SNC-Meister: Admitting More Tenants with Tail Latency SLOs

Timothy Zhu
Carnegie Mellon University
timothyz@cs.cmu.edu

Daniel S. Berger
University of Kaiserslautern
berger@cs.uni-kl.de

Mor Harchol-Balter
Carnegie Mellon University
harchol@cs.cmu.edu

Abstract

Meeting tail latency Service Level Objectives (SLOs) in shared cloud networks is both important and challenging. One primary challenge is determining limits on the multi-tenancy such that SLOs are met. Doing so involves estimating latency, which is difficult, especially when tenants exhibit bursty behavior as is common in production environments. Nevertheless, recent papers in the past two years (Silo, QJump, and PriorityMeister) show techniques for calculating latency based on a branch of mathematical modeling called Deterministic Network Calculus (DNC). The DNC theory is designed for adversarial worst-case conditions, which is sometimes necessary, but is often overly conservative. Typical tenants do not require strict worst-case guarantees, but are only looking for SLOs at lower percentiles (e.g., 99th, 99.9th).

This paper describes SNC-Meister, a new admission control system for tail latency SLOs. SNC-Meister improves upon the state-of-the-art DNC-based systems by using a new theory, Stochastic Network Calculus (SNC), which is designed for tail latency percentiles. Focusing on tail latency percentiles, rather than the adversarial worst-case DNC latency, allows SNC-Meister to pack together many more tenants: in experiments with production traces, SNC-Meister supports 75% more tenants than the state-of-the-art.

Categories and Subject Descriptors C.4 [*PERFORMANCE OF SYSTEMS*]: Modeling techniques; D.4.8 [*OPERATING SYSTEMS*]: Performance—Stochastic analysis, Modeling and prediction

Keywords stochastic network calculus, tail latency guarantees, quality of service

SoCC '16, October 05-07, 2016, Santa Clara, CA, USA.
© 2016 ACM. ISBN 978-1-4503-4525-5/16/10…$15.00.
DOI: http://dx.doi.org/10.1145/2987550.2987585

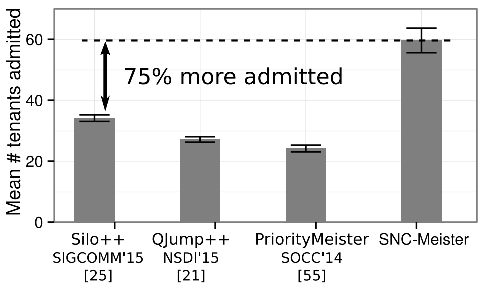


Figure 1. Admission numbers for state-of-the-art admission control systems and SNC-Meister in 100 randomized experiments. In each experiment, 180 tenants, each submitting hundreds of thousands of requests, arrive in random order and seek a 99.9% SLO randomly drawn from $\{10\text{ms}, 20\text{ms}, 50\text{ms}, 100\text{ms}\}$. While all systems meet all SLOs, SNC-Meister is able to support on average 75% more tenants with tail latency SLOs than the next-best system.

1. Introduction

Meeting tail latency SLOs is important

Meeting tail latency Service Level Objectives (SLOs) in multi-tenant cloud environments is challenging. A tail latency SLO such as a 99th percentile of 50ms requires that 99% of requests complete within 50ms. Researchers and companies such as Amazon and Google repeatedly stress the importance of achieving tail latency SLOs at the 99th and 99.9th percentiles [4, 12, 13, 21, 24, 25, 37, 43, 45, 46, 51, 55]. As demand for interactive services increases, the need for latency SLOs will become increasingly important. Unfortunately, there is little support for specifying tail latency requirements in the cloud. Latency is harder to guarantee since it is affected by the burstiness of each tenant and the congestion between them, whereas bandwidth is much easier to divide between tenants. Tail latency is particularly affected by burstiness, and recent measurements show that the 99.9th latency percentile can vary tremendously and is typically an order of magnitude above the median [33].

The case for request latency SLOs

Throughout this paper, we consider cloud tenants that issue a series of requests over time for data items on another server VM within the same datacenter. For example, in Fig. 2, the blue tenant, residing on VM V_1, sends requests to server

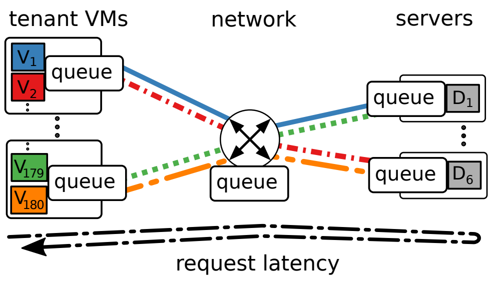


Figure 2. SNC-Meister meets tail latency SLOs for tenants in the network shown. Our evaluation experiments involve 180 tenant VMs (on 12 machines), which replay recent production traces, and six servers running memcached.

VM D_1, which hosts its data. We define SLOs over a pair of VMs (e.g., (V_1, D_1)), which is known in literature as the pipe model. We define SLOs in terms of *request latency* (a.k.a., flow completion time), which is the total time from when a tenant issues a request until *all* the requested data is received. Request latency is different from *packet latency*, which is the time it takes a single packet to traverse through the network. Packet latency is the right metric when requests are small and load is light. However, as the amount of data used increases, request latency becomes the most relevant granularity (as argued in [53]).

Queueing is inevitable for request latency

High request latency is almost always due to excessive queueing delay [21, 25]. Queueing is inevitable. In production environments, traffic is typically bursty. When these bursts happen simultaneously, the result is high queueing delays.

Queueing can occur both within the network (*in-network queueing*) and at the end-hosts (*end-host queueing*). Some works (e.g., Fastpass [37], HULL [3]) claim to eliminate or significantly reduce queueing. What they actually mean is that they eliminate *in-network queueing* by shifting the queueing to the end-hosts with rate limiting. This produces great benefits for packet latency, which does not include this end-host queueing time. However, these techniques do not solve the problem for request latency, which by definition captures the entire queueing time, both in-network queueing and end-host queueing. Both forms of queueing delay comprise the biggest portion of request latency, particularly when looking at the tail percentiles [21]. This paper focuses solely on the effects of queueing (i.e., congestion between tenants) and leaves the mitigation of other sources of tail latency (e.g., VM scheduling, TCP artifacts) to other works (e.g., [50, 51]).

Dual goals: meeting tail latency SLOs and achieving high multi-tenancy

The goal of this paper is two-fold: 1) We want to meet tail request latency SLOs; and 2) we want to admit as many tenants as possible. Clearly, there is a tradeoff between achieving both goals. Admitting few tenants will likely meet SLOs due to limited queueing. Admitting many tenants, in contrast, creates the possibility of SLO violations due to high contention between tenants. *Admission control* is the component that limits the multi-tenancy so as to guarantee that the system *only* admit tenants whose SLOs can be met.

A key challenge in admission control is predicting upper bounds on the request latency for each tenant. Predicting latency bounds is only possible with assumptions on tenant behavior. We address the typical behavior of tenants (i.e., not flash crowds, faulty hardware, etc). We assume that typical tenant behavior can be characterized (or at least upper bounded) by a stationary trace of past behavior. The trace allows us to extract information about the load and burstiness of the tenant (Fig. 3(a) shows three example traces). Note that bursts are short lived (on the order of seconds), and that these bursts are not caused by diurnal (hourly) trends. In this work, we specifically focus on this short-term burstiness, which is separate from time-varying load[1]. These short-term bursts occur during every hour of our traces, and they are known to have a large impact on performance [23].

The state of the art in admission control: worst-case bounds on the request latencies

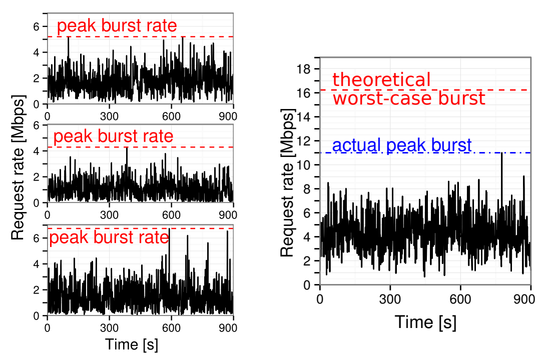
The state-of-the-art in admission control are Silo (SIGCOMM 2015 [25]), QJump (NSDI 2015 [21]), and PriorityMeister (SoCC 2014 [55]). These systems perform admission control by using Deterministic Network Calculus (DNC) to calculate upper bounds on the request latency. Typically, DNC-based systems assume a tenant's request process is characterized based on a maximum arrival rate and burst size. They then use DNC to calculate each tenant's worst-case latency based on the tenants' maximum rate/burst constraints. If the worst-case latency for a tenant is higher than its SLO, the tenant is not admitted.

