Reduced State Fair Queuing for Edge and Core Routers

Ramana Rao Kompella and George Varghese
University of California, San Diego
{ramana,varghese}@cs.ucsd.edu

ABSTRACT

Despite many years of research, fair queuing still faces a number of implementation challenges in high speed routers. In particular, in spite of proposals such as DiffServ, the state needs for even simple schedulers are still large for heavily channelized core routers and for edge routers. An earlier proposal, Stochastic Fair Queuing, reduces state but at the expense of added unfairness between certain flows. Another earlier scheme, Core Stateless Fair Queuing, requires header changes and does not address the state needs of edge routers. By contrast, our paper proposes a randomization technique that removes the need to store deficit counters per flow in Deficit Round Robin and its variants. Even without the counters, we show, using both analysis and simulation, that randomized technique preserves throughput fairness properties of DRR. This randomization technique introduced in this paper can be used to considerably reduce the state requirements of high speed schedulers in edge and core routers, making hardware designs feasible. The randomization idea in this paper can also be applied to other round robin schedulers as well as potentially in entirely different scenarios wherever deficits need to be tracked over time explicitly.

Categories and Subject Descriptors: C.2.6 [**Computer Communication Networks**]: Internetworking –*Routers;* F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Non-numerical Algorithms and Problems – Sequencing and Scheduling

General Terms: Algorithms, Performance

Keywords: Scheduling, Quality of service, Fair Queuing

1. INTRODUCTION

As introduced by Demers, Keshav and Shenker[1], fair queuing initially captured the imagination of researchers as a way to enforce fairness and provide traffic isolation required for applications such as Video-Conferencing, VoIP, etc. The argument for fair queuing became even more compelling when Parekh and Gallager[2] showed delay bounds for WFQ; later, similar delay bounds were shown for Virtual clock[3, 4]. Subsequent advances [5, 6, 7, 8, 3] improved the worst-case complexity of doing a scheduling decision in fair queuing from $O(N)$ to $O(log\ N)$.

However, $O(log\ N)$ complexity is still an impediment for high speed routers. Thus, simpler round-robin schemes like weighted round robin[9], deficit round robin (DRR) [10], and carry-over round robin [11] have been suggested for routers with link capacities greater than 1 Gbps. Such schemes allow $O(1)$ scheduling decisions and throughput fairness at the cost of providing coarser delay bounds. This appears to be an attractive trade-off for high speed router design. For example, a very popular core router, the Cisco GSR[12], implements a modified form of DRR called MDRR at 10 Gbps. MDRR is DRR with one extra queue that is always given high priority (for say VoIP). In fact, most modern routers provide some form of scheduling, limiting themselves to throughput fairness, priority, and token buckets.

Thus, for much of this paper we will concentrate on throughput fair schedulers. However, it is worth noting that a recent result shows that a set of DRR-like schedulers can be used to achieve reasonable delay bounds with small added complexity [13]. We will briefly consider such delay bounds in Section 5.

The Need to Minimize State: The previous paragraphs argue that simple schedulers like DRR can be implemented at high speeds, and can even be modified to provide delay bounds. However, most modern implementations of throughput-fair schedulers like DRR use *a limited number of flow queues*. Clearly, using one queue for every possible concurrent flow is impossible because the number of concurrent flows is in the order of millions [14], and this amount of high speed SRAM is infeasible. Thus, some form of flow aggregation is required. For instance, the Cisco GSR assigns a packet to one of 15 flow queues based on the IP TOS bits (even before the DiffServ proposal). DiffServ goes further and formalizes this assignment based on a class of aggregated behaviors called PHBs.

Unfortunately, even this solution does not appear to suffice as core routers scale up to higher speed (e.g., 40 Gbps) links. This is because such links are increasingly being *channelized*. For example, most routers that have 40 Gbps links allow the link to be channelized down to OC-3 speeds. Channelization is useful to allow ISPs to flexible provisioning for customers who would otherwise be unable to use such bandwidth. While each channel can be treated by a separate scheduler, separate scheduler state must be maintained for each channel to provide differential treatment to (say) customers on different channels.

If we multiply the number of channels per link, and the number of DiffServ[15] queues per channel, the number of flow queues (even after DiffServ aggregation) is still quite large. There are several modern routers that have about 64,000 queues per link and this number appears only to be increasing.

