Finite Abstractions of Network Calculus Elements for Formal Verification of Network Bounds

Bassilio Dahlan

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Finite Abstractions of Network Calculus Elements for Formal Verification of Network Bounds

Master of Science Thesis

For the degree of Master of Science in Embedded Systems at Delft University of Technology

Bassilio Dahlan

September 9, 2013

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) for acceptance a thesis entitled

Finite Abstractions of Network Calculus Elements for Formal Verification of Network Bounds

by

Bassilio Dahlan

in partial fulfillment of the requirements for the degree of

Master of Science Embedded Systems

Dated: September 9, 2013

Supervisor(s):

Prof. dr. ir. B. De Schutter

Dr. ir. M. Mazo Espinosa

Reader(s):

Dr. ir. F. Kuipers

Dr. ir. A. Abate

D. Adzkiya, MSc

Abstract

This thesis presents a method to semi-automate the reasoning about network bounds, namely the backlog and delay. Network Calculus offers a rich theory for modeling network elements. Using the modeling techniques of network calculus, network elements are represented by discrete-time systems whose dynamics are linear in min-plus algebra. From the state-space representation of a min-plus linear (MiPL) system, a finite-state symbolic model or quotient system can be constructed. The construction of such quotient systems is done by first partitioning the state-space according to the MiPL dynamics and certain atomic propositions defined over the states. The transitions between the quotient states are then computed based on the dynamical system. The resulting finite abstraction simulates the infinite-state discrete-time model, which permits applying formal verification to reason about the properties of the discrete-time system. The work is finalized by transforming the backlog and virtual delay properties of the original model to equivalent specifications, given by Linear Temporal Logic (LTL) formulas, on the finite abstraction.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Prof. Bart De Schutter and Dr. Fernando Kuipers for accepting to be in my thesis committee to assess my work. I am very thankful to my supervisor Dr. Manuel Mazo Espinosa for his supervision, guidance, and regular feedback during all the stages of writing this thesis. Special thanks goes to Dieky Adzkiya for his constant support and patience especially during our very long discussions. Thanks also to Dr. Alessandro Abate for his assistance and helpful comments.

Delft, University of Technology — Bassilio Dahlan
September 9, 2013

This work is dedicated to my parents and brother whose love, support, and encouragement sustained me throughout.

Chapter 1

Introduction

In the recent years, a lot of work has been done in the emerging field of cyber-physical systems (CPS) to develop methods and techniques to bridge the gap between the cyber and the physical sides of control systems' implementations. Much of the effort was invested in finding suitable approaches to automate the process of verification and/or synthesis of correct-by-design control systems [1], [2], [3], [4]. Symbolic modeling is a central technique in CPS that is based on extracting a finite-state model of any infinite-state system in order for the model to be used for further automated processing. The symbolic model, also called the finite abstraction, can be used to verify certain properties about the physical system and, if possible, automatically synthesize a symbolic controller for it. Various techniques have been developed to obtain symbolic models from the mathematical description of physical systems given in the form of differential equations. These techniques have shown great potentials for solving many control problems very effectively.

Many challenges arise when analyzing controllers whose control loops are closed via shared communication channels as in Networked Control Systems (NCS) [5]. NCS rely on networks for information exchange between their elements. In NCS, sampled data from sensors must travel through a network to reach a controller. After performing the necessary computations, the actuation data is then sent back also through the network to the actuators. Since the last decade, a lot of research was devoted to study and understand the effects of the shared communication infrastructures on control systems [6], [7], [8]. With sensing and actuation paths going through a network, time predictability is greatly compromised. Timing properties are very crucial to ensure the stability and performance guarantees of the control systems. Controllers in networked systems must deal with new problems that arise from the fact that networks are shared resources. As the number of control loops implemented over the shared communication channel increases, bandwidth scarcity becomes a significant issue requiring a more conservative usage by each controller. These new challenges rendered the conventional periodic execution paradigms inefficient for managing network resources. For this reason, researchers have proposed alternative approaches to the control problem that depend on aperiodic updates such as the event-based schemes [9], [10], [11], [12]. Despite partially solving the problem of bandwidth, aperiodic control schemes require guaranteeing

certain delay bounds for traffic. For aperiodic traffic with known deadlines, there are well-studied scheduling techniques that offer guaranteed delay bounds for the traffic [13], [14], [15]. However, this problem remains less studied for cases involving sporadic traffic with unknown deadlines generated in event-based schemes.

Reasoning about network parameters is in general not a simple task. The challenge is even magnified if the reasoning is to be done automatically during the controller synthesis to make the design more optimal for the network. Inspired by the above problems, the work in this thesis will investigate an approach to study networks based on Network Calculus [16]. Network calculus theory offers a very rich set of mathematical tools to model network elements and quantitatively assess the quality of service offered. What makes network calculus different from and more attractive than the traditional queuing theory is that it is concerned with the worst-case scenario rather than the average case or steady-state behavior. From the perspective of NCS, the worst-case analysis is the most important and relevant type of analysis to guarantee the correct operation of controllers. Interestingly, network calculus allows modeling traffic sources in a way that can readily incorporate aperiodic traffic flows. Moreover, physically different network elements can be modeled with very similar abstract elements simplifying the analysis of compositions in the actual networks. Strongly motivated by these characteristics, this work's main objective is to create a suitable modeling framework based on network calculus concepts to allow the application of verification techniques to network problems using finite abstractions. Being highly relevant to control applications, the work focuses particularly on the virtual delay and backlog bounds of a network. In the context of control systems, studying the virtual delay is useful for quantifying the worst-case delay in the feedback measurements. At the same time, the backlog bound can be studied to prevent network congestion and avoid packet drop-outs. Although these parameters can already be analyzed using network calculus, the goal of this work is to automate the process using symbolic abstraction approaches similar to those proposed for the automated synthesis of controllers. In future, this could open the door for simultaneous verification of controller and communication software. Furthermore, it could provide a suitable means for optimizing correct-by-design controllers that are targeted to operate in networked systems.

This thesis is divided into six chapters. In Chapter 2, the reader is provided with the theoretical background that is required for understanding the modeling and abstraction techniques presented later. The network problem is formulated in Chapter 3, where network calculus is used to create state-space representations of conventional network elements. The abstraction technique is presented in Chapter 4 together with the algorithms that can be used to generate the symbolic models automatically. The problem of automated reasoning about network parameters is also approached in Chapter 4 by representing network properties in temporal logic for formal verification. In Chapter 5, the presented techniques are applied to an example where the symbolic abstraction is created automatically via MATLAB. The concluding remarks and the discussion of future work are finally given in Chapter 6.

Chapter 2

Background

2-1 Notation

Let $\mathbb{R} = (-\infty, +\infty)$ be the set of all real numbers, $\mathbb{R}_\varepsilon = \mathbb{R} \cup \{+\infty\}$ be the extended set of real numbers, $\mathbb{Z} = \{\cdots -2, -1, 0, 1, \cdots\}$ be the set of all integers, and $\mathbb{N} = \{0, 1, 2, \cdots\}$ be the set of natural numbers. The empty set $\{\}$ is denoted by $\varnothing$. Given a set of numbers W, W^+ denotes the set of all non-negative numbers in W, that is $W^+ = \{w \in W \mid w \geq 0\}$. Given a set Z, $|Z|$ denotes the cardinality of the set and Z is said to be countable if and only if $|Z| \leq |\mathbb{N}|$. The super set of Z, denoted by 2^Z, is the set of all subsets of Z. For a set Z, Z^ω denotes the set of all infinite strings obtained from concatenating elements in Z. An element $\mathsf{z} \in Z^\omega$ is a map $\mathsf{z} : \mathbb{N} \mapsto Z$ represented by $\mathsf{z} = z_0 z_1 z_2 \cdots$ with $\mathsf{z}(i)$ being the ith element in z and $\mathsf{z}[k]$ being an infinite string obtained from removing the first k elements from z.

The set $\mathcal{F}$ is the set of all non-negative wide-sense increasing functions and sequences formally defined as $\mathcal{F} = \{f : \mathbb{R} \mapsto \mathbb{R}_\varepsilon^+ \mid f(t) \geq f(s) \text{ for all } t \geq s\}$. Also, $\mathcal{F}_0$ is the set of all functions and sequences in $\mathcal{F}$ satisfying $f(0) = 0$. Given a function $f : X \mapsto Y$, the image of a set $K \subseteq X$ under f is $f(K) = \{y \in Y \mid y = f(k) \text{ for some } k \in K\}$ and $f|_K : K \mapsto Y$ describes a restriction of f to K defined by $f|_K(k) = f(k)$ for every $k \in K$.

Let T be a partially ordered set. For a set $S \subseteq T$, $\sup[S]$ is the least upper bound of S in T, and $\inf[S]$ is the greatest lower bound of S in T. If $\inf[S] \in S$ (resp. $\sup[S] \in S$), then it is called the minimum (resp. maximum) element denoted by $\min[S]$ (resp. $\max[S]$). Given a function $f : X \mapsto Y$ and a set $K \subseteq X$, the greatest lower limit of f over K is $\inf_{k \in K}[f(k)] = \inf[f(K)]$ and the least upper limit of f over K is $\sup_{k \in K}[f(k)] = \sup[f(K)]$. In min-plus algebra, the minimum operation is denoted by $\oplus$ and the addition operation is denoted by $\otimes$. The min-plus operations are defined over the set $\mathbb{R}_\varepsilon$ with the null element $\varepsilon = \infty$ and the unity (also identity) element $e = 0$. The clipping operator $[\cdot]^+$ is defined as $[x]^+ = \max[x, 0]$. Given an n-dimensional vector $\begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}$, $\begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}^T$ denotes the transpose of the vector.

An equivalence relation Q on a set Z induces a partition denoted by Z/Q. For any $z \in Z$, $[z] = \{z' \in Z \mid (z, z') \in Q\}$ is the equivalence class to which the element z belongs and

$\pi_Q : z \mapsto Z/Q$ is the canonical projection map that maps each element of the set to its equivalence class such that $\pi_Q(z) = [z] \in Z/Q$. If $|Z/Q|$ is finite, i.e. the partition is composed of a finite number equivalence classes, then Q is said to be a finite equivalence relation.

The operators $\wedge$, $\vee$, and $\neg$ of propositional logic are used to denote the conjunction, disjunction, and negation operations, respectively. Logical implication and equivalence are denoted by $\Longrightarrow$ and $\Longleftrightarrow$, respectively. The existential and universal quantifiers of predicate logic are denoted by $\exists$ and $\forall$, respectively. The temporal logic operators are: $\Diamond$ for "eventually," $\Box$ for "always," $\bigcirc$ for "next," and $\mathcal{U}$ for "until."

2-2 Network calculus

Since the early development of telephony, queuing theory has served as a very handy modeling tool for the analysis of communication systems. The development of new technologies for computer communications and the emergence of the Internet have rapidly increased the complexity of the networks to be analyzed. It was soon observed that packet data traffic of the Internet do not possess the memoryless property of Poisson processes. This observation rendered many useful results from the classical queuing theory inapplicable, and increased the need for a new modeling paradigm, especially with the advent of architectures for quality of service. The emergence of new challenges in the modeling of modern communication networks soon paved the way for the development of network calculus, which was introduced by Cruz in his seminal paper [17] and subsequently in [18]. With the collaboration of more researchers, the theory was expanded to target more complex quality of service architectures. Most of the results of the theory are compiled in a useful reference book [16] by Le Boudec and Thiran. A more elaborate approach based on min-plus system theory is given in the book by Chang [19]. The concepts of network calculus were later expanded to include stochastic analysis, creating a branch called *stochastic network calculus* (SNC). Jiang gives a good introduction to stochastic network calculus in [20]. Since this thesis is based exclusively on the *deterministic* branch of network calculus (DNC), the theory related to SNC will be excluded from the following discussion and the modifier "deterministic" will be dropped for being redundant.

Network calculus provides a rich set of results that allows a precise analysis of performance parameters of complex communication networks. It is mathematically founded on the *min-plus* system theory, which is, unlike the conventional system theory, based on the *min-plus algebra*. The most important result in network calculus is that the performance analysis of a complex network can be done with a limited knowledge of the traffic flow through the network given that certain requirements are met. The first requirement is that input flows must have a limited arrival rate in the long run in addition to limited burstness, which is the amount of data units (bits, cells, packets, etc.) that can arrive at once. The second requirement is that the networks must commit to providing a minimum service guarantee. Given that the basic requirements are met, one can use network calculus to compute bounds of very important performance parameters including the *backlog* and the *virtual delay*. A more detailed description of each performance parameter will be given later in this section. The rest of this section contains a quick overview of important results from network calculus theory that are used in this thesis.

2-2-1 Basic Concepts in Network Calculus

The two basic modeling elements in network calculus are the traffic *shaper* (also called *regulator*) and the *server* (also called *service node*). The shaper is the element used at the input of a network to force the data flow that is about to enter the network to conform to a specific predefined flow constraint expected by the service network. Shapers that are used to check conformance of the data without affecting the flow are called *policers*. The flow constraint is mathematically described by an *arrival curve*, often symbolized by α.

On the other hand, servers abstract physical network elements such as links, routers, and schedulers. The key power of network calculus is the ability to model each of the physically different elements along the flow path by the same network calculus element, that is the server. This is so powerful because then these elements can be lumped together according to the rules of network calculus into only one server representing the whole network, which extensively simplifies further analysis. Servers are characterized by the minimum service guarantee offered to an input flow that is mathematically described by a *service curve*, symbolized by β.

Remark 2.1. The two terms "traffic" and "flow" are used as synonyms referring to the collection of data transmitted from a certain source to a destination through the service network.

Definition 2.1 (**Process [21, §2.1]**). *A process $A : \mathbb{R} \mapsto \mathbb{R}^+_\varepsilon$ is a wide-sense increasing function of time, with $A(s) \leq A(t) \ \forall\, s \leq t$. In addition, the process $A(t)$ is said to be causal if it satisfies: $A(t) = 0 \ \forall\, t < 0$.*

A causal process, $A(t)$, can be used to count the data arrivals at a certain location in the network during the interval $[0, t]$. In that case, $A(t)$ is called an *arrival process*. On the other hand, if $A(t)$ is counting data departures, then it is called a *departure process*. The naming is merely a matter of perspective. It follows that for all $s \leq t$, $A(t) - A(s)$ is the number of arrivals or departures in the interval $(s, t]$.

The definition of process assumes an infinitely divisible data model, which is called the *fluid model*. Since, in reality, data is fundamentally discrete, the fluid model is viewed as an ideal model that is convenient for the general mathematical analysis. A depiction of the basic scenario considered in network calculus with a single flow passing through the network is given in Figure 2-1.

Figure 2-1: A basic network calculus model. $R(t)$ is an arrival process and $R^*(t)$ is a departure process.

Definition 2.2 (**Arrival Curve**). *The arrival curve is a non-negative wide-sense increasing function or sequence that specifies the maximum amount of arrivals allowed for a particular*

arrival process in a given time interval. Let $R(t)$ be an arrival process. We say that R is upper-constrained by an arrival curve α if and only if for all $0 \leq s \leq t$

$$R(t) - R(s) \leq \alpha(t-s) \tag{2-1}$$

which is equivalent to

$$R(t) \leq \inf_{0 \leq s \leq t} [R(s) + \alpha(t-s)] \stackrel{\text{def}}{=} (R \circledast \alpha)(t) \tag{2-2}$$

The right-hand side of the second inequality is defined as a min-plus *convolution* operation. More details on this operation will come later. Some texts refer to arrival curves as *departure curves* when they are used as traffic constraints for departure processes. For simplicity and since both are mathematically the same, we will use arrival curve as a general term to refer to any traffic constraint regardless of whether it is for an arrival or a departure process.

An important property to be possessed by an arrival curve is the *subadditivity*.

Definition 2.3 (**Subadditivity**). *A function $f : \mathbb{R} \mapsto \mathbb{R}_{\varepsilon}$ is said to be subadditive if and only if*

$$f(t+s) \leq f(t) + f(s), \quad \forall\, s, t \in \mathbb{R} \tag{2-3}$$

One of the most widely used class of arrival curves is the class of the *affine* curves, shown in Figure 2-2, that have the form

$$\alpha_{r,b}(t) = \begin{cases} rt + b, & \text{if } t > 0 \\ 0, & \text{if } t \leq 0 \end{cases} \tag{2-4}$$

where $r \geq 0$ is the peak rate and $b \geq 0$ is the burstiness limit.

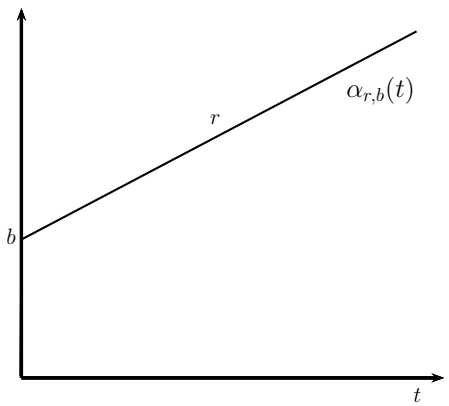

Figure 2-2: A generic affine curve

The affine arrival curve is obviously subadditive. It was first used by Cruz in [17] to explain the concept of the traffic constraint, which was later called arrival curve. Besides its simplicity,

the affine arrival curve is physically realizable by the *leaky bucket*, which is a traffic shaper introduced by Turner in [22]. In fact, it is proven that the leaky bucket is an optimal shaper for regulating traffic to have $\alpha_{r,b}$ as an arrival curve [19, Th. 2.2.12]. The details of the leaky bucket mechanism will be discussed in Chapter 3.

Definition 2.4 (**Service Curve**). *The service curve is a mathematical abstraction of the service guarantee offered by a server. A server is said to offer the service curve β to a traffic flow with an arrival process R and departure process R^* if and only if β is a wide-sense increasing function with $\beta(0) = 0$ and*

$$R^*(t) \geq \inf_{0 \leq s \leq t} [R(s) + \beta(t-s)] = (R \circledast \beta)(t) \tag{2-5}$$

Given a server with service curve β such as the one in Figure 2-3, Eq. (2-5) is equivalent to saying that $\forall t \geq 0$ there exits some $t_0 \geq 0$ with $t_0 \leq t$ such that

$$R^*(t) - R(t_0) \geq \beta(t - t_0)$$

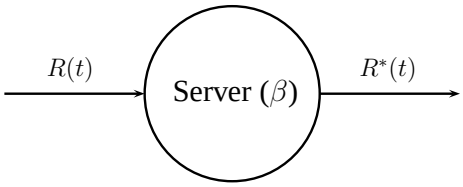

Figure 2-3: A basic model of a server with service curve β.

A server is said to offer a *strict* service curve if, during any backlogged (busy) period Δ, the output flow (departure process) is at least equal to $\beta(\Delta)$ [23]. This property provides an easy way to comprehend the more abstract concept of general service curves. One of the most widely used service curves is the *latency-rate* curve, shown in Figure 2-4, with the form

$$\beta_{c,d}(t) = c \cdot [t-d]^+ \tag{2-6}$$

where $c \geq 0$ is the rate and $d \geq 0$ is the latency of the server.

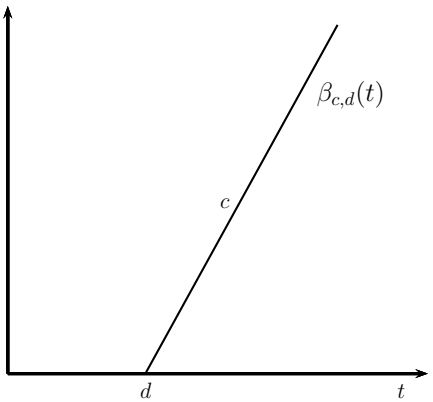

Figure 2-4: A generic latency-rate service curve

Servers that provide this service curve are called latency-rate servers. The latency-rate service curve is very useful to characterize many popular packet schedulers like the Weighted Fair Queuing (WFQ) [24], which can be generalized as guaranteed-rate (GR) [25] or latency-rate [26] schedulers. The GR service curve can be seen as a special case of the latency-rate service curve with $d = 0$.

