Bachelor Thesis

Deterministic Performance Analysis of FIFO-multiplexing Feed-forward Networks

by

Alexander Scheffler
Matr.-No. 388551

November 2, 2017

University of Kaiserslautern
Department of Computer Science
Distributed Computer Systems Lab

Examiner: Prof. Dr.-Ing. Jens B. Schmitt
Supervisor: Dr.-Ing. Steffen Bondorf

Abstract

The presented work discusses the Least Upper Delay Bound (LUDB) — a heuristic worst-case analysis principle in deterministic network calculus that is integrated into the DiscoDNC tool and compared to other analysis principles such as Separate Flow Analysis (SFA) and Pay Multiplexing Only Once (PMOO). Moreover, the integration of the LUDB supports any feed-forward network structure and not only tandems.

Die vorliegende Arbeit bespricht den Least Upper Delay Bound (LUDB) — eine heuristische Worst-Case Analysemethode im deterministischen Network Calculus. Diese wurde in das DiscoDNC Tool integriert und mit den üblichen Analysemethoden wie Separate Flow Analysis (SFA) und Pay Multiplexing Only Once (PMOO) verglichen. Darüber hinaus unterstützt sie die Analyse von beliebigen Feed-Forward Netzwerkstrukturen, ist also nicht nur auf Tandems beschränkt.

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit mit dem Thema "Deterministic Performance Analysis of FIFO-multiplexing Feed-forward Networks" selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

(Ort, Datum) (Unterschrift)

Acknowledgement

First of all, I would like to thank my advisor, Dr. Steffen Bondorf, who introduced me to the theory of Network Calculus as well as the DiscoDNC tool. He always had the time for insightful discussions, provided feedback and suggestions for improvement.

Contents

1 Network Calculus Foundations

Network calculus is a theory for (deterministic) network performance analysis [1] which helps us to gain insights into characteristics of integrated services networks, window flow control, scheduling and buffer or delay dimensioning [2]. It has a wide range of applications such as wireless sensor networks, Ethernet installations and Systems-on-Chip [1].

1.1 Data Modeling

Data flows are modeled with wide-sense increasing cumulative functions [2], i.e. functions that are in $\mathcal{F} = \{f : \mathbb{R} \to \mathbb{R} \cup \{+\infty\} | \forall s \leq t : f(s) \leq f(t), \forall d < 0 : f(d) = 0\}$. Such a function $R(t)$ represents the amount of (work) units (e.g. bits or packets) which arose in the time interval $[0, t]$ and we will assume that $R(0) = 0$ holds [2].

1.2 Min-plus Algebra

Definition (Min-plus Convolution [2]). Let f and g be in $\mathcal{F}$. The min-plus convolution of f and g is defined as

$$(f \otimes g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}$$

Definition (Min-plus Deconvolution [2]). Let f and g be in $\mathcal{F}$. The min-plus deconvolution of f and g is defined as

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

1.3 Arrival Curves

Definition (Arrival Curve [2]). An arrival curve $\alpha : \mathbb{R}_+ \to \mathbb{R}_+ \cup \{+\infty\}$ is a wide-sense increasing function with $\alpha(0) = 0$ for input functions $R(t)$ iff

$$R(t) - R(t-s) \leq \alpha(s) \ \forall s \leq t$$

We say that R is α-smooth.

Example (Affine Curve [2]). The affine (arrival) curve $\gamma_{r,b}$ is defined as follows:

$$\gamma_{r,b}(t) = (b + r \cdot t) \cdot 1_{\{t>0\}} = \begin{cases} 0 & \text{if } t \leq 0 \\ b + r \cdot t & \text{otherwise} \end{cases}$$

b is called burst and r is the rate. Hence, this kind of arrival curve is suitable for flows in which b work units can occur immediately and the arrival rate of work units does not exceed r in the long run.

Affine curves are often called leaky-bucket or token-bucket curves. First, we will discuss the token-bucket algorithm [3] (see also figure 1.1).

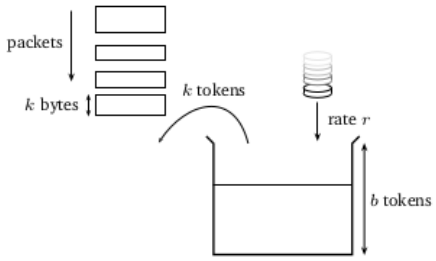

Figure 1.1: Token-bucket algorithm [3]

The bucket has a capacity of b tokens and is filled at the beginning. Tokens arrive at the bucket with a rate r and are added to it if it is not full, otherwise the arriving tokens are dropped. If a packet of size k needs to be sent, then k tokens are taken from the bucket. If there are not enough tokens in it, then the packet will be dropped (or queued). As a result of this, $\gamma_{r,b}$ is a valid arrival curve for the packets to be sent.

We now discuss the leaky-bucket algorithm [3] (see also figure 1.2).

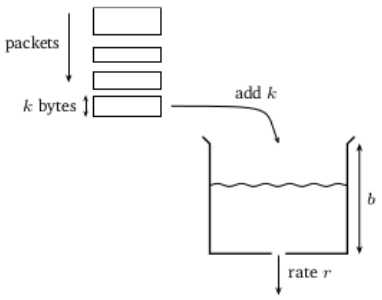

Figure 1.2: Leaky-bucket algorithm [3]

The bucket has a capacity of b fluid units and is empty at the beginning. Furthermore, it has a leak, so the fluid leaves the bucket with rate r (if it is not empty of course). If a packet of size k has to be sent, then k fluid units are added to the leaky bucket if it does not lead to an overflow. Otherwise, the packet is dropped (or queued). As a result of this, $\gamma_{r,b}$ is a valid arrival curve for the packets to be sent.

Example (Pseudoaffine Curve [1]). A pseudoaffine curve has the following structure:

$$\pi = \delta_D \otimes \left[\bigotimes_{1 \le x \le n} \gamma_{\sigma_x, \rho_x} \right]$$
$$\overset{(*)}{=} \delta_D \otimes \left[\bigwedge_{1 \le x \le n} \gamma_{\sigma_x, \rho_x} \right]$$

$(*)$ Let $f, g \in \mathcal{F}$ with $f(0) = g(0)$ be concave functions. Then $f \otimes g = f \wedge g$ [1].
The curve δ_D is defined as follows (slightly modified stages compared to [1]):

$$\delta_D(t) = \begin{cases} +\infty & \text{if } t > D \\ 0 & \text{otherwise} \end{cases}$$

δ_D shifts any function $f \in \mathcal{F}$ (with $f(0) = 0$) D units to the right:

$$\begin{aligned} (\delta_D \otimes f)(t) &= \inf_{0 \le s \le t} \{\delta_D(t-s) + f(s)\} \\ &= \inf_{0 \le s \le t} \{f(t-s) + \delta_D(s)\} \\ &= \begin{cases} 0 & \text{if } t \le D \\ f(t-D) & \text{otherwise} \end{cases} \end{aligned}$$

Taken together, π has an offset of D and then takes the "pointwise" minimum of the multiple affine curves (between brackets) – figure 1.3 illustrates this concept. The latter are called leaky-bucket stages [1].

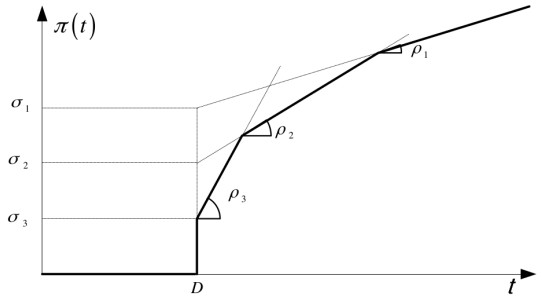

Figure 1.3: A pseudoaffine curve with three leaky-bucket stages [1]

1.4 Service Curves

Definition (Service Curve [2]). Let S be a system and a flow traverses it with input and output function R and R'. Then β is a service curve iff

$$\beta \in \mathcal{F} \text{ with } \beta(0) = 0 \text{ and } R' \geq R \otimes \beta$$

Example (Rate-latency Curve [1]). The rate-latency (service) curve $\beta_{\theta,R}$ is defined as follows:

$$\beta_{\theta,R}(t) = R \cdot [t-\theta]^+ = \begin{cases} 0 & \text{if } t-\theta \leq 0 \\ R(t-\theta) & \text{otherwise} \end{cases}$$

Rate-latency curves are pseudoaffine since

$$\begin{aligned} \beta_{\theta,R}(t) &= \begin{cases} 0 & \text{if } t-\theta \leq 0 \\ R(t-\theta) & \text{otherwise} \end{cases} \\ &= \begin{cases} 0 & \text{if } t \leq \theta \\ R(t-\theta) & \text{otherwise} \end{cases} \\ &= (\delta_\theta \otimes \gamma_{0,R})(t) \end{aligned}$$

holds.

Definition (Strict Service Curve [2]). Let S be a system. S offers a strict service curve β to a flow iff for every backlogged period of duration u, the output of the flow is at least $\beta(u)$.