At a glance, one might conclude that 64,000 queues might not cause such an implementation issue given that hardware implementations are getting cheaper with improvements in technology. While this might be true for core routers now, it is unclear how even simple DRR based implementations can scale if we want to provide increasing differentiation between various services coupled with ever increasing number of applications that might heavily depend on the existence of fair queuing policies in the core routers. Given the high speeds of operation of these core routers, ways to reduce the amount of state (both in terms of update complexity and hardware resources) continue to be of high importance. Besides, excess hardware means excess power consumption. Power consumption is one of the other main issues faces by the core router

NOSSDAV'04, June 16–18, 2004, Cork, Ireland.

implementations. Therefore, getting rid of additional complexity in core routers as far as fair queuing goes is of tremendous importance even to reduce the power consumption, real estate costs and state update complexity of these routers.

So far, we have only argued that the scheduling state requirements of high speed *core* routers are increasing because of link channelization. However, for *edge* routers the situation is even worse. This is because the DiffServ buck stops, as it were, at the edge. Edge routers are routinely called on to perform large amount of "service differentiation" by classifying a large number of packet header types into DiffServ classes or MPLS tunnels. Note also that routers originally introduced as core routers are routinely conscripted into being edge routers a few years later when core speeds increase.

Thus, it is quite common in modern high speed router design (for either the edge or the core) that a single scheduler ASIC design struggles to maintain the scheduling state for a large number of queues using a bounded amount of on-chip SRAM. The state needs may often require the use of a more expensive ASIC (with larger amounts of on-chip SRAM) over an FPGA. Even ASIC design becomes challenging because large state requirements can require economies in other parts of the chip.

Related Work: The two best known approaches to reducing state for throughput-fair schedulers like DRR (besides DiffServ) are Stochastic Fair Queuing (SFQ) and Core Stateless Fair Queuing (CSFQ). SFQ [16] can aggregate traffic aggregates further using a hash function; however, this is often unsuitable because of the inherent unfairness caused by a bad flow (e.g., bad customer) on all good flows within its bucket[1]. Core stateless Fair Queuing[17] is an elegant scheme that completely avoids per-flow state and even per-flow queues without causing unfairness; it does so by essentially passing rate information placed by edge routers to core routers. Unfortunately, Core Stateless requires header changes that may be difficult to deploy. Hence, most of the current core routers still use DRR or a variant of DRR for fair queuing.

Further, neither Core Stateless nor DiffServ does anything about edge routers which must compensate (in both proposals) for the state reduction in the core by maintaining per-flow state. In summary, reducing state for even simple throughput-fair schedulers remains an important problem for both heavily channelized core routers as well as edge routers.

Paper Contributions: In this paper, we attack the problem of state reduction by designing a throughput-fair scheduler that is extremely parsimonious in its use of state, reducing by an order of magnitude the amount of state required by comparable throughput-fair schedulers. Unlike Stochastic Fair Queuing, the algorithm provides good traffic isolation; unlike Core Stateless Fair Queuing, our algorithm does not require global changes, requiring only uncoordinated local implementation changes in each router.

We use Deficit Round Robin (DRR) as our point of departure. We observe that the state needs of DRR are threefold: first, the space for the queue head pointers for the flow queues together with the quantum size for each queue; second, the space required for the deficit counters; and third, the space required for the list of active queues. In the standard implementation of DRR, all three storage overheads are $O(N)$, where N is the number of queues: this can be accounted for by N queues, N deficit counters, and a $O(N)$ sized active list. By contrast, in this paper, we show how to reduce the space for the deficit counter to zero using a simple randomized algorithm.

There are a lot of variants of DRR such as Elastic Round Robin[18], Pre-order Deficit Round Robin[19], Nested Round Robin[20], etc., that have been proposed in the literature to address some of the deficiencies of DRR. However, these algorithms continue to use a deficit counter to keep track of the deficits in a single round of DRR. While we present our randomized technique in the context of reducing state requirement of all these DRR based algorithms, it can be applicable in any scenario that requires one to keep track of deficits over multiple rounds over time.

The reader may object that we have not completely solved the problem because of the space that remains for queue heads. We observe that in many practical implementations, pointers to queue heads are stored along with the filter rules of a lookup engine (next hop entries for core routers, packet classification filters for edge routers). Such lookup tables are already quite large (e.g., 20 bytes or larger) due to the presence of say multiple hops for load balancing purposes, or statistics counters for monitoring purposes. Thus, there is often room in this next hop table for quantum sizes and head pointers. Secondly, packet buffers are often stored in DRAM as opposed to SRAM; queue heads can be stored along with these buffers.

Thus, reducing the storage needs and processing overhead of the basic scheduling algorithm (as we do in this paper, by an order of magnitude) is still of significant practical interest. Secondly, we believe that the algorithmic technique, applying randomization to scheduling without losing fairness (as in SFQ) is of independent theoretical interest. Thirdly, we also believe, that the techniques suggested in these papers could be applied in other contexts where a variant of Deficit Round Robin Scheduling is used.

