A Flow Scheduler Architecture

Dinil Mon Divakaran, Giovanna Carofiglio, Eitan Altman, Pascale Primet

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Flow Scheduler Architecture

Dinil Mon Divakaran, Giovanna Carofiglio, Eitan Altman, Pascale Vicat-Blanc Primet

N° 7133

Dec. 2009

ISSN 0249-6399 ISRN INRIA/RR--7133--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

A Flow Scheduler Architecture

Dinil Mon Divakaran, Giovanna Carofiglio, Eitan Altman, Pascale Vicat-Blanc Primet

Thème : THCom
Équipe-Projet RESO

Rapport de recherche n° 7133 — Dec. 2009 — 17 pages

Abstract: Scheduling flows in the Internet has sprouted much interest in the research community, leading to the development of many queueing models, capitalizing on the heavy-tail property of flow size distribution. Theoretical studies have shown that 'size-based' schedulers improve the delay of small flows without almost no performance degradation to large flows. On the practical side, the issues in taking such schedulers to implementation have hardly been studied. This work looks into practical aspects of making size-based scheduling feasible in future Internet. In this context, we propose a flow scheduler architecture comprising three modules — Size-based scheduling, Threshold-based sampling and Knockout buffer policy — for improving the performance of flows in the Internet. Unlike earlier works, we analyze the performance using five different performance metrics, and through extensive simulations show the goodness of this architecture.

Key-words: Scheduling, Sampling, QoS, Future Internet, Architecture

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Architecture d'un ordonnanceur de flux

Résumé : L'ordonnancement des flux dans l'Internet suscite un vif intérêt dans la communauté scientifique et qui a conduit à l'élaboration de nombreux modèles de files d'attente exploitant la propriété "queue lourde" des distributions des tailles de fichiers. Des études théoriques ont montré que les ordonnanceurs basés sur la taille des flux, pouvaient améliorer le délai de transfert de petits flux, sans induire de dégradation des performance des larges flux. Sur le plan pratique, les questions relatives à la mise en oeuvre de tels ordonnanceurs n'ont guère été étudiées. Avec pour objectif d'explorer cet aspect pratique, cet article propose une architecture d'ordonnanceur de flux composé de trois modules - un ordonnancement basé sur la taille, une politique de définition des seuils fondée sur l'échantillonnage et une gestion de tampon avec un mécanisme de knockout- pour améliorer la performance des flux dans l'Internet. Nous analysons les performances relativement à cinq métriques différentes par des simulations approfondies qui mettent en évidence les intérêts de cette architecture.

Mots-clés : Ordonnancement, Échantillonnage, QoS, Internet du Futur, Architecture

1 Introduction

Recent works have advocated the importance of networks being 'flow-aware'. Bonald *et al.* have listed the need for having a flow-aware architecture [1]. In a flow-aware network, the performance is measured at flow level. This is in line with the utility of end-users, where e.g., the delay of small flows, throughput of large flows, instantaneous rate of streaming traffic etc. are most often more important than packet-level QoS metrics.

In this context, our goal is to come up with a flow scheduler architecture for improving the delay performance of small (and middle size) flows. The current Internet architecture has a FCFS scheduler and Droptail buffer at each of its nodes. These, along with the fact that most of the flows in the Internet are carried by TCP, makes this current architecture biased against small TCP flows for the following reasons. (i) A packet loss to a small flow most often results in a timeout due the small window size; whereas, a large flow is most probably in the congestion avoidance phase, and hence has large congestion window size. Therefore packet losses are usually detected using duplicate ACKs, instead of timeouts, thus avoiding slow-start. (ii) The increase in round trip time (RTT) due to large queueing delays hurts the small flows more the than large flows. Again, for the large flows, the large window size makes up for the increase in RTT; whereas, this is not the case for small flows.

These problems faced by small flows being well-known, researchers have explored scheduling algorithms that give priority to small flows. These flow scheduling algorithms range from SRPT (Shortest Remaining Processing Time) to LAS (Least Attained Service) to MLPS (Multi-level Processor Sharing) scheduling mechanisms [2]. While SRPT requires the knowledge of flow size in advance, LAS is a 'blind' scheduling policy — requires no in-advance information on flow size. These differentiating policies perform better in terms of delay, when compared with the naive PS (processor sharing) system[1]. SRPT discipline is proved to be optimal with respect to the mean delay, among the anticipating disciplines [3]; and among the blind ones, LAS is proved to be optimal if the service-time distribution has a decreasing hazard rate (as mentioned in [4]). The MLPS scheduling discipline is a generalized version with high flexibility, having N different priority levels distinguished by $N-1$ thresholds, and strict priority among these levels. The levels are used to classify flows based on their running sizes. We refer the readers to [2] for in-depth details.

While scheduling algorithms give priority in time, buffer management policies give priority in space. There has been previous work that showed the gain in performance attained by giving space priority to small flows [5]; but it is a stand-alone concept that does not consider giving time-priority to small flows. To the best of our knowledge, LAS scheduling policy is the only policy that gives space priority to packets of small flows [6], thereby giving priority in both time and space. It does so by inserting incoming packet in the appropriate position (depending on the attained service counter) and dropping from the tail whenever the buffer is full. But, it has been observed that LAS is unfair to very large flows [7]. Moreover, it is challenging to perform a strict ordering of packets of each flow at high line rates.

