Bachelor's Thesis

Modelling and Analysis of Timing Constraints of an Industrial Control System

by

Malte Schütze

29th September 2017

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

University of Kaiserslautern
Department of Computer Science
Distributed Computer Systems Lab

Examiner:	Prof. Dr.-Ing. Jens B. Schmitt
Supervisor:	Dipl.-Ing. (FH) Mathias Kreider

Abstract

In high-risk industrial systems, such as fly-by-wire or particle accelerator control systems, it is imperative to give upper bounds on the end-to-end delay of network messages in order to verify that each message reaches its target in time. Network calculus is a mathematical framework for verification of these constraints, though tools are usually limited in the type of input functions they accept. In this thesis, I developed and compared several methods to derive such constrained functions from more intuitive input formats.

In industriellen Hochrisikosystemen wie Fly-By-Wire oder Kontrollsystemen von Teilchenbeschleunigern ist es unabdingbar, obere Schranken für die Ende-zu-Ende-Verzögerungen von Netzwerknachrichten zu geben, um zu verifizieren dass jede Nachricht ihr Ziel rechtzeitig erreicht. Network calculus ist ein mathematisches Gerüst zur Verifikation solcher Schranken, obwohl Werkzeuge oft darin beschränkt sind, welche Art von Eingabefunktionen sie akzeptieren. In dieser Arbeit entwickelte und verglich ich mehrere Methoden um solch eingeschränkte Funktionen von intuitiveren Eingabeformaten herzuleiten.

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit mit dem Thema "Modelling and Analysis of Timing Constraints of an Industrial Control System" 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)

Acknowledgements

I'd like to thank Paul Nikolaus, Michael Beck, Mathias Kreider and Steffen Bondorf for fruitful discussions, guidance and for proofreading the many revisions that my thesis went through.

Contents

1. Introduction

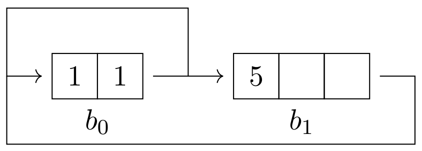
At first glance, control systems for nuclear power plants, fly-by-wire and particle accelerators seem to be very different systems. However, they share the following property: If they fail, consequences can be catastrophic. Verification of safety-critical non-functional aspects is therefore an important part of the development and operation of such facilities (e.g. [1], [2], [3]).

Part of this verification is the analysis of message delays to ensure each message reaches its target in time. *Network Calculus* [4] is a set of methods to help with, or even provide in first place, these verifications. In particular, tools such as the DiscoDNC [5] have been developed to support analysis of given models and computation of performance bounds for large networks. However, there are usually restrictions on what form of modelling functions are accepted. In particular, the DiscoDNC is only able to work with arrival curves represented as concave hulls consisting of piecewise affine segments [6].

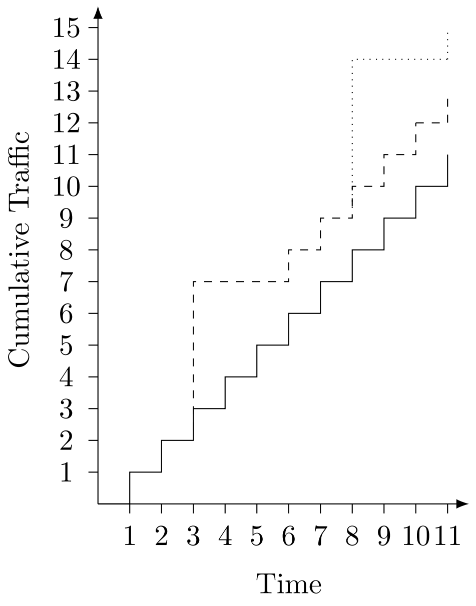
The *GSI Helmholtz Centre for Heavy Ion Research* is a state-of-the-art particle accelerator facility in Germany. Its largest component, the *Facility for Antiproton and Ion Research (FAIR)*, will contain 3.5 kilometres of tubes [7], more than 2000 endpoints [8] and several kilometres of cables when construction is finished. Part of the development of FAIR has been the introduction of a highly accurate, low-latency control system system employing *White Rabbit* [9] timing technology. At any given point, it is required that any message reaches its target within 500 microseconds. To verify that these constraints hold, all possible message sequences for an experiment need to be verified. For this, traffic specifications in a graph-based format as shown in Figure 1.1a were introduced.

A specification is split up in multiple blocks, e.g. b_0 and b_1 in Figure 1.1a. Each block represents a deterministic sequence of messages of some fixed length. Messages inside a block have a size, representing an abstract measure of size (for example the number of bytes) and an offset from the beginning of the block. Therefore if a block is entered at time t, a message with offset τ is considered to be sent at time $t + \tau$. Program execution starts at the beginning of some block. At the end of each block, program execution has a choice between several other blocks. The block where execution continues is not fixed but restricted by the given edges. Overall program execution can therefore be considered to be non-deterministic (for example, see the several possible cumulative models of Figure 1.1b).

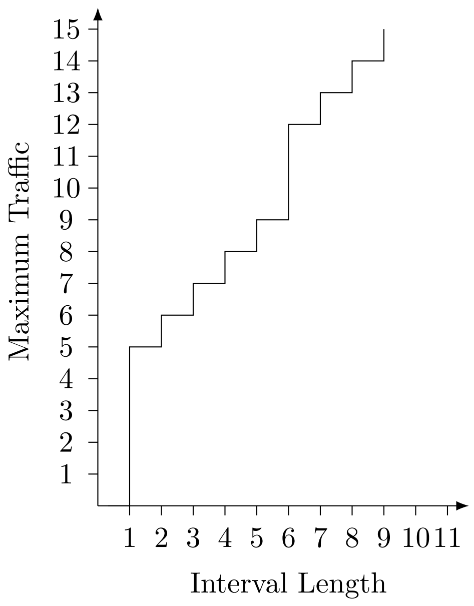
Additionally, each specification induces an arrival curve, denoted α (see Figure 1.1c). The value of the arrival curve at τ is an upper bound on the total traffic entering the system in any given interval of length τ [4], in our case by being generated by the control system based on the specification. Combined with a service curve β, which is a lower bound on the forwarding capabilities of the system (for example the minimum provided rate of a network interface), this allows the calculation of maximum backlog and delay.



(a) An example of a specification in graphical format. In this thesis, we always assume that edges leave blocks on the right and enter on the left.



(b) Potential cumulative models for Figure 1.1a.



(c) Arrival curve of Figure 1.1a.

Figure 1.1.: A specification and its curves.

This forms the basis of Network Calculus.

In this thesis, I first formalize the semantics of the GSI's graphical model in Chapter 2, Section 2.1. I then present and investigate several approaches to derive ultimately pseudo-periodic approximations of the arrival curves induced by the specification in Section 2.2, giving counterexamples for those that do not work and formal proof for those that do work. Afterwards, I present an algorithm to calculate a tight concave hull over an ultimately pseudo-periodic discrete function in Section 2.3. I present several optimizations to make the calculations feasible in practice in Chapter 3 and finally compare the different approaches with respect to accuracy and runtime in Chapter 4.

2. Theoretical Foundations

2.1. Formalization of specifications

Common Notations We use the following common definitions, listed here in the interest of clarity:

Notation	means
$\mathbb{N}_0$	The natural numbers, including 0: $\{0, 1, \ldots\}$
$\mathbb{N}_+$	The natural numbers, excluding 0: $\{1, 2, \ldots\}$
$\mathbb{R}_+$	The real numbers, restricted to strictly positive numbers
$\mathcal{P}(S)$	The power set of set S
$\|S\|$	Cardinality of S, the number of elements in S
S^*	Words over S: all ordered sequences that can be formed with elements from S
ϵ	The empty word

Let $d \in \mathbb{R}_+$ be the period, $c \in \mathbb{N}_0$ be the increment and $T \in \mathbb{N}_0$ be the offset of the period part. We call f an ultimately pseudo-periodic function [10] if

$$\exists T \in \mathbb{N}_0 : \exists (c, d) \in \mathbb{R} \times \mathbb{N}_0 : \forall t \geq T, f(t + d) = f(t) + c.$$

The advantage of an ultimately pseudo-periodic function is that the periodic part gives us the slope of the final segment of its concave hull, and for the limited non-periodic part, we can calculate the concave hull.

Specifications On an abstract level, a traffic specification is a set of blocks B and a successor relationship E over B. We can model this as a directed graph (B, E).

Each block contains an ordered sequence of messages being sent in that block $\mathcal{M} \subseteq M$, where M is the set of all messages. We assume that each message belongs to exactly one block.

We therefore define the following attributes for blocks $b \in B$:

$$\text{Duration}\, \delta \colon B \to \mathbb{N}_+ \qquad \text{(The duration of a block)}$$

$$\text{Messages}\, \mu \colon B \to \mathcal{P}(M) \qquad \text{(The sequence of messages in a block)}$$

We also define the following attributes for messages $m \in M$:

$$\begin{aligned} \text{size}\colon M &\to \mathbb{N}_0 && \text{(The size of a message)} \\ \text{offset}\colon M &\to \mathbb{N}_0 && \text{(The time offset of a message)} \\ & && \text{(from the beginning of a block)} \end{aligned}$$

We define the traffic generated by some block b as

$$\tau(b) \coloneqq \sum_{m \in \mu(b)} \text{size}(m)$$

and additionally define

$$\tau_{\leq n}(b) \coloneqq \sum_{\substack{m \in \mu(b) \\ \text{offset}(m) \leq n}} \text{size}(m)$$

as the maximum traffic generated up to offset $n \in \mathbb{N}_0$ (symmetrically for $<$, $\geq$, $>$).

Multiple messages can have the same offset, and we write $b[i]$ for messages m in $\mu(b)$ with offset i.

Message sequences as words Occasionally it makes sense to consider μ as a word over M, that is, an element of M^* instead. In this notation ($\mu(b) = m_1 \dots m_k$), we assume messages are ordered by their offsets, that is, $\text{offset}(m_i) \leq \text{offset}(m_{i+1})$. For messages with identical offsets, we assume an arbitrary but fixed ordering.

For a word $m \in M^*$, we write $m[0]$ ($m[i]$) for the first (i-th) message in m and $m[-1]$ ($m[-i]$) as the last message ($(i-1)$-th message from the back). We also write $m[i, j]$ for messages $m_i m_{i+1} \dots m_j$.