2-2-2 Min-plus Algebra and Filtering Theory

The nonlinear mathematical relations described in (2-2) and (2-5) can be interpreted as linear relations under min-plus algebra. In min-plus algebra, *min* operation, denoted with $\oplus$, replaces the conventional addition and the usual addition operation, denoted with $\otimes$, substitutes the multiplication operation. According to the used terminology, the conventional algebra would be called "plus-times" algebra. Based on the properties of $\oplus$ and $\otimes$ operations, the algebraic structure $(\mathbb{R}_\varepsilon, \oplus, \otimes)$ is formally a *commutative dioid* [27]. More elaborately, besides the closure of $\mathbb{R}_\varepsilon$ under $\oplus$ and $\otimes$, operations in min-plus algebra possess the following properties:

1. **(Associativity of $\oplus$ and $\otimes$)**

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) \quad \forall\, a, b, c \in \mathbb{R}_\varepsilon$$

$$(a \otimes b) \otimes c = a \otimes (b \otimes c) \quad \forall\, a, b, c \in \mathbb{R}_\varepsilon$$

2. **(Commutativity of $\oplus$ and $\otimes$)**

$$a \oplus b = b \oplus a \quad \forall\, a, b \in \mathbb{R}_\varepsilon$$

$$a \otimes b = b \otimes a \quad \forall\, a, b \in \mathbb{R}_\varepsilon$$

3. **(Distributivity of $\otimes$ over $\oplus$)**

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c) \quad \forall\, a, b, c \in \mathbb{R}_\varepsilon$$

$$c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b) \quad \forall\, a, b, c \in \mathbb{R}_\varepsilon$$

4. **(Existence of null element $\varepsilon = +\infty$)**

$$\exists \varepsilon \in \mathbb{R}_\varepsilon \text{ such that } a \oplus \varepsilon = \varepsilon \oplus a = a \quad \forall\, a \in \mathbb{R}_\varepsilon$$

5. **(Absorption by null element ε)**

$$a \otimes \varepsilon = \varepsilon \otimes a = \varepsilon \quad \forall\, a \in \mathbb{R}_\varepsilon$$

6. **(Existence of unity element $e = 0$)**

$$\exists e \in \mathbb{R}_\varepsilon \text{ such that } a \otimes e = e \otimes a = a \quad \forall\, a \in \mathbb{R}_\varepsilon$$

7. **(Idempotency of $\oplus$)**

$$a \oplus a = a \quad \forall\, a \in \mathbb{R}_\varepsilon$$

The min-plus operations can be extended to matrices. Given the matrices $A, B \in \mathbb{R}_{\varepsilon}^{n\times n}$, $C \in \mathbb{R}_{\varepsilon}^{n\times m}$ and the constant $K \in \mathbb{R}_{\varepsilon}$, $\forall\, i,j \in \{1,\cdots,n\}$

$$\begin{aligned}[A \oplus B]_{i,j} &= [A]_{i,j} \oplus [B]_{i,j} \\ [K \otimes A]_{i,j} &= K \otimes [A]_{i,j} \\ [A \otimes C]_{i,j} &= \bigoplus_{v=1}^{n} [A]_{i,v} \otimes [C]_{v,j}\end{aligned}$$

The null matrix, denoted by ε, is a matrix with all elements equal to $+\infty$. The $n \times n$ identity matrix, denoted by E_n, is a matrix with all diagonal elements equal to 0 and all off-diagonal elements equal to $+\infty$. Note that, similar to the conventional matrix multiplication, the min-plus matrix multiplication is not commutative in general. The analysis can be further extended to functions and sequences. Given the two functions or sequences $f, g \in \mathcal{F}$

- Two functions are said to be equal, $f = g$, if $f(t) = g(t)$ for all t.
- A function is bounded by another, $f \leq g$, if $f(t) \leq g(t)$ for all t.
- Minimum of two functions is defined as a pointwise operation with

$$(f \oplus g)(t) = f(t) \oplus g(t)$$

- Convolution of two functions under min-plus algebra is defined as

$$(f \circledast g)(t) = \bigoplus_{s\in[0,t]} [f(s) \otimes g(t-s)]$$

- The minimum and convolution operations are monotonic. That is, $\forall f, f'g, g' \in \mathcal{F}$ if $f \leq f'$ and $g \leq g'$, then

$$\begin{aligned}f \oplus g &\leq f' \oplus g' \leq f' \\ f \circledast g &\leq f' \circledast g'\end{aligned}$$

The algebraic structure $(\mathcal{F}, \oplus, \circledast)$ is also a commutative dioid with the null element ε, defined as $\varepsilon(t) = +\infty \ \ \forall t$, and the identity element δ, defined as $\delta(t) = 0 \ \ \forall t \leq 0$ and $\delta(t) = +\infty \ \ \forall t > 0$.

Remark 2.2. Network calculus results are often presented in the literature using inf and sup operations instead of the min and max operations. This distinction only matters for the continuous-time case where the least upper limit or the greatest lower limit of a function $f \in \mathcal{F}$ over an interval $X \subseteq \mathbb{R}$ generally does not belong to $f(X)$. This can occur when X is an open or a half-closed interval. In the discrete-time case, inf (resp. sup) and min (resp. max) can be used interchangeably.

The input/output relationship of the server in Eq. (2-5) greatly resembles those of the *linear time-invariant* (LTI) filters. Borrowing the terminology from the conventional filtering theory, the service curve, β is the impulse response of the server. Unlike the conventional LTI filters, the input-output relationship of servers is in general governed by an inequality. Servers characterized by equality in Eq. (2-5) instead of inequality are called *exact* servers. Shapers

that are used to model the mechanisms of traffic regulation can be also modeled as min-plus linear filters as well. The most prominent example of such shapers is the *greedy shaper*. A shaper is said to be greedy if it delays the input data units in a buffer to avoid the violation of an arrival curve constraint α, but outputs them as soon as possible otherwise. This useful property allows a greedy shaper with subadditive arrival curve α to be modeled as an exact server with service curve α [28],[29].

The filtering theory can be extended to time varying systems introduced in [30],[31]. However, for the purpose of this thesis, the discussion is limited to time-invariant systems.

2-2-3 Important Results from Network Calculus

The literature contains many useful results of network calculus that are used by the community. The network calculus analysis was successfully applied to the window flow control problem by Cruz and Okino in [32] and later by Chang in [33]. Internet quality of service is perhaps the most prominent field of application of network calculus, for example, the guaranteed services in the Integrated Services (IntServ) networks [34],[35]. Network calculus concepts were also used to define the Expedited Forwarding Per-Hop Behavior [36],[37] of the Differentiated Services (DiffServ) [38] network architecture. A comprehensive overview of Internet quality of service is given by Firoiu et al. in [39]. Many current parameter-based and measurement-based admission control schemes use network calculus to achieve quality of service. The following discussion will include the most important results that are relevant to this thesis.

Definition 2.5 (**Backlog**). *The backlog, $B(t)$, is formally the amount of data that is seen at the input to the server at time t but not yet at the output. Let $R(t)$ be the arrival process at the input and $R^*(t)$ be the departure process at the output. In a network where data is only forwarded, the backlog can be calculated as*

$$B(t) = R(t) - R^*(t) \tag{2-7}$$

If the network is composed of a single buffer, the backlog would correspond to the queue size at any instant. A server is said to be *causal* if and only if the backlog is never negative.

Theorem 2.1 (**Backlog Bound [16, Th. 1.4.1]**). *Let α be the arrival curve that constrains the input traffic and β be the service curve guarantee offered by the server. The backlog, $B(t)$, is bounded by*

$$B(t) \leq \sup_{s \geq 0} [\alpha(s) - \beta(s)] = v(\alpha, \beta) \tag{2-8}$$

where $v(\alpha, \beta)$ is practically the maximum vertical distance between the α curve and the β curve. If $v(\alpha, \beta)$ is infinite, then the backlog in the server is unbounded.

Definition 2.6 (**Virtual Delay**). *The virtual delay, $d(t)$, is the amount of time that a data unit that entered the network at time t has to wait until it is serviced if all the previous units were serviced before it. Let $R(t)$ be the arrival process at the input and $R^*(t)$ be the departure*

process at the output. In a network where data is only forwarded, the virtual delay can be calculated as

$$d(t) = \inf [\tau \geq 0 \mid R(t) \leq R^*(t + \tau)] \tag{2-9}$$

Theorem 2.2 (Virtual Delay Bound [16, Th. 1.4.2]). *Let α be the arrival curve that constrains the input traffic and β be the service curve guarantee offered by the server. The virtual delay, $d(t)$, is bounded by*

$$d(t) \leq \sup_{s \geq 0} [\inf [\tau \geq 0 \mid \alpha(s) \leq \beta(s + \tau)]] = h(\alpha, \beta) \tag{2-10}$$

where $h(\alpha, \beta)$ is the maximum horizontal distance between the α curve and the β curve as shown in Figure 2-5. Same as with the backlog, if $h(\alpha, \beta)$ is infinite, then the virtual delay in the server is unbounded.

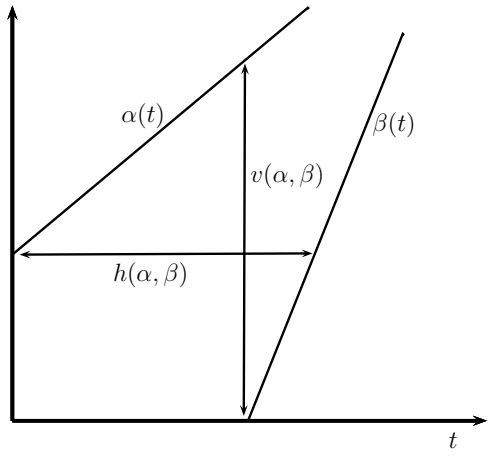

Figure 2-5: Performance bounds of a server based on the arrival and service curves

The bounds given by the previous two theorems are tight as long as both α and β are wide-sense increasing functions with $\alpha(0) = \beta(0) = 0$ and α being subadditive [16, Th. 1.4.4]. A powerful result of network calculus is the ability to replace a group of servers in tandem by an equivalent server. Consider a situation shown in Figure 2-6

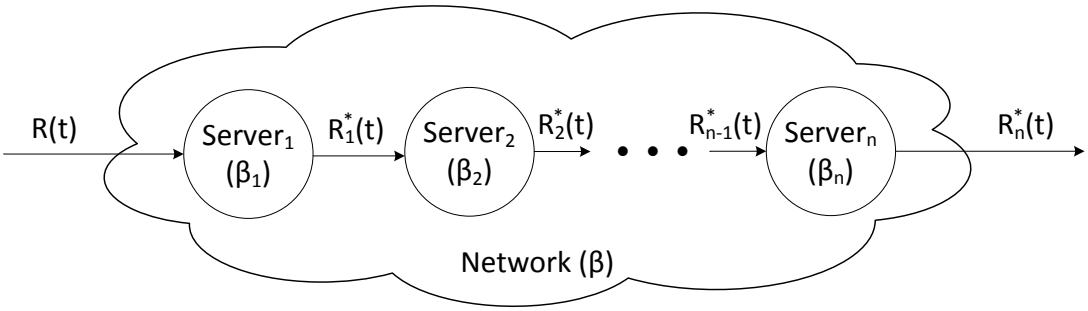

Figure 2-6: A flow passing through a group of servers in tandem.

Theorem 2.3 (**Concatenation of Servers [16, Th. 1.4.6]**). *Given a traffic flow that passes through n servers in tandem each having a service curve β_i for $i = 1, \cdots, n$. The flow will experience the same service guarantee as that offered by a single server with service curve β that is computed as*

$$\beta = \beta_1 \circledast \beta_2 \circledast \cdots \circledast \beta_n \tag{2-11}$$

This theorem allows reducing very complicated problems to ones that are easier to analyze. In this way, the whole network can be substituted with a single server that will provide an equivalent service guarantee to the traffic flow. Finally, since min-plus convolution is commutative, the order of the servers is irrelevant and has no effect on the overall service experienced by the traffic flow.

2-3 Systems and Relations

In this thesis, systems are mathematical models of dynamical phenomena. In order to transcend problem specific models of dynamical phenomena, a unified versatile notion of system is used. As will be apparent in the discussion, this notion is equipped with powerful tools to establish relationships among different systems.

Definition 2.7 (**System [2, §1.1]**). *A* system *S is a sextuple $(X, X_0, U, \delta, Y, H)$ consisting of:*

- *the set of states X;*
- *the set of initial states $X_0 \subseteq X$;*
- *the set of inputs U;*
- *a transition relation $\delta \subseteq X \times U \times X$;*
- *the set of outputs Y;*
- *an output map $H : X \mapsto Y$*

The transition relation prescribes the way the system can evolve. Note that the $(x, u, x') \in \delta$ is represented by $x \xrightarrow{u} x'$ for simplicity. Also, $U(x) \subseteq U$ represents the set of inputs possible at state x. The properties that can be possessed by a system are summarized below:

- A system is *finite-state* if X is a finite set
- A system is *countable* if X is a countable set
- A system is *blocking* if there is a state $x \in X$ such that $U(x) = \varnothing$
- A system is *deterministic* if for any state $x \in X$ and any input $u \in U(x)$, $x \xrightarrow{u} x'$ and $x \xrightarrow{u} x''$ implies that $x' = x''$
- A system is *output deterministic* if for any state $x \in X$ and any inputs $u, u' \in U(x)$, $x \xrightarrow{u} x'$ and $x \xrightarrow{u'} x''$ with $H(x') = H(x'')$ implies that $x' = x''$

Since a system models dynamical phenomenon, understanding the way the system evolves over its state space is very important to characterize the dynamics.

Definition 2.8 (**System Behavior [2, §1.2]**). *Given a system $S = (X, X_0, U, \delta, Y, H)$, the system internal behavior generated from some state $x \in X$ is a sequence of transitions over the system state space starting from state x*

$$x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} \cdots \xrightarrow{u_{n-2}} x_{n-1} \xrightarrow{u_{n-1}} x_n$$

where $x_0 = x$. Using the output map, the internal behavior from $x \in X$ defines an external behavior with $H(x_i) = y_i \in Y$ for all $i \in \{0, \cdots, n\}$

$$y_0 \longrightarrow y_1 \longrightarrow \cdots \longrightarrow y_{n-1} \longrightarrow y_n$$

The behavior is *infinite* if the transition sequence is nonterminating. The set of finite (resp. infinite) external behaviors of a system S that are defined by the internal behaviors generated from a state x is denoted by $\mathcal{B}_x(S)$ (resp. $\mathcal{B}^\omega_x(S)$). As a result, the finite (resp. infinite) external behavior of a system is the set of all external behaviors generated from initial set X_0 denoted by $\mathcal{B}(S)$ (resp. $\mathcal{B}^\omega(S)$), that is

$$\mathcal{B}(S) = \bigcup_{x \in X_0} \mathcal{B}_x(S) \quad \text{and} \quad \mathcal{B}^\omega(S) = \bigcup_{x \in X_0} \mathcal{B}^\omega_x(S)$$

If a system is *nonblocking*, then $\mathcal{B}^\omega(S) \neq \varnothing$. Infinite behaviors are well-suited to model behaviors of reactive systems that are supposed to interact indefinitely with their environments. When S is infinite-state it is possible to create a finite-state symbolic model called the *quotient system*. This can be done with the help of an equivalence relation Q on state set of S. As a result, states that are related by the equivalence relation are indistinguishable from each other in the quotient system.

Definition 2.9 (**Quotient System [2, §4.2]**). *Let $S = (X, X_0, U, \delta, Y, H)$ and let Q be an equivalence relation on X such that $(x, x') \in Q$ implies $H(x) = H(x')$. The* quotient *of S by Q, denoted by $S_{/Q}$, is the system $(X_{/Q}, X_{/Q0}, U_{/Q}, \delta_{/Q}, Y_{/Q}, H_{/Q})$ consisting of:*

- $X_{/Q} = X/Q$;
- $X_{/Q0} = \left\{x_{/Q} \in X_{/Q} \mid x_{/Q} \cap X_0 \neq \varnothing\right\}$;
- $U_{/Q} = U$;
- $x_{/Q} \xrightarrow[/Q]{u} x'_{/Q}$ *if there exists* $x \xrightarrow{u} x'$ *in* S *with* $x \in x_{/Q}$ *and* $x' \in x'_{/Q}$;
- $Y_{/Q} = Y$;
- $H_{/Q}(x_{/Q}) = H(x)$ *for some* $x \in x_{/Q}$

For the symbolic model to be finite, there must be a finite number of equivalence classes induced by a finite equivalence relation. Based on the previous definition, the equivalence relation Q is finite when the set of outputs Y (or, more precisely, the range of H) is finite.

2-3-1 System Relations

An important relation that can be established between systems is the *simulation* relation.

Definition 2.10 (**Simulation Relation [2, §4.2]**). *Let* S_a *and* S_b *be two systems with* $Y_a = Y_b$. *A relation* $R \subseteq X_a \times X_b$ *is a* simulation relation *from* S_a *to* S_b *if the following conditions hold:*

1. *for every* $x_{a0} \in X_{a0}$, $\exists x_{b0} \in X_{b0}$ *such that* $(x_{a0}, x_{b0}) \in R$;
2. *for every* $(x_a, x_b) \in R$, *it is true that* $H_a(x_a) = H_b(x_b)$;
3. *for every* $(x_a, x_b) \in R$, *it is true that* $x_a \xrightarrow[a]{u_a} x'_a$ *in* S_a *implies the existence of* $x_b \xrightarrow[b]{u_b} x'_b$ *in* S_b *satisfying* $(x'_a, x'_b) \in R$

Given two systems S_a and S_b with $Y_a = Y_b$, system S_a is simulated by system S_b, denoted by $S_a \preceq_{\mathcal{S}} S_b$, if there exists a simulation relation from S_a to S_b. In more concrete terms, the simulation relation implies that all the behaviors of S_a are also behaviors of S_b, which could probably have more behaviors [2, Prop 4.9]. That is, $\mathcal{B}^\omega(S_a) \subseteq \mathcal{B}^\omega(S_b)$, which is called *behavioral inclusion*, and is denoted by $S_a \preceq_{\mathcal{B}} S_b$. Hence, proving that a property holds for the behaviors of S_b automatically proves it for those of S_a. However, no conclusion can be made about the behaviors of S_a if the property does not hold for S_b.

Theorem 2.4 (**[2, Th. 4.18]**). *Let* $S = (X, X_0, U, \delta, Y, H)$ *and let* Q *be an equivalence relation on* X *such that* $(x, x') \in Q$ *implies* $H(x) = H(x')$. *The relation* Γ *defined as:*

$$\Gamma(\pi_Q) = \left\{(x, x_{/Q}) \in X \times X_{/Q} \mid x_{/Q} = \pi_Q(x)\right\}$$

is a simulation relation from S *to* $S_{/Q}$.

Definition 2.11 (**Bisimulation**). *Given two systems* S_a *and* S_b *with* $Y_a = Y_b$, S_a *and* S_b *are said to be bisimilar, denoted by* $S_a \cong_{\mathcal{S}} S_b$, *if and only if* $S_a \preceq_{\mathcal{S}} S_b$ *and* $S_b \preceq_{\mathcal{S}} S_a$.

Bisimulation is a strong notion of equivalence between systems which also implies behavioral equivalence, that is the behaviors of bisimilar systems are equivalent [2, Prop 4.14]. Hence, any property proven for one of the systems implies that the other one possesses the same property. Also, showing that a property fails to hold in one system sufficiently implies the same for the other system .

2-4 Formal Verification

System verification is the procedure of showing that the design under consideration possesses certain properties that come from the system specification. The system specification describes the desired behavior of the system, that is what the system should do and what should not. If all the properties obtained from the specification are satisfied, the system is said to be *correct*. Hence, the correctness is always relative to the system specification.

Over the years, engineers have devised and used many verification techniques to help them discover the errors in their designs. The main objective, especially for safety-critical systems, is to uncover as many design errors as possible and as early as possible. As the system become more complex, the complexity of verification increases drastically rendering many manual or ad-hoc techniques unfeasible. For this reason, more systematic approaches were introduced such as the *formal methods*.

