The DISCO Network Calculator – A Toolbox for Worst Case Analysis

Jens B. Schmitt and Frank A. Zdarsky
disco | Distributed Computer Systems Lab, University of Kaiserslautern, Germany
{jschmitt,zdarsky}@informatik.uni-kl.de

ABSTRACT

In this paper we describe the design, implementation, and analytical background of the DISCO Network Calculator. The DISCO Network Calculator is an open-source toolbox written in Java^{TM} which we developed for worst-case analyses based on network calculus. To our knowledge it is the first of its kind. It allows to do network analyses regarding performance characteristics such as delay and backlog bounds for piecewise linear arrival and service curves. We illustrate the tool's usefulness by two comprehensive example applications.

Categories and Subject Descriptors

C.2 [**Computer Systems Organization**]: Computer Communication Networks; C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling Techniques*

Keywords

Network calculus, deterministic guarantees, tool support.

1. INTRODUCTION

1.1 Motivation

Bounding performance characteristics in communication networks is a fundamental issue and has important applications in network design and control. Network calculus, which is a set of relatively new developments that provide deep insights into flow problems encountered in networks of queues [11], provides a deterministic framework for worst-case analysis of delay and backlog bounds. Its mathematical foundation lies in min-plus algebra [2]. Network calculus has found numerous applications, most prominently in the Internet's Quality of Service (QoS) proposals IntServ and DiffServ, but also in other scenarios as, for example, wireless sensor networks [15], switched Ethernets [17], or even Systems-on-a-Chip (SoC) [3]. To mention a further, somewhat different usage, it has been applied in order to speed-up simulations as well [9]. Hence, besides queueing theory it has established as a valuable methodology to model networks of queues. Yet, up to now there is no publicly available tool support for network calculus-based analyses. We believe that such a tool support is a crucial ingredient for the further flourishment of the promising network calculus methodology. So, as a first step we have implemented the DISCO Network Calculator, a toolbox consisting of min-plus algebraic base operations as well as comprehensive algorithms for different ways of analysing complete networks. The DISCO Network Calculator class library is written in Java^{TM} and is publicly available[1]. We hope it will be of use to other researchers interested in network calculus, and that it will be the basis for even more powerful network calculus tools in the future.

Valuetools '06, October 11-13, 2006, Pisa,Italy

1.2 Network Calculus Background

Network calculus is a min-plus system theory for deterministic queuing systems. The important concept of *minimum service curve* was introduced in [6], [13], [4], [10], [1] and the concept of *maximum service curve* in [7]. As network calculus is built around the notion of cumulative functions for input and output flows of data, the set of real-valued, non-negative, and wide-sense increasing functions passing through the origin plays a major role:

$$\mathcal{F} = \left\{ f : \mathbb{R}^+ \to \mathbb{R}^+, \forall t \geq s : f(t) \geq f(s), f(0) = 0 \right\}$$

In particular, the input function $F(t)$ and the output function $F^*(t)$ are $\in \mathcal{F}$. They cumulatively count the number of bits that are input to, respectively output from, a system $\mathcal{S}$. Throughout the paper, we assume in- and output functions to be continuous in time and space.

DEFINITION 1. *(Min-plus Convolution and Deconvolution) The min-plus convolution and deconvolution of two functions f and g are defined to be*

$$(f \otimes g)(t) = \begin{cases} \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\} & t \geq 0 \\ 0 & t < 0 \end{cases}$$

$$(f \oslash g)(t) = \sup_{u \geq 0} \{f(t+u) - g(u)\}$$

Let us now turn to the performance characteristics of flows which can be bounded by network calculus means:

DEFINITION 2. *(Backlog and Delay) Assume a flow with input function F that traverses a system $\mathcal{S}$ resulting in the*

[1] http://disco.informatik.uni-kl.de/content/Downloads

output function F^*. The backlog of the flow at time t is defined as

$$b(t) = F(t) - F^*(t)$$

Assuming first-in-first-out delivery, the delay for a bit input at time t is defined as

$$d(t) = \inf\{\tau \geq 0 : F(t) \leq F^*(t+\tau)\}$$

Now the arrival and server processes specified by input and output functions are bounded based on the central network calculus concepts of arrival and service curves:

DEFINITION 3. *(Arrival Curve) Given a flow with input function F a function $\alpha \in \mathcal{F}$ is an arrival curve for F if and only if*

$$\forall t, s \geq 0, s \leq t : F(t) - F(t-s) \leq \alpha(s)$$

$$\Leftrightarrow F \leq F \otimes \alpha \Leftrightarrow \alpha \geq F \oslash F$$

Note that an arrival curve which is not sub-additive can be improved by its sub-additive closure [11]. As a further remark, remember that any concave function is sub-additive. The next corollary is a simple implication of the definition of arrival curves and the isotonicity of the min-plus convolution.

COROLLARY 1. *(Knowledge of Several Arrival Curves) If α_1 and α_2 are arrival curves for a flow F then $\alpha_1 \otimes \alpha_2$ is also an arrival curve for F.*

DEFINITION 4. *(Minimum and Maximum Service Curve) If the service provided by a system $\mathcal{S}$ for a given input function F results in an output function F^* we say that $\mathcal{S}$ offers a minimum service curve β respectively a maximum service curve $\bar{\beta}$ if and only if*

$$F^* \geq F \otimes \beta \quad \textit{respectively} \quad F^* \leq F \otimes \bar{\beta}$$

There is also a *strict* (minimum) service curve definition which is less general than the minimum service curve but is sometimes required to allow for an analysis.

DEFINITION 5. *(Strict Minimum Service Curve) Let $\beta \in \mathcal{F}$. We say that system $\mathcal{S}$ offers a* strict *minimum service curve β to a flow if, during any backlogged period of duration u, i.e. for any t for which $\forall s, 0 \leq s < u : b(t-s) > 0$, the output of the flow is at least equal to $\beta(u)$.*

Using those concepts it is possible to derive basic performance bounds on backlog, delay, and output:

THEOREM 1. *(Performance Bounds) Consider a system $\mathcal{S}$ that offers a minimum service curve β and a maximum service curve $\bar{\beta}$. Assume a flow F traversing the system has an arrival curve α. Then we obtain the following performance bounds:*
Backlog: $\forall t : b(t) \leq (\alpha \oslash \beta)(0) =: v(\alpha, \beta)$
Delay: $\forall t : d(t) \leq \inf\{t \geq 0 : (\alpha \oslash \beta)(-t) \leq 0\} =: h(\alpha, \beta)$
Output (arrival curve α^* for F^*): $\alpha^* \leq (\alpha \otimes \bar{\beta}) \oslash \beta$

One of the strongest results of network calculus (albeit being a simple consequence of the associativity of $\otimes$) is the concatenation theorem that enables us to collapse tandems of systems into a single system:

THEOREM 2. *(Concatenation Theorem for Tandem Systems) Consider a flow that traverses a tandem of systems $\mathcal{S}_1$ and $\mathcal{S}_2$. Assume that $\mathcal{S}_i$ offers a minimum service curve β_i and a maximum service curve $\bar{\beta}_i$, $i = 1, 2$ to the flow. Then the concatenation of the two systems offers a minimum service curve $\beta_1 \otimes \beta_2$ and a maximum service $\bar{\beta}_1 \otimes \bar{\beta}_2$ to the flow.*

So far we have only covered the tandem network case, the next result factors in the existence of other interfering flows.