For some block b and its messages $\mu(b) = m_1 \cdots m_n$, we define the relative offset of messages with respect to the directly preceding message

$$\text{offset}_{\text{rel}}(m_i) \coloneqq \begin{cases} \text{offset}(m_i) - \text{offset}(m_{i-1}) & i > 0 \\ \text{offset}(m_i) & \text{otherwise.} \end{cases}$$

We also define the previous and next message to be

$$\begin{aligned} \text{next}(m_i) &\coloneqq m_{i+1} \text{ for } i < n \\ \text{previous}(m_i) &\coloneqq m_{i-1} \text{ for } i > 0. \end{aligned}$$

Flows We now have formalizations of blocks and messages, but in general we are not interested in sequences made purely from complete blocks, nor arbitrary sequences of messages. Instead what we are looking for are well-formed sequences of messages that adhere to in-block message ordering and the successor relationship between blocks, which we will call *flows.*

How do we define such sequences? The central part of a flow is uniquely characterized by the messages it contains[1], but we might need padding at the front and back when a flow starts or ends between two messages. We will call the padding at the front of the flow pad_l and the padding at the end pad_r.

We therefore call a 3-tuple $f = (\mathrm{pad}_\mathrm{l}, \sigma, \mathrm{pad}_\mathrm{r})$ from $(\mathbb{N}_0 \times \mathcal{M}^* \times \mathbb{N}_0)$ a *flow* if there is a decomposition $\sigma = \sigma_0 \cdots \sigma_n$ such that

$$\begin{aligned}
\exists b_s \in B\colon\ & \sigma_0 \text{ is a suffix of } \mu(b_s) \\
\exists b_e \in B\colon\ & \sigma_n \text{ is a prefix of } \mu(b_e) \\
\forall 0 < i < n\ \exists b_i \in B\colon\ & \sigma_i = \mu(b_i) && \text{(In-block message ordering)} \\
\forall i < n\colon\ & (b_i, b_{i+1}) \in E && \text{(Successor relationship between blocks)} \\
& \mathrm{pad}_\mathrm{l} < \mathrm{offset}_\mathrm{rel}(\sigma[0]) \\
& \mathrm{pad}_\mathrm{r} < \mathrm{offset}_\mathrm{rel}(\mathrm{next}(\sigma[-1])) && \text{(Padding)}
\end{aligned}$$

Note that this definition does not allow us to represent flows that end between to messages m, m' if $\mathrm{offset}(m) = \mathrm{offset}(m')$. However, this is not a great restriction for us, since we are generally interested in the maximum traffic generated in any given interval. This will never be achieved by excluding some concurrent messages from the flow.

We say a flow *begins on a block border* if $\mathrm{pad}_\mathrm{l} = \mathrm{offset}(\sigma[0])$ and *ends on a block border* if $\mathrm{pad}_\mathrm{r} = \delta(b_e) - \mathrm{offset}(\sigma[-1])$.

We now extend the measures of duration and traffic from blocks to flows by defining the traffic generated by f as

$$\tau(f) \coloneqq \sum_{m \in \sigma} \mathrm{size}(m)$$

and the duration of f as

$$\delta(f) \coloneqq t + \left(\sum_{0<i<n} \delta(b_i) \right) + u$$

where $t = \delta(b_s) - \mathrm{offset}(\sigma[0]) + \mathrm{pad}_\mathrm{l}$ and $u = \mathrm{offset}(\sigma[-1]) + \mathrm{pad}_\mathrm{r} + 1$.

We call $\mathcal{F}$ the set of all flows.

Maximum traffic The maximum traffic generated in an interval of length n is

$$\alpha(n) \coloneqq \max \{\ \tau(f) \mid f \in \mathcal{F} \wedge \delta(f) \leq n\ \}$$

and the flows with the maximum traffic are

$$\mathrm{flows}_\alpha(n) \coloneqq \operatorname*{arg\,max}_{\delta(f) \leq n}\ \tau(f).$$

By definition, α is an arrival curve, and it is also sub-additive: Consider a flow f of length $n+k$ with $f \in \mathrm{flows}_\alpha(n+k)$. We can decompose f into an initial part f_1 of length n and a final part f_2 with a duration of k. By definition we know that $\tau(f_1) \leq \alpha(n)$ and $\tau(f_2) \leq \alpha(k)$. Therefore $\tau(f) = \alpha(n+k) \leq \alpha(n) + \alpha(k)$.

[1]Unless it passes through no messages at all – but in this case, all such flows have the same behaviour. They are also typically of no interest to us when trying to approximate worst-case traffic.

For some subset $B' \subseteq B$, we consider the set of all flows beginning on the border of a block in B'. Of particular interest is the maximum traffic generated by flows of that form: Every flow that crosses at least one block border can be decomposed into three flows: one that ends on a block border, one that begins on a border and one that only consist of complete blocks. This can help us finding approximations for α.

Consider such a maximal flow f starting on the border of some block b. If the duration $n = \delta(f)$ is less than or equal to the duration of b, the maximum traffic is simply $\tau_{<n}(b)$ (recall: offsets of messages start at zero). Otherwise, we use the whole traffic $\tau(b)$ and the maximum of all flows of the remaining duration $n - \delta(b)$ of blocks that directly follow b, that is $(b, b') \in E$.

Put together, we define the maximum traffic of all flows of duration beginning on the border of any block $b \in B'$ as

$$\llbracket\alpha\rangle_{B'}(n) := \max_{b \in B'} \left\{ \begin{cases} \tau_{<n}(b) & \delta(b) \geq n \\ \tau(b) + \llbracket\alpha\rangle_{b' \in B:(b,b') \in E}(n - \delta(b)) & \text{otherwise.} \end{cases} \right\}$$

Consequently, the maximum traffic of all flows ending on a block border in B' is

$$\langle\alpha\rrbracket_{B'}(n) := \max_{b \in B'} \left\{ \begin{cases} \tau_{\geq \delta(b) - n}(b) & \delta(b) \geq n \\ \tau(b) + \langle\alpha\rrbracket_{b' \in B:(b',b) \in E}(n - \delta(b)) & \text{otherwise.} \end{cases} \right\}$$

For convenience we write $\llbracket\alpha\rangle$ for $\llbracket\alpha\rangle_B$ and $\langle\alpha\rrbracket$ for $\langle\alpha\rrbracket_B$ when we consider *all* blocks B.

Example As an example, consider the traffic specification in Figure 2.1:

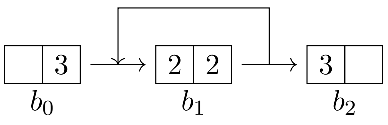


Figure 2.1.: Example specification for max-prefix and max-suffix calculation.

If we ask what the value of $\llbracket\alpha\rangle(1)$ (recall: this means $\llbracket\alpha\rangle_B(1)$) is, we consider all flows that start on any block border with a duration of at most 1. There are four such flows: $(0, \epsilon, 1)$ and $(1, \epsilon, 0)$ in b_0, $(0, b_1[0], 0)$ in b_1 and $(0, b_2[0], 0)$ in b_2. We see that the two empty flows in b_0 are functionally the same, and that the flow with maximum traffic is $f = (0, b_2[0], 0)$ with $\tau(f) = 3$. Thus $\llbracket\alpha\rangle(1) = 3$.

Consider now $\langle\alpha\rrbracket(2)$. There are three flows of duration 2 that end on a block border ($(1, b_0[1], 0)$, $(0, \mu(b_1), 0)$ and $(0, b_2[0], 1)$, and this time it is flow $f' = (0, \mu(b_1), 0)$ with a total traffic of 4 that gives us $\langle\alpha\rrbracket(2) = 4$.

Finally, consider $\langle\alpha\rrbracket_{\{b_1\}}(3)$. This time, we restrict our flows to those that end on b_1's right border. There are two such flows: $(0, b_1[1]\mu(b_1), 0)$ and $(0, b_0[1]\mu(b_1), 0)$ (no flow can leave b_2 and therefore no flow that must end in b_1 can pass through it). In this case it can be seen that $\langle\alpha\rrbracket_{\{b_1\}}(3) = 7$.

Alternative models From a specification (B, E), we can derive the *fully connected model* (B, E') with $E' := \{(b_1, b_2) \mid b_1, b_2 \in B\}$. We can also derive the *fully connected n-Block model* by building "super blocks" from sequential blocks in the original specification: we let our new model be (B', E') with $B' := \{b_1 b_2 \ldots b_n \mid (b_i, b_{i+1}) \in E\}$ and $E' := \{(b_1, b_2) \mid b_1, b_2 \in B'\}$.

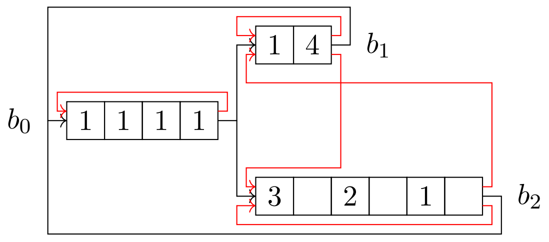


Figure 2.2.: Fully-connected 1-block model (original edges in black, new edges in red).

As it can easily be seen, each flow in the original specification is also a flow in both the fully connected model and the fully connected n-Block model. Therefore α can only grow, never shrink, by moving to a fully connected model and any curve derived from this model is a valid over-approximation of α. We can also see that increasing the number of consecutive blocks increases the accuracy of the model by filtering out some paths that exists in the fully-connected model but not in the original specification.

We can also derive the *rescaled model* (B', E) by identifying the minimum block length $\delta_{min} = \min_{b \in B} \{ \delta(b) \}$. We then rescale all blocks to that duration by letting $B' = \{b' \mid b \in B\}$ where $\delta(b') = \delta_{min}$. Additionally, we move the offsets of messages so their relative offset is the same: $\mu(b') = \{m' \mid m \in \mu(b)\}$ with $\text{offset}(m') = \lfloor \frac{\text{offset}(m)\, \delta_{min}}{\delta(b)} \rfloor$. Since this means that the absolute delay between messages is reduced, this is a valid over-approximation of α.

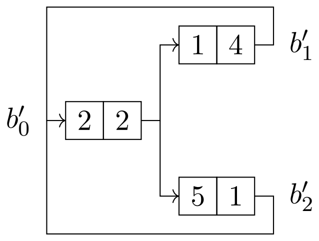


Figure 2.3.: Rescaled model based on the original specification in Figure 2.2.

Additionally these models can be combined, for example as follows:

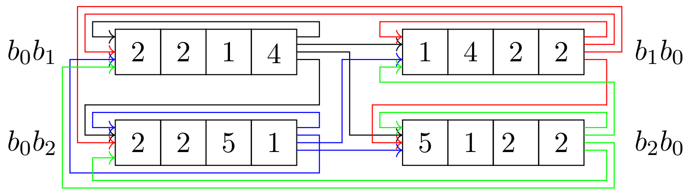


Figure 2.4.: Rescaled fully-connected 2-block model (each source block with a unique color).

Both these models will help us find approximations by lifting some restrictions on the original specification: The fully-connected model allows us to ignore the successor relationship between blocks and the rescaled model removes differences in block durations as a factor, making it easier to pick an optimal block.

2.2. Approaches

The crucial question is how to extract a concave hull from the arrival curve induced by the traffic specification. While the arrival curve is computable at all points, this alone does not give us enough information to construct a concave hull. However, we are able to compute ultimately pseudo-periodic over-approximations for the arrival curve (denoted α_{apx}), which makes constructing a concave hull feasible.