[1] At the flow level, the queues in the Internet are generally modelled as an $M/G/1-PS$ system, even when the queue is served using a FCFS policy at the packet level.

This work proposes a flow scheduler architecture, that gives priority in time as well as space to small flows, and uses sampling for performing size-based scheduling. To be precise, our flow scheduling architecture combines three essential modules that help in improving the delay performance of flows:

1. Generalized size-based scheduling;
2. Threshold-based sampling;
3. Knockout buffer policy.

We detail the architecture in Section 3. We perform extensive simulations and compare different performance metrics to show how each of these three strategies contributes in improving the performance of small flows, without affecting the performance of large flows. Unlike most previous works, where the performance was analyzed using just one metric (usually the conditional mean response time), we consider five different metrics. They are:

1. Conditional mean completion time of small flows;
2. Number of timeouts encountered by small flows;
3. Mean completion time for range of flow sizes;
4. Mean completion time for small flows, large flows and all flows;
5. Maximum completion time of small flows.

The rest of the paper is organized as follows. The problem is motivated as the related works are discussed in Section 2. The goal of the simulations and the setting are described in Section 4. In Section 5, we compare different strategies to show the performance gain in having the Knockout policy. In Section 6, we compare the proposed flow scheduler architecture with other schemes. We conclude in Section 7.

2 Related works and Motivation

This section summarizes some of the previous works that have proposed mechanisms to improve the delay performance at flow level in the Internet. By and large, the improvement is achieved by giving priority to small flows. The literature in this research area being vast, we limit the references to a small but important subset.

A large number of researchers have considered giving priority in time to small flows. These have given rise to the study of scheduling disciplines like SRPT [3], LAS [8] and MLPS [7, 9, 10] disciplines in the context of Internet flows. As said before, SRPT has the disadvantage that it needs to anticipate the sizes of arriving flows. LAS, on the other hand overcomes this problem, besides giving preferential treatment to packets of small flows in queue. But the drawbacks such as unfairness and scalability issue, have motivated researchers to explore other means of giving priority to small flows, one such being the strict $PS + PS$ model proposed in [7].

The $PS + PS$ model, as the name indicates, uses two PS queues, with priority between them. The first θ packets of every flow are served in the higher priority queue (Q_1), and the remaining in the lower priority queue (Q_2). The service discipline is such that, Q_2 is served only when Q_1 is empty. Therefore, it is a strict $PS + PS$ model. This model can be seen as a specific case of MLPS where the number of levels is two, distinguished by threshold θ, and having PS scheduling discipline within each level. This work also takes a step forward in performance analysis of size-based scheduling systems, by analyzing another metric — maximum response time — other than the usual conditional mean response time. In addition, the authors proposed an implementation of this model; but it relies on TCP sequence numbers, requiring them to start from a set of possible initial numbers. This not only makes the scheme TCP-dependent, but also reduces the randomness of initial sequence numbers. Again, this is another work which does not account for space priority for small flows.

Authors of [5] considered prioritizing small flows in space. This is achieved by preferential treating small flows inside the bottleneck queue which implement RIO (RED with In and Out). Small and large flows were assigned different drop functions. To facilitate this, they proposed an architecture where the edge routers mark packets as belonging to small or large flow, using a threshold-based classification. But with priority given only in space, the performance gains in terms of average response times (apart from analyzing the fairness) is not complete.

We observe that most of the works dealing with giving preferential treatment based on the size (or age) assume that the router keeps per-flow information. In fact, this assumption is challenged by the scalability factor, as the number of flows in progress is in the order of hundreds of thousands under a high load. One solution is to use sampling to detect large flows (thus classifying them), and use this information to perfor size-based scheduling. Since the requirement here is only to differentiate between small and large flows, the sampling strategy need not necessarily track the exact flow size. A simple way to achieve this is to probabilistically sample every arriving packet, and store the information of sampled flows along with the sampled packets of each flow [11]. SIFT, proposed in [12], uses such a sampling scheme along with the $PS + PS$ scheduler. A flow is 'small' as long as it is not sampled. All such undetected flows go to the higher priority queue until they are sampled. The authors analyzed the system using the 'average delay' (average of the delay of all small flows, and all large flows) for varying load, as a performance metric. Though it is an important metric, it does not reveal the worst-case behaviours in the presence of sampling. This is more important here, as the sampling strategy has a disadvantage: there can be false positives; i.e., small flows if sampled will be sent to the lower priority queue. In such scenarios, it is necessary to compare other performance metrics, which we listed earlier.

3 Architecture

3.1 The modules

The flow scheduler architecture consists of three modules: a scheduler providing differentiated service based on size, a threshold-based sampling technique to

detect large flows, and a Knockout buffer policy [13] to give space priority to small flows.

