Token Bucket-based Throughput Constraining in Cross-layer Schedulers

Jeremy Van den Eynde and Chris Blondia

University of Antwerp - imec
IDLab - Department of Mathematics and Computer Science
Sint-Pietersvliet 7, 2000 Antwerp, Belgium
{jeremy.vandeneynde,chris.blondia}@uantwerpen.be

Abstract

In this paper we consider upper and lower constraining users' service rates in a slotted, cross-layer scheduler context. Such schedulers often cannot guarantee these bounds, despite the usefulness in adhering to Quality of Service (QoS) requirements, aiding the admission control system or providing different levels of service to users.

We approach this problem with a low-complexity algorithm that is easily integrated in any utility function-based cross-layer scheduler. The algorithm modifies the weights of the associated Network Utility Maximization problem, rather than for example applying a token bucket to the scheduler's output or adding constraints in the physical layer.

We study the efficacy of the algorithm through simulations with various schedulers from literature and mixes of traffic. The metrics we consider show that we can bound the average service rate within about five slots, for most schedulers. Schedulers whose weight is very volatile are more difficult to constrain.

Keywords

Cross-layer Scheduling, Quality of Service, Token Buckets, Resource allocation

Part of this research work was carried out at UAntwerpen, in the frame of Research Project FWO nr. G.0912.13 'Cross-layer optimization with real-time adaptive dynamic spectrum management for fourth generation broadband access networks'. The scientific responsibility is assumed by its authors.

1. Introduction

In shared communication networks users often have to compete for service. For example, in wireless networks, the total available capacity is constantly fluctuating due to interference, non-stationary objects etc. Hence, users have to cooperate with each other to ensure the available capacity is shared fairly.

Also in some wired network scenarios, such as DSL, there is competition between users due to cross-talk [1]. This cross-talk occurs in the copper cables when a user's signal leaks into other users' cables, thereby reducing the rate region, the set of all users' possible simultaneous data rates. This rate region is considered convex, implying that increasing one user's service rate, will decrease other users' rates.

In these two settings, the physical layer has many Pareto-optimal operating points, which are all considered equally good to the physical layer. To the upper layers they are, however, not all equally good: allocations that satisfy the user's QoS are preferable. For example, a live video feed has different requirements than video on demand, with regard to delay and data rates. Through cross-layering, the passing of information over the boundaries of the traditional OSI model layers, the upper layers can steer the physical layer. This cross-layering is often abstracted into a Network Utility Maximization (NUM) problem, described by Equation 1.

$$\mathbf{C}^* = \arg\max_{\mathbf{C}\in\mathcal{R}} \sum_{n=1}^{N} u^n(C^n, \mathcal{S}) \tag{1}$$

Natarajan Meghanathan et al. (Eds) : CSEIT, CMLA, NeTCOM, CIoT, SPM, NCS, WiMoNe, Graph-hoc - 2019
pp. 209-219, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.91318

Here n is the user index, N the number of users, and utility function $u^n(\cdot)$ is an interface between the layers, representing the value of a user receiving a service rate C^n. The state of the system $\mathcal{S}$ can include any observable information, such as queue lengths, arrivals, delays and others. We will omit $\mathcal{S}$ for readability when it is clear from the context. In general $u^n(\cdot)$ is an increasing, concave and differentiable function. Examples of cross-layer schedulers are mentioned in subsection 1.2.

Equation 1 chooses the operating point $\mathbf{C}^*$ from the rate region (all possible combinations of service rates) such that average utility over all users is maximized. In this system, a user n's service rate depends on the weight in relation to all other users. This implies that it is challenging to control a single user's service rate, since there is no correspondence between one user's isolated weight and its received rate. In spite of that, sometimes we want to be able to constrain the short-term (and long-term) average rates, for example to ensure the QoS of the users. We give more reasons for implementing rate constraints in subsection 1.1.

Most cross-layer schedulers, such as the ones listed in subsection 1.2. do not offer the option to confine the service rate. Therefore, after reviewing the system model in section 2. we describe our contribution, the Token Bucket Rate Modifier (TBRM) algorithm, in section 3. It is a generic low-complexity algorithm that constrains the short- and long-term average service rates of all users in a utility function-based cross-layer scheduler. Each time slot t, the NUM problem is solved, but each user's weight is replaced by $\phi^n \exp(\frac{k_g^n}{\sigma_g^n} + \frac{k_M^n}{\sigma_M^n})$. Two counters, k_g^n and k_M^n, track the deficiency and excess in service, respectively. If a user n has received less service than ρ_g^n, the amount of tokens will accumulate, and the user's weight will increase, therefore raising the probability of receiving more data rate. The parameters σ_g^n and σ_M^n are a measure for the slowness to react to deficiency and excess respectively. The variable ϕ^n accounts for non-positive weights. The algorithm is very easy to incorporate into any scheduler, as it does not require manipulating the original scheduler's weight function.

In section 4. we evaluate our algorithm through multiple simulations. We compare our results with the unbounded scenarios, and look at the influence of the slot size τ and parameter σ. The metrics indicate that we can bound the average service rate for all users within a limited amount of time. Schedulers whose weight fluctuate heavily are more difficult to constrain.

We close the paper with related works in section 5. and a conclusion in section 6..