In the rest of this section, we discuss different approaches to generating such pseudo-periodic approximations, and present a simple algorithm for generating a concave hull over an ultimately pseudo-periodic discrete function in the next section.

2.2.1. Self-deconvolution

The typical approach to derive an arrival curve for a given sequence of arrivals is self-deconvolution of the cumulative input function R [4]. However, in our model, it is not entirely clear how to define such a cumulative function in the first place. For example, neither can we measure the system (since we want to verify delay bounds in advance), nor we can we simply take the maximum over all flows starting in some initial block b, that is, defining $R(i) = [\![\alpha\rangle_{\{b\}}(i)$. As an example, consider Figure 2.5 with initial block b_0.

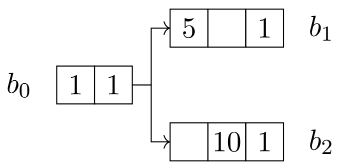


Figure 2.5.: A specification where self-deconvolution of the maximum of all flows produces incorrect results.

This specification and choice of initial block results in the values for R listed in Table 2.1:

t	$\llbracket\alpha\rangle_{\{b_0\}}(t)$
1	1
2	2
3	7
4	12
5	13

Table 2.1.: Cumulative input of Figure 2.5

Clearly the self-deconvolution of R provides $(R \oslash R)(1) = R(3+1) - R(3) = 5$. However, obviously the real value of $\alpha(1)$ is 10.

2.2.2. n-block sequences

Recall that we showed in Definition 5 that α is sub-additive. In particular, sub-additivity means that early steps impact the value of α much more than later steps. We therefore use the following approach:

For a traffic specification $S = (B, E)$, let the *(n-)sequential connections*

$$\begin{aligned} \mathrm{SEQ}(b_1, \dots, b_n) &:= (\ \{b_1, \dots, b_n\}\ ,\ \{(b_i, b_{i+1}) \mid i < n\} \cup \{(b_n, b_1)\}\ ) \\ \mathrm{NSEQ}_S(n) &:= \{\ \mathrm{SEQ}(b_{\varphi(1)}, \dots, b_{\varphi(n)})\ \mid\ (b_{\varphi(i)}, b_{\varphi(i+1)}) \in E\ \} \end{aligned}$$

where φ is some permutation of the numbers between 1 and $|B|$. This is a decomposition of one specification into several, each consisting of n sequential blocks. As an example consider Figure 2.6

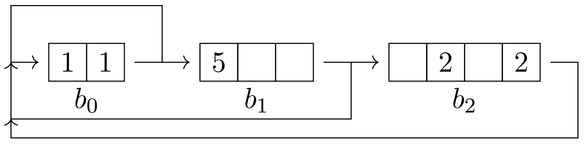


Figure 2.6.: Sample original specification for the n-block sequences.

which leads to the set of 2-block sequential connections visualized in Figure 2.7:

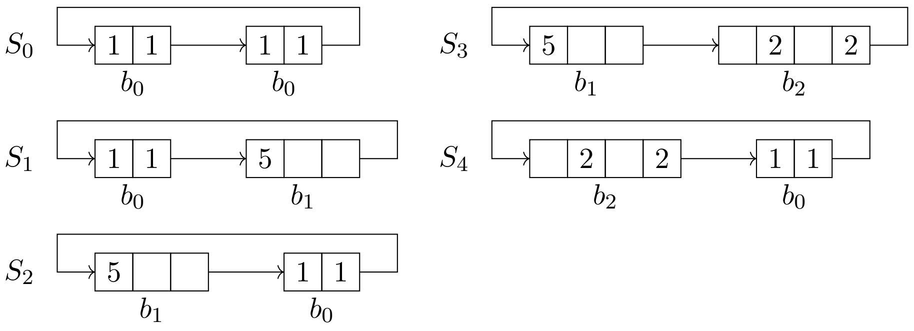


Figure 2.7.: 2-block approximations derived from Figure 2.6.

For each $S_i \in \text{NSEQ}_S(n)$, we now have an arrival curve α_i, and we define

$$\alpha_{\text{apx}}(c) = \max \{ \alpha_i(c) \}.$$

However, a small counterexample reveals that for any fixed number of sequential blocks $n \in \mathbb{N}_+$, there exists a specification such that for at least one interval length $c \in \mathbb{N}_0$, $\alpha(c) > \alpha_{apx}(c)$. In other words, there exists at least one flow in the original specification that produces more traffic than any flow of equal length in any of the n-block approximations. For example, consider Figure 2.8.

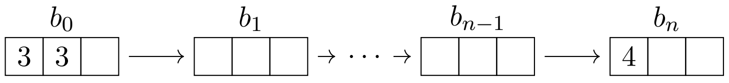


Figure 2.8.: A counterexample to the n-block sequence approximation.

This specification leads to the approximation listed in Table 2.2:

c	α_{apx}	via	α	via
0	0	–	0	–
1	4	$b_n[0]$	4	$b_n[0]$
2	6	$b_0[0,1]$	6	$b_0[0,1]$
...	...	...	...	...
3n-1	6	$b_0 \cdots b_{n-1}[0,1]$	6	$b_0 \cdots b_{n-1}[0,1]$
3n	6	$b_0 \cdots b_{n-1}$	7	$b_0[1,2] \cdots b_{n-1} b_n[0]$
3n+1	9	$b_0 \cdots b_{n-1} b_0[0]$	10	$b_0 \cdots b_{n-1} b_n[0]$

Table 2.2.: Approximation and actual arrival curve

In this case the problem is that only n consecutive blocks are considered. Therefore for durations less than $3n$ the approximation is accurate – the actual flow with the most traffic of duration $3n-1$ does start in b_0 to include the first two message and therefore ends in the second block of b_{n-1}. However, the maximum flow of duration $3n$ starts at the second message of b_0 and end at the first message of b_n, for a total traffic of 7. But by definition, there is no n-block sequence including both b_0 and b_n. Therefore the approximation fails to over-approximate the arrival curve.

Even when choosing n depending on the number of blocks in the specification, this approach will not work, as the counter example in Figure 2.9 shows.

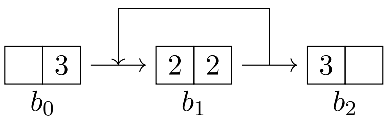
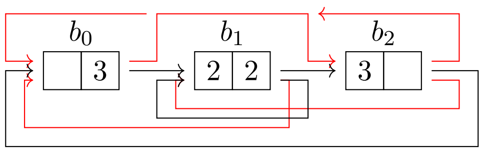


Figure 2.9.: A counterexample to the n-block approximation with specification-dependant n.

Any flow of even length $2k$ greater than or equal to 4 starts at the last message of b_0, passes $k-1$ times through b_1 and ends at the first message of b_2. Therefore, for any fixed number of sequential blocks n, this approximation will fail for intervals with length greater than $2n-1$, because at that point $n-1$ iterations through b_1 are required for maximum traffic. Together with the initial b_0 and the ending in b_2, this is a sequence of $n+1$ blocks, which cannot be represented by an n-block model.

2.2.3. n-block sequences in the fully-connected model

The problem with Figure 2.8 is the lack of direct links between blocks. Adding edges between b_0 and b_n would give a good over-approximation. However, even in the fully connected model, Figure 2.9 is a counterexample where for at least one c, $\alpha(c) > \alpha_{apx}(c)$. Consider also Figure 2.10, which visualizes the additional links introduced by the fully-connected model in Figure 2.9.



Implied edges in red

Figure 2.10.: Counterexample for fully-connected n-block sequences.

The specification in Figure 2.10 gives rise to the approximation and actual arrival curve listed in Table 2.3.

c	α_{apx}	via	α	via
0	0	–	0	–
1	3	$b_0[1]$	3	$b_0[1]$
2	6	$b_0[1]b_2[0]$	5	$b_0[1]b_1[0]$
3	7	$b_0[1]b_1$	7	$b_0[1]b_1$
...	...	...	...	...
$2n-2$	$4n+2$	$b_0[1]b_{1_1}b_{1_2}\cdots b_{1_{n-2}}b_2[0]$	$4n+2$	$b_0[1]b_{1_1}b_{1_2}\cdots b_{1_{n-2}}b_2[0]$
$2n-1$	$4n+3$	$b_0[1]b_{1_1}b_{1_2}\cdots b_{1_{n-2}}b_{1_{n-1}}$	$4n+3$	$b_0[1]b_{1_1}b_{1_2}\cdots b_{1_{n-2}}b_{1_{n-1}}$
$2n$	$4n+4$	$b_{1_1}b_{1_2}\cdots b_{1_{n-1}}b_{1_n}$	$4n+6$	$b_0[1]b_{1_1}b_{1_2}\cdots b_{1_{n-2}}b_{1_{n-1}}b_2[0]$

Table 2.3.: Approximation and arrival curve for counter-example

Consider the approximation for duration 2. In the actual specification, the maximum traffic is 5, either via the last message in b_0 and the first message in b_1, or via the last message in b_1 followed by the first in b_2. In the approximation, we added a connection from b_0 to b_2 so the maximum flow of duration 2 includes the last message in b_0 followed by the first message in b_2, for a total of 6. This is an over-approximation.

However, this does not change the fact that just as in the original specification, even in the fully connected specification the maximum flow of duration $2n$ passes through $n+1$ blocks – the last message of b_0 followed by $n-1$ passes through b_1 and finally the first message of b_2. No matter that this is a fully-connected model, we still cannot represent such a flow with a sequence of n (arbitrary) blocks.

2.2.4. Sub-additivity of the arrival curve

Using once more the sub-additive property of arrival curves, we can approximate α by observing that

$$\begin{aligned}\alpha(i \cdot k + n) &\leq \alpha(i \cdot k) + \alpha(n) \\ &\leq i \cdot \alpha(k) + \alpha(n).\end{aligned}$$

Recall that we are looking for ultimately pseudo-periodic approximations and that we defined such functions in Definition 1 to have a period d, an increment c and an offset T. Using this, we pick some threshold k and define α_{apx} to be an ultimately pseudo-periodic function with initial offset 0, period k and increment $\alpha(k)$ with $\alpha_{\text{apx}}(i) = \alpha(i)$ for $i < k$.

2.2.5. Approximation by rescaling in the fully-connected model

As noted in Section 2.1, we can rescale all blocks to a shorter duration without violating our worst-case bound. Let k be the duration of the shortest block, and assume we have a fully connected model with all block durations rescaled to this length. Each flow with a duration $\geq k$ touches at least one block boundary (that is, either begins or ends on a block boundary, or begins in one block and ends in another), and each flow with a duration $\geq 2k$ passes through at least one full block. Since all blocks have the same duration, there exists some block b_{max} with $\tau(b_{\text{max}}) \geq \tau(b)$ for all blocks b. Since we are in the fully connected model, we can replace each full block by b_{max} in the worst case.

