Towards Certifying Network Calculus

Etienne Mabille[1], Marc Boyer[2], Loïc Fejoz[1], and Stephan Merz[3]

[1] RealTime at Work, Nancy, France
[2] ONERA, The French Aerospace Lab, Toulouse, France
[3] Inria & LORIA, Nancy, France

1 Result Certification for Network Calculus

Network Calculus (NC) [5] is an established theory for determining bounds on message delays and for dimensioning buffers in the design of networks for embedded systems. It is supported by academic and industrial tool sets and has been widely used, including for the design and certification of the Airbus A380 AFDX backbone [1, 3, 4]. However, while the theory of NC is generally well understood, results produced by existing tools have to be trusted: some algorithms require subtle reasoning in order to ensure their applicability, and implementation errors could result in faulty network design, with unpredictable consequences.

Tools used in design processes for application domains with strict regulatory requirements are subject to a qualification process in order to gain confidence in the soundness of their results. Nevertheless, given the safety-critical nature of network designs, we believe that more formal evidence for their correctness should be given. We report here on work in progress towards using the interactive proof assistant Isabelle/HOL [6] for certifying the results of NC computations. In a nutshell (cf. Figure 1), the NC tool outputs a trace of the calculations it performs, as well as their results. The validity of the trace (w.r.t. the applicability of the computation steps and the numerical correctness of the result) is then established offline by a trusted checker.

The approach of result certification is useful in general for computations performed at design time, as is the case with the use of NC tools, and the idea of using interactive theorem provers for result certification is certainly not new. In particular, it is usually easier to instrument an existing tool in order to produce a checkable trace than to attempt a full-fledged correctness proof. Also, the NC tool can be implemented by a tool provider using any software development process, programming language, and hardware, and it can be updated without having to be requalified, as long as it still produces certifiable traces.

In the remainder, we give a brief introduction to NC, outline our ongoing work on formalizing NC in Isabelle/HOL, and finally illustrate its use for the certification of bounds on the message delay in a toy network.

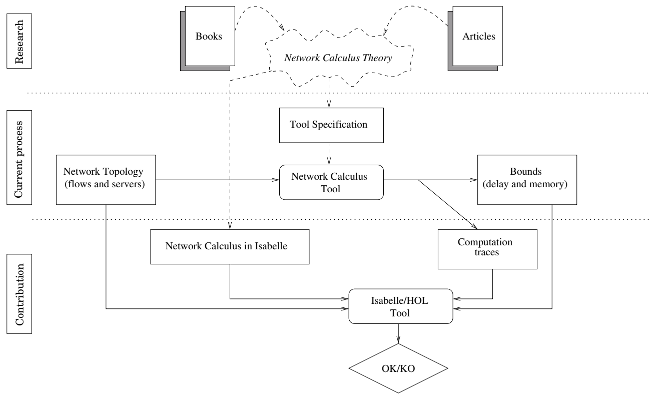

Fig. 1. Proof by instance of NC computations.

2 Network calculus

Network calculus [5] is a theory for computing upper bounds in networks. Its mathematical background is a theory of the set of functions

$$\mathcal{F} = \{\, f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \cup \{\infty\} \mid x \leq y \implies f(x) \leq f(y) \,\} \tag{1}$$

that form a dioid under the operations $\sqcap$ and $+$ defined as pointwise minimum and addition. Practical applications make frequent use of four families of functions, defined as $\delta_d(t) = 0$ if $t \leq d$ and ∞ otherwise, $\beta_{R,T}(t) = 0$ if $t \leq T$ and $R(t - T)$ otherwise, and $\gamma_{r,b}(t) = 0$ if $t \leq 0$ and $rt + b$ otherwise (all parameters denote real numbers).

Operations of interest on $\mathcal{F}$ include convolution $*$, deconvolution $\oslash$, and the sub-additive closure f^*.

$$\begin{aligned} (f * g)(t) &= \inf_{0 \leq u \leq t} (f(t-u) + g(u)) && (2)\\ (f \oslash g)(t) &= \sup_{0 \leq u} (f(t+u) - g(u)) && (3)\\ f^* &= \delta_0 \sqcap f \sqcap (f * f) \sqcap (f * f * f) \sqcap \cdots && (4) \end{aligned}$$

A *flow* is represented by its cumulative function $R \in \mathcal{F}$, where $R(t)$ is the total number of bits sent by this flow up to time t. A flow R has function $\alpha \in \mathcal{F}$ as *arrival curve* (denoted $R \preceq \alpha$) if $\forall t, s \geq 0 : R(t+s) - R(t) \leq \alpha(s)$, meaning that, from any instant t, the flow R will produce at most $\alpha(s)$ new bits of data in s time units. Using convolution, this condition can be equivalently expressed as $R \leq R * \alpha$. If α is an arrival curve for R, so is α^*, and also any $\alpha' \geq \alpha$.

$$R \preceq \alpha \implies R \preceq \alpha^* \quad (5) \qquad\qquad R \preceq \alpha, \alpha \leq \alpha' \implies R \preceq \alpha' \quad (6)$$