3.1.1 Size-Based scheduling:

Since sampling introduces errors in the detection of large flows, thereby permitting misclassification, using a strict priority-scheduling strategy is not advisable. Therefore, we take a generalized model of the strict $PS + PS$ scheduling, called generalized size-based scheduling, or simply SB scheduling. As before, packets of all flows are served in Q_1, as long as the ongoing size is less than θ packets. Once the ongoing flow size crosses θ, it is queued in Q_2. But instead of giving the whole capacity to Q_1, only a fraction of the capacity is assigned to Q_1. That is, the high priority queue is assigned a weight $0 \leq w \leq 1$. If C is the link capacity, Q_1 and Q_2 are serviced at rates wC and $(1-w)C$ respectively, whenever the queues are not empty. If Q_1 is empty, Q_2 is served at full capacity C. We assume that the scenario of Q_2 being empty and Q_1 being non-empty is a rare possibility. Note that, if $w = 1$, this becomes the $PS + PS$ scheduling policy. The scheduling module in the figure is shown as deciding which queue to dequeue based on the parameter w.

3.1.2 Threshold-Based sampling:

For the sampling part, we use the well-studied 'Sample and hold' strategy proposed for detecting large flows [14, 15]. It works as follows. For every sampled packet, a flow entry is created in the flow table if it does not exist. A packet of s bytes is sampled with a probability p, which is expressed in terms of byte-sampling probability β. We have, $p = 1 - (1-\beta)^s$. When a packet arrives, a flow-table lookup is performed. If the arriving packet is found to be part of an existing flow, the flow-size counter in the flow table is updated. Thus, for each sampled flow, there is a counter that maintains the estimated size. This process is performed during every measurement interval. Thresholds are used to reduce false positives, and to preserve continuing large flows across intervals. Observe that the flow table lookup is done for every arriving packet, and size update is performed for every detected flow. This is costly in terms of processing, but reduces the flow table's size considerably, thereby making it possible to use SRAM to store the flow table for efficient lookups.

A useful property of this sampling strategy is that, by choosing an appropriate threshold and using the estimated size of each sampled flow, false positives can be avoided completely. For example, if θ is the threshold used to distinguish small flows from large flows, then a decision criterion such as '*a sampled flow is large only if the estimated size is greater than* θ' will definitely remove false positives (as the actual size can not be lesser than the estimated size, and therefore will be greater than or equal to θ).

3.1.3 Knockout buffer policy:

The third part is the Knockout buffer policy for giving space priority to small flows. Though there is only one single physical queue, it is shared by two virtual queues, one for enqueueing packets of small flows, the other for enqueueing packets of large flows. These correspond to the two queues Q_1 and Q_2 described

earlier. The policy is different from Droptail only during packet discard instants. Upon the arrival of a packet when the physical buffer is full, the Knockout policy operates thus: if the packet belongs to a large flow, it is dropped. If the arriving packet belongs to a small flow, the last packet from Q_2 is 'knocked out' making space for this new packet. In the scenario of Q_2 being empty (i.e., the physical buffer has packets of only small flows), the arriving packet is dropped. Assuming most large flows are carried by TCP, dropping a packet from a large flow is meaningful as it will be retransmitted by the TCP source.

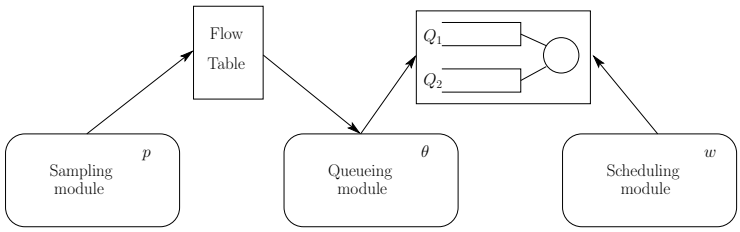


Figure 1: Flow scheduler architecture

Fig. 1 gives a pictorial representation of the architecture. An arriving packet first goes to the sampling module, which does a flow-table lookup. Packet sampling and flow-table update are performed if necessary. The queueing module decides to queue the packet in Q_1 or Q_2 based on the flow-size estimate available from the flow table and the parameter θ. If the physical buffer is full, the Knockout policy is used to select the packet to be dropped. The scheduling module uses the weight parameter w to perform SB scheduling.

3.2 Implementation cost

SB scheduling requires two queues. These can be implemented as virtual queues, on top of the physical queue. The scheduling of packets as such can then be easily implemented by assigning weights to these queues.

For the sampling, an SRAM with sufficient size to hold the flow table is required. There is extra processing for updating the flow size of detected flows. The flow-table lookup can be combined along with route-table lookup.

Knockout policy uses the two virtual queues, Q_1 and Q_2, with Q_1 being the higher priority queue. Observe that a virtual queue can grow to the actual size of the physical queue with the other virtual queue being empty. To be able to knock out an already queued packet from Q_2, the tail of Q_2 needs to be tracked. All these can be achieved if the physical queue is implemented as a link list, and pointers to the head and tail of the two virtual queues are maintained.