1.5 Basic Results

Theorem (Backlog Bound [2]). Let S be a system with service curve β and a flow traverses it with arrival curve α, input and output function R and R'. Then the backlog is constrained by

$$R(t) - R'(t) \leq \sup_{u \geq 0} \{\alpha(u) - \beta(u)\} \quad \forall t \geq 0$$

Theorem (Delay Bound [2]). Let S be a system with service curve β and a flow traverses it with arrival curve α. Then the virtual delay $d(t)$ is constrained by

$$\begin{aligned} d(t) &= \inf\{\tau \geq 0 : R'(t+\tau) \geq R(t)\} \\ &\leq \sup_{s \geq 0}\{\inf\{\tau \geq 0 : \alpha(s) \leq \beta(s+\tau)\} \\ &= h(\alpha, \beta) \quad \forall t \geq 0 \end{aligned}$$

Theorem (Output Bound [2]). Let S be a system with service curve β and a flow traverses it with arrival curve α. Then the output arrival curve is computed as follows

$$\alpha' = \alpha \oslash \beta$$

Theorem (Concatenation [2]). A flow traverses two systems S_1 and S_2 in sequence which offer service curves β_1 and β_2, respectively. Then the concatenation of the two systems offers a service curve $\beta_1 \otimes \beta_2$ to the flow.

Theorem (Arbitrary Multiplexing — Left-Over Service [2]). Consider a node arbitrarily multiplexing two flows, 1 and 2. Assume that the node guarantees a strict service curve β to the aggregate of the two flows and that flow 2 is α_2-smooth. Then

$$\beta^1 = [\beta - \alpha_2]^+$$

is a service curve for flow 1 if $\beta^1 \in \mathcal{F}$.

Theorem (FIFO Multiplexing — Left-Over Service [2]). Consider a node serving two flows, 1 and 2, in FIFO order — i.e. First-In, First-Out, so no overtaking takes place. Assume that the node guarantees a service curve β to the aggregate of the two flows. Assume that flow 2 has α_2 as an arrival curve. Then for any $\theta \geq 0$

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

is a service curve for flow 1 if $\beta^1_\theta \in \mathcal{F}$.
Note that we get a valid left-over service curve for each $\theta \geq 0$ (if $\beta^1_\theta \in \mathcal{F}$) — so there is an infinite amount of possibilities in choosing this parameter. This thesis focuses on this problem in the context of "minimizing" the delay bound for FIFO-multiplexing nodes.

Lemma (Convolution of Pseudoaffine Curves [1]). The convolution of two pseudoaffine curves, π and π', is a pseudoaffine curve, whose offset is the sum of the offsets of the operands, and whose leaky-bucket stages are the union of the leaky-bucket stages of both operands.
Proof.

$$\pi = \delta_D \otimes \left[\bigotimes_{x_{l_i}, i \in \{1,\ldots,n\}} \gamma_{\sigma_{x_{l_i}}, \rho_{x_{l_i}}} \right], \quad \pi' = \delta_{D'} \otimes \left[\bigotimes_{x_{r_i}, i \in \{1,\ldots,n'\}} \gamma_{\sigma_{x_{r_i}}, \rho_{x_{r_i}}} \right]$$

$$\pi \otimes \pi' = \left(\delta_D \otimes \left[\bigotimes_{x_{l_i}, i \in \{1,\ldots,n\}} \gamma_{\sigma_{x_{l_i}}, \rho_{x_{l_i}}} \right] \right) \otimes \left(\delta_{D'} \otimes \left[\bigotimes_{x_{r_i}, i \in \{1,\ldots,n'\}} \gamma_{\sigma_{x_{r_i}}, \rho_{x_{r_i}}} \right] \right)$$

$$= (\delta_D \otimes \delta_{D'}) \otimes \left(\left[\bigotimes_{x_{l_i}, i \in \{1,\ldots,n\}} \gamma_{\sigma_{x_{l_i}}, \rho_{x_{l_i}}} \right] \otimes \left[\bigotimes_{x_{r_i}, i \in \{1,\ldots,n'\}} \gamma_{\sigma_{x_{r_i}}, \rho_{x_{r_i}}} \right] \right)$$

$$\overset{*}{=}_{\substack{L := \bigcup l_i, R := \bigcup r_j \\ i \in \{1,\ldots,n\}, j \in \{1,\ldots,n'\}}} \delta_{D+D'} \otimes \left[\bigotimes_{x_k, k \in L \cup R} \gamma_{\sigma_{x_k}, \rho_{x_k}} \right]$$

$(*)$ Min-plus convolution is associative and commutative [2].

Lemma (Delay Bound for Pseudoaffine Curves [1]). Let π be a pseudoaffine curve, i.e. $\pi = \delta_D \otimes \left[\bigotimes_{1 \le x \le n} \gamma_{\sigma_x, \rho_x} \right]$, and let $\alpha = \gamma_{\sigma,\rho}$. If $\rho_\pi^* := \min_{x=1,\dots,n} \rho_x \ge \rho$, then:

$$h(\alpha, \pi) = \pi^{-1}(\sigma) = D + \bigvee_{1 \le x \le n} \left[\frac{\sigma - \sigma_x}{\rho_x} \right]^+$$

[$\vee$ is the max operator.] Intuition: σ is the burst of the arrival curve and is crucial for the delay bound since we have an affine arrival curve and a pseudoaffine service curve with $\rho_\pi^* \ge \rho$.

Lemma (FIFO Left-Over for Pseudoaffine Curves [1]). Let π be a pseudoaffine curve, i.e. $\pi = \delta_D \otimes \left[\bigotimes_{1 \le x \le n} \gamma_{\sigma_x, \rho_x} \right]$, and let $\alpha = \gamma_{\sigma,\rho}$ with $\rho_\pi^* \ge \rho$. If a FIFO-multiplexing node with service curve π serves two flows, 1 and 2, and flow 2 has α as an arrival curve, then the set of functions $\{\overline{E}(\pi, \alpha, s), s \ge 0\}$, with:

$$\overline{E}(\pi, \alpha, s) = \delta_{D + \bigvee_{1 \le i \le n} \left[\frac{\sigma - \sigma_i}{\rho_i} \right]^+ + s} \otimes \left[\bigotimes_{1 \le x \le n} \gamma_{\rho_x \{ s + \bigvee_{1 \le i \le n} \left[\frac{\sigma - \sigma_i}{\rho_i} \right]^+ - \frac{\sigma - \sigma_x}{\rho_x} \}, \rho_x - \rho} \right]$$

are pseudoaffine service curves for flow 1.
One can show that this set of service curves is "sufficient" in the sense that no service curves are skipped that might yield better delay bounds [1].

The presented lemmas concerning pseudoaffine curves will be used for the computation of the so-called Least Upper Delay Bound (LUDB) — we assume pseudoaffine (left-over) service curves and affine arrivals for which we can set up LPs that compute the minimal delay.

2 Feed-forward Network Analysis Principles

This chapter concerns different approaches to compute the worst case delay for a flow of interest (foi) in a feed-forward network. They are all heuristics in the sense that the bounds are not necessarily tight. The presented work implements a heuristic that combines nested / non-nested tandem decomposition, as well as the convolution of sub-tandems (see [1]) with (local) assignment of the free variables from the DiscoDNC.

Note that there exists also an approach that uses a Mixed Integer Linear Program (MILP) to compute the actual worst case delay, i.e. the delivered bounds are tight [4] [5].

2.1 TFA

The Total Flow Analysis (TFA) [2] for a foi computes the delay per node on foi's path and adds it to the end-to-end delay. Note that all flows (including foi) that cross the respective node are taken together (aggregated), i.e. "viewed" as one flow. Since the burst term of the foi is considered at every node, the Pay Bursts Only Once (PBOO) property is not fulfilled.

2.2 SFA

The Separated Flow Analysis (SFA) [2] for a foi computes the left-over service curve for every node on foi's path — i.e. "subtracts" the cross-flow arrivals. Then, the foi left-over service curves are convolved in order to get the end-to-end left-over which is then used to compute the delay bound. Thus, the PBOO property is achieved. Nevertheless, cross-flows that have more than one node with foi in common, are considered multiple times. Hence, the Pay Multiplexing Only Once (PMOO) effect is not exploited.

2.3 PMOO

The idea behind Pay Multiplexing Only Once (PMOO) [6] is to convolve (left-over) service-curves before cross-flows are "subtracted". The end-to-end left-over service curve is then used to compute the delay bound. Thus, the PBOO property is also attained.

Example. Server i has a strict service curve β_i and flow f_i has arrival curve α_i for $i \in \{1, 2, 3\}$(see also figure 2.1). The (PMOO) left-over service curve for flow f_1 is

$$\beta_{f1}^{PMOO} = [([(\beta_1 \otimes \beta_2) - \alpha_2]^+ \otimes \beta_3) - \alpha_3]^+$$

There exists scenarios in which SFA yields better bounds than PMOO [6]. Thus, it is useful to consider both approaches.