1.1. Motivation for service rate constraints

In a multi-user environment, it is useful to ensure that flows are guaranteed a bound on the average service rate. A provider might offer different QoS guarantees to different users, depending on the subscribed model. This might include a guaranteed and/or maximal rate.

There are other reasons to ensure a minimal rate. For example applications like audio and video need a minimal rate for a satisfying Quality of Experience (QoE). Additionally, the authors of [2] observe that TCP-based applications can lead to large queues when the throughput is too small. Finally, a guaranteed rate ensures that misbehaving competing flows, cannot smother flows from receiving their fair share.

Applying an upper bound on a user's capacity is also useful. For example, to accommodate a new flow into a network, admission control algorithms often require an upper bound on the data rates [3]. If a flow disrespects this rate, other applications in the network can suffer deteriorated QoS. By limiting the maximal data rate, a misbehaving flow is isolated and cannot negatively impact the other applications, but will only punish itself. In addition, it can also be useful to provide different service levels to users, where an operator may choose to cap the data rate for cheap data services, and remove this limit for the more expensive premium services.

Although rate constraints can be implemented into the physical layer, it might be interesting to handle it at higher layers. First, it reduces the degrees of cross-layer freedom, and limits the communication necessary. Second, this allows a more flexible approach, allowing for temporary violations. Finally, it might not always be possible to introduce the rate constraints into the physical layer. For example, the scheduler is implemented in hardware or closed-source and cannot be modified, or the corresponding NUM problem's complexity might increase too much due to the additional constraints.

Imposing upper bound constraints can be easily accomplished using a token bucket counter on a user's output stream. This is wasteful though: as the medium is shared, increasing one user's service rate means decreasing other users rates. By applying a token bucket, the excess capacity is thrown away as it is reserved by a particular user.

1.2. Cross-layer schedulers

There are many existing cross-layer schedulers, each employing different metrics. We list here schedulers that are used in the simulations in section 4.

Most schedulers found in literature are linear, meaning that the utility is of the form $u(C^n) = \omega^n C^n$. For example, the Max-Weight (MW) scheduler, presented in the seminal work [4], has $\omega^n = Q^n$.

The Modified Largest Weighted Delay First (M-LWDF) [5] uses $\omega^n = \alpha^n \Gamma^n \frac{1}{\bar{C}^n}$, where $\alpha^n = \frac{-\log(\epsilon)}{\hat{T}^n}$, ϵ is maximal delay violation probability, $\hat{T}$ the delay upper bound, Γ^n is the Head-of-line delay and $\bar{C}^n$ the exponentially averaged assigned data rates.

The Exponential/PF (EXPPF) [6] scheduler, is a combination of Proportionally Fair scheduler and an exponent. It calculates the weights for real-time flows as $\exp(\frac{\alpha^n \Gamma^n - \chi}{1+\sqrt{\chi}}) \frac{1}{\bar{C}^n}$, where $\chi = \frac{1}{N} \sum_{n=1}^{N} \alpha^n \Gamma^n$.

The Maximal Delay Utility (MDU) scheduler [7] employs the average waiting time: $\omega^n = \frac{|u'^n(\bar{U}^n)|}{\bar{\lambda}^n}$, where u'^n is the derivative of the traffic class' utility function, $\bar{U}^n$ the average waiting time and $\bar{\lambda}^n$ the average arrival rate.

The MD scheduler is the non-linear equivalent of the MW scheduler, as it uses the utility function $u(C^n) = Q^n/C^n$. Finally, the Minimal Delay Violation (MDV) scheduler [8] is a non-linear scheduler of the form $u(C^n) = -\frac{\omega^n}{C^n}$, where the weight ω^n tries to minimize the amount of delay violations by looking at the queue and past delays.

2. System model

Time in our model is divided into slots of τ seconds. There are N users, indexed by $n \in [1, N]$, each of which can send $\tau \cdot C^n(t)$ bits during slot $t \in \mathbb{N}$, where $0 \leq C^n(t) \leq \hat{C}^n$ is the capacity for user n.

The user's arrivals and departures during slot t are denoted by $A^n(t)$ and $D^n(t)$ respectively while the queue at the start of slot t is indicated by $Q^n(t)$. The capacities $\boldsymbol{C}(t)$ are determined by a scheduler, based on weights $\boldsymbol{\omega}(t)$: at the start of slot t, a request is made to the scheduler, the reply of which is applied at the start of slot $t+1$. There is thus a delay of τ seconds between a request and application of the rates. The scheduler takes the best matching rate from the rate region $\mathcal{R} \subset \mathbb{R}_+^N$.

Each of the N users has one traffic stream with delay upper bound $\hat{T}^n$, with the additional constraint that flow n's average service rate is bounded: $0 \leq \rho_g^n \leq C^n \leq \rho_M^n \leq \hat{C}^n$.

3. The Token Bucket Rate Modifier algorithm

3.1. Token buckets

Token buckets are found in various situations, such as describing traffic flows [9–11], to check conformance of incoming or outgoing traffic (policing and shaping [12]), traffic marking in DiffServ [13, 14] and rate estimation [15].