The above systems all use DNC, but in somewhat different ways. Silo uses DNC to calculate the amount of queueing within the network and performs admission control to ensure that network switch buffers do not overflow. QJump offers several classes with different latency-throughput trade-offs, for which latency guarantees are calculated with DNC. PriorityMeister considers different prioritizations of tenants. For each priority ordering, PriorityMeister uses DNC to calculate the worst-case latency of each tenant. PriorityMeister aims to choose a priority ordering that maximizes the number of tenants that can meet their SLOs if admitted.

The limitations of DNC

The DNC theory predicts worst-case latencies for the adversarial case where the worst possible bursts of all tenants happen simultaneously. While some tenants may be adversarially correlated, it is very conservative to assume all tenants are correlated with each other. The difference between assuming independence and dependence is substantial; as an example, Fig. 3(b) shows the aggregate behavior of the three traces in Fig. 3(a). The peak burst in each trace is marked with a horizontal line. As DNC is an adversarial worst-case anal-

[1] Time-varying load can be accommodated by using a trace from a period of high load or by updating the tenant's trace over time. Our work is still relevant to the short-term burstiness that occurs during periods of high load.



(a) Individual burstiness (b) Aggregate burstiness

Figure 3. Three example production traces and their aggregate trace. A worst-case analysis assumes that all three individual traces have their worst peaks at the same time, which is overly conservative as shown in the aggregate trace.

ysis technique, its equations account for the scenario where each of the peak bursts happen at the same time. But as shown in the aggregate trace, the actual peak is much lower than the adversarial sum of peaks. As a result, DNC's worst-case assumption limits the number of tenants that can be admitted into the system for any given SLOs.

The case for Stochastic Network Calculus (SNC)

Typical users do not seek strict worst-case guarantees. Instead, users target tail latency percentiles lower than the 100%, e.g., the 99.9th latency percentile [13]. DNC only supports calculating the 100th percentile latency (i.e., adversarial worst-case), so given 99.9th percentile SLOs, DNC-based systems simply pretend they are 100th percentile SLOs, resulting in admission decisions which are conservative.

We therefore instead turn to an emerging branch of probabilistic theory called Stochastic Network Calculus (SNC). SNC provides request latency bounds for any user-specified latency percentile, e.g., the 99th, 99.9th, or 99.99th latency percentile. By not making adversarial worst-case assumptions, it is possible to admit many more tenants, even for high percentiles (several 9s).

Support for dependencies in SNC

The SNC theory also supports having certain tenants being dependent on each other, as indicated by a user-specified dependency graph (Fig. 4). A user running several related tenants can specify that a group of tenants are dependent on each other. Dependent tenants in a group are allowed to be adversarially correlated with each other, but are assumed to behave independently in relation to tenants in other groups. Thus, it is possible to capture the benefits of independence without assuming all tenants are fully independent of every other tenant.

Our SNC-based system: SNC-Meister

Our new system, SNC-Meister, uses SNC to upper bound request latency percentiles for multiple tenants sharing a network. Admission decisions are made for the specific latency and percentile requested by each tenant. We implement and

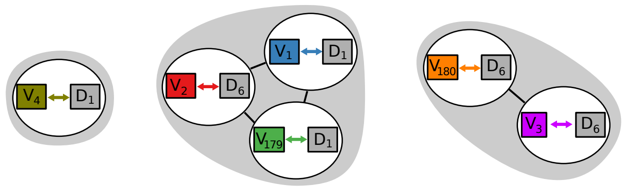


Figure 4. User-specified tenant dependency graph with three groups. Tenants in a dependent group are assumed to be adversarially correlated with each other.

run SNC-Meister on a physical cluster, and our experiments with production traces show that SNC-Meister can support many more tenants than the state-of-the-art systems by considering 99.9th percentile SLOs (see Fig. 1).

This paper makes the following main contributions:

• **Bringing SNC to practice:** SNC is a new theory that has been developed in a purely theoretic context and has never been implemented in a computer system. Our primary contribution is identifying and overcoming multiple practical challenges in bringing SNC to practice (details in Sec. 4). For example, it is an open problem how to effectively apply SNC in non-trivial network topologies and how to incorporate tenant dependencies. We prove the correctness of SNC-Meister's analysis and show that SNC-Meister improves the tightness of SNC latency bounds by 2-4$\times$ (Sec. 4).

• **Extensive evaluation:** We implement SNC-Meister and evaluate it on an 18-machine cluster running the widely-used memcached key-value store (setup shown in Fig. 2, details in Sec. 5). We compare against three state-of-the-art admission control systems, two of which we enhance to boost their performance[2]. Across 100 experiments each with 180 tenants represented by recent production traces, SNC-Meister is able to support on average 75% more tenants than the enhanced state-of-the-art systems (Fig. 1) while meeting SLOs of all admitted tenants. This improvement means that SNC-Meister allows tenants to transfer 88% more bytes in the median (Sec. 6.1). SNC-Meister is also within 7% of an empirical offline maximum, which we determined through trial-and-error experiments (Sec. 6.2).

• **Open-source release of SNC-Meister:** Code for SNC-Meister is available at `https://github.com/timmyzhu/SNC-Meister`. We design SNC-Meister to operate in existing infrastructures alongside best effort tenants without requiring kernel, OS, or application changes. To simplify user adoption, SNC-Meister only requires high-level user input (e.g., SLO, trace) and automatically generates SNC models and corresponding configuration parameters. Our representation of SNC in code is simple and efficient, which results in the ideal linear scaling of computation time for admission decisions in terms of the number of tenants.

[2] Silo++ admits 10% more tenants than a hand-tuned Silo baseline, and QJump++ admits 5$\times$ more tenants than a hand-tuned QJump baseline.

2. SNC-Meister's admission control process

In determining admission, SNC-Meister works with per-tenant tail latency SLOs and traces representing the burstiness and load added by each tenant. Traces consist of a sequence of request arrival times and sizes, and they can be extracted from historical logs or captured during operation. Using traces avoids the burden of having users specify many complex parameters to describe their traffic.

To understand the implications of selecting a representative trace, we first need to consider the differences between short-term burstiness and long-term load variations. Short-term burstiness denotes second/sub-second variations of a tenant's bandwidth requirements. Long-term load variation denotes trends over the course of hours, such as diurnal patterns. While both types of variation affect latency, tail latency is mainly caused by transient network queues due to short-term burstiness. Short-term burstiness can lead to tail latency SLO violations even under low load: in our experiments, SLO violations occurred for network utilizations as low as 40%. In our production traces (described in Sec. 5.3), short-term peaks have a rate that is $2\times$ to $6\times$ higher than the average rate. By comparison, the difference between day-hour rates to night-hour rates is often less than $2\times$.

After receiving a tenant's SLO and trace, SNC-Meister determines admission through the following three steps. First, SNC-Meister analyzes the tenant's trace to derive a statistical characterization understood by the SNC theory (see Sec. 4.3). Second, SNC-Meister assigns a priority to the tenant based on its SLO where the highest priorities are assigned to tenants with the tightest SLOs (i.e., lowest latency value). We opt for this simple prioritization scheme since our experiments with a more complex prioritization scheme [55] show similar results. Third, SNC-Meister calculates the latency for each tenant based on SNC (see Sec. 4.1) and checks if each tenant's predicted latency is less than its SLO. If the previously admitted tenants and the new tenant all meet their SLOs, then the new tenant is admitted at its priority level. Otherwise, the tenant is rejected and can only run at the lowest priority level as best-effort traffic.

SNC-Meister enforces its priorities both in switches and at the end-hosts. To enforce priority at the end-hosts, SNC-Meister configures the HTB queueing module in the Linux Traffic Control interface. To enforce priority at the network switch, we use the Differentiated Services Code Point (DSCP) field (a.k.a. TOS IP field) and mark priorities in each packet's header with the DSMARK module in the Linux Traffic Control interface. Our switches support 7 levels of priority for each port; using this functionality simply requires enabling DSCP support in our switches.

3. Stochastic Network Calculus background

At the heart of SNC-Meister is the Stochastic Network Calculus (SNC) calculator. SNC is a mathematical toolkit for calculating upper bounds on latency at any desired percentile (e.g., 99th percentile). This is in contrast to DNC, which computes an upper bound on the worst-case latency (i.e., 100th percentile). Throughout this paper, we calculate *upper bounds* on latency, which we refer to in the shorthand as calculating latency. Sec. 3.1 describes the core concepts of SNC by way of example (Fig. 5). Sec. 3.2 describes the necessary mathematical details needed to implement SNC.

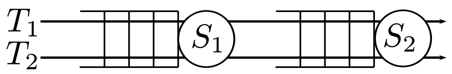


Figure 5. Example network with two tenants T_1 and T_2 flowing through two queues S_1 and S_2.