The rest of this paper is organized as follows. In Section 2, we introduce randomized DRR. In Section 3 we analytically compare the fairness of randomized DRR with conventional DRR. In Section 4, we show using standard ns-2 based simulations that the behavior of randomized DRR is indistinguishable from DRR in end-to-end settings. In Section 5, we briefly discuss delay bounds and finally conclude the paper in Section 6.

2. RANDOMIZED DEFICIT ROUND ROBIN

The standard DRR scheduler [10] assigns a quantum to each queue in proportion to the service rate guaranteed for that queue. In other words, if a flow is assigned a quantum Q, a flow that has been assigned twice the bandwidth is assigned a quantum of $2Q$. In each round of scheduling, if a queue was unable to send a packet because its packet size was too large, the remainder from the previous quantum is added to the quantum for the next round. Thus, deficits are kept track off; queues that were shortchanged in a round are compensated in the next round.

As can be noted from the above, *the standard DRR scheme needs per flow deficit counters that keep track of the credits left over in the previous round of scheduling.* If the algorithm is to be implemented in high speed scheduler chips servicing say a million queues, the chip would require about 16 Mbits just for storing these per flow counters, assuming a 16-bit counter per queue. It may also be difficult to store such a large number of counters on chip, forcing the design to maintain counters off-chip, thereby increasing contention for already precious memory bandwidth. Instead, in this section, we propose a *randomized deficit round robin* scheduler that keeps no per flow state for deficit counters. This is useful for schedulers that service a large number of queues.

Recall that standard deficit round robin tries to send as many packets as possible subject to the proviso that the total number of bytes serviced is less than or equal to the quantum Q (plus any leftover) allocated to this particular flow. After sending as many packets as possible without exceeding its allocation for the round,

[1]The unfairness could potentially be distributed across various buckets by changing the hash function periodically

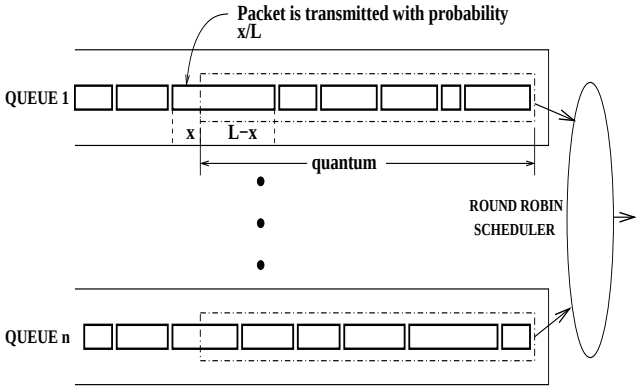

Figure 1: Randomized Deficit Round Robin

the left over deficit is kept track off and is added to the quantum during the next scheduling interval.

Our randomized DRR finesses the need for maintaining such deficit counters at the cost of providing *statistical* guarantees. It does so by observing that there is only one packet that can straddle the boundary between staying within and exceeding the quantum. By deciding to transmit this boundary packet with probability proportional to the number of credits that would have been left over (in a conventional DRR scheduler), the algorithm ensures that the *expected* deficit accrued is zero without maintaining any state.

Thus, the boundary packet is sometimes sent (going over quantum) and sometimes not sent (staying under quantum), but the proportion of these two outcomes is adjusted to make the net gain (or loss) equal to zero. Notice also that the reason we do not need state is because the decision in randomized DRR is only based on the quantum value and the packet lengths in the current queue being considered.

Formally, let $p_1^i, p_2^i, \ldots, p_m^i$ denote packets with lengths l_1^i, l_2^i, $\ldots$, l_m^i for a backlogged queue B_i. Let the quantum allocated to B_i be Q_i according to the queue's allocation. Then the algorithm proceeds as follows

1. The scheduler services queues in a round robin fashion and in each service interval, it schedules packets $p_1^i, p_2^i, \ldots, p_j^i$ that belong to Q_i such that
$$\sum_{k=1}^{j} l_k^i \leq Q_i < \sum_{k=1}^{j+1} l_k^i$$
.

2. Denote $D^i = Q_i - \sum_{k=1}^{j} l_k^i$ as the remaining credits left. Then, the scheduler transmits packet p_{j+1} with probability p_i and does not transmit the packet with probability $1 - p_i$, where p_i is given by $p_i = \frac{D^i}{l_{j+1}^i}$.

3. If the queue is not active, that is it has no packets to send, then the scheduler moves to the next queue and tries to service packets from that queue. If the queue has packets to send, but the queue is not backlogged, then the scheduler transmits all the packets for that particular queue.