Formal verification methods aim at establishing system correctness with mathematical rigor. In other words, they target the problem of mathematically proving that systems satisfy the properties they need to satisfy. This makes them great candidates for verifying safety-critical systems that require great deal of reliability. The following discussion in this section covers a brief introduction to formal verification by model checking in addition to an overview of the temporal logic that is used to formulate system specifications.

2-4-1 Model Checking

In model-based design approaches, an unambiguous mathematical description of a system is given in the form of a transition system (TS). The main advantage of such descriptions is that the verification on these models can be automated via *model checkers*.

Definition 2.12 (**Model Checking**). *Model checking refers to a formal verification method that is based on the exhaustive exploration of the system model in order to make inferences about the system behavior. In addition, model checkers are software tools that perform model checking.*

A comprehensive presentation of model-checking methods is given in the book by Baier and Katoen [40]. Because it is based on exploring all the possible system states, model checking is applicable to systems with finite state space. Thus, to verify an infinite-state system via model checking the system must be abstracted by a finite state transition system, that i.e. symbolic model. For the verification of the system to be valid, the symbolic model must at least simulate the original system [2]. Establishing the simulation relation ensures that the behavior of the abstracted system is included in the behavior of the symbolic model.

In order to verify the properties of the model, the property specifications must be mathematically formalized in a way that can be verified by model checkers. Temporal logics offer a great framework to mathematically formulate system specification.

2-4-2 Linear Temporal Logic (LTL)

Temporal logic is a logical formalism that extends propositional and predicate logic by modal operators that allow describing infinite behavior of systems. Temporal logic is well suited for

reactive systems, where the correctness of operation depends on the timeliness of the behavior in addition to input/output computation. In temporal logic, the underlying nature of time can be either linear or branching. These different views of time in addition to other factors gave rise to different flavors of temporal logic that are collectively referred to as temporal logics. Note that time here is referred to in the abstract sense, in which time is discrete and advances as the system evolves from one state to the next. Introduced by Pnueli [41], LTL is one prominent example of temporal logic that is based on the linear time perspective. Intuitively, an LTL formula specifies a property that should apply to all behaviors starting from the initial set. The following discussion includes a review of the syntax and semantics of LTL.

Definition 2.13 (**Syntax of LTL [40, §5.1.1]**). *LTL formulas over the set of atomic propositions AP is generated by the following grammar:*

$$\varphi ::= true \mid a \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

where $a \in AP$

Regarding the precedence, the unary operators bind stronger than the binary ones and the $\mathcal{U}$ (until) operator takes precedence over the propositional logic operators ($\vee$, $\wedge$, $\neg$). The parentheses have the highest precedence, while $\neg$ and $\bigcirc$ (next) bind equally strong. The temporal modalities, $\Diamond$ and $\Box$, can be defined in terms the $\mathcal{U}$ operator as the following:

$$\Diamond\varphi \stackrel{\text{def}}{=} \text{true}\,\mathcal{U}\varphi \qquad \Box\varphi \stackrel{\text{def}}{=} \neg\Diamond\neg\varphi$$

Intuitively, $\Diamond\varphi$ ensures that φ will finally be true in future, while $\Box\varphi$ ensures that φ will always be true from now and forever in future.

Semantics of LTL formulas can be defined over infinite sequences $\mathsf{z} \in (2^{AP})^\omega$

Definition 2.14 (**Semantics of LTL [40, §5.1.2]**). *Let φ be an LTL formula over a set of atomic propositions AP. Given a satisfaction relation $\models \subseteq (2^{AP})^\omega \times LTL$, the semantics of LTL formula φ is defined as a language $L(\varphi) = \left\{\mathsf{z} \in (2^{AP})^\omega \mid \mathsf{z} \models \varphi\right\}$, which is the set of all infinite sequences over 2^{AP} that satisfy φ under the following satisfaction rules*

- $\mathsf{z} \models true;$
- $\mathsf{z} \models a \iff a \in \mathsf{z}(0);$
- $\mathsf{z} \models \varphi_1 \vee \varphi_2 \iff \mathsf{z} \models \varphi_1$ *or* $\mathsf{z} \models \varphi_2;$
- $\mathsf{z} \models \varphi_1 \wedge \varphi_2 \iff \mathsf{z} \models \varphi_1$ *and* $\mathsf{z} \models \varphi_2;$
- $\mathsf{z} \models \neg\varphi \iff \mathsf{z} \not\models \varphi;$
- $\mathsf{z} \models \bigcirc\varphi \iff \mathsf{z}[1] \models \varphi;$
- $\mathsf{z} \models \varphi_1 \mathcal{U} \varphi_2 \iff \exists k \geq 0$ *s.t.* $\mathsf{z}[k] \models \varphi_2$ *and* $\mathsf{z}[j] \models \varphi_1 \, \forall j \in \{0, \cdots, k-1\};$

Let S be a system with $Y = 2^{AP}$ and consequently $H : X \mapsto 2^{AP}$. The system S satisfies φ, denoted $S \models \varphi$, if $\mathcal{B}^\omega(S) \subseteq L(\varphi)$.

Let $S_{/Q}$ be a quotient system of S constructed as given in Definition 2.9. Because of the behavioral inclusion implied by the simulation relation, when $S_{/Q}$ is shown to satisfy a property φ it is sufficient to conclude that S also satisfies the same property. Mathematically,

$$\mathcal{B}^\omega(S) \subseteq \mathcal{B}^\omega(S_{/Q}) \wedge \mathcal{B}^\omega(S_{/Q}) \subseteq L(\varphi) \implies \mathcal{B}^\omega(S) \subseteq L(\varphi)$$

Remark 2.3 ([40, Rem. 5.9]). Note that the statements $S \not\models \varphi$ and $S \models \neg\varphi$ are not equivalent in general. However, it is true that $S \models \neg\varphi$ implies $S \not\models \varphi$.

Two LTL formulas φ_1 and φ_2 are said to be equivalent, denoted $\varphi_1 \equiv \varphi_2$, if $L(\varphi_1) = L(\varphi_2)$. The following are some important equivalence rules for LTL:

$$\begin{aligned}
\neg \bigcirc \varphi &\equiv \bigcirc \neg\varphi \\
\neg \Diamond \varphi &\equiv \Box \neg\varphi \\
\neg \Box \varphi &\equiv \Diamond \neg\varphi \\
\bigcirc (\varphi_1 \mathcal{U} \varphi_2) &\equiv (\bigcirc \varphi_1) \mathcal{U} (\bigcirc \varphi_2) \\
\Diamond (\varphi_1 \vee \varphi_2) &\equiv (\Diamond \varphi_1) \vee (\Diamond \varphi_2) \\
\Diamond (\varphi_1 \wedge \varphi_2) &\not\equiv (\Diamond \varphi_1) \wedge (\Diamond \varphi_2) \\
\Box (\varphi_1 \wedge \varphi_2) &\equiv (\Box \varphi_1) \wedge (\Box \varphi_2) \\
\Box (\varphi_1 \vee \varphi_2) &\not\equiv (\Box \varphi_1) \vee (\Box \varphi_2)
\end{aligned}$$

Chapter 3

Problem Formulation

3-1 Problem Description

The main premise of network calculus is that when a data traffic that conforms to a certain arrival curve passes through a network with some service curve guarantee, the upper bounds of the backlog and virtual delay can be calculated from the maximum vertical and horizontal translations between the arrival and service curves. The upper bounds in Eq. (2-8) and Eq. (2-10) theoretically require infinite-horizon computations. In simple cases, these computations might still be possible however that would not be as general as one would like to have as a modeling framework. As a remedy to the problem a slightly different approach will be taken based on the min-plus filtering theory discussed in the previous chapter. In the system theoretic view of network calculus, the server in a network acts as a min-plus linear filter with an impulse response equal to its service curve. By exploiting this idea, an alternative mathematical description of the server can be obtained by representing the server as a state-space system. The state-space description of the server will prepare the system for the application of finite abstraction techniques to get a finite-state system that represents the server. It is important to note that since the exact input-output mapping of the server is unknown, the min-plus convolution of the input and the service curve captures the worst-case scenario for the departure process or the output flow. Hence, if the inequality in the server equation changed to equality, the new equation would represent an exact server that provides a service equivalent to the worst-case possible service of the actual server. Since the target of the analysis is the worst-case scenario in the server, assuming the worst-case output would be a very natural choice. Since the attention is shifted from the curves to the worst-case flow propagation through the server, the upper bounds of the delay and backlog can be obtained by directly checking $R(t)$ and $R^*(t)$ since $R^*(t)$ would be the worst possible departure. However, in order for the basic premise of network calculus to hold, the input flow must always conform to an arrival that is known apriori. Therefore, the system must check the conformance of the input flow with the expected traffic behavior besides propagating the flow through the server as shown in Figure 3-1. As a result, one can say that as long as the input passes the conformance check, correct inferences can be made about the delay and backlog of the server based on $R(t)$ and $R^*(t)$.

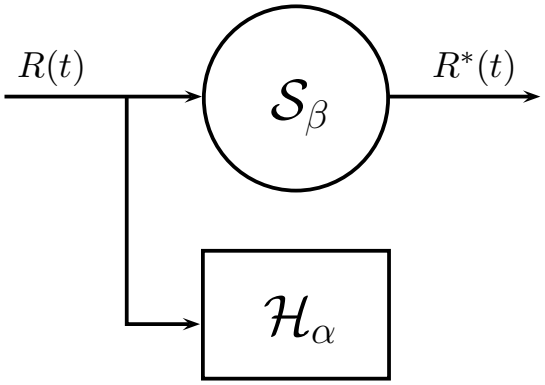

Figure 3-1: General model with a single server $\mathcal{S}$ and a policer (conformance checker) $\mathcal{H}$

The two systems work simultaneously together to capture the dynamics of a single server with a constrained input flow. $\mathcal{H}$ is constructed based on the required arrival curve α of the input flow. $\mathcal{S}$ is constructed based on the expected service curves β of the server.

The derivation of the state-space representation will assume a sampled version of the ideal fluid model. This means that the arrival curves, service curves, and processes will be sequences taken from set $\mathcal{F}_0$ (cf. Section 2-1). Let t_s be the sampling period, the discrete-time sequences are defined as the following:

$$R|_{\mathbb{Z}\cdot t_s} = u$$
$$R^*|_{\mathbb{Z}\cdot t_s} = y$$

where $\mathbb{Z}\cdot t_s = \{\cdots -2t_s,\ -t_s,\ 0,\ t_s,\ 2t_s,\ 3t_s,\ \dots\}$. The sampling period is assumed to be small enough to capture all the details needed. Without loss of generality, t_s will be taken to be equal to 1 in this thesis. To avoid ambiguity, the discrete variable $k \in \mathbb{Z}$ will be used as a time parameter of the functions from now on to emphasize that they are sequences. In other words, $\alpha(k)$, $\beta(k)$, $u(k)$, and $y(k)$ are all discrete-time sequences. As a result, the general model will look like Figure 3-2

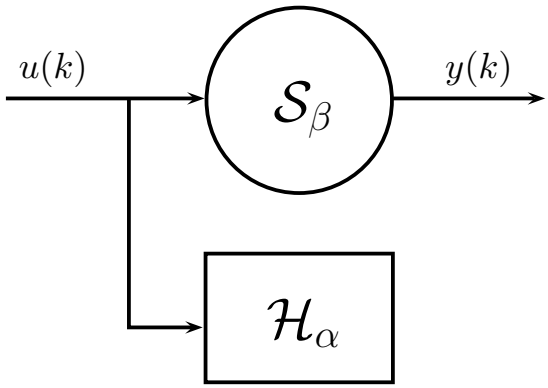

Figure 3-2: Discrete-time model

3-2 State-Space Representation of Deterministic Servers

To be able to derive the state-space representation of a server, its impulse response or the service curve β must be known. Network calculus provides a variety of service curves that can be used to characterize different network elements. Because of its versatility in modeling, the latency-rate curve will be used in the derivation. Latency-rate service curve can be mathematically represented as the min-plus convolution of two other simpler curves [16]. In other words, the latency-rate server is equivalent to a concatenation of two other simpler servers, namely the *maximum delay server* and the *guaranteed rate server*. Exploiting this fact will not only make the derivation more elegant, but also give the state-space representation of the constituent servers as a byproduct. In the rest of the section, the state-space representation of these simpler servers is first discussed and then combined together to form the representation of the latency-rate server.

3-2-1 Maximum-Delay Servers

Traffic transmitted through a network is subject to arbitrary delays. To account for these delays, network calculus offers a very useful model of worst-case delay in networks. This network element is the maximum-delay server [19] which can be characterized by the service curve δ_d, shown in Figure 3-3 defined as

$$\delta_d(k) = \begin{cases} +\infty, & \text{if } k > d \\ 0, & \text{if } k \leq d \end{cases} \tag{3-1}$$

where $d \in \mathbb{N}$ is the maximum delay experienced by an input traffic.

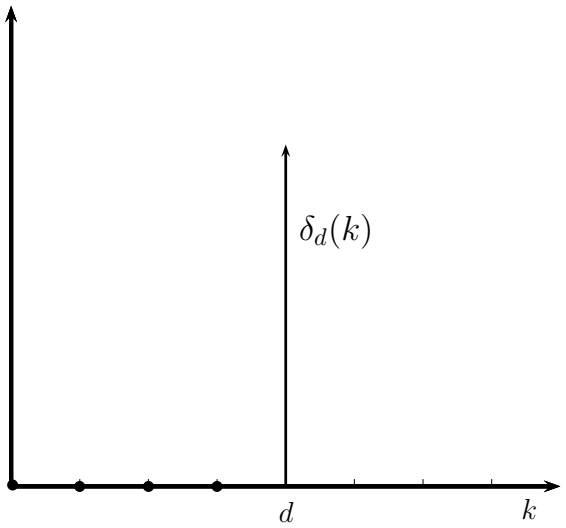

Figure 3-3: Maximum-delay service curve

Maximum-delay service curves are sometimes referred as delay-burst curves [16]. Let y_L

denote the output of the delay server.

$$\begin{aligned} y_L(k) &\geq (\delta_d \circledast u)(k) \\ &= \bigoplus_{s=0}^{k} [\delta_d(k-s) + u(s)] \\ &= u([k-d]^+) \end{aligned} \tag{3-2}$$

The expression of the worst-case output $y_L(k)$, can be rewritten by introducing internal states $v_1, v_2, \dots$ in the following manner:

$$\begin{aligned} v_1(k) &= u(k) \\ v_i(k) &= v_{i-1}(k-1), \quad i = 2, \dots, d \\ y_L(k) &= v_d(k-1) \end{aligned} \tag{3-3}$$

The obtained form in Eq. (3-3) is the state-space representation of the maximum-delay, or more correctly the exact-delay, server.

3-2-2 Guaranteed-Rate (GR) Servers

Guaranteed-rate servers are characterized by the service curve λ_c, shown in Figure 3-4, defined as

$$\lambda_c(k) = \begin{cases} c \cdot k, & \text{if } k > 0 \\ 0, & \text{if } k \leq 0 \end{cases} \tag{3-4}$$

where $c \in \mathbb{R}^+$ is the service rate.

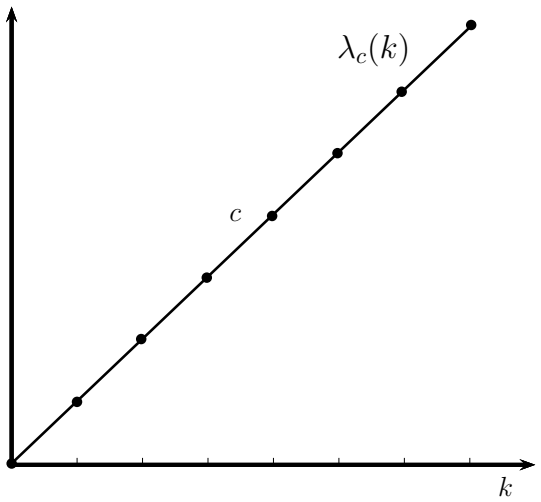

Figure 3-4: Guaranteed-rate service curve

The output $y_R(k)$ of a GR server is

$$\begin{aligned} y_R(k) &\geq (u \circledast \lambda_c)(k) \\ &= \bigoplus_{s=0}^{k} [\lambda_c(k-s) + u(s)] \\ &= \bigoplus_{s=0}^{k} [c \cdot (k-s) + u(s)] \\ &= w(k) \end{aligned} \tag{3-5}$$

where w is the state variable of the GR server. From Eq. (3-5), the state evolution equation can be derived as the following

$$\begin{aligned} w(k) &= \bigoplus_{s=0}^{k} [c \cdot (k-s) + u(s)] \\ &= \bigoplus_{s=0}^{k} [c \cdot (k-1-s) + c + u(s)] \\ &= \bigoplus_{s=0}^{k-1} [c \cdot (k-1-s) + c + u(s)] \oplus (c \cdot (k-1-k) + c + u(k)) \\ &= \left(c + \bigoplus_{s=0}^{k-1} [c \cdot (k-1-s) + u(s)] \right) \oplus u(k) \\ &= (c + w(k-1)) \oplus u(k) \end{aligned} \tag{3-6}$$

3-2-3 Latency-Rate Servers

Recall from Chapter 2 that the latency-rate server is characterized by the service curve $\beta_{c,d}$ defined as

$$\beta_{c,d}(k) = \begin{cases} c \cdot (k-d), & \text{if } k > d \\ 0, & \text{if } k \leq d \end{cases} \tag{3-7}$$

Based on the fact that $\beta_{c,d} = \delta_d \circledast \lambda_c$, the output $y(k)$ of the latency-rate server can be represented as

$$\begin{aligned} y(k) &\geq (u \circledast \beta_{c,d})(k) \\ &= (u \circledast (\delta_d \circledast \lambda_c))(k) \\ &= ((u \circledast \delta_d) \circledast \lambda_c)(k) \end{aligned} \tag{3-8}$$

The state-space representation of the latency-rate server will be derived by concatenating the delay and guaranteed-rate servers together. Assuming that the output of the delay server enters the input of the guaranteed-rate server, the state evolution equation of the guaranteed-rate server is equivalent to

$$\begin{aligned} w(k) &= (c + w(k-1)) \oplus y_L(k) \\ &= (c \otimes w(k-1)) \oplus v_d(k-1) \end{aligned} \tag{3-9}$$

Let $x_1, \ldots, x_{d+1}$ denote the state variables of the latency-rate server. The state variables $v_1, \ldots, v_d$ of the delay server are assigned to $x_1, \ldots, x_d$, and the state variable w of the guaranteed-rate server is assigned to x_{d+1}. As a result, the state-space representation of the latency-rate server is

$$\begin{aligned} x_1(k) &= u(k) \\ x_i(k) &= x_{i-1}(k-1), \quad i = 2, \ldots, d \\ x_{d+1}(k) &= x_d(k-1) \oplus (c \otimes x_{d+1}(k-1)) \\ y(k) &= x_{d+1}(k) \end{aligned} \tag{3-10}$$

3-3 Conformance Checking

Conformance checking is important for making correct inferences about quality of service provided by the server. If the input does not conform to the predefined arrival curve, the backlog and delay bounds would be violated. In real networks, policers make sure that incoming traffic is upper constrained by the expected arrival curve. One of the most widely used shaper/policer, is the leaky bucket [22]. As mentioned in Chapter 2, the leaky bucket is optimal for shaping a traffic to have an affine arrival curve $\alpha_{r,b}$. The discussion in this section will focus on the policing or conformance checking aspect of leaky buckets.

Remark 3.1. There is some ambiguity in the literature about the term leaky bucket. To prevent any confusion, it must be noted that the term used in this thesis refers to the same mechanism as the one discussed by Cruz in [18] and Chang in [19]. This version of leaky bucket is sometimes referred to as *token bucket*.