3.1 SNC core concepts

SNC is based on a set of operators that manipulate probabilistic distributions. We refer to these distributions as *arrival processes* (A_1 and A_2 for tenants T_1 and T_2 in Fig. 5) and *service processes* (S_1 and S_2 in Fig. 5). One of the main results from SNC is a *latency operator* for taking an arrival process (e.g., A_1), a service process (e.g., S_1), and a percentile (e.g., 0.99), and calculating an upper bound on the tail latency. We write this as $Latency(A_1, S_1, 0.99)$. The latency operator works for any arrival and service process.

As an example, consider calculating the 99th percentile latency for T_1 in Fig. 5. Since T_1 and T_2 share the first queue, T_1 does not experience service process S_1 since there is congestion introduced by T_2. Rather, T_1 experiences the leftover (a.k.a. residual) service process after accounting for T_2. In SNC, this is handled by the *leftover operator*, $\ominus$, which is used in our example to calculate a new service process $S'_1 = S_1 \ominus A_2$. T_1's 99th percentile latency at the first queue is then calculated by using the latency operator with S'_1 (i.e., $Latency(A_1, S'_1, 0.99)$).

Calculating T_1's latency at the second queue in Fig. 5 requires arrival processes at the second queue, which are precisely the output (a.k.a. departure) processes from the first queue. In SNC, this is handled by the *output operator*, $\oslash$, which is used in our example to calculate T_1's output process, A'_1, as $A'_1 = A_1 \oslash S'_1$ where S'_1 is as defined above. T_2's output process, A'_2, is calculated similarly. T_1's latency at the second queue is then calculated as $Latency(A'_1, S_2 \ominus A'_2, 0.99)$.

One might try to calculate T_1's total latency by adding up the latencies from each queue (i.e., $Latency(A_1, S'_1, 0.99) + Latency(A'_1, S_2 \ominus A'_2, 0.99)$). However, this is not a 99th percentile latency anymore. To get a 99th percentile overall latency, higher percentiles are needed for each queue (e.g., 99.5th percentile)[3]. There are in fact many options for percentiles at each queue (e.g., 99.5 & 99.5; 99.3 & 99.7; 99.1 & 99.9) for calculating an overall 99th percentile latency. Choosing the option that provides the best latency bound is time consuming, so SNC provides a *convolution operator*, $\otimes$, which avoids this problem by treating a series of queues as a single queue with a merged service process. In

[3] This is formally known as the union bound.

Purpose	$\rho(\cdot)$	$\sigma(\cdot)$
Arrival process A for MMPP with transition matrix Q and diagonal matrix $E(\theta)$ of each state's MGF	$\rho_A(\theta) = sp(E(\theta)\, Q)$	$\sigma_A(\theta) = 0$
Service process S for network link with bandwidth R	$\rho_S(\theta) = -R$	$\sigma_S(\theta) = 0$
Leftover operator $\ominus$ for service process S and arrival process A	$\rho_{S\ominus A}(\theta) = \rho_A(\theta) + \rho_S(\theta)$	$\sigma_{S\ominus A}(\theta) = \sigma_A(\theta) + \sigma_S(\theta)$
Output operator $\oslash$ for service process S and arrival process A	$\rho_{A\oslash S}(\theta) = \rho_A(\theta)$	$\sigma_{A\oslash S}(\theta) = \sigma_A(\theta) + \sigma_S(\theta) - \frac{1}{\theta}\log\left(1 - e^{\theta(\rho_A(\theta)+\rho_S(\theta))}\right)$
Aggregate operator $\oplus$ for arrival process A_1 and arrival process A_2	$\rho_{A_1\oplus A_2}(\theta) = \rho_{A_1}(\theta) + \rho_{A_2}(\theta)$	$\sigma_{A_1\oplus A_2}(\theta) = \sigma_{A_1}(\theta) + \sigma_{A_2}(\theta)$
Convolution operator $\otimes$ for service process S_1 and service process S_2	$\rho_{S_1\otimes S_2}(\theta) = \max\{\rho_{S_1}(\theta), \rho_{S_2}(\theta)\}$	$\sigma_{S_1\otimes S_2}(\theta) = \sigma_{S_1}(\theta) + \sigma_{S_2}(\theta) - \frac{1}{\theta}\log\left(1 - e^{-\theta\lvert\rho_{S_1}(\theta)-\rho_{S_2}(\theta)\rvert}\right)$
Tail latency L for percentile p, arrival process A, and service process S	$L = \min_{\theta} \frac{1}{\theta\rho_S(\theta)} \log\Big((1-p) * \big(1 - \exp(\theta * (\rho_A(\theta) + \rho_S(\theta)))\big)\Big) - \frac{1}{\rho_S(\theta)}(\sigma_A(\theta) + \sigma_S(\theta))$	

Table 1. The SNC operators and equations used by SNC-Meister for independent tenants.

our example, the convolution operator is applied to T_1's leftover service process at each queue as $S_1' \otimes (S_2 \ominus A_2')$. This new service process is then used to calculate T_1's latency as $Latency(A_1, S_1' \otimes (S_2 \ominus A_2'), 0.99)$.

Lastly, SNC has an *aggregation operator*, $\oplus$, which calculates the multiplexed arrival process of two tenants. For example, the aggregate operator can be used to analyze the multiplexed behavior of T_1 and T_2 as $A_1 \oplus A_2$.

The SNC literature provides this set of operators along with proofs of correctness. However, little is known on how to best combine these operators together to analyze networks, and this is a challenging open problem. Sec. 4 describes the challenges in bringing SNC to practice and how we overcome them in SNC-Meister.

3.2 Mathematics behind SNC

In this section, we expand upon the high level description of the SNC concepts in Sec. 3.1 and describe the mathematics behind SNC. To begin, we define the arrival process of a tenant T_1 as $A_1(m,n)$, which represents the number of bytes sent by T_1 between time m and n. As arrival processes are probabilistic in nature, SNC is based on moment generating functions (MGFs), which are an equivalent representation of distributions. Directly working with MGFs is unfortunately quite challenging mathematically, so SNC operates on an upper bound on the MGF, parameterized by two subcomponents $\rho(\theta)$ and $\sigma(\theta)$. For example, the MGF of $A_1(m,n)$, written $MGF_{A_1(m,n)}(\theta)$, is upper bounded by:

$$MGF_{A_1(m,n)}(\theta) \leq e^{\theta(\rho_{A_1}(\theta)(n-m)+\sigma_{A_1}(\theta))} \quad \forall\theta > 0$$

MGFs are parameterized by a variable θ to represent all moments of a distribution (e.g., $A_1(m,n)$). All arrival processes are specified in terms of the two subcomponents $\rho(\theta)$ and $\sigma(\theta)$, and all SNC operators provide equations for these subcomponents (see Tbl. 1 for an overview, and Appendix A.2 in our tech report [54] for full details).

To calculate the $\rho_{A_1}(\theta)$ and $\sigma_{A_1}(\theta)$ for T_1, we need to assume a stochastic process for T_1, such as a Markov Modulated Poisson Process (MMPP) (see Sec. 4.3). A MMPP is useful for representing bursty arrival rates. For example, a 2-MMPP switches between high-rate phases and low-rate phases using a Markov process. The MMPP's transition matrix is given by Q, which for a 2-MMPP has four entries:

$$Q = \begin{pmatrix} p_{hh} & p_{hl} \\ p_{lh} & p_{ll} \end{pmatrix}$$

where, e.g., p_{hl} indicates the probability of switching to a low-rate phase (l) after a high-rate phase (h). The distribution of the arrival rate and request size for each phase is captured in the matrix E, which is a diagonal matrix of the MGF for each phase:

$$E(\theta) = \begin{pmatrix} MGF_h(\theta) & 0 \\ 0 & MGF_l(\theta) \end{pmatrix}$$

Finally, the $\rho_{A_1}(\theta)$ and $\sigma_{A_1}(\theta)$ for T_1 is calculated as:

$$\rho_{A_1}(\theta) = sp(E(\theta)\cdot Q) \text{ and } \sigma_{A_1}(\theta) = 0$$

where $sp(\cdot)$ is the spectral radius of a matrix.

Service processes are defined similarly to arrival processes with the same two subcomponents $\rho(\theta)$ and $\sigma(\theta)$. Rather than working with lower bounds on the amount of service provided, SNC works with an upper bound:

$$MGF_{S_1(m,n)}(-\theta) \leq e^{\theta(\rho_{S_1}(\theta)(n-m)+\sigma_{S_1}(\theta))} \quad \forall\theta > 0$$

where the MGF has an extra negative sign on the θ parameter, which transforms a lower bound into an upper bound. For lossless networks, the $\rho_{S_1}(\theta)$ and $\sigma_{S_1}(\theta)$ have a simple form:

$$\rho_{S_1}(\theta) = -R \text{ and } \sigma_{S_1}(\theta) = 0$$

where R is the bandwidth of the network link.