4 Simulation

4.1 Goal

The goal of the simulations is to evaluate the performance of the flow scheduler architecture. As described earlier, small flows are biased against large flows when it comes to timeouts. Hence, we are interested in analyzing not only the improvement in delay performance, but also the reduction in number of

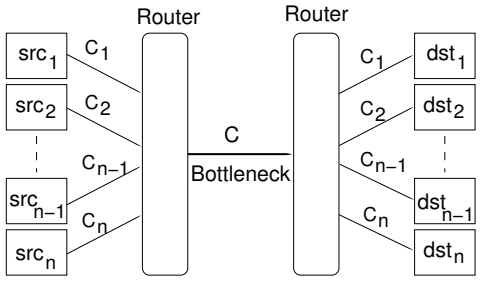


Figure 2: Topology considered for simulations

timeouts faced by small flows. On the other hand, since prioritizing small flows should not adversely affect large flows, the mean completion times of large flows conditioned on their flow sizes are also analyzed. To see the improvement over today's Internet architecture, we compare results with the FCFS scheduler. Along with the FCFS scheduler, the buffer policy used in all simulations here is Droptail (as is the case in Internet nodes), though not stated explicitly in figures.

4.2 Settings

Simulations are performed using NS-2. A dumbbell topology as seen in Fig. 2 was used throughout. The bottleneck link capacity was set to 1 Gbps, and the capacities of the source nodes were all set to 100 Mbps. The delays on the links were set such that the base RTT (consisting of only propagation delays) is equal to 100 ms. The size of the bottleneck queue is set in bytes, as the bandwidth delay product (BDP) for 100 ms base RTT. There were 100 node pairs, with the source nodes generating flows according to a Poisson traffic. The flow arrival rate is adapted to have a packet loss rate of around 1.25%. We consider two traffic scenarios:

Scenario 1: Flow sizes are taken from a Pareto distribution with $\alpha = 1.1$, and mean flow size set to 500 KB.

Scenario 2: 85% of flows are generated using an Exponential distribution with a mean 20 KB. The remaining 15% are contributed by large flows using Pareto distribution with shape $\alpha = 1.1$, and mean flow size set to 1 MB.

All flows are carried by TCP, in particular, using the SACK version. Packet size is kept constant and is equal to 1000 B. For simplicity, we keep the threshold in packets; θ is set to 25 packets in all the scenarios, unless explicitly stated otherwise. For post-simulation analysis, we define 'small flow' as a flow with size less than or equal to 20 KB, and 'large flow' as one with size greater than 20 KB. Here the flow size is the size of data generated by the application, not including any header or TCP/IP information. Also note that, a small flow of 20 KB can take more than 25 packets to transfer the data, as it includes control packets (like *SYN*, *FIN* etc.) and retransmitted data packets.

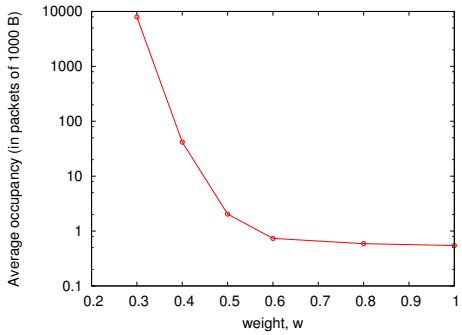


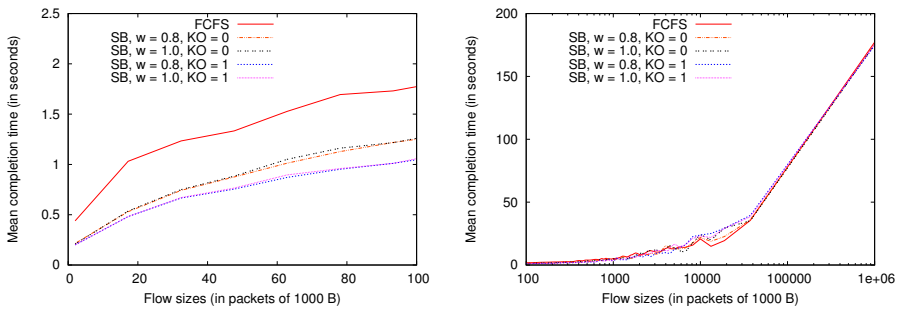
Figure 3: Occupancy for Q_1 for different weights

5 Performance analysis of the scheduler using Knockout

Here we analyze the SB scheduler using the Knockout buffer policy, but without sampling. The focus is to show the importance of having the Knockout buffer policy.

Before choosing the weights, we present an observation. First, it should be noted that, by giving priority to small flows, a policy essentially tries to keep the corresponding buffer for small flows almost empty. With this in mind, we conducted simulations to analyze the average occupancy of Q_1 for different weights. The result is shown in Fig. 3. The number of packets of small flows in queue is almost constant for weights $w \geq 0.6$. Hence, any $w \geq 0.6$ should give close performance for small flows. Dynamically adapting w according to the buffer occupancy being outside the scope of this work, we set w to 0.8 for SB scheduler in our simulations. The other scenario considered is with w set to 1.0. Thus we also analyze the strict $PS + PS$ system. Even when there is no sampling involved, we see that there is no notable gain in using a strict SB scheduler.

