Scalable Distributed-FDPS Algorithm for QoS Provisioning

Chee Kheong Siew, Shuai Peng, Wuqiong Luo, Peng Tang, and Yanting Mo
School of Electrical & Electronic Engineering
Nanyang Technological University, Singapore
ecksiew@ntu.edu.sg
peng0018@e.ntu.edu.sg
luow0002@e.ntu.edu.sg

Abstract: With rapid growth of Internet traffic, much effort has been focused on possible QoS provisioning mechanisms for different types of services. Many mechanisms, such as MultiServ and Virtual Time Reference System, have been proposed to provide QoS guarantees but scalability remains a challenging problem. In this paper, a novel Distributed-FDPS algorithm is proposed to remove the flow state and its computation tasks from routers, thereby solving the scalability problem. In this mechanism, the complexity is pushed to user end-systems, leaving simple priority scheduling at all routers. We analyse the process of this mechanism to demonstrate its enhancement and use ns-2 simulation to validate our analytical results. Our ns-2 simulation results also demonstrate that out-of-order packets are eliminated.

1 Introduction

In the literature on QoS provisioning, two well-known QoS architecture: Integrated Service [BCS94] and Differentiated Service [BBC$^+$98] have been proposed and a lot of QoS provisioning schemes are proposed based on the two architectures. The purpose of the Integrated Service is to provide per-flow QoS guarantee in the Internet paradigm. However, the computational complexity of packet scheduling and frequent updating of flow state limit its scalability. Recognising the problem, Differentiated Service classifies flows into a limited number of traffic classes and applies priority scheduling to serve the packets of each class. Therefore, Differentiated Service provides a per-hop per-aggregate service to flows whose end-to-end characteristic is the convolution of these per-hop per-aggregate behaviours in its path leading to coarse QoS guarantees. Based on the Differentiated Service architecture, a MultiServ mechanism [SE06] has been proposed to provide Service Curve assurance for per-flow deterministic delay guarantees with Flow-state-dependent Dynamic Packet Scheduling (FDPS) [SGFE05]. Using packets' flow states, FDPS provides fine granularity packet service differentiation within each class and achieves per-flow end-to-end delay bound independent of the number of intermediate nodes along the path of the flow. With integrating this algorithm and the Differentiated Service Framework, the MultiServ mechanism provides a significant enhancement to deterministic QoS guarantee. However, it requires the maintenance of flow state at every node on the flow path,

which limits its scalability. Also, MultiServ produces a significant amount of out-of-order packets due to packet service differentiation.

To solve these problems in the MultiServ [SE06], a novel QoS service mechanism is proposed in this paper. It employs a new Distributed-FDPS algorithm which assigns dynamic packet priority and shapes flows at the traffic source before sending out the packets into the network. The network nodes on the path only perform simple priority scheduling tasks according to flow class and packet priority.

In this paper, we firstly present the flow classification and link capacity allocation mechanism in Section 2. The Distributed-FDPS algorithm is presented and analysed in Section 3 and 4. In Section 5, we validate the proposed QoS service mechanism by Network Simulator ns-2 and finally conclude this paper in Section 6.

2 Flow Classification and Link Capacity Allocation

In this proposed QoS service mechanism, Internet traffic flows are classified at the traffic sources based on the end-to-end scheduling delay budget of each flow [SE06]. This can provide an end-to-end scheduling delay bond guarantee on a per-class basis. Let there be *M* traffic classes in total in this mechanism, *M* end-to-end scheduling delay bounds are specified for each class respectively, which can be represented by a vector $\mathbf{D} = \{D_M, D_{M-1}, \ldots, D_1\}$ with $D_M < D_{M-1} < \cdots < D_1$. Class *M* and class *1* represent the highest priority and the lowest priority respectively. Also, class *1* is designated as the best effort class with unspecified delay bound which ensures that best effort traffic does not interfere with QoS traffic in higher classes. For any link *j* in the network, the capacity of the link *j* is allocated for the *M* classes of traffic according to a link capacity allocation vector which is defined as $\xi_j = [\alpha_M, \alpha_{M-1}, \ldots, \alpha_1]$, where α_K is the fraction of the capacity allocated to class *K* and $\sum_1^M \alpha_K = 1$.