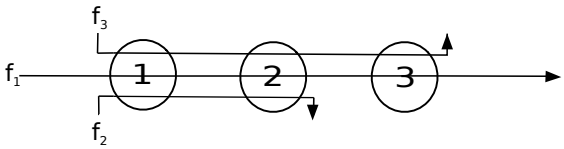

Figure 2.1: Simple tandem network

2.4 LUDB

In the following we will assume a tandem in which the foi traverses it and cross-flows start within the tandem. Note that the DiscoDNC tool also creates this "view" by aggregating flows with the same path within the tandem and computing the (aggregated) (output) arrival curve at the ingress node (see chapter 3 for more details). Moreover, we assume that nodes serve flows in FIFO order and that the service curve of node k is $\beta_k = \beta_{\theta^k,R^k}$ and the arrival curve of flow i is $\alpha_i = \gamma_{\sigma_i,\rho_i}$ [1]. In our conducted experiments we actually assumed a homogenous network with rate-latency, leaky-bucket curves, i.e. $\beta_i = \beta = \beta_{\theta,R}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho}$ (see chapter 6 for more details). For ease of notation, let the flow of interest traverse the nodes denoted by $1..N$ and identify flows by start and end node (i, j) with $1 \leq i \leq j \leq N$ [1]. Depending on the paths of the cross-flows, we differentiate between nested and non-nested tandems [1]. In a nested tandem, flows are either disjoint or are fully "overlapping", i.e. the path of a flow is a subpath of another flow. More formally, a nested tandem does not contain two flows (i, j), (h, k) with $i < h \leq j < k$ [1]. Figure 2.2 shows a nested tandem with three nodes and four flows (with f_1 as foi). Flow f_4 is nested within flow f_3. Moreover both flows and flow f_2 are nested within foi. Let the level of nesting $l(f_i)$ be defined as the number of flows into which it is nested (including itself) [1]. For example, $l(f_1) = 1$, $l(f_2) = l(f_3) = 2$ and $l(f_4) = 3$. The level of nesting of the tandem is the maximum level of nesting of one of its flows [1]. A non-nested tandem contains at least two flows (i, j), (h, k) with $i < h \leq j < k$, i.e. they have an interdependency. Figure 2.3 shows a non-nested tandem — f_2 and f_3 are interdependent.
As the other presented analysis techniques, we want to find an end-to-end delay bound for the flow of interest but it differs tremendously in the way how cross-flows are "subtracted" and especially how the free operators, i.e. the ones that we get from the FIFO multiplexing theorem, are chosen.

2.4.1 Nested Tandems

The LUDB computation for nested tandems creates a so-called nesting tree (see e.g. figure 2.4) whose nodes basically store the left-over service curve for the respective

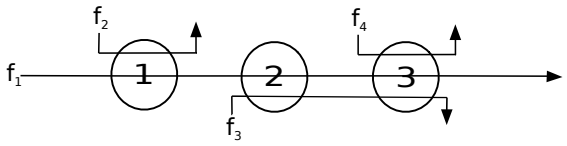

Figure 2.2: Simple nested tandem

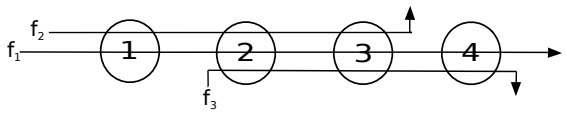

Figure 2.3: Simple non-nested tandem

(cross-) flow [1]. The root contains the foi's left-over service curve. Note that the left-over service curves contain the free parameters as variables, i.e. no specific values are set during the construction of the nesting tree. The reason for this is that once foi's left-over service curve is available, the lemma concerning delay bounds for pseudoaffine curves can be used and that delay term can be minimized — global optimization of the free parameters in the context of end-to-end delay bounding.
The following definitions will be used for constructing the nesting tree [1]:

$$S_{(h,k)} = \{(i,j) : h \leq i \leq j \leq k \text{ and } l(i,j) = l(h,k) + 1\},$$
$$C_{(h,k)} = \{l : h \leq l \leq k \text{ and } \forall (i,j) \in S_{(h,k)},\ l < i \text{ or } l > j\}$$

$S_{(h,k)}$ contains the set of flows which are directly nested into flow (h,k) and $C_{(h,k)}$ contains the nodes that are not crossed by the flows in $S_{(h,k)}$. If $S_{(h,k)} = \emptyset$, then $C_{(h,k)} = \{h, h+1, ..., k\}$. The nodes in the nesting tree are called t-nodes and there are two types of it — leaf t-nodes and non-leaf t-nodes [1]. Leaf t-nodes represent sets of nodes (with their respective convolved service curves) and non-leaf t-nodes represent flows with their respective left-over service curve. The root node contains the foi $(1, N)$. Every non-leaf node with flow content (h,k) has all flows $(i,j) \in S_{(h,k)}$ as children due to the construction of the left-over service curve (explained below). In addition, if $C_{(h,k)} \neq \emptyset$, (h,k) has one more child that represents $C_{(h,k)}$ — (h,k) gets the (convolved) service from the nodes in $C_{(h,k)}$ as there are no flows that are nested into (h,k) which cross them. Note that (h,k) also gets the left-over service from the other children nodes (explained below). Once the nodes have flows or (network-) nodes as content, the next step is to store the respective left-over service curves. For that, the nodes are visited in a post order fashion and the following is computed [1]:

(i) current node is a leaf t-node with $C_{(h,k)}$ as content:

$$\pi^{C_{(h,k)}} = \bigotimes_{j \in C_{(h,k)}} \beta^j$$

(ii) current node is a non-leaf t-node with (h,k) as content:

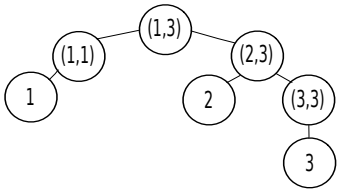

Figure 2.4: Nesting tree

$$\pi^{\{h,k\}} = \pi^{C_{(h,k)}} \otimes \left[\bigotimes_{(i,j)\in S_{(h,k)}} \overline{E}(\pi^{\{i,j\}}, \alpha_{(i,j)}, s_{(i,j)}) \right]$$

If $C_{(h,k)} = \emptyset$ then set $\pi^{C_{(h,k)}} = \beta_{0,+\infty}$, i.e. the neutral element concerning convolution. The computation for $\pi^{\{h,k\}}$ is the convolution of the (convolved) service curves of the nodes in $C_{(h,k)}$ (if existent) and the left-overs of the child non-leaf t-nodes. The latter takes the child left-over $\pi^{\{i,j\}}$ and the arrival curve of the child $\alpha_{(i,j)}$ (both available in the child node) and applies the lemma concerning FIFO left-over for pseudoaffine curves which results in $\overline{E}(\pi^{\{i,j\}}, \alpha_{(i,j)}, s_{(i,j)})$.
When the computation is done, the root contains foi's left-over service curve denoted by $\pi^{\{1,N\}}$. Let F_X be the set of cross-flows, then the LUDB for foi is the following [1]:

$$V = \min_{\substack{s_{(i,j)}\geq 0,\\ (i,j)\in F_X}} \{h(\alpha_{(1,N)}, \pi^{\{1,N\}}(s_{(i,j)} : (i,j) \in F_X))\}$$

This means that we have to find an assignment for the free variables such that the delay is minimal. With the made assumptions, it is easy to see that $\pi^{\{1,N\}}$ is pseudoaffine and with the affine arrival curve $\alpha_{(1,N)} = \gamma_{\sigma_{(1,N)},\rho_{(1,N)}}$ we can apply the delay lemma for pseudoaffine curves. More specifically,

$$\pi^{\{1,N\}} := \delta_D \otimes \left[\bigwedge_{1\leq x\leq n} \gamma_{\sigma_x,\rho_x} \right]$$

since $\pi^{\{1,N\}}$ is pseudoaffine. Applying the delay lemma and rewriting V yields [1]:

$$V = \min\{D + \bigvee_{1\leq x\leq n} \left[\frac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \right]^+ \}$$
$$s.t.$$
$$s_{(i,j)} \geq 0 \quad \forall (i,j) \in F_X$$

Note that the objective is not a linear function (due to the max operator). Hence, the idea is to set up an optimization problem for every possible case [1].

$$\forall x, 1 \leq x \leq n :$$

$$V_x = \min\{D + \frac{\sigma_{(1,N)} - \sigma_x}{\rho_x}\}$$
$$s.t.$$
$$s_{(i,j)} \geq 0 \quad \forall (i,j) \in F_X$$
$$\frac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \geq \frac{\sigma_{(1,N)} - \sigma_y}{\rho_y} \quad \forall y, 1 \leq y \leq n$$
$$\frac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \geq 0$$

We also need to take the case $[...]^+ = 0$ into account:

$$V_{n+1} = \min\{D\}$$
$$s.t.$$
$$s_{(i,j)} \geq 0 \quad \forall (i,j) \in F_X$$
$$\frac{\sigma_{(1,N)} - \sigma_y}{\rho_y} \leq 0 \quad \forall y, 1 \leq y \leq n$$

Then, we can compute the LUDB:

$$V = \min_{1 \leq x \leq n+1} \{V_x\}$$

Note that V_i may not be a LP because of recursive max-operators. In such a case we would further "split" the problem until only LPs are left.