THEOREM 3. *(Blind Multiplexing Nodal Service Curve) Consider a node blindly multiplexing two flows 1 and 2. Assume that the node guarantees a* strict *minimum service curve β and a maximum service $\bar{\beta}$ to the aggregate of the two flows. Assume that flow 2 has α_2 as an arrival curve. Then*

$$\beta_1 = [\beta - \alpha_2]^+$$

is a minimum service curve for flow 1 if $\beta_1 \in \mathcal{F}$. $\bar{\beta}$ remains the maximum service curve also for flow 1 alone. Here, the $[.]^+$ operator is defined as $[x]^+ = x \vee 0$, where $\vee$ denotes the maximum operator.

Theorem 3 makes no assumptions on the way flows are multiplexed at the node (blind multiplexing). There is also a known network calculus result telling us how to model the service curve of a *FIFO* node under cross-traffic from other flows:

THEOREM 4. *(FIFO Nodal Service Curve) Consider a FIFO node multiplexing two flows 1 and 2. Assume that the node guarantees a minimum service curve β to the aggregate of the two flows. Assume that flow 2 has α_2 as an arrival curve. Define the family of functions*

$$\beta_\theta^1(t) = [\beta(t) - \alpha_2(t-\theta)]^+ 1_{\{t>\theta\}}$$

Then for any $\theta \geq 0$ flow 1 is guaranteed a service β_θ^1 (if β_θ^1 is wide-sense increasing). Here, the indicator function $1_{\{cond\}}$ is defined as 1 if cond is true and 0 otherwise.

There is very profound work on aggregate multiplexing in *general networks*, the most well-known result being probably given in [5]. The delay bounds derived in these works are based on a FIFO multiplexing assumption. Here we only extract the sufficient existence condition for a delay bound which is valid under FIFO as well as under blind multiplexing:

THEOREM 5. *(Aggregate Multiplexing in General Networks) Assume a general network of diameter h with any node n providing a sustained rate R_n and node n having a bound*

on the rate of all incoming traffic C_n. Let us define the utilization for node n as $u_n = \frac{\sum_{n \in f} r_f}{R_n}$, with r_f denoting the sustained rate of a flow f and $n \in f$ meaning that flow f traverses node n. Then a delay bound exists if

$$\bigvee_n u_n < \bigwedge_n \sigma_{\{C_n > R_n\}} \left(\frac{C_n}{(C_n - R_n)(h-1) + R_n} \right)$$

with $\sigma_{\{cond\}}(x)$ equals x if cond is true and 1 otherwise. $\wedge$ denotes the minimum operator.

Since even moderately sized general networks are only stable for very low utilizations, it is often more suitable to restrict network calculus analyses to so-called *feed-forward networks*.

DEFINITION 6. *(Feed-Forward Network) A network is feed-forward if it is possible to find a numbering of its links such that for any flow through the network the numbering of its traversed links is an increasing sequence.*

A sink-tree is a simple example of a feed-forward network. It is well-known and easy to show that feed-forward networks are stable, i.e. a finite delay bound exists, for any utilization ≤ 1 [11]. While many networks are obviously not feed-forward, a considerable number of important instances like switched networks, wireless sensor networks, or MPLS networks are. Furthermore, there are very effective techniques to make a general network feed-forward. One of the advanced techniques (in comparison to a simple spanning tree) is the so-called turn-prohibition algorithm [18].

2. NETWORK CALCULUS WITH PIECEWISE LINEAR CURVES

To be able to compute performance bounds in networks of queues, we need to apply basic network calculus operations like min-plus convolution and deconvolution on the arrival and service curves we encounter in a given scenario. A class of functions which is both tractable as well as general enough to express all common cases is that of piecewise linear functions. Hence, in this section we present basic network calculus operations for general piecewise linear arrival and service curves, which are implemented in the DISCO Network Calculator. On these operations the network analysis will be built in Section 3.

2.1 A Catalog of Useful Functions

In the definition of piecewise linear arrival and service curves the following catalog of functions from $\mathcal{F}$ is helpful:

DEFINITION 7. *(Auxiliary functions from $\mathcal{F}$)*

Burst delay functions: $\delta_T(t) = \begin{cases} +\infty & t > T \\ 0 & t \leq T \end{cases}$

Affine function (token bucket): $\gamma_{r,b}(t) = \begin{cases} rt + b & t > 0 \\ 0 & t \leq 0 \end{cases}$

Rate-latency function: $\beta_{R,T}(t) = \begin{cases} R(t-T) & t > T \\ 0 & t \leq T \end{cases}$

From these functions general piecewise linear functions can be constructed using the $\bigvee$ and $\bigwedge$ operators, as we will encounter in the next subsection. Note that, as mentioned in [11], it applies that $\forall f \in \mathcal{F} : (f \otimes \delta_T)(t) = f(t - T)$. Hence, a convolution with the burst delay function δ_T results in a shift along the x-axis according to the value of T.

2.2 Choice of Arrival and Service Curves

As already discussed, focusing on piecewise linear functions as arrival and service curves is no practical restriction, since this is still general enough to cover virtually any realistic case of traffic and server models, or at least approximate them closely.

2.2.1 Arrival Curve

Let us assume a piecewise linear concave arrival curve:

$$\alpha = \bigwedge_{i=1}^{n} \gamma_{r_i, b_i}$$

On the one hand it is possible to already use such a function as a traffic source description in order to be able to closely approximate a source's worst case behaviour. On the other hand, looking at the network analysis we cannot avoid to model arrival curves *inside* the network as general piecewise linear concave functions when we factor in maximum service curves. This is due to the fact that more and more complex piecewise linear functions result from the addition of multiple flows induced by their multiplexing. To assume concavity is no major restriction. As discussed in [11], non-concave functions (unless they are not sub-additive, to be accurate) can be improved by pure knowledge of themselves, thus they cannot be tight.

2.2.2 Minimum Service Curve

Let us assume a piecewise linear convex minimum service curve:

$$\beta = \bigvee_{j=1}^{m} \beta_{R_j, T_j}$$

A piecewise linear convex minimum service curve results from deriving the service curve for a flow of interest at a node that multiplexes this flow with other flows which, as an aggregate, have a piecewise linear concave arrival curve. Again, it might also be useful to model a node's service curve using several linear segments. To assume convexity is not a major restriction, as it might for instance apply if a node also has other duties. Though, in contrast to the arrival curve which is not sensible for non-concave functions (non-sub-additive to be exact), there are potentially sensible non-convex service curves as for example presented in [19], [14], and [12].