In summary, randomized DRR is exactly equivalent to DRR except for the last step: in DRR, the deficits are explicitly stored in a per flow counter, whereas in randomized DRR, the deficits are taken care off stochastically with no per-flow counter. For analytical tractability, we discuss the average case, worst case, variance and other statistical properties of the randomized algorithm in the next section.

3. ANALYSIS OF RANDOMIZED DRR

In this section, we analyze the behavior of Randomized DRR described in the previous section. First, we describe the average case behavior of the algorithm, followed by an analysis of the variance. Then, we compute worst case bound on the performance of Randomized DRR.

3.1 Average Case Analysis

Theorem 1 *The expected total number of bytes sent by the scheduler over n rounds, $Exp[S]$ for a backlogged queue with Quantum Q is*

$$Exp[S] = Q \cdot n$$

Proof: Denote X_i as the random variable that counts the number of bytes serviced in a round from a particular backlogged queue with quantum Q. Let the packets, $p_1, p_2, \ldots, p_m$ be packets such that, $\sum_{k=1}^{m} l_k < Q$. If the sum is equal to the quantum, then the number of bytes transmitted would be exactly equal to the quantum. For the case, when it is less than the quantum, there is enough credits to send only part of the next packet with length p_{m+1}. Hence, the expected value of X_i is given by the following equation

$$Pr[p_{m+1} \text{ is transmitted}] = \frac{(Q - \sum_{k=1}^{m} l_k)}{l_{m+1}}$$

The expectation of X_i is given by,

$$\begin{aligned} Exp[X_i] &= \sum_{k=1}^{m} l_k + Pr[p_{m+1} \text{ is transmitted}] \cdot l_{m+1} \\ &= \sum_{k=1}^{m} l_k + (Q - \sum_{k=1}^{m} l_k) \quad = Q \end{aligned}$$

Now, $Exp[S] = Exp[\sum_{k=1}^{n} X_k]$. By linearity of expectations, $Exp[S] = \sum_{k=1}^{n} Exp[X_k] = n \cdot Q$.

In summary, the average number of bytes serviced per queue is in proportion to the weight of each queue.

3.2 Computing Variance

Note that the average-case analysis above holds for any packet size distribution. However, statistically speaking, users are not just interested in the average but also in the standard deviation (or variance). Intuitively, a higher variance implies the possibility of worse short-term fairness for a queue. In this section, we analyze the variance of the amount of service received by each queue.

To calculate the variance of this algorithm from the expected service over n rounds of scheduling, we first calculate the variance for one round of scheduling. Let $\sigma^2_{X_i}$ denote the variance of X_i during one round of scheduling. Also, let $S = \sum_{k=1}^{m} l_k$, where $p_1, p_2, \ldots, p_m$ denote set of packets that add up less than the allocated quantum size Q_i and $S + l_{m+1} \geq Q$. Usually, S can vary depending on the packet size but for now, let us consider the conditional expectation that S and l_{m+1} are given.

$$\begin{aligned} Exp[X_i^2|S, l_{m+1}] &= (S + l_{m+1})^2 \cdot Pr[p_{m+1} \text{ transmitted}|S, l_{m+1}] \\ &\quad + S^2 \cdot Pr[p_{m+1} \text{ not transmitted}|S, l_{m+1}] \end{aligned}$$

$$
\begin{aligned}
\sigma^2_{X_i} &= Exp[X_i^2] - \mu^2 \\
&= \sum_S \sum_{l_{m+1}} Exp[X_i^2 | S, l_{m+1}] - \mu^2 \\
&= \sum_S \sum_{l_{m+1}} \{(S + l_{m+1})^2 \cdot \frac{(Q-S)}{l_{m+1}} \\
&\quad + S^2 \cdot (1 - \frac{Q-S}{l_{m+1}}) - Q^2\} \cdot Pr[S] \cdot Pr[l_{m+1}] \\
&= \sum_S \sum_{l_{m+1}} (l_{m+1} - (Q-S))(Q-S) \cdot Pr[S] Pr[l_{m+1}] \\
&< \sum_S \sum_{l_{m+1}} (\frac{l_{m+1}^2}{4}) \cdot Pr[S] Pr[l_{m+1}] \\
&< \sum_S \sum_{l_{m+1}} (\frac{P_{max}^2}{4}) \cdot Pr[S] Pr[l_{m+1}] \\
&= \frac{P_{max}^2}{4}
\end{aligned}
$$

Note that the first inequality above follows since, the variance $\sigma^2_{X_i}$ is maximized when $(Q - S) = l_{m+1}$. As can be seen from the above equation, Variance $\sigma^2_{X_i}$ is dependent on the packet size of the boundary packet that crosses the allocated quantum but can be bounded using the maximum packet size.