This end-to-end scheduling delay based flow classification can minimise the possibility of bandwidth over-provisioning by providing service differentiation between flows with different delay requirements. Furthermore, the unused capacity in the higher traffic class will be rolled over to the next lower class until it reaches the best effort class. Thus, the proposed mechanism creates *M-1* number of QoS layers above best effort layer. QoS layers above the best effort layer are reserved for transmitting respective higher class traffic only when needed. Once the higher class traffic finished transmitting, capacities allocated to these layers would be released to the best effort layer. Moreover, this structure provides customers the option to buy different QoS services over specific periods according to their needs.

3 Distributed-FDPS Algorithm

3.1 The Enhanced Dual Token Bucket Structure

Similar to FDPS algorithm, Distributed-FDPS also implements packet priority assignment on per-flow basis. However, it deploys an Enhanced Dual Token Bucket structure only at the traffic source instead of at every network node on the path. Every QoS traffic flow will have its own Enhanced Dual Token Bucket at the sender. As shown in Figure 1, this Enhanced Dual Token Bucket consists of two token buckets. The priority bucket is used for priority assignment and the rate control bucket regulates the output traffic to prevent the packets received at the destination from suffering out-of-order packet problem.

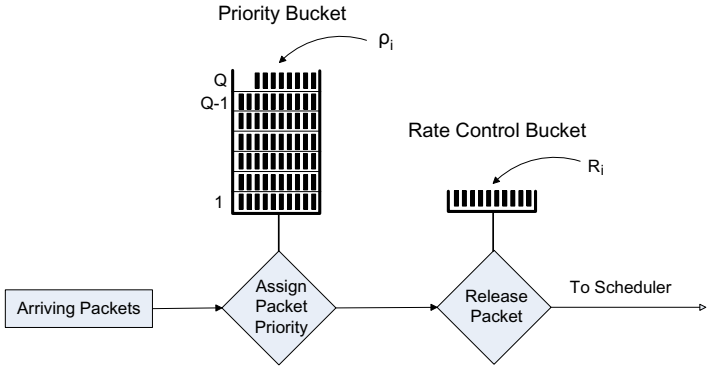

Figure 1: Structure of Enhanced Dual Token Bucket

3.1.1 Priority Bucket

The priority bucket does not regulate the traffic but assigns the priority of a packet according to its state. Flow parameters of any flow i are $(\sigma_i, \rho_i, d_i^q)$, which denote flow bustiness, reserve rate and scheduling delay bond respectively. There are Q layers of tokens in a priority bucket and the width of the bucket is σ_i/Q. From top to bottom, each layer represent packet priority of Q to *1* respectively.

When a new packet of flow *i* arrives, one token is deducted for every byte. Its packet priority will be decided by the layer from which it takes the token for its last byte. For example, if the packet takes the token from layer *3* for its last byte, priority *3* will be assigned to the packet.

The above assignment process will be performed only when there are enough tokens in the bucket for every byte of the newly arrived packet. However, if tokens are not enough at the time, this packet will be assigned to priority *0*, which indicates the packet is considered as non-conformant and will be treated the same as best effort traffic. The pseudo-code for the packet priority algorithm is given below.

Packet Priority Algorithm:

Consider the priority bucket of flow *i* with refilling rate ρ_i, Q as the highest priority and *1* as the lowest priority of conformant packets, and *0* as priority of non-conformant packets. Upon the arrival of k^{th} packet of flow *i*, denoted by P_i^k, a corresponding dynamic priority q_i^k is computed and assigned to P_i^k.