2.2.3 Maximum Service Curve

Let us assume a piecewise linear *almost concave* maximum service curve:

$$\bar{\beta} = \left(\bigwedge_{k=1}^{l} \gamma_{\tilde{r}_k, \tilde{b}_k} \right) \otimes \delta_L$$

By *almost concave* we mean that the curve is only concave for values of $t > L$ and is 0 for values $t \leq L$ (this kind of curves has also been coined pseudo-affine in [12]). If $L > 0$ this models a node that has a certain minimum latency.

The piecewise concavity models again the fact that a node may also have other duties. The concavity of the maximum service curve results from taking a best case perspective, unlike the worst case perspective of the minimum service curve.

Before discussing the required network calculus operations on these curves, we first make an observation on how the output bound of Theorem 1 can be improved under maximum service curves as they are assumed here.

2.3 Tightening the Output Bound

Realising that the maximum service curve induces arrival constraints on the output we can improve the output bound from Theorem 1 by the following lemma (we state it slightly more general than necessary for our purposes—concave instead of just piecewise linear concave; a proof of this result and the following lemmas and theorems can be found in [16]):

LEMMA 1. *(Improvement of Output Bound) Under the same assumptions as in Theorem 1, let additionally L be such that $L = \sup\{t \geq 0 : \bar{\beta}(t) = 0\}$. L can be regarded as minimum/fixed delay for the system. If $\bar{\beta} \otimes \delta_{-L}$ is concave then the output bound for any flow F can be calculated as*

$$\alpha^* = \left(\left(\alpha \otimes \bar{\beta}\right) \oslash \beta\right) \otimes \left(\bar{\beta} \otimes \delta_{-L}\right)$$

2.4 Network Calculus Operations

We now have everything set to examine the network calculus operations we require as basic building blocks for the network analysis algorithms implemented in the DISCO Network Calculator. First of all, computing output bounds of the flows in the network is an important operation, as it allows to separate flows of interest from interfering flows by using Theorem 3 or Theorem 4, respectively the results we present in Theorem 6. Lemma 1 gives us the general rule to compute the output bound:

$$\alpha^* = \left(\left(\alpha \otimes \bar{\beta}\right) \oslash \beta\right) \otimes \left(\bar{\beta} \otimes \delta_{-L}\right)$$

Hence, starting from the innermost operation, the convolution of the arrival curve and the maximum service must be determined first:

$$\alpha \otimes \bar{\beta} = \alpha \otimes \left(\left(\bigwedge_{k=1}^{l} \gamma_{\tilde{r}_k, \tilde{b}_k}\right) \otimes \delta_L\right) = \left(\alpha \otimes \bigwedge_{k=1}^{l} \gamma_{\tilde{r}_k, \tilde{b}_k}\right) \otimes \delta_L$$

$$= \left(\alpha \wedge \left(\bar{\beta} \otimes \delta_{-L}\right)\right) \otimes \delta_L =: \sigma$$

Here we used first the associativity of $\otimes$, then the fact that $\otimes$ is equivalent to $\wedge$ for concave functions (see [11]). While σ might look complex, it is easy to compute for the DISCO Network Calculator: first shift the maximum service curve to the left by its latency, then take the minimum with the concave arrival curve and shift the result to the right by the latency of the maximum service curve. Next, the deconvolution of the resulting almost concave function σ with the minimum service curve is calculated as:

$$\sigma \oslash \beta = (\sigma \otimes \delta_{-X}) - \beta(X) =: \zeta$$

where $X = \sup_{t \geq 0} \left\{ \frac{d\sigma}{dt}(t) \geq \frac{d\beta}{dt}(t) \right\}$. X must be at one of the inflexion points of σ and β, such that an intelligent search could be implemented in the DISCO Network Calculator to efficiently find X. When X has been determined, the operations are again simple for the DISCO Network Calculator: shift σ by X to the left and push the result down by $\beta(X)$. Note that the resulting ζ is concave since always $X \geq L$. So finally ζ is convolved with the right-shifted maximum service curve (which is also concave):

$$\zeta \otimes \left(\bar{\beta} \otimes \delta_{-L}\right) = \zeta \wedge \left(\bar{\beta} \otimes \delta_{-L}\right)$$

This convolution is again simply computed by the DISCO Network Calculator as the minimum between the concave ζ and the maximum service curve shifted to the left by its latency. So, in terms of the initial curves we receive as an overall result for the output bound:

$$\alpha^* = \left(\left(\left(\alpha \wedge \left(\bar{\beta} \otimes \delta_{-L}\right)\right) \otimes \delta_{L+X}\right) - \beta(X)\right) \wedge \left(\bar{\beta} \otimes \delta_{-L}\right)$$

Equipped with this, a single node analysis can be accomplished. However, another basic operation consists of using the concatenation theorem to collapse systems in sequence into one large system by calculating their min-plus convolution. So the convolution of piecewise linear minimum and maximum service curves must be computed by the DISCO Network Calculator. For the minimum service curves we can draw upon a rule from [11], which says that piecewise linear convex functions $\in \mathcal{F}$ can be min-plus convolved by putting together their linear segments in the order of increasing slopes. For maximum service curves the next lemma gives us the computation of the min-plus convolution of two almost concave functions. Again, it consists of simple shifts and could thus easily and efficiently be implemented in the DISCO Network Calculator:

LEMMA 2. *(Min-Plus Convolution of Almost Concave Functions) Consider two almost concave piecewise linear functions $\bar{\beta}_1$ and $\bar{\beta}_2$ with latencies L_1 and L_2, then their min-plus convolution can be computed as follows*

$$\bar{\beta}_1 \otimes \bar{\beta}_2 = \left(\bar{\beta}_1 \otimes \delta_{-L_1}\right) \wedge \left(\bar{\beta}_2 \otimes \delta_{-L_2}\right) \otimes \delta_{L_1+L_2}$$

3. NETWORK ANALYSIS ALGORITHMS

In this section, we first discuss different alternatives to calculate end-to-end service curves for certain flows of interest. Next, we present different network analysis methods, of which none is strictly dominating the others, but their relative performance depends upon the given network scenario. Thus we implemented all these options for different service curves and network analysis algorithms in the DISCO Network Calculator.

3.1 End-to-End Service Curve Calculation

Before we come to the actual algorithms for network analysis, applying the basic operations that were just presented, we investigate different alternatives to derive the end-to-end service curve for a flow of interest through a network of queues. For the ease of presentation we confine the discussion to blind multiplexing nodes, although we realised the FIFO multiplexing case in the DISCO Network Calculator as well (wherever possible). One possibility is to derive the end-to-end service curve based on the concatenation theorem and the result for single node blind multiplexing in Theorem 3. This evident method is mentioned in [11]. For

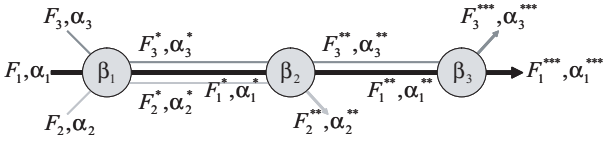

Figure 1: Nested interfering flows scenario.

