Burst Shaping Queueing

Vasilios Darlagiannis, Martin Karsten, Ralf Steinmetz

KOM, Darmstadt University of Technology, Germany
{Vasilios.Darlagiannis, Martin.Karsten, Ralf.Steinmetz}@KOM.tu-darmstadt.de

Abstract: In this paper, we present a novel queue management mechanism, called Burst Shaping Queueing (BSQ). The main objective is to minimize the burstiness of the traffic on packet switched routers, by interleaving packets that are going to follow different links on next hops. The complexity of that algorithm is O(1), which makes it deployable in high speed networks.
From the simulated experiments we performed, it is proven that BSQ offers a better trade-off between link utilization and end-to-end delay, compared with other queueing mechanisms. Combined with RED, it can improve the link utilization, without any significant increase on the end-to-end delay.
Keywords: burstiness, network queues, shaping, scheduling, load balance

1 Introduction

In packet-based switched networks, congestion occurs basically because of the interference of not synchronized, independent flows, sharing common resources, in cases where the demand for those resources is higher than the offered ones.
This is a very complex problem that requires coordinated efforts in different components of the network, as well its design. Network designers provision to offer sufficient resources to often congested links. But, it has been experienced that even in over-provisioned networks, it is a matter of time to become congested by always more demanding users and applications.

Therefore, appropriate mechanisms are used both in the end-points and the routers of a network. End-points react to the congestion signalling they receive (e.g. in TCP flows, the senders interpret the acknowledges they receive to increase or decrease their transmission rate appropriately). Additionally, packet pacing has been introduced to minimize the burstiness of the flows, by evenly distributing the departure of the packets of a window-based flow, over the estimated RTT.

In routers, different queue management techniques are employed, aiming to augment the congestion avoidance, by trying to detect congestion in advance, in order to inform end-points to reduce their transmission rate, or by offering more fair shares to the flows sharing a particular link and punishing this way the more aggressive flows.

The most common queue discipline in the Internet today is FIFO[1] (First-In-First-Out), although it is reported in many works [1], [2] that FIFO is responsible for higher end-to-end delay because of full queues, unfair share of the available bandwidth and lock out phenomena, where few flows monopolize the queue space. Many researchers have proposed alternative schemes with better performance characteristics, but either because of their moderate improvement, or because of their suitability in particular scenarios only, or even further because of high complexity/scalability issues, they still have not been widely deployed.

Some of the proposed schemes, like Random Early Detection (RED) [5] or Blue [3], operate on a single queue, where the main operation is either dropping probabilistically a packet or marking it, based on the average queue length. Other approaches construct multiple queues, either to discriminate the offered service, or to share the bandwidth more fairly. In the case of the multiple queues, scheduling mechanisms are employed to select the next sub-queue to be served. To a rough approximation [2], in order to discriminate the various algorithms taking place at routers to handle incoming packets:

- queue management algorithms manage the length of packet queues by dropping packets when necessary or appropriate, while
- scheduling algorithms determine which packet to send next; they are used primarily to manage the allocation of bandwidth among flows.

Moreover, traffic engineering techniques are supplementary solutions, used to balance evenly the traffic over alternative paths between end-points.

In our work, we examine the potential to enhance the congestion avoidance probability, by changing the scheduling of the packets on an outgoing link, based on the information “which are the links that those packets are going to follow, in successive routers”. Packets that are going to follow different paths in following routers are interleaved, avoiding, as much as possible, the sequential submission of packets that are going to follow the same links. The order of the packets that are going to follow the same links is not affected. This way, the burst-

1. Usually the terms FCFS (First-Come-First-Served) or Drop-Tail are used alternatively.

iness in that individual outgoing link remains the same, but the burstiness in following ones is reduced. This new queueing mechanism is called Burst Shaping Queueing (BSQ).

The rest of this paper is organized as follows: Section 2 describes briefly work that is alternative or supplementary to our research, while Section 3 provides the motivation for using BSQ. Section 4 describes the algorithms of BSQ and their complexity and some experiments are provided in Section 4 for its evaluation. Finally, Section 6 provides the conclusion of our work, as well as some thoughts for further investigation.