Example. Figure 2.2 shows the nested tandem and the respective LUDB problem is as follows:

- $\overline{E}(\beta^3, \alpha_{(3,3)}, s_{(3,3)}) = \delta_{\theta^3 + \left[\frac{\sigma_{(3,3)}}{R^3}\right]^+ + s_{(3,3)}} \otimes \left[\gamma_{R^3\{s_{(3,3)} + \left[\frac{\sigma_{(3,3)}}{R^3}\right]^+ - \frac{\sigma_{(3,3)}}{R^3}\}, R^3 - \rho_{(1,3)}}\right]$
- $\beta^2 \otimes \overline{E}(\beta^3, \alpha_{(3,3)}, s_{(3,3)}) = \delta_{\theta^3 + \theta^2 + \left[\frac{\sigma_{(3,3)}}{R^3}\right]^+ + s_{(3,3)}} \otimes \left[\gamma_{0,R^2} \otimes \gamma_{R^3\{s_{(3,3)} + \left[\frac{\sigma_{(3,3)}}{R^3}\right]^+ - \frac{\sigma_{(3,3)}}{R^3}\}, R^3 - \rho_{(3,3)}}\right]$
- $\overline{E}(\beta^2 \otimes \overline{E}(\beta^3, \alpha_{(3,3)}, s_{(3,3)}), \alpha_{(2,3)}, s_{(2,3)})$
  $= \delta_{\theta^3 + \theta^2 + \left[\frac{\sigma_{(3,3)}}{R^3}\right]^+ + s_{(3,3)} + \vee \left[\frac{\sigma_{(2,3)}}{R^2}, \frac{\sigma_{(2,3)} - R^3\{s_{(3,3)} + \left[\frac{\sigma_{(3,3)}}{R^3}\right]^+ - \frac{\sigma_{(3,3)}}{R^3}\}}{R^3 - \rho_{(3,3)}}\right]^+ + s_{(2,3)}}$

$$\otimes \left[\gamma_{R^2\{s_{(2,3)}+\vee\left[\frac{\sigma_{(2,3)}}{R^2}, \frac{\sigma_{(2,3)}-R^3\{s_{(3,3)}+\left[\frac{\sigma_{(3,3)}}{R^3}\right]^+-\frac{\sigma_{(3,3)}}{R^3}\}}{R^3-\rho_{(3,3)}}\right]^+-\frac{\sigma_{(2,3)}}{R^2}\},R^2-\rho_{(2,3)}} \right.$$

$$\left. \otimes \gamma_{(R^3-\rho_{(3,3)})\{s_{(2,3)}+\vee\left[\frac{\sigma_{(2,3)}}{R^2}, \frac{\sigma_{(2,3)}-R^3\{s_{(3,3)}+\left[\frac{\sigma_{(3,3)}}{R^3}\right]^+-\frac{\sigma_{(3,3)}}{R^3}\}}{R^3-\rho_{(3,3)}}\right]^+-\frac{\sigma_{(2,3)}-R^3\{s_{(3,3)}+\left[\frac{\sigma_{(3,3)}}{R^3}\right]^+-\frac{\sigma_{(3,3)}}{R^3}\}}{R^3-\rho_{(3,3)}}\},*} \right]$$

$$* = R^3 - \rho_{(3,3)} - \rho_{(2,3)}$$

- $\overline{E}(\beta^1, \alpha_{(1,1)}, s_{(1,1)}) = \delta_{\theta^1+\left[\frac{\sigma_{(1,1)}}{R^1}\right]^+ + s_{(1,1)}} \otimes \left[\gamma_{R^1\{s_{(1,1)}+\left[\frac{\sigma_{(1,1)}}{R^1}\right]^+-\frac{\sigma_{(1,1)}}{R^1}\},R^1-\rho_{(1,1)}} \right]$

- Assuming that $\sigma_{(i,j)} \geq 0, R^k > 0 \Rightarrow \left[\frac{\sigma_{(i,j)}}{R^k}\right]^+ = \frac{\sigma_{(i,j)}}{R^k}$, yields the following left-over service curve for foi $(1,3)$:

$$\overline{E}(\beta^1, \alpha_{(1,1)}, s_{(1,1)}) \otimes \overline{E}(\beta^2 \otimes \overline{E}(\beta^3, \alpha_{(3,3)}, s_{(3,3)}), \alpha_{(2,3)}, s_{(2,3)})$$
$$= \delta_{\theta^1+\theta^2+\theta^3+\frac{\sigma_{(1,1)}}{R^1}+s_{(1,1)}+\frac{\sigma_{(3,3)}}{R^3}+s_{(3,3)}+\vee\left[\frac{\sigma_{(2,3)}}{R^2}, \frac{\sigma_{(2,3)}-R^3\cdot s_{(3,3)}}{R^3-\rho_{(3,3)}}\right]^+ + s_{(2,3)}}$$
$$\otimes \left[\gamma_{R^1\cdot s_{(1,1)}, R^1-\rho_{(1,1)}} \right.$$
$$\otimes \gamma_{R^2\{s_{(2,3)}+\vee\left[\frac{\sigma_{(2,3)}}{R^2}, \frac{\sigma_{(2,3)}-R^3\cdot s_{(3,3)}}{R^3-\rho_{(3,3)}}\right]^+-\frac{\sigma_{(2,3)}}{R^2}\},R^2-\rho_{(2,3)}}$$
$$\left. \otimes \gamma_{(R^3-\rho_{(3,3)})\{s_{(2,3)}+\vee\left[\frac{\sigma_{(2,3)}}{R^2}, \frac{\sigma_{(2,3)}-R^3\cdot s_{(3,3)}}{R^3-\rho_{(3,3)}}\right]^+-\frac{\sigma_{(2,3)}-R^3\cdot s_{(3,3)}}{R^3-\rho_{(3,3)}}\},R^3-\rho_{(3,3)}-\rho_{(2,3)}} \right]$$

- Applying the delay bound lemma yields:

$$V = \min\{ \left[\theta^1 + \theta^2 + \theta^3 + \frac{\sigma_{(1,1)}}{R^1} + s_{(1,1)} + \frac{\sigma_{(3,3)}}{R^3} + s_{(3,3)} + \vee \left[\frac{\sigma_{(2,3)}}{R^2}, \frac{\sigma_{(2,3)}-R^3\cdot s_{(3,3)}}{R^3-\rho_{(3,3)}}\right]^+ + s_{(2,3)} \right]$$
$$+ \vee \left[\frac{\sigma_{(1,3)}-R^1\cdot s_{(1,1)}}{R^1-\rho_{(1,1)}}, \frac{\sigma_{(1,3)}-R^2\{s_{(2,3)}+\vee\left[\frac{\sigma_{(2,3)}}{R^2}, \frac{\sigma_{(2,3)}-R^3\cdot s_{(3,3)}}{R^3-\rho_{(3,3)}}\right]^+-\frac{\sigma_{(2,3)}}{R^2}\}}{R^2-\rho_{(2,3)}}, \right.$$
$$\left. \frac{\sigma_{(1,3)}-(R^3-\rho_{(3,3)})\{s_{(2,3)}+\vee\left[\frac{\sigma_{(2,3)}}{R^2}, \frac{\sigma_{(2,3)}-R^3\cdot s_{(3,3)}}{R^3-\rho_{(3,3)}}\right]^+-\frac{\sigma_{(2,3)}-R^3\cdot s_{(3,3)}}{R^3-\rho_{(3,3)}}\}}{R^3-\rho_{(3,3)}-\rho_{(2,3)}} \right]^+ \}$$

$$s.t. \quad s_{(i,j)} \geq 0 \quad \forall (i,j) \in \{(1,1),(2,3),(3,3)\} = F_X$$

There are 12 $(= 3 \cdot 4)$ possible V_i , e.g.

$$V_1 = \min\{\left[\theta^1 + \theta^2 + \theta^3 + \frac{\sigma_{(1,1)}}{R^1} + s_{(1,1)} + \frac{\sigma_{(3,3)}}{R^3} + s_{(3,3)} + s_{(2,3)} + \frac{\sigma_{(2,3)}}{R^2}\right] + \frac{\sigma_{(1,3)} - R^1 \cdot s_{(1,1)}}{R^1 - \rho_{(1,1)}}\}$$

$$s.t. \quad \frac{\sigma_{(2,3)}}{R^2} \geq \frac{\sigma_{(2,3)} - R^3 \cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}}$$

$$\frac{\sigma_{(2,3)}}{R^2} \geq 0 \text{ (fulfilled due def.)}$$

$$\frac{\sigma_{(1,3)} - R^1 \cdot s_{(1,1)}}{R^1 - \rho_{(1,1)}} \geq \frac{\sigma_{(1,3)} - R^2 \cdot s_{(2,3)}}{R^2 - \rho_{(2,3)}}$$

$$\frac{\sigma_{(1,3)} - R^1 \cdot s_{(1,1)}}{R^1 - \rho_{(1,1)}} \geq \frac{\sigma_{(1,3)} - (R^3 - \rho_{(3,3)})\{s_{(2,3)} + \frac{\sigma_{(2,3)}}{R^2} - \frac{\sigma_{(2,3)} - R^3 \cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}}\}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}}$$

$$\frac{\sigma_{(1,3)} - R^1 \cdot s_{(1,1)}}{R^1 - \rho_{(1,1)}} \geq 0$$

$$s_{(i,j)} \geq 0 \quad \forall (i,j) \in \{(1,1),(2,3),(3,3)\} = F_X$$

2.4.2 Non-Nested Tandems

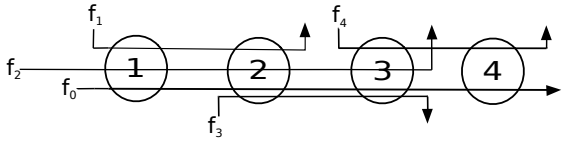

Figure 2.5: Another non-nested tandem example

First, the original idea for the LUDB in non-nested tandems is presented [1] (2010) / MSA (Multiple Subtandem Analysis) [7] (follow-up paper, 2012). The idea is to cut the tandem in a way that only nested subtandems have to be solved. Then, each subtandem is solved individually (LUDB nested tandem problem) and the resulting delay bounds are added up.
Flows (i,j) and (h,k) have an interdependency if $i < h \leq j < k$ and cutting any node in $[h, j+1]$ will resolve it [1]. E.g., f_1 and f_2 (see figure 2.5) have an interdependency and cutting any node in $[2,3]$ will resolve it. Figure 2.6 shows the cutted tandem — node 2 is used for cutting. Since the second subtandem "$2-3-4$" has interdependecies between the flows f_4, f_3 and f_2, at least one further cut is necessary, e.g. at node 4. Figure 2.7 shows the cutted tandem in which every subtandem is a nested one.