We therefore define α_{apx} to be an ultimately pseudo-periodic function with initial offset k, period k and increment $\tau(b_{max})$, with the initial part of the function defined as follows:

$$\alpha_{\text{apx}}(i) = \begin{cases} \alpha(i) & i < k \\ \max_{0 \leq j \leq i} \{ \ \langle \alpha \rrbracket(j) + \llbracket \alpha \rangle(i-j) \ \} & k \leq i \leq 2k \end{cases}$$

A disadvantage of this approach is the high inaccuracy incurred if there is a large difference between the durations of the longest and shortest blocks. It is however possible to generalize this approach to the unscaled fully-connected model, as we will see in Sub-Section 2.2.6

2.2.6. Approximation without rescaling

We derive the unscaled approach by first focusing on prefix flows, that is, flows which begin on a block boundary. Later we generalize to all flows.

First, we define the *efficiency* of a block as $e(b) := \frac{\tau(b)}{\delta(b)}$. From this, we define

$$\begin{aligned} b_{\text{max}} &:= \arg\max_{b \in V} e(b) \\ \tau_{\text{max}} &:= \tau(b_{\text{max}}) \\ \delta_{\text{max}} &:= \delta(b_{\text{max}}) \\ e_{\text{max}} &:= e(b_{\text{max}}). \end{aligned}$$

Note that this is a different definition of $b_{\max}$ than in the previous section, since we are now interested in a different metric. Finally, we let the longest block be $b_{\text{longest}} := \arg\max_{b \in V} \delta(b)$ and its duration $\delta_{\text{longest}} := \delta(b_{\text{longest}})$.

The general idea is to cover some part of the flow by iterating b_{max} and the remainder by some function α' that we will derive later.

Consider a prefix flow $f := f_{\text{prefix}} b_1 f_{\text{suffix}}$ with a duration greater than δ_{longest}. This decomposition always holds: Since we start on a block boundary, a flow of duration greater than or equal to the length of the longest block must pass through at least one full block. Of all these full blocks passed, we pick an arbitrary one to be b_1 and let everything before b_1 be f_{prefix} and the remainder f_{suffix}. See also Figure 2.11 for a visualization:

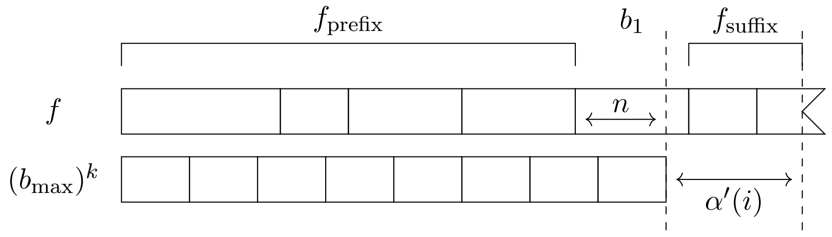


Figure 2.11.: A visualization of f and its decomposition.

As we can see, we can rewrite the duration of f as $k\,\delta_{\max} + i$, picking i to be minimal but greater than or equal to δ_{longest}. This ensures that $(b_{\max})^k$ ends inside a full block. We now pick b_1 to be the block that $(b_{\max})^k$ ends up in. We know the following about the durations of the parts of f:

$$\begin{aligned}
\delta(f_{\text{prefix}}) &= k \cdot \delta_{\max} - n \\
\delta(b_1) &= n + n' \\
\delta(f_{\text{suffix}}) &= i - n'.
\end{aligned}$$

We can over-approximate $\tau(f)$ as follows:

$$\begin{aligned}
& \tau(f_{\text{prefix}}) + \tau(b_1) + \tau(f_{\text{suffix}}) \\
= \quad & \tau(f_{\text{prefix}}) + (n + n')e(b_1) + \tau(f_{\text{suffix}}) \\
\leq \quad & \tau(f_{\text{prefix}}) + n \cdot e_{\max} + n' \cdot e_{\max} + \tau(f_{\text{suffix}}) && \text{(Definition of } e_{\max}) \\
\leq \quad & (k \cdot \delta_{\max} - n)e_{\max} + n \cdot e_{\max} + n' \cdot e_{\max} + \tau(f_{\text{suffix}}) \\
\leq \quad & k \cdot \tau_{\max} + n' \cdot e_{\max} + [\![\alpha\rangle(i - n').
\end{aligned}$$

We do not know the actual value of n', but we can approximate it again by taking the maximum over all possible values of n':

$$\alpha'(i) = \max_{0 \leq n' \leq i} \{ \ n' \cdot e_{\max} + [\![\alpha\rangle(i - n') \ \}$$

Therefore we can approximate $\tau(f)$ with

$$k \cdot \tau_{\max} + \alpha'(i).$$

We now consider arbitrary flows. Let f be a flow $f_{\text{prefix}} f_1 b f_2$ of duration $2\,\delta_{\text{longest}} + j$. We choose the decomposition so f_{prefix} is the part of the flow before the first block boundary, f_1 has a duration of less than j but $f_1 b$ has a duration greater than or equal to j. The remainder of the flow is then f_2.

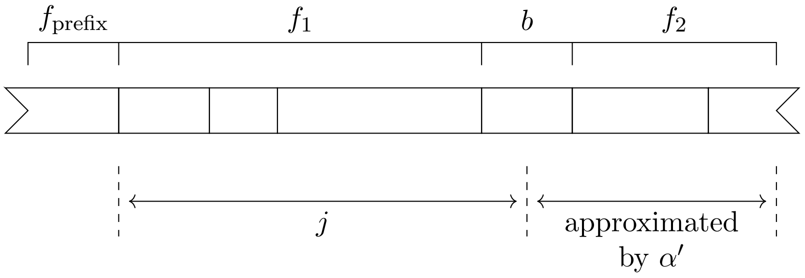


Figure 2.12.: A visualization of f and its decomposition.

This gives rise to the following (in-)equalities:

$$\begin{aligned} \delta(f_{\text{prefix}}) &\le \delta_{\text{longest}} \\ \delta(f_1) &= j - n \\ \delta(b) &= n + n' \\ \delta(f_2) &\le 2\,\delta_{\text{longest}} \end{aligned}$$

Since we pick f_{prefix} to be less than a full block, we know that $\delta(f_1 b f_2) \ge \delta_{\text{longest}} + j$. As we have shown above, we can rewrite this as $k\,\delta_{\max} + i$ with $i \ge \delta_{\text{longest}}$. We then have

$$\begin{aligned} \tau(f_1 b f_2) &= (j + n')e_{\max} + [\![\alpha\rangle(\delta(f_1 b f_2) - j - n') \\ &\le j e_{\max} + \alpha'(\delta(f_1 b f_2) - j) \end{aligned}$$

and we know that $\tau(f_{\text{prefix}}) \le \langle\alpha]\!](\delta(f_{\text{prefix}}))$ because it ends on a block border. However, we do not know what part of the duration is spent in f_{prefix} and what part is spent in f_2. Therefore, we once more use the maximum to over-approximate and define

$$\gamma = \max_{0 \le k \le \delta_{\text{longest}}} \{\ \langle\alpha]\!](k) + \alpha'(2\,\delta_{\text{longest}} - k)\ \}.$$

With this, we can finally define α_{apx} as pseudo-periodic function with initial offset $2\,\delta_{\text{longest}}$, period 1 and increment $e_{\max}$:

$$\alpha_{\text{apx}}(i) = \begin{cases} \alpha(i) & i < 2\,\delta_{\text{longest}} \\ i \cdot e_{\max} + \gamma & i \ge 2\,\delta_{\text{longest}}\,. \end{cases}$$

2.2.7. Summary

We have considered several approaches to generating ultimately pseudo-periodic approximations. The sub-additive approximation from Sub-Section 2.2.4 is very appealing due to its simplicity, but will always have inherent inaccuracy due to repeatedly incorporating the burst at the beginning of α. The rescaled approximation from Sub-Section 2.2.5 is also fairly simple, but will have a high degree of inaccuracy due to rescaling. Finally, the fully-connected approximation from Sub-Section 2.2.6 can be perfectly accurate, but is also very complex.

2.3. Concave hull of ultimately pseudo-periodic discrete functions

We now have an ultimately pseudo-periodic approximation of the arrival curve. The DiscoDNC however can only work with concave functions [6]. We therefore need to further process our function.

The concave hull h_f of a function f is a piecewise affine and concave function, where the endpoint of each affine segment is the initial point of the next segment and at all points it holds that $h_f(i) \geq f(i)$. In particular, we want a function h_f that intercepts f at as many points as possible to reduce the over-approximation introduced by the hull. If f is ultimately pseudo-periodic with parameters c, d, T (recall Definition 1), we can easily calculate such a tight concave hull.

Lemma 1. *The last affine segment of the concave hull h_f has a slope of $\frac{c}{d}$.*

Proof. Since f is ultimately pseudo-periodic, we know that for any $t \geq T$, the slope between t and $t+d$ in f is $\frac{c}{d}$.

Since we want to keep the inaccuracy introduced by the hull as small as possible, we also want to choose the slope of the final segment to be as small as possible: Any unnecessarily large slope will obviously introduce an absolute error that is proportional to the argument: If the final segment slope if $\frac{c+\varepsilon}{d}$ and at time i, $h_f(i) = f(i) + k$, then at time $i+d$, $h_f(i+d) = f(i+d) + k + d\varepsilon$.

Assume instead the last segment has a slope of $\frac{c-\varepsilon}{d}$ for some $\varepsilon > 0$ and for some $i, n \in \mathbb{N}_0$, $h_f(i) = f(i) + n$. Then we know that at point $x = i + \frac{nd}{\varepsilon}$, $h_f(x) = f(x)$ and at $x+d$, $h_f(x+d) = f(x+d) - \varepsilon$, violating our constraint that $h_f(x) \geq f(x)$.

Therefore, assume the last segment has a slope of $\frac{c}{d}$ and for each point $i < T + d$, $h_f(i) \geq f(i)$. The pseudo-periodicity of f implies that $h_f(i) \geq i$ also holds for $i \geq T+d$. Therefore the slope $\frac{c}{d}$ is the lowest slope a final segment can have.

As a visualization, refer to Figure 2.13.

□

Lemma 2. *The last affine segment of the concave hull has a y-offset of*

$$y_0 = \max \left\{ \ f(x) - x \cdot \frac{c}{d} \mid x < T + d \ \right\}.$$

Proof. Lemma 1 gives us a slope for the last segment, but with the restriction that $h_f(i) \geq f(i)$ for $i < T + d$. We therefore pick the highest point of f relative to the line given by $y = x\frac{c}{d}$ as the y-offset at $x = 0$. This means that at any point $i < T + d$ we have $i \cdot \frac{c}{d} + y_0 = i \cdot \frac{c}{d} + \max \{ \ f(x) - x \cdot \frac{c}{d} \mid x < T + d \ \} \geq i \cdot \frac{c}{d} + f(i) - i\frac{c}{d} \geq f(i)$. □

We know consider the other segments of h_f. Let $h_1, \dots h_n$ be the segments of the concave hull with intervals $[x_0, x_1), [x_1, x_2), \dots [x_{n-1}, \infty)$. By concavity we know that $\text{slope}(h_i) \geq \text{slope}(h_{i+1})$.