Conceptually, a token bucket $TB(\rho, \sigma)$ consists of a bucket holding k tokens (e.g. bits). Tokens are added at a constant rate ρ to the bucket, which is capped at σ tokens. Whenever a packet of B bit passes and there are sufficient tokens, B tokens are removed from the bucket and the packet continues its journey. If $k < B$, the packet is considered non-conforming and an appropriate action is taken, such as being color-marked non-conforming, queued (shaping) or dropped (policing). Such a token bucket will limit the long-term average outgoing rate to ρ. On a short-term scale, bursts of up to σ bits can be served.

3.2. Algorithm

In the following algorithm, this token bucket principle is used to lower and upper bound the service rate in a cross-layer scheduler setting. But, in contrast to a regular token bucket, we now do not cap the tokens to σ. Rather, they are used to indicate the severity of the excess.

In the algorithm, instead of solving

$$\arg\max_{\mathbf{C} \in \mathcal{R}} \sum_n f(C^n)\omega^n \tag{2}$$

where $f(C) = C$ for the linear, and $f(C) = C^{-1}$ for the reciprocal variant, the NUM problem is modified to

$$\arg\max_{\mathbf{C} \in \mathcal{R}} \sum_n f(C^n)\phi^n \exp(\frac{k_g^n}{\sigma_g^n} + \frac{k_M^n}{\sigma_M^n}) \tag{3}$$

Here, $k_g^n \in [0, \infty[$ and $k_M^n \in]-\infty, 0]$ are the tokens for the guaranteed and maximal token buckets, respectively. Every slot, the tokens are updated according to the following rules:

$$k_g^n(t+1) = \max\{0, k_g^n(t) + (\rho_g^n - C^n(t))\tau\} \tag{4}$$

$$k_M^n(t+1) = \min\{0, k_M^n(t) + (\rho_M^n - C^n(t))\tau\} \tag{5}$$

When the received capacity for a user n in the past slots is less than the guaranteed rate ρ_g^n, the virtual token counter k_g^n will continue to increase as long as there is a deficit in received service, and hence the weight will exponentially increase. Likewise, if a user n has received more than ρ_M^n service, the virtual token counter k_M^n will have a negative drift, as long as more data rate is assigned to the user. This will reduce the user's weight exponentially. When the service rate is less than ρ_M^n, the token counter will return to 0.

We introduce

$$\phi^n = \begin{cases} \bar{\omega}, & \text{if } \omega^n \leq \epsilon \text{ and } \frac{k_g^n}{\sigma_g^n} + \frac{k_M^n}{\sigma_M^n} \neq 0 \\ \omega^n, & \text{else} \end{cases} \tag{6}$$

to account for non-positive weights ω^n. Here $\bar{\omega}$ is the exponentially averaged sum of all the positive weights, and ϵ a small number that will result in a capacity close to zero (for the simulations we used $\epsilon = \max_n\{\omega^n\} \cdot 10^{-5}$). If $\omega^n \in]0, \epsilon]$ it becomes difficult to increase the bandwidth reliably, and in the case of $\omega^n = 0$, it is even impossible, since the weight will remain zero. For a negative weight, which can occur for example for best effort flows in the EXP/PF scheduler, the additional factor would just result in an even lower weight, and would also inhibit us from receiving service. $\bar{\omega}$ is used to approximate a valid weight that is reasonably stable. This weight is scheduler and traffic dependent, and thus must be calculated at run time.

Note that if $\rho_g^n = 0$ or $\rho_M^n \geq \hat{C}^n$, then respectively the first and second exponent will always be 1, and the respective bound is disabled.

It can be seen that if a flow stays within the bounds, then the tokens k_g^n and k_M^n will remain zero, and $\phi^n = \omega^n$, resulting in the unmodified weight. Only if some rate guarantee will not be met, weights will be adapted.

3.3. Discussion

Parameters ρ_g^n and ρ_M^n
The choice of ρ_g and ρ_M influences the speed at which the rate can adapt. For example, if the guaranteed rate $\rho_g^n = 0.75\hat{C}^n$, then in each slot the tokens can increase by at most $0.75\hat{C}^n$, and the negative drift is at most $0.25\hat{C}^n$. Thus, if this flow has been receiving no service, then the tokens - and thus the weight too - will increase quickly. If it is receiving service at a rate $\hat{C}^n$, then the tokens will decrease more slowly. A small ρ_g^n thus also implies a small positive and large negative drift. A similar reasoning can be applied to the maximal rate ρ_M^n.

Parameters σ_g^n and σ_M^n
In the traditional token bucket algorithms, σ is a measure for the burstiness of a flow. For example, large values of σ mean that large bursts are allowed. In our algorithm, the σ can be interpreted as a measure for slowness to react. A large value of σ_M^n means that longer periods of above-guaranteed service rates are possible, because our weight will decrease more slowly. Small values of σ_M^n will react quicker and can lead to an overreaction. The two token buckets can also influence each other: in case of an overreaction, the other token bucket will also have a sudden excess, and in turn have a fiercer reaction. This can be observed for small values of σ in the simulations of subsubsection 4.5.2.