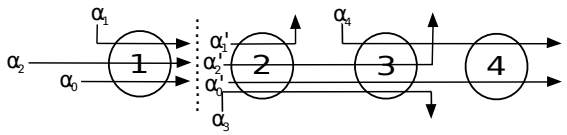

Figure 2.6: non-nested tandem cutted at server 2

In the following, we discuss how a non-nested tandem can be partitioned without trying out all possible cut-alternatives. As mentioned in the introductory part of LUDB, we assume that the foi, identified by $(1,N)$, crosses the nodes denoted by $1..N$. Let the tandem be defined by those nodes, i.e. $T = \{i : 1 \leq i \leq N\}$ and let $T_i = \{k : c_{i-1} \leq$

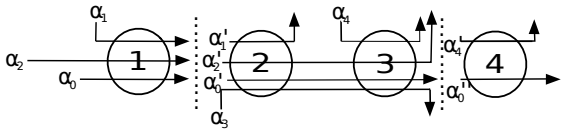

Figure 2.7: non-nested tandem cutted at server 2 and 4

$k \leq c_i - 1\}$ be a subtandem with $1 \leq i \leq m$ and $c_0 = 1, c_m = N + 1, c_i \leq c_{i+1}$ [1]. A set of cuts is defined as a set of nodes $SC = \{c_i : 1 \leq i \leq m\}$ with $\bigcup_{1 \leq i \leq m} T_i = T$ and $T_i \cap T_j = \emptyset \ \forall i, j = 1, ..., m$ with $i \neq j$ (no overlapping of subtandems) [1]. Moreover, only sets of cuts are relevant for which every subtandem is a nested one — those are called Nesting SC (NSC) [1]. Primary SC (PSC) is a subset of NSC in the sense that $\forall X \in PSC, \forall c_j \in X : \quad X \setminus \{c_j\}$ is not a NSC [1]. The intuition behind is that we do not want to introduce unnecessary cuts. An unnecessary cut would split a nested subtandem into two separate nested subtandems that would be solved individually and thus would yield a worse delay bound.
Example. Concerning figure 2.5 and omitting the trivial cuts (c_0 and c_m):

$$NSC = \{\{2,3\},\{2,4\},\{3\},\{2,3,4\},\{3,4\}\}$$
$$PSC = \{\{2,4\},\{3\}\}$$

It is important to note that both SCs in PSC need to be considered although they differ in the amount of cuts [1].

The flow and dependency matrix are quite useful when it comes to computing the PSC [1]. The $N \times N$ flow matrix F represents the path of the flows. More specifically, $F_{i,j} = 1$ if flow (i, j) exists. Then, this matrix can be used to determine the flow dependencies — flow (i, j) is interdependent with flows that are in the matrix "area"

$$[1, i-1] \times [i, j-1] \cup [i+1, j] \times [j+1, N]$$

(compare with the flow interdependency definition) [1]. If all dependencies are found, one can compute the dependency matrix with dimension $N \times d$ where d is the amount of found dependencies. Row k of the matrix indicates which flow interdependencies are resolved if a cut at k is placed. Then a set of cuts is a NSC if all dependencies in $D = \bigcup_{n=1}^{N} D_n$ are resolved [1].

Example. Concerning figure 2.5 the flow and dependency matrix F and DM, respectively, would be [1]:

$$F = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & & x & x & x \\ 2 & & & x & \\ 3 & & & & x \\ 4 & & & & \end{array}$$

DM		$(1,2)$ $(2,3)$	$(1,3)$ $(3,4)$	$(2,3)$ $(3,4)$
	1			
	2	x		
	3	x	x	x
	4		x	x

Algorithm 1 computes valid sets of cuts — sets of cuts that are candidates to be in PSC. More specifically, *computeValidSetsOfCutsInit* starts with every node (except node 1 since it is always in the cut). The method first checks if the current node, x, does resolve any dependencies. If this is not the case, then we can safely skip the node. Otherwise, we check if the node might already sever all dependencies in which case the node itself is already a valid set of cut, $\{x\}$ and thus would be added to the set VSC. In the other case, further nodes need to be added in order to sever all interdependencies, hence the method *computeValidSetsOfCuts* (with the respective parameters) is called. That method then checks for every node $x' \in \{x+1, ..., N\}$ if it is useful to add x' to the current set of cuts or not and where necessary if further nodes need to be added. If $\{x'-1, x'-2\} \subseteq SC$ then we do not need to cut at x' because every two-node tandem — in this case "$node_{x'-1} - node_{x'}$" — is nested anyway [1].

Algorithm 1 ComputeValidSetsOfCuts

```
1:  VSC = ∅;
2:  procedure computeValidSetsOfCutsInit
3:      for x = 2 to N
4:          if D_x = ∅
5:              continue;
6:          if D_x = D
7:              VSC = VSC ∪ {x};
8:          else
9:              computeValidSetsOfCuts({x}, D_x, x);
10: procedure computeValidSetsOfCuts(SC, SD, x)
11:     for x' = x+1 to N
12:         if D_x' ⊆ SD  or  {x'-1, x'-2} ⊆ SC
13:             continue;
14:         else
15:             SC' = SC ∪ {x'};
16:             SD' = SD ∪ D_x';
17:              if SD' = D
18:                  VSC = VSC ∪ SC';
19:              else
20:                  computeValidSetsOfCuts(SC', SD', x');
```

Example. Again consider figure 2.5. Let the flow dependencies be identified by d_1, d_2, d_3 (in the same order as in the dependency matrix). *computeValidSetsOfCutsInit* does the following:

- $x = 2 : SC = \{2\}$, $SD = \{d_1\}$, $x = 2$ (call *computeValidSetsOfCuts*)
  $x' = 3 : SD = D \Rightarrow VSC = \{\{(2, 3\}\}$
  $x' = 4 : SD = D \Rightarrow VSC = \{\{(2, 3\}, \{2, 4\}\}$
- $x = 3 : SC = \{3\}$, $SD = D \Rightarrow VSC = \{\{(2, 3\}, \{2, 4\}, \{3\}\}$ (no *computeValidSetsOfCuts* call since $D_x = D$)
- $x = 4 : SC = \{4\}$, $SD = \{d_2, d_3\}$ (*computeValidSetsOfCutsInit* terminates)

Returned set: $VSC = \{\{(2, 3\}, \{2, 4\}, \{3\}\}$

To get the set PSC, set of cuts that are unnecessary are removed from VSC. In the above example $\{2, 3\} \setminus \{2\} = \{3\} \in VSC$, hence $PSC = \{\{2, 4\}, \{3\}\}$.
When the PSCs are available, MSA computes a delay bound for every subtandem and adds them up, i.e. there will be an added delay bound for every SC in PSC and the minimal (delay) will be chosen (see also figure 2.7). [Note that cutting a tandem involves also computing arrival bounds for cutted flows which results in further LUDB computations, see chapter 3 for more details that also discusses how to deal with special cases such as crossflows have the same path as foi in the tandem.] The problem with this approach is that at every cut we have to pay for a burst, i.e. the PBOO property is not attained. Lenzini et al. presented in a follow-up paper from 2012 [7] another way to solve non-nested tandems called Single Tandem Analysis (STA). It also computes PSC but instead of solving a nested LUDB problem for every subtandem, the whole "pseudo-cutted" — foi is not cutted but cross-flows are arrival bounded at the "cuts" — tandem is "viewed" as a nested one (see also figure 2.8). As a result of this, we get an end-to-end service curve for foi which means that the PBOO property can be exploited. Concerning tightness, STA is superior compared to MSA [7].
The LUDB in our DiscoDNC tool (see also chapter 3) currently works very similar to STA (ignoring the global optimization). More specifically, it computes delays and foi left-over service curves for the subtandems and adds up the delays. Then, the pair (foi left-over service curve, added subtandem delays) with the minimal added subtandem delays is chosen — foi left-over service curve is the convolution of the left-over service curves of the subtandems. Finally, foi's end-to-end delay bound is computed using the foi left-over service curve in order to make use of the PBOO property.

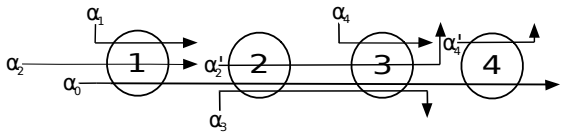

Figure 2.8: STA

2.4.3 Tightness of LUDB

[1] shows that there are examples in which the LUDB does not provide tight bounds. The idea is to extend a tandem T to another tandem $\overline{T}$ with worst case delay $\overline{W}$ that is not smaller than the worst case delay of T denoted by W. Let V and $\overline{V}$ be the LUDBs of T and $\overline{T}$. Then $V > \overline{V}$ implies $V > W$ since $V > \overline{V} \geq \overline{W} \geq W$, i.e. the LUDB V is not tight after all. A technique that is quite useful when it to comes to finding such a tandem $\overline{T}$ is called flow extension. In a nutshell, it prolongates flows, e.g. a flow $(j, N-1)$ could be prolongated to (j, N) [1].

3 Integration of LUDB into the Compositional Feed-forward Analysis

This chapter discusses the integration of the LUDB into the DiscoDNC tool as well as using it for arrival bounding in order to analyze any feed-forward network structure (and not only tandems). Moreover, some special cases are discussed that are not presented in chapter 2. Note that we can not just reuse the Deborah tool ([1]) because the licenses are incompatible and we would need to write a wrapper class.