2 Related Work

RED [5] is possibly the most explored queue management mechanism. Alternatives to the original algorithm are Flow Random Early Drop (FRED), RED with penalty box, Stabilized RED [14], Gentle RED (GRED), Adaptive RED [4] and Self-configured RED. In the original algorithm, packets are served in a FIFO order, but the buffer management is significantly more sophisticated than Drop-Tail. RED starts to drop packets probabilistically, based on the average queue length. RED trades-off lower utilization for lower end-to-end delay. Four parameters are used to configure RED, to make it appropriate for different scenarios. Fine tuning of those parameters is always a headache for the researchers. The latest version of RED, Adaptive RED overpass this problem, by using some sophisticated algorithms.

Deficit Round Robin (DRR) [16] is a scheduling algorithm, aiming to offer fair shares of the bandwidth to all of the flows competing in an outgoing link. DRR construct a number of multiple queues (called buckets) and then assigns each flow in one of those buckets, by hashing the destination IP address. More than one flows could share the same bucket, in case of collisions in the hashing procedure The scheduler periodically visits all the buckets using the round robin algorithm, and adding a quantum (an increase on the maximum size in bytes allowed to be transmitted from that bucket) to the deficit of that bucket. If the size of the next packet to be transmitted is smaller than the deficit, then it is transmitted. Otherwise the scheduler proceeds to the next bucket.

Core-Stateless Fair Queueing (CSFQ) [17] is an approach that discriminates the operations performed by the core routers from the operations performed by the edge routers in a subnet. Since the number of flows in the core routers is very large and the transmission speed is very high, it is not possible to apply a queueing technique, requesting for state per flow. Deployability of a statefull technique is more feasible in edge routers. CSFQ calculates the rate of each flow in the edge routers and this rate information is inserted in every packet. Core routers calculates the fair share of each flow and probabilistically drop packets from flows that exceed their fair share. The drawbacks of that approach are mainly the facts that each packet has to be labeled/de-labeled in the edge routers and the complexity of the edge routers still remains high. Mice flows might be handled unfriendly, since the estimation of their rate can be miscalculated due to their short live.

3 Motivation

In order to motivate our work, we take as an example the network shown in Figure 1. In that example, we consider a snapshot where some flows are directed from nodes 1 and 2, through 3 to nodes 4, 5 and 6. The dotted arrows represent a set of flows that are going to be directed to node 4, the dashed arrows to node 5 and the solid ones to node 6. Although it is possible that none of the flows is highly bursty, it can very likely be the aggregation of them. Node 3 can not regulate the burstiness of its outgoing links, unless it under-utilize them. But, possibly nodes 1 and 2 can augment node 3 in that.

Consider that at an arbitrary point in time, the content of the queue of the outgoing link on node 1, is the one shown on the left side of Figure 2. Using the usual FIFO discipline, the order of the transmission of the packets is identical to their order in the queue, as it is shown in the upper right part of the figure. Here we notice that the packets are mostly transmitted in groups of 3, meaning that after node 3, that group is going to follow the same outgoing link. On the other hand, employing BSQ at node 1, the output order is the one shown in the lower right part of the same figure.

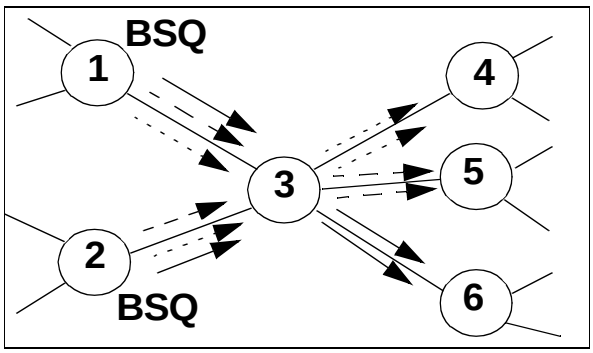

Figure 1: Network example

We can notice that using BSQ we have bursts of one packet, with the exception of the end of the queue, where 4 packets are creating an even higher burst than FIFO. This is the case because in that static figure, we can not capture the real-time, dynamic characteristics of a traffic passing through the queue, where packets are always

coming into, allowing further interleaving of them, without causing such large bursts.

