Towards Unified Tool Support for Real-time Calculus & Deterministic Network Calculus

Philipp Schon and Steffen Bondorf
Distributed Computer Systems (DISCO) Lab,
University of Kaiserslautern, Germany
email: {schon,bondorf}@cs.uni-kl.de

***Abstract*—Real-Time Calculus (RTC) and Deterministic Network Calculus (DNC) both provide the tooling to derive worst-case performance bounds for characteristics of distributed and networked systems: nodal backlog and communication delay. In fact, both were derived from a common basis, yet, their respective evolution let them diverge in various aspects. These are also reflected in the strengths of current, separate tools for RTC and DNC. In this paper, we investigate a potential unification of their tool support. To that end, we provide insights on similarities and differences, suggestions to exploit and overcome them, as well as a benchmark showing first benefits of a unified tool.**

I. Application Domain & Challenge

Knowledge about rutime performance is central to the operation as well as the certification of systems with real-time constraints. For formal verification, worst-case analyses are applied. These guarantee to deterministically bound the potential worst-case behavior of a system w.r.t. certain performance metrics. In distributed computing and communication systems, these are foremost the buffer demand of system components and worst-case execution or traversal times, i.e., delay. Real-Time Calculus (RTC) [1] and Deterministic Network Calculus (DNC) [2] are two versatile methodologies that can compute such bounds. They both employ envelope functions that deterministically bound system behavior, i.e., worst-case service offerings and task or data arrivals, as their principal model. These so-called curves have been accompanied with operations. They manipulate curves to compute the aforementioned worst-case performance bounds [2], [3].

Verification of worst-case performance measures becomes ever more important for industries like avionics. For instance, generations of in-cabin network designs by Airbus, formerly EADS, have been analyzed with RTC (Heterogeneous Communications System [4]) and DNC (AFDX data networks [5]). Yet, these efforts look rather distinct due to the calculi's different analysis approaches. We illustrate them on the prime freely available tools: the RTC toolbox [6] and the DiscoDNC [7].

The RTC toolbox aims at modular performance analysis. Its standard way of usage is via its Matlab frontend. It provides an abstraction of the calculus operations to components such as the Greedy Processing Component (GPC). These are crossed by the resources considered in the analysis. When crossing a GPC, available service becomes left-over service to be used by a subsequent component and data arrival constraints become data output constraints to be fed into another component. In fact, the analysis frontend thus requires to provide the order of calculus operations by interconnecting components correctly.

The DiscoDNC [7], in contrast, automated the derivation of the order of the calculus' operations. It takes a network model consisting of servers that is crossed by data flows – both deterministically modeled with service curves and arrival curves. The order of operations to apply to the curves is then derived by the DiscoDNC. The tool offers alternative derivations that consider different analysis principles. Depending on the analyzed network size, they can vastly increase the quality of end-to-end delay bounds as well as analysis run times [8].

In this paper, we investigate a potential unification of tool support for RTC and DNC.

II. Motivation

In the previous section, we have seen that RTC and DNC have fundamental similarities as well as defining differences. Our working hypothesis states that there are three basic parts of each RTC and DNC tool:

1) the curve backend,
2) the algebraic calculus operations, and
3) the analysis framework (e.g., components or networks).

In theory, a specific part should be exchangeable with the implementation provided by the respective other tool. Then, a unified tool can be composed to provide the strengths of the RTC toolbox and the DiscoDNC on demand or simultaneously. Moreover, work regarding the performance of executing the analysis itself matured in RTC and DNC [9], [10]. It could thus immediately improve both with unified tool support.

In this paper, we investigate practical aspects that might inhibit potential to unify RTC and DNC tools. We initially focus on part 1) at the very basis of all tools: the curve implementation. Both, the RTC toolbox and the DiscoDNC provide a Java implementation of curves, yet, degrees of freedom already result in crucial differences and incompatibilities, as we will show later. As it turns out, isolating this part and swapping it out is not a simple task. Also note, that among these tools only the DiscoDNC is open-source software.

