.

In summary, it can be said that the advantage of AFQ are the following.

- It has faster processing leading to a higher throughput.

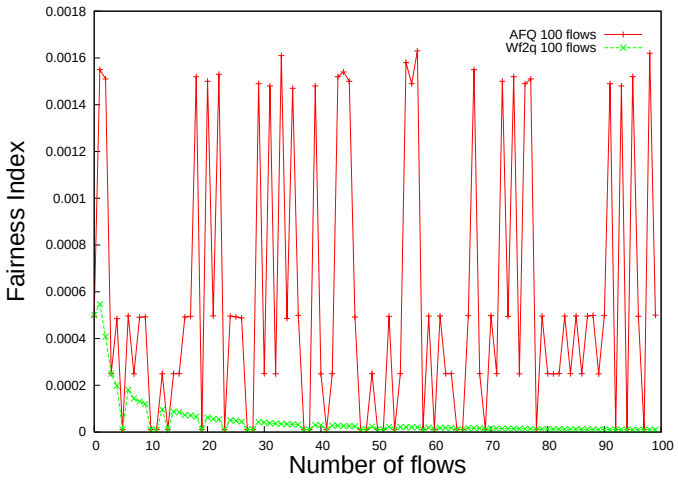

Figure 6: Bennet-Zhang fairness - 100 flows.

- It is deadline-aware which allows far more flows to meet their deadlines. In systems, where streaming is implemented (that is playing back a stream when only a part of it has been received), AFQ will provide a much shorter startup latency.

The drawback of AFQ is its wider variation making it's actual measured performance less predictable than that of Wf2q. Note that if variation is measured in terms of the ability to respect deadlines, then AFQ actually is much more stable and is unaffected by the increase in the number of flows.

6 Conclusion

In this paper, we proposed a new application deadline-aware scheduling algorithm called the Approximate Fair Queuing. The key idea is to schedule flows, and not packets, based on the deadline of the packet at the head of the flow. This is different from the concept of deadline that is used in conventional GPS scheduling algorithms. Once scheduled, a flow sends multiple packets based on its share of the bandwidth and the size of the individual packets. This considerably reduces the frequency of execution of the expensive sorting algorithm and considerably impacts the application-level deadline.

Through extensive simulation, we show that Approximate Fair Queueing is better at meeting application

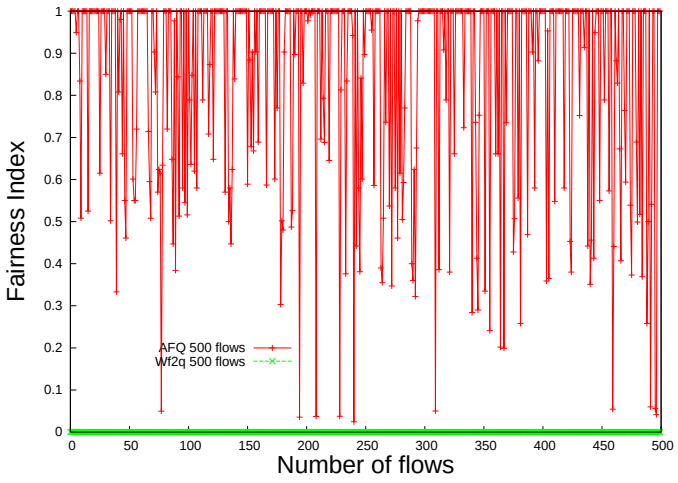

Figure 7: Bennet-Zhang fairness - 500 flows.

deadlines by taking that into consideration while making scheduling decisions. Further, by reducing the overall time of execution of the scheduling algorithm, it significantly improves the throughput. Finally, it shows remarkable stability when the number of flows increase by an order of magnitude. As such, we believe that Approximate Fair Queueing is better suited for embedded networks.

References

[1] Dolby laborotories inc. www.dolby.com.

[2] Moving picture experts group, october 2006 http://www.chiariglione.org/mpeg.

[3] www.cisco.com. cisco gsr.

[4] S. S. abd G. Varghese and G. Chandranmenon. Leap forward virtual clock: An o(loglogn) queuing scheme with guaranteed delays and throughput fairness. In *Infocom*, 1997.

[5] J. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. In *SIGCOMM*, 1996.

[6] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. *Internet RFC 1633*, June 1994.