1) Initialise: at system start time

$token = \sigma_i$;
$w = \sigma_i/Q$; (priority bucket width)
$t_{last} = 0$; (time of last priority assignment)

2) Upon the arrival of a new packet P_i^k

a) Refill the bucket

$token = min(token + (t_{current} - t_{last}) * \rho_i, \sigma_i)$;

b) Assign the packet priority

If $(token \geq packet\_size)\{$

$token = token - packet\_size$;
$q_i^k = floor(token/w) + 1$;
$t_{last} = t_{current}$;

$\}$ Else

$q_i^k = 0$;

Endif

c) Forward packet P_i^k to rate control bucket

3.1.2 Rate Control Bucket

During the packet priority assignment, packets might be assigned with a higher priority compared to the priority of the earlier packets if the token refilling rate ρ_i is high enough. In that case, the latter packets with higher priorities might be scheduled and transmitted before the earlier packet. As a result, packets received at destination will be out-of-order. The process is shown in Figure 2.

The rate control bucket is implemented to solve that problem. The bucket depth is set to the largest packet size L_{max} in flow *i*. The bucket refilling rate R_i is set by admission control function that could be implemented possibly by means of a Bandwidth Broker as in [SE06]. Note that in a real network, the end system should not be fully trusted. So the rate control bucket might also need to be implemented at the ingress node to ensure the traffic conformity.

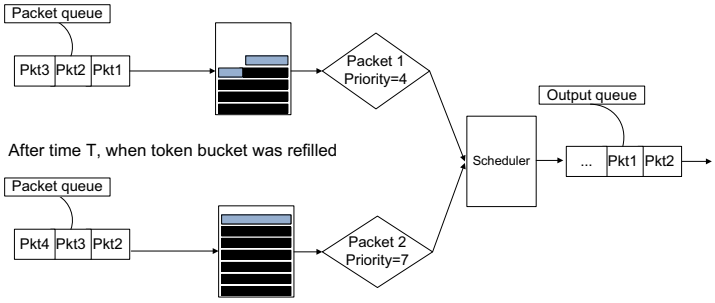

Figure 2: Out-of-order Packet Problem

3.2 Packet Scheduling

The scheduling process in MultiServ mechanism has two levels of granularity: coarse and fine. At coarse granularity, queue from the highest priority class with backlogged packets is always selected for service first. At fine granularity, packets belonging to a class are scheduled by the FDPS algorithm [SGFE05] which provides Service Curves assurance according to its traffic specifications and QoS requirements.

However, it is not necessary to implement the FDPS algorithm at every network node along the path of the flow because the priority of a packet remains unchanged. After recognising that, the Distributed-FDPS mechanism proposed in this paper assigns the dynamic priority only at the traffic source via the implementation of the Enhanced Dual Token Bucket structure as mentioned above.

The network nodes on the path will implement simple priority scheduling algorithm based on the traffic class and packet priority. Therefore, the scheduling mechanism still has two levels of granularity. At coarse granularity, a total of *M* schedulers are needed for these *M* classes of traffic as shown in Figure 3 and only one scheduler is processing its packets at the output link at any time. New packets arriving at a node will be passed to the respective scheduler according to its flow classification. Then, the packets in the scheduler will be served according to packet priority. At both level of granularity, the packets with higher priority (combination of traffic class and packet priority) will always be served first.

4 Analysis of Proposed Mechanism

4.1 Service Curve Assurance

In network calculus [Cru91] [Cru91] [LBT01], if the cumulative arrival process $R^{in}(t)$ be the number of bits that arrive at a system during the interval $[0, t)$, then the minimal arrival curve $A^{min}(t)$ is defined by