5.1 Results with traffic scenario 1



(a) small flows (b) large flows

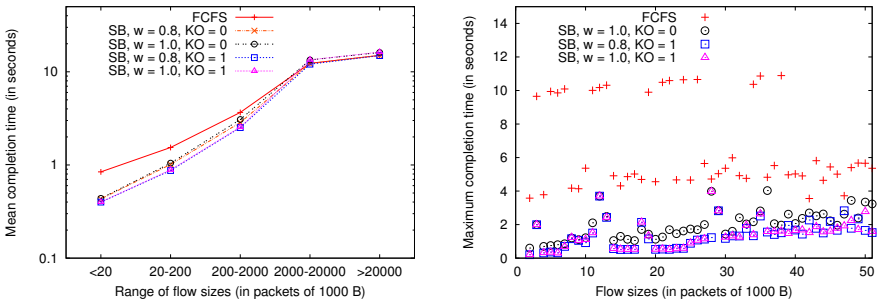
Figure 4: Conditional mean completion time

Metrics	small TOs	small $\overline{CT}$	large $\overline{CT}$	all $\overline{CT}$
FCFS	579	0.8432	2.3294	1.9022
$w = 0.8, KO = 0$	386	0.4325	1.7532	1.3736
$w = 1.0, KO = 0$	449	0.4375	1.8540	1.4468
$w = 0.8, KO = 1$	6	0.3996	1.5715	1.2347
$w = 1.0, KO = 1$	5	0.3997	1.6219	1.2706

Table 1: Comparison on different metrics.

Fig. 4(a) shows the mean completion time conditioned on the flow sizes, for small flows. The naive packet-level FCFS scheduling policy is shown as a comparison. The other curves correspond to SB scheduling with different weights, and with and without the Knockout policy. A value of '0' for KO implies that the Knockout policy is not in use, and '1' implies the contrary. The figure reveals that Knockout policy performs better as it guarantees buffer space for packets of small flows, as long as there are some packets in queue contributed by large flows that can be knocked out. With this metric, there is no notable difference using a weight of 0.8 or 1.0.

Fig. 4(b) indicates that the large flows are not affected by giving priority to small flows (both in space and time). In fact, it can be seen in Fig. 5(a) that the SB scheduler with $w = 0.8$ and $KO = 1$, gives the same mean delay for very large flows, as does the FCFS scheduler. Fig. 5(a) plots the mean completion time of flows within different size ranges (e.g., 0-20 packets, 21-200 packets etc.). The mean values show that, in general, the SB scheduler also performs better for medium flows (those with a size around 2000 packets).



(a) Mean completion time (b) Maximum completion time

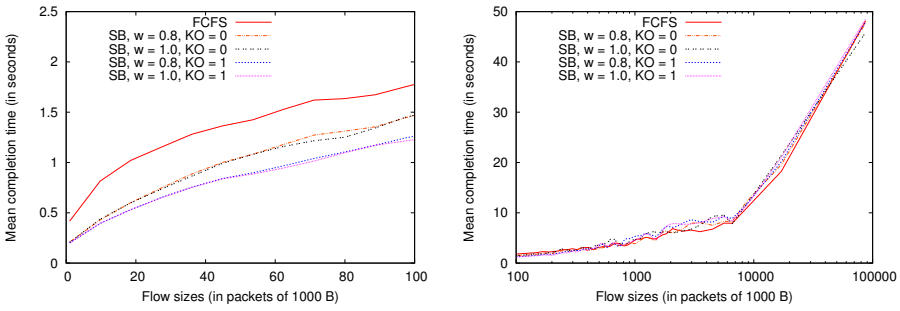
Figure 5: Other metrics

Next, we compare the total number of timeouts faced by small flows in each of these settings. Table 1 shows this, along with the mean CT (indicated by $\overline{CT}$) for small, large and all flows. We see that the timeout is highest for FCFS, followed by the schedulers without Knockout. This happens as some of the flows in Q_1, after being served with priority for the first θ packets, come back with more packets (due to a larger congestion window) and join Q_2, thereby increasing the total buffer occupancy. Without space priority, the packets of small flows are dropped when the buffer is full. With the Knockout policy, we

see that the timeouts are brought down tremendously as the packets of small flows are the last to experience drops. Fig. 5(b), which plots the worst-case completion time per flow size for small flows, also supports the necessity of giving space priority in addition to time priority (the figure does not plot the scenario of $\{w = 0.8, \text{KO} = 0\}$ for better clarity).