Lastly, tail latency is calculated by chaining together the equations in Tbl. 1 based how the SNC operators are combined and using the latency equation (last line in Tbl. 1). Sec. 4.4 describes how we represent arrival and service processes in code and how we evaluate the latency equation with the θ parameter.

4. Implementation

In this section, we describe four challenges we overcome in implementing SNC-Meister.

First, SNC is a new theory, and it is currently an open problem how to effectively apply SNC to network topologies (e.g., Fig. 2). The SNC literature is primarily concerned with the theorems and proofs behind individual SNC operators, but little is known about putting them together. Sec. 4.1 describes SNC-Meister's novel network analysis technique and the corresponding improvement in accuracy.

Second, the SNC literature does not consider the analysis of dependencies between tenants. Sec. 4.2 discusses how SNC-Meister handles dependencies and its effect on latency.

Third, real traffic exhibits bursty behavior, particularly at second/sub-second granularities, and it is important to capture this behavior to properly characterize tail latency. Sec. 4.3 describes how SNC-Meister models burstiness and how it estimates model parameters from trace data.

Fourth, it is non-trivial how to work with full representations of probabilistic distributions as required by SNC. Sec. 4.4 describes how SNC-Meister is implemented in code.

4.1 Analyzing networks with SNC-Meister

Analyzing networks with SNC requires an algorithm for combining the SNC operators described in Sec. 3.1. Even with the simple example in Fig. 5, there are multiple ways to analyze the latency for T_1. For example, Sec. 3.1 describes how the latency can be analyzed one queue at a time (i.e., $Latency(A_1, S'_1, 0.995) + Latency(A'_1, S_2 \ominus A'_2, 0.995)$) as well as through a convolution operator (i.e., $Latency(A_1, S'_1 \otimes (S_2 \ominus A'_2), 0.99)$). Yet there is even another approach by first applying the convolution operator on S_1 and S_2 before accounting for the congestion from T_2 (i.e., $Latency(A_1, (S_1 \otimes S_2) \ominus A_2, 0.99)$). While each approach is correct as an upper bound on tail latency, they are not equally tight. One of our key findings is that some approaches can introduce "artificial dependencies" where arrival and service processes are treated as dependent processes even though they should be independent. For example, in $Latency(A'_1, S_2 \ominus A'_2, 0.995)$, A'_1 $(= A_1 \oslash (S_1 \ominus A_2))$ and A'_2 $(= A_2 \oslash (S_1 \ominus A_1))$ are artificially dependent because they are both derived from common sources A_1, A_2, and S_1. Likewise, the convolution $S'_1 \otimes (S_2 \ominus A'_2)$ has an artificial dependency because S'_1 $(= S_1 \ominus A_2)$ and A'_2 are both derived from S_1 and A_2. In reality, there shouldn't be any dependencies between A_1, A_2, S_1, and S_2, but the ordering of SNC operators can introduce these artificial dependencies. A more comprehensive

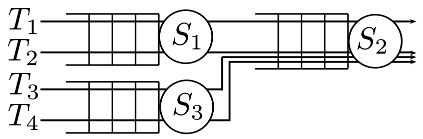


Figure 6. Extending Fig. 5's example with tenants T_3 and T_4 flowing through queues S_3 and S_2.

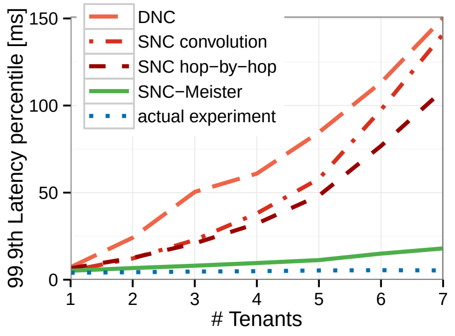


Figure 7. The tail latency calculated using DNC and multiple SNC methods, SNC convolution [15], SNC hop-by-hop [5], and SNC-Meister. In this micro-experiment, we vary the number of tenants connecting from a single client to a single server through two queues.

example for artificial dependencies can be found in Appendix A.3 in our tech report [54].

In our SNC-Meister SNC algorithm, we identify two key ideas that allow us to eliminate artificial dependencies.

Key idea 1. When analyzing T_1, SNC-Meister performs the convolution operator before the leftover operator for any tenants sharing the same path as T_1. For example, $Latency(A_1, (S_1 \otimes S_2) \ominus A_2, 0.99)$.

Using this idea in our Fig. 5 example avoids the artificial dependencies at the second queue. However, there are other sources of artificial dependencies. Fig. 6 shows a slightly more complex scenario with additional traffic from T_3 and T_4. Calculating T_1's latency now requires accounting for the effect of T_3 and T_4 at the second queue S_2. The straightforward approach is to apply the output operator on A_3 and A_4 to get arrival processes A'_3 $(= A_3 \oslash (S_3 \ominus A_4))$ and A'_4 $(= A_4 \oslash (S_3 \ominus A_3))$ at the second queue. However, this approach introduces an artificial dependency between A'_3 and A'_4 because they are derived from S_3, A_3, and A_4.

Key idea 2. When handling competing traffic from the same source, SNC-Meister applies the aggregate operator before the output operator. For example, $(A_3 \oplus A_4) \oslash S_3$.

Using this idea, the aggregate flow to the second queue now does not have any artificial dependencies. Combining the two ideas for our Fig. 6 example, the latency of T_1 is calculated as $Latency(A_1, (S_1 \otimes (S_2 \ominus ((A_3 \oplus A_4) \oslash S_3))) \ominus A_2, 0.99)$. Through these two ideas, SNC-Meister is able to produce much tighter bounds (see Fig. 7) than the straightforward approaches (analyzing one queue at a time: SNC

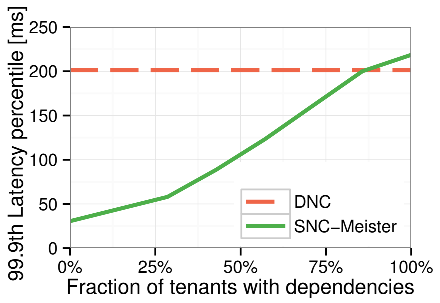


Figure 8. The tail latency calculated using DNC and SNC-Meister as we vary the fraction of tenants that are dependent on each other. In this micro-experiment, seven identical tenants connect from a single client to a single server, and a fraction of them (x-axis) are marked as dependent on each other.

hop-by-hop [5]; applying convolution to a tenant's leftover service process at each queue: SNC convolution [15]). A formal description of SNC-Meister's SNC algorithm can be found in Appendix A.4 in our tech report [54] and the proof of correctness is given in Theorem 7 in Appendix A.5.

4.2 Dependencies between tenants

Since not all tenants are necessarily independent, SNC-Meister also supports users specifying groups of dependent tenants. Dependent tenants are analyzed assuming they can have adversarially correlated bursts. This can be useful, for example, when multiple tenants are part of the same load balancing group.

SNC-Meister incorporates user-specified dependencies by tracking dependency information with arrival and service processes. When aggregating multiple arrival processes (as with key idea 2), SNC-Meister also uses the dependency information to minimize the number of SNC operators that assume dependence (proved in Theorem 6 in Appendix A.5 in our tech report [54]).

Fig. 8 shows the effect of tenant dependency on latency. In this experiment, we take a fraction of the tenants and mark them as dependent on each other. As this fraction varies from 0% (i.e., all independent) to 100% (i.e., all dependent), we see the latency calculated by SNC-Meister increases. This is expected since dependent tenants can have higher latencies due to simultaneous bursts. Nevertheless, SNC-Meister's latency is almost always[4] under DNC since DNC assumes adversarial correlation for all tenants.

4.3 Modeling tenant burstiness

Properly characterizing tail latency entails representing the burstiness and load that each tenant contributes. In SNC-Meister, we use a Markov Modulated Poisson Process (MMPP) as an expressive and analytically tractable model for burstiness. A MMPP can be viewed as a set of phases with different arrival rates and a set of transition probabilities between the phases. A phase with high arrival rate can represent a bursty period, while a phase with low arrival rate can represent a non-bursty period. The MMPP is flexible in that the number of phases can be increased to reflect additional levels of burstiness.

The MMPP parameters for each tenant are determined from its trace. The traces contain the arrival times of requests and their sizes, where the size of a request is the number of bytes being requested. SNC-Meister first determines the number of MMPP phases needed to represent the range of burstiness in the trace. We use an idea similar to [22] where each phase is associated with an arrival rate and covers a range of arrival rates plus or minus two standard deviations. SNC-Meister then maps time periods in the trace to MMPP phases and empirically calculates transition probabilities between the MMPP phases.

