Stateful DBA Hypervisor Supporting SLAs with Low Latency & High Availability in Shared PON

Frank Slyne, Sanwal Zeb and Marco Ruffini
CONNECT Center, Trinity College Dublin, Ireland
slynef@tcd.ie, szeb@tcd.ie, marco.ruffini@tcd.ie

Abstract: We present a stateful DBA hypervisor for shared PONs capable of meeting specific flow-level service level agreements, supporting services requiring strict latency and high availability. We report considerable improvement compared to standard stateless priority-based mechanisms.

1. Introduction

Since the release of the Central Office Rearchitected as a Data Centre (CORD) [1] framework, disaggregation and virtualisation of the access networks have progressed steadily. CORD further evolved into the ONF SDN-Enabled Broadband Access (SEBA), and more recently culminated with the release of AETHER, a virtualisation framework integrating the concept of virtual central office with mobile access and edge cloud. One of the key advantages of such network virtualisation approaches is that they provide a much higher degree of flexibility and customisation, which are key for supporting new 5G and beyond services and their related business models.

Customisation is especially important to support the next generation of applications, requiring high capacity and strict latency and availability. These include system critical applications for automation, across several areas, such as e-Health, industry robotics and transportation. Such applications often require precise target performance to operate correctly, which can be expressed in terms of service level agreements (SLAs), e.g., that 99% of packets are delivered within a given latency threshold. While this type of SLA was in the past only applicable to specific physical link performance, we envisage that the flexibility brought about by virtualisation will provide the means to scale application of SLAs to individual services or traffic flows. This is indeed a key milestone for supporting Ultra-Reliable and Low Latency Communications (URLLC) type of services.

Today, the way Quality of Service (QoS) is implemented in Passive Optical Networks (PONs) is still based on stateless, relative prioritisation of packets, which is common across many packet switching technologies. Different packets are marked with different priority levels and then queue management algorithms are applied to determine how packet prioritisation is applied. This operates using priority scheduling mechanisms, ranging from simple round robin to more complex algorithms such as priority-based deficit weighted round robin (PB-DWRR). While these can be effective in assigning relative importance to different services, they are unable to capture the complexity of specific SLAs, especially in a heterogeneous and shared network environment. This shortcoming leads to inefficient utilisation or resources and inability to guarantee performance.

In this work we address this issue for a shared PON, where different Virtual Network Operators (VNOs) present independent SLA requirements. Our starting point is a virtualised shared PON, where each VNO runs its own virtual DBA, which can thus indicate the exact grant allocation required by the services it supports (which is key to support low-latency applications). Our contribution is the design of a hypervisor for the DBA, which can schedule allocations, minimising the likelihood of breaching any of the SLAs.

Our target scenario is shown in Fig. 1, with a virtual PON [2] that enables multiple VNOs to run their own virtual DBA, according to the applications they need to satisfy. A typical example is when the PON connects the Remote Unit (RU) and Distributed Unit (DU) of a Cloud-RAN. Here the system needs to run a Cooperative Transport Interface (CTI) [3] in order for the PON and Cloud-RAN schedulers to synchronise their upstream scheduling, to minimise latency. In this use case, DBA virtualisation enables different independent Cloud-RANs (i.e., from different VNOs) to share the same PON infrastructure. In previous work [4], the QoS issues was addressed through prioritisation, by giving certain flows higher priority than others. However, as mentioned above, this mechanism cannot be directly mapped to SLA requirements, which for example could define the percentage of packets that need to be delivered within a target latency threshold. In this paper we show how our proposed stateful mechanism can instead support SLA-oriented services (such as Cloud-RAN fronthaul).

2. SLA-oriented DBA Hypervisor

The implementation of an SLA-oriented DBA hypervisor requires a stateful mechanism that can make prioritisation decisions based on the history of grant allocations at a flow level. This enables the hypervisor to track how close a given flow is to breach its SLA and to priotitise grant scheduling accordingly. The scope of our hypervisor is to minimise the likelihood of breaching any of the SLAs associated to the different services from multiple VNOs. Our system stores SLA parameters, as shown in the example in Fig. 1 in the table on the right hand side, indicating

the latency threshold (in μs) and target availability (i.e., the percentage of grants it to be delivered below threshold before breaching the SLA).

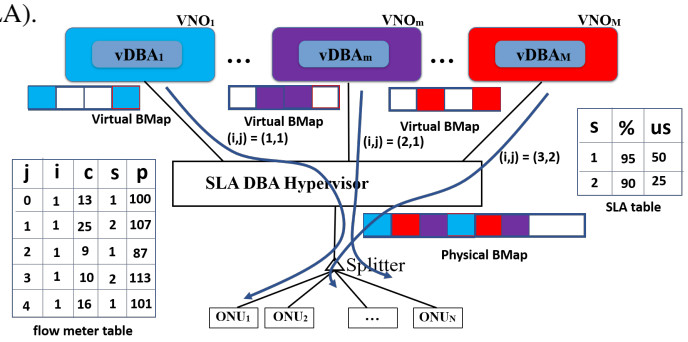


Fig. 1: *Architecture of the SLA-oriented DBA hypervisor, implementing a stateful mechanism through meter tables.*

The hypervisor keeps a flow meter table, shown in the left hand of Fig. 1, recording, for each SLA-oriented flow the following information: the VNO ID (column labelled j); the flow ID (i); the number of grant allocations for which the latency threshold was exceeded (c); the SLA identifier (s), linked to the SLA table; and the total allocations count (p). Because of the large number of flows served by any given OLT, only SLA-oriented flows are allocated an entry in the flow meter table. This puts a limit on the memory and processing time associated with the execution of the algorithm.