Slot size
In our system, the slot size implies a delay between a request for and subsequent assignment of the capacity. A larger slot size means that changes will be slower, and that predicting future traffic becomes more important. This also matters to the rate constraint algorithm, since the scheduler's response to weights becomes more unpredictable, hence modifying the weights. The simulations of subsubsection 4.5.3. briefly look at increasing slot sizes.

exp
The function exp is chosen here to modify the token fractions, but any continuous, strictly increasing function $\alpha(\cdot)$ for which holds that $\alpha(0) = 1$, $\lim_{x\to-\infty} \alpha(x) = 0$ and $\lim_{x\to\infty} \alpha(x) = \infty$ will give rate guarantees, albeit with different bounds. Tests with different functions resulted in more short-time erratic behavior.

Additive form
Instead of using a product, it is also possible to use an additive form,

$$\arg\max_{\mathbf{C}\in\mathcal{R}} \sum_n f(C^n)\omega^n + (\alpha(\frac{k_g^n}{\sigma_g^n}) + \alpha(\frac{k_M^n}{\sigma_M^n}))\beta$$

Here α is a continuous, strictly increasing function with the properties $\alpha(0) = 0$, $\lim_{x\to-\infty} \alpha(x) = -\infty$ and $\lim_{x\to\infty} \alpha(x) = \infty$. An additional factor β must be introduced to account for the fact that ω^n is usually not unitless.

We ran some simulations for $\alpha(x) = x$ and $\alpha(x) = x^3$, and $\beta = \bar{\omega}$. The simulations showed that this approach is also possible, and avoids the non-positive weight problem which forced us to introduce the factor ϕ^n. However, in the NUM problem, what matters is the relative weights, rather than the absolute difference, which the additive form expresses. Even though on larger timescales this leads to nicely averaged data rates, on short timescales the behavior is very extreme, where $C^n(t)$ alternates between 0 and rates close to $\hat{C}^n$ in successive slots.

Complexity
The space and time complexity of the TBRM algorithm is very low. Every slot we update the N users' token counters k_g^n and k_M^n (Equations (4) and (5)). Additionally, we have to select a suitable ϕ^n for all n. The exponentially weighted $\bar{\omega}$ is a constant time operation $\mathcal{O}(1)$. The resulting time complexity is thus $\mathcal{O}(3N + 1) = \mathcal{O}(N)$.
Likewise, the space requirements are equally low: we track the $2N$ counters, and a single exponentially weighted $\bar{\omega}$. The space complexity is in this case $\mathcal{O}(2N + 1) = \mathcal{O}(N)$.

Other considerations
Applying rate guarantees transforms a work-conserving scheduler into a non-work conserving scheduler. I.e. the scheduler might have capacity assigned, even though there are no jobs available.

Additionally, throughput constraints reduces the stability region of a scheduler. Roughly, a scheduler is called stable for an arrival process if all the queues remain bounded. The set of arrival rates for which a scheduler is stable, is called the stability region. Schedulers such as MW [4] and MDU [7] have been proven to be stable for the widest range of arrivals.
Applying a constraint on the data rate for such schedulers reduces its stability region. Enforcing a maximal data rate inside the stability region, clearly decreases this region. Also supporting a minimal throughput constraint influences the stability region: a minimal throughput constraint can be rewritten as a (more complex) maximal throughput constraint on the other users.

4. Simulations

4.1. Simulation setup

We evaluated the TBRM algorithm using simulations for the schedulers listed in subsection 1.2. We ran simulations in OMNeT++ using the INET framework. Every $\tau = 50$ms the original weights and weight modifiers were computed. The resulting NUM problem was then solved with the help of the nlopt [16] library, by first applying the local variant of the DIviding RECTangles algorithm [17], followed by the COBYLA algorithm [18], to obtain the final rate, applying them in the next slot.

4.2. Rate region

The rate region was artificially generated by the following formula, which is based on the n-sphere formula.

$$r^n = \hat{C}^n \prod_{i=1}^{n-1} \sin(\phi^i)^{1-\gamma} \cos(\phi^n)^{1-\gamma}, r \in [1, N-1]$$
$$r^N = C_{max}^N \prod_{i=1}^{N-1} \sin(\phi^i)^{1-\gamma}$$

For N users, the rate region is the set of points generated by varying $\phi^i \in \left[0, \frac{\pi}{2}\right], \forall i \in [1, N-1]$. The modifier $\gamma \in [-1, 1]$ changes the shape of the rate region. If we set $\hat{C}^n = M, \forall n$, then the shape ranges from an n-simplex for $\gamma = -1$, over an n-sphere with radius M for $\gamma = 0$, to a hypercube for $\gamma = 1$.

4.3. Scenarios

The scenarios listed in Table 1 show the different types of traffic and the applied constraints.

The traffic types behave differently on short and large timescales. The first type of traffic consists of a sine-wave, superimposed with a faster oscillating sine-wave. Some flows will oscillate slowly (Sine2VS) while others oscillate fast (Sine2F). The second type of traffic is the heavy tail traffic, which is either a trace file of a video file, such as Starwars, or a self-similar flow, generated by a superposition of Pareto-distributed sources [19]. The last class of traffic, SAT, tries to send as much traffic as possible, by ensuring the queue is always backlogged.

These scenarios are run for $\tau = 0.05s$ in subsubsection 4.5.1. In subsubsection 4.5.2. we vary σ, and in subsubsection 4.5.3. it is τ that changes.