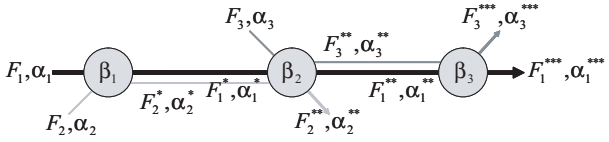

Figure 2: Overlapping interfering flows scenario.

example, if a scenario as depicted in Figure 1 is to be analysed for flow 1, a straightforward end-to-end service curve for flow 1 would be determined as follows (using the notation in Figure 1):

$$\beta_1^{SF} = [\beta_1 - \alpha_2 - \alpha_3]^+ \otimes [\beta_2 - \alpha_2^* - \alpha_3^*]^+ \otimes [\beta_3 - \alpha_3^{**}]^+$$

Yet, another way to analyse the system is to concatenate node 1 and 2, subtract flow 2 and thus receive the service curve for flow 1 and 3 together, concatenate this with node 3 and subtract flow 3, essentially making optimal use of the sub-path sharing between the interfering flows:

$$\beta_1^{PS} = \left[[(\beta_1 \otimes \beta_2) - \alpha_2]^+ \otimes \beta_3 - \alpha_3\right]^+$$

If, for example, $\beta_i = \beta_{3,0}$, $i = 1, 2, 3$ and $\alpha_2 = \alpha_3 = \gamma_{1,1}$ we obtain: $\beta_1^{SF} = \beta_{1,12\frac{1}{2}}$ and $\beta_1^{PS} = \beta_{1,2}$. Hence, exploiting the sub-path sharing between the interfering flows, we obtain an end-to-end service curve with considerably lower latency. Put in other words, we have to pay for the blind multiplexing of the flows only once, which is why we also call this phenomenon the *pay multiplexing only once (PMOO) principle*, in analogy to the well-known pay bursts only once principle [11]. Basically the same observation was made in [8], [12] for *FIFO multiplexing* under the special case of token bucket arrival curves and rate-latency service curves. We have derived an end-to-end service curve for *blind multiplexing* exploiting the PMOO principle under *general piecewise linear concave arrival curves* and *general piecewise linear convex service curves* in [16].

The difficulty in obtaining the end-to-end service under blind multiplexing for a flow of interest lies in situations as depicted in Figure 2. Here, in contrast to the scenario of nested interfering flows as in Figure 1, we have a scenario of overlapping interfering flows: flows 2 and 3 interfere with flow 1, our flow of interest, and share some servers with each other but each of them also traverses servers the other does not traverse. For such a scenario the end-to-end service curve cannot be derived as easily as demonstrated before but requires to look deeper into the input and output relationships of the flows. The following theorem states how to calculate the end-to-end service curve under blind multiplexing, exploiting the PMOO principle for the canonical example of overlapping interfering flows in Figure 2.

THEOREM 6. *(E2E Minimum Service Curve under Blind Multiplexing – Pay Multiplexing Only Once Principle) Consider a scenario as shown in Figure 2, a flow of interest F_1 is interfered by two overlapping flows F_2 and F_3. F_2 and F_3 have arrival curves α_2 and α_3. The three servers offer (strict) piecewise linear convex minimum service curves $\beta_1 = \bigvee_{i=1}^{n_i} \beta_{R_i,T_i}$, $\beta_2 = \bigvee_{j=1}^{n_j} \beta_{\hat{R}_j,\hat{T}_j}$, and $\beta_3 = \bigvee_{k=1}^{n_k} \beta_{\tilde{R}_k,\tilde{T}_k}$. If $\alpha_2 = \bigwedge_{l=1}^{n_l} \gamma_{r_l,b_l}$ and $\alpha_3 = \bigwedge_{m=1}^{n_m} \gamma_{\hat{r}_m,\hat{b}_m}$ are piecewise linear concave arrival curves then*

$$\phi = \bigvee_{i=1}^{n_i} \bigvee_{j=1}^{n_j} \bigvee_{k=1}^{n_k} \bigvee_{l=1}^{n_l} \bigvee_{m=1}^{n_m} \beta_{R_{i,j,k,l,m},T_{i,j,k,l,m}}$$

with

$$R_{i,j,k,l,m} = (R_i - r_l) \wedge \left(\hat{R}_j - r_l - \hat{r}_m\right) \wedge \left(\tilde{R}_k - \hat{r}_m\right)$$

and

$$T_{i,j,k,l,m} = T_i + \hat{T}_j + \tilde{T}_k$$

$$+ \frac{b_l + \hat{b}_m + r_l\left(T_i + \hat{T}_j\right) + \hat{r}_m\left(\hat{T}_j + \tilde{T}_k\right)}{(R_i - r_l) \wedge \left(\hat{R}_j - r_l - \hat{r}_m\right) \wedge \left(\tilde{R}_k - \hat{r}_m\right)}$$

constitutes a strict *(piecewise linear convex) end-to-end service curve for the flow of interest, in particular $F_1^{***} \geq F_1 \otimes \phi$.*

Hence, for a scenario as depicted in Figure 2 the end-to-end service curve for the flow of interest is determined as follows: A rate-latency function is calculated for each combination of (1) token buckets in the arrival curve descriptions of interfering flows and (2) rate-latency functions of the nodal service curve descriptions. The rate of this function is the minimal residual rate at the nodes after the rates of the respective token buckets of interfering flows have been deducted. More interestingly, the latency is composed of the sum of the nodal latency terms and the *burst terms* of the interfering flows (also accounting for the burstiness increase due to the latency of servers), each *paid only once at the server with lowest residual rate.* From all the resulting rate-latency functions the pointwise maximum has to be taken to deliver the end-to-end service curve for the flow of interest. The generalization to an arbitrary number of nodes is notationally complex, therefore it is left out here. It had of course to be taken into account for the implementation of the DISCO Network Calculator. The derivation of the end-to-end maximum service curve is simple and given in the next lemma:

LEMMA 3. *(E2E Maximum Service Curve) Consider a flow that traverses a sequence of systems $\mathcal{S}_i$, $i = 1, \ldots, n$, where it is multiplexed with an arbitrary number of flows. Assume each of the systems offers a nodal maximum service $\bar{\beta}_i$, $i = 1, \ldots, n$, then the flow is offered $\bar{\beta} = \bigotimes_{i=1}^{n} \bar{\beta}_i$ as a maximum service on its path.*

Algorithm 1 Total Flow Analysis

ComputeDelayBound (flow of interest f)
ComputeOutputBound(sink(f), {all flows to sink(f)})
$delay^{total} = 0.0$
forall nodes i on the path from source to sink(f)
$delay_{total} += h\left(\alpha_i^{pred}, \beta_i^{eff}\right)$
return $delay_{total}$

ComputeOutputBound(from node i, flows $\mathbb{F}$)
forall pred(i)
α_i^{pred} += ComputeOutputBound(pred(i), {flows to node i from pred(i)}$\cap\mathbb{F}$)
α_{excl} += ComputeOutputBound(pred(i), {flows to node i from pred(i)}$\setminus\mathbb{F}$)
$\beta_i^{eff} = [\beta_i - \alpha_{excl}]^+$
return $\left(\left(\alpha_{pred} \otimes \bar{\beta}_i\right) \oslash \beta_i^{eff}\right) \otimes \left(\bar{\beta}_i \otimes \delta_{L_i}\right)$