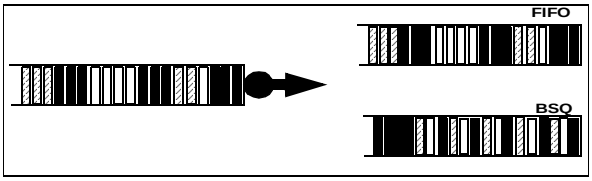

Figure 2: Queue content snapshot

It is important to mention that although the output of the link on node 1 should not cause any congestion on the output links of node 3 towards nodes 4, 5 and 6 (if we assume identical link characteristics), it is very possible to be the case by interfering with the output link of node 2 towards node 3(or even with more links that carry packets that will be routed through the aforementioned links).

From that case description, it is observable why BSQ can offer better performance, which is achieved by lower average queue lengths, by controlling the scheduling of the packets.

4 Algorithm Description

4.1 Assumptions

In order for a router to be able to classify a packet based on its destination IP address, it is assumed that it includes the required information in its own routing table. The amount of that required information depends on the depth into the network every router is interested in (level of interest). For example, for depth value equal to one, the router is assumed to include in its own routing table the routing tables of all of each neighbours. The specific way of implementing it is out of scope, but alternatives can be either a two step lookup, or even in one step, where tuples are returned.

Additionally, link characteristics like bandwidth can be required for a weighted and more efficient scheduling.

4.2 Design Issues

Every router assigns the neighbour router on each specific outgoing link in the root of a tree with depth equal to the level of interest. Figure 3 presents the structure of that tree for node 1 of Figure 1, and more specifically, for the outgoing link towards node 3, with level of interest 1. Based on the topology of that network, root Q3 has three children, Q4, Q5, Q6, representing the sub-queues on that router. Each sub-queue is responsible to handle the packets that are going to follow the respective outgoing link on the next router. Global information is required as well, to manage the total buffer space.

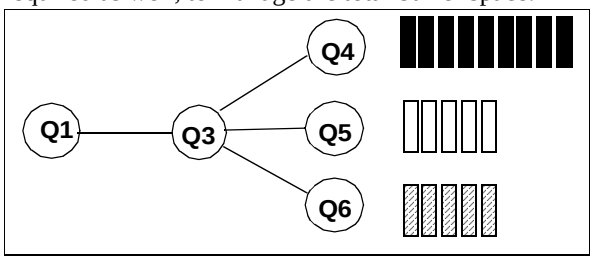

Figure 3: Tree structure on node 1 for the outgoing link towards node 3

There are two main operations that take place in a queue system: Adding a packet (enqueueing) and removing a packet (dequeueing).

When a new packet arrives in the queue, it is processed iteratively to find the correct position into the tree of the sub-queues. A new sub-queue is created when the requested branch does not exist, taking as parameter the bandwidth of the link, to ensure weighted scheduling. When the appropriate leaf of the tree is reached, the packet is stored in the corresponding sub-queue. If the total number of storage space required to store the packets of the queue is greater than the maximum allowed one, BSQ drops the head of largest sub-queue[1].

```
void enque(packet) {
        Get the address of this node
        Current node is the root of the tree
        Search until the maximum levelOfInterest is
        reached or the final destination is found {
                Lookup in the extended routing table
                which is the next hop from the
                current one to the
                final destination
                Get the child in the subtree that fits with
                that next hop
                If that child does not exist, create it
                Make the current node this new one
        }
        Increase the number of packets that are stored
        in this subtree
        Enqueue the packet in that subtree
        Accumulate the total size of the stored packets
        If there is buffer overflow, drop a packet from the
        subtree with the largest number of  packets
        (or the maximum length of them){
                Find the subtree with the maximum
                number of packets
                Update the internal structures
                Dequeue the packet to be dropped
                Drop the packet
        }
}
```

Figure 4: Enqueue pseudocode

4.2.1 Enqueueing

Figure 4 provides the pseudocode which describes the operation of adding a new packet into BSQ.