Scenario	User 1	User 2	User 3	User 4	User 5
1	SAT [150,250]	SAT [250,350]	SAT [350,400]	SAT [150,350]	SAT [50,100]
2	Starwars [50,150]	Alice [250,350]	Self-Similar [150,350]	SAT [150,350]	Sine2VS [50,120]
3	Starwars [50,150]	Sine2F [250,350]	Self-Similar [150,350]	SAT [150,350]	Sine2VS [50,120]
4	Sine2VS [150,250]	Sine2VS [150,250]	Sine2VS [250,300]	Sine2VS [150,350]	Sine2VS [50,400]
5	Sine2VS [150,250]	Sine2VS [150,250]	Sine2VS [250,300]	Sine2VS [150,350]	Self-Similar [0,0]

Table 1: Summary of scenarios. Listed for each user are traffic type, and $[\rho_g, \rho_M]$ in Mbps.

4.4. Metrics

We examined three different metrics. The m2 and m3 metrics are defined on windows of size G, which groups G consecutive slots.

- m1 the percentage of slots that would be marked non-conforming by a token bucket process $TB(\rho_g, \rho_g \tau x)$ and $TB(\rho_M, \rho_M \tau x)$ for respectively the guaranteed and maximal rate. $x \in \mathbb{R}^+$ is a variable indicating the allowed burstiness. Increasing x allows for more burstiness, and will result in a smaller percentage of non-conforming slots.
- m2 The average amount of excess bits per window G. If we define the amount of bit reserved in window w as $C^G(w) = \sum_{t=wG}^{(w+1)G} C(t)\tau$, and W as the total number of windows, then the m2 metric for respectively the guaranteed and maximal rate can be formally described as $\langle\{\max\{\rho_g G\tau - C^G(w), 0\} | w = 0..W-1\}\rangle$ and $\langle\{\max\{C^G(w) - \rho_M G\tau, 0\} | w = 0..W-1\}\rangle$. This is a representation of the severity of the average violation. A larger number indicates more severe violations. The metric can be visualized by imagining the surface above or below the required rate. Increasing G decreases the m2 metric as we smooth out excess bits over a larger window.
- m3 $\langle B^G \rangle$: where B^G is the set of consecutive violating windows. This metric gives an idea of how grouped violations are. For example, if this number is large, it means that a violation is resolved slowly.

4.5. Results

4.5.1. Regular scenarios

In the following plots, we averaged over all schedulers and scenarios, as showing the individual schedulers would result in a cluttered plot. Important discrepancies between schedulers will be discussed in the text.

Each plot has two curves, one of which displays the results for which no rate constraints were applied, as a base case, and the other has our TBRM algorithm applied.

m1

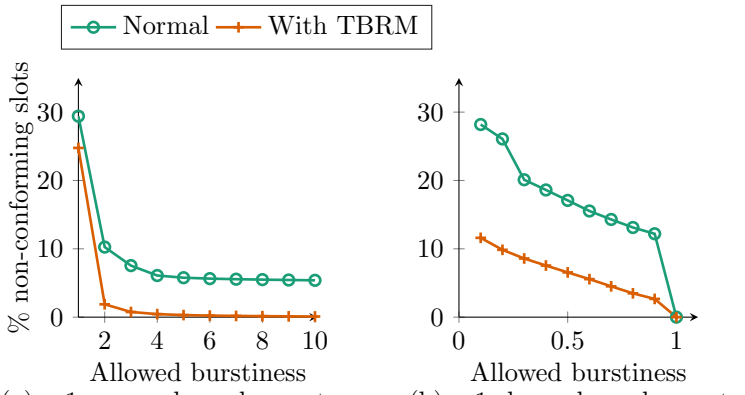
The m1 metric is shown in Figure 1, which displays on the x-axis the allowed burstiness, and on the y-axis the percentage of non-conforming slots.

If we examine Figure 1a, which shows the m1 metric for the upper bound, then we can see that for $x = 1$, the number of violations is close to the results of the unconstrained simulations. Increasing x, the allowed burstiness, however, we can observe that the violation probability quickly drops for our TBRM algorithm, and becomes almost 0 when the allowed burst size is $5\rho_M\tau$. This indicates that the violations occur irregularly spread. The unconstrained results remain fixed around 6% for a long time.

The underlying data shows that for the constrained scenarios all the schedulers inhibit the same behavior: there is a steep decline in violations, going from $x = 1$ to $x = 2$, and then they gradually go to almost 0 for $x = 5$.

This behavior is the same for all schedulers over all traffic classes. However, the initial violation probability for SAT class is slightly lower than the video, self-similar and sine classes. The SAT traffic is easier to correct due to its queue based nature.

In the m1 plot of the guaranteed rate in Figure 1b, our domain is limited to $]0, 1]$: if $x = 1$, then it means that approximately in every slot we allow a deficit of $\rho_g\tau$ bit, which is obviously the maximum deficit we can attain per slot. In the plot, one can see that there is a much wider gap between the constrained and unconstrained scenarios, confirming the efficacy of our algorithm.