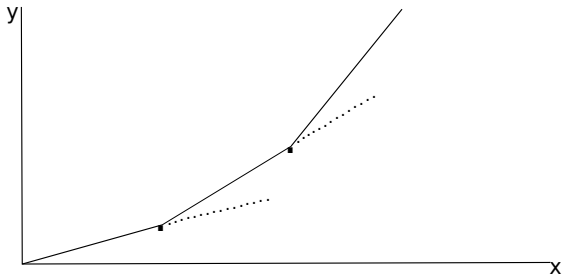

Figure 3.1: curve

3.1 Curves

A curve (*Curve.java*) is implemented as a list of linear segments. In general, each linear segment has a starting point and a slope — the starting point does not need to be included though (leftopen). For instance, see figure 3.1. *ArrivalCurve.java* and *ServiceCurve.java* are subclasses of it.

3.2 Operations

Needless to say, the DiscoDNC tool provides various methods for curve operations such as the addition of curves, convolution and deconvolution.

The following method will be used by $LUDB$:
ServiceCurve fifoMux(ServiceCurve service_curve, ArrivalCurve arrival_curve)
retrieves the ServiceCurve from the infinite set of service curves (FIFO-multiplexing) with the smallest latency and is used for SFA_FIFO as well as $LUDB$, i.e. (currently) the $LUDB$ does not perform a global optimization — with every FIFO-subtraction of the cross-flow(s), one specific assignment for the free parameter is used instead of making

that decision at the very end, i.e. when foi's end-to-end delay bound is computed.

3.3 LUDB and Arrival Bounding

[As mentioned earlier, our tool can perform the *LUDB* analysis for any feed-forward network structure. Therefore, given the path of foi, we need to compute the arrival bounds of flows that have an intersection with that path.]
In order to analyze a feed-forward network with *LUDB*,

new LudbAnalysis(someNetwork).performAnalysis(foi);

is called. The purpose of the **LudbAnalysis** class is to compute the end-to-end delay bound for a flow (**performAnalysis** is called) or for *LUDB* arrival bounding (**Set<ServiceCurve> getServiceCurves(Flow flow_of_interest, Path path, Set<Flow> flows_to_serve** is called explicitly by the **LudbArrivalBound** class). More specifically, **performAnalysis** calls the aforementioned **getServiceCurves** method and uses the left-over service curve to obtain foi's end-to-end left over service curve.

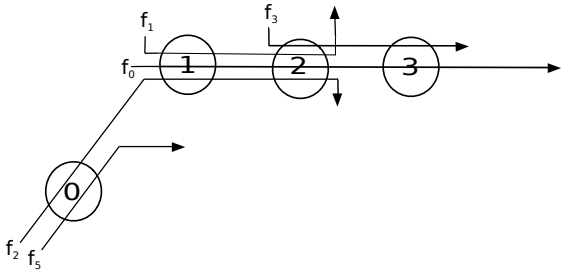

Figure 3.2: Network view before grouping

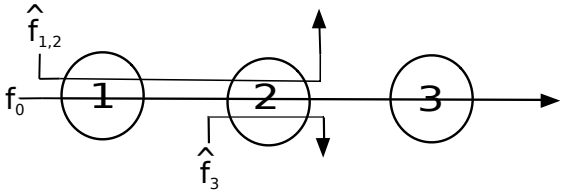

Figure 3.3: Network view after grouping

getServiceCurves groups cross-flows on foi's path and then for each group an (aggregated) (LUDB) arrival bound is computed. [The grouping is based on the path within foi's path.] Each group is substituted by a (substitute) flow with the respective path and arrival curve — see figures 3.2 and 3.3 with foi f_0 for a concrete example. After that, the flow matrix for the current "tandem" (internal view with foi and substitute cross-flows

within foi's path) is built in order to determine if it is nested or non-nested. In the nested case, it is possible that some crossflows have the same path as foi (especially due to arrival bounding calls). Therefore, the left over for the crossflow substitute with that path is computed which is then used to get foi's left over. In the other case, we do not have to make that distinction because it is already handeled by **LudbNonNested** (subtandems in which the subpath of a cross-flow has subtandem as a path deal with this analogously as aforementioned).

LudbNested(Path tandem, Flow flow_of_interest, List<Flow> flows) with flow_of_interest's path equals tandem and flows include foi as well as the (substitute) cross-flows. This class does the (non-nested) computations discussed in chapter 2, i.e. determining $S_{(h,k)}$ (stored in HashMap<Flow, ArrayList<Flow>> flow_directly_nested_flows_map), $C_{(h,k)}$ (stored in HashMap<Flow, LinkedList<Server>> flow_nodes_map) and using those maps to compute the nesting tree and foi's left-over service curve. Concerning the nesting tree, we created the **TNode** class which stores the content of the current node — either flow or list of servers as well as the left-over service curve and parent child nodes.

LudbNonNested(Network network, AnalysisConfig configuration, Path tandem, Flow flow_of_interest, List<Flow> flows, Map<Path,Set<Flow>> xtx_subpath_grouped) takes the network and configuration as paramter because they are needed to call the static arrival bounding method **computeArrivalBounds(Network network, AnalysisConfig configuration, Server server, Set<Flow> flows_to_bound, Flow flow_of_interest)**. Map<Path,Set<Flow>> xtx_subpath_grouped are the real flows per group — they are needed for arrival bounding in subtandems (see below for the reason behind this).
As mentioned in chapter 2, PSC (stored in ArrayList<ArrayList<Integer>> psc) is computed in order to only consider "useful" sets of cuts. For each set of cuts, the respective subtandems are created and for each subtandem the left-over service curve for foi is computed and then used for deriving an output arrival bound (for the next subtandem) as well as a subtandem-delay. The subtandem-delays for a set of cuts are added up and the set of cuts with the minimal added delay will be chosen. In order to use the PBOO property the respective end-to-end left over service curve (convolution of the left-overs from the subtandems) is used to derive foi's end-to-end delay. It is important to note that in a subtandem some cross-flows that were in different groups may have the same path within the current subtandem. Hence, new groups concerning the cross-flows in the current subtandem are computed as well as their aggregated arrival curve to avoid enforced segregation [8] [9]. Moreover, cross-flows that have the same path as foi in a subtandem can not be aggregated with foi (in a total flow analysis manner) because we use the convolution of the subtandems as end-to-end left-over (and eventually arrival bounding).

computeArrivalBounds(Server server, Set<Flow> flows_to_bound, Flow flow_of_interest) in **ArrivalBound** computes the aggregated arrival curve for flows_to_bound

at server. More specifically, for every (relevant) ingoing link **new LudbArrivalBound(network, configuration).computeArrivalBound(link, flows_to_bound, flow_of _interest)** is called which determines the common subpath of flows_to_bound ("starting" (backtracking) from link's destination server). Then a respective **LudbAnalysis** problem is solved to get the left-over service curve for the common subpath and with the arrival bound at the source of that subpath the output bound is finally comnputed.

4 Experimental Results

This chapter concerns some conducted experiments using the DiscoDNC in order to compare the different heuristic approaches for worst case delay bounding in feed-forward networks. As mentioned earlier, we assume a homogenous network with rate-latency, leaky-bucket curves, i.e. $\beta_i = \beta = \beta_{\theta,R}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho}$. For each network that appears in the following table, the actual network structure as well as the curves can be found in chapter 6. Please note that PMOO is only available for arbitrary multiplexing nodes.

Network	foi	LUDB	nested*	SFA_FIFO	SFA_ARB	PMOO
FF_3S_1SC_2F_1AC_2P	f0	54.479	y	54 23/48	74 4/9	74 4/9
FF_3S_1SC_2F_1AC_2P	f1	69.479	y	69 23/48	87 7/9	87 7/9
FF_4S_1SC_3F_1AC_3P	f0	74.792	y	74 19/24	97 2/9	97 2/9
FF_4S_1SC_3F_1AC_3P	f1	55.807	y	55 155/192	77 16/27	77 16/27
FF_4S_1SC_3F_1AC_3P	f2	49.167	y	49 1/6	65	65
FF_4S_1SC_4F_1AC_4P	f0	95.313	y	95 5/16	193 1/3	216 2/3
FF_4S_1SC_4F_1AC_4P	f1	71.875	y	71 7/8	172 1/2	172 1/2
FF_4S_1SC_4F_1AC_4P	f2	52.969	y	52 31/32	90 5/18	101 2/3
FF_4S_1SC_4F_1AC_4P	f3	87.813	y	87 13/16	186 2/3	190 5/6
TA_2S_1SC_1F_1AC_1P	f0	22.5	y	22.5	22.5	22.5
TA_2S_1SC_2F_1AC_2P	f0	42.917	y	42 11/12	50	50
TA_2S_1SC_2F_1AC_2P	f1	27.917	y	27 11/12	36 2/3	36 2/3
TA_2S_1SC_4F_1AC_1P	fn	25.5	y	34.5	82.5	60
TA_3S_1SC_2F_1AC_1P	fn	62.917	y	80 5/12	106 2/3	83 1/3
TR_3S_1SC_2F_1AC_2P	fn	47.917	y	47 11/12	56 2/3	56 2/3
TA_3S_1SC_3F_1AC_3P	f0	42.917	y	49 1/6	65	56 2/3
TA_3S_1SC_3F_1AC_3P	f1	33.229	y	34 167/192	48 1/3	52 1/27
TA_3S_1SC_3F_1AC_3P	f2	64.167	y	70 5/12	93 1/3	85
TA_4S_1SC_2F_1AC_2P	f0	82.917	y	89 1/6	105	96 2/3
TA_4S_1SC_2F_1AC_2P	f1	47.917	y	59 1/6	78 1/3	63 1/3
TR_7S_1SC_3F_1AC_3P	f0	102.083	y	165	289 1/6	177 1/2
TR_7S_1SC_3F_1AC_3P	f1	115.417	y	165	275 5/6	187 1/2
TR_7S_1SC_3F_1AC_3P	f2	110.0	y	147 1/2	252 1/2	177 1/2
TA_4S_1SC_5F_1AC_5P	f0	116.771	n	135.117	444.537	285.0
TA_4S_1SC_5F_1AC_5P	f1	49.583	y	61.25	185.0	160.0
TA_4S_1SC_5F_1AC_5P	f2	86.458	n	103.59375	383.889	245.0
TA_4S_1SC_5F_1AC_5P	f3	70.625	y	91.5625	398.333	286.667
TA_4S_1SC_5F_1AC_5P	f4	94.479	y	106.172	427.5	486.667