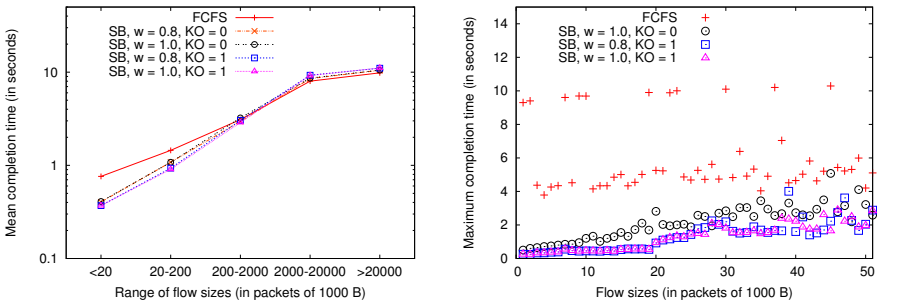
Comparing the mean CTs, it can be noted that the Knockout policy gives better results for all the means, compared to those without Knockout policy. Note that the prioritized service enjoyed by the first θ packets of a large flow helps in having a 'quicker' slow-start phase when compared with the FCFS-Droptail system. Similarly, non-strict schedulers give better performance (in terms of means) for large flows, compared to the strict counterparts (both with KO =0, and K0 = 1). At the same time, the mean CT of small flows remain almost the same. With these comparisons, it becomes clear that a non-strict scheduler with Knockout buffer policy outperforms a strict scheduler (strict $PS + PS$) without Knockout buffer.

5.2 Results with traffic scenario 2



(a) small flows (b) large flows

Figure 6: Conditional mean completion time



(a) Mean completion time (b) Maximum completion time

Figure 7: Other metrics

The results are similar for the second traffic scenario. Figures 6(a) and 6(b) give the conditional mean completion times for small and large flows, respectively. The mean completion times for different flow ranges are plotted in Fig.

Metrics	small TOs	small $\overline{CT}$	large $\overline{CT}$	all $\overline{CT}$
FCFS	792	0.7603	1.7491	1.2168
$w = 0.8, KO = 0$	768	0.4044	1.4214	0.8738
$w = 1.0, KO = 0$	879	0.4091	1.4542	0.8915
$w = 0.8, KO = 1$	14	0.3698	1.3101	0.8039
$w = 1.0, KO = 1$	15	0.3699	1.2706	0.7856

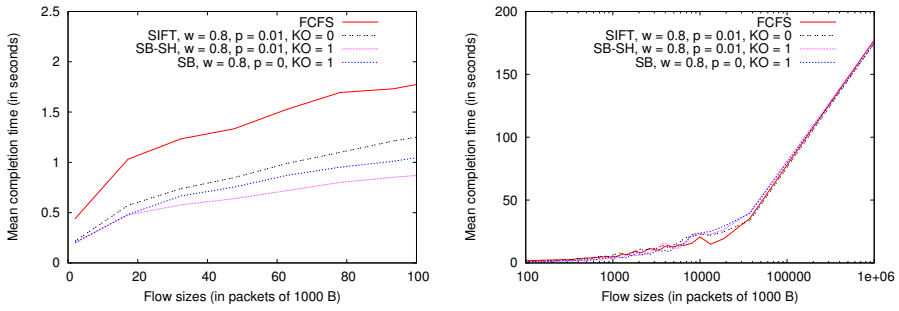
Table 2: Comparison on different metrics.

7(a), and the graph of worst-case completion time per flow size is seen in Fig. 7(b). Table 2 compares the timeouts and mean flow completion times. The table shows that the mean CT of large flows are best with the strict policy. This resulted as the traffic generated didn't have very large flows. The maximum flow size in this scenario was $87,362$ packets, whereas in the previous scenario the maximum flow size was $1,331,365$ packets.

6 Performance analysis of the scheduler with sampling

This section analyzes the performance of the flow scheduler architecture, which combines the SB scheduler, the threshold-based sampling strategy and the Knockout buffer policy. For the scheduler, we set the weight w to 0.8. The results are compared with the SIFT scheme [12]. Note that SIFT does not use the Knockout policy; nor does it use a threshold to classify large flows. Instead, a sampled flow is considered 'large' and send to Q_2; all other undetected flows go to Q_1. To see the degradation due to sampling, we also compare these schemes with the basic SB scheduling scheme (with no sampling). The packet-sampling probability is set to 1/100 in the sampling schemes of both SIFT and our flow scheduler architecture. In the figures below, the name 'SB-SH' represents our scheme, coming from Size-Based scheduling using 'Sample and Hold'.

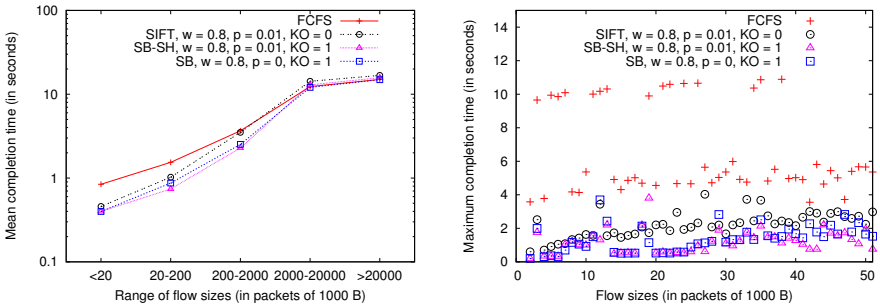
6.1 Results with traffic scenario 1



(a) small flows (b) large flows

Figure 8: Conditional mean completion time

Figures 8(a), 8(b), 9(a) and 9(b) show the results. The conditional mean completion time curves for small flows in Fig. 8(a) reveal that sampling-cum-scheduling strategies (including SIFT) give improved performance for small flows in comparison to FCFS scheduling. This is an anticipated result, as most small flows go undetected and get prioritized. Even the maximum delay experienced by small flows using sampling-cum-scheduling is lesser as seen in Fig. 9(b). In the figure, it can be noted that the completion time in SIFT is sometimes close to FCFS. These are cases when SIFT samples small flows early, and de-prioritizes them. Between the sampling-cum-scheduling strategies, SB-SH scheme is seen to give smaller delay to small flows than SIFT, both in the mean case and in the worst case.