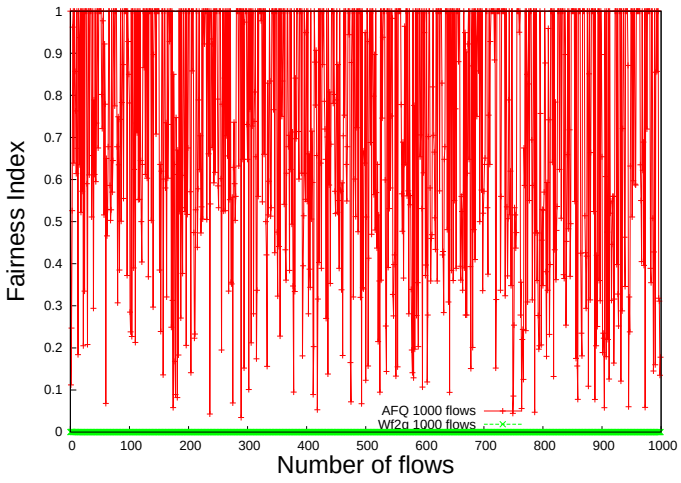

Figure 8: Bennet-Zhang fairness - 1000 flows.

[7] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (rsvp) – version 1 functional specification. *Internet RFC 2205*, September 1997.

[8] S. .Cheung and C. Pencea. Bsfq: Bin sort fair queuing. In *INFOCOM*, 2002.

[9] G. Chuanxiong. An o(1) time complexity packet scheduler for flows in multi-service packet networks. In *SIGCOMM*, 2001.

[10] D. Clark. The design philosophy of darpa internet protocols. In *ACM SigComm*, 1998.

[11] D. Clark and W. Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking*, 1998.

[12] J. Davidson, M. Bhatia, S. Kalidindi, S. Mukherjee, and J. Peters. *VoIP: An In-Depth Analysis*. Cisco Press, 2006.

[13] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. In *SIGCOMM*, 1989.

[14] D.Saha, S. Mukherjee, and S. Tripathi. Carry-over round robin: A simple cell scheduling mechanism for atm networks. *IEEE/ACM Transactions on Networking*, 1998.

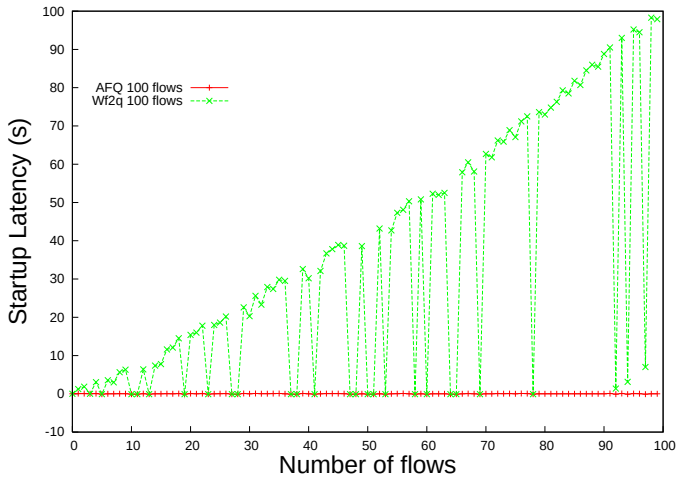

Figure 9: Latency - 100 flows.

[15] N. Figueira and J. Pasquale. Leave-in-time: A new service discipline for real-time communications in a packet-switching network. In *SIGCOMM*, 1995.

[16] N. Figueira and J. Pasquale. An upper bound on the delay for the virtual clock service discipline. *IEEE/ACM Transactions on Networking*, 1995.

[17] N. Figueira and J. Pasquale. A schedulability condition for deadline-based service disciplines. *IEEE/ACM Transactions on Networking*, 1997.

[18] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking*, 1999.

[19] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1993.

[20] S. Floyd and V. Jacobson. Link-share and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 1995.

[21] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOM*, 1994.

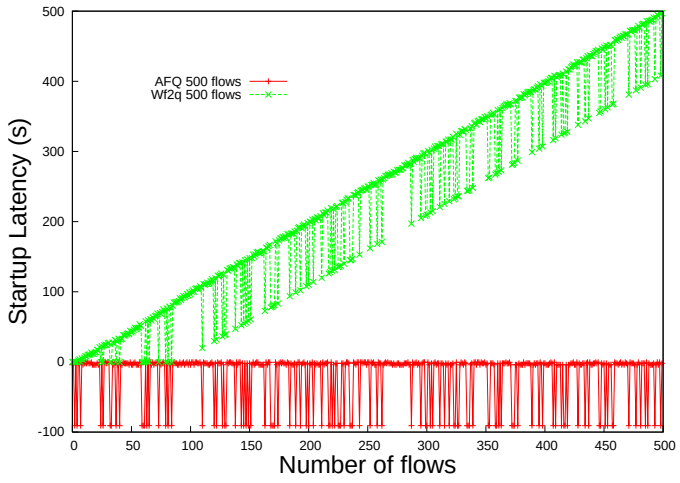

Figure 10: Latency - 500 flows.