Network	foi	LUDB	nested*	SFA_FIFO	SFA_ARB	PMOO
TA_6S_1SC_5F_1AC_5P	f0	181.212	n	271.624	766.627	445.0
TA_6S_1SC_5F_1AC_5P	f1	70.583	y	102.0	339.167	265.0
TA_6S_1SC_5F_1AC_5P	f2	156.983	n	218.240	651.933	405.0
TA_6S_1SC_5F_1AC_5P	f3	171.367	n	273.600	820.156	495.0
TA_6S_1SC_5F_1AC_5P	f4	88.0	y	122.648	486.406	461.25
FF_7S_1SC_7F_1AC_7P	f0	166.354	n	194.180	543.981	415.0
FF_7S_1SC_7F_1AC_7P	f1	49.583	y	61.25	185.0	160.0
FF_7S_1SC_7F_1AC_7P	f2	86.458	n	103.594	383.889	245.0
FF_7S_1SC_7F_1AC_7P	f3	70.625	y	91.563	398.333	286.667
FF_7S_1SC_7F_1AC_7P	f4	94.479	y	106.172	427.5	486.667
FF_7S_1SC_7F_1AC_7P	f5	102.674	n	140.410	609.907	237.5
FF_7S_1SC_7F_1AC_7P	f6	57.943	n	64.424	360.278	197.5
FF_10S_1SC_9F_1AC_9P	f0	236.439	n	263.672	946.389	581.667
FF_10S_1SC_9F_1AC_9P	f1	49.583	y	61.25	185.0	160.0
FF_10S_1SC_9F_1AC_9P	f2	86.458	n	103.594	383.889	245.0
FF_10S_1SC_9F_1AC_9P	f3	70.625	y	91.563	398.333	286.667
FF_10S_1SC_9F_1AC_9P	f4	94.479	y	106.172	427.5	486.667
FF_10S_1SC_9F_1AC_9P	f5	187.498	n	246.168	2110.679	565.0
FF_10S_1SC_9F_1AC_9P	f6	107.674	n	161.348	1151.667	480.0
FF_10S_1SC_9F_1AC_9P	f7	176.724	n	231.397	2141.235	823.889
FF_10S_1SC_9F_1AC_9P	f8	166.602	n	199.703	1813.194	1121.667

Note: Results rounded to 3 decimal places. * no, if at least one non-nested "tandem" is computed (might also happen at arrival bounding for a crossflow).

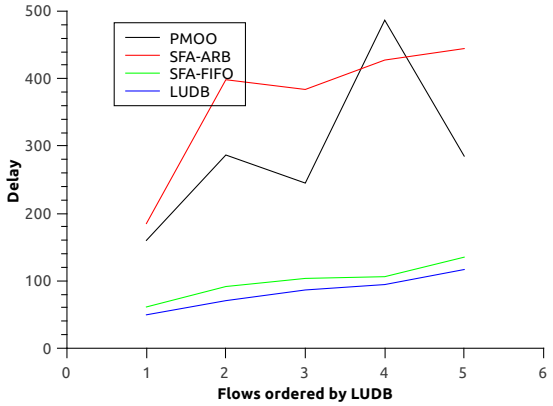

Figure 4.1: Plot for TA_4S_1SC_5F_1AC_5P

Following observations can be made (see also plots 4.1 4.2)

- SFA_ARB is not superior compared to $PMOO$ and vice versa (e.g. `TA_3S_1SC_3F_1AC_3P`)

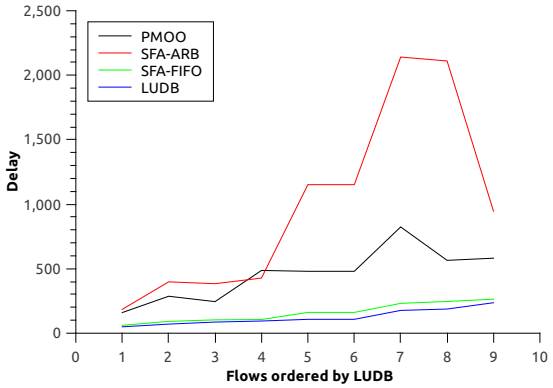

Figure 4.2: Plot for FF_10S_1SC_9F_1AC_9P

as already discussed in chapter 2

- In general, the FIFO analyses — $LUDB$ and SFA_FIFO — yield better results than the arbitrary ones, mainly because arbitrary implicitly sets the free parameters to zero while the FIFO analyses make an explicit choice using the fifoMux method (see also chapter 3)

- Comparing $LUDB$ with SFA_FIFO is the most interesting part. As we can see, there are a few cases in which both methods deliver exactly the same worst case delay. In the other cases, $LUDB$ seems to be superior than SFA_FIFO.
  For instance, regarding `FF_3S_1SC_2F_1AC_2P` (see chapter 6), the delay bounds for f_0 and f_1 are the same. More specifically, considering the foi f_0 both analysis principles compute the left-over service curve in the same way:

$$\beta^{l.o.f_0} = (\beta_0 \ominus^{FIFO} \alpha_1) \otimes (\beta_2 \ominus^{FIFO} \alpha_1^2)$$
$$\alpha_1^2 = \alpha_1 \oslash ((\beta_0 \ominus^{FIFO} \alpha_0) \otimes \beta_1)$$

  The left-over service curve for f_1 is analogously computed.

  Cases in which $LUDB$ is superior than SFA_FIFO can be explained with the `TA_3S_1SC_3F_1AC_3P` network (see chapter 6). The left-over service curve for foi f_2 is computed as follows:

$$\beta_{LUDB}^{l.o.f_2} = ((\beta_0 \otimes \beta_1) \ominus^{FIFO} \alpha_0) \otimes (\beta_2 \ominus^{FIFO} \alpha_1)$$
$$\beta_{SFA\_FIFO}^{l.o.f_2} = (\beta_0 \ominus^{FIFO} \alpha_0) \otimes (\beta_1 \ominus^{FIFO} \alpha_0^1) \otimes (\beta_2 \ominus^{FIFO} \alpha_1)$$
$$\alpha_0^1 = \alpha_0 \oslash (\beta_0 \ominus^{FIFO} \alpha_2)$$

As we can see, SFA_FIFO computes an arrival bound at server 1 for flow f_0, α_0^1, although no cuts are necessary in this scenario because it is a nested tandem.

In general, SFA_FIFO cuts at every server, i.e. computes delay bounds for the cross-flows although it might not be necessary to do so. As opposed to this, $LUDB$ only computes the arrival bounds of cross-flows when a cut is necessary.

5 Conclusion

The discussed LUDB analysis principle for FIFO-multiplexing feed-forward networks which was integrated into the DiscoDNC tool uses local optimization (see chapter 3) of the free parameters (instead of global as discussed in [1]). However, it can deal with any feed-forward network strucutres and is not limited to tandems. From the conducted experiments we observed and discussed that the fifo analyses yield better results than the arbitrary ones and LUDB only cuts when necessary, thus is able to find better bounds than SFA_FIFO which cuts at every server. It should also be noted that in some cases both analyses actually do the same and that the LUDB does not guarantee tight bounds as discussed in chapter 2.

6 Appendix

FF_3S_1SC_2F_1AC_2P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
FF_4S_1SC_3F_1AC_3P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
FF_4S_1SC_4F_1AC_4P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
TA_2S_1SC_1F_1AC_1P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{10,10}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
TA_2S_1SC_4F_1AC_1P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{10,10}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{10,2}$
TA_3S_1SC_2F_1AC_1P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
TR_3S_1SC_2F_1AC_2P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
TA_3S_1SC_3F_1AC_3P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
TA_4S_1SC_2F_1AC_2P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
TR_7S_1SC_3F_1AC_3P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
TA_4S_1SC_5F_1AC_5P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
TA_6S_1SC_5F_1AC_5P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,25}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
FF_7S_1SC_7F_1AC_7P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$
FF_10S_1SC_9F_1AC_9P: $\beta_i = \beta = \beta_{\theta,R} = \beta_{20,20}$ and $\alpha_i = \alpha = \gamma_{\sigma,\rho} = \gamma_{25,5}$