Lemma 3. *The last affine segment of the concave hull has an initial offset of*

$$x_{n-1} = \min \left\{ \ \arg\max \left\{ \ f(x) - x \cdot \frac{c}{d} \mid x \leq T + d \ \right\} \ \right\}.$$

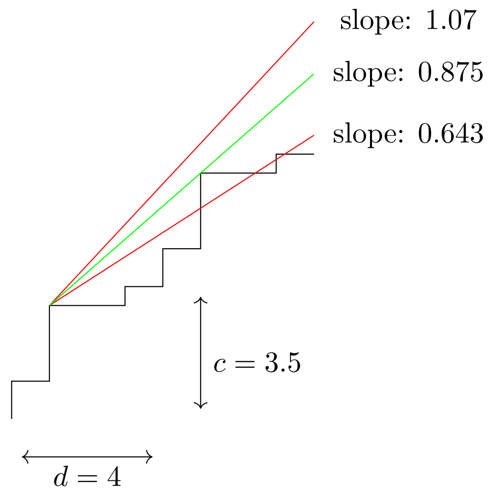


Figure 2.13.: A pseudo-periodic function with period 4 and increment 3.5.

Proof. Assume we have an offset $x' > x_{n-1}$. Consider the slope of segments between x_{n-1} and x'. It cannot be smaller than $\frac{c}{d}$ without violating concavity constraints, and it cannot be larger than $\frac{c}{d}$ without intercepting f at x_{n-1}. Therefore all segments between x_{n-1} and x' would need to have the same slope and we can combine them into one segment.

Now, assume we have an offset $x' < x_{n-1}$. By definition of x_{n-1}, the segment will not touch f at any point between x' and x_{n-1}. Therefore we can begin the segment at x_{n-1} and increase the slope of the segments between x' and x_{n-1}. $\square$

Lemma 4. *If $h_f = (h_i)_{i \leq n}$ is a tightest concave hull over f, then $h_i(x_i) = f(x_i)$.*

Proof. Assume that $\exists i \leq n : h_i(x_i) \neq f(x_i)$.

If $\text{slope}(h_i) = \text{slope}(h_{i-1})$, then we can combine h_{i-1} and h_i into one segment. This also reduces the computational effort of the DiscoDNC as it iterates over the linear segments of the hull.

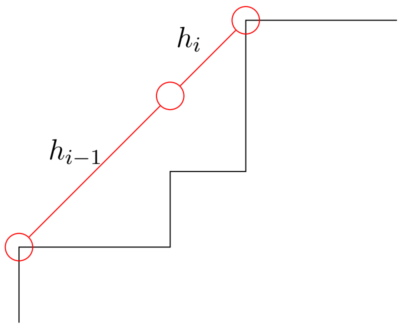


Figure 2.14.: Two segments with the same slope.

If there exists some $x : x_i \leq x < x_{i+1}$ with $h_i(x) = f(x)$, then we introduce a new segment h'_i with starting at x. We then look at h_i again and assume that one of the other conditions now applies.

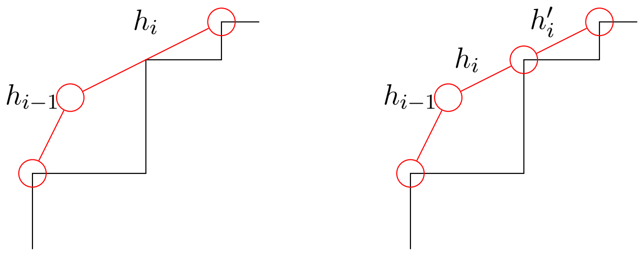


Figure 2.15.: Segment h_i does not intercept f at x_i, but at some point $x > x_i$.

Otherwise we can increase the slope of h_i and decrease the slope of h_{i-1} until either $h_i(x_i) = f(x_i)$, slope(h_i) = slope(h_{i-1}) or $\exists x : x_i \leq x \leq x_{i+1} : h_i(x) = f(x)$. This does not violate our concavity constraint: We cannot increase slope(h_i) above the original slope of h_{i-1} since in this case we would already have fulfilled our second condition and the same holds for decreasing the slope of h_{i-1} below the original slope of h_i.

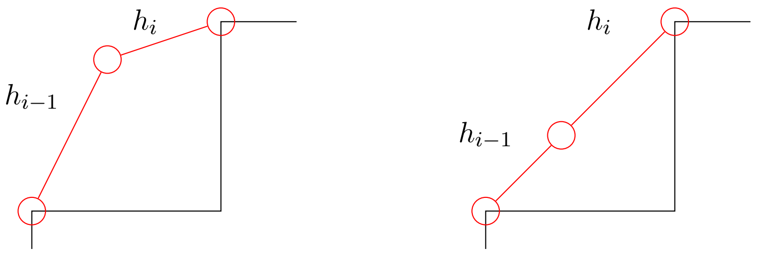


Figure 2.16.: Adjusting slopes of neighbouring segments.

□

Assume $\exists j : h_j, h_{j+1}, \ldots, h_n$ are fixed. Let x be a candidate offset for h_{j-1}. Any other candidate offset $x' < x$ for a segment h'_{j-1} with slope $h_{j-1} <$ slope h'_{j-1} will intercept f at x. Therefore for every segment h_i we only need to consider offsets with monotonically decreasing slopes.

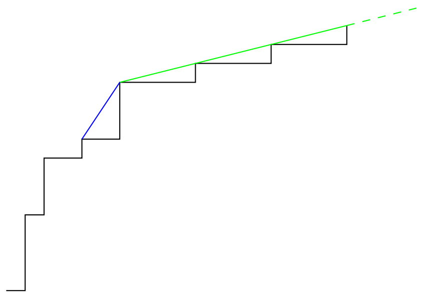
By concavity, the slope of h_{j-1} cannot decrease below the slope of h_j. If we sink below the slope of h_j, we can abort our search and conclude that no concave hull containing the segments $h_j, h_{j+1}, \ldots h_n$ exists.

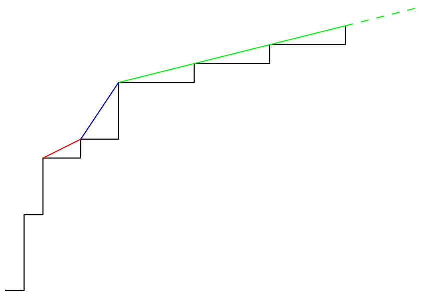
From this, we develop Algorithm 1:

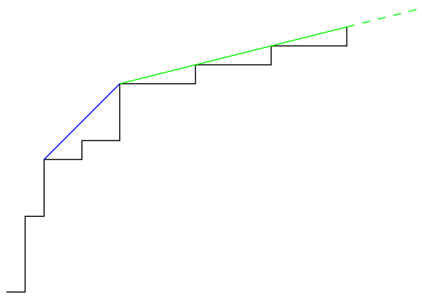
Algorithm 1 Concave Hull for Pseudo-Periodic Functions

```
function ConcaveHull(segment_end, min_slope) → segment offsets
# Compare Figure 2.17a
    segment_begin = segment_end − 1
    if segment_begin = 0 then
        return [0]
    end if
    max_slope = ∞
# Abort when no valid segments left
    while slope(segment_begin, segment_end) ≥ min_slope do
        current_slope = slope(segment_begin, segment_end)
        if current_slope ≤ max_slope then # Check for interception
            hull = ConcaveHull(segment_begin, current_slope)
            if hull ≠ "concave hull not found" then
# Found a valid hull, compare Figure 2.17d
                return hull + [segment_begin]
            end if
# Cannot have higher slope than this for the current segment without intercepting at segment_begin
            max_slope = current_slope
        end if
# Consider the next segment, compare Figure 2.17c
        segment_begin = segment_begin − 1
    end while
# Compare Figure 2.17b
    return "concave hull not found"
end function
```

For an original function f with n steps below $T + d$, in the worst case, this algorithm is equivalent to trying all possible compositions of 1 to n and therefore has a runtime of $\mathcal{O}(2^n)$.


(a) Initial step.


(b) Monotonicity of slope violated.


(c) Skipping a segment.

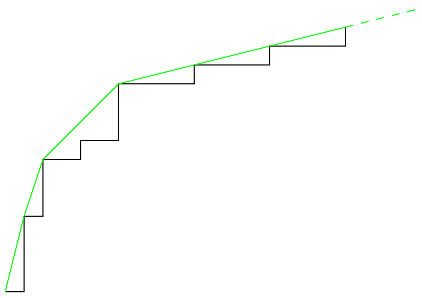

(d) Final hull.

Figure 2.17.: Concave hull algorithm.

3. Practical Considerations

3.1. Improving runtime for calculating single values of $\alpha(i)$

Looking at practical examples, it becomes obvious that brute-forcing the calculations is not feasible. Typical values for the length of blocks for the GSI control system are in the range of 10^9 nanoseconds. However there are usually only 10^2 to 10^3 messages in a block. We therefore store the point at which a message is sent in conjunction with the total traffic sent at the time, relative to the beginning of the block, in MESSAGES[1] and TRAFFIC_VALUES (this corresponds to tuples $\{ (\text{offset}(m), \tau_{<\text{offset}(m)}(b)) \mid m \in \mu(b) \}$). Additionally, we store the total traffic generated by a block $\tau(b)$ in TOTAL_TRAFFIC. Then by running a binary search over the messages sent in the blocks, values for MAXPREFIX (corresponding to $[\![\alpha\rangle_b$) can be efficiently calculated. Computation of MAXSUFFIX (corresponding to $\langle\alpha]\!]_b$) works analogously.

Algorithm 2 Maximum Prefix in a Single Block.

```
function Block::MAXPREFIX(time) → traffic
    if time = 0 then
        return 0 # Empty interval produces no traffic
    end if
    if time > duration then
# Query successor blocks
        return total_traffic + $\max_{b \in next}$ b.MAXPREFIX(time − duration)
    end if
# Find index of the step containing TIME
    traffic_increment_time = BINARY_SEARCH(messages, time)
# Return traffic at that step
    return traffic_values[traffic_increment_times]
end function
```

Additionally, by observing that for any maximal flow, there exists a flow with the same traffic that starts on a message being sent, we can efficiently calculate α by taking the maximum over the self-deconvolution of MAXPREFIX in each block, that is, $\alpha(i) = \max_{b \in B} \max_{k < \delta(b)} \{ [\![\alpha\rangle_{\{b\}}(k+i) - [\![\alpha\rangle_{\{b\}}(k) \}$.