Nonetheless, a unification is worthwhile. The DiscoDNC only implements support for aperiodic, ultimately affine curves [11] while the RTC toolbox can operate on more complex curves as well, e.g., staircase functions with a periodic tail. Thus, as a proof of concepts for the benefits of a unified tool, we aim to integrate the RTC toolbox's curve backend in the automated network analyses of the DiscoDNC.

This work is supported by a Carl Zeiss Foundation grant.

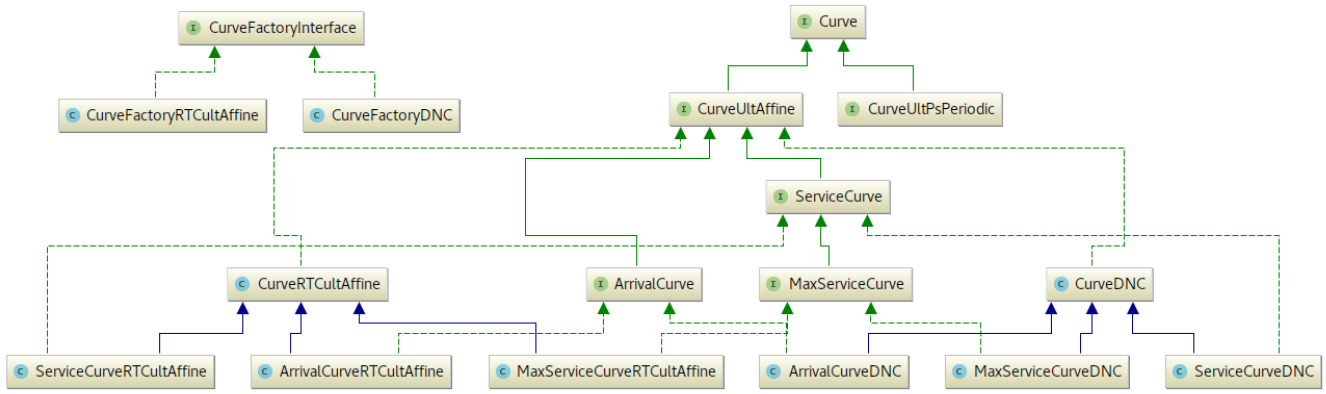

Fig. 1: Structure of the configurable curve backend for the DiscoDNC. The curve factory (upper left side) handles the creation of RTC or DNC curves according the tool's configuration. A hierarchy of interfaces (labeled by circled `I`) determines the curve shape (ultimative affine, *`ultAffine`, [11]) and their semantic (`ArrivalCurve`, `ServiceCurve`, `MaxServiceCurve`). The interfaces are implemented by the DiscoDNC as well as wrappers classes for RTC toolbox (labeled by circled `C`).

III. Problem Statement

A. Interfacing Functionality of the Curve Classes

We base our work on the latest versions the tools: RTC toolbox version 1.2.beta.100 and the DiscoDNC v2.3. These are already modularized according the three basic parts identified in the previous section. Each tool provides a class to create and manipulate piece-wise linear curves. Therefore, the algebraic calculus operations building on the curve classes should, in principle, be able to operate with either implementation. However, there is no common interface for curves. Detailed background regarding classification of curves as well as closure of algebraic operations applied to curves from the presented classes can be found in [11]. Yet, the provided functionality regarding the manipulation of curves differs significantly between the RTC toolbox and the DiscoDNC. This is a result of the different classes of curves they implement as well as the calculus operations applied to them. I.e., there is actually an interdependency between the two fundamental parts 1) and 2) to overcome. Unfortunately, none of the tools offers a superset of the functionality required by all operator implementations.