4.2.2 Dequeueing

Removing a packet from BSQ in order to be transmitted is taking place in two steps: First to find the next sub-queue to be scheduled and second to perform the usual dequeue operation as it is for a normal FIFO queue. The second step is considered trivial and it is not described in detail.

```
scheduledSubQueue getNextPQ(){
        Search in the current root of the subtree and the
        subtree itself{
                If in this iteration the root is chosen to
                be checked
                        If there are packets in the root{
                                scheduled SubQueue is found
                                Notify to check in the next
                                 iteration the first child
                }
                Otherwise, check the chosen child of the
                subtree{
                        Get recursively the
                        scheduledSubQueue, using the
                        chosen child as the new root
                        If this is a valid scheduledSubQueue
                                Notify to check the next child
                                in the following iteration
                }
                If this node has children{
                        Notify to check the next child in the
                        following iteration
                        If the last child is currently visited
                                Notify to check the root in
                                 the next iteration
                }
        }
        return the scheduledSubQueue
}
```

Figure 5: Dequeue pseudocode

Figure 5 provides the pseudocode which describes the operation of finding the next sub-queue to be scheduled. It is a round robin scheduler applied recursively on each level of the tree. An alternative technique that we explored was based on Smoothed Round Robin [6], which provides a scheduling pattern for almost perfect interleaving of the packets. Unfortunately, it proved to be too complex and inefficient because of the microscopic view of the system, applied on the content of a relatively short queue, where the relative ratio of the packets destined to one direction, to the packets destined to another one, was changing very frequently.

4.3 Complexity

The complexity of a queue algorithm is very critical for its deployment. In high speed networks, complexity of even O(log(n)) might make an algorithm useless.

BSQ, although it increases the requirements in the size of the routing tables, it does not keep state for each active queue, but for each neighbor's link, which is into the level of interest. This is a relatively static information, which needs to be updated only when there are changes on the topology of the network, a task that is going to take place anyway. The usual mechanisms of constructing the routing tables can be reused for the purposes of BSQ. Therefore, the complexity of BSQ is O(1).

5 Simulations

5.1 Simulation Environment

In order to evaluate the performance of BSQ, we used ns-2 [7]. In those experiments, compared BSQ with Drop-Tail, DRR and RED. The choice was based on the fact that they are very representative examples and available with the distribution of ns-2.

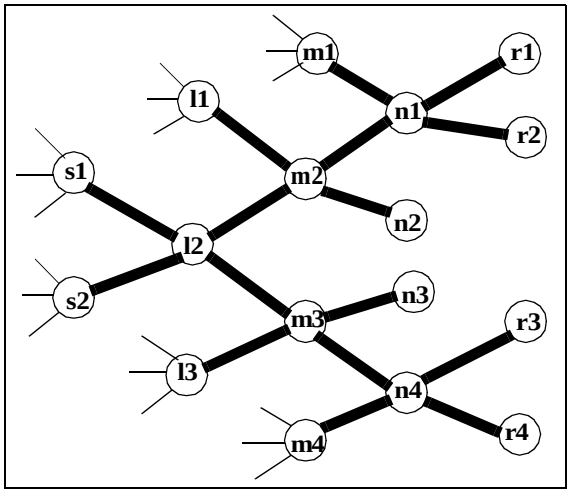

Figure 6: Network topology for the experiments

Figure 6 shows the network topology for the experiments. It represents a part of a backbone example network that can be considered as a graph transformation into a tree to include three levels of cascading networks similar to the one in Figure 1.

In most of the experiments, the sources are located in the open links of the figure, connected to nodes s1, s2, l1, l3, m1 and 4. The destinations are the nodes n2, n3, r1, r2, r3 and r4. The capacity of the links has been set to 12.5MBps and the queue size to 20000 bytes. The flows

1. A different flavor of BSQ, which combines RED characteristics is described later, which randomly picks up the packet to be dropped.

are both TCP that consider the received feedback, as well as self-similar background traffic [10], encapsulated in UDP packets, to make the simulation as much realistic as possible. The version of TCP was in those experiments was TCP Sack. The self-similar traffic has been generated modifying a self similar traffic generator [9], in order to make the output traces compatible with ns-2. The packet size for every flow, both TCP and self-similar, is 500 bytes.