[1] SMALL CAPS text is used to differentiate code from mathematical expressions

Algorithm 3 Maximum Traffic in a Single Block.

```
function BLOCK::MAXTRAFFIC(time) → traffic
    max = 0
    for $0 \leq$ i < traffic_values.LENGTH do
        offset = messages[i]
# Calculate $[\![\alpha\rangle_{\{b\}}(\text{OFFSET})$
        traffic_before_this_point = traffic_values[i-1]
# Calculate $[\![\alpha\rangle_{\{b\}}(\text{OFFSET} + \text{TRAFFIC})$
        long_traffic = this.MAXPREFIX(offset + time)
        traffic_in_interval = long_traffic − traffic_before_this_point
        max = MAX(max, traffic_in_interval)
    end for
    return max
end function
```

3.2. Step Functions

Looking at Algorithm 2, we see that when exceeding the block length, we need to consider every successor block. This quickly causes exponential blow-up in the running time of the algorithm. We can alleviate this problem by caching values for MAXPREFIX. Though caching the value for every single time slot is infeasible due to space requirements (four bytes per time slot means 7.5 GiB used per 10^9 time slots), we can use the fact that MAXPREFIX is a monotonic step-wise function with large gaps between steps to derive the data structure for step functions described in Listing 3.1, a logical extension of what we did in Algorithm 2 for messages in a single block:

Listing 3.1: Step Function

```
struct StepFunction {
    stepTimes: List[long]
    stepValues: List[double]
}
```

In this case, `stepTimes` are the times at which the function increases, sorted in ascending order, and `stepValues` the value of the function at that time. From the structure of the function, we also know that if `stepTimes`$[i] = t_i$ and `stepTimes`$[i+1] = t_{i+1}$, then $f(t) =$ `stepValues`$[i]$ for $t_i \leq t < t_{i+1}$.

This representation has some nice properties, namely for a function of n steps we can:

1. Find the value at time t in $\mathcal{O}(\log n)$ via binary search over `stepTimes`.
2. Find the first time the function exceeds some value in $\mathcal{O}(\log n)$ via binary search over `stepValues`.
3. Finding the maximum traffic over all intervals of some fixed length in $\mathcal{O}(n \log n)$ via self-deconvolution.

Of particular interest for optimization is Property 2: Assume we have some block b with a total traffic $\tau(b)$ and a maximum prefix function MAXPREFIXwith a maximum value of v. When calculating the time this function increases next, we just need to ask all successor blocks when their MAXPREFIX first exceeds $v - \tau(b)$.

Now, when we cache values of MAXPREFIX once calculated, we can effectively calculate $\alpha(i)$ by pre-calculating MAXPREFIX$(i+\delta(b))$ for each block and then querying each blocks MAXPREFIXfor the maximum interval of length i where the start of the interval does not exceed $\delta(b)$.

3.3. Runtime Analysis

We now extend Algorithm 2 with the previously mentioned optimizations.

Algorithm 4 Calculating MAXPREFIX.

```
function Block::firstTimeExceeding(traffic) → time
    while maxprefix.maxValue ≤ traffic do # Find the next step of the function
        trafficToReach = maxprefix.maxValue − totalBlockTraffic
        minDelay = min_{b∈next} b.firstTimeExceeding(trafficToReach)
        generatedTraffic = max_{b∈next} b.maxprefix(min)

        time = period + minDelay
        value = totalBlockTraffic + generatedTraffic
        add step to maxprefix at time time to value value
    end while
    return maxprefix.firstTimeExceeding(traffic)
end function

function Block::maxprefix(time) → traffic
    while maxprefix.validTo is not defined up to time do
        firstTimeExceeding(maxprefix.maxValue)
    end while
    return maxprefix.valueAt(time)
end function
```

Theorem 1. *Let $s(b,t)$ be the number of steps in the prefix function of block b up to and including time t. Let t be fixed and the maximum number of steps be $S = \max_{b\in B} \{ s(b,t) \}$. Calculating* MAXTRAFFIC *for all blocks up to time t has a runtime bound of $\mathcal{O}(|B|^2 \cdot S \cdot \log S + |B| \cdot S^2 \cdot \log S)$.*

Proof. Consider a single call of MAXPREFIX(t). This results in at most S calls of FIRST-TIMEEXCEEDING: each call of FIRSTTIMEEXCEEDING with its current maximum value as parameter provides exactly one step to the step function.

Each call of FIRSTTIMEEXCEEDING results in up to $|B|$ calls of FIRSTTIMEEXCEEDING on other blocks. Each of these may result in another $|B|$ calls of FIRSTTIMEEXCEEDING.

These calls however will return immediately in $\log S$, resulting in a runtime of $\mathcal{O}(|B|^2 \cdot \log S)$:

After a call of FIRSTTIMEEXCEEDING(u) for some block b and some traffic u, the step function of all other blocks is calculated at least up to FIRSTTIMEEXCEEDING$(u - \tau(b))$. The next call of FIRSTTIMEEXCEEDING$(u + u')$ will query the other blocks b' at $t + u' - \tau(b)$, which will, in turn, query other blocks at $u + u' - \tau(b) - \tau(b')$. Since the traffic generated in a single step can not exceed the traffic generated in any block, this means that $u' \leq \tau(b')$ and therefore $u + u' - \tau(b) - \tau(b') \leq u - \tau(b)$. In other words, it falls below the pre-calculation threshold and the call returns immediately. Total time to calculate MAXPREFIX(t) is therefore bounded by $\mathcal{O}(|B|^2 \cdot S \cdot \log S)$: S steps, each of which can be calculated in $\mathcal{O}(|B|^2 \log S)$).

Now consider the calls of MAXPREFIX(t) on the other blocks. The maxprefix step function has already been calculated up to FIRSTTIMEEXCEEDING$(u - \tau(b))$ where u is the maximum value of MAXPREFIX for block b. For all other blocks b', calls of FIRST-TIMEEXCEEDING$(u - \tau(b) + u')$ query the other blocks at $u - \tau(b) + u' - \tau(b')$, which falls below the pre-calculation threshold. This call therefore terminates in $\mathcal{O}(|B| \log S)$. Iterating over all blocks gives a runtime of $\mathcal{O}(|B|^2 \log S)$, which must, in turn, be repeated for all steps between FIRSTTIMEEXCEEDING$(u - \tau(b))$ and t, which cannot be more than S. Therefore, calculating MAXPREFIX(t) for all other blocks also takes $\mathcal{O}(|B|^2 \cdot S \cdot \log S)$.

Finally, calculating MAXTRAFFIC for one block takes $\mathcal{O}(S \cdot \log S)$, so for all block this is $\mathcal{O}(|B| \cdot S \cdot \log S)$. Doing this for all steps of α from 0 to t has therefore a runtime of $\mathcal{O}(|B| \cdot S^2 \cdot \log S)$. □

4. Analysis

We have developed three different approaches to estimate the arrival curve which we now want to compare in two different dimensions: How accurate each algorithm is, and how fast it terminates, in both cases with respect to their tuning parameters: The evaluation threshold for the sub-additive approximation, and the number of consecutive blocks for both fully-connected approximations. There are of course many more interesting dimensions, such as the density of messages, the number of blocks in the specification or the average branching factor of blocks, for which there was not enough time to test.

The following tests are all based on the *cryring_fictional.dot* specification (see Listing A.1 and Figure 4.1), where the sequence with the highest average traffic was `B_CRY_HALT` with 1 message per $5 \cdot 10^5$ ns, and the longest block had a length of $2.75 \cdot 10^9$ ns. The error is calculated based on the approximated slope of the final segment compared to the "true" slope of the final segment ($2 \cdot 10^{-6}$).

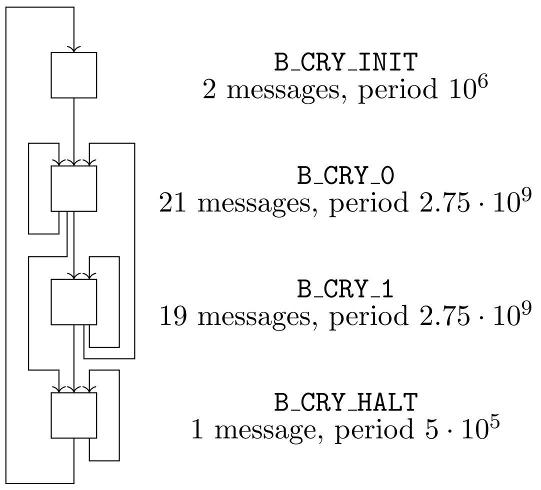

B_CRY_INIT
2 messages, period 10^6
B_CRY_0
21 messages, period $2.75 \cdot 10^9$
B_CRY_1
19 messages, period $2.75 \cdot 10^9$
B_CRY_HALT
1 message, period $5 \cdot 10^5$

Figure 4.1.: The *cryring_fictional* specification

4.1. Accuracy

Threshold	Slope of Final Segment	Error (abs.)	Error (rel.)
$2.75 \cdot 10^{2}$	$1.45 \cdot 10^{-2}$	$1.45 \cdot 10^{-2}$	7272.5
$2.75 \cdot 10^{4}$	$1.45 \cdot 10^{-4}$	$1.43 \cdot 10^{-4}$	72.7
$2.75 \cdot 10^{6}$	$4 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	2.0
$2.75 \cdot 10^{8}$	$2.01 \cdot 10^{-6}$	$1.09 \cdot 10^{-8}$	1.006
$2.75 \cdot 10^{10}$	$2.0001 \cdot 10^{-6}$	$1 \cdot 10^{-10}$	1.0001

Table 4.1.: Accuracy of sub-additive approximation for different thresholds (Ground truth slope is $2 \cdot 10^{-6}$). Absolute error: Calculated slope minus ground truth. Relative error: Calculated slope divided by ground truth.

As Table 4.1 shows, with the sub-additive approximation the error quickly approaches 0. However, it will never reach zero as the sub-additive approximation pays bursts for each multiple of the threshold instead of only once.

Num. Blocks	Slope of Final Segment	Error (abs.)	Error (rel.)
1	$4.2 \cdot 10^{-5}$	$4 \cdot 10^{-5}$	21.0
2	$4.2 \cdot 10^{-5}$	$4 \cdot 10^{-5}$	21.0
3	$4.2 \cdot 10^{-5}$	$4 \cdot 10^{-5}$	21.0
4	$4.2 \cdot 10^{-5}$	$4 \cdot 10^{-5}$	21.0

Table 4.2.: Accuracy of rescaled approximation for different numbers of consecutive blocks.

As expected, the rescaling approximation method is fairly inaccurate. In particular, the relative error will not fall below the maximum number of messages in any block (21) to the number of messages in the most effective block (1). Increasing the number of consecutive blocks does not improve the results, because no inaccuracy is introduced by the fully-connected model: Since in the original model the block `B_CRY_HALT` is connected to itself, the optimal sequence of blocks is (`B_CRY_HALT`)*. See Figure 4.2 for a specification where increasing the number of consecutive block *does* affect accuracy.