In order to compute the variance over multiple rounds of scheduling, the random variables X_i need to be independent of each other. This also requires knowing the packet size distribution. Note that, we did not make any assumptions on the packet sizes when we calculated the mean. We need packet size distribution only for the calculation of the variance over multiple rounds of scheduling, and later when we develop high confidence Chernoff bounds.

Let us consider a simple case when packets are of constant size[2], P. If all the packet sizes are of constant size, as in the case of ATM cells, then, each X_i is independent of each other. This is because, irrespective of whether the packet crossing the quantum has been transmitted or not in the previous round, since the packet sizes are the same, in each round, the scheduler sees only a list of constant size packets. The variance in this case for single round of scheduling would reduce to $\sigma^2_{X_i} = \frac{P^2}{4}$ The variance over n rounds of scheduling, σ^2_X where $X = \sum_{k=1}^{n} X_k$ would be

$$\sigma^2_X = \sum_{k=1}^{n} \sigma^2_{X_k} = \frac{P^2}{4} \cdot n$$

For example, $n = 100$ and $P_{max} = 1500$bytes, then the standard deviation in the amount of service received by any queue from the mean is 3750 bytes which is extremely small. In summary, the standard deviation over n rounds for these simple distributions, scales roughly as $O(\sqrt{(}n))$. While this is reasonably small, we can obtain tail probability bounds on the discrepancy using Chernoff bounds as we show next.

3.3 Chernoff Bound

Under the assumption that the random variables X_i for different rounds of scheduling are independent of each other, we can apply the Chernoff Bound to bound the probability that the random variable deviates from the mean.

We show the analysis only for the case when packets are of same size. We model the random variables Y_i as a 0,1 random variable with $Pr[Y_i = 1] = p_i$. Each variable Y_i assumes a 0, if the last packet is not transmitted in the round i and 1, if the last packet that crosses the quantum is transmitted. The probability $p_i = \frac{(Q \bmod P)}{P}$ remains the same for all the rounds of scheduling since the packets are of same size, P. If $Y = \sum_{i=1}^{n} Y_i$, then,

$$\mu_Y = Exp[Y] = \sum_{k=1}^{n} p_i = n \cdot \frac{(Q \bmod P)}{P}$$

Also, X, the total number of bytes over n rounds of scheduling, is given by,

$$
\begin{aligned}
X &= \sum_{i=1}^{n} (Q - (Q \bmod P) + P \cdot Y_i) \\
&= n(Q - (Q \bmod P)) + P \cdot Y \text{ and,} \\
Exp[X] &= \mu_X = n \cdot Q
\end{aligned}
$$

Applying Chernoff Bounds on the random variable Y, we get,

$$
\begin{aligned}
Pr[Y > (1+\delta)\mu_Y] &< e^{-\mu_Y \delta^2/3} \\
Pr[Y < (1-\delta)\mu_Y] &< e^{-\mu_Y \delta^2/2}
\end{aligned}
$$

Substituting these probabilities in the expression for X and after further simplification, we obtain the following bounds for X.

$$
\begin{aligned}
Pr[X > \mu_X + n \cdot (Q \bmod P) \cdot \delta)] &< e^{-\mu_Y \delta^2/3} \\
Pr[X < \mu_X - n \cdot (Q \bmod P) \cdot \delta)] &< e^{-\mu_Y \delta^2/2}
\end{aligned}
$$

As an example, consider a queue with Quantum, $Q = 10000$ bytes, $\delta = 0.3$, $n = 100$ rounds of scheduling, and the packet size, $P = 1500$ bytes, then, $Q \bmod P = 1000$, $Pr[(X - \mu_X) > 30000] < 0.13$ and $Pr[(X - \mu_X) < -30000] < 0.05$. Therefore, from these Chernoff bounds it is clear that the probability that the amount of service that any queue receives is within a factor δ from the mean of the random variable Y is exponential in n, the number of rounds of scheduling. So, as the queue gets serviced many times, with high probability the service that a queue receives is very close to the mean (fair share).

3.4 Bounding Worst-Case Discrepancy

So far we have analyzed the expected behavior of the algorithm under standard stochastic assumptions. It is also instructive to bound the worst-case fairness discrepancy caused assuming that the random number generator is not just bad, but also adversarial in that it picks the worst possible outcome at each point. Theorem 2 shows this worst-case discrepancy for the randomized algorithm. For brevity, we skip the proof of this Theorem.