Figure 3-5 shows an illustrative model of a discrete-time leaky bucket

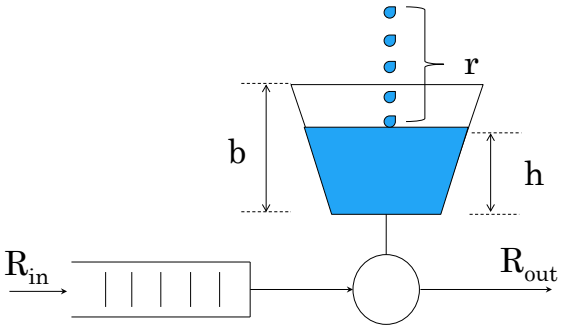

Figure 3-5: Discrete-time leaky bucket

The discrete-time leaky bucket is composed of a token buffer (bucket) with a token generation rate r and a maximum buffer size b. Such a leaky bucket is also referred to as (b,r)-leaky bucket. As long as the buffer is not full, r tokens are leaked into the buffer every time unit. When data arrives, enough tokens must be available in the buffer in order to allow the data to move forward. If the amount of data arrivals at some point in time is greater than the amount of tokens available in the buffer, a traffic violation is detected and the flow is considered nonconformant. If leaky bucket is used as a shaper, then it will delay to

nonconformant data arrivals until enough tokens become available in the buffer, after which it releases the data. Based on the description, one can represent the content of the bucket by state h. Mathematically, the trajectory $h(k)$ should obey the following:

$$h(k) = \min\left[h(k-1) + r - a(k)\,,\, b\right] \qquad h(0) = b \tag{3-11}$$

where $a(k) = u(k) - u(k-1)$ is the number of data arrivals at time k with $a(0) = 0$. As a result, the conformance checking of the leaky bucket is transformed to checking whether the following condition is satisfied or not

$$h(k) \geq 0 \qquad \forall k \in \mathbb{Z} \tag{3-12}$$

Eq. (3-11) can be further manipulated to obtain a simpler form. Let $z(k) = h(k) + u(k)$ with $z(0) = h(0) + u(0) = b + u(0)$.

$$\begin{aligned} z(k) &= \min\left[h(k-1) + r - (u(k) - u(k-1))\,,\, b\right] + u(k) \\ &= \min\left[h(k-1) + r - \cancel{u(k)} + u(k-1) + \cancel{u(k)}\,,\, b + u(k)\right] \\ &= \min\left[r + z(k-1)\,,\, b + u(k)\right] \end{aligned} \tag{3-13}$$

The conformance condition becomes

$$z(k) - u(k) \geq 0 \qquad \forall k \in \mathbb{Z} \tag{3-14}$$

A very important property of leaky buckets is that several different leaky buckets can be concatenated to be used for more complicated arrival curves.

Theorem 3.1 (**Concatenation of Leaky Buckets [19, §2.3.3]**). *Consider the concatenation of n (b_i,r_i)-leaky buckets with $i \in \{1, \ldots, n\}$. The resulting shaper can be characterized by a subadditive piecewise-affine arrival curve*

$$\alpha = \alpha_{r_1,b_1} \circledast \alpha_{r_2,b_2} \circledast \cdots \circledast \alpha_{r_n,b_n} = \alpha_{r_1,b_1} \oplus \alpha_{r_2,b_2} \oplus \cdots \oplus \alpha_{r_n,b_n} \tag{3-15}$$

Moreover, the concatenation of leaky buckets is equivalent to the parallel composition of the same leaky buckets all joined by the $\oplus$ operation at the output as shown in Figure 3-6.

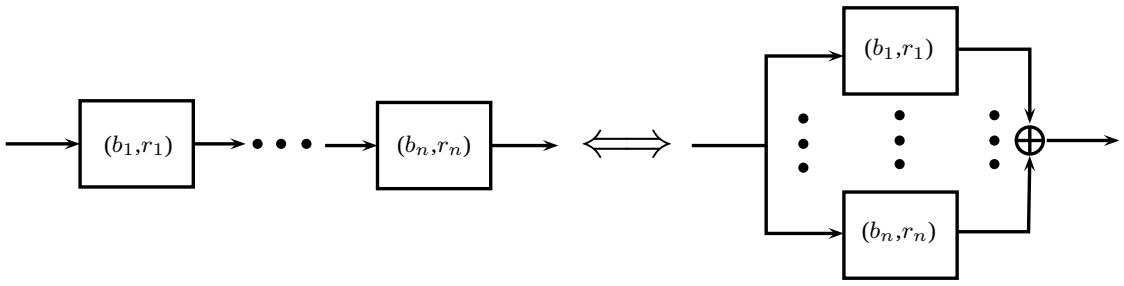

Figure 3-6: Equivalence of serial and parallel compositions of leaky buckets

This theorem can be used to create an $\mathcal{H}$ system for more complicated piecewise-affine arrival curves. All what needs to be done is to represent each affine piece by a (b,r)-leaky bucket. Then, the dynamics of each leaky bucket can be described by equations similar to Eq. (3-13). Finally, the conformance checking condition will be equivalent to checking if all of the individual leaky bucket conditions are satisfied simultaneously or not.

3-4 Combined Dynamics of $\mathcal{S}$ and $\mathcal{H}$

The dynamics of the two systems $\mathcal{S}$ and $\mathcal{H}$ can be combined by taking the Cartesian product of their state spaces. Let the state variables x_1 to x_{d+1} represent the state variables of the latency-rate server. Note that for each leaky bucket considered in $\mathcal{H}$, a single state is needed in the combined system. Assuming that $\mathcal{H}$ represents a single leaky bucket, the additional state variable x_{d+2} denotes the state variable z related to the leaky bucket dynamics. The dynamics of the resulting system can be represented as the following:

$$\begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_d(k) \\ x_{d+1}(k) \\ x_{d+2}(k) \end{bmatrix} = \left[\begin{array}{ccccc|c} \infty & \infty & \cdots & \infty & \infty & \infty \\ 0 & \infty & \cdots & \infty & \infty & \infty \\ \infty & 0 & \cdots & \infty & \infty & \infty \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \infty & \infty & \cdots & 0 & c & \infty \\ \hline \infty & \infty & \cdots & \infty & \infty & r \end{array}\right] \otimes \begin{bmatrix} x_1(k-1) \\ x_2(k-1) \\ \vdots \\ x_d(k-1) \\ x_{d+1}(k-1) \\ x_{d+2}(k-1) \end{bmatrix} \oplus \left[\begin{array}{c} 0 \\ \infty \\ \vdots \\ \infty \\ \infty \\ \hline b \end{array}\right] \otimes u(k) \quad (3\text{-}16)$$

$$y(k) = x_{d+1}(k) \quad (3\text{-}17)$$

with the following initial conditions:

$$\begin{bmatrix} x_1(0) & x_2(0) & \cdots & x_{d+1}(0) & x_{d+2}(0) \end{bmatrix}^T = \begin{bmatrix} \mu & \mu & \cdots & \mu & (b+\mu) \end{bmatrix}^T$$

where μ is the number of data units that already passed through the system in the past.

In simple terms, the initial condition corresponds to an empty backlog and a full leaky bucket.

Chapter 4

Finite Abstractions and Property Specification

To be able to apply model checking techniques, the modeled infinite-state systems must be abstracted by finite-state quotient systems, also called finite abstractions or symbolic models. This chapter will target the problem of generating finite abstractions of dynamical systems like the one obtained in the end of Chapter 3. Since these dynamical systems are min-plus linear (MiPL) systems, the discussion in this chapter is general and could be applied to other MiPL systems. The approach that is implemented mainly depends on a novel technique developed by Adzkiya et al. [42],[43] for Max-Plus Linear (MPL) systems [27] based on the piecewise-affine (PWA) [44] representation of the MPL dynamics. The underlying mathematical structure of MiPL systems is very similar to MPL systems. This similarity stems from the fact that both types of systems are founded on the commutative dioid algebraic structure, which gives them the same algebraic properties. The main difference between the two structures is that the "min" operation is substituted by the "max" operation (with its null element $\varepsilon = -\infty$) for max-plus algebra. This difference has a minor effect on the abstraction procedure adapted from [43]. After obtaining the quotient system, the model is ready for verification via model checking. The last section discusses how to formulate the backlog and virtual delay bounds as LTL properties that can be checked against the symbolic model.

4-1 Min-Plus Linear (MiPL) Systems

Min-plus linear (MiPL) systems are dynamical systems with the following form:

$$x(k) = A \otimes x(k-1) \oplus B \otimes u(k) \tag{4-1}$$

where $A \in \mathbb{R}_{\varepsilon}^{n \times n}$, $B \in \mathbb{R}_{\varepsilon}^{n \times m}$, $x \in \mathbb{R}_{\varepsilon}^{n}$, $u \in \mathbb{R}_{\varepsilon}^{m}$ for all $k \in \mathbb{N}$. When the MiPL is autonomous, all the elements of B are ε. As for other discrete-time dynamical systems, a smooth map $f : \mathbb{R}_{\varepsilon}^{n} \times \mathbb{R}_{\varepsilon}^{m} \mapsto \mathbb{R}_{\varepsilon}^{n}$ describes the current state reached by the MiPL system based on the current input u and the previous state x. For MiPL systems, f is linear in the min-plus

algebraic sense (cf. Section 2-2-2). In other words, each component of f is a min-plus linear combination of state and input variables.

Definition 4.1 (**MiPL Dynamical System**). *Formally, an* MiPL dynamical system *can be defined as a quadruple* $\Sigma = (\mathbb{R}_\varepsilon^n, \mathbb{R}_\varepsilon^m, A, B)$ *consisting of:*

- *the state space* $\mathbb{R}_\varepsilon^n$*;*
- *the input space* $\mathbb{R}_\varepsilon^m$*;*
- *the matrices* $A \in \mathbb{R}_\varepsilon^{n\times n}$ *and* $B \in \mathbb{R}_\varepsilon^{n\times m}$ *that describe the state evolution*

Definition 4.2 (**Infinite-State System associated with the MiPL Dynamics**). *Let* $\Sigma = (\mathbb{R}_\varepsilon^n, \mathbb{R}_\varepsilon^m, A, B)$ *be an MiPL dynamical system, AP be a set of atomic propositions defined over its state-space , and* $I \subseteq \mathbb{R}_\varepsilon^n$ *be a specified set of initial states. The infinite-state system associated with* Σ*, AP, and I is a sextuple, denoted by* $S_{AP,I}(\Sigma)$*, consisting of:*

- $X = \mathbb{R}_\varepsilon^n$*;*
- $X_0 = I$*;*
- $U = \mathbb{R}_\varepsilon^m$*;*
- $x \xrightarrow{u} x'$ *if* $\exists u \in U(x)$ *such that* $x' = A \otimes x \oplus B \otimes u$*;*
- $Y = 2^{AP}$*;*
- $H = X \mapsto 2^{AP}$

The choice of atomic propositions for a particular MiPL system mainly depends on the underlying problem and the design preferences. Since verification is the targeted problem of this thesis, the set of atomic propositions are selected to facilitate the formulation of the system properties as LTL formulas for model checking. This will become clearer in Section 4-3. An equivalence relation Q over the state space of a dynamical system is said to be *output-preserving* if the following condition holds

$$(x, x') \in Q \implies H(x) = H(x')$$

The partition induced by such an equivalence relation is called output-preserving partition.

4-2 State-Space Partitioning Procedure

In a nutshell, the abstraction technique, adapted from [43], treats the MiPL as a PWA system by partitioning the state space into regions of different affine dynamics. This partition is refined by an equivalence relation based on the output map of the states. As a result, states that belong to the same equivalence class abide by the same affine dynamics and are mapped to the same output, i.e. the set of atomic propositions. The obtained set of equivalence classes makes up the state space of the quotient system. The last step in creating the quotient system is to compute the allowed transitions between the equivalence classes.

PWA representation of MiPL Dynamics

In general, PWA systems [44] are described by a set of affine dynamics together with their corresponding regions in the input/state space. The PWA regions have non-overlapping interiors and each can be described by a set of linear inequalities in the input/state space. Let $g = (g_1, \cdots, g_n) \in \{1, \cdots, n\}^n$ be a vector of coefficients for the affine dynamics. The PWA representation of an autonomous MiPL dynamical system ($x(k-1) = A \otimes x(k)$) is

$$R_g = \bigcap_{i=1}^{n} \bigcap_{\substack{j=1 \\ j \neq g_i}}^{n} \{x \in \mathbb{R}^n \mid [A]_{i,g_i} + x_{g_i} \leq [A]_{i,j} + x_j\} \tag{4-2}$$

$$x_i(k) = [A]_{i,g_i} + x_{g_i}(k-1) \quad \forall\, i \in \{1, \cdots, n\} \tag{4-3}$$

A point in the state space belongs to the region R_g if

$$\bigoplus_{j=1}^{n} ([A]_{i,j} + x_j) = [A]_{i,g_i} + x_{g_i} \quad \forall\, i \in \{1, \cdots, n\}$$

Example 4.1. *Consider the following autonomous MiPL system*

$$\begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix} \otimes \begin{bmatrix} x_1(k-1) \\ x_2(k-1) \end{bmatrix} = \begin{bmatrix} \min[\,1 + x_1(k-1)\,,\, 2 + x_2(k-1)] \\ \min[\,4 + x_1(k-1)\,,\, 3 + x_2(k-1)] \end{bmatrix}$$

Since A is a (2×2)-matrix, there are at most four different regions in the state space, namely: $R_{(1,1)}$, $R_{(1,2)}$, $R_{(2,1)}$, and $R_{(2,2)}$. Applying Eq. (4-2) region $R_{(1,1)}$ can be obtained as

$$\begin{aligned} R_{(1,1)} &= \{x \in \mathbb{R}^n \mid 1 + x_1 \leq 2 + x_2\} \cap \{x \in \mathbb{R}^n \mid 4 + x_1 \leq 3 + x_2\} \\ &= \{x \in \mathbb{R}^n \mid x_1 - x_2 \leq 1\} \cap \{x \in \mathbb{R}^n \mid x_1 - x_2 \leq -1\} \\ &= \{x \in \mathbb{R}^n \mid x_1 - x_2 \leq -1\} \end{aligned}$$

The dynamics corresponding to region $R_{(1,1)}$ according to Eq. (4-3) are

$$\begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} = \begin{bmatrix} 1 + x_1(k-1) \\ 4 + x_1(k-1) \end{bmatrix}$$

Other regions can be computed in a similar fashion to obtain

$$\begin{aligned} R_{(1,2)} &= \{x \in \mathbb{R}^n \mid -1 \leq x_1 - x_2 \leq 1\} \\ R_{(2,2)} &= \{x \in \mathbb{R}^n \mid x_1 - x_2 \geq 1\} \\ R_{(2,1)} &= \varnothing \end{aligned}$$

The PWA regions in the state space together with the corresponding affine dynamics are given in Figure 4-1

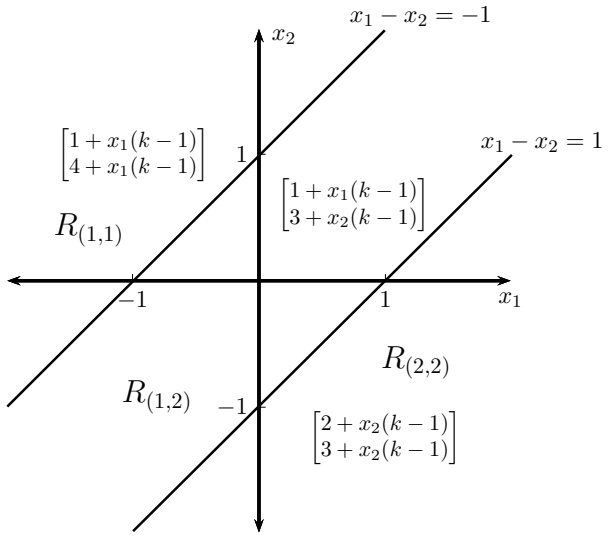

Figure 4-1: PWA regions and their corresponding affine dynamics of the autonomous MiPL example

Algorithm 1 can be used to automatically obtain a PWA representation of an MiPL system.

Algorithm 1 Compute the PWA representation of A

Input: $A \in \mathbb{R}_{\varepsilon}^{n \times p}$, an $(n \times p)$ matrix
Output: R^{PWA}, the set of PWA regions; A^{PWA}, and B^{PWA}, PWA dynamics

1: **function** MIPL2PWA(A)
2: Initialize R^{PWA}, A^{PWA}, and B^{PWA}
3: **for all** $g = (g_1, \cdots, g_n) \in \{1, \cdots, p\}^n$ **do**
4: $R_g \leftarrow \mathbb{R}^p$, $\mathsf{A}_g \leftarrow [0]^{n \times p}$, $\mathsf{B}_g \leftarrow [0]^n$
5: **for all** $i \in \{1, \cdots, n\}$ **do**
6: **for all** $j \in \{1, \cdots, p\}$ **do**
7: $R_g \leftarrow R_g \cap \{[A]_{i,g_i} + x_{g_i} \leq [A]_{i,j} + x_j\}$
8: **end for**
9: $[\mathsf{A}_g]_{i,g_i} \leftarrow 1$, $[\mathsf{B}_g]_i \leftarrow [A]_{i,g_i}$
10: **end for**
11: **if** $R_g \neq \varnothing$ **then**
12: $R^{\text{PWA}} \leftarrow R^{\text{PWA}} \cup \{R_g\}$, $A^{\text{PWA}} \leftarrow A^{\text{PWA}} \cup \{\mathsf{A}_g\}$, $B^{\text{PWA}} \leftarrow B^{\text{PWA}} \cup \{\mathsf{B}_g\}$
13: **end if**
14: **end for**
15: **return** R^{PWA}, A^{PWA}, and B^{PWA}
16: **end function**

Algorithm 1 is adapted from [43, Alg. 2], with the only difference being in line 7 where $\leq$ substitutes $\geq$. This is necessary to take care of the fact that the particular affine dynamics that apply in a certain region of the MiPL state space is the minimum one rather than the maximum one as in MPL systems. The worst-case time complexity of Algorithm 1 is

$\mathcal{O}\left(p^n(np+p^3)\right)$ [43, §II-C]. Note that the given algorithm does not require the A matrix to be square, which is important for later usage.

Each PWA region can be represented by a data structure called Difference-Bound Matrix [45].

Definition 4.3 (**Difference-Bound Matrix (DBM)**). *A DBM D in $\mathbb{R}^n$ is an $(n+1)$-dimensional matrix representing the intersection of finitely many sets defined by*

$$\bigcap_{i,j=0}^{n} \{x \in \mathbb{R}^n \mid x_i - x_j \bowtie [D]_{i+1,j+1}\}$$

where $x_0 = 0$, $\bowtie \in \{<, \leq\}$, and $[D]_{i,j} \in \mathbb{R} \cup \{+\infty\}$ for all $i, j \in \{1, \cdots, n+1\}$.

DBMs are particularly interesting to PWA systems because they can be manipulated efficiently, especially when it comes to computing the image of a DBM through the affine dynamics [43].

The set R^{PWA} obtained in Algorithm 1 is a cover of the state space. The set R^{PWA} is not a partition of the state space because boundaries between regions are shared between these regions. To obtain a partition, another procedure must be implemented to resolve the common boundaries and attach them to one region only. Two regions that share the same boundary are called *adjacent regions*

Definition 4.4 (**Adjacent Regions [43, Def. 9]**). *Let R_g and $R_{g'}$ be two regions in $\mathbb{R}^n$. The two regions are said to be adjacent, denoted by $R_g \geq R_{g'}$, if there exists an $i \in \{1, \cdots, n\}$ such that $g_i > g'_i$ and $g_j = g'_j$ for all $j \neq i$.*

Algorithm 2 [43, Alg. 6] loops over all the PWA regions obtained in Algorithm 1 and refines them to make each common boundary belong to exactly one of the adjacent regions. As a result, the regions will be pairwise disjoint and, hence, forming a partition of the state space.

Algorithm 2 Generate a partition of the state space by boundary refinements of PWA regions

Input: R^{PWA}, the set of PWA regions forming a cover of $\mathbb{R}^n$
Output: R^{PWA}, set of regions forming a partition of $\mathbb{R}^n$

```
1: function PWARegionRefine(R^PWA)
2:     for all R_g, R_g' ∈ R^PWA do
3:         if R_g ≥ R_g' then
4:             R_g' ← R_g' \ R_g
5:             if R_g' = ∅ then
6:                 R^PWA = R^PWA \ {R_g'}
7:             end if
8:         end if
9:     end for
10:    return R^PWA
11: end function
```

The worst-case time complexity of Algorithm 2 is $\mathcal{O}\left(n^{2n+1}\right)$ [43, §III-B].