While SNC-Meister adapts to the range of burstiness on a per-tenant basis using multiple MMPP phases, the specific number of phases is not critical. In our experimentation, we find a big difference going from a single phase (i.e., a standard Poisson Process) to two phases, but less of a difference with more than two phases. If computation speed is a limiting factor, it is possible to tune SNC-Meister to compute latency faster using fewer phases.

4.4 How SNC-Meister represents SNC in code

In this section, we describe how SNC-Meister represents the core building blocks in SNC, arrival and service processes, as objects in code. We first show how to combine the SNC operators by walking through the example in Fig. 5 and then delve into details on how SNC operators are represented internally.

To analyze the Fig. 5 example, we start with two arrival processes (A_1 and A_2) and two service processes (S_1 and S_2):

```
ArrivalProcess* A1 = new MMPP(traceT1);
ArrivalProcess* A2 = new MMPP(traceT2);
ServiceProcess* S1 = new NetworkLink(bandwidth);
ServiceProcess* S2 = new NetworkLink(bandwidth);
```

We proceed to calculate the latency of T_1, mathematically written $Latency(A_1, (S_1 \otimes S_2) \ominus A_2, 0.99)$. First, the queues are combined to create a service process for the convolution of S_1 and S_2 (i.e., $S_1 \otimes S_2$), which is yet another service process (named `S1x2`):

```
ServiceProcess* S1x2 = new Convolution(S1, S2);
```

Second, T_1's service process is calculated by using the leftover operator on `S1x2` and T_2's arrival process (i.e., $(S_1 \otimes S_2) \ominus A_2$):

```
ServiceProcess* S1x2_A2 = new Leftover(S1x2, A2);
```

Finally, the 99th percentile latency of T_1 is calculated by:

```
double L_A1 = calcLatency(A1, S1x2_A2, 0.99);
```

SNC-Meister is designed to allow the SNC operators to compose any algebraic expression (e.g., $(S_1 \otimes S_2) \ominus A_2$ is `new Leftover(new Convolution(S1, S2), A2)`). This is accomplished by having all of the operators as subclasses of

[4] SNC-Meister can generate higher latencies than DNC when nearly all tenants are dependent because the SNC equations are not tight upper bounds, whereas our DNC analysis is tight.

the ArrivalProcess and ServiceProcess base classes, which have a standardized representation using the $\rho(\theta)$ and $\sigma(\theta)$ form (see Sec. 3.2). To symbolically represent these $\rho(\theta)$ and $\sigma(\theta)$ functions in code, the base classes define pure virtual functions for `rho` and `sigma` that every operator overrides with the equations in Tbl. 1.

Lastly, calculating latency requires optimizing the θ parameter in the Tbl. 1 equations. In particular, the latency equation produces valid upper bounds on latency for every value of $\theta > 0$. Thus, to improve the accuracy of the latency bound, SNC-Meister searches for a θ that produces the minimum latency by sweeping over a range of values at a coarse granularity (e.g., $\theta = 1, 2, 3, ..., 10$) and then progressively narrowing down to finer granularities (e.g., $\theta = 2.1, 2.2, ..., 2.9$).

5. Experimental setup

To demonstrate the effectiveness of SNC-Meister in a realistic environment, we evaluate our implementation of SNC-Meister and of three state-of-the-art systems in a physical testbed running memcached as an example application. This section describes the state-of-the-art systems (Sec. 5.1), our physical testbed (Sec. 5.2), our traces (Sec. 5.3), and the procedure of each experiment (Sec. 5.4).

5.1 State-of-the-art admission control systems

We compare against three state-of-the-art systems: Silo [25], QJump [21], and PriorityMeister [55]. We enhance Silo and QJump to account for end-host queueing delay and to automatically configure tenant parameters (e.g., rate limits).

Silo [25]: Silo offers tenants a worst-case packet latency guarantee under user-specified rate limits. Admission control is performed by verifying that no switch queue in the network overflows using equations from DNC. The maximum packet latency is calculated by adding up all maximum queue sizes along a packet's path.

A limitation with Silo is that choosing a rate limit (i.e., bandwidth and maximum burst size) is left to the user. In the Silo experiments, the burst size is fixed to 1.5KB, and bandwidth is chosen by trial and error. Selecting too high a bandwidth causes few tenants to be admitted. On the other hand, selecting a small bandwidth (e.g., the average bandwidth of a tenant) entails a high end-host queueing delay due to being slowed down by the rate limiting, and compensating for the effect of end-host queueing is left to the user.

Silo++: We extend Silo with an algorithm to automatically choose the minimal bandwidth so that each tenant's request latency SLO can be guaranteed. This is achieved by profiling each tenant's traffic requirements and selecting rate limits using the effective bandwidth approach from DNC theory [29]. We also add support for calculating the end-host queueing delay using DNC, which is used in conjunction with Silo's packet latency guarantee to check whether each tenant can meet its SLO.

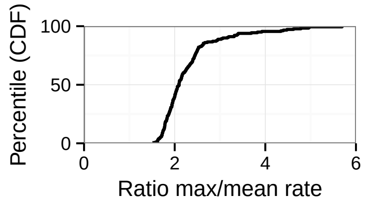


Figure 9. The ratio between maximum and mean request rate is high for many of our traces.

QJump [21]: QJump offers multiple classes of service with different latency-throughput trade-offs. The first class receives the highest priority along with a worst-case latency guarantee using DNC-based equations [35, 36], but is aggressively rate limited. For the other classes, tenants are allowed to send at higher rates, but at lower priorities and without any latency guarantee. There are two limitations in employing the original QJump proposal: 1) users don't know which class to pick because the respective latency guarantee is unknown in advance; and 2) users don't know the end-host queueing delay caused by the rate limiting of each class.

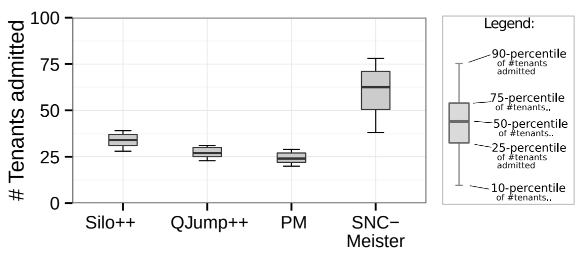
QJump++: We extend QJump with an algorithm to automatically assign tenants to a (near) optimal class. The algorithm iteratively increases the QJump level for tenants that do not meet their SLOs. We add support for calculating the latency for each class as well as the end-host delay, which allows QJump++ to check if a tenant can meet its SLO.

Additionally, we find that instantiating the QJump classes using the QJump equation (Eq. (4) in [21]) severely limits the number of admitted tenants (5x fewer on average). By fixing a set of throughput values independent of the number of tenants, we significantly boost the number of admitted tenants for QJump++.

PriorityMeister (PM) [55]: PriorityMeister uses DNC to offer each tenant a worst-case request latency guarantee based on rate limits that are automatically derived from a tenant's trace. PriorityMeister automatically configures tenant priorities to meet latency SLOs across both network and storage, and in this work, we tailor it to focus only on network latency.

5.2 Physical testbed

Our physical testbed comprises an otherwise idle, 18-machine cluster of Dell PowerEdge 710 machines, configured with two Intel Xeon E5520 processors and 16GB of DRAM. We use the setup shown in Fig. 2. Six machines are dedicated as memcached servers running the most recent version (1.4.25) of memcached. Twelve machines run a set of tenant VM's using the standard kvm package (qemu-kvm-1.0) to provide virtualization support. Each tenant VM runs 64-bit Ubuntu 13.10 and replays a trace using libmemcached. Each physical machine runs 64-bit Ubuntu 12.04, and we use the associated Linux Traffic Control interface without modifications. The top-of-rack switch connecting the machines is a



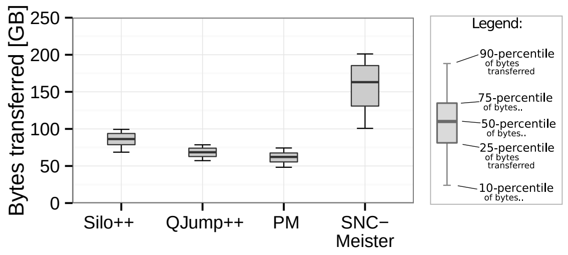


Figure 10. Comparison of three state-of-the-art admission control systems to SNC-Meister for 100 randomized experiments. In each experiment, 180 tenants, each submitting hundreds of thousands of requests, arrive in random order and seek a 99.9% SLO randomly drawn from {10ms, 20ms, 50ms, 100ms}. The left box plot shows that across the 100 experiments, SNC-Meister admits more tenants than state-of-the-art systems. The right plot shows that SNC-Meister achieves a similar improvement with respect to the volume of bytes transferred in each experiment.

Dell PowerConnect 6248 switch, providing 48 1Gbps ports, with DSCP support for 7 levels of priority.