(a) m1: upper bound on rate (b) m1: lower bound on rate

Figure 1: m1 for the regular scenarios

The data shows here that the majority of the violations come from the MW and M-LWDF schedulers, and more specifically for the video streams. For example, in the TBRM scenarios, for $x = 0.1$, both schedulers have a violation probability of about 20%, while the other schedulers are closer to 6%.

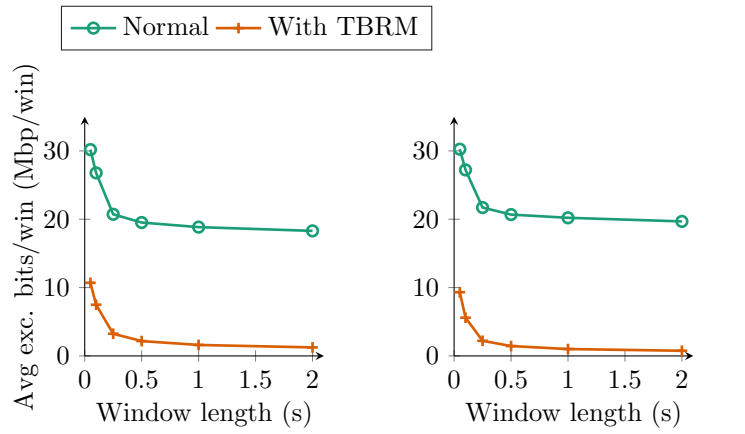
This difference can be explained by the fact that in those linear schedulers the queue length is used as a weight. This number is immediate, which causes a more unpredictable weight (especially in combination with a linear scheduler), making it more difficult to estimate a suitable weight modifier. Additionally, the weight can become 0 very easily. This requires the use of the additional ϕ^n modifier. Even though $\bar{\omega}$ is smoother, the switch between $\bar{\omega}$ and ω^n can also be disruptive. However, this extra factor is necessary, as simulations without this correction ϕ^n, result in a much higher violation probability.

The curve looks quite linear. This can be explained by the fact that the guaranteed rate violations are more evenly spread out.

m2

Ideally, we can limit the rate immediately. However, there is an inherent delay of 1 slot, and an elasticity in the form of a burst factor. Therefore, we study the rate, when we group $\frac{G}{\tau}$ slots into windows of size G.

The m2 metric in Figure 2, displays the average amount of violated bits per window, for increasing window sizes G.



(a) m2: upper bound on rate (b) m2: lower bound on rate

Figure 2: m2 for the regular scenarios

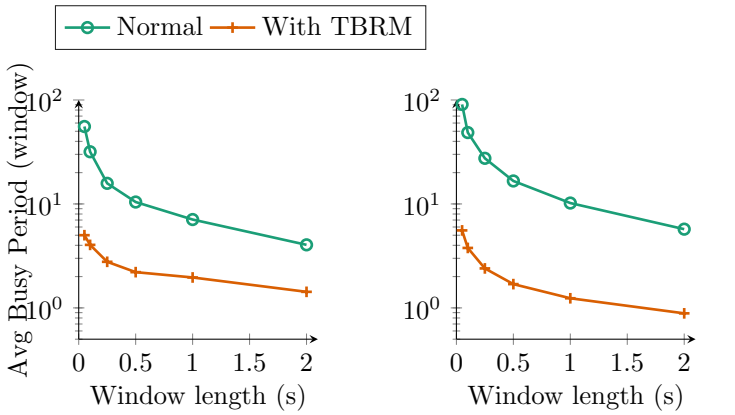
It can be observed that for a window size of $G = 0.05$s, for the unconstrained scenarios there is an average of 30Mbit/window in excess of the target rate. When we constrain it using our algorithm, this drops to about 10Mbit/window.

Increasing the window size G, averages out bursts. Like the results for m1, there is a steep decline until $G = 0.25$s, which coincides with 5 slots, after which the bit violations remains stable in the upper bound case. Though less pronounced, also here do the MW, M-LWDF and MD schedulers fare the worst for small window sizes, mainly for the Self-Similar traffic.

The decline implies that bursts are usually short-lived: overflow and good windows are usually close together, as they don't violate the constraints when merged. This is also confirmed in the m3 metric, below. The rate of decline is similar for the scenarios with and without TBRM applied.

m3

The last metric discusses the average length of a violation streak. Figure 3 shows the average number of successive windows that violate their constraints in a log-plot. The m3 metric, like the m2 metric, initially decreases quickly as the window size increases, and then slowly decreases. It can be clearly seen that, regardless of the unconstrained behavior, the TBRM algorithm limits the bursts to 5 windows, for $G = \tau$, dropping to 2 windows for $G = 5\tau$. These short bursts confirm that the algorithm is able to fix excesses within about 5 slots.