Num. Blocks	Slope of Final Segment	Error (abs.)	Error (rel.)
1	$2 \cdot 10^{-6}$	0	0
2	$2 \cdot 10^{-6}$	0	0

Table 4.3.: Accuracy of unscaled fully-connected approximation for different numbers of consecutive blocks.

Table 4.3 demonstrates the advantage of the unscaled fully-connected approximation when the number of consecutive blocks is a multiple of the number of blocks in the most effective sequence. If this holds and the sequence contains no alternate paths, the approximation is perfect. To clarify the case where this does not hold, consider the specification in Figure 4.2:

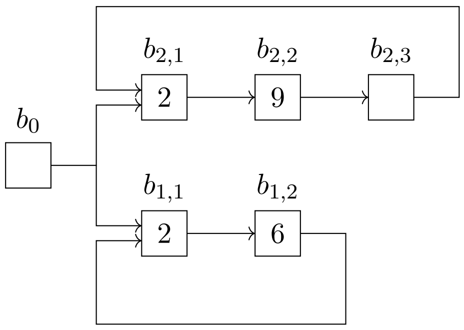


Figure 4.2.: 2-block inaccuracy.

In Figure 4.2, we can see that the most effective cycle has a duration of 2, consisting of $b_{1,1}$ and $b_{1,2}$ with an average traffic per timeslot of 4. However in the fully-connected 2-block model we consider $b_{2,1}b_{2,2}$ to be the most effective cycle. To get an accurate result, we would need to pick a cycle length of 6 (the least common multiple of the two cycles).

Num. Blocks	Slope of Final Segment	Error (abs.)	Error (rel.)
1	9	5	2.25
2	5.5	1.5	1.375
3	4.7	0.7	1.175
4	5	1	1.25
6	4	0	1

Table 4.4.: (In-)Accuracy of unscaled fully-connected approximation.

Let us look at some of the entries in Table 4.4 more closely: For three consecutive blocks, the sequence $b_{1,2}b_{1,1}b_{1,2}$ produces the most traffic (14). For four consecutive blocks, the tightest loop is $b_{2,2}b_{2,3}b_{2,1}b_{2,2}$ with a total traffic of 20. Finally, for 6 consecutive blocks, the most effective sequence is $(b_{1,1}b_{1,2})^3$ with a total traffic of 24.

This demonstrates that a fairly large number of consecutive block may be required to get an accurate result. However, it also shows that an increase in the number of blocks does not necessarily correspond to an increase in accuracy. Rather, increasing the block count can lead to sequences with a higher relative traffic – in Table 4.4, demonstrated by the 4-block model.

It should be noted that there are specifications for which the unscaled fully-connected model is unable to find a perfect approximation at all. In Figure 4.2, this could be achieved by adding a connection from $b_{1,2}$ to $b_{2,2}$.

4.1.1. Summary of accuracy measurements

Looking back at the results of the accuracy tests, we clearly see the inaccuracy of the rescaled approximation. This will likely make it too inaccurate for a lot of applications. On the other hand, both the sub-additive and the fully-connected approximation produce fairly accurate results and seem feasible in practice. Sadly calculating the error is not possible in the general case, so estimating the error is not feasible, but since every result is an over-approximation one can simply sample several results and pick the one with the lowest slope.

4.2. Running Time

The following tests were run on a Windows 10 computer with an quad-core Intel i7-4710HQ with a clock rate of 2.5GHz and 16GB RAM with a speed of 1600MHz. However, they are not meant to be authoritative, but rather give a rough estimate of the speed of each approximation.

Threshold	**Mean time**	**Standard deviation (SD)**
$2.75 \cdot 10^2$	6ms	19ms
$2.75 \cdot 10^4$	6ms	17ms
$2.75 \cdot 10^6$	8ms	22ms
$2.75 \cdot 10^8$	36ms	38ms
$2.75 \cdot 10^{10}$	1.154s	537ms

Table 4.5.: Runtime of sub-additive approximation for different thresholds over 10 repetitions.

As one can see in Table 4.5, the sub-additive approximation is very fast, returning very accurate results within a couple of seconds. Runtime increases roughly linear with the threshold, which is exactly what one would expect from the current implementation of the algorithm.

Cons. Blocks	**Blocks in Model**	**Mean time**	**SD**
1	4	21ms	39ms
2	9	84ms	63ms
3	22	228ms	159ms
4	52	604ms	303ms

Table 4.6.: Runtime of rescaling approximation for different numbers of consecutive blocks over 10 repetitions.

The runtime of the rescaling approximation appears to be roughly linear in the number of blocks in the model. However, this can be exponential in the number of consecutive blocks in the worst case – given an average connectedness of blocks c and k consecutive blocks, the number of blocks in the final model is in $\mathcal{O}(nc^k)$.

Cons. Blocks	Blocks in Model	Mean time	SD
1	4	57.727s	3.995s
2	9	9m 44.571s	1m 31.025s
3	22	50m 10.819s	1m 38.401s

Table 4.7.: Runtime of fully-connected approximation for different numbers of consecutive blocks over 10 repetitions.

As we can see, the runtime of the fully-connected approximation is much larger than those of the other two approximation. On top of the high connectedness and high number of blocks caused by the fully-connected model, the algorithm is not only exponential in the number of consecutive blocks, but also suffers when the lengths of the most effective block and the longest block differ.

4.2.1. Summary of time measurements

Looking back at the results in this chapter, we see that both the sub-additive and the rescaling approximation run in an acceptable time. However, the runtime of the fully-connected model quickly becomes prohibitively expensive (for four blocks, estimated to be around 10 hours). It's therefore recommended to not use the fully-connected model unless a few blocks are sufficient for accuracy purposes.

4.3. Summary

We now have seen two different dimensions that should influence our choice in algorithms.

It appears that the rescaled approximation is fast, but not accurate enough for the general case. However, it should be noted that the *cryring_fictional* specification had a large discrepancy between the number of messages in the most efficient cycle (the block `B_CRY_HALT` with one message and a duration of $5 \cdot 10^5$ ns) and the number of messages in the block with the most messages (21). In specifications where blocks are generally more similar in length and/or message count, this inaccuracy can be less extreme.

The fully-connected approximation is the most accurate approximation, but this accuracy is paid for with a very high running time. However, if the necessary number of sequential blocks is known in advance and the approximation is calculated as part of a longer verification step, the time cost may be less important.

Finally, the sub-additive approximation has a good ratio of accuracy to run time, converging quickly to a very small error. In most cases, unless the specification is utilizing the system to capacity, this will be enough.

5. Summary

In this thesis, I motivated why there is a need to determine delay bounds for high-risk systems. I then introduced a formal way to model traffic specifications.

I developed and discussed several approaches to extract ultimately pseudo-periodic functions from specifications as used by the GSI, three of which proved feasible in theory: the sub-additive approximation, the rescaling approximation and the fully-connected approximation.

I then focused on implementation details which make sparse specifications (those where the number of messages per block is several orders of magnitude below the length of the blocks) at all possible.

Finally, I compared the three algorithms with respect to their accuracy and running time. What remains is the question which approximation is best for the use-case of the FAIR. In general, the sub-additive approximation has the best balance of accuracy and speed, and I suspect that it's accurate enough at high thresholds for most applications. On the other hand, if there are specifications where the ideal sequential block count is immediately obvious (such as in *cryring_fictional*), the fully-connected approximation has the potential to be more accurate. However, as Figure 4.2 has shown, the required number of sequential blocks can grow very quickly and therefore make this approach infeasible.

Lastly, it is very likely that every arrival curve is ultimately pseudo-periodic and it should be possible to develop an approach that finds it in the general case. However, it was not possible to do so in the scope of this thesis. It therefore remains open to future research.

Bibliography

[1] J. Yoo, S. Cha, and E. Jee, “A verification framework for fbd based software in nuclear power plants,” in *2008 15th Asia-Pacific Software Engineering Conference*, Dec 2008, pp. 385–392.

[2] C. J. Tomlin, I. Mitchell, A. M. Bayen, and M. Oishi, “Computational techniques for the verification of hybrid systems,” *Proceedings of the IEEE*, vol. 91, no. 7, pp. 986–1001, July 2003.

[3] A. Singh, “RTCA DO-178B (EUROCAE ED-12B),” 04 2011.

[4] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet.* Springer, 2001.

[5] S. Bondorf and J. Schmitt, “The DiscoDNC v2 - A Comprehensive Tool for Deterministic Network Calculus,” *EAI Endorsed Transactions on Internet of Things*, vol. 15, no. 4, 2 2015.

[6] J. Schmitt and F. A. Zdarsky, “The DISCO Network Calculator - A Toolbox for Worst Case Analysis,” in *Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS’06), Pisa, Italy.* ACM, Nov. 2006. [Online]. Available: /discofiles/publicationsfiles/SZ06-1.pdf

[7] How FAIR is being built. [Online]. Available: http://www.fair-center.eu/construction/how-fair-is-being-built.html

[8] R. Huhmann, R. Bär, D. Beck, J. Fitzek, G. Fröhlich, L. Hechler, U. Krause, and M. Thieme, “The Fair Control System - System Architecture and First Implementations,” in *Proceedings of ICALEPCS2013*, 2013, pp. 328–331.

[9] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, and G. Gaderer, “White rabbit: Sub-nanosecond timing distribution over ethernet,” in *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, Oct 2009, pp. 1–5.