Theorem 2 *If the total number of bytes serviced from any backlogged queue is S over n scheduling intervals and the quantum allocated for the queue is Q,*

$$|S - (Q \cdot n)| < P_{max} \cdot n$$

Hence, in the worst case even if the random number generator is completely misbehaving, the loss in bandwidth for any backlogged queue is less than one packet for each round of scheduling. Intuitively, this is because a discrepancy from normal DRR can only arise for the boundary packet; without using deficit counters and with a broken random number generator, the algorithm can persistently fail to transmit the boundary packet. The worst case discrepancy for a queue of Quantum 10000, after 100 rounds of scheduling, would be 150000 bytes. However, as we have shown in the Chernoff bounds, the probability of discrepancy greater than 30000 bytes is less than 0.13 for the randomized deficit round robin.

[2]Other distributions such as uniform, bimodal are more complicated and we do not deal with them in this paper.

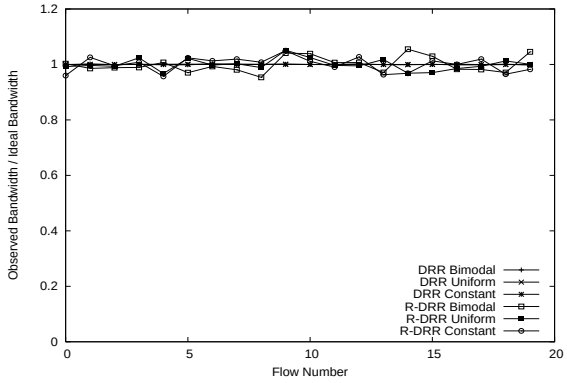

Figure 2: Comparison of DRR and Randomized DRR with constant packet sizes, Uniformly distributed packet sizes, and Bimodal distribution of packet sizes

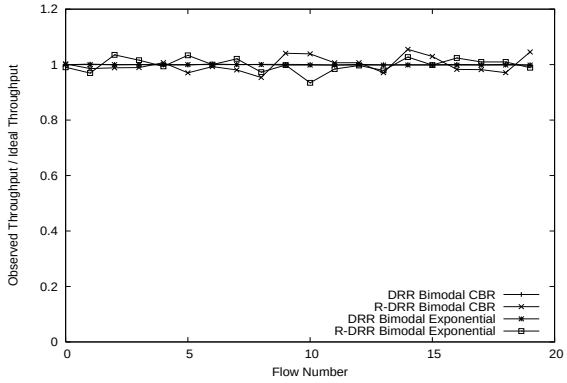

Figure 3: Comparison of DRR and Randomized DRR with Constant and Exponential Traffic Distribution with bimodal packet size distribution

4. SIMULATION RESULTS

In this section, we describe simulation results for Randomized DRR. First, we compare it with DRR to show that randomized DRR has throughput fairness that is similar to DRR. Later, we compare Randomized DRR with CSFQ and DRR for the topologies described in the CSFQ paper[17]. All simulations were implemented on the ns-2 simulator.

Comparison with DRR: For the purposes of comparison with DRR, our simulation topology consists of a 1Mbps link shared by 20 UDP flows. Each of these flows transmits packets at twice their fair share of bandwidth. To show that Randomized DRR is fair across different packet size distributions, we simulated three different distributions; constant packet size of 1000 bytes, uniform packet distribution between 1 and 1000 bytes, and bimodal distribution of 100 and 1000 bytes in Figure 2. The y-axis in the Figure 2 is the ratio between the observed bandwidth and the fair share of bandwidth and the x-axis represents the flow number. As can be observed from Figure 2, almost all flows get close to their fair share for both DRR as well as Randomized DRR.

To show that Randomized DRR is fair across different traffic distributions, we simulated an exponential traffic distribution and a Constant Bit Rate distribution in Figure 3. Again, for both the traffic patterns, the ratio of the observed bandwidth to the fair share bandwidth is plotted in the y-axis. Figure 3 shows that irrespective of the traffic pattern, Randomized DRR provides good throughput fairness.