Refinement to Obtain an Output-Preserving Partition

Let $AP = \{a_1, a_2, \cdots, a_r\}$ be a finite set of atomic propositions, each of which is defined over a DBM region in the state space with $|AP| = r$. Then, the set of DBMs corresponding to all atomic propositions is denoted by $D_{AP} = \{D_{a_1}, D_{a_2}, \cdots, D_{a_r}\}$ with D_{a_i} being the DBM of a_i for all $a_i \in AP$. In addition, let D^c_{AP} be the set of DBM complements defined as $D^c_{AP} = \{D^c_{a_1}, D^c_{a_2}, \cdots, D^c_{a_r}\}$, where $D^c_{a_i} = \mathbb{R}^n \setminus D_{a_i}$ for all $i \in \{1, \cdots, r\}$. Note that each element in D^c_{AP} is in general not a DBM but a finite union of DBMs. Let $\ell = (\ell_1, \ell_2, \cdots, \ell_r) \in \{0,1\}^r$ a vector of boolean coefficients. Given the set of r atomic propositions, the state space can be partitioned into regions, denoted R_ℓ, where

$$R_\ell = \bigcap_{i=1}^{r} \begin{cases} D_{a_i}, & \text{if } \ell_i = 1 \\ D^c_{a_i}, & \text{if } \ell_i = 0 \end{cases}$$

Proposition 4.1. *The set regions $\mathcal{P}_{AP} = \{R_\ell \subseteq \mathbb{R}^n \mid \forall\, \ell \in \{1,0\}^r$ such that $R_\ell \neq \varnothing\}$ is a partition of $\mathbb{R}^n$.*

Proof. To prove the proposition, one must show that $\mathcal{P}_{AP}$ is a cover of $\mathbb{R}^n$ and that its elements are nonempty pairwise-disjoint regions.

- The first statement is equivalent to showing that $\mathbb{R}^n$ is a subset of $\mathcal{P}_{AP}$. Let x be a point $\mathbb{R}^n$. For each $i \in \{1, \cdots, r\}$ check whether $x \in D_{a_i}$ is true or false. When true, set $\ell_i = 1$, otherwise set $\ell_i = 0$. Thus, x is an element of R_ℓ with $\ell = (\ell_1, \cdots, \ell_r)$.
- To prove the second part, one can say that, by definition, all the elements of $\mathcal{P}_{AP}$ are nonempty. Then, suppose that there exist ℓ and ℓ' such that $\ell \neq \ell'$ and $R_\ell \cap R_{\ell'} \neq \varnothing$. Because $\ell \neq \ell'$, there exists an $i \in \{1, \cdots, r\}$ such that $\ell_i \neq \ell'_i$. Without loss of generality, assume that $\ell_i = 1$ and $\ell'_i = 0$. Notice that $R_\ell \subseteq D_{a_i}$ and $R_{\ell'} \subseteq D^c_{a_i}$. Since $D_{a_i} \cap D^c_{a_i} = \varnothing$, it must be true that $R_\ell \cap R_{\ell'} = \varnothing$, which contradicts the initial assumption. Thus, for any $\ell \neq \ell'$, $R_\ell \cap R_{\ell'} = \varnothing$.

□

As a final step before computing the transitions, the partition obtained from Algorithm 2 can be refined with the partition $\mathcal{P}_{AP}$ to obtain an output-preserving partition of the state space. This can be done with the aid of Algorithm 3. Algorithm 3 takes each region in the PWA partition and, depending on where it lies with respect to the $\mathcal{P}_{AP}$ partition, divides it into smaller regions such that each new region is mapped to a single output. As a result, the set of equivalence classes in the refined partition can be used as the finite state space of the quotient system.

Computation of Transitions

The state-space partitioning is done by considering only the autonomous part of the state evolution equation, i.e. independent from the input space. To compute the transitions between the states (X/Q) of the quotient system, the input space must be taken into consideration. This can be done in three steps. The first step is to obtain the PWA representation in the

Algorithm 3 Generate an output-preserving partition of the state space by refinment

Input: R^{PWA}, a partition of $\mathbb{R}^n$ based on the PWA dynamics; $\mathcal{P}_{AP}$, a partition of $\mathbb{R}^n$ based on the atomic propositions

Output: X/Q, an output-preserving partition of $\mathbb{R}^n$

```
1: function PartitionRefine(R^PWA, P_AP)
2:     X/Q ← ∅
3:     for all R_g ∈ R^PWA do
4:         for all R_ℓ ∈ P_AP do
5:             if R_ℓ ∩ R_g ≠ ∅ then
6:                 X/Q ← X/Q ∪ {R_ℓ ∩ R_g}
7:             end if
8:         end for
9:     end for
10:    return X/Q
11: end function
```

input/state augmented space. Basically, A and B can be augmented together in one matrix $\bar{A} = [A\, B]$ and the nonautonomous MiPL system becomes

$$x(k) = [A\, B] \otimes \begin{bmatrix} x(k-1) \\ u(k) \end{bmatrix} \tag{4-4}$$

To obtain the PWA representation of the augmented space, $\bar{A}$ is passed as an input to Algorithm 1. The only purpose of the augmented PWA representation is to help in computing the transitions.

The second step of finding the transitions is to create a finite partition of the input space $U \subseteq \mathbb{R}^m$. This procedure is very similar to the procedure of partitioning the state space according to the set atomic propositions that was mentioned earlier in Section 4-2. The only difference is that, in the case, the space considered is the input space $U \subseteq \mathbb{R}^m$ rather than the state space $\mathbb{R}^n$. The set of regions forming the partition of the input space is denoted by $U/Q \subseteq \mathbb{R}^m$, which is defined as $U/Q = \{U_1, U_2, \cdots, U_{|U/Q|}\}$ such that $U_i \cap U_j = \varnothing \, \forall \, i \neq j$.

The final step of computing the transitions is to apply the forward reachability approach [43, §III-C] to form the transitions between the quotient states. Given a set of inputs $U/Q \subseteq \mathbb{R}^m$, a transition exists between two s and s' in the quotient system if and only if $\exists U_i \in U/Q$ for some $i \in \{1, \cdots, |U/Q|\}$ such that $f(X_s, U_i) \cap X_{s'} \neq \varnothing$, where $X_s = \{x \in \mathbb{R}^n \mid \pi_Q(x) = s'\}$ and $X_{s'} = \{x \in \mathbb{R}^n \mid \pi_Q(x) = s'\}$. Computing the next state set $f(X_s, U_i)$ requires the usage of the PWA representation of the augmented system. This is done by finding the Cartesian product $X_s \times U_i$ and then intersecting it with the PWA regions of the augmented space. Depending on where it lies in augmented space, the proper PWA dynamics are used to compute the image of $X_s \times U_i$. The image of $X_s \times U_i$ is then intersected with $X_{s'}$ to determine if a transition from s to s' is possible or not. This procedure is automated in Algorithm 4. Algorithm 4 is very similar to [43, Alg. 9] and therefore a similar complexity analysis can be applied. The complexity of computing the image of $f(X_s, U_i)$ through the dynamics and then computing the intersection with $X_{s'}$, line 5, is $\mathcal{O}\left((n+m)^3 \cdot |\bar{R}^{\text{PWA}}|\right)$ [43, §III-C-2]. The loop over all the inputs, line 4, has the worst-case time complexity of $\mathcal{O}(|U/Q|)$, and the outer loop has

Algorithm 4 Computing transitions between the states of the quotient system

Input: X/Q, a partition of $\mathbb{R}^n$; $U/Q \subseteq \mathbb{R}^m$, the input set; $\bar{A} \in \mathbb{R}_{\varepsilon}^{n \times n+m}$, an augmented matrix; $\bar{R}^{\text{PWA}}$, the PWA regions of the augmented system;
Output: $\delta_{/Q} \subseteq X/Q \times U/Q \times X/Q$, a transition relation

```
function ComputeTransition(X/Q, U/Q, Ā, R̄^PWA )
    δ_/Q ← ∅
    for all s, s' ∈ X/Q do
        for all U_i ∈ U/Q do
            if f(X_s, U_i) ∩ X_s' ≠ ∅ then
                δ_/Q ← δ_/Q ∪ {(s, U_i, s')}
            end if
        end for
    end for
    return δ_/Q
end function
```

the worst-case complexity of $\mathcal{O}\left(|X/Q|^2\right)$. Finally, the overall worst-case complexity is the product of the individual complexities, which is $\mathcal{O}\left((n+m)^3 \cdot |\bar{R}^{\text{PWA}}| \cdot |U/Q| \cdot |X/Q|^2\right)$.

By construction, the obtained quotient system, $S_{/Q}(\Sigma)$, in general simulates the infinite-state system, $S(\Sigma)$ (cf. Theorem 2.4).

4-3 Formulation of the verification properties

As discussed earlier, one of the main goals of this thesis is to transform the problems concerned with maximum backlog and virtual delay into verification problems. By doing so, the reasoning can be automated via model checking instead of manual computation. As a result, the reasoning procedure could be incorporated with the automatic controller synthesis tools to generate more optimized controllers based on network analysis. The following discussion shows how to translate the delay and backlog bounds into LTL formulas.

The first step is to decide about the atomic propositions that will be used. A clever choice of atomic propositions can allow a smooth and intuitive translation to LTL property specification. The chosen set of atomic propositions for this problem is given below

$$AP = \{\mathsf{ConformantInput},\ \mathsf{BacklogBounded},\ \mathsf{DelayBounded}\}$$

where,

- ConformantInput is true wherever the input is conformant, i.e. conformance checker detects no violation of arrival constraint in the past time step.
- BacklogBounded is true whenever the backlog is bounded by some predefined parameter $B_{max} \in \mathbb{R}$
- DelayBounded is true whenever the virtual delay is bounded by some predefined parameter $D_{max} \in \mathbb{N}$

These atomic propositions can be defined as DBMs in the state space. Going back to the system in Section 3-4, ConformantInput DBM can be defined as the following:

$$\mathsf{ConformantInput} = \{x \in \mathbb{R}^n \mid x_{d+2}(k) - x_1(k) \geq 0\} \tag{4-5}$$

To simplify the discussion, states of $S_{/Q}$ where ConformantInput is true are referred to as conformant states. On the other hand, states where ConformantInput is false are referred to as nonconformant states. The DBM of the other two atomic propositions are discussed in their corresponding subsections.

4-3-1 Backlog Property

Although, the input is unpredictable, the conformance checking mechanism embedded in the model allows making inferences about the past incoming signals. Recall from network calculus, the backlog is bounded whenever the input abides by an arrival constraint. Referring to the system in Section 3-4 and given B_{max} as an upper bound of the backlog, the BacklogBounded DBM can be defined as the following

$$\begin{aligned} \mathsf{BacklogBounded} &= \{x \in \mathbb{R}^n \mid 0 \leq u(k) - y(k) \leq B_{max}\} \\ &= \{x \in \mathbb{R}^n \mid 0 \leq x_1(k) - x_{d+1}(k) \leq B_{max}\} \end{aligned} \tag{4-6}$$

When thinking about the LTL property formulation, one can notice that the properties that are of interest apply to certain paths (trajectories) in the system and not for others. Verbally, the system must satisfy the property that along all paths over which states are always conformant, it is also true that the backlog is always bounded by the predefined backlog bound. The quotient system contains many paths that include nonconformant states (i.e. $\neg$ConformantInput). However, those paths are not very interesting to the discussion. After all, the backlog bound is not supposed to be respected if the input to the system at some point was not conformant. On the hand, the backlog bound must be respected as long as the input is conformant.

Based on intuition, a LTL formulation can be written as the following

$$\varphi_1 = \square\,(\mathsf{ConformantInput}) \implies \square\,(\mathsf{BacklogBounded}) \tag{4-7}$$

Notice that the LTL formula φ_1 will not be satisfied by the symbolic model if there is at least one path over ConformantInput states which also includes $\neg$BacklogBounded states. In addition, there are no backlog restrictions over paths that include $\neg$ConformantInput states as they always satisfy φ_1.

Note that φ_1 is a well-formed LTL formula and therefore can be verified using an LTL model checker such as SPIN [46]. It is not hard to see that the verification of the backlog property depends on the backlog parameter B_{max}, which might be imposed as a design parameter. If the property is satisfied by the symbolic model, then one can confidently say that backlog in the actual system is bounded by the given parameter. However, if the property is not satisfied, no conclusion can be made about the backlog bound in the actual system. When the property is not satisfied, the model checking procedure can be iteratively applied with each time increasing the parameter B_{max} until the property is satisfied.

4-3-2 Virtual Delay Property

Following the same idea of the backlog property, the virtual delay property is formulated to check if the virtual delay is bounded over all the paths where the input is conformant. Using Eq. (2-9) of the virtual delay, the statement that the virtual delay in a discrete-time server is bounded by $D \in \mathbb{N}$ is mathematically equivalent to

$$u(k) \leq y(k+D_{max}) \iff u(k-D_{max}) \leq y(k) \iff u(k-D_{max})-y(k) \leq 0 \quad \forall\, k \in \mathbb{N} \tag{4-8}$$

Alternatively, any value of $D \in \mathbb{N}$ that satisfies the inequality (4-8) for all k is also an upper bound of the virtual delay. The LTL property formulation is based on this observation. Checking the value of the input D_{max} units back in time requires storing the history of the input for at least $D+1$ time units. This can be done by introducing $D+1$ state variables in the following way

$$\begin{aligned} x_1(k) &= u(k) \\ x_i(k) &= x_{i-1}(k-1) = u(k-(i-1)) \quad \forall\, i \in \{2, \cdots, D_{max}+1\} \end{aligned}$$

Looking back at the system in Section 3-4, one can see that there are already some states that store the input, namely x_1 to x_d. However, if there is a need to store the input for a longer time ($D_{max} > d$), additional state variables can be added to the system. After the introduction of the additional state variables if needed, the atomic proposition DelayBounded can be defined as a DBM in the state space.

$$\begin{aligned} \mathsf{DelayBounded} &= \{x \in \mathbb{R}^n \mid u(k-D_{max}) - y(k) \leq 0\} \\ &= \{x \in \mathbb{R}^n \mid x_{D_{max}+1}(k) - y(k) \leq 0\} \end{aligned} \tag{4-9}$$

The verification property of virtual delay, following the same arguments for the backlog property, can be formulated in LTL as the following

$$\varphi_2 = \square\,(\mathsf{ConformantInput}) \implies \square\,(\mathsf{DelayBounded}) \tag{4-10}$$

As with the formula of the backlog bound, if φ_2 is not satisfied, no conclusion can be made about the actual system. This might require adjusting D_{max} iteratively until the formula is satisfied for some value of D_{max}.

The backlog and delay properties can be combined and represented by one LTL formula as the following

$$\begin{aligned} \varphi &= \varphi_1 \wedge \varphi_2 \\ &= \square\,(\mathsf{ConformantInput}) \implies \square\,(\mathsf{BacklogBounded} \wedge \mathsf{DelayBounded}) \end{aligned} \tag{4-11}$$

The combined LTL formula φ checks whether the the backlog and delay bounds are respected simultaneously. Because it is a conjunction of two well-formed LTL formulas, φ is also a well-formed LTL formula.

Chapter 5

Example

After presenting the theoretical results, it makes sense to work on an example to assess the overall procedure. All the proposed procedures to build the MiPL system and to generate the finite abstraction have been implemented as MATLAB scripts and are included in Appendix B. The presentation in this chapter is in the form of steps that can be applied to any similar example. The example that is discussed is based on the system derived in Chapter 3, which basically captures the dynamics of a single server together with a conformance-checking leaky bucket as shown in Figure 5-1.

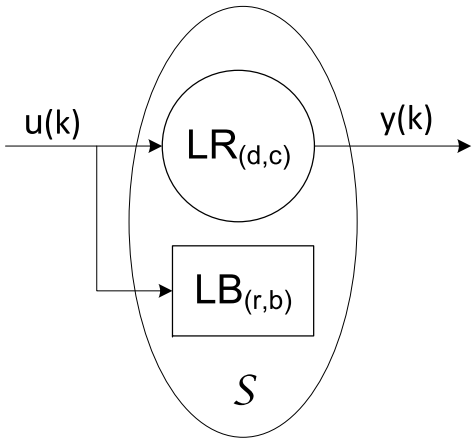

Figure 5-1: Latency-Rate server with a leaky bucket conformance checker used in the example

The parameters used are given in Table 5-1 below

Table 5-1: Table of parameters

Server Latency(d)	Server Rate (c)	Leaky Bucket Burstiness (b)	Leaky bucket Rate (r)
3	5	2	4

The parameters were arbitrarily chosen for the purpose of the example. Besides keeping the values small so that the state space does not become very large, the service rate was chosen to be greater than the rate of the leaky bucket, $c > r$. This is because the server must cope with the incoming data rate, or otherwise the backlog and delay cannot be bounded. After choosing the system parameters, a choice must be made for maximum delay D_{max} that needs to be verified. Recall from Section 4-3-2, the value of D_{max} dictates the number of additional states that must be added to the system in order to store the input for enough time units. Applying the theorem in Section 2-2-3, the maximum virtual delay in the server is equal to $d + c/b = 5.5$. The parameter D_{max} is chosen to be 6. The maximum backlog from network calculus is $r \cdot d + b = 14$ and it is the choice made for B_{max} to be used in the atomic proposition of the backlog. Note that the choices of D_{max} and B_{max} need not to be based on network calculus calculations as they are supposed to be problem dependent. In this case, the computation via network calculus is simple. However, the ultimate goal is to have these parameters passed as a design requirements to be checked if they are satisfied via model checking.

Step 1: Define the state-space representation of the system

With this choice of parameters the system state-space representation ($x(k) = A \otimes x(k-1) \oplus B \otimes u(k)$) is as the following:

$$A = \left[\begin{array}{ccccccccc} \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & 0 & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \hline \infty & \infty & 0 & \infty & \infty & \infty & \infty & 5 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 4 \end{array}\right] \quad B = \left[\begin{array}{c} 0 \\ \infty \\ \infty \\ \hline \infty \\ \infty \\ \infty \\ \infty \\ \hline \infty \\ 2 \end{array}\right] \tag{5-1}$$

Comparing the above matrices with the ones in Eq. (3-17), one can notice the effect of adding the 4 additional state variables (represented by the middle 4 rows in A and B) in order to keep the input as far as $D_{max} = 6$ units in time. Note that the seventh column is empty because the value of the input is not stored further.

After defining the state-space representation of the system in Figure 5-1 as prescribed in Section 3-4, one can start applying the abstraction techniques to obtain a symbolic model.

Step 2: Define the DBMs of the atomic propositions and partition the state space accordingly

According to the state-space representation, $u(k) = x_1(k)$, $u(k - D_max) = x_7(k)$, and $y(k) = x_8(k)$. Based on this observation, the DBMs of the atomic propositions are defined as the following:

- ConformantInput $= \{x \in \mathbb{R}^9 \mid x_9 - x_1 \geq 0\}$
- BacklogBounded $= \{x \in \mathbb{R}^9 \mid 0 \leq x_1 - x_8 \leq B_{max} = 14\}$
- DelayBounded $= \{x \in \mathbb{R}^9 \mid x_7 - x_8 \leq 0\}$

Based on the atomic propositions, one can obtain a partition of the state space. In this case, there are $2^3 = 8$ different regions obtained, each of which is a union of DBMs. Table 5-2 gives the number of DBMs obtained in each region together with the numbering of the DBMs. Each DBM is considered a state. The boolean vector beside each region is the vector l (cf. Section 4-2), which shows the contribution of each atomic proposition to the particular region. The order from left to right is: (DelayBounded, BacklogBounded, ConformantInput). For example, Region (1,0,1) is the region where DelayBounded is true, BacklogBounded is false, and ConformantInput is true.