The primary design concern of that simulation environment is to examine how the BSQ queues, located in nodes s1 and s2 are going to reduce the burstiness in node l2, and sequentially, how each level is reducing the burstiness entering in the following one.

5.2 Simulation Results

Figure 7 shows the average queue length of BSQ, DRR and Drop Tail over time, on the outgoing link from node m2 towards n1, since it is a very representative example. It is very clear that DRR constructs much longer queues, resulting in higher end-to-end transmission delays. BSQ and DRR construct queues with similar length on average, but BSQ has lower deviation, resulting in more stable systems and smaller jitter, a very important property for multimedia applications.

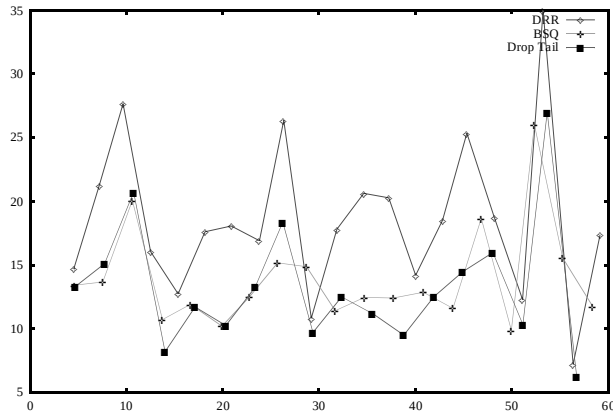

Figure 7: Average queue length for DRR, BSQ and Drop Tail

Table 1 summarizes the most important information from the experiments, comparing the performance among BSQ, DRR and Drop Tail. What is visually obvious in Figure 7, is expressed with the average queue length. DRR constructs queues with almost 50% higher length, compared with BSQ and Drop Tail. As a result, end-to-end delay is also higher, although not that clearly, since the provided numbers include link transmission delay and not only queueing delay. But longer queues has not only drawbacks, but also some benefits. DRR provides the higher utilization (taking in account the number of transmitted TCP packets), compared to BSQ and Drop Tail.

It seems that DRR provides lower goodput for the self-similar UDP traffic, but this might be the case, because that traffic is the aggregation of many others, and DRR can not discriminate between them, to give a more fair share. Finally, DRR suffers from a higher drop rate, compared to BSQ and Drop Tail.

Table 1:

	DRR	BSQ	Drop Tail
Average Queue Length (on link m2-> n1)	18.94	13.98	13.53
TCP delay	70.6 ms	67.8 ms	64.9 ms
TCP packets	125227	104371	91952
UDP delay	11.5 ms	10.2 ms	10.8 ms
UDP packets	257009	263987	265998
Dropped packets	26234	18399	17107

As a second set of experiments, we combined BSQ with RED to examine if it is possible to improve the main drawback of RED[1], which is a low link utilization [12]. Figure 8 displays the average queue length of RED and BSQRED, where they follow a similar behaviour, with smaller length oscillation for BSQRED.

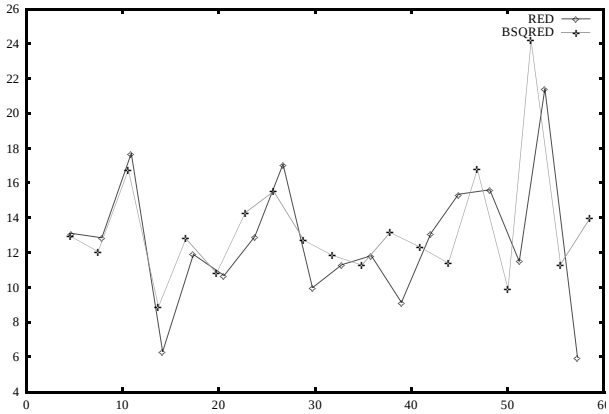

Figure 8: Average queue length for RED and BSQRED

Table 2 shows, as expected the higher link utilization of BSQRED compared with Adaptive RED. It should be mentioned also, that BSQRED drops a higher number of packets.