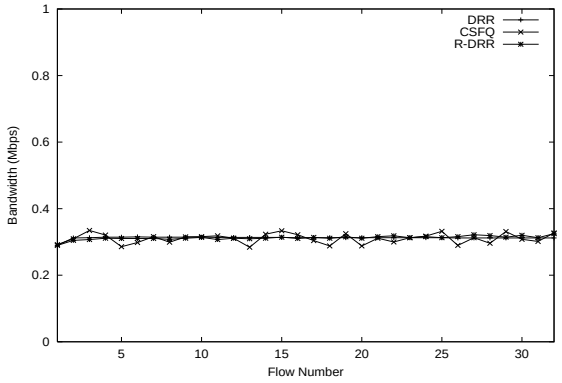

Figure 4: Shared Link with all UDP sessions

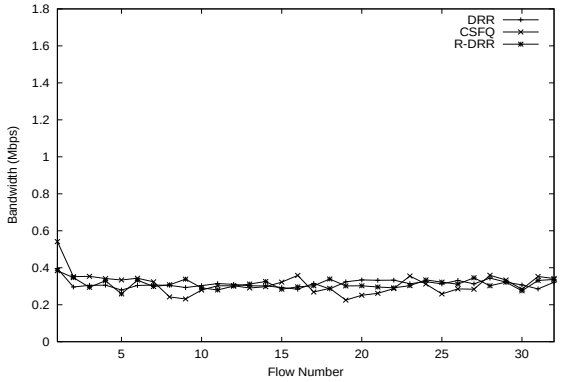

Figure 5: Shared Link with one misbehaving UDP session and 32 other TCP sessions

Comparison between CSFQ, DRR and Randomized DRR: To show that Randomized DRR compares well with regular DRR as well as the well-known CSFQ scheme, we carry out three different experiments on a single congested link of capacity 10Mbps. We chose the buffer size so that the throughput of both DRR and Randomized DRR are not penalized due to lack of buffer space. The averaging constants for CSFQ remain the same as in [17], namely $K = K_\alpha = K_c = 100ms$. Finally, in all topologies the first router on the path of each flow is assumed to be an edge router; all other routers being core routers.

In the first experiment, we have 32 UDP flows, indexed from 0, where flow i sends $i + 1$ times more than its fair share. Figure 4 shows that Randomized DRR performs almost the same as CSFQ as well as regular DRR. In the second experiment, we simulated an ill-behaved UDP flow that sends at 10Mbps along with 31 TCP flows. The x-axis represent the flow number (UDP flow is with index 0). The y-axis represents the observed throughput of the various flows (Expected Fair Share = 0.3125Mbps). Figure 5 shows that Randomized DRR performs as good as DRR in protecting the TCP flows against the ill-behaved UDP flow.

Finally, in the third experiment, we investigate how a single TCP flow performs in the presence of other misbehaving UDP flows that send twice their fair share of traffic. Figure 6 shows that both DRR and Randomized DRR perform again perform well, followed by CSFQ. In conclusion, from the various experiments, it is clear that randomized DRR performs well in providing throughput fairness even with reduced state.

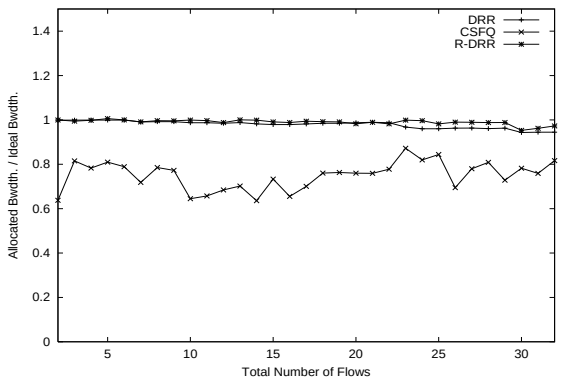

Figure 6: Shared Link with one TCP session and 32 other misbehaving UDP sessions

5. DELAY BOUNDS

One important property of Deficit Round Robin is that, if all queues have equal bandwidth reservations, the quantum values allocated to various queues would be close to each other. In particular, suppose, all queues are assigned rates such that the maximum rate is at most twice as big as the minimum rate, then the maximum quantum Q_{max} would be at most twice as big as the minimum quantum Q_{min}. This means that any queue would suffer a delay bound of at most $L_{max} \cdot 2N$, where L_{max} is the size of the maximum packet and N is the number of queues.

However, if the variation in the rates is high, then the delay bound becomes really large. This is because, bigger flows are awarded huge quantum values causing a huge delay for the flows with smaller quantum values. In order to alleviate this, we can group all the queues that have bandwidths reserved between 2^j and $2^j + 1$, for some j. Then, we can perform time-stamp based scheduling over these groups and DRR inside these groups, to provide better delay bounds.

The above observation has been used by Ramabhadran and Pasquale in [13] to provide a low bandwidth scheduler called Stratified Round Robin with reasonable delay bounds. The randomization technique in our paper can be used even in Stratified Round Robin to get rid of per-flow state. Also, the groups can be statically scheduled using a scheme like Smoothed Round Robin that has been developed in [21].

In summary, Deficit Round Robin combined with either Smoothed Round Robin or Stratified Round Robin can be used to provide reasonable delay bounds. The algorithms in our paper can thus be applied to DRR to provide such delay bounds with minimal storage.

6. CONCLUSIONS

Round-robin schedulers are commonly used in router architectures due to their extremely low complexity. Many practical core router architectures often employ some form of Deficit Round Robin scheduler for throughput fairness without paying too much attention to delay bounds. However, simplest DRR-based schedulers use per-flow state in terms of a deficit counter that keeps track of the deficits in various rounds of scheduling. Our paper proposes the use of randomization to eliminate this counter requirement, so as to minimize overall scheduler state.