B. Continuity of Curves

As we aim at integration of the RTC toolbox's curve backend into the DiscoDNC, we investigated the required but missing functionality from its point of view. These features are almost exclusively simple queries for curve properties. Yet, there is one significant exception: In contrast to the DiscoDNC, the RTC toolbox does not explicitly store information about continuity. I.e., there might be spots defined by two linear segments, at the very end of the support of a first segment and the start of the support of an immediately following, second segment. This mostly concerns the staircase curves, yet, also the token bucket curve used to represent shaped arrivals:

$$\texttt{Arrival Curve} \quad \gamma_{r,b}(t) = \begin{cases} 0 & t \leq 0 \\ b + r \cdot t & \text{otherwise} \end{cases}$$

where b denotes the maximum burstiness and r is the maximum arrival rate of a flow or an event stream. According to the definition's case distinction, any $\gamma_{r,b}(t)$ curve is constructed from two linear segments. The second segment's support is $(0, \infty)$ and the first segment's support is $(\infty, 0]$, yet, arrivals for times $t < 0$ are not relevant to the analysis and thus not stored. The DiscoDNC's operations depend on explicit labeling of continuity where the curve's defining segment changes. In conformity with the DNC literature [3], we decided to assume left-continuity in case there is no continuity label.

IV. Proposed Approach and Preliminary Results

We implemented the following additions and adaptations in order to be able to swap the curve backend of the DiscoDNC:

- Interface definitions
- Wrapper classes for the RTC toolbox
- Factory pattern for curve creation

The resulting code structure is depicted in Fig. 1. It allows for the first comparative evaluation of both curve implementation and substantiates the feasibility of our approach.

A. Interface Definitions and Wrapper Classes

We focused on the curves that can be properly handled by the DiscoDNC operations, i.e., piece-wise linear, ultimately affine curves (abbreviated ultAffine). The new interface is thus called `CurveUltAffine` and any interface or class integrating RTC toolbox functionality is suffixed accordingly in order to reflect this restriction (see Fig. 1).

In detail, we implemented interfaces that define all relevant curve manipulations and mutuator methods needed by the DiscoDNC operations. Then, we adapted the DiscoDNC's parts 2) and 3), operations and analyses, to work on objects implementing these interfaces. Thus, we achieved the decoupling of parts as envisioned in Section II. Completing this approach, we implemented interfaces for the specific curves of RTC and DNC (`ArrivalCurve`, `ServiceCurve`, and `MaxServiceCurve` derived from generic `Curve`) as well as the linear segments that define a curve (`LinearSegment`).

As the RTC toolbox is not open source, we were not able to implement our newly defined interfaces in it. Instead, we investigated the possibility to integrate closed-source implementations by wrapper classes. Wrappers for all curve classes as well as the linear segments class provide proxy functions calling the actual functions of the RTC toolbox API. The RTC toolbox has a single curve class that all wrappers for use in the DiscoDNC map to. I.e., we can also adjust such specific differences of RTC and DNC tool implementations. In the current state of our work, we also block access to the RTC toolbox's complex curve constructors because wrappers only implement the `*ultAffine` interfaces. Moreover, these wrappers handle the implicit left-continuity and the problems arising from it. E.g., the token bucket curve's spot in the origin is problematic as curves are supposed to be defined for positive reals only. All this adds complexity to the code path using the RTC toolbox backend. Therefore, we benchmark performance of both curve backends at the end of this section.

B. The Curve Factory

Our goal was to provide a flexible solution to switch between curve backends by setting a configuration flag. To do so, we followed the factory method pattern that allows to create objects without specifying the exact class. I.e., the factory checks a global configuration flag and defers instantiation of objects to the according factory class for either the RTC toolbox or the DiscoDNC (see Fig. 1, upper left part). Each curve object's type is one of the `*Curve` interfaces the remaining parts of the DiscoDNC were adapted to work with.

As a first result, we can report that all the DiscoDNC's functional tests succeed with both curve backends – RTC toolbox and DiscoDNC. This already shows that we can overcome significant differences with sophisticated wrapper classes. Their impact on performance will be investigated next.

C. A First Benchmark

For the performance benchmark of both curve implementations, we used the test networks provided in [8]. Due to their sizes, it becomes impractical to model the proceeding of the analysis by manually defining and connecting components. Instead, for each flow in each network, the DiscoDNC derives the end-to-end delay bound with all three analyses automated by it: Total Flow Analysis (TFA), Separated Flow Analysis (SFA), and Pay Multiplexing Only Once (PMOO). As in the literature [8], we measured the run time of computations on a dedicated server to ensure reproducibility of results. The entire network analysis run times are shown in Fig. 2.