Table 5-2: Regions of the state-space partition based on the atomic propositions (AP partition)

Region	$(0,0,0)$	$(0,0,1)$	$(0,1,0)$	$(0,1,1)$
Number of DBMs	2	2	1	1
Assigned States	$\{1,2\}$	$\{3,4\}$	$\{5\}$	$\{6\}$
Region	$(1,0,0)$	$(1,0,1)$	$(1,1,0)$	$(1,1,1)$
Number of DBMs	2	2	1	1
Assigned States	$\{7,8\}$	$\{9,10\}$	$\{11\}$	$\{12\}$

Step 3: Partition the state-space based on the PWA dynamics

After partitioning the state space based on the atomic propositions, the next step is to partition the state space based on the PWA dynamics. Since all the elements of the first row in A are infinite, it can be ignored for this procedure. The maximum number of PWA regions depend on the number of finite elements in each row-finite row. In this case there are two PWA regions because there are only two different possibilities for vector g (c.f Eq. (4-2)), namely $g^1 = (1,2,3,4,5,6,3,9)$ and $g^2 = (1,2,3,4,5,6,8,9)$. The two regions are:

$$R_g^1 = \left\{x \in \mathbb{R}^9 \mid x_3 - x_8 < 5\right\}$$
$$R_g^2 = \left\{x \in \mathbb{R}^9 \mid x_3 - x_8 \geq 5\right\}$$

Note that the regions shown after resolving the boundary issue, that is after applying Algorithm 2.

Step 4: Refine PWA partition by AP partition to obtain an output-preserving partition of the state space

Having the two partitions of the state space, the last step in forming the states of the quotient system is to apply the refinement procedure described by Algorithm 3. The total number of regions after refinement is $8 \times 2 = 16$. Since some regions in the AP partition are composed of unions of DBMs, the number of quotient states, with each state being a DBM, is $12 \times 2 = 24$. A summary of the partition is given in Table 5-3, where each region in the refined partition is marked by the contributing regions of the coarser partitions. For example, Region $\{R_g^2, (0,0,1)\}$ is the region obtained from the intersection of AP region $(0,0,1)$ and PWA region R_g^2.

Table 5-3: Regions of the output-preserving partition

Region	$\{R_g^1, (0,0,0)\}$	$\{R_g^1, (0,0,1)\}$	$\{R_g^1, (0,1,0))\}$	$\{R_g^1, (0,1,1)\}$
Number of DBMs	2	2	1	1
Assigned States	$\{1,2\}$	$\{3,4\}$	$\{5\}$	$\{6\}$
Region	$\{R_g^1, (1,0,0)\}$	$\{R_g^1, (1,0,1)\}$	$\{R_g^1, (1,1,0))\}$	$\{R_g^1, (1,1,1)\}$
Number of DBMs	2	2	1	1
Assigned States	$\{7,8\}$	$\{9,10\}$	$\{11\}$	$\{12\}$
Region	$\{R_g^2, (0,0,0)\}$	$\{R_g^2, (0,0,1)\}$	$\{R_g^2, (0,1,0)\}$	$\{R_g^2, (0,1,1)\}$
Number of DBMs	2	2	1	1
Assigned States	$\{13,14\}$	$\{15,16\}$	$\{17\}$	$\{18\}$
Region	$\{R_g^2, (1,0,0\}$	$\{R_g^2, (1,0,1)$	$\{R_g^2, (1,1,0)\}$	$\{R_g^2, (1,1,1)\}$
Number of DBMs	2	2	1	1
Assigned States	$\{19,20\}$	$\{21,22\}$	$\{23\}$	$\{24\}$

Step 5: Compute the transitions between the symbolic states

To compute the transitions, one must first specify a partition on the input space. In this case, there is a single input and thus the input space is $\mathbb{R}$. Because the input must always be nonnegative, the input space can be partitioned into two regions $U_1 = \{u \in \mathbb{R} \mid u \geq 0\}$ and $U_2 = \{u \in \mathbb{R} \mid u < 0\}$. Consequently, the transitions are formed based on the input U_1. After applying Algorithm 4, the full transition system is obtained. Because the transition graph is huge and will not be very useful if presented as it is, all the transitions to nonconformant states are removed for the graph to be presentable. Looking back at Table 5-3, the nonconformant states that are part of the regions marked as $\{R_g^1, (-,-,0)\}$ or $\{R_g^2, (-,-,0)\}$, where $-$ denotes a "don't care" value that can be replaced by either 0 or 1. Thus, the set of nonconformant states is $\{1,2,5,7,8,11,13,14,17,19,20,23\}$. An interested reader can consult Appendix A for the full transition system. The states that satisfy all the atomic propositions are $\{12,24\}$, and are marked by green color in Figure 5-2. Other conformant states are not colored. Notice the nondeterminism in the quotient system.

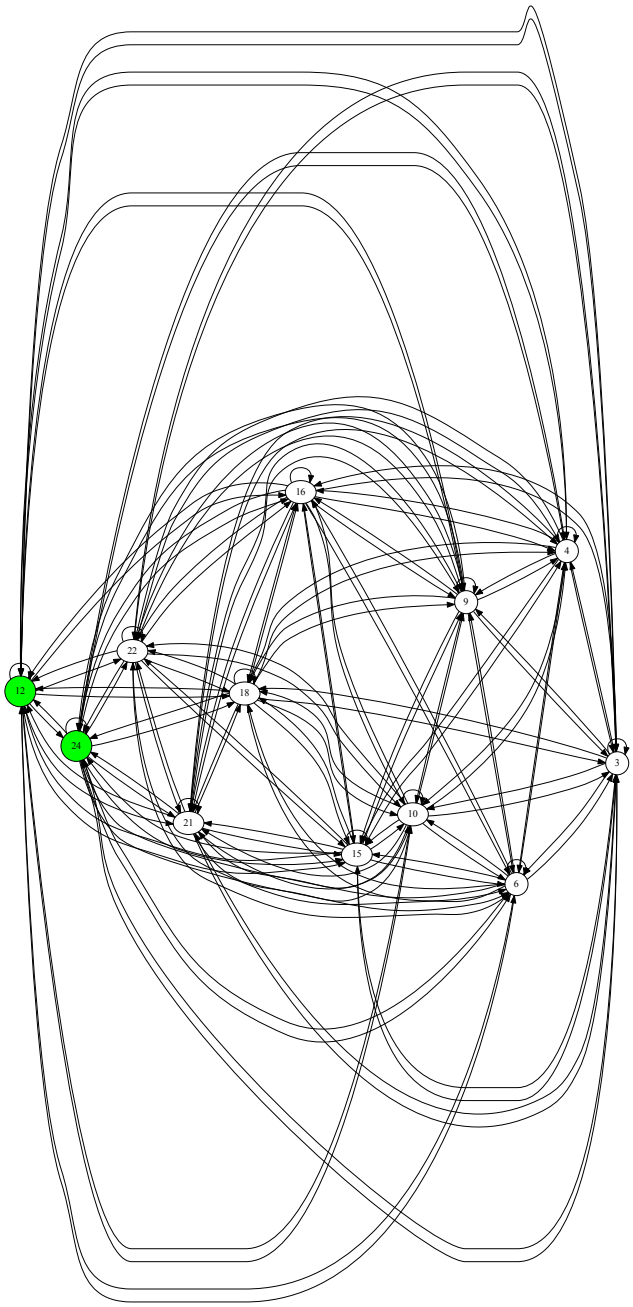

Figure 5-2: Transition graph of the quotient system after removing the nonconformant states

Step 6: Use model-checking to verify the properties

Since the original system is a deterministic dynamical system, the nondeterminism of the quotient system is a clear indication that it does not bisimulate the original system. The construction of the quotient system by the abstraction technique discussed in Chapter 4 only guarantees that the quotient system simulates the original system (cf. Theorem 2.4). In some cases, iteratively refining the state-space partition can lead to a bisimulating quotient system. However, this procedure is not guaranteed to terminate in general [43], as it might be the case that the infinite-state system cannot be bisimulated by a finite-state quotient system.

The resultant quotient system is exported as a Promela file to be used by the SPIN model checker [46] for the verification of the given LTL specifications. Unfortunately, the symbolic abstraction was too coarse to satisfy the LTL properties φ_1 and φ_2. To better understand why the properties were not satisfied, the transition graph given in Figure 5-2 can be consulted since it is small enough to be visually analyzed. Although the transition graph given is only part of the full transition system, it is still useful to deduce the counterexamples that violate each of the LTL formulas. For easier reference, the two LTL formulas are listed again below:

$$\varphi_1 = \Box\,(\mathsf{ConformantInput}) \implies \Box\,(\mathsf{BacklogBounded})$$
$$\varphi_2 = \Box\,(\mathsf{ConformantInput}) \implies \Box\,(\mathsf{DelayBounded})$$

In order to perform model checking, the initial state of the quotient system must be specified. Recall from Section 3-4 that the initial states of the original system are those that correspond to zero backlog and full leaky bucket. After performing the symbolic abstraction, these initial states are part of the symbolic state 12. One path that acts as a counterexample violating φ_1 is the infinite path $\mathsf{z}_1 = (12, 22, 22, 22, \dots)$. Obviously, z_1 satisfies $\Box\,(\mathsf{ConformantInput})$ since both states, 12 and 22, are conformant states. However, z_1 does not satisfy $\Box\,(\mathsf{BacklogBounded})$ because $\mathsf{BacklogBounded}$ is not satisfied in state 22. As a result, the implication in φ_1 is not respected by at least one path, z_1, in the quotient system, which renders the LTL property unsatisfied.

Another path that acts as a counterexample violating φ_2 is the infinite path $\mathsf{z}_2 = (12, 18, 18, 18, \dots)$. Similar to z_1, z_2 also satisfies $\Box\,(\mathsf{ConformantInput})$ since state 18 is a conformant state as well. At the same time, $\mathsf{DelayBounded}$ is not satisfied in state 18, which causes $\Box\,(\mathsf{DelayBounded})$ not to be satisfied by z_2. Hence, the implication in φ_2 is also not respected and the LTL property not satisfied.

Arriving to this result where both properties are not satisfied by the quotient system, no conclusion can be made about the original infinite-state system. At this stage, there are two options. The first option is to increase the parameters B_{max} and D_{max} and repeat the abstraction process again. The second option is to refine the state-space partition with hope of obtaining a bisimulating quotient system or at least a simulating system that verifies the properties. The procedure in the second option should be applied with care since it is not guaranteed to terminate as discussed earlier.

Chapter 6

Conclusion and Future Work

The event-triggered control paradigm has great potentials in modern networked control systems' applications. Nevertheless, challenges imposed by the network are far from being trivial. In this thesis, a finite abstraction modeling technique was proposed based on network calculus. Targeting worst-case analysis, network calculus offers a well-suited ground for developing discrete-time systems that model network elements. The state-space representation of such discrete-time models was derived obtaining MiPL systems. To construct finite abstractions of the MiPL systems, a useful technique, which was developed for the abstractions of max-plus linear systems, was applied with some modifications. The abstraction procedure was fully described in Chapter 4. Based on the intuitive meaning of the backlog and virtual delay, the properties of the network were formulated as LTL specifications. The abstraction procedures were applied to an example. Despite successfully obtaining a finite abstraction of the discrete-time system, the symbolic model included many more behaviors than the original infinite-state system causing the LTL properties not to be satisfied.

Reflecting back on the abstraction procedure, one can conclude that the main cause of the coarse abstraction is the way the procedure deals with the input. Because the input is cumulative, the input space is considered to be unbounded to account for any possible input to the system. As a result, even after partitioning the input space into a finite number of symbolic inputs, at least one of the symbolic inputs will be unbounded. In addition, the current abstraction procedure does not accommodate the fact that the input is a nondecreasing sequence. The abstraction procedure allows the input to change less restrictively than it can do in original model. As a result, all trajectories that are only possible when the input is allowed to decrease are possible in the resultant quotient system. This result clearly manifests itself in the quotient system by having transitions between nearly any two symbolic states, as shown in Figure 5-2 and Figure A-1.

One possible improvement is to consider the input increments instead of the input itself. The advantage of working with the increments is that they are essentially bounded because the arrivals during any bounded time interval are finite. Another advantage is that dealing with the nondecreasing nature of the input becomes more possible. For example, an internal variable can be introduced to continuously accumulate the increments and represent the

nondecreasing input. Showing great potentials, this idea is worth further investigation in future.

Finally, the future work for this thesis can be summarized by the following bullet points:

- Improve the abstraction techniques presented in this thesis to allow generating symbolic models that better capture the dynamics of the discrete-time dynamical systems. Considering the idea of input increments could offer valuable improvements.
- Improve the efficiency of the abstraction procedures by utilizing more efficient data structures, such as Binary Decision Diagram [47], to allow working with very large state spaces.
- Analyze more complicated arrival curves that can better model the data transmitted in networked control systems.
- Improve the server model to incorporate more sophisticated features that are useful for actual networked control system.
- Extend the analysis to min-plus nonlinear compositions of systems that can arise in real networks. Nonlinear compositions place some challenges in network calculus. Finding an efficient technique to compose the symbolic models nonlinearly can open the door for more complicated types of analysis for networked control systems.

Appendix A

The full state transition graph of the quotient system obtained from the example

Figure A-1 shows the full state transition graph of the quotient system presented in the example of Chapter 5. Note that nonconformant states are colored with red, states where all the atomic propositions are satisfied are colored with green, and the other conformant states are kept in white.

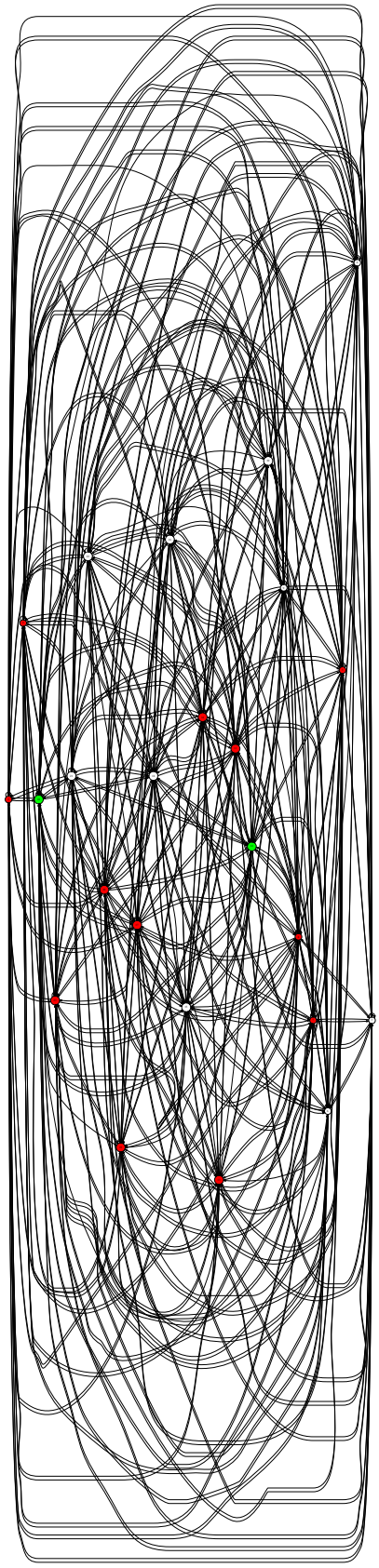

Figure A-1: Transition graph of the quotient system obtained in the example

Appendix B

MATLAB Files

All the MATLAB files that were developed to implement the abstraction techniques presented in this thesis are given below. It should be mentioned that the implementation is partially based on the implementation of the VeriSiMPL toolbox [48].

B-1 example.m

```
% This script can reproduce the results of the example described in
% Chapter 5 of the thesis
%% Step 1: Define the state-space representation of the system
clear, clc;
d = 3;
c = 5;
r = 4;
b = 2;

B_max = r*d+b; % The maximum from NC
D_max = d+c/b; % The maximum from NC

B_margin = 0;
B_max = B_max + B_margin;  % Maximum Baklog to be tested

D_margin = 0;
D_max = ceil(D_max + D_margin); % Maximum delay to be tested

[A,B,n] = generate_AB([d,c],[r,b],D_max);

%% Step 2: Define the DBMs of the atomic propositions and partition the
    state space accordingly

% Create a DBM for the region where x(n)-x(1)>=0 is true
```

```
24 % This is the region where the input flow conforms with the arrival
       policy
25 AP_1 = R(n); % Initialize DBM
26
27 % x(n)-x(1)>=0  ==> x(1)-x(n)<=0 ==> D(n+1,2) = 0
28 i = (1)+1;
29 j = (n)+1;
30 AP_1{1}(j,i) = 0;
31 AP_1{2}(j,i) = 1;
32
33 % Create a DBM for the region where u(k)-y(k)<=B_max (x(1)-x(n-1)<=B_max)
34 % is true
35 % This is the region where the backlog is bounded.
36 AP_2 = R(n); % Initialize DBM
37
38 i = (1)+1;
39 j = (n-1)+1;
40
41 % Upper bound:
42 AP_2{1}(j,i) = B_max;
43 AP_2{2}(j,i) = 1;
44
45 % Lower bound:
46 AP_2{1}(i,j) = 0;
47 AP_2{2}(i,j) = 1;
48
49 % Virtual Delay Bound AP
50 % Create a DBM for the region u(k-D_max)-y(k)<=0 ( x(D_max+1)-x(n-1)<=0)
51 % This is the region where the backlog is within the limit.
52 AP_3 = R(n); % Initialize DBM
53
54 % x(D_max+1)-x(n-1)<=0
55 %D_max = 1;
56 i = (D_max+1)+1;
57 j = (n-1)+1;
58
59 % Upper bound:
60 AP_3{1}(j,i) = 0;
61 AP_3{2}(j,i) = 1;
62
63 % Combine the three DBMs into one DBM representing the three APs
64 AP = cell(1,2);
65 AP{1}(:,:,1) = AP_1{1};
66 AP{1}(:,:,2) = AP_2{1};
67 AP{1}(:,:,3) = AP_3{1};
68
69 AP{2}(:,:,1) = AP_1{2};
70 AP{2}(:,:,2) = AP_2{2};
71 AP{2}(:,:,3) = AP_3{2};
72
73 % Partition the state space into a set of DBMs based on the atomic
74 % propositions
75 AP_regions = dbm_sspart(AP);
```

```
76
77 AP_Partition = label_AP_partition(AP_regions);
78 %% Step 3: Partition the state-space based on the PWA dynamics
79
80 U = R(1);           % DBM describing the set of possible inputs
81
82 % The input U(k) is a counter and therefore cannot be negative. So we
       need
83 % to restrict the DBM of U to the non-negative region.
84 U{1}(2,1) = 0;
85 U{2}(2,1) = 1;
86
87 % Generate the PWA system of the autonomous min-plus system, i.e no input
88 [A_pwa,B_pwa,D_pwa] = minpl2pwa(A(2:end,:));
89
90 % Refine the cover to form a partition of the state-space
91 [A_pwa,B_pwa,D_pwa] = minpl2pwa_refine(A(2:end,:),A_pwa,B_pwa,D_pwa);
92
93
94 %% Step 4: Refine PWA partition by AP partition to obtain an output-
       preserving partition of the state space
95
96 % Generate a refined partition based on the PWA and AP partitions
97 Refined_part = AP_PWA_part_refine(D_pwa,AP_Partition);
98
99 % Find the number of symbolic states in the refined partition
100 num_states = size(Refined_part{1},3);
101 %% Step 5: Compute the transitions between the symbolic states
102
103 % Form the Transition System of the nonautonomous min-plus system
104 % Generate the PWA system of the augmented min-plus system
105 [Anon,Bnon,Dnon] = minpl2pwa([A B]);
106
107 % Create the transitions between the states according to the dynamics of
108 % the  PWA systems.
109 Refined_part_unlabelled = {Refined_part{1},Refined_part{2}};
110 adj = minplnon2ts_trans(Refined_part_unlabelled,Anon,Bnon,Dnon,U);
111
112 All= -1; % Constant used for clarity indicating all the possible values
113 BoundedBacklogDelay_set = prop2state([ 1  , 1  ,1], Refined_part);
114 NonConformant_set       = prop2state([All,All,0], Refined_part);
115 Conformant_set          = prop2state([All,All,1], Refined_part);
116
117 adj_safe_only = remove_set(adj,NonConformant_set); % Remove the unsafe
       set
118
119 % Create a graphical representation of the transition system in graphviz
120 % code
121 ts2graphviz_color(adj_safe_only,BoundedBacklogDelay_set, ...
122                   NonConformant_set,'ts','mygraph');
123 %% Step 6 : Generate the Promela file that can be used to verify the
       properties using SPIN
124
```

```
125 initial_set = prop2state([1,1,1], Refined_part);
126
127 % Mark the initial set by adding a new initial node with a transition to
128 % each state in the initial set
129 adj_new = mark_initial_set(adj,initial_set);
130
131 s0_dbm = R(n);
132 s0_label = '111';
133 Refined_part{1}(:,:,num_states+1) = s0_dbm{1};
134 Refined_part{2}(:,:,num_states+1) = s0_dbm{2};
135 Refined_part{3}{num_states+1} = s0_label;
136
137 s0 = size(adj_new,1); % Initial state is the last one added
138
139 % Generate the Promela file for transition system that can be fed to SPIN
140 mipl_ts2promela(Refined_part,adj_new ,'ts',s0);
```