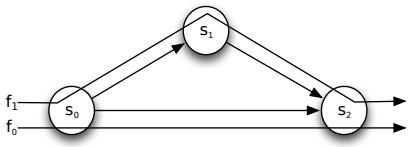

Figure 6.1: FF_3S_1SC_2F_1AC_2P

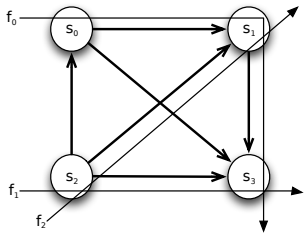

Figure 6.2: FF_4S_1SC_3F_1AC_3P

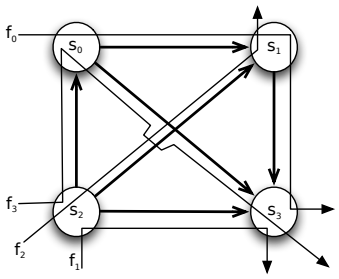

Figure 6.3: FF_4S_1SC_4F_1AC_4P

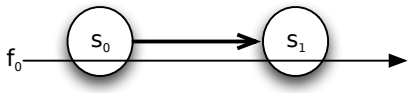

Figure 6.4: TA_2S_1SC_1F_1AC_1P

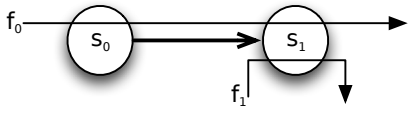

Figure 6.5: TA_2S_1SC_2F_1AC_2P

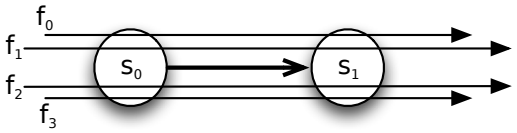

Figure 6.6: TA_2S_1SC_4F_1AC_1P

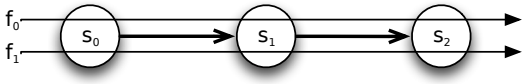

Figure 6.7: TA_3S_1SC_2F_1AC_1P

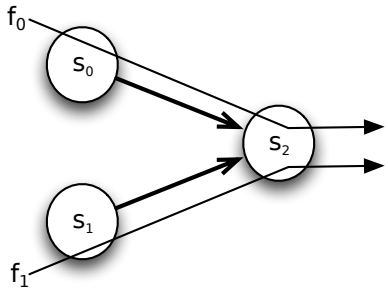

Figure 6.8: TR_3S_1SC_2F_1AC_2P

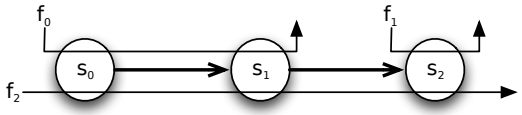

Figure 6.9: TA_3S_1SC_3F_1AC_3P

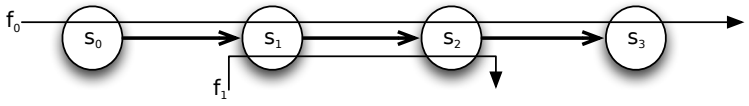

Figure 6.10: TA_4S_1SC_2F_1AC_2P

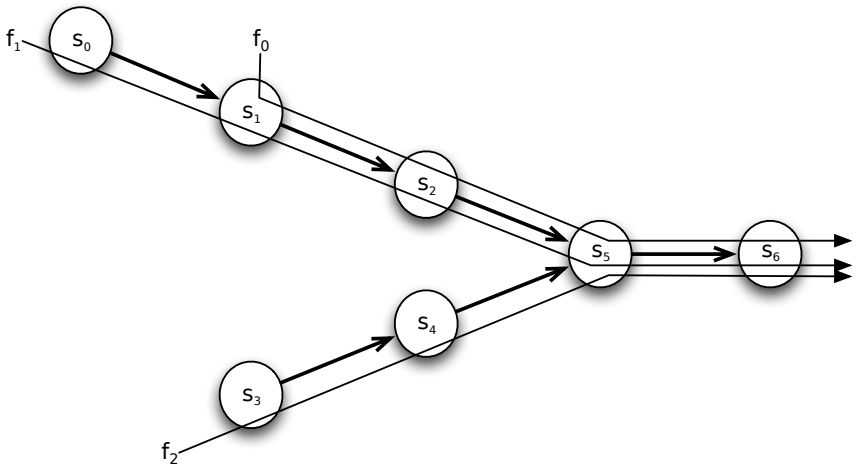

Figure 6.11: TR_7S_1SC_3F_1AC_3P

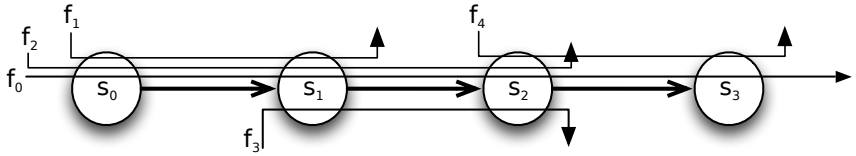

Figure 6.12: TA_4S_1SC_5F_1AC_5P

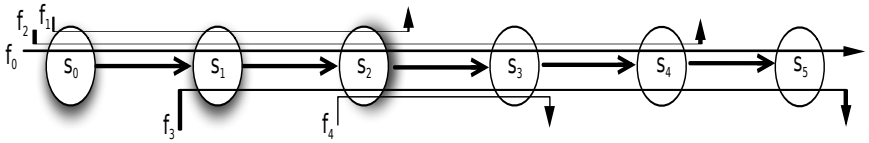

Figure 6.13: TA_6S_1SC_5F_1AC_5P

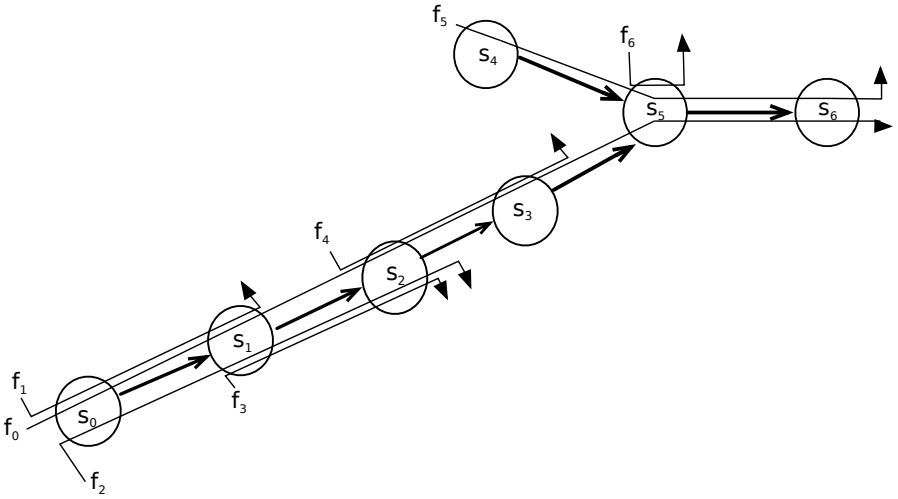

Figure 6.14: FF_7S_1SC_7F_1AC_7P

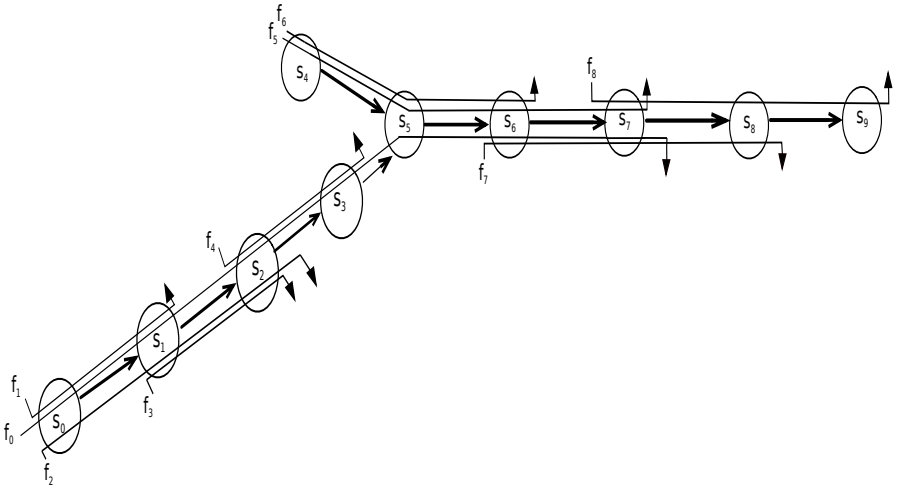

Figure 6.15: FF_10S_1SC_9F_1AC_9P

References

[1] L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea, "Computation and tightness assessment of end-to-end delay bounds in fifo-multiplexing tandems," Dipartimento di Ingegneria dell'Informazione, University of Pisa, Italy, Tech. Rep., May 2010.

[2] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Online version of the book, Springer, 2012.

[3] A. V. Bemten and W. Kellerer, "Network calculus: A comprehensive guide," Communication Networks, Technical University Munich, Germany, Tech. Rep., October 2016.

[4] A. Bouillard and G. Stea, "Exact worst-case delay in fifo-multiplexing feed-forward networks," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1387–1400, October 2015.

[5] ——, "Exact Worst-Case Delay for FIFO-Multiplexing Tandems," in *Proceedings of the 6th International Conference on Performance Evaluation Methodologies and Tools (ValueTools 2012)*, October 2012.

[6] J. B. Schmitt, F. A. Zdarsky, and M. Fidler, "Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch..." in *Proc. IEEE INFOCOM*, April 2008, pp. 1669–1677.

[7] L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea, "Numerical analysis of worst-case end-to-end delay bounds in fifo tandem networks," *Springer Real-Time Systems Journal, 48(5)*, p. 527–569, 2012.

[8] S. Bondorf and J. B. Schmitt, "Should network calculus relocate? an assessment of current algebraic and optimization-based analyses," in *Proceedings of the 13th International Conference on Quantitative Evaluation of Systems (QEST 2016)*, August 2016.

[9] S. Bondorf, P. Nikolaus, and J. B. Schmitt, "Quality and cost of deterministic network calculus – design and evaluation of an accurate and fast analysis," in *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2017)*, 2017.