Algorithm 2 Separated Flow Analysis

ComputeDelayBound (flow of interest f)
forall nodes $i \in$ path(f) starting at sink(f)
forall pred(i)
α_{pred}+= ComputeOutputBound(pred(i), {flows to node i from pred(i)}$\setminus\{f\}$)
$\beta_i^{eff} = [\beta_i - \alpha_{pred}]^+$
$\beta^{SF} = \bigotimes_{i=1}^{n} \beta_i^{eff}$
return $h\left(\alpha_f, \beta^{SF}\right)$

ComputeOutputBound(from node i, flowset $\mathbb{F}$)
forall pred(i)
α_{pred}+ =ComputeOutputBound(pred(i), {flows to node i from pred(i)}$\cap\mathbb{F}$)
α_{excl}+= ComputeOutputBound(pred(i), {flows to node i from pred(i)}$\setminus\mathbb{F}$)
$\beta_i^{eff} = [\beta_i - \alpha_{excl}]^+$
return $\left(\left(\alpha_{pred} \otimes \bar{\beta}_i\right) \oslash \beta_i^{eff}\right) \otimes \left(\bar{\beta}_i \otimes \delta_{L_i}\right)$

Next, we present the algorithms to analyse networks of queues. We focus on the delay bound computation. The backlog bound computation is analogous and just requires to compute the vertical deviation v where for the delay bound the horizontal deviation h is calculated. Of course, delay as well as backlog bound computations are implemented in the DISCO Network Calculator.

3.2 Total Flow Analysis

The first network analysis algorithm is conceptually the easiest one. It is effectively a node-by-node analysis, where for each node all flows that share this node with the flow of interest are analysed as a whole, hence its name *total flow analysis*. So, the total flow analysis does not make use of the concatenation theorem like the other algorithms in the following sections. Therefore it usually performs worse in larger network scenarios. For smaller scenarios, where a flow of interest is comparatively small to the overall flow, it can also perform better, though.

The algorithm (on a high level) to perform the network analysis is given in Algorithm 1. First we have to recursively compute the output bounds of all sensor nodes on the path from source to sink for the flow of interest (starting the recursion at the sink). Here, we sum all traffic of flows joining the flow of interest and having the same sink as the flow of interest. Next we sum all cross-traffic that joins the flow of interest but also leaves it again on the way to a sink different from that of the flow of interest. The reduction in the service curve guarantee by a given node on the path of the flow of interest is then calculated based on Theorem 3 using the cross-traffic (bounded by α_{excl}) and the nodal service curve β_i. At the end of the output bound computation, the min-plus deconvolution of the overall traffic towards the sink of the flow of interest (bounded by α_i^{pred}) and the effective service curve of the given node β_i^{eff} is computed.

For each node on the path of the flow of interest the effective service curves β_i^{eff} and the bound on the overall traffic towards the sink of the flow of interest α_i^{pred} are stored. These are then used in the delay bound computation which now works its way up starting from the source of the flow of interest to its sink. At each node the horizontal deviation between α_i^{pred} and β_i^{eff} (i.e. the per-hop delay bound) is calculated and summed up resulting in an end-to-end delay bound for the flow of interest.

3.3 Separated Flow Analysis

The next network analysis algorithm uses the concatenation theorem. It essentially separates the flow of interest from the cross-traffic by other flows joining it on its way from source to sink, which is why we have called it *separated flow analysis*. It is again based on Theorem 3 and the way it is done is described on a high level in Algorithm 2. It consists of two procedures: One to compute the delay bound for the flow of interest using the nodal service curve under blind multiplexing from Theorem 3 and convolving all nodal service curves to find the end-to-end service curve for the flow of interest which is then used to compute a delay bound for the flow. To be able to compute the nodal service curve, again a procedure to compute the output bound for all interfering flows at a certain node is required. This is a recursive procedure which needs to take into account the effect of a general feed-forward network that even flows which never share a server with the flow of interest may affect the flow of interest transitively, by interfering with a flow which in turn interferes with the flow of interest. As already mentioned above, the separated flow analysis is often superior to using the total flow analysis, since it leverages the power of the concatenation theorem and thus pays the burst of the flow of interest only once. However, as presented in Section 3.1 there is a different way to compute the end-to-end service curve for the flow of interest which is often superior to straightforwardly using Theorem 3 and which allows to pay the multiplexing with each interfering flow only once.

3.4 PMOO Analysis

This network analysis algorithm is based on Theorem 6 and the way it is done is described in Algorithm 3. It consists of three procedures. The simplest is the one which computes the delay bound by calling another procedure to compute the end-to-end service curve for the flow of interest according to Theorem 6. From the result it then calculates the delay bound for the flow of interest. The main complexity lies in the procedure to compute the PMOO service curve.

Algorithm 3 PMOO Analysis

ComputeDelayBound (flow of interest f)
β^{PMOO}= ComputePMOOServiceCurve(path(f), $\{f\}$)
return $h\left(\alpha_f, \beta^{PMOO}\right)$

ComputePMOOServiceCurve(path p, flowset $\mathbb{F}$)
eliminate rejoining interfering flows
merge parallel interfering flows into flowsets $\mathbb{F}_i$
forall interfering flowsets $\mathbb{F}_i$
  forall pred($n_{\mathbb{F}_i}$) of the ingress node $n_{\mathbb{F}_i}$ of $\mathbb{F}_i$
    $\alpha_{\mathbb{F}_i}$+= ComputeOutputBound(pred($n_{\mathbb{F}_i}$), $\mathbb{F}_i$)
calculate β^{PMOO} using $\alpha_{\mathbb{F}_i}$ according to Theorem 6
return β^{PMOO}

ComputeOutputBound(from node i, flowset $\mathbb{F}$)
find shared path p of flows in $\mathbb{F}$ starting at node i
call s the last node on path p ($\rightarrow$split point)
forall pred(s)
  α_{split}+= ComputeOutputBound(pred(s),
    {flows to node s from pred(s)}$\cap\mathbb{F}$)
β_p^{PMOO}= ComputePMOOServiceCurve(p, $\mathbb{F}$)
calculate $\bar{\beta}_p^{PMOO}$ according to Lemma 3
return $\left(\left(\alpha_{split} \otimes \bar{\beta}_p^{PMOO}\right) \oslash \beta_p^{PMOO}\right) \otimes \left(\bar{\beta}_p^{PMOO} \otimes \delta_{L_p}\right)$

Initially, the path of the flow of interest needs to be prepared to enable an efficient application of the PMOO result. At first, flows that join and leave the flow of interest several times are eliminated by introducing fresh flows for each later rejoin (with the arrival constraints of the original flow at that node in the network, of course). Next, parallel interfering flows, i.e. flows that have the same ingress and egress nodes, are merged together into a flow set which is treated as one flow for the computation of the service curve. The merging helps to keep the procedure as efficient as possible, since fewer flows have to be taken into account. However, care must be taken when computing the arrival constraints for each of the flow sets, since the flows contained in a set are generally not in parallel throughout the whole network. This is done by a separate procedure which we discuss in a moment. After the arrival constraints of each of the flow sets have been computed, Theorem 6 is applied to compute the desired end-to-end PMOO service curve. This statement sounds harmless, however the computation of the maximum over all token buckets of the piecewise linear arrival curves of interfering flows and rate-latency functions of the piecewise linear service curve of the nodes on the path of the flow of interest can be a computationally intensive task. We come back to this issue in the next section. The last procedure serves to compute the output bound of a flow set behind a given node. Here, the PMOO result is reused as much as possible by first finding the sub-path shared by all flows in the flow set, respectively the node where the flow set splits. From that split point the procedure recurs on itself for each of the smaller flow sets after the split point. With the sum of those output bounds after the split point, the procedure now calls in indirect recursion the procedure for the computation of the PMOO service curve for the shared path of the flow set. Furthermore, the maximum service curve according to Lemma 3 is calculated. Together, minimum and maximum service curve are applied to calculate the output bound according to Lemma 1.