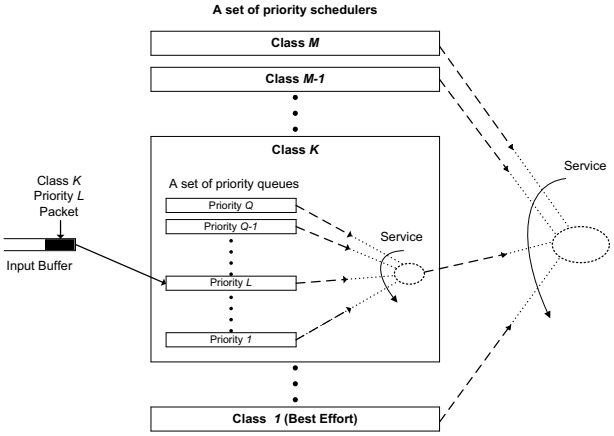

Figure 3: Scheduling Process at the Router

$$A^{min}(t) = sup_{v \geq 0}\{R^{in}(t+v) - R^{in}(v)\}, t \geq 0.$$

The function $A^{min}(t)$ specifies the maximum burstiness for any interval size *t*. Any function that satisfies $A(t) \geq A^{min}(t)$, for all $t \geq 0$, is called a traffic envelope or an arrival curve of $R^{in}(t)$. Let $R^{out}(t)$ be the number of bits that depart the system during the interval $[0, t)$, then the system is said to provide a service curve $B(t)$ if the following holds

$$R^{out}(t) \geq inf_{0 \leq s \leq t}\{A(t-s) + B(s)\}, t \geq 0.$$

For a given flow with arrival curve $R(t)$ and delay bound d. On accepting the flow, the network must provide a service curve $S(t)$ that satisfies, $S(t) \geq R(t-d), t \geq d \geq 0$, which will guarantee scheduling delay no greater than d.

Theorem 1: For a flow with arrival curve $R(t)$ and delay bound d, the Distributed-FDPS algorithm will provide a service curve $S(t)$ that satisfies, $S(t) \geq R(t-d), t \geq d \geq 0$.

Proof: The proof is omitted due to space limitations and we demonstrate the delay bound via ns-2 simulations.

4.2 Router Stateless

The proposed Distributed-FDPS mechanism uses a core stateless algorithm by gracefully eliminating per-flow router state management, removing per-flow state from core routers in [SE06] and pushing the complexity to end-systems.

Another interesting stateless architecture called Virtual Time Reference System (VTRS) [DZYL04] employs packet virtual time stamps to indicate the packet state. As packets traverse through each core router, virtual time stamp carried in each packet is retrieved. Together with some fixed parameters, packet virtual time stamps are computed and up-

dated at each core router. However, since computation is needed for each arriving packet at each core router, the issue of scalability for large number of flows needs to be solved.

For the Distributed-FDPS mechanism, all the priority assignment process has been taken care of at the traffic source. Both edge routers and core routers are relieved from complex computation except for simple priority scheduling. Therefore, the Distributed-FDPS algorithm is not only core stateless but simplifies the scheduling process at all routers thereby addressing the issue of scalability successfully.

Furthermore, for virtual time reference system, edge routers need to maintain the per-flow state. They also need to write packet states into the packet virtual time stamps when packets are injected into the network. Moreover, edge routers need to ensure that the packets of a flow will never be injected into the network at a rate exceeding the flow's reserved rate [DZYL04]. Distributed-FDPS mechanism let the traffic source maintain the per-flow state and assign the priority of a packet before transmitting it. The main complexity of dynamic priority assignment for a packet occurs only once at the traffic source. The edge routers are relieved from performing QoS control functions. Thus the complexity has been pushed to the traffic source instead of the edge and core routers.

4.3 Movable Boundary

Another advantage of the proposed architecture is that the link bandwidth is allocated according to the allocation vector, which can be easily changed by the Bandwidth Broker to maximise the link usage. If the number of users that require a specific QoS service is higher than the capacity of the corresponding class, the boundary of that class can be moved to meet this requirement. For example, suppose there is a link with the following allocation vector, ξ=$[0.3, 0.3, 0.3, 0.1]$ for class $[4, 3, 2, 1]$ respectively. If the demanded capacity for class *4* is lower than the allocated capacity, the allocation factor for class *4* can be simply decreased to release the unused capacity to other classes with higher demand. This feature might help the ISP maximise its profit by adjusting the allocation vector to achieve minimum unused capacity in each class.

5 Simulation

5.1 Topology

In this section, we present a test suite to validate the proposed QoS mechanism by using the Network Simulator ns-2 [M$^+$00].

As shown in Figure 4, two routers are created in this test suite. Congestion is more likely to occur at the bottleneck link between the two routers, which has 10M of bandwidth and 5ms of propagation delay.

Test parameters are shown in Table 1 and Table 2 below. Every simulation lasts 30 seconds.