(a) m3: upper bound on rate (b) m3: lower bound on rate

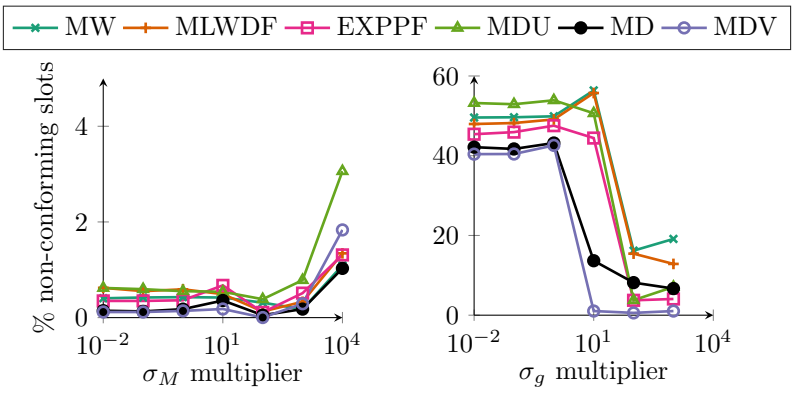
Figure 3: m3 for the regular scenarios

The main contributor to the average in this metric is the MDU scheduler. The data show that this is because whereas other schedulers consist of many smaller busy periods, the MDU scheduler has only one or two large busy periods, increasing the average significantly.

Without the MDU data, the average for 5 windows is about 1, for the minimal rate constraint, and 2 for the maximal rate constraint.

4.5.2. Study of parameter σ

In this section we look at the results of varying burst parameter σ. We let $\sigma_g = i\tau\rho_g$ and $\sigma_M = i\tau\rho_M$, for $i \in [10^{-2}, 10^4]$, and look at the effect on the m1 metric in Figure 4, which shows the results for the individual schedulers.



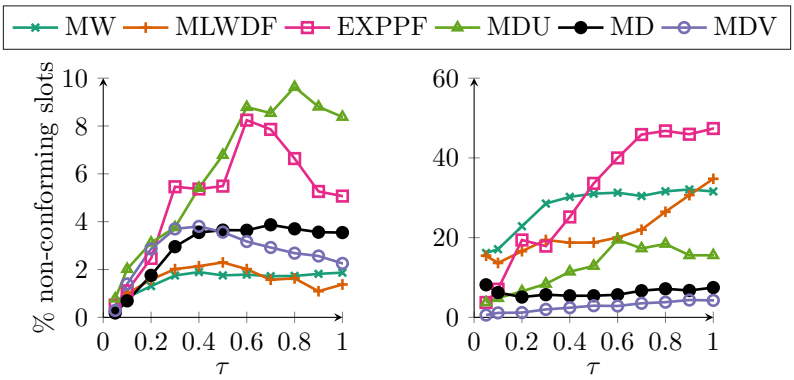
(a) m1: upper bound on rate (b) m1: lower bound on rate

Figure 4: m1 for the σ scenarios

The plot shows that for the upper bound constraints, the violation probability is always very low (about 4% at most for $i = 10^4$). The lower bound, however, starts around 50% violation probability, and suddenly drops to smaller probabilities for $i = 10^2$.

Indeed, a small σ_M will overflow quickly, which causes its weight to be reduced swiftly, hence there will be less violations. As σ_M grows, the weight modifier will decrease much more slowly, leaving more room for violations. For the guaranteed rate, on the other hand, k_g cannot build up a deficit as fast as k_M, as discussed before. It is only when the growths of the deficits are balanced that the m1 metric can lower, which is around $i = 10^2$ and upwards.

The schedulers that perform the worst are, unsurprisingly, the MW and M-LWDF schedulers.



(a) m1: upper bound on rate (b) m1: lower bound on rate

Figure 5: m1 for the τ scenarios ($\sigma = 5\tau\rho$)

4.5.3. Study of parameter τ

In the previous simulations, we assumed a slot length of $\tau = 0.05s$. This study observes how the TBRM algorithm changes in function of τ.

In Figure 5 the m1 metric for $\sigma = 5\tau\rho$ is plotted. It can be observed that mainly the lower bound in Figure 5b is sensitive to an increasing slot length. This might be because with an increasing τ also grows the probability of a larger delay: if in a slot a low capacity was assigned erroneously, the delays or queues will increase and additionally it takes longer to correct, causing larger queues. Especially the EXPPF scheduler suffers from this, as it is of the form $\exp(\Gamma)$, where Γ is the Head-of-line. As the non-linear Min-Delay (MD) and MDV schedulers try to minimize the delay, they suffer less from an increase of slot size.

For the upper bound in Figure 5a only the MDU and EXPPF schedulers seem to suffer from the increased slot size. This is probably due to the fact that it is easier to receive a service lower than ρ_M.

5. Related work

In [5, 20] the authors use virtual tokens as a measure for the average waiting time, and incorporate it with the M-LWDF [5] and EXPPF [6] scheduling algorithm to warrant a minimal rate. It is, however, not transferable to other schedulers. In other schedulers, the guaranteed rate constraint is built into the scheduler itself [21, 22], but they are all scheduler-specific and don't allow enforcing a maximal data rate. The authors of [23] consider utility based throughput allocation subject to certain properties, but is only valid for linear utility functions. In [24] a related problem of maintaining an optimal service rate is proposed. In [25], the authors consider a generic algorithm with minimum and maximum rate constraints. It is, however, only applicable to schedulers that operate in function of an average rate. As such, it excludes for example the MW [4], MD and M-LWDF schedulers. Other schedulers, such as, MDU [7] and MDV [8] have a more elaborate utility function and are more difficult to characterize. The authors of [26] also employ a token system, but assume that users lie about their demands to strategically maximize their utility. In [27] constraints are applied to network slices of traffic aggregates in a 5G context, using an additive approach.