1. Parameter setting is also very difficult in RED, and for that reason we used the latest version of Adaptive RED.

6 Conclusions & Outlook

In this paper, BSQ, a novel queueing algorithm, has been presented. The main objective of that algorithm is to minimize the burstiness of the traffic on packet switched routers, by interleaving packets that are going to follow different links on next hops. The complexity of that algorithm is O(1), which makes it deployable in high speed networks.

From the experiments we performed, it is proved that BSQ offers a better trade-off between link utilization and end-to-end delay, compared with DRR and Drop Tail. Combined with RED, it can improve the link utilization, without any significant increase on the end-to-end delay.

It is important to mention that the particular experiment results for BSQ are heavily based on the selected packet scheduler, that we do not claim that it is optimal. But still, it is sufficient to prove the benefits of BSQ. Finding the optimum scheduler for this dynamic environment is a great challenge. Prediction mechanisms could improve a lot the performance.

We plan to perform more experiments to compare BSQ with other available queueing algorithms. It is very important to examine the available possibilities to increase the fairness of the algorithm. Also, alternatives to the level-of-interest-based round robin algorithm seems a promising direction to increase even further the performance of BSQ.

Table 2:

	A-RED	BSQRED
Average Queue Length (on link m2->n1)	12.73	13.19
TCP delay	65.17	67.3 ms
TCP packets	88279	103001
UDP delay	10.4 ms	9.96 ms
UDP packets	265497	263351
Dropped packets	17609	25455

Acknowledgements

The authors would like to thank Jens Schmitt for the very useful discussions we had on BSQ.

References

[1] I. D. F. M. Brandauer, Ziegler. Comparison of tail drop and active queue management performance for bulk-data and web-like internet traffic. In *ISCC'2001 (6th IEEE Symposium on Computers and Communications), Hammamet, Tunisia,*, 2001.

[2] B. Branden. et al. Recommendations on queue management and congestion avoidance in the internet. *RFC 2309*, 1998.

[3] W. Feng, D. Kandlur, D. Saha, and K. Shin. Stochastic fair blue: A queue management algorithm for enforcing fairness. In *Proc. IEEE Infocom*, 2001.

[4] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An algorithm for increasing the robustness of RED, 2001.

[5] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[6] C. Guo. SRR: An 0(1) Time Complexity Packet Scheduler for Flows in Multi-Service Packet Networks. *In Proceedings of the ACM SIGCOMM '01 Conference, San Diego, California*, 2001.

[7] K. Harju and S. Korventausta. Network simulation and protocol implementation using network simulator 2.

[8] Y. Joo, V. Ribeiro, A. Feldmann, A. Gilbert, and W. Willinger. The impact of variability on the buffer dynamics in IP networks, 1999.

[9] G. Kramer. On generating self-similar traffic using pseudo-pareto distribution.

[10] G. Kramer. Self-similar network traffic: the notion and effects of self-similarity and long range dependence.

[11] D. Lin and R. Morris. Dynamics of random early detection. In *SIGCOMM '97*, pages 127–137, Cannes, France, september 1997.

[12] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In *Proc. of 7th. International Workshop on Quality of Service (IWQoS'99), London*, 1999.

[13] H. Ohsaki and M. Murata. Steady State Analysis of the RED Gateway: Stability, Transient Behavior, and Parameter Setting. *IEICE Trans. Commun.*, E85-B(1), 2002.

[14] T. J. Ott, T. V. Lakshman, and L. H. Wong. Sred : Stabilized RED. In *INFOCOM*, pages 1346–1355, 1999.

[15] P. K. Panos Gevros, Jon Crowcroft and S. Bhatti. Congestion control mechanisms and the best effort service model. *IEEE Network Special Issue on the Control of Best Effort Traffic, May/June*, 2001.

[16] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM*, pages 231–242, 1995.

[17] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. *In Proceedings of the ACM SIGCOMM '98 Conference*, pages 118–130, 1998.