(a) Mean completion time (b) Maximum completion time

Figure 9: Other metrics

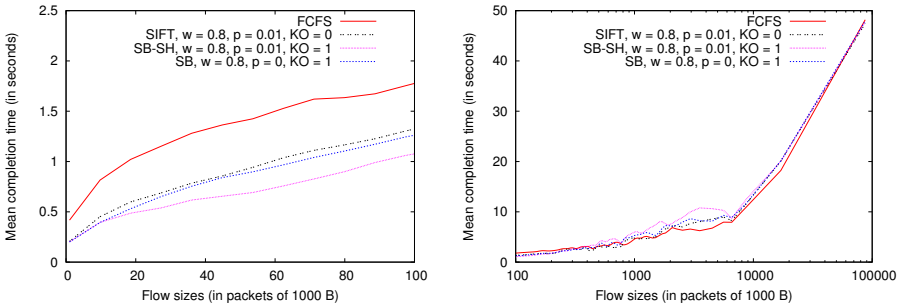
Metrics	small TOs	small $\overline{CT}$	large $\overline{CT}$	all $\overline{CT}$
FCFS	579	0.8432	2.3294	1.9022
$p = 0.01, KO = 0$, SIFT	502	0.4585	1.9483	1.5200
$p = 0.01, KO = 1$, SB-SH	5	0.3998	1.4406	1.1414
$p = 0.0, KO = 1$, SB	6	0.3996	1.5715	1.2347

Table 3: Comparison on different metrics.

Additional metrics are compared in the Table 3. In all the size-based schedulers, the weights are the same ($w = 0.8$), and hence not made explicit in the table. The SIFT scheme induces large number of timeouts for small flows, as it gives no space priority to the packets of small flows. In addition, a small flow that is sampled, gets de-prioritized in SIFT, leaving it to compete with the large flows. This is also clear from Fig. 9(b), which plots the maximum delay per size for small flows. From Fig. 9(a) and Table 3, it is seen that the delay for large flows is higher in SIFT than in the SB-SH scheme. Observe that we have the same sampling probability for both schemes. This means, a flow once sampled is de-prioritized immediately in the SIFT scheme; whereas a sampled flow still enjoys priority (both in time and space) for the next θ packets in SB-SH scheme. This helps the large flows to attain a large TCP congestion window faster (than in FCFS and SIFT).

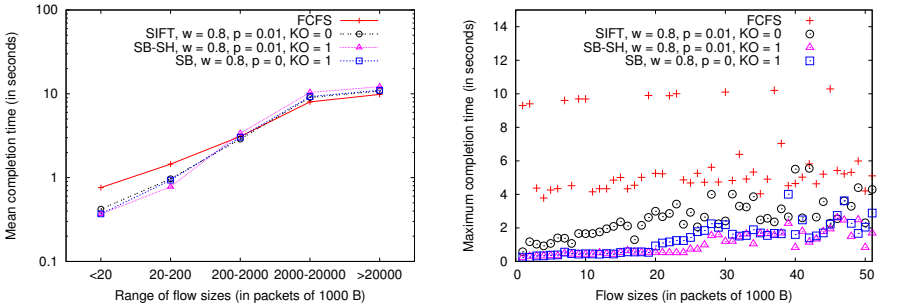
Comparison of SB-SH scheme with the naive SB scheduling (without sampling), which shows that the former performs better than the latter, might appear to be surprising. But in fact, it is not — recall, that we have not tried to find the optimal threshold, θ, in our study here. The false negatives that results from the sampling strategy increase the mean number of packets being served at Q_1, which is similar to increasing the threshold θ. Increasing the threshold, increases the rate at which the TCP congestion window increases (due to negligible queueing delay and very few losses). To confirm, we performed SB scheduling (with Knockout policy, and without sampling) where θ was set to 100 packets. It was found that number of timeouts for small flows was 5, and the mean CTs for small, large and all flows were 0.3997, 1.3740, and 1.0939 respectively. Except for the mean CT for small flow, which is almost the same for all, these values are better than all the results shown in Table 3.