[22] P. Goyal and H. Vin. Generalized guaranteed rate scheduling algorithms: A framework. *IEEE/ACM Transactions on Networking*, 1997.

[23] J. B. H. and Zhang. Wf2q : Worst case fair weighted fair queuing. In *INFOCOM*, 1996.

[24] J. Hasan and T. N. Vijaykumar. Dynamic pipelining:making ip-lookup truly scalable. *ACM SIGCOMM Computer Communication Review*, 35(4), 2005.

[25] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared systems. *DEC Research Report TR-301*, 1984.

[26] L. Lenzini, E. Mingozzi, and G. Stea. Aliquem: A novel drr implementation to achieve better latency and fairness at o(1) complexity. In *IWQoS*, 2002.

[27] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Transactions on Networking*, 1, 1993.

[28] S. Ramabhadran and J. Pasquale. The stratified round robin scheduler:design, analysis and implementation. *IEEE/ACM Transactions on Networking (TON)*, 14(6), December 2006.

[29] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *SIGCOMM*, 1995.

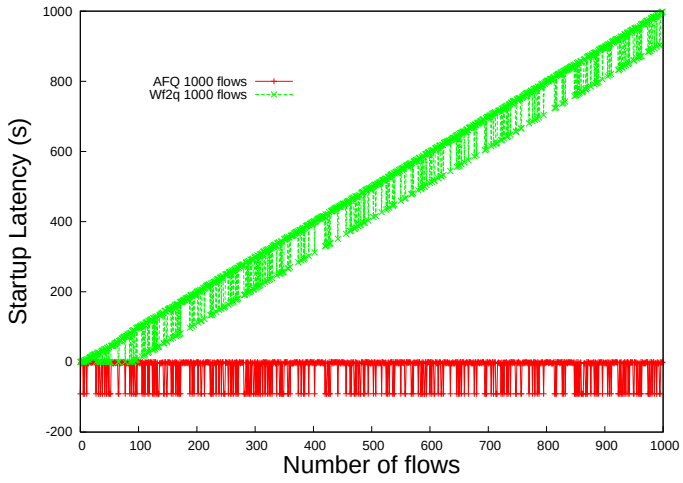

Figure 11: Latency - 1000 flows.

[30] D. Stiliadis and A. Varma. Efficient fair queueing algorithms for packet switched networks. *IEEE/ACM Transactions on Networking*, 1998.

[31] D. Stiliadis and A. Varma. Rate proportional servers: A design methodology for fair queueing algorithms. *IEEE/ACM Transactions on Networking*, 1998.

[32] J. Xu and R. Lipton. On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms. In *SIGCOMM*, 2002.

[33] L. Zhang. A new architecture for packet switching network protocols. *PhD thesis, Massachusetts Institute of Technology*, 1989.

[34] L. Zhang. Virtual clock: A new traffic control scheme for packet switching networks. In *SIGCOMM*, 1990.

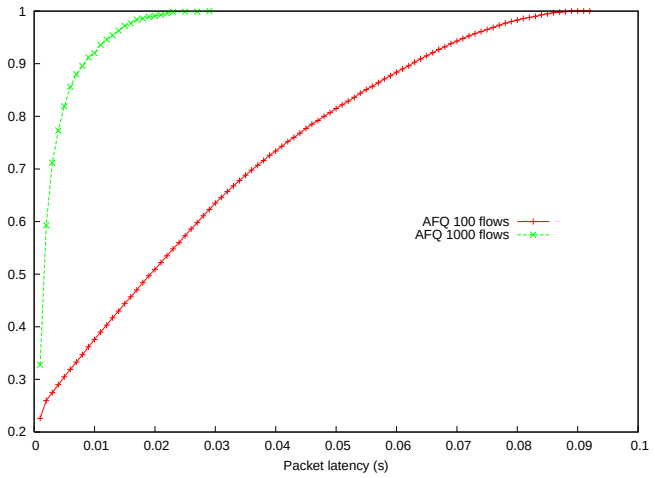

Figure 12: CDF of packet latencies - AFQ.