B-2 generate_AB.m

```
1  % Description: This function is used to generate the state-space
2  % representaion of the system based on the Latency-rate and leaky bucket
3  % parameters
4
5  function [A,B,n] = generate_AB(LR_param,LB_param,tau)
6
7  d = LR_param(1);
8  c = LR_param(2);
9  r = LB_param(1);
10 b = LB_param(2);
11
12 mem_states = tau+1; % Number of memory states needed to store the input
13 LB_states = 1; % Number of states for leaky bucket dynamics
14 GR_states = 1; % Number of states for guaranteed-rate server dynamics
15 n = mem_states + GR_states + LB_states; % Dimension of State Space
16
17
18 %% Create A matrix
19 A = Inf(n,n);
20
21 % Assign the values related to the memory states
22 for k = 2:mem_states
23     A(k,k-1) = 0;
24 end
25
26 % Assign the values related to the guaranteed-rate server dynamics
27 %w(k) = min[ x_d(k-1), c + w(k-1)]
28 %x_{N-1} <-- w
29 A(n-1,d) = 0;
30 A(n-1,n-1) = c;
31
32 % Assign the values related to the leaky bucket dynamics
33 A(n,n) = r;
```

```
34
35 %% Create B matrix
36 B = Inf(n,1);
37 B(1) = 0;
38 B(n) = b;
39
40 end
```

B-3 dbm_relax.m

```
1  % Description: This function removes all the redundant elements from a
       DBM.
2  % The function keeps only the elements that are needed to describe the
3  % region covered by the DBM correctly.
4
5  function D_relaxed = dbm_relax(D)
6
7  n = size(D{1},1);   % size of the DBM
8  D = floyd_warshall(D);  % Get the canonical form of D, which is unique
9
10 D_new = D;  %initialize D_new
11
12 i = 1;  % Initialize row counter 'i'
13 while (i <= n)    % Loop over the rows of the DBM
14
15     j = 1;  % Initialize column counter 'j'
16
17     while (j <= n)   % Loop over the columns of the DBM
18
19         % Enter if the current element is  bound is finite
20         if (j ~= i && D{1}(i,j)<Inf)
21             % Relax the current bound
22             D_new{1}(i,j) = Inf;
23             D_new{2}(i,j) = 0;
24
25             % Calculate the canonical form of the relaxed DBM
26             C = floyd_warshall(D_new);
27
28             %Check if the relaxed DBM still represents the same region
29             if(~isequal(C,D))
30                 % Restore the previous values of the relaxed bound
31                 D_new{1}(i,j) = D{1}(i,j);
32                 D_new{2}(i,j) = D{2}(i,j);
33             end
34         end
35
36         j = j+1;    % Update column counter
37
38     end
39
40     i = i+1;    % Update row counter
41
```

```
42 end
43
44 D_relaxed = D_new;
45
46 end
```

B-4 dbm_complement.m

```
1  % Description: This function generates the complement of a DBM. The
2  % complement of a DBM is in general a region composed of finite union of
3  % DBMs
4
5  function D_comp = dbm_complement(D)
6  n = size(D{1},1)-1; % Get the dimension of the state space
7
8  % Get the equivalent DBM with the least number bounds possible
9  D = dbm_relax(D);
10
11 num_inf = (D{1} < Inf);
12 num_dbm = sum(sum(num_inf))-(n+1) ; % Number of finite off-diagonal
       elements
13
14 % Define the DBM of the entire state space R^n
15 R_n= cell(1,2);
16 R_n{1} = Inf(n+1,n+1);
17 for i = 1:n+1
18     R_n{1}(i,i) = 0;
19 end
20 R_n{2} = eye(n+1,n+1);
21
22 % Prepare the DBMs of the complement and initialize each one to R^n
23 D_comp = dbm_repmat(R_n,num_dbm);
24
25 A = R_n;  % Initiallize A to R_n. In each round, 'A' will be filled by a
26             % new element from 'D'.
27 i = 1;      % Counter over the rows of 'D'
28 k = 1;      % Counter over the DBMs that form the complement of 'D'
29
30 while ((i<=(n+1)) && (k <=num_dbm))
31     j = 1;  % Counter over the columns of 'D'
32     while (j<=(n+1))
33         if (j ~= i) && (D{1}(i,j) < Inf)
34             % Start by storing the complement of the half-space defined
35             % the current upper bound value D{1}(i,j)
36             D_comp{1}(j,i,k) = -D{1}(i,j);
37             D_comp{2}(j,i,k) = ~D{2}(i,j);
38
39             % No need to find the intersection for k = 1 since A = R_n.
40             if(k>1)
41                 %Store the current complement DBM in 'C'
42                 C = {D_comp{1}(:,:,k),D_comp{2}(:,:,k)};
43
```

```
44                  % Find the intersection between C and A
45                  C = dbm_and(C,A);
46
47                  %Store the intersection back in the current complement
                        DBM
48                  D_comp{1}(:,:,k) = C{1};
49                  D_comp{2}(:,:,k) = C{2};
50              end
51
52              %Store the D(i,j) in A(i,j) for the next round
53              A{1}(i,j) = D{1}(i,j);
54              A{2}(i,j) = D{2}(i,j);
55
56              k = k+1;    % Go to next complement DBM to be filled
57          end
58          j = j+1;  % Go to next column of 'D'
59      end
60      i = i+1;    % Go to next row of 'D'
61  end
62
63  end
```

B-5 union_dbm_and.m

```
1  % Description: This function takes two finite union of DBMs and finds the
2  % intersection between them, which is also a finite union of DMBs.
3  % This function can be seen as the generalization of "dbm_and" function,
4  % which finds the intersection between single DBMs.
5
6  function P = union_dbm_and(P1,P2)
7  n =  size(P1{1},1)-1;   % Find out the dimension of the state space
8  M1 = size(P1{1},3); % Number of DBMs in region P1
9  M2 = size(P2{1},3); % Number of DBMs in region P2
10
11 % Define the DBM of the entire state space R^n
12 R_n= cell(1,2);
13 R_n{1} = Inf(n+1,n+1);
14 for i = 1:n+1
15     R_n{1}(i,i) = 0;
16 end
17 R_n{2} = eye(n+1,n+1);
18
19 M = M1*M2; %Maximum number of DBMs that can make up the intersection
       region
20
21 % Prepare the DBMs of the intersection and initialize each one to R^n
22 P = dbm_repmat(R_n,M);
23
24 i = 1;  % Counter over the DBMs of region P1
25 k = 1;  % Counter over the DBMs of the intersection region P
26 while (i<=M1)
27     % Store the ith DBM of region P1 in DBM_i
```

```
28      DBM_i = {P1{1}(:,:,i),P1{2}(:,:,i)};
29
30      j = 1;  % Counter over the DBMs of region P2
31      while (j<=M2)
32          % Store the ith DBM of region P2 in DBM_j
33          DBM_j = {P2{1}(:,:,j),P2{2}(:,:,j)};
34
35          % Find the intersection between DBM_i and DBM_j
36          C = dbm_and(DBM_i,DBM_j);
37
38          % Store the intersection
39          P{1}(:,:,k) = C{1};
40          P{2}(:,:,k) = C{2};
41
42          k = k+1;    % Go to next element of region P
43          j = j+1;  % Go to next element of region P2
44      end
45      i = i+1;    % Go to next element of region P1
46  end
47
48  % Remove the DBMs in P that are subsets of another DBM.
49  P = dbm_delsubset(P);
50  end
```

B-6 dbm_sspart.m

```
1   % Description : This function creates a state-space partition based on a
2   % set of atomic propositions that are passed as DBMs
3
4   function P = dbm_sspart(D)
5   num_dbm = size(D{1},3);
6
7   D_comp = cell(1,num_dbm);   % Cell array to store the complement DBMs
8   P = cell(1,num_dbm); % Cell array to store the DBMs and their complements
9   k = 1;
10  while (k <= num_dbm)
11      D_comp{k} = dbm_complement({D{1}(:,:,k),D{2}(:,:,k)});
12      P{k} = {D_comp{k},{D{1}(:,:,k),D{2}(:,:,k)}};
13      k = k+1;
14  end
15
16  k = 0;
17  N = num_dbm;
18  while (N >= 2)
19      isOdd = mod(N,2);
20      N = ceil(N/2);
21      i = 1; % Counter over the elements in the P
22      while(i <= N-1)
23          % Form the intersections between the two "property" sets
24          P{i} = property_combine(P{i},P{i+1});
25          P(i+1) = [];  % Remove cell from cell array
26          k = k+1;
```

```
27          i = i+1;
28      end
29      if(isOdd == 0)
30           % Form the intersections between the two "property" sets
31          P{i} = property_combine(P{i},P{i+1});
32          P(i+1) = [];  % Remove cell from cell array
33          k = k+1;
34      end
35  end
36
37  % After merging all the elements in P, P will have only 1 element which
        is
38  % a cell array of 2^num_dbm elements. In other words, result we need is
        in
39  % P{1}.
40  P = P{1};  % Adjust P so that the needed DBMs are directly accesible
41  end
```

B-7 property_combine.m

```
1   % Description: This function combines two atomic properties in one
2   % partition
3
4   function Poperties = property_combine(P1,P2)
5   % Each input is composed of a set of "property regions". A property
        region
6   % is given by a set of DBMs that span the set of states in which a
        certain
7   % property holds.
8   num_regions_1 = length(P1); % number of different "property regions" in
        P1
9   num_regions_2 = length(P2); % number of different "property regions" in
        P2
10  Poperties = cell(1,num_regions_1*num_regions_2);
11  i = 1;
12  while(i <=num_regions_2)
13      j = 1;
14      while(j <= num_regions_1)
15          index = (i-1)*num_regions_1+j;
16
17          % Find the intersection between the two finite unions of DBMs
18          Poperties{index} = union_dbm_and(P2{i},P1{j});
19          j = j+1;
20      end
21      i = i+1;
22  end
23
24  end
```

B-8 label_AP_partition.m

```
% Description: This function adds labels to the AP partition. The label
% is a binary string representing the contribution of each atomic
% proposition. For example, string '011' representes a DBM in which AP3 =
% false, AP2 = true and AP1 = true.

function P_labeled = label_AP_partition(P)

num_regions = length(P);    % Find the total number of regions (=2^|AP|)
n_bits = log2(num_regions); % Find the length of the labeling string
P_labeled = cell(1,3);

k_end = 0; %initialize
for k = 1:num_regions
    num_dbms = size(P{k}{1},3);
    k_begin = k_end+1;
    k_end = k_begin + num_dbms-1;
    P_labeled{1}(:,:,k_begin:k_end) = P{k}{1};
    P_labeled{2}(:,:,k_begin:k_end) = P{k}{2};

    for i = k_begin:k_end
        P_labeled{3}{i} = dec2bin(k-1,n_bits); % Add the binary label
    end
end

end
```

B-9 minpl2pwa.m

```
% MINPL2PWA Generating a piecewise-affine system of the MiPL system
% Adapted from maxpl2pwa(Ampl) by Dieky Adzkiya

function [A,B,D] = minpl2pwa(Ampl)

% in general Ampl is not square, it is m by n matrix
[m n] = size(Ampl);

% F, find the location of finite values in each row of Ampl
% idxmax, keep the number of finite values in each row of Ampl
% idxmax will be used to calculate the maximum number of regions
idxmax = ones(1,m);
F = cell(1,m);
for i = 1:m
    F{i} = find(Ampl(i,:) ~= Inf); % >>> Original: F{i} = find(Ampl(i,:)
        ~= -Inf)
    idxmax(i) = length(F{i});
end

% N, maximum number of regions
N = prod(idxmax);

% initialize the dynamical systems
A = zeros(m,N);
```

```
24 B = zeros(m,N);
25
26 % initialize regions (DBM)
27 D = cell(1,2);
28 D{1} = zeros(n+1,n+1,N);
29 D{2} = false(n+1,n+1,N);
30
31 % Nr, the number of inserted (nonempty) regions
32 Nr = 0;
33
34 % number of elements of idx is the same with rows of A
35 % position/column of finite element at each row w.r.t. F (not for A)
36 idx = ones(1,m);
37
38 % level, this variable is used in the pruning technique
39 lvl = 1;
40
41 % some temporary regions used to build the tree in the pruning technique
42 Di = cell(m,2);
43
44 while lvl > 0
45     % while there exists an unobserved node in the tree
46
47     % constructing a region and save it to the temporary region
48     Di(lvl,:) = row2dbm_pwa_min(Ampl(lvl,:),idx(lvl),F{lvl});
49     if lvl > 1
50         % do not forget to intersect with the upper region
51         Di(lvl,:) = dbm_and(Di(lvl,:),Di(lvl-1,:));
52     end
53
54     % check emptiness of the new region, indexed by i-lvl
55     [empty,Di(lvl,:)] = dbm_isempty(Di(lvl,:));
56     % the new region is in the canonical-form representation
57
58     if empty
59         % the new region is empty, we go to the right
60         [idx,lvl] = next_comb(idx,idxmax,lvl);
61     else
62         % the new region is not empty, we go down
63
64         % check whether the new region has a child
65         if lvl == m
66             % the new region does not have a child
67             % we are in the leaf/bottom
68             % save the region and its dynamical system
69
70             % update the number of inserted (nonempty) regions
71             Nr = Nr + 1;
72
73             % insert the new (nonempty) region
74             D{1}(:,:,Nr) = Di{lvl,1};
75             D{2}(:,:,Nr) = Di{lvl,2};
76
```

```
77              % constructing the dynamical system from idx
78              for i = 1:m
79                  % construct A and B
80                  A(i,Nr) = F{i}(idx(i));
81                  B(i,Nr) = Ampl(i,A(i,Nr));
82              end
83
84              % since we are in the bottom and the region is not empty, we
                    go
85              % to the right
86              [idx,lvl] = next_comb(idx,idxmax,lvl);
87          else
88              % since we are not in the bottom, we go down
89              lvl = lvl + 1;
90          end
91      end
92  end
93
94  % delete the unused spaces
95  A(:,Nr + 1:end) = [];
96  B(:,Nr + 1:end) = [];
97  D{1}(:,:,Nr + 1:end) = [];
98  D{2}(:,:,Nr + 1:end) = [];
99
100 end
```

B-10 row2dbm_pwa_min.m

```
1  % ROW2DBM_PWA_MIN Constructing a difference-bound matrix from a row
2  % vector.
3  % Adapted from row2dbm_pwa() by Dieky Adzkiya
4
5  function D = row2dbm_pwa_min(A,idx,F)
6
7  % n, the dimension of the state space
8  n = length(A);
9
10 % initialize the output with the state space
11 % the size is n+1 because of the zero variable
12 D = cell(1,2);
13 D{1} = Inf(n+1);
14 D{2} = false(n+1);
15
16 % D{1}(i,i) = 0, for 1 <= i <= n+1
17 % the main diagonal of D{1} is initialized to zero
18 D{1}(1:n+2:(n+1)*(n+1)) = 0;
19 % D{2}(i,i) = 0, for 1 <= i <= n+1
20 % the main diagonal of D{2} is initialized to one
21 D{2}(1:n+2:(n+1)*(n+1)) = 1;
22
23 % check the number of elements in F
24 if length(F) == 1
```

```
25      % if F contains one element, the result is the state space
26
27      % avoid D{1} and D{2} defined as an empty matrix
28      return;
29  end
30
31  % j, indices of finite elements other than idx
32  % j is not empty, because length(F) > 1
33  j = F(F ~= F(idx));
34  D{1}(F(idx)*(n + 1) + j + 1) = A(j) - A(F(idx));
35  D{2}(F(idx)*(n + 1) + j + 1) = 1;
36
37  end
```

B-11 minpl2pwa_refine.m

```
1   % MINPL2PWA_REFINE Refining the regions
2   % Adapted from maxpl2pwa_refine() by Dieky Adzkiya
3
4   function [A,B,D] = minpl2pwa_refine(Ampl,A,B,D)
5
6   for i = 1:size(A,1)
7
8       % idx_fi is the position of finite elements in i-th row of Ampl
9       idx_fi = find(Ampl(i,:) < Inf);
10
11      for j = 1:size(A,2)
12          % g = A(:,j) and gi = A(i,j)
13
14          % idx is the position of finite elements, on the right of gi
15          %idx = find(idx_fi > A(i,j));
16          idx = idx_fi(idx_fi > A(i,j));
17
18          % check whether there are some finite elements
19          if isempty(idx)
20              continue;
21          end
22
23          % find "bigger" adjacent region(s)
24          % the previous data stored on idx can be overwritten
25          idx = intersect([repmat(A(1:i-1,j),1,length(idx)); ...
26              idx; ...
27              repmat(A(i+1:end,j),1,length(idx))]', ...
28              A','rows')';
29
30          % check the existence of bigger adjacent region(s)
31          if isempty(idx)
32              continue;
33          end
34
35          % compute the set difference
36          % be careful, idx is a matrix
```

```
37          Dj = row2dbm_pwa_min(Ampl(i,:), ...
38              sum(idx_fi <= A(i,j)), ...
39              idx_fi);   % >>> Original : row2dbm_pwa(...)
40          Dj{2}(idx(i,:)+1,A(i,j)+1) = 0;
41
42          % the result is saved on Dj temporarily
43          Dj = dbm_and({D{1}(:,:,j),D{2}(:,:,j)},Dj);
44          % finally save it on D
45          D{1}(:,:,j) = Dj{1};
46          D{2}(:,:,j) = Dj{2};
47      end
48  end
49
50  % remove empty regions and the corresponding affine dynamics, if it
        exists
51  % idx = 1, if the DBM is empty
52  [idx,D] = dbm_isempty(D);
53  D{1}(:,:,idx == 1) = [];
54  D{2}(:,:,idx == 1) = [];
55  A(:,idx == 1) = [];
56  B(:,idx == 1) = [];
57
58  end
```

B-12 AP_PWA_part_refine.m

```
1  % Description: This function takes two partitions one based on the PWA
2  % and another based on the atomic propositions. It computes the refined
3  % partition. The resultant refined partition is labelled based on the
4  % atomic propositions.
5
6  function Refined_part = AP_PWA_part_refine(PWA_part,AP_part)
7
8  num_PWA_DBMs = size(PWA_part{1},3); % number of DBMs in PWA_part
9  num_AP_DBMs = size(AP_part{1},3); % number of DBMs in AP_part
10
11 % Define a cell array that will store the DBMs of the refined partition
12 Refined_part = cell(1,3) ;
13
14 index = 0;
15 for i = 1:num_PWA_DBMs
16     DBM_1 = {PWA_part{1}(:,:,i),PWA_part{2}(:,:,i)};
17     for j = 1:num_AP_DBMs
18         DBM_2 = {AP_part{1}(:,:,j),AP_part{2}(:,:,j)};
19
20         % Find the intersection between the two DBMs
21         DBM_intersection =  dbm_and(DBM_1,DBM_2);
22
23         %Store the intersection if it is not empty
24         if (~dbm_isempty(DBM_intersection))
25
26             % index = (i-1)*num_AP_DBMs+j;
```

```
27              index = index+1;
28              Refined_part{1}(:,:,index) = DBM_intersection{1};
29              Refined_part{2}(:,:,index) = DBM_intersection{2};
30
31              % Store the AP label of the intersection
32              Refined_part{3}{index} = AP_part{3}{j};
33          end
34      end
35  end
36
37  end
```