6.2 Results with traffic scenario 2



(a) small flows (b) large flows

Figure 10: Conditional mean completion time



(a) Mean completion time (b) Maximum completion time of small flows

Figure 11: Other metrics

Similar graphs showing the results of simulations with the second traffic scenario is shown in figures 10(a), 10(b), 11(a) and 11(b). Table 4 compares the comparison of the interested metrics. Comparing the values in the table reveals that the results are similar to that with the first traffic scenario. Note

Metrics	small TOs	small $\overline{CT}$	large $\overline{CT}$	all $\overline{CT}$
FCFS	792	0.7603	1.7491	1.2168
$p = 0.01, KO = 0$, SIFT	778	0.4204	1.3195	0.8354
$p = 0.01, KO = 1$, SB-SH	0	0.3671	1.2327	0.7666
$p = 0.0, KO = 1$, SB	14	0.3698	1.3101	0.8039

Table 4: Comparison on different metrics.

that, as the number of small flows is higher in this scenario, SIFT gives a worse performance for the maximum completion time of small flows in this scenario (Fig. 11(b)) in comparison to the previous scenario (Fig. 9(b)).

Fig. 11(a) shows that for large flows, the SB-SH scheme gives the largest mean delays. Recall that this traffic was generated with 85% flows from Exponential distribution of mean 20 packets. When the number of priority-enjoying flows increase, the flows that demands larger service time is more affected. Therefore, we see that, in such scenarios, the flows in the tail of the distribution experiences smallest mean delay under FCFS scheduling system. Though the increase in delay is small (without increasing the overall mean CT for large flows), this is an interesting observation, as none of the other metrics in the table reveal this.

7 Conclusions

In this paper, we proposed a new flow scheduler architecture to improve the performance of flows in the Internet. Through arguments and simulations we have emphasized the importance of each of the modules in the architecture. The architecture is shown to improve the performance of flows in comparison to the naive FCFS scheduler. Besides, in comparison to SIFT, the flow scheduler architecture brings in better performance in terms of conditional mean completion time and timeouts for small flows, and mean CTs (for small, large, and all flows). Apart from these, the worst-case delay performance is also appealing.

In general, our study confirms previous observation that size-based scheduling induces negligible degradation to large flows. While sampling is known to be a practical solution in tracking large flows, here we also see that it does not affect the performance of small flows, although the parameters (such as θ and p) were chosen arbitrarily.

This work opens different directions for future work. The parameters such as threshold θ, weight w and sampling probability p were kept constant here. Finding the right values for each of these so as to obtain the optimal delay performance is dependent on the other two parameters. All of them have an influence on the mean queue length. A larger value of θ will result in a larger number of packets sent to Q_1, a smaller value of w indicates a reduction in the service rate at Q_1, and decreasing the sampling probability will also increase the average number of packets of large flows served at Q_1. So, the variation in the average queue length (for a given load) can be used to decide the optimal values for these parameters.

References

[1] T. Bonald, S. Oueslati-Boulahia, and J. Roberts, "IP traffic and QoS control: the need for a flow-aware architecture," in *World Telecommunications Congress*, Sep. 2002.

[2] Leonard Kleinrock, *Queueing Systems, Volume II: Computer Applications*, Wiley Interscience, 1976.

[3] L. Schrage, "A proof of the optimality of the Shortest Remaining Processing Time Discipline.," *Operations Research*, , no. 16, pp. 687–690, 1968.

[4] S. F. Yashkov, "Processor-sharing queues: some progress in analysis," *Queueing Syst. Theory Appl.*, vol. 2, no. 1, pp. 1–17, 1987.

[5] Liang Guo and Liang Ibrahim Matta, "The war between mice and elephants," in *ICNP '01: Proceedings of the Ninth International Conference on Network Protocols*, 2001, p. 180.

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

[7] Konstantin Avrachenkov, Urtzi Ayesta, Patrick Brown, and Eeva Nyberg, "Differentiation Between Short and Long TCP Flows: Predictability of the Response Time," in *INFOCOM*, 2004.

[8] Idris A. Rai, Guillaume Urvoy-Keller, Mary K. Vernon, and Ernst W. Biersack, "Performance analysis of las-based scheduling disciplines in a packet switched network," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 106–117, 2004.

[9] Samuli Aalto, Urtzi Ayesta, and Eeva Nyberg-Oksanen, "Two-level processor-sharing scheduling disciplines: mean delay analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 97–105, 2004.

[10] Samuli Aalto and Urtzi Ayesta, "Mean delay analysis of multi level processor sharing disciplines," in *INFOCOM*, 2006.

[11] T. Zseby et al, "RFC 5475: Techniques for IP Packet Selection," http://www.rfc-editor.org/rfc/rfc5475.txt, Mar 2009, Network Working Group.

[12] K. Psounis, A. Ghosh, B. Prabhakar, and G. Wang., "SIFT: A simple algorithm for tracking elephant flows, and taking advantage of power laws," in *43rd Annual Allerton Conference on Control, Communication and Computing*, 2005.

[13] C. G. Chang and H. H. Tan, "Queueing analysis of explicit policy assignment push-out buffer sharing schemes for atm networks," in *Proceedings of the 13th IEEE Networking for Global Communications*, Jun 1994, vol. 2, pp. 500–509.

[14] Cristian Estan and George Varghese, "New directions in traffic measurement and accounting," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 323–336, 2002.

[15] Cristian Estan and George Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, 2003.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399