5.3 2015 production traces

Our evaluation uses 180 recent traces captured in 2015 from the datacenter of a large Internet company. The traces capture cache lookup requests issued by a diverse set of Internet applications (e.g., social networks, e-commerce, web, etc.). Each trace contains a list of anonymized requests parameterized by the arrival time and object size being requested, ranging from 1 Byte to 256 KBytes with a mean of 28 KBytes. Each trace is 30 minutes long and contains 100K to 600K requests, with a mean of 320K requests. We find that these traces exhibit significant short-term burstiness, and Fig. 9 shows that the CDF for the ratio of peak to mean request rate ranges from 2 to 6. We also perform standard statistical tests [1, 34] to verify the stationarity and mutual stochastic independence of our traces as required by SNC.

5.4 Experimental procedure

In our experiments, we run up to 180 tenants that replay memcached requests from each tenant's associated trace. For each experiment, tenants arrive to the system one by one in a random order with a 99.9% SLO drawn uniformly randomly from {10ms, 20ms, 50ms, 100ms}. When a tenant arrives, the admission system makes its decision based on the tenant's SLO and the first half of the tenant's trace (15 mins). After the admission decisions for all 180 tenants have been made, each admitted tenant starts a VM to replay the second half of its request trace (15 mins). All tenants replay their traces in an open loop fashion, which properly captures the end-to-end latency and the effects of end-host queueing [41]. All admission systems meet the tenant SLOs, as verified by monitoring the total memcached request latency for every request (i.e., completion time - arrival time in the trace) and checking that the 99.9% latency across 3min time intervals for each tenant is less than its SLO. Thus, we evaluate the performance of the admission control systems under the following two metrics: 1) the number of tenants admitted by each system; and 2) the total volume of bytes transmitted by admitted tenants. Metric 1 indicates how many tenants with tail latency SLOs can be concurrently supported by each system. Metric 2 prevents a system from scoring high on metric 1 by admitting only low-load tenants.

6. Results

In this section, we experimentally evaluate the performance and practicality of SNC-Meister. Sec. 6.1 shows that SNC-Meister is able to support 75% more tail latency SLO tenants than state-of-the-art systems across a large range of experiments. SNC-Meister also transfers 88% more bytes, which shows that SNC-Meister supports a higher network utilization. Sec. 6.2 shows that SNC-Meister's performance is within 7% of an empirical offline solution. Sec. 6.3 demonstrates that SNC-Meister is able to support low-bandwidth tenants with very tight SLOs alongside high-bandwidth tenants. Sec. 6.4 investigates the sensitivity of the SNC latency prediction to the SLO percentile. Sec. 6.5 evaluates the scalability of SNC-Meister and shows that both its computation time and performance scale linearly with the number of tenants.

6.1 SNC-Meister outperforms the state-of-the-art

This section compares SNC-Meister with enhanced versions of the state-of-the-art tail latency SLO systems (described in Sec. 5.1). We run 100 experiments, each with 180 tenants arriving in a random order with random SLOs (described in Sec. 5.4). All four systems, including SNC-Meister, meet the SLOs for all admitted tenants, but differ in how many tenants each system admits.

Fig. 10 shows a box plot of the number of admitted tenants and a box plot of the volume of transferred bytes. We see that the three state-of-the-art systems (Silo++, QJump++, PriorityMeister) perform roughly the same as they draw upon the same underlying DNC mathematics. SNC-Meister achieves a significant improvement over all three systems across all 100 experiments. Silo++ admits slightly more than QJump++ and PriorityMeister, which is due to the effective bandwidth enhancement of Silo++ (see Sec. 5.1). Nevertheless, SNC-Meister outperforms Silo++ by a large margin: of the 100 experiments, the 10-percentile of SNC-Meister is above the 75-percentile of Silo++ for both the

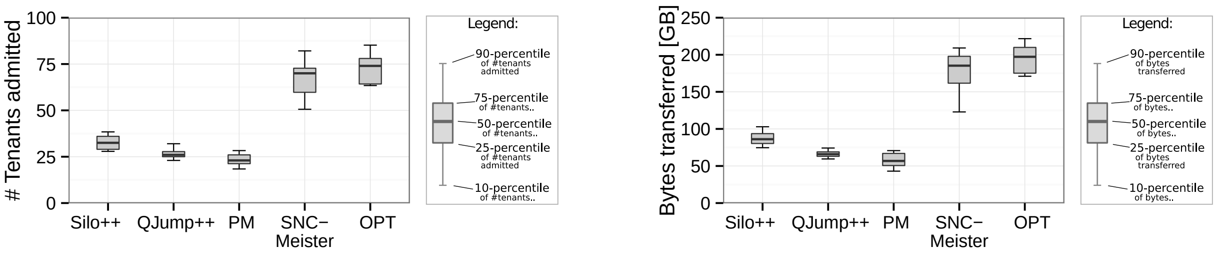


Figure 11. Comparison between state-of-the-art systems, SNC-Meister, and an empirical optimum (OPT) for 10 of the 100 experiments in Fig. 10. The left box plot shows that the number of tenants admitted by SNC-Meister is close to OPT, whereas the other state-of-the-art systems admit less than half of OPT. The right plot shows that SNC-Meister is also close to OPT with respect to the volume of bytes transferred in each experiment.

number of admitted tenants and bytes transferred. The fact that SNC-Meister performs well for both metrics shows that SNC-Meister's improvement is not just due to admitting more low-load tenants, but is due to allowing higher utilization.

6.2 Comparison to empirical optimum

To evaluate how well SNC-Meister compares to an empirical optimum, we determine the maximum number of tenants that can be admitted without SLO violations (labeled OPT) via trial and error experiments. In order to determine the maximum in a reasonable time frame, OPT only considers tenants in the order that they arrive. Thus, OPT is defined as the largest *n* such that the first *n* tenants to arrive meet their SLOs. Determining OPT via trial and error is time consuming and hence we only do this for a random subset[5] of 10 out of the 100 experiments from Sec. 6.1.

Fig. 11 compares the state-of-the-art, SNC-Meister and OPT. We find that SNC-Meister performs almost as well as OPT. Specifically, SNC-Meister is within 7% of OPT in the median, whereas the state-of-the-art admission systems achieve only half of OPT. Thus, SNC-Meister captures most of the statistical multiplexing benefit without needing to run trial and error experiments.

6.3 Small-request tenants

While we have focused on request latency, many related works focus on packet latency and the effects on small requests (i.e., single packet-sized requests).

As SNC-Meister supports prioritization (Sec. 2), we demonstrate that SNC-Meister can also support tenants with small requests and very tight SLOs. Fig. 12 shows the results from an experiment with a set of eleven tenants with single packet requests and tight SLOs (4ms) along with twenty-one other tenants with larger requests and higher SLOs (50ms). Like before, we see that SNC-Meister is able to admit many more tenants than the state-of-the-art systems. Here, PriorityMeister does better than Silo++ and QJump++ since it does not need to reserve a lot of bandwidth for the tight SLOs. Nevertheless, all three of these state-of-the-art systems

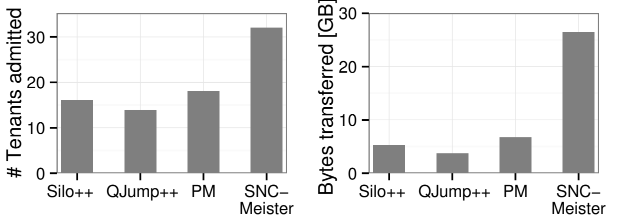


Figure 12. Number of admitted tenants (left) and bytes transferred by admitted tenants (right) in an experiment with two groups of tenants: a set of small-request low-latency (4ms) tenants and a set of large-request higher-latency (50ms) tenants. SNC-Meister admits more tenants and over three times as many bytes as the state-of-the-art systems.

suffer from the drawbacks of DNC and are unable to admit many of the large-request tenants once they've admitted the small-request tenants with tight SLOs. This can particularly be seen in the graph of the number of bytes transferred by admitted tenants. SNC-Meister admits both the small-request tenants as well as many large-request tenants, resulting in a higher network utilization. SNC-Meister is able to do so since, probabilistically, the small-request tenants do not have a large effect on the large-request tenants. The large-request tenants are assigned a lower priority due to their higher SLOs and thus do not affect the small-request tenants.

6.4 Tail latency percentiles

One might wonder how SNC-Meister performs for latency SLOs other than the 99.9th percentile. We address this question by comparing SNC-Meister's latency prediction to the DNC latency prediction used by state-of-the-art systems.