4. A TOY EXAMPLE FEATURING THE DISCO NETWORK CALCULATOR

All of the algorithms and basic operations described so far are implemented in the DISCO Network Calculator, as well as some further features (for a complete documentation see `http://disco.informatik.uni-kl.de/content/Downloads`). In Figure 3, we provide for illustrative purposes an almost complete code segment which utilizes the DISCO Network Calculator Java classes to analyse a toy network scenario. The idea of this example is to expose the usage of the DISCO Network Calculator.

In the `main()` method of the `DNCLibDemo` class at first a simple network consisting of 11 nodes is created. Note that *network topologies* can also be imported if they have been generated elsewhere as long as they adhere to the *GraphML* format. In particular, we provide an *export filter* for the popular *BRITE*[2] topology generator. Anyway, the network topology is then transformed into an equivalent topology where the links of the original topology, whose function is to be modelled by service curves, are now represented by nodes in the so-called server graph. Furthermore, we apply the *turn-prohibition* algorithm from [18], which is also implemented as part of the DISCO Network Calculator. Next, each node is assigned simple rate-latency minimum and maximum service curves as well as each flow from a source to a sink is assigned a simple token bucket arrival curve. The flow of interest for which a network analysis shall be performed is then chosen arbitrarily from the set of all flows. Before the network analyses can be performed some flags can be set. These are governing whether the maximum service curves shall be used for the analysis, whether FIFO or blind multiplexing shall be assumed, and whether the tightened output bound from Lemma 1 shall be exploited. Now everything is set for a comparison of different ways to analyse this simple scenario. As we also implemented the calculation of the bounds from [5] (often called the *Charny bound*) as part of the DISCO Network Calculator, we compute delay and backlog bound for these first. Next, we provide the results for an idealised situation where it is assumed that all servers distribute their resources fairly among the flows traversing them. Also this analysis method, called *fair queueing*, is implemented in the DISCO Network Calculator, mainly as a benchmark in order to compare the worst-case analyses with an emulation of the best possible case. And last not least, we invoke the analysis methods we presented in Section 3: total flow analysis, separated flow analysis, and PMOO analysis.

The idea of this toy example is not to discuss the results of these analysis methods now (if the reader is interested in playing around a little bit with this toy example, it is provided in the DISCO Network Calculator package), but it should demonstrate how easy it is to use the DISCO Network Calculator. Finally, it may additionally be mentioned that a simple class for graphically viewing piecewise-linear curves is also provided in the DISCO Network Calculator. We found this very helpful in combination with sampled versions of min-plus convolution and deconvolution in order to get a feeling for these basic min-plus algebraic operations for given types of curves.

[2] http://www.cs.bu.edu/brite

Figure 3: Java Code for the Toy Example

```
public class DNCLibDemo {
  private String filename;
  private DirectedSparseGraph network_graph;
  private DirectedSparseGraph server_graph;
  private NetworkAnalyser na;
  public DNCLibDemo(String filename) {this.filename = filename;}


  public DirectedSparseGraph createDemoNetwork() {
    DirectedSparseGraph network_graph = new DirectedSparseGraph();
    for (int i = 0; i < 11; i++)
      network_graph.addVertex(new SimpleSparseVertex());
    Vertex[] vertices = (Vertex[]) network_graph.getVertices().toArray(
    new Vertex[network_graph.numVertices()]);
    network_graph.addEdge(new DirectedSparseEdge(vertices[0], vertices[1]));
    ...
    // make graph bidirected
    ...
    return network_graph;
  }


  public DirectedSparseGraph convert2ServerGraph(DirectedSparseGraph network_graph){
    DirectedSparseGraph server_graph;
    Map link_server_map = new HashMap();
    Map router_turns_map = new HashMap();
    server_graph = GraphUtils.createServerGraph(network_graph,
                                  link_server_map, router_turns_map);
    TurnProhibition tp = new TurnProhibition(network_graph);
    tp.runTurnProhibition();
    tp.removeProhibitedTurns(server_graph, link_server_map,router_turns_map);
    return server_graph;
  }


  public static void main(String[] args) {
    // create a network graph manually...
    network_graph = createDemoNetwork();
    // ...  or by loading an existing graph, e.g.  created and exported using BRITE
    server_graph = convert2ServerGraph(network_graph);
    // create a new network analyser instance for the server graph
    na = new NetworkAnalyser(server_graph);
    // create a service curve for each server
    Set servers = GraphUtils.getServerSet(server_graph);
    na.setServiceCurve(servers, Curve.createRateLatency(0.01, 10.0e6));
    na.setMaxServiceCurve(servers, Curve.createRateLatency(0.001, 100.0e6));
    // create a flow between every source-sink-pair if a path exists
    DijkstraShortestPath shortest_paths = new DijkstraShortestPath(server_graph);
    Curve flow_prototype = Curve.createTokenBucket(0.1 * 0.1e6, 0.1e6);
    Set sources = GraphUtils.getSourceSet(server_graph);
    Set sinks = GraphUtils.getSinkSet(server_graph);
    for (Iterator sourceIter = sources.iterator(); sourceIter.hasNext();;) {
      Vertex source = (Vertex) sourceIter.next();
      for (Iterator sinkIter = sinks.iterator(); sinkIter.hasNext();;) {
        Vertex sink = (Vertex) sinkIter.next();
        List path = shortest_paths.getPath(source, sink);
        if (path.size() > 0)
          na.addFlow(new Flow(source, sink, flow_prototype.copy(), path));
      }
    }
    // select the flow of interest
    Flow flow_of_interest = (Flow) na.getFlows().get(0);
    System.out.println("Flow of interest :  " + flow_of_interest.toString());
    // analyse the network
    Curve beta;
    Map server_bound_map;
    na.setUseFifoService(false);
    na.setUseGamma(false);
    na.setUseExtraGamma(false);
    System.out.println("--- Charny Bound ---");
    System.out.println("cur.  utilization:  " + na.getHighestServerUtilization());
    System.out.println("max.  utilization:  " + na.getMaxUtilizationForCharnyBound());
    System.out.println("delay bound :  " + na.getCharnyDelayBound());
    server_bound_map = na.getCharnyBacklogBounds();
    if (!server_bound_map.isEmpty()) {
      System.out.println("backlog bound :  " +
                         Collections.max(server_bound_map.values()));
      System.out.println(" per server :  " + server_bound_map.values().toString());
    } else {
      System.out.println("backlog bound :  " + Double.POSITIVE_INFINITY);
    }
    System.out.println("--- Fair Queueing ---");
    beta = na.performFairQueueingAnalysis(flow_of_interest);
    System.out.println("delay bound :  " +
                         Curve.getDelayBound(flow_of_interest.ac, beta));
    System.out.println("backlog bound :  " +
                         Curve.getBacklogBound(flow_of_interest.ac, beta));
    System.out.println("--- Total Flow Analysis ---");
    server_bound_map.clear();
    double delay = na.performTotalFlowAnalysis(flow_of_interest, server_bound_map);
    System.out.println("delay bound :  " + delay);
    System.out.println("backlog bound :  " +
                         Collections.max(server_bound_map.values()));
    System.out.println(" per server :  " + server_bound_map.values().toString());
    System.out.println("--- Separated Flow Analysis ---");
    beta = na.performSeparatedFlowAnalysis(flow_of_interest);
    System.out.println("delay bound :  " +
                         Curve.getDelayBound(flow_of_interest.ac, beta));
    System.out.println("backlog bound :  " +
                         Curve.getBacklogBound(flow_of_interest.ac, beta));
    System.out.println("--- PMOO Analysis ---");
    beta = na.performPMOOAnalysis(flow_of_interest);
    System.out.println("delay bound :  " +
                         Curve.getDelayBound(flow_of_interest.ac, beta));
    System.out.println("backlog bound :  " +
                         Curve.getBacklogBound(flow_of_interest.ac, beta));
  }

}
```