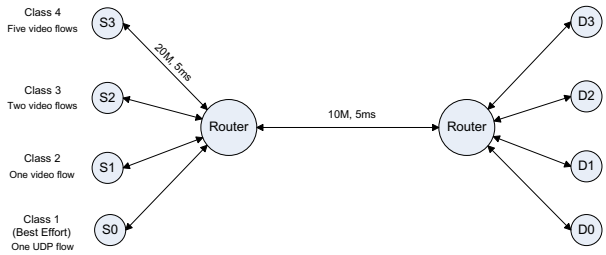

Figure 4: Topology of Test Network

Parameter	Video Traffic	UDP Traffic
Reserved Rate ρ	500 kbps	800 kbps
Burstiness σ	40000 bits	-
Average Packet Size	4000 bits	8000 bits

Table 1: Flow Parameters

5.2 Result Analysis

In the first experiment, we test the mechanism without the rate control bucket to see if there is any out-of-order packet. We also record the actual scheduling delay for comparison.

As shown in Figure 5, out-of-order packets recur for about every 50 packets in each class if the rate control bucket is not implemented. However, the maximum scheduling delays are about 16ms, 33ms and 42ms for Class *4*, Class *3* and Class *2* respectively, which means that the delay bound for every class is upheld even with out-of-order packet problem.

In the second experiment, we turn on the rate control bucket and use the same network topology and traffic parameters. Figure 6 shows that, with the rate control bucket implemented, the out-of-order packet problem is eliminated, the actual scheduling delays for each class are reduced significantly, which are about 4ms, 26ms and 34ms for Class *4*, Class *3* and Class *2* respectively.

Through the first test suite, we notice that the actual rate used by reserved flows account for less than 50% of the link capacity. In order to increase the link utilisation, the allocation

Parameter		Class 4	Class 3	Class 2
Allocation Factor		0.3	0.3	0.3
Scheduling Delay Bound		20 ms	40 ms	80 ms
Priority Bucket	Bucket Depth	40000 bits	40000 bits	40000 bits
	Token Rate	500 kbps	500 kbps	500 kbps
Rate Control Bucket	Bucket Depth	4000 bits	4000 bits	4000 bits
	Token Rate	2 Mbps	1 Mbps	0.5 Mbps

Table 2: Parameters for Distributed-FDPS Algorithm

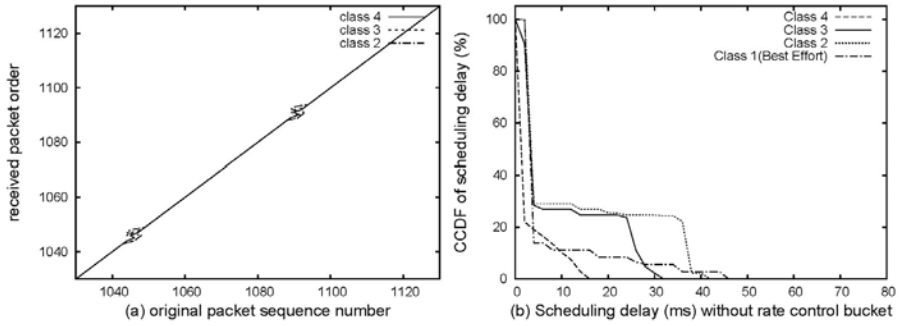

Figure 5: Out-of-order Analysis and Delay Distribution without Rate Control Bucket