Fig. 13 shows the latency prediction of SNC-Meister and DNC vs. the number of 9s in the SLO percentile, where three 9s represents the 99.9th percentile. SNC-Meister's latency increases with the SLO percentile, as expected, and only exceeds the DNC latency with thirty-three 9s. Thus, SNC-Meister's benefit primarily comes from considering non-100th percentile SLOs. The relative difference between one and three 9s is small compared to the difference between non-100th percentiles (i.e., SNC-Meister) and the 100th percentile (i.e., DNC). Thus, experiments with 90th percentile SLOs

[5] Note that the results from the 10 experiments in Fig. 11 are representative because the state-of-the-art systems and SNC-Meister perform similarly to the 100 experiments in Fig. 10.

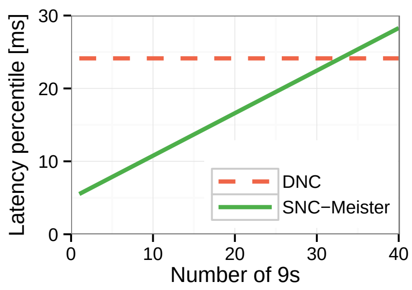


Figure 13. Comparison between the latency predictions of SNC-Meister and DNC for different SLO percentiles. Specifically, the x-axis denotes the number of 9s, where three 9s represents the 99.9th percentile. As expected, the latency using SNC increases with the SLO percentile, but is still superior to DNC even with thirty 9s.

and experiments where tenants have mixtures of 90th, 99th, and 99.9th percentile SLOs show similar results to the case with all tenants having 99.9th percentile SLOs. DNC's worst-case analysis is conservative in accounting for rare events that probabilistically should never occur, whereas SNC-Meister is unaffected by these improbable events for almost any percentile.

6.5 Scalability

In this section, we study the scalability of SNC-Meister. Fig. 14 shows the runtime for computing latency as a function of the number of tenants. We see that SNC-Meister's runtime scales linearly with the number of tenants, which is ideal since each tenant's latency is calculated one by one. This is promising, given that the computation is currently single threaded, and the analysis of each of the tenants can easily be parallelized.

Fig. 15 shows how the number of admitted tenants scales with the size of the cluster. We use the same setup with 180 tenants, but replicate the tenants and number of machines by a scaling factor (x-axis) to show the effect of larger scale. The order and assignment of tenant VMs to data servers is random as before. As expected, the performance of SNC-Meister scales linearly with the size of the cluster.

7. Related work

SNC-Meister addresses meeting tail latency SLOs, which is an active research area with a rich literature. The related work can be divided into four major lines of work. First, there is a body of work that ensures that tail latency SLOs are met based on worst-case latency bounds; unfortunately these works are unable to achieve high degrees of multi-tenancy due to the conservative nature of worst-case analysis. To overcome these limitations, theoreticians have developed a second line of work that provides probabilistic tail latency bounds via Stochastic Network Calculus (SNC). These works are entirely theoretical and have never been implemented for any computer system. Third, there is a body of work that proposes techniques for significantly reducing the tail latency;

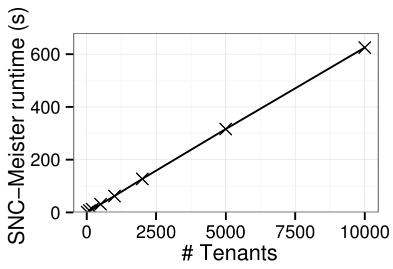


Figure 14. SNC-Meister's runtime scales linearly with the number of tenants.

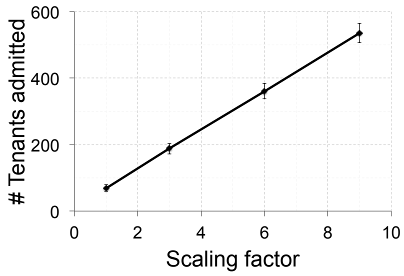


Figure 15. The number of admitted tenants in SNC-Meister scales linearly with the cluster size.

ensuring that request latency SLOs are met is not within the scope of that work. Fourth, there are some recent systems that try to meet SLOs based on measured latency; unfortunately, these works aren't suited for admission control and don't cope well with bursty tenants.

Guaranteed latency systems

There are three recent state-of-the-art systems that provide SLO guarantees: Silo [25], QJump [21], and PriorityMeister [55] (described in Sec. 5.1 and listed in the top half of Tbl. 2). All three systems are designed for worst-case latency guarantees. For tenants seeking a lower percentile tail guarantee (e.g., a guarantee on the 99.9*th* percentile of latency), these systems are overly conservative in their admission decisions: they admit half the number of tenants as compared to SNC-Meister (see Sec. 6.1).

Besides these recent proposals, there has been a long history of DNC-based worst-case latency admission control algorithms in the context of Internet QoS [14, 28, 32, 44, 49]. These older proposals are not tailored to datacenter applications, and also suffer from the conservative nature of worst-case latency guarantees.

Stochastic Network Calculus (SNC)

The modern SNC theory evolved as an alternative to the DNC theory to capture statistical multiplexing gains and enable accurate guarantees for any latency percentile [5–11, 15–17, 20, 27, 31, 38, 39, 42, 52]. However, all of this work is in theory, and we are not aware of any implementations that use SNC in computer systems. The only practical applications of SNC are in the modeling of critical infrastructures such as avionic networks [40] and the power grid [19, 48], which support the robustness of SNC theory.

			tail latency SLO	multi-tenancy	parameter configuration
guaranteeing tail latency	SNC-based	SNC-Meister	Any (e.g., 99.9th)	high	automated
	worst-case admission control	Silo [25]	100th	low	manual
		QJump [21]	100th	low	manual
		PriorityMeister [55]	100th	low	automated
reducing tail latency	datacenter scheduling	pHost [18]	no	n/a	manual
		Fastpass [37]	no	n/a	manual
		pFabric [4]	no	n/a	manual
	congestion control	D2TCP [45]	no	n/a	n/a
		DCTCP [2]	no	n/a	n/a
	other	[12, 24, 43, 46, 51, 53]	no	n/a	n/a

Table 2. While many systems aim to reduce tail latency (bottom half of table), few provide tail latency guarantees (top half).

Reducing tail latencies

There are many systems that demonstrate how to reduce tail latency (listed in the bottom half of Tbl. 2). Datacenter schedulers, such as pHost [18], Fastpass [37], and pFabric [4], improve tail latency by bringing near-optimal schedulers (such as earliest-deadline first) to the datacenter. These approaches can also shift queueing from within the network to the end-hosts, which greatly reduces tail packet latency and the latency of short requests. These approaches, however, are not designed to ensure tail latency SLO compliance.

Latency-aware congestion control algorithms, such as D2TCP [45] and DCTCP [2], aggressively scale down sending rates and prioritize flows with deadlines. These approaches react to congestion, which can lead to SLO violations in the face of bursty traffic [4, 25]. HULL [3] keeps tail latencies low by controlling the network utilization through rate limiting, but can still experience SLO violations [21].

Other techniques for reducing tail latency include issuing redundant requests [12, 24, 46], latency-adaptive machine selection [43, 51], and latency-adaptive load balancing [53]. While these techniques can reduce the tail latency, they are not designed for meeting tail latency SLOs.

Measurement-based approaches

Several recent works measure the latency and adapt the system to try to meet tail latency SLOs [30, 47]. Unfortunately, these approaches aren't suited for admission control where admission decisions cannot be made dynamically. Furthermore, prior work has shown that reactive approaches struggle with bursty tenants and often do not meet their SLOs [55].

The recent Cerebro [26] work uses measurements to characterize the latency of requests composed of multiple sub-requests. Unlike SNC-Meister, Cerebro is not designed to account for the interaction between multiple tenants, which is a primary benefit of SNC.

8. Conclusion and discussion

SNC-Meister is a new system for meeting tail request latency SLOs while achieving higher multi-tenancy than the state-of-the-art. In experiments with production traces on a physical implementation testbed, we show that SNC-Meister can admit two to three times as many tenants as the state-of-the-art while meeting tail latency SLOs. SNC-Meister benefits from applying a new probabilistic theory called Stochastic Network Calculus (SNC) to calculate tail latencies, while prior systems use the conservative worst-case Deterministic Network Calculus (DNC) theory.

As SNC is a new theory, there are many challenges in bringing it to practice, and there is much room for further research. One challenge we identify is the important role of the order in which SNC operators are applied – a fundamental problem that was not previously considered in SNC literature. Our novel algorithm for analyzing networks with SNC makes a significant step forward in making SNC a practical tool. We also add support in SNC-Meister for dependencies between subsets of tenants, which addresses a practical issue that is generally ignored in SNC theory. Nevertheless, it is still an open question on how to better apply SNC techniques to get tighter bounds.