[10] A. Bouillard and É. Thierry, “An algorithmic toolbox for network calculus,” *Discrete Event Dynamic Systems*, vol. 18, no. 1, pp. 3–49, Mar 2008. [Online]. Available: https://doi.org/10.1007/s10626-007-0028-x

Appendices

A. Sample traffic specifications

Listing A.1: cryring_fictional

```
digraph G {
graph [root="CRY_INIT_CAL_0", nodesep=0.6, mindist=1.0, ranksep=1.0, overlap=false]
node [shape="oval", color="black"]
edge [type="defDst", color="red"]

//Cryring Init (fictional)
CRY_INIT_CAL_0         [type="TMsg", fid=0, gid=201, evtno=257,  sid=1, bpid=2, tOffs=0,
par=5214769];
CRY_INIT_CAL_1         [type="TMsg", fid=0, gid=201, evtno=257,  sid=1, bpid=2, tOffs=500,
par=80471];
B_CRY_INIT             [type="Block", shape="rectangle", color="red", tPeriod=1000000];

//Cryring Standard Operation (real)
CRY_0_SEQ_START_0    [type="TMsg", fid=0, gid=201, evtno=257,  sid=1, bpid=2, tOffs=0,          par=0];
CRY_0_SEQ_START_1    [type="TMsg", fid=0, gid=202, evtno=257,  sid=1, bpid=3, tOffs=0,          par=0];
CRY_0_SEQ_START_2    [type="TMsg", fid=0, gid=203, evtno=257,  sid=1, bpid=1, tOffs=0,          par=0];
CRY_0_SEQ_START_3    [type="TMsg", fid=0, gid=210, evtno=257,  sid=1, bpid=5, tOffs=0,          par=0];
CRY_0_RF_PREP_PAUSE  [type="TMsg", fid=0, gid=202, evtno=291,  sid=1, bpid=3, tOffs=488100000,  par=0];
CRY_0_RF_STOP_PAUSE  [type="TMsg", fid=0, gid=202, evtno=292,  sid=1, bpid=3, tOffs=490700000,  par=0];
CRY_0_RF_PREP        [type="TMsg", fid=0, gid=202, evtno=290,  sid=1, bpid=3, tOffs=491700000,  par=0];
CRY_0_BUMPER_CHARGE  [type="TMsg", fid=0, gid=210, evtno=1024, sid=1, bpid=5, tOffs=491885893,  par=0];
CRY_0_BI_TRIGGER     [type="TMsg", fid=0, gid=202, evtno=280,  sid=1, bpid=2, tOffs=491950000,  par=0];
CRY_0_BEAM_ON_0      [type="TMsg", fid=0, gid=201, evtno=518,  sid=1, bpid=2, tOffs=492200000,  par=0];
CRY_0_BEAM_ON_1      [type="TMsg", fid=0, gid=202, evtno=518,  sid=1, bpid=3, tOffs=492200000,  par=0];
CRY_0_BI_MEAS_1      [type="TMsg", fid=0, gid=201, evtno=281,  sid=1, bpid=2, tOffs=492201000,  par=0];
CRY_0_BEAM_ON_2      [type="TMsg", fid=0, gid=210, evtno=518,  sid=1, bpid=5, tOffs=492203393,  par=0];
CRY_0_BI_MEAS_2      [type="TMsg", fid=0, gid=201, evtno=282,  sid=1, bpid=2, tOffs=492951000,  par=0];
CRY_0_BEAM_OFF_0     [type="TMsg", fid=0, gid=201, evtno=520,  sid=1, bpid=2, tOffs=493000000,  par=0];
CRY_0_BEAM_OFF_1     [type="TMsg", fid=0, gid=202, evtno=520,  sid=1, bpid=3, tOffs=493000000,  par=0];
CRY_0_SEQ_START_4    [type="TMsg", fid=0, gid=210, evtno=257,  sid=1, bpid=4, tOffs=500000000,  par=0];
CRY_0_SYNCH_0        [type="TMsg", fid=0, gid=210, evtno=312,  sid=1, bpid=4, tOffs=620000000,  par=0];
CRY_0_BP_START       [type="TMsg", fid=0, gid=210, evtno=256,  sid=1, bpid=5, tOffs=1620000000, par=0];
CRY_0_SYNCH_1        [type="TMsg", fid=0, gid=210, evtno=312,  sid=1, bpid=5, tOffs=1740000000, par=0];
CRY_0_BEAM_OFF_2     [type="TMsg", fid=0, gid=210, evtno=520,  sid=1, bpid=5, tOffs=2740000000  par=0];
B_CRY_0              [type="Block", shape="rectangle", color="red", tPeriod=2750000000];

//Cryring Dry-Run Operation (fictional)

CRY_1_SEQ_START_0    [type="TMsg", fid=0, gid=201, evtno=257,  sid=1, bpid=2, tOffs=0,          par=0];
CRY_1_SEQ_START_1    [type="TMsg", fid=0, gid=202, evtno=257,  sid=1, bpid=3, tOffs=0,          par=0];
CRY_1_RF_PREP_PAUSE  [type="TMsg", fid=0, gid=202, evtno=291,  sid=1, bpid=3, tOffs=408100000,  par=0];
CRY_1_RF_STOP_PAUSE  [type="TMsg", fid=0, gid=202, evtno=292,  sid=1, bpid=3, tOffs=490000000,  par=0];
CRY_1_RF_PREP        [type="TMsg", fid=0, gid=202, evtno=290,  sid=1, bpid=3, tOffs=490000000,  par=0];
CRY_1_BUMPER_CHARGE  [type="TMsg", fid=0, gid=210, evtno=1024, sid=1, bpid=5, tOffs=491885893,  par=0];
CRY_1_BI_TRIGGER     [type="TMsg", fid=0, gid=202, evtno=280,  sid=1, bpid=2, tOffs=491900000,  par=0];
CRY_1_BEAM_ON_0      [type="TMsg", fid=0, gid=201, evtno=518,  sid=1, bpid=2, tOffs=492200000,  par=0];
CRY_1_BEAM_ON_1      [type="TMsg", fid=0, gid=202, evtno=518,  sid=1, bpid=3, tOffs=492200000,  par=0];
CRY_1_BI_MEAS_1      [type="TMsg", fid=0, gid=201, evtno=281,  sid=1, bpid=2, tOffs=492201000,  par=0];
CRY_1_BEAM_ON_2      [type="TMsg", fid=0, gid=210, evtno=518,  sid=1, bpid=5, tOffs=492203393,  par=0];
CRY_1_BI_MEAS_2      [type="TMsg", fid=0, gid=201, evtno=282,  sid=1, bpid=2, tOffs=492951000,  par=0];
CRY_1_BEAM_OFF_0     [type="TMsg", fid=0, gid=201, evtno=520,  sid=1, bpid=2, tOffs=493000000,  par=0];
CRY_1_BEAM_OFF_1     [type="TMsg", fid=0, gid=202, evtno=520,  sid=1, bpid=3, tOffs=493000000,  par=0];
CRY_1_SEQ_START_4    [type="TMsg", fid=0, gid=210, evtno=257,  sid=1, bpid=4, tOffs=500000000,  par=0];
CRY_1_SYNCH_0        [type="TMsg", fid=0, gid=210, evtno=312,  sid=1, bpid=4, tOffs=620000000,  par=0];
CRY_1_BP_START       [type="TMsg", fid=0, gid=210, evtno=256,  sid=1, bpid=5, tOffs=1000000000, par=0];
CRY_1_SYNCH_1        [type="TMsg", fid=0, gid=210, evtno=312,  sid=1, bpid=5, tOffs=1500000000, par=0];
CRY_1_BEAM_OFF_2     [type="TMsg", fid=0, gid=210, evtno=520,  sid=1, bpid=5, tOffs=2740000000  par=0];
B_CRY_1              [type="Block", shape="rectangle", color="red", tPeriod=2750000000];

//Cryring Halt (fictional)
CRY_HALT             [type="TMsg", fid=0, gid=201, evtno=257,  sid=1, bpid=2, tOffs=0,          par=80471];
B_CRY_HALT           [type="Block", shape="rectangle", color="red", tPeriod=500000];


CRY_INIT_CAL_0       -> CRY_INIT_CAL_1
CRY_INIT_CAL_1       -> B_CRY_INIT
B_CRY_INIT           -> CRY_0_SEQ_START_0
CRY_0_SEQ_START_0     -> CRY_0_SEQ_START_1
CRY_0_SEQ_START_1     -> CRY_0_SEQ_START_2
CRY_0_SEQ_START_2     -> CRY_0_SEQ_START_3
CRY_0_SEQ_START_3     -> CRY_0_RF_PREP_PAUSE
CRY_0_RF_PREP_PAUSE   -> CRY_0_RF_STOP_PAUSE
CRY_0_RF_STOP_PAUSE   -> CRY_0_RF_PREP
CRY_0_RF_PREP         -> CRY_0_BUMPER_CHARGE
CRY_0_BUMPER_CHARGE   -> CRY_0_BI_TRIGGER
CRY_0_BI_TRIGGER      -> CRY_0_BEAM_ON_0
CRY_0_BEAM_ON_0       -> CRY_0_BEAM_ON_1
CRY_0_BEAM_ON_1       -> CRY_0_BI_MEAS_1
CRY_0_BI_MEAS_1       -> CRY_0_BEAM_ON_2
CRY_0_BEAM_ON_2       -> CRY_0_BI_MEAS_2
CRY_0_BI_MEAS_2       -> CRY_0_BEAM_OFF_0
CRY_0_BEAM_OFF_0      -> CRY_0_BEAM_OFF_1
CRY_0_BEAM_OFF_1      -> CRY_0_SEQ_START_4
CRY_0_SEQ_START_4     -> CRY_0_SYNCH_0
CRY_0_SYNCH_0         -> CRY_0_BP_START
CRY_0_BP_START        -> CRY_0_SYNCH_1
CRY_0_SYNCH_1         -> CRY_0_BEAM_OFF_2
CRY_0_BEAM_OFF_2      -> B_CRY_0
B_CRY_0               -> CRY_0_SEQ_START_0
B_CRY_0               -> CRY_1_SEQ_START_0 [type="altDst", color="black"];

CRY_1_SEQ_START_0     -> CRY_1_SEQ_START_1
CRY_1_SEQ_START_1     -> CRY_1_RF_PREP_PAUSE
CRY_1_RF_PREP_PAUSE   -> CRY_1_RF_STOP_PAUSE
CRY_1_RF_STOP_PAUSE   -> CRY_1_RF_PREP
CRY_1_RF_PREP         -> CRY_1_BUMPER_CHARGE
CRY_1_BUMPER_CHARGE   -> CRY_1_BI_TRIGGER
CRY_1_BI_TRIGGER      -> CRY_1_BEAM_ON_0
CRY_1_BEAM_ON_0       -> CRY_1_BEAM_ON_1
CRY_1_BEAM_ON_1       -> CRY_1_BI_MEAS_1
CRY_1_BI_MEAS_1       -> CRY_1_BEAM_ON_2
CRY_1_BEAM_ON_2       -> CRY_1_BI_MEAS_2
CRY_1_BI_MEAS_2       -> CRY_1_BEAM_OFF_0
CRY_1_BEAM_OFF_0      -> CRY_1_BEAM_OFF_1
CRY_1_BEAM_OFF_1      -> CRY_1_SEQ_START_4
CRY_1_SEQ_START_4     -> CRY_1_SYNCH_0
CRY_1_SYNCH_0         -> CRY_1_BP_START
CRY_1_BP_START        -> CRY_1_SYNCH_1
CRY_1_SYNCH_1         -> CRY_1_BEAM_OFF_2
CRY_1_BEAM_OFF_2      -> B_CRY_1
B_CRY_1               -> CRY_1_SEQ_START_0
B_CRY_1               -> CRY_0_SEQ_START_0 [type="altDst", color="black"];

B_CRY_0               -> CRY_HALT [type="altDst", color="black"];
B_CRY_1               -> CRY_HALT [type="altDst", color="black"];
CRY_HALT              -> B_CRY_HALT
B_CRY_HALT            -> B_CRY_HALT
B_CRY_HALT            -> CRY_INIT_CAL_0 [type="altDst", color="black"];


}
```