The algorithm implemented in the hypervisor is reported in Fig. 2. It uses a cost function (metertable.cost) to schedule the allocations within the next physical frame. Such cost function is proportional to the likelihood of breaching a flow SLA. Firstly, the algorithm assesses the latest time (metertable.maxtime) at which allocations across all virtual BWMAPs can be scheduled to keep within the bounds of their SLA. Secondly, it calculates an initial estimate of the cost of placing the allocations, considering the time assigned by the originating vDBAs in the virtual BWMAPs. Thirdly, it resolves collisions that arise when allocations from different virtual BWMAPs compete for the same allocation slots calculated in the initial estimation. The algorithm schedules the allocation with the lowest cost sooner, and moves instead higher cost allocations, thereby avoiding breaches of SLAs that are close to their availability limit. Lastly, in the event that an allocation is scheduled to a time that exceeds its SLA threshold, the algorithm updates the counter for that flow (the value 'c' in the flow meter table). The flows whose counter is closer to the SLA availability limit will have lower likelihood to be delayed in the event of future collisions. The collision resolution mechanism is based on the Insertion Sort algorithm, making the DBA hypervisor efficient in terms of memory and computation.

Finally, at the end of the SLA breach detection window, the hypervisor calculates, for each SLA flow, the ratio between the number of instances where a grant allocation exceeded the latency threshold and the total number of grant request. If this ratio is higher than allowed by the availability parameters for that SLA, a breach is detected.

```
Result: SLA Bandwidth Map respecting SLA
Input: metertable, slatable, alloc[] array
Output: metertable costed by breach and delay
for i=1 to metertable.length do
  // lastest scheduling threshold
 metertable.maxtime(i) = alloc(i).scheduletime + slatable(alloc(i).sla))
  // Calculate Cost Function
  metertable.cost(i) = 100/(metertable.count(i)+1/metertable.maxtime(i))
// Apply Cost functions highest to lowest
for j=2 to metertable.length do
      key=metertable.priority[j]
      i=j-1
      while I > 0 and metertable.priority[i] > key do
            // Detect Collison
            metertable.cost[i+1]=metertable.cost[i]
            i=i-1
            metertable.priority[i+1] = key
      endwhile
endfor
// Update the metertable with the Actual scheduled times
for i=1 to metertable.length do
      if metertable.scheduletime(i) > metertable.maxtime(i) then
      // the threshold is exceeded, so increase the meter for that flow SLA
      metertable.count(i) = metertable.count(i)+1
```

Fig. 2: *Pseudo-code for the implementation of the SLA-oriented DBA hypervisor*

3. Results showing improvement over standard stateless prioritisation

In order to evaluate the performance of our SLA-oriented DBA hypervisor, we consider a scenario with two VNOs sharing a PON. We assume equal sharing of capacity, i.e., each VNO is allocated 50% of the upstream capacity, aggregated across multiple flows. Both VNOs allocate their BWMAP independently, though their vDBA algorithms, filling in completely their allocation. However, in the simulations we vary the proportion of traffic that is SLA-oriented to be between 10% and 90% of the total. Intuitively, the lower the proportion of SLA-oriented traffic, the higher the compliance with such SLAs.

A relevant parameter that can affect performance is the size of grant allocations, because when grants from two VNOs overlap, a larger grant size increases the number of bytes by which grants will be shifted, thus increasing the probability of exceeding the latency threshold. Our upstream frame is split into 1152 allocation units. One unit is the smallest upstream grant that can be allocated to one ONU and for a 10G PON it has a size of 135 bytes and approximate duration of 0.11 µs [2]. In our scenario we consider three different average sizes for the grants of 10, 35 and 70 units, measuring respectively, 1.3KB, 4.7KB and 9.5KB. The number of flows considered is inversely

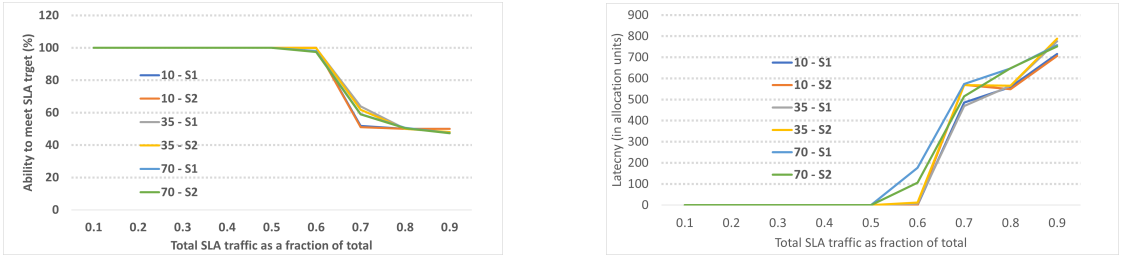
proportional to the grant size, in order to maintain the product of the flow size by the number of flows constant for each VNO (i.e., the VNOs aggregate allocation remains constant).