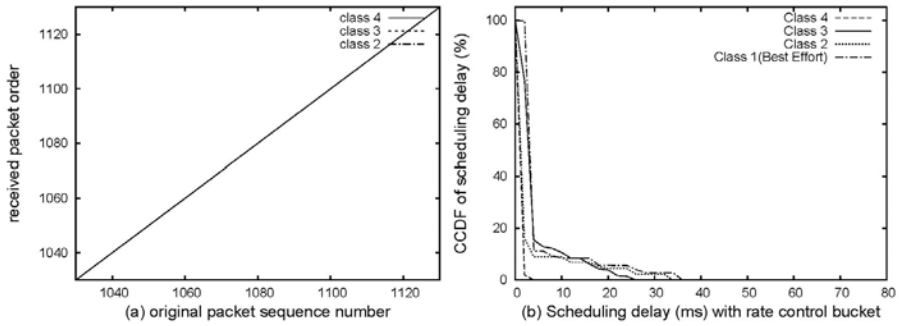

Figure 6: Out-of-order Analysis and Delay Distribution with Rate Control Bucket

vector is changed to $[0.1, 0.2, 0.3, 0.4]$, the average packet size for UDP traffic is changed to 3600 bits and there are 2, 4, 5, 1 flows for Class *4*, *3*, *2* and best effort class respectively. Other parameters remain unchanged and each simulation lasts for 30 seconds in the second test suite.

The simulation results of actual scheduling delay without and with rate control bucket implemented are shown in Figure 7. While Figures 7(a) and 7(b) demonstrate that scheduling delay bounds are upheld in both cases, most packets are delivered well below their delay bounds and narrower delay spread when individual source rate control bucket is implemented.

6 Conclusion

Although the MultiServ can effectively reduce the computational complexity when providing fine granularity scheduling, the scheduling tasks inside routers are still very heavy. This is because routers have to maintain flow state information, which also happened in

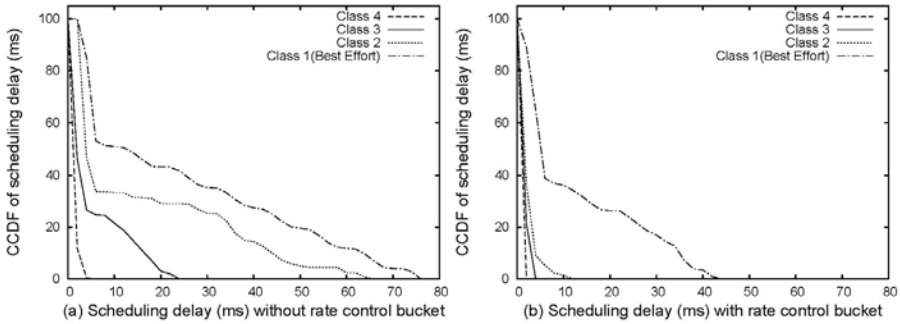

Figure 7: Scheduling Delay Analysis For Second Test Suite

the edge routers of Virtual Time Stamp System. Another serious problem of MultiServ mechanism is the out-of-order packet issue. To solve these problems, we proposed a novel Distributed-FDPS algorithm that pushes the packet priority assignment to the traffic source and remove flow state from all routers. Furthermore, rate control bucket is added to solve the out-of-order packet problem. We validated the proposed mechanism with ns-2 simulator to show that the delay bounds are upheld and the out-of-order packet problem is solved. In conclusion, the proposed mechanism can effectively simplify the router scheduling task and be integrated with the current best-effort service gracefully to provide deterministic QoS guarantees.

References

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC2475: An Architecture for Differentiated Service. *RFC Editor United States*, 1998.

[BCS94] R. Braden, D. Clark, and S. Shenker. RFC1633: Integrated Services in the Internet Architecture: an Overview. *RFC Editor United States*, 1994.

[Cru91] R.L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE transactions on information theory*, 37(1):114–131, 1991.

[DZYL04] Z. Duan, Zhang. ZL, Hou. YT, and Gao. L. A core stateless bandwidth broker architecture for scalable support of guaranteed services. *IEEE Transactions on Parallel and Distributed Systems*, pages 167–182, 2004.

[LBT01] J.Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Verlag, 2001.

[M+00] S. McCanne et al. Network simulator ns-2, 2000.

[SE06] C.K. Siew and M.H. Er. Multiservice provisioning mechanism with service curves assurance for per-class scheduling delay guarantees. *IEE Proceedings-Communications*, 153:846, 2006.

[SGFE05] C.K. Siew, Feng. G, Long. F, and M.H. Er. Congestion control based on flow-state-dependent dynamic priority scheduling. *IEE Proceedings-Communications*, 152(5):548–558, 2005.