5. REAL-WORLD APPLICATION EXAMPLES

After the illustrative toy example from the preceding section we briefly present two studies we have performed with the help of the DISCO Network Calculator.

5.1 Performance Bounds in Turn-Prohibited Networks

In this first example application of the DISCO Network Calculator we compute bounds in networks of arbitrary topologies. In order to create a representative network we used the BRITE topology generator with the following recommended parameter settings: 2-level-top-down topology type with 10 ASes with 10 routers each, Waxman's model with $\alpha = 0.15, \beta = 0.2$, and random node placement with incremental growth type. The diameter of the generated topology is 11 hops. Hosts are created randomly at access routers: 30% random routers in each AS get a uniform random number of hosts between 1 and 5.

This topology is then transformed into a feed-forward network using the turn-prohibition algorithm. The turn-prohibition algorithm resulted in a change of 13% of the shortest path routes with an average change of 1.9 hops and a standard deviation of 1.2. So, altogether the routes were changed only by 0.25 hops on average. This can be considered a success for the turn-prohibition algorithm, because it was able to transform this general network with a diameter of 11 into a feed-forward network. Thus it is now suitable to be analysed by the methods presented in Section 3 for all possible network utilizations ≤ 1, whereas if it had been analysed as a general network using, e.g., the bound from [5] it could have been analysed only for utilizations below 10% (according to Theorem 5), so we are in a much better position now without paying a high cost in excessive route prolongation.

Now, for each source a flow is generated towards a randomly assigned sink. Each flow's arrival curve is drawn from a set $\mathbb{S}_\alpha$ of candidate arrival curves. Specifically, for this investigation we restricted the set members to token buckets. The minimum service curves were restricted to be rate-latency functions. These provide 1 of 3 rates: C, $2C$, or $3C$, where C is chosen such that the most loaded link can carry its load (plus a margin of 10%) and for the other links the rate is chosen closest to their carried load. The latency is chosen to be 10ms network-wide. If a maximum service curve is used its latency is set to zero and its rate equal to the rate of the corresponding nodal minimum service curve.

PMOO Analysis vs. Separated Flow Analysis

One of the goals of the more comprehensive study in [16] was to find out how the separated flow analysis and the PMOO analysis compare to each other. We perform this comparison for two different traffic scenarios, one with low burstiness ($\mathbb{S}_\alpha = \{\gamma_{r,0.05[s]r}\}$, with $r = 10$ Mbit/s) and the other one with significantly higher burstiness ($\mathbb{S}_\alpha = \{\gamma_{r,0.05[s]r}, \gamma_{r,0.25[s]r}, \gamma_{r,0.5[s]r}\}$, with $r = 10$ Mbit/s). Furthermore, we want to isolate the effects by (1) the service curve computation using the PMOO principle, (2) the output bound computation using the PMOO principle, and (3) the maximum service curve in its improved version according to Lemma 1. We use all available sinks for flow genera-

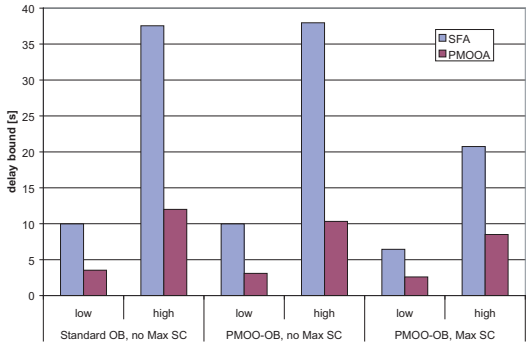

Figure 4: PMOO analysis vs. separated flow analysis in turn-prohibited networks.

tion and compute 10 replications with different seeds. The results can be found in Figure 4. First we take a look at the effect of the PMOO service curve alone: It is considerably better than its separated flow counterpart for both low and high burstiness of the traffic. As expected, the burstier traffic incurs higher absolute delays, but the percentage of improvement is roughly the same for both types of flows: $\approx 66 \pm 6.5\%$ at a 95% level of confidence. When the PMOO output bound is used this gives a further significant improvement of $\approx$ 13% for the PMOO analysis (for both traffic types). When the maximum service curve is used this reduces the bounds by another $\approx$ 16%. The maximum service curve also improves the separated flow analysis by $\approx$ 35% (low) and $\approx$ 45% (high) and thus keeps its promise to be valuable for a good delay bound analysis although it leads to more complex models.

5.2 Sensor Network Calculus with Multiple Sinks

In this second example application we switch to the very different domain of wireless sensor networks. In [15] we have described the basic setting regarding how to use network calculus in sensor networks, coining the term *sensor network calculus*. In this study we use the total flow analysis because there are several arguments why the separated flow analysis or PMOO analysis are not so well suited when one starts to think of integrating the typical in-network processing in sensor networks into the models. Yet, we do not want to go into a detailed discussion here, the interested reader is referred to [15]. Differently from the first application as well, we assume FIFO instead of blind multiplexing now, since the sensors might be assumed to be FIFO nodes.

The wireless sensor network scenario is chosen with the intention of describing realistic and common application scenarios, yet they are certainly simplifying matters to some degree for illustrative purposes. The goal of this study was to show how sensor network calculus may be able to shed some light upon how the number of sinks affects the worst case message transfer delay in typical wireless sensor networks. The experimental setup is as follows: we assume a flat sqare of 100x100 m^2 on which the sensor nodes are randomly distributed (that means their x- and y-coordinates are chosen from a uniform random distribution over [0,100]).