B-13 minplnon2ts_trans.m

```
1   % MINPLNON2TS_TRANS Constructing the transitions
2   % Adapted from maxplnon2ts_trans() by Dieky Adzkiya
3
4   function adj = minplnon2ts_trans(D,Anon,Bnon,Dnon,U)
5
6   % nA, the number of regions generated by the autonomous MPL model
7   nA = size(D{1},3);
8
9   % nAB, the number of regions generated by the nonautonomous MPL model
10  nAB = size(Dnon{1},3);
11
12  % initialize the output variable adj
13  adj = sparse(nA,nA);
14
15  parfor i = 1:nA
16
17      % 1. compute the cross product of X and U, denoted by XU
18      % the result is a DBM in the augmented space
19      % 2. intersect XU and each region in the augmented space
20      % 3. check whether the intersections are empty
21      [empty,Ej] = dbm_isempty(dbm_and(dbm_repmat(xu_and( ...
22          {D{1}(:,:,i),D{2}(:,:,i)},U),nAB),Dnon));
23
24      % eliminate empty regions
25      Ej{1}(:,:,empty == 1) = [];
26      Ej{2}(:,:,empty == 1) = [];
27
28      % compute the image w.r.t. the augmented dynamics of the intersection
29      % the images are (nonempty) DBM in the state space
30      Ej = dbm_image(Anon(:,~empty),Bnon(:,~empty),Ej);
31
32      % find partitioning regions that intersect the images
33      for j = 1:size(Ej{1},3)
34          % for each (nonempty) image
35
36          % 1. intersect the image with each partitioning region
37          % 2. check whether the intersection is empty
38          % the negation, 1 = region is empty -> 0 = no relations
```

```
39          % the transpose, the output of dbm_isempty_vect is a row vector
                but
40          % we need a column vector
41          adj(:,i) = max(adj(:,i),~dbm_isempty(dbm_and(dbm_repmat( ...
42              {Ej{1}(:,:,j),Ej{2}(:,:,j)},nA),D))');
43      end
44  end
```

B-14 minplnon2ts_part.m

```
1  % MINPLNON2TS_PART Partitioning the state space and constructing a
2  % piecewise-affine system over the augmented space
3
4  % Adapted from maxplnon2ts_part() by Dieky Adzkiya
5
6  function [D,Anon,Bnon,Dnon] = minplnon2ts_part(Ampl,Bmpl)
7
8  % generate a partition of the state space using the MPL approach based on
9  % the autonomous MPL model
10 D = minpl2pwa_noab(Ampl);
11
12 % generate a PWA system based on the nonautonomous MPL model
13 [Anon,Bnon,Dnon] = minpl2pwa([Ampl Bmpl]);
14
15 end
```

B-15 minpl2pwa_noab.m

```
1  % MINPL2PWA_NOAB Generating the regions of a PWA system
2  % Adapted from maxpl2pwa_noab() by Dieky Adzkiya
3
4  function D = minpl2pwa_noab(Ampl)
5
6  % in general Ampl is not a square matrix, it is m by n matrix
7  [m n] = size(Ampl);
8
9  % F, find the location of finite values in each row of Ampl
10 % idxmax, keep the number of finite values in each row of Ampl (temporary
       )
11 % idxmax will be used to calculate the maximum number of regions
12 idxmax = ones(1,m);
13 F = cell(1,m);
14 for i = 1:m
15     F{i} = find(Ampl(i,:) ~= Inf);
16     idxmax(i) = length(F{i});
17 end
18
19 % update idxmax, based on the maximum number of regions generated by the
20 % MPL approach
21 idxmax = 2.^idxmax - 1;
22
```

```
% N, maximum number of regions
N = prod(idxmax);

% initialize regions and its relation to the dynamical system
D = cell(1,2);
D{1} = zeros(n+1,n+1,N);
D{2} = false(n+1,n+1,N);

% Nr, the number of inserted (nonempty) regions
Nr = 0;

% number of elements of idx is the same with rows of A
% position/column of finite element at each row w.r.t. F (not for A)
idx = ones(1,m);

% level, this variable is used in the pruning technique
lvl = 1;

% some temporary regions used to build the tree in the pruning technique
Di = cell(m,2);

while lvl > 0
    % while there exists an unobserved node in the tree

    % constructing a region and save it to the temporary region
    Di(lvl,:) = row2dbm_min(Ampl(lvl,:),idx(lvl),F{lvl});
    if lvl > 1
        % do not forget to intersect with the upper region
        Di(lvl,:) = dbm_and(Di(lvl,:),Di(lvl-1,:));
    end

    % check emptiness of the new region, indexed by i-lvl
    [empty,Di(lvl,:)] = dbm_isempty(Di(lvl,:));
    % the new region is in the canonical-form representation

    if empty
        % the new region is empty, we go to the right
        [idx,lvl] = next_comb(idx,idxmax,lvl);
    else
        % the new region is not empty, we go down

        % check whether the new region has a child
        if lvl == m
            % the new region does not have a child
            % we are in the leaf/bottom
            % save the region and its dynamical system

            % update the number of inserted (nonempty) regions
            Nr = Nr + 1;

            % insert the new (nonempty) region
            D{1}(:,:,Nr) = Di{lvl,1};
            D{2}(:,:,Nr) = Di{lvl,2};
```

```
76
77              % since we are in the bottom and the region is not empty, we
                    go
78              % to the right
79              [idx,lvl] = next_comb(idx,idxmax,lvl);
80          else
81              % since we are not in the bottom, we go down
82              lvl = lvl + 1;
83          end
84      end
85  end
86
87  % delete the unused spaces
88  D{1}(:,:,Nr + 1:end) = [];
89  D{2}(:,:,Nr + 1:end) = [];
90
91  end
```

B-16 prop2state.m

```
1   % Description: This function returns all the symbolic states that possess
2   % the property needed. The property is passed as an array with each
3   % element corresponding to one of the atomic propositions. The elements
4   % of the array are 1 whenever the corresponding atomic proposition is
5   % true, or 0 whenever the corresponding atomic proposition is false. An
6   % elemente with -1 marks a don't care value.
7
8   function states = prop2state(Property, P_labeled)
9
10  num_dbms = size(P_labeled{1},3);
11  num_AP = length(Property);
12  label_list = cell(1,1);
13
14  % Loop over the property input to see which lables are involved
15  for i = 1:num_AP
16      val = Property(i);
17      if(val  == 0)
18          % append zero to all elements in the label_list
19          label_list =  strcat(label_list,'0');
20
21      elseif (val == 1)
22          % append ones to all elements in the label_list
23          label_list =  strcat(label_list,'1');
24      elseif (val == -1)
25          % Create a duplicate of the label_list
26          duplicate_list = label_list;
27          % Append '0' to all elements in the original label_list
28          label_list =  strcat(label_list,'0');
29          % Append '1' to all elements in the duplicate label_list
30          duplicate_list =  strcat(duplicate_list,'1');
31          % Merge the two lists
32          label_list = [label_list,duplicate_list];
```

```
33      end
34  end
35
36  % Loop over all the states to find which ones have labels that are
        included
37  % in the label_list.
38  states = [];
39  for k = 1:num_dbms
40      state_label = P_labeled{3}{k};
41
42      if(any(ismember(label_list,state_label)))
43          states = [states,k];
44      end
45  end
46
47  end
```

B-17 R.m

```
1   % This function returns a DBM representing a region that covers the
2   % entire state space R^n.
3
4   function D = R(n)
5   D = cell(1,2);
6   D{1} = Inf(n+1,n+1);
7   for i = 1:n+1
8       D{1}(i,i) = 0;
9   end
10  D{2} = eye(n+1,n+1);
11
12  end
```

B-18 remove_set.m

```
1   % Description: This function removes a set of states from the transition
2   % system, by deleting all the transitions to and from the states.
3
4   function adj = remove_set(adj, set)
5
6   for i = 1:length(set)
7       state = set(i);
8       adj(:,state) = 0;
9       adj(state,:) = 0;
10  end
11
12  end
```

B-19 ts2graphviz_color.m

```
% TS2GRAPHVIZ_COLOR Generating a graphviz code with colored states
% Adapted from ts2graphviz() by Dieky Adzkiya

function ts2graphviz_color(adj,favored_set,unsafe_set,filename,gname)

% add dot extension to filename
filename = sprintf('%s.dot',filename);

% open the given filename
fid = fopen(filename,'w');

% write the declaration of digraph
fprintf(fid,'digraph %s {\n',gname);
fprintf(fid,'\t rankdir="LR" \n\t node [ width=0.02, height=0.02];\n');

% the main part
% r = row, c = column
[r,c] = find(adj ~= 0);

% h = hist([r;c],max([r;c]));

for state = 1:size(adj,1)
    if(sum(find(state==favored_set)) ~=0)
        fprintf(fid,'\t %d[style=filled, fillcolor=green ];\n', state);
    elseif(sum(find(state==unsafe_set)) ~=0)
        fprintf(fid,'\t %d[style=filled, fillcolor=red ];\n', state);
    else
        fprintf(fid,'\t %d[style=filled, fillcolor=white];\n', state);
    end

end

% create an unlabeled transition system
fprintf(fid,'\t %d -> %d [ len=2 ];\n',[c';r']);

% write the closing bracket of digraph
fprintf(fid,'}');

% close the file
fclose(fid);
```

B-20 mark_initial_set.m

```
% Description: This function adds a new hypothetical vertex to the
% transition system and adds transitions from this vertex to all the
% vertices in the set of initial states.


function adj_new = mark_initial_set(adj_old,initial_set)

num_init = length(initial_set);   %Get the number of initial states

```

```
10 [dst,src,w] = find(adj_old); % Get the information about the transitions
11
12 [m,n] = size(adj_old);
13
14 % % Shift the vertex numbers by one to make a room for the additional
15 % % initial vertex
16 % dst_new = dst+1;
17 % src_new = src+1;
18 % initial_set = initial_set+1;
19
20 dst_new = [dst; initial_set']; % Append initial set to the destination
21                                % vector
22 new_state_index =(m+1);
23
24 % Append the source vector with indecies of the newly created state
25 src_new = [src ; new_state_index*ones(num_init,1)];
26
27 w_new = [w;ones(num_init,1)]; % Append the weights of the additional
       edges
28
29 adj_new = sparse(dst_new,src_new,w_new,m+1,n+1); % Form the new adjcency
30
31 end
```

B-21 mipl_ts2promela.m

```
1  % Description: This function generates the Promela file that represents
2  % the transition system of the finite abstraction.
3
4  function mipl_ts2promela(P_labeled,adj,filename,s0)
5
6  % Extract the label of s0
7  s0_label = P_labeled{3}{s0};
8
9  % n is the number of atomic propositions
10 n = length(s0_label);
11
12 % N is the number of states in the transition system
13 N = size(adj,1);
14
15 % add pml extension to filename
16 filename = sprintf('%s.pml',filename);
17
18 % open a file, if the file is already exists then it will be overwritten
19 % the promela code for spin will be written to this file
20 fid = fopen(filename,'w');
21
22 %% Start Promela Code
23
24 % state is a variable to represent the current state
25 fprintf(fid,'int state;\n\n');
26
```

```
27
28 % initialize p1, p2, ..., pn variables
29 for i = 1:n
30     if(s0_label(n+1-i) == 0)
31         fprintf(fid,'bool p%d = false; \n',i);
32     else
33         fprintf(fid,'bool p%d = true; \n',i);
34     end
35 end
36
37 % constructing the transition system
38 fprintf(fid,'\n\nproctype fsm()\n{\n   do\n');
39
40 % remember, the relation is coming from src and going to dest
41 [dest,src] = find(adj);
42
43 for k = 1:length(dest)
44     % d_step is used to improve the spin's performance
45     fprintf(fid,'   :: d_step\n      {\n         (state == %d) -> state =
           %d;\n',[src(k),dest(k)]);
46
47     % Extract the label of the destination state
48     dest_label = P_labeled{3}{dest(k)};
49
50     % Add some space
51     fprintf(fid,'         ');
52
53     % Assign p1, p2, ..., pn variables according to the label of sk
54     for i = 1:n
55         % The least significant bit (LSB) of state label corresponds to
              p1
56         if(dest_label(n+1-i) == '0')
57             fprintf(fid,'p%d=false;  ',i);
58         else
59             fprintf(fid,'p%d=true;  ',i);
60         end
61     end
62
63     % closing bracket for d_step
64     fprintf(fid,'\n      }\n');
65 end
66
67 % closing bracket for proctype fsm
68 fprintf(fid,'   od\n}');
69
70
71 % init proc, similar to main in c
72 fprintf(fid,'\n\ninit\n{\n   d_step\n   {\n      state=%d;\n',s0);
73
74 % Add some space
75 fprintf(fid,'      ');
76
77 % Assign p1, p2, ..., pn variables according to the label of s0
```

```
78 for i = 1:n
79     if(s0_label(n+1-i) == 0)
80         fprintf(fid,'p%d=false;  ',i);
81     else
82         fprintf(fid,'p%d=true;  ',i);
83     end
84 end
85
86 % closing bracket for d_step and init proc, respectively
87 fprintf(fid,'\n   }\n   run fsm();\n}');
88
89 % End Promela Code
90 %%
91
92 % close the file
93 fclose(fid);
94
95 end
```

Bibliography

[1] R. Alur, T. A. Henzinger, and E. D. Sontag, *Hybrid Systems III: Verification and Control*, vol. 3. Springer, 1996.

[2] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer London, Limited, 2009.

[3] G. E. Fainekos, A. Girard, and G. J. Pappas, "Hierarchical synthesis of hybrid controllers from temporal logic specifications," in *Hybrid Systems: Computation and Control*, pp. 203–216, Springer, 2007.

[4] P. Tabuada and G. J. Pappas, "Linear time logic control of discrete-time linear systems," vol. 51, no. 12, pp. 1862–1877, 2006.

[5] D. Hristu-Varsakelis and W. S. Levine, eds., *Handbook of networked and embedded control systems*. Control Engineering, Boston, MA: Birkhäuser Boston Inc., 2005.

[6] P. Antsaklis and J. Baillieul, "Special issue on technology of networked control systems," *Proceedings of the IEEE*, vol. 95, pp. 5 –8, Jan. 2007.

[7] G. Nair, F. Fagnani, S. Zampieri, and R. Evans, "Feedback control under data rate constraints: An overview," *Proceedings of the IEEE*, vol. 95, pp. 108 –137, Jan. 2007.

[8] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, pp. 138–162, Jan. 2007.

[9] K. J. Åström and B. M. Bernhardsson, "Comparison of Riemann and Lebesgue sampling for first order stochastic systems," in *Proceedings of the 41st IEEE Conference on Decision and Control*, vol. 2, pp. 2011 – 2016, Dec. 2002.

[10] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Transactions on Automatic Control*, vol. 52, pp. 1680 –1685, Sept. 2007.

[11] M. Velasco, J. Fuertes, and P. Marti, "The self triggered task model for real-time control systems," in *Work in Progress Proceedings of the 24th IEEE Real-Time Systems Symposium*, pp. 67–70, 2003.

[12] W. Heemels, J. Sandee, and P. van den Bosch, "Analysis of event-driven controllers for linear systems," *International Journal of Control*, vol. 81, no. 4, pp. 571–590, 2008.

[13] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Real-Time Systems*, vol. 1, no. 1, pp. 27–60, 1989.

[14] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, vol. 24. Springer, 2011.

[15] G. C. Walsh and H. Ye, "Scheduling of Networked Control Systems," *IEEE Control Systems Magazine*, 2001.

[16] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Lecture Notes in Computer Science, Springer, 2001.

[17] R. L. Cruz, "A calculus for network delay, part i: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, 1991.

[18] R. L. Cruz, "A calculus for network delay, part ii: Network analysis," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 132–141, 1991.

[19] C.-S. Chang, *Performance Guarantees in Communication Networks*. London, UK, UK: Springer-Verlag, 2000.

[20] Y. Jiang, "A basic stochastic network calculus," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, (New York, NY, USA), pp. 123–134, ACM, 2006.

[21] R. Agrawal, R. L. Cruz, C. M. Okino, and R. Rajan, "A framework for adaptive service guarantees," in *Proc. Allerton Conference on Communication, Control, and Computing*, pp. 693–702, Sept. 1998.

[22] J. Turner, "New directions in communications (or which way to the information age?)," *Communications Magazine, IEEE*, vol. 24, no. 10, pp. 8–15, 1986.

[23] R. L. Cruz, "Quality of service guarantees in virtual circuit switched networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1048–1056, 1995.

[24] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Symposium proceedings on Communications architectures & protocols*, SIGCOMM '89, (New York, NY, USA), pp. 1–12, ACM, 1989.

[25] P. Goyal, S. S. Lam, and H. M. Vin, "Determining end-to-end delay bounds in heterogeneous networks," in *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, NOSSDAV '95, (London, UK, UK), pp. 273–284, Springer-Verlag, 1995.

[26] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, pp. 611–624, Oct. 1998.

[27] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and Linearity: an algebra for discrete event systems*. Wiley, 1992.

[28] C.-S. Chang, "A filtering theory for deterministic traffic regulation," in *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, vol. 2, pp. 436–443 vol.2, 1997.

[29] J.-Y. Le Boudec, "Application of network calculus to guaranteed service networks," *IEEE Transactions on Information Theory*, vol. 44, pp. 1087–1096, May 1998.

[30] C.-S. Chang and R. L. Cruz, "A time varying filtering theory for constrained traffic regulation and dynamic service guarantees," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 63–70 vol.1, 1999.

[31] C.-S. Chang, R. L. Cruz, J.-Y. Le Boudec, and P. Thiran, "A min, + system theory for constrained traffic regulation and dynamic service guarantees," *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, pp. 805–817, 2002.

[32] R. L. Cruz and C. M. Okino, "Service guarantees for window flow control," in *Proc. Allerton Conference on Communication, Control, and Computing*, Oct. 1996.

[33] C.-S. Chang, "On deterministic traffic regulation and service guarantees: a systematic approach by filtering," *IEEE Transactions on Information Theory*, vol. 44, no. 3, pp. 1097–1110, 1998.

[34] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview." RFC 1633 (Informational), Jun. 1994.

[35] S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service." RFC 2212 (Proposed Standard), Sept. 1997.

[36] B. Davie, A. Charny, J. C. Bennet, K. Benson, J.-Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis, "An expedited forwarding phb (per-hop behavior)," RFC 3246, Mar. 2002.

[37] A. Charny, J. Bennet, K. Benson, J. Boudec, A. Chiu, W. Courtney, S. Davari, V. Firoiu, C. Kalmanek, and K. Ramakrishnan, "Supplemental information for the new definition of the ef phb (expedited forwarding per-hop behavior)," RFC 3247, Mar. 2002.

[38] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services." RFC 2475 (Informational), Dec. 1998. Updated by RFC 3260.

[39] V. Firoiu, J.-Y. Le Boudec, D. Towsley, and Z.-L. Zhang, "Theories and models for internet quality of service," *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1565–1591, 2002.

[40] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[41] A. Pnueli, "The temporal logic of programs," in *Proceedings of 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 46–57, 1977.

[42] D. Adzkiya, B. De Schutter, and A. Abate, "Abstraction and verification of autonomous max-plus-linear systems," in *American Control Conference (ACC), 2012*, pp. 721–726, 2012.

[43] D. Adzkiya, B. De Schutter, and A. Abate, "Finite abstractions of max-plus-linear systems," *IEEE Transactions on Automatic Control*, Dec. 2013. To be published.

[44] E. D. Sontag, "Nonlinear regulation: The piecewise linear approach," *IEEE Transactions on Automatic Control*, vol. 26, no. 2, pp. 346–358, 1981.

[45] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *Automatic Verification Methods for Finite State Systems* (J. Sifakis, ed.), vol. 407 of *Lecture Notes in Computer Science*, ch. 17, pp. 197–212, Springer Berlin Heidelberg, 1990.

[46] G. J. Holzmann, "The model checker spin," *IEEE Transactions on Software Engneering*, vol. 23, pp. 279–295, May 1997.

[47] I. Wegener, *Branching Programs and Binary Decision Diagrams - Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[48] D. Adzkiya and A. Abate, "VeriSiMPL: Verification via biSimulations of MPL models," vol. 8054 of *Lecture Notes in Computer Science*, pp. 253–256, Springer Berlin Heidelberg, Sept. 2013.