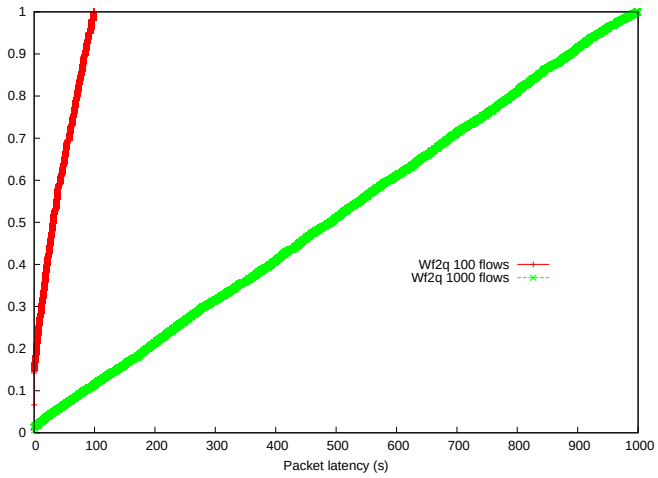

Figure 13: CDF of packet latencies - W2FQ.

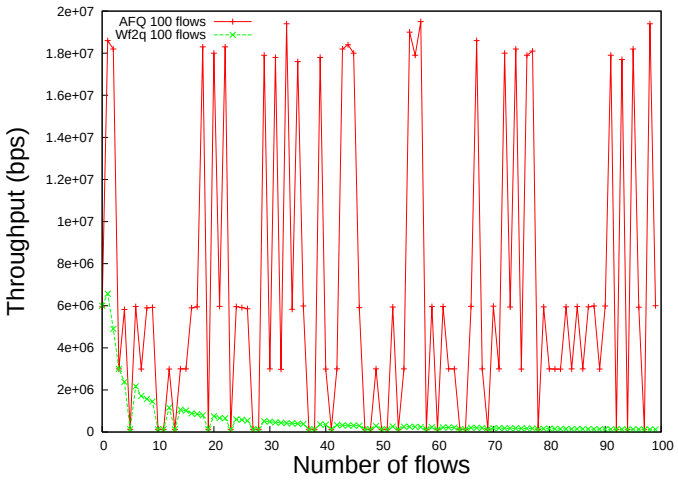

Figure 14: Throughput - 100 flows.

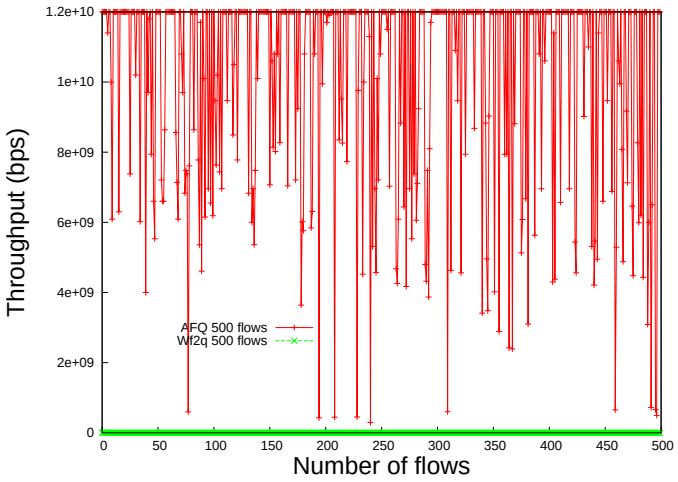

Figure 15: Throughput - 500 flows.

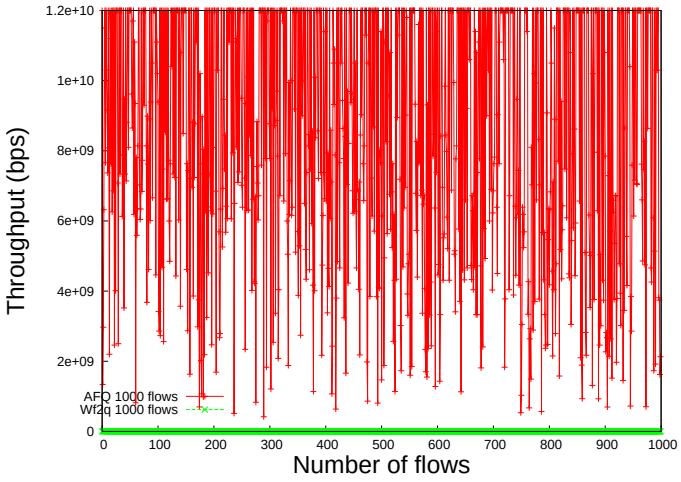

Figure 16: Throughput - 1000 flows.