6. Conclusion

In this paper we looked at restricting the service rates given to users in a cross-layer scheduler setting. We implemented this using a low-complexity algorithm that modifies the weights in a Network Utility Maximization problem, using the concept of token buckets. We first discussed cross-layer scheduling, and the need to both upper and lower limit data rates assigned to users. Then we proposed the TBRM algorithm, and followed up with simulation results. We ran simulations for six different schedulers, and

multiple scenarios demonstrating that using our approach it is possible to limit the service rate, within error, after about five slots for the maximal and guaranteed service rate for most schedulers. Schedulers that progress smoothly are easier to constrain than schedulers that can behave wildly, such as EXPPF for long slot times, MW or M-LWDF. These are more difficult to restraint with respect to guaranteeing a lower bound on the service rate.

References

[1] J. Verdyck, C. Blondia, and M. Moonen, "Network utility maximization for adaptive resource allocation in dsl systems," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 787–791.

[2] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt, "Flow aggregation for enhanced tcp over wide-area wireless," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3. IEEE, 2003, pp. 1754–1764.

[3] J. Qiu and E. W. Knightly, "Measurement-based admission control with aggregate traffic envelopes," *IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 2, pp. 199–210, 2001.

[4] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE transactions on automatic control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[5] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, P. Whiting, and R. Vijayakumar, "Providing quality of service over a shared wireless link," *IEEE Communications magazine*, vol. 39, no. 2, pp. 150–154, 2001.

[6] R. Basukala, H. M. Ramli, and K. Sandrasegaran, "Performance analysis of exp/pf and m-lwdf in downlink 3gpp lte system," in *Internet, 2009. AH-ICI 2009. First Asian Himalayas International Conference on*. IEEE, 2009, pp. 1–5.

[7] G. Song, "Cross-layer resource allocation and scheduling in wireless multicarrier networks," Ph.D. dissertation, Citeseer, 2005.

[8] J. Van den Eynde, J. Verdyck, M. Moonen, and C. Blondia, "A delay-based cross-layer scheduler for adaptive dsl," in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[9] P. P. Tang and T.-Y. Tai, "Network traffic characterization using token bucket model," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 1999, pp. 51–62.

[10] R. L. Cruz, "A calculus for network delay. i. network elements in isolation," *IEEE Transactions on information theory*, vol. 37, no. 1, pp. 114–131, 1991.

[11] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2001, vol. 2050.

[12] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Transactions on Networking (ToN)*, vol. 6, no. 5, pp. 611–624, 1998.

[13] J. Heinanen and R. Guerin, "Rfc 2697–a single rate three color marker," *IETF, September*, 1999.

[14] ——, "Ietf rfc 2698.","," *A Single Rate Three Colour Marker*, 1999.

[15] E. Zhang and L. Xu, "Capacity and token rate estimation for networks with token bucket shapers," *Computer Networks*, vol. 88, pp. 1–11, 2015.

[16] Steven G. Johnson, "The nlopt nonlinear-optimization package." [Online]. Available: https://github.com/stevengj/nlopt

[17] J. M. Gablonsky and C. T. Kelley, "A locally-biased form of the direct algorithm," *Journal of Global Optimization*, vol. 21, no. 1, pp. 27–37, 2001.

[18] M. J. Powell, "A direct search optimization method that models the objective and constraint functions by linear interpolation," in *Advances in optimization and numerical analysis*. Springer, 1994, pp. 51–67.

[19] X. Bai and A. Shami, "Modeling self-similar traffic for network simulation," *arXiv preprint arXiv:1308.3842*, 2013.

[20] S. Shakkottai and A. L. Stolyar, "Scheduling algorithms for a mixture of real-time and non-real-time data in hdr," in *Teletraffic Science and Engineering*. Elsevier, 2001, vol. 4, pp. 793–804.

[21] M. Mohseni, R. Zhang, and J. M. Cioffi, "Optimized transmission for fading multiple-access and broadcast channels with multiple antennas," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1627–1639, 2006.

[22] X. Wang and N. Gao, "Stochastic resource allocation over fading multiple access and broadcast channels," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2382–2391, 2010.

[23] X. Liu, E. K. Chong, and N. B. Shroff, "A framework for opportunistic scheduling in wireless networks," *Computer networks*, vol. 41, no. 4, pp. 451–474, 2003.

[24] S. Borst and P. Whiting, "Dynamic channel-sensitive scheduling algorithms for wireless data throughput optimization," *IEEE Transactions on Vehicular Technology*, vol. 52, no. 3, pp. 569–586, 2003.

[25] M. Andrews, L. Qian, and A. Stolyar, "Optimal utility based multi-user throughput allocation subject to throughput constraints," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4. IEEE, 2005, p. 2415—2424.

[26] S. M. Zahedi, S. Fan, and B. C. Lee, "Managing heterogeneous datacenters with tokens," *ACM Transactions on Architecture and Code Optimization*, vol. 15, no. 2, p. 18, 2018.

[27] S. Mandelli, M. Andrews, S. C. Borst, and S. Klein, "Satisfying network slicing constraints via 5g mac scheduling," *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 2332–2340, 2019.