For SLA-oriented flows, we define two SLAs: S1 requires that any additional latency introduced by the DBA hypervisor is maintained below 25 μs, a condition that needs to be satisfied for 95% of the grant allocations; S2 has a target latency of 12.5 μs for 90% of the grant allocations. In the simulation, SLAs breaches are calculated every 1000 frames (corresponding to 125 ms, which is the size of our breach detection window), after which counters are reset. Such SLA violation could be considered a micro-breach of SLA. For a Cloud-RAN service for example, such breach would trigger a number of excess HARQ transmissions, leading to small waste of capacity. For a practical implementation, multiple micro-breaches could be aggregated over a certain period of time (e.g., monthly) and penalty fees be charged by the VNO to the infrastructure provider accordingly.

Our simulations were carried out on a custom, mixed C and Python simulator with NumPy statistical and numerical processing libraries. All simulations results reported below were averaged over 100 independent runs.

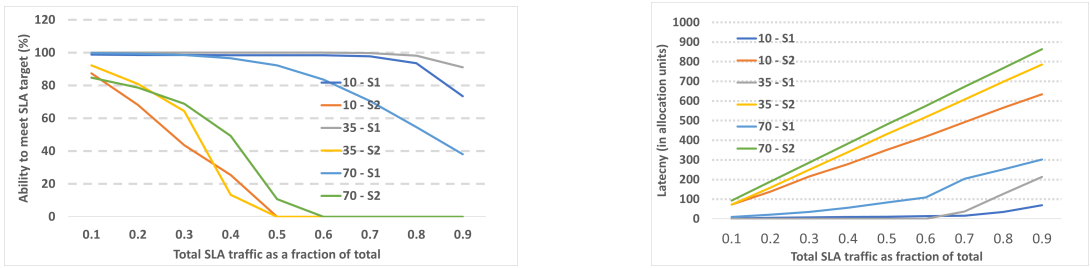
Figure 3a reports the results of SLA compliance for the six SLA-oriented flows: 3 allocation sizes (10, 35, 70) and 2 SLAs (S1, S2), as a function of the proportion of traffic that is SLA-oriented. Here a 100% value on the vertical axis indicates that the SLAs are met without any breach, while lower values indicate the proportion of detection windows that indicated SLA breaches (detection windows are 1000 frames long, as mentioned above). We can see that almost all allocation types always meet SLAs up to 60% of SLA-oriented traffic offered (with the larger 70 grant size just slightly dropping at 50% traffic), after which availability starts dropping significantly. Performance are also similar with respect to grant size, with larger grants suffering slightly increase in latency, as it can be seen more clearly in Fig. 3b.

The key advantage of our SLA-oriented DBA hypervisor is however that both SLA services are equally maintained up to that point, independently of their latency and availability details. In Fig. 4a for comparison we adopt a stateless algorithm, as in [2], associating SLAs to different priority levels: we give flows with SLA S1 higher priority over SLA S2, which in turn get higher priority over non-SLA traffic (the latency for this case is reported in Fig. 4b). Here we notice that SLA compliance cannot be controlled adequately (for 70 grant size even SLA S1 is not met above 30% load) and generally SLA S1 availability is maintained at the expense of SLA S2 flows, which are totally unable to meet their SLA. **This shows once more the necessity of using stateful approaches over stateless algorithms to support SLA-oriented traffic.**



(a) *Ability to meet SLA for varying amount of SLA traffic* (b) *Average delay (expressed in allocation unit of 0.11 µs each)*

Fig. 3: *Performance of our proposed stateful SLA-oriented DBA hypervisor*



(a) *Ability to meet SLA for varying amount of SLA traffic* (b) *Average delay (expressed in allocation unit of 0.11 µs each)*

Fig. 4: *Baseline comparison of stateless priority-based QoS mechanism*

Acknowledgements

Financial support from Science Foundation Ireland grants 12/RC/2276P2, 14/IA/2527 and 13/RC/2077 is acknowledged.

References

1. L. Peterson et al. Central Office Re-Architected as a Data Center. Commag, Oct. 2016.
2. M. Ruffini et al. The Virtual DBA: virtualizing passive optical networks to enable multi-service operation in true multi-tenant environments. JOCN, Apr. 2020.
3. O-RAN. Fronthaul Cooperative Transport Interface Transport Control Plane Specification 1.0. Apr. 2019.
4. A. Ahmad et al. Capacity sharing approaches in multi-tenant, multi-service PONs for low-latency fronthaul applications based on cooperative-DBA. M1K.3, OFC 2020.