While this work focuses on the admission control problem, the ideas behind SNC-Meister and SNC are applicable to many applications beyond admission control. One such example is the datacenter provisioning problem. By being able to analyze tenant behavior and compute tail latency, SNC-Meister could be extended to deciding when (and how many) more resources are required for meeting tail latency SLOs. Similarly, these techniques could apply to tenant placement problems: SNC could be used to identify bottlenecks and make placement decisions in a tail latency aware fashion. Also, the mathematics behind SNC is applicable to other resources beyond networks such as storage bandwidth. We thus believe that the SNC theory can develop into a practical tool for working with tail latency.

Acknowledgments

This research is supported in part by Intel as part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), by a Google Faculty Research Award 2015/16, and by the National Science Foundation under awards CMMI-1538204, CMMI-1334194, CSR-1116282, and XPS-1629444. We also thank the member companies of the PDL Consortium for their interest, insights, feedback, and support.

References

[1] A. Agresti. Building and applying logistic regression models. *Categorical Data Analysis, Second Edition*, pages 211–266, 2007.

[2] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp DCTCP. In *ACM SIGCOMM*, pages 63–74, 2011.

[3] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *USENIX NSDI*, pages 19–19, 2012.

[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *ACM SIGCOMM*, pages 435–446, 2013.

[5] M. A. Beck and J. Schmitt. The disco stochastic network calculator version 1.0: when waiting comes to an end. In *Valuetools*, pages 282–285, 2013.

[6] A. Burchard, J. Liebeherr, and F. Ciucu. On superlinear scaling of network delays. *IEEE/ACM Transactions on Networking (TON)*, 19(4):1043–1056, 2011.

[7] C.-S. Chang. Stability, queue length, and delay of deterministic and stochastic queueing networks. *IEEE Transactions on Automatic Control*, 39(5):913–931, 1994.

[8] C.-S. Chang. *Performance guarantees in communication networks*. Springer Science & Business Media, 2000.

[9] F. Ciucu, A. Burchard, and J. Liebeherr. A network service curve approach for the stochastic analysis of networks. In *ACM SIGMETRICS*, pages 279–290, 2005.

[10] F. Ciucu and J. Schmitt. Perspectives on network calculus: No free lunch, but still good value. In *ACM SIGCOMM*, pages 311–322, 2012.

[11] R. Cruz. Quality of service management in integrated services networks. In *Proceedings of the 1st Semi-Annual Research Review, CWC*, 1996.

[12] J. Dean and L. A. Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, Feb. 2013.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *ACM SOSP*, pages 205–220, 2007.

[14] D. Ferrari and D. C. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE JSAC*, 8(3):368–379, 1990.

[15] M. Fidler. An end-to-end probabilistic network calculus with moment generating functions. In *IEEE International Workshop on Quality of Service (IWQoS)*, pages 261–270, 2006.

[16] M. Fidler and A. Rizk. A guide to the stochastic network calculus. *IEEE Communications Surveys & Tutorials*, 17(1):92–105, 2015.

[17] V. Firoiu, J.-Y. Le Boudec, D. Towsley, and Z.-L. Zhang. Theories and models for internet quality of service. *Proceedings of the IEEE*, 90(9):1565–1591, 2002.

[18] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker. phost: Distributed near-optimal datacenter transport over commodity network fabric. In *ACM CoNEXT*, 2015.

[19] Y. Ghiassi-Farrokhfal, S. Keshav, and C. Rosenberg. Toward a realistic performance analysis of storage systems in smart grids. *IEEE Transactions on Smart Grid*, 6(1):402–410, 2015.

[20] Y. Ghiassi-Farrokhfal and J. Liebeherr. Output characterization of constant bit rate traffic in fifo networks. *IEEE Communications Letters*, 13(8):618–620, 2009.

[21] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues don't matter when you can jump them! In *USENIX NSDI*, 2015.

[22] D. P. Heyman and D. Lucantoni. Modeling multiple ip traffic streams with rate limits. *Networking, IEEE/ACM Transactions on*, 11(6):948–958, 2003.

[23] S. Islam, S. Venugopal, and A. Liu. Evaluating the impact of fine-scale burstiness on cloud elasticity. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 250–261, New York, NY, USA, 2015. ACM.

[24] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan. Speeding up distributed request-response workflows. In *ACM SIGCOMM*, pages 219–230, 2013.

[25] K. Jang, J. Sherry, H. Ballani, and T. Moncaster. Silo: Predictable message latency in the cloud. In *ACM SIGCOMM*, pages 435–448. ACM, 2015.

[26] H. Jayathilaka, C. Krintz, and R. Wolski. Response time service level agreements for cloud-hosted web applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 315–328, New York, NY, USA, 2015. ACM.

[27] J. Kurose. On computing per-session performance bounds in high-speed multi-hop computer networks. In *ACM SIGMETRICS*, 1992.

[28] J.-Y. Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Transactions on Information Theory*, 44(3):1087–1096, 1998.

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

[30] N. Li, H. Jiang, D. Feng, and Z. Shi. Pslo: Enforcing the xth percentile latency and throughput slos for consolidated vm storage. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 28:1–28:14, New York, NY, USA, 2016. ACM.

[31] J. Liebeherr, Y. Ghiassi-Farrokhfal, and A. Burchard. On the impact of link scheduling on end-to-end delays in large networks. *IEEE JSAC*, 29(5):1009–1020, 2011.

[32] J. Liebeherr, D. E. Wrege, and D. Ferrari. Exact admission control for networks with a bounded delay service. *IEEE/ACM Transactions on Networking (TON)*, 4(6):885–901, 1996.

[33] J. C. Mogul and R. R. Kompella. Inferring the network latency requirements of cloud tenants. In *Usenix HotOS XV*, 2015.

[34] G. Nason. A test for second-order stationarity and approximate confidence intervals for localized autocovariances for locally stationary time series. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(5):879–904, 2013.

[35] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

[36] A. K. Parekh and R. G. Gallagher. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM Transactions on Networking*, 2(2):137–150, 1994.

[37] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized zero-queue datacenter network. In *ACM SIGCOMM*, pages 307–318, 2014.

[38] F. Poloczek and F. Ciucu. Scheduling analysis with martingales. *Performance Evaluation*, 79:56–72, 2014.

[39] J.-y. Qiu and E. W. Knightly. Inter-class resource sharing using statistical service envelopes. In *IEEE INFOCOM*, volume 3, pages 1404–1411, 1999.

[40] J.-L. Scharbarg, F. Ridouard, and C. Fraboul. A probabilistic analysis of end-to-end delays on an afdx avionic network. *IEEE Transactions on Industrial Informatics*, 5(1):38–49, 2009.

[41] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: A cautionary tale. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 18–18, Berkeley, CA, USA, 2006. USENIX Association.

[42] D. Starobinski and M. Sidi. Stochastically bounded burstiness for communication networks. In *IEEE INFOCOM*, volume 1, pages 36–42, 1999.

[43] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *USENIX NSDI*, 2015.

[44] G. Urvoy-Keller, G. Hébuterne, and Y. Dallery. Traffic engineering in a multipoint-to-point network. *IEEE JSAC*, 20(4):834–849, 2002.

[45] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *ACM SIGCOMM*, pages 115–126, 2012.

[46] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. In *ACM CoNEXT*, pages 283–294, 2013.

[47] A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica. Cake: enabling high-level slos on shared storage systems. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 14:1–14:14, New York, NY, USA, 2012. ACM.

[48] K. Wang, F. Ciucu, C. Lin, and S. H. Low. A stochastic power network calculus for integrating renewable energy sources into the power grid. *IEEE JSAC*, 30(6):1037–1048, 2012.

[49] S. Wang, D. Xuan, R. Bettati, and W. Zhao. Providing absolute differentiated services for real-time applications in static-priority scheduling networks. *IEEE/ACM Transactions on Networking*, 12(2):326–339, 2004.

[50] Y. Xu, M. Bailey, B. Noble, and F. Jahanian. Small is better: Avoiding latency traps in virtualized data centers. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 7:1–7:16, New York, NY, USA, 2013. ACM.

[51] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey. Bobtail: Avoiding long tails in the cloud. In *USENIX NSDI*, pages 329–342, 2013.

[52] O. Yaron and M. Sidi. Performance and stability of communication networks via robust exponential bounds. *IEEE/ACM Transactions on Networking*, 1(3):372–385, 1993.

[53] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. Detail: reducing the flow completion time tail in datacenter networks. In *ACM SIGCOMM*, pages 139–150, 2012.

[54] T. Zhu, D. S. Berger, and M. Harchol-Balter. SNC-Meister: Admitting More Tenants with Tail Latency SLOs. Technical Report CMU-CS-16-113, School of Computer Science, Carnegie Mellon University, May 2016. available at `http://reports-archive.adm.cs.cmu.edu/anon/2016/CMU-CS-16-113.pdf`.

[55] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger. PriorityMeister: Tail Latency QoS for Shared Networked Storage. In *ACM SOCC*, pages 29:1–29:14, New York, NY, USA, 2014. ACM.