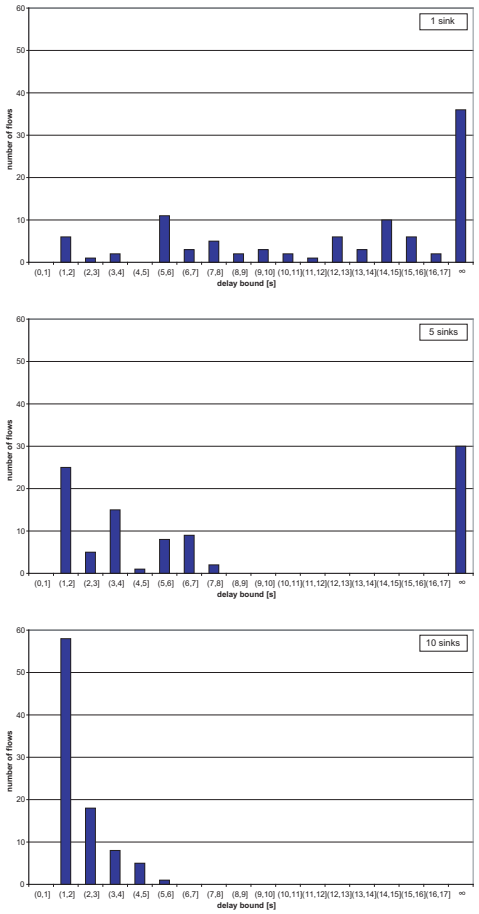

Figure 5: Worst case delay distribution for different numbers of sinks in a wireless sensor network.

Each sensor has a transmission range of 20 meters. The routing from the sensors to the sinks is done based on shortest paths from sensors to sinks, each sensor node is associated with its nearest sink. The sinks are randomly chosen from the sensor nodes (which effectively releases them from being sensor nodes). Initially we create 100 nodes and then designate the respective number of sinks for the given experiment. The sensor node models we use mimic Mica-2 sensors running TinyOS. In particular we assume a duty cyle of 1% which results in a latency of 1.096 s and a forwarding rate of 258 b/s. Furthermore, we assume a periodic sensing task: each sensor sends a 36 byte TinyOS packet every 30 seconds.

This scenario was readily implemented using the DISCO Network Calculator. Some of the results of the experiments, i.e. the worst-case delay calculation for each sensor node based on a total flow analysis, with different number of sinks are shown in Figure 5. Here we show for 1, 5, and 10 sinks in the sensor field the worst case delay distribution over all possible flows of interest in the form of histograms where each bar gives the number of flows in intervals of duration 1 s. Note that for some flows there may not exist a finite delay bound (represented by the last bar in the histograms) since under worst case conditions the amount of incoming

traffic of a node on the path of that flow may be higher than its forwarding rate. It can be clearly seen from Figure 5 how the delay distribution improves with the number of sinks used in the sensor field. In fact, the average worst case delay improves from 9.58 to 3.34 to 1.82 s for the 1, 5, and 10 sink scenario, respectively (not counting the flows with infinite delay bounds for the 1 and 5 sink case, of course). So we see how sensor network calculus may advise us to find the number of sinks that we shall use in order to receive a certain message transfer delay performance.

6. CONCLUSION

In this paper we presented a tool to facilitate network calculus analyses: the DISCO Network Calculator. The basic min-plus algebraic operations in the domain of piecewise linear arrival and service curves were discussed as they form the basis for the operation of the toolbox. Furthermore we presented different network analysis algorithms that are contained in the DISCO Network Calculator. To illustrate the application of the tool a simple toy example was presented as well as two example applications from real-world studies we conducted.

From our experience, the DISCO Network Calculator has proven to be a valuable tool for our research so that we decided to make it publicly available. Thereby, we hope to support the development of network calculus as a fundamental theory for the performance modelling of distributed systems.

7. REFERENCES

[1] R. Agrawal, R. L. Cruz, C. Okino, and R. Rajan. Performance bounds for flow control protocols. *IEEE/ACM Transactions on Networking*, 7(3):310–323, June 1999.

[2] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Probability and Mathematical Statistics. John Wiley & Sons Ltd., West Sussex, Great Britain, 1992.

[3] S. Chakraborty, S. Kuenzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Computer Networks*, 42(5):641–665, 2003.

[4] C.-S. Chang. On deterministic traffic regulation and service guarantees: A systematic approach by filtering. *IEEE Transactions on Information Theory*, 44(3):1097–1110, May 1998.

[5] A. Charny and J.-Y. Le Boudec. Delay bounds in a network with aggregate scheduling. In *Proc. QofIS*, pages 1–13, September 2000.

[6] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, August 1995.

[7] R. L. Cruz. SCED+: Efficient management of quality of service guarantees. In *Proc. IEEE INFOCOM*, volume 2, pages 625–634, March 1998.

[8] M. Fidler and V. Sander. A parameter based admission control for differentiated services networks. *Computer Networks*, 44(4):463–479, 2004.

[9] H. Kim and J.C. Hou. Network calculus based simulation: theorems, implementation, and evaluation. In *Proc. IEEE INFOCOM*, March 2004.

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

[11] J.-Y. Le Boudec and P. Thiran. *Network Calculus – A Theory of Deterministic Queuing Systems for the Internet*. Number 2050 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 2001.

[12] L. Lenzini, L. Martorini, E. Mingozzi, and G. Stea. Tight end-to-end per-flow delay bounds in FIFO multiplexing sink-tree networks. *Performance Evaluation*, 63:956–987, 2006.

[13] H. Sariowan, R. L. Cruz, and G. C. Polyzos. Scheduling for quality of service guarantees via service curves. In *Proc. IEEE ICCCN*, pages 512–520, September 1995.

[14] J. Schmitt. On the allocation of network service curves for bandwidth/delay-decoupled scheduling disciplines. In *Proc. of IEEE GLOBECOM*, pages 1544–1548. IEEE, November 2002.

[15] J. Schmitt and U. Roedig. Sensor network calculus - a framework for worst case analysis. In *Proc. IEEE/ACM DCOSS*, pages 141–154, June 2005.

[16] J. Schmitt, F. Zdarsky, and I. Martinovic. Performance bounds in feed-forward networks under blind multiplexing. Technical Report 349/06, University of Kaiserslautern, Germany, April 2006.

[17] T. Skeie, S. Johannessen, and O. Holmeide. Timeliness of real-time ip communication in switched industrial ethernet networks. *IEEE Transactions on Industrial Informatics*, 2(1):25–39, February 2006.

[18] D. Starobinski, M. Karpovsky, and L. A. Zakrevski. Application of network calculus to general topologies using turn-prohibition. *IEEE/ACM Transactions on Networking*, 11(3):411–421, 2003.

[19] I. Stoica, H. Zhang, and T. S. E. Ng. A hierarchical fair service curve algorithm for link-sharing, real-time and priority services. In *Proc. ACM SIGCOMM*, pages 249–262, September 1997.