We note that stateless fair queuing is a hard problem at high speeds. Core Stateless Fair Queuing (and its variants) is an elegant attempt to shift the problem in space, moving it from the core to the edge. Unfortunately, it requires global changes making it difficult to deploy. By contrast, our paper is an attempt towards reducing (but not completely eliminating) the state required for fair queuing at the core and the edge using randomization.

Although our schemes provide considerable improvement for practical routers being built today, there are still other sources of state overhead for such schedulers, such as queue heads and quantum values. While we have argued that these can be stored along with lookup tables, it would be desirable to find techniques to reduce even this overhead. Doing so would enable the vision of nearly stateless fair queuing at the core *and* the edge, without the difficulties associated with global protocol changes.

Acknowledgements

This paper benefitted greatly from discussions with Sriram Ramabhadran and Cristian Estan. We would also like to thank the anonymous reviewers for their insightful comments on this paper. This work was made possible by the NIST Grant 60NANB1D0118 towards the Sensilla project.

7. REFERENCES

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Symposium proceedings on Communications architectures & protocols*. 1989, pp. 1–12, ACM Press.

[2] A. Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Network"*, Ph.D. thesis, Masschusetts Institute of Technology, Feb. 1992.

[3] L. Zhang, "Virtualclock: a new traffi c control algorithm for packet switching networks," *ACM Transactions on Computer Systems*, vol. 9, pp. 101–124, May 1991.

[4] Norival R. Figueira and Joseph Pasquale, "An upper bound on delay for the VirtualClock service discipline," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 399–408, 1995.

[5] Dimitrios Stiliadis and Anujan Varma, "Rate-proportional servers: a design methodology for fair queueing algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 164–174, 1998.

[6] Dimitrios Stiliadis and Anujan Varma, "Effi cient fair queueing algorithms for packet-switched networks," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 175–185, 1998.

[7] Jon C. R. Bennett and Hui Zhang, "WF 2 q: Worst-case fair weighted fair queueing," in *INFOCOM (1)*, 1996, pp. 120–128.

[8] Pawan Goyal, Harrick M. Vin, and Haichen Cheng, "Start-time fair queuing: A scheduling algorithm for integrated services packet switching networks," in *IEEE/ACM Transactions on Networking*, 1997, vol. 5.

[9] J. Nagle, "On packet siwtches with infi nite storage," in *IEEE Transactions on Communications*, Apr. 1987, pp. 35(4):435–438.

[10] M. Shreedhar and George Varghese, "Effi cient fair queueing using defi cit round robin," in *ACM SIGCOMM*, 1995, pp. 231–242.

[11] Debanjan Saha, Sarit Mukherjee, and Satish K. Tripathi, "Carry-over round robin: a simple cell scheduling mechanism for atm networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 6, no. 6, pp. 779–796, 1998.

[12] Cisco Systems, Inc., "GSR 1200 Router Series," `http://www.cisco.com/warp/public/cc/pd/rt/12000/prodlit/gsrlc_ov.htm`.

[13] Sriram Ramabhadran and Joe Pasquale, "A low complexity packet scheduler with bandwidth fairness and bounded delay," in *ACM SIGCOMM*, Aug. 2003.

[14] K. Thompson, G. Miller, and R. Wilder, "Wide-area traffi c patterns and characterizations," in *IEEE Network*, December 1997.

[15] S. Blake et.al, "An architecture for differentiated services," *RFC 2475*, Dec. 1998.

[16] P. McKenney, "Stochastic fairness queuing," in *IEEE INFOCOM*, June 1990.

[17] Ion Stoica, Scott Shenker, and Hui Zhang, "Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks," in *SIGCOMM*, 1998, pp. 118–130.

[18] Salil Kanhere and Harish Sethu, "Low-latency guaranteed-rate scheduling using elastic round robin," in *Computer Networks 25 (14)*, 2001, pp. 1315–1322.

[19] S. Tsao and Y. Lin, "Pre-order defi cit round robin: a new scheduling algorithm for packet-switched networks," in *Computer Networks 35 (2-3)*, 2001, pp. 287–305.

[20] Salil Kanhere and Harish Sethu, "Fair, effi cient and low-latency packet scheduling using nested defi cit round robin," in *IEEE Transactions on Parallel and Distributed Systems*, 2002, pp. 324–326.

[21] Chuanxiong Guo, "SRR: An 0(1) time complexity packet scheduler for flows in multi-service packet networks," in *ACM SIGCOMM*, Aug. 2001.