Run times of both backends scale similarly with the network size. However, the RTC toolbox's backend achieves visibly better run times in larger networks that demand more curve manipulations. This comes to a surprise as the RTC is accessed via the performance-degrading wrappers whereas we directly implemented the interfaces in the DiscoDNC. Moreover, the DiscoDNC's curve class is specialized to the aperiodic, ultimately affine curves used in this benchmark. The RTC toolbox is not. Its storage for linear segments as well as their manipulation mapped to by the wrapper are designed to work with more general classes of curves. Thus, we assume its code to be more complex as well.

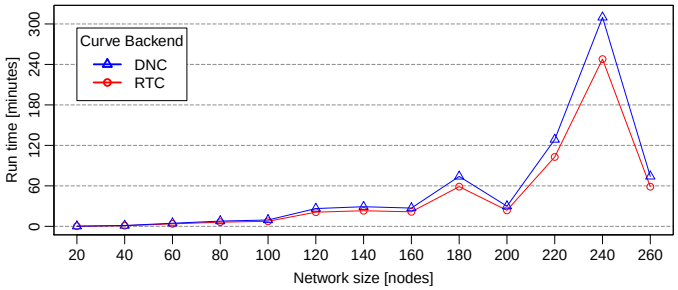

Fig. 2: Run time comparison of the curve backends analyzing networks given in [8]. While the difference is negligible for networks up to 100 nodes, better performance of the RTC toolbox curve backend becomes visible in larger networks.

V. Envisioned Solution

In this paper, we could demonstrate feasibility and benefits of our unification approach. As future work, we will define interfaces for the remaining classes of curves defined in [11]. Yet, we could already achieve a first goal. The RTC toolbox's analysis framework usually sums up component-local delays to an end-to-end delay. This is comparable with DNC's TFA procedure that, however, has been superseded by the end-to-end analyses SFA and PMOO. Hence, we also made the first step towards using these analyses with RTC toolbox curves.

Moreover, we want to define the interface between the algebraic operations of part 2) and the analyses frameworks of part 3) of the tools. Then, more parts can become exchangeable. Thus, we hope to be able to create a flexible unified tool that provides functionality not attainable today, foremost, an automated DiscoDNC analysis of a network modeled by periodic RTC toolbox curves not part of the DiscoDNC. This analysis has the potential to yield the most accurate results with fastest analysis run times [9], [10].

References

[1] L. Thiele, S. Chakraborty, and M. Naedele, "Real-Time Calculus for Scheduling Hard Real-Time Systems," in *Proc. ISCAS*, March 2000.

[2] R. L. Cruz, "A Calculus for Network Delay. Part I: Network Elements in Isolation and Part II: Network Analysis," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–141, January 1991.

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

[4] U. Suppiger, S. Perathoner, K. Lampka, and L. Thiele, "Modular performance analysis of large-scale distributed embedded systems: An industrial case study," ETH Zurich, Tech. Rep. 330, Nov 2010.

[5] J. Grieu, "Analyse et évaluation de techniques de commutation ethernet pour l'interconnexion des systèmes avioniques," Ph.D. dissertation, INPT, 2004.

[6] E. Wandeler and L. Thiele, "Real-Time Calculus (RTC) Toolbox," `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[7] S. Bondorf and J. B. Schmitt, "The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus," in *Proc. ValueTools*, 2014.

[8] S. Bondorf, P. Nikolaus, and J. B. Schmitt, "Quality and cost of deterministic network calculus – design and evaluation of an accurate and fast analysis," in *Proc. ACM SIGMETRICS*, 2017.

[9] K. Lampka, S. Bondorf, and J. B. Schmitt, "Achieving efficiency without sacrificing model accuracy: Network calculus on compact domains," in *Proc. IEEE MASCOTS*, 2016.

[10] K. Lampka, S. Bondorf, J. B. Schmitt, N. Guan, and W. Yi, "Generalized finitary Real-Time calculus," in *Proc. IEEE INFOCOM*, 2017.

[11] A. Bouillard and E. Thierry, "An Algorithmic Toolbox for Network Calculus," *JDEDS*, 2008.