A *server* S is a relation between an input flow R, and an output flow R' (denoted $R \xrightarrow{S} R'$) such that $R' \leq R$ (representing the intuition that the flow

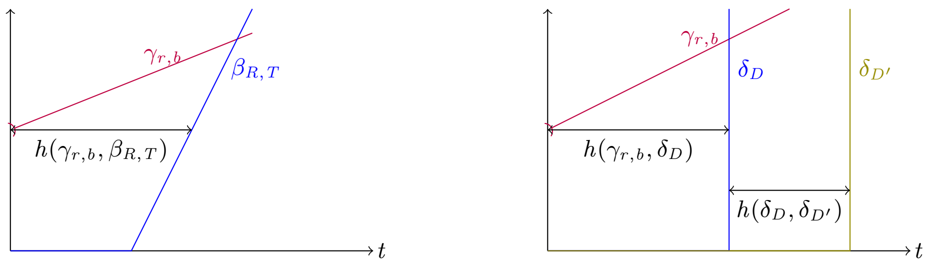

Fig. 2. Common curves and delay.

crosses the server, and that the output is produced after the input). Such a server has a *service curve* β if $R' \geq R * \beta$ holds. The delay incurred by the flow R can be bounded by the maximal horizontal difference between curves α and β, formally defined as

$$h(\alpha, \beta) \;=\; \sup_{s \geq 0} \; (\inf \{ \tau \geq 0 \mid \alpha(s) \leq \beta(s + \tau) \}) \tag{7}$$

(cf. also Fig. 2). If R has arrival curve α and S has service curve β then $\alpha \oslash \beta$ and $\alpha \oslash \delta_{h(\alpha,\beta)}$ are two possible arrival curves for R'.

This presentation gives a flavor of network calculus as a collection of algebraic results useful for computing bounds on curves and delays. Consider a configuration with a flow R crossing two servers S_1, S_2 in sequence: $R \xrightarrow{S_1} R' \xrightarrow{S_2} R''$. Assume that R has arrival curve α and that each server S_i offers a service of curve β_i. Then, the delay of R in S_1 can be bounded by $d = h(\alpha, \beta_1)$, and $\alpha' = \alpha \oslash \delta_d$ is a possible arrival curve for R'. Its sub-additive closure $(\alpha')^*$ is also an arrival curve for R' (by Eq. 5), but may be too expensive to compute. A simpler approximation is given by $\alpha' \sqcap \delta_0 \geq (\alpha')^*$ (using Eq. 6), and the delay of R' in S_2 can be bounded by $h(\alpha' \sqcap \delta_0, \beta_2)$. The end-to-end delay can also be bounded by the sum of bounds on local delays, *i.e.* $h(\alpha, \beta_1) + h(\alpha' \sqcap \delta_0, \beta_2)$.

This simple example illustrates that implementations of NC analysis may choose between different approximations, involving tradeoffs between the accuracy of the result, the difficulty of implementing the necessary computations, and their time complexity.

3 Encoding Network Calculus in Isabelle

The first step towards developing a result certifier consists in formalizing the theory underlying NC to the extent that it is used by algorithms we are interested in. As a side benefit of this formalization, we obtain a rigorous development of NC, including all possible corner cases that may be overlooked in pencil-and-paper proofs. The objective of the work reported here was to evaluate the feasibility of developing a result certifier in Isabelle that would at least be able to check computations for simple, but representative networks. Our NC formalization is currently incomplete, with many theorems only partly proved; we nevertheless outline the main definitions and results.

The set $\mathcal{F}$ (Eq. 1) of non-decreasing functions used to represent flows is represented in Isabelle/HOL as the type

$$\textbf{typedef}\ \mathit{ndf} = \{ f :: \mathit{ereal} \Rightarrow \mathit{ereal}\ .\ (\forall r \leq 0.\, f\, r = 0) \wedge \mathit{mono}\ f\, \}$$

where *ereal* is a pre-defined type corresponding to $\mathbb{R} \cup \{\infty\}$. Compared to (1), we extend the domain of f to $\mathbb{R} \cup \{\infty\}$ (including negative numbers and ∞) but require that $f\ r$ be zero for negative r. This insignificant change of definition turned out to simplify the subsequent development. Over type *ndf*, we define operations such as addition, multiplication, and comparison by pointwise extension and establish some basic algebraic properties: for example, the resulting structure forms an ordered commutative monoid with 0 and 1.

We introduce operations such as convolution and deconvolution (Eq. 2) and characteristic properties such as sub-additivity, and prove fundamental results. For example, the convolution of two sub-additive flows is itself sub-additive.[4]

definition *is-sub-additive* **where**
  $\mathit{is\text{-}sub\text{-}additive}\ f \ \equiv\ \forall x\, y.\ f \cdot (x + y) \leq f \cdot x + f \cdot y$
lemma *convol-sub-add-stable*:
  assumes *is-sub-additive f* **and** *is-sub-additive g*
  shows *is-sub-additive* $(f * g)$

A *simple server* is represented as a left-total relation between flows such that the output flow is not larger than the input flow

$$\textbf{typedef}\ \mathit{server} = \{\, s :: (\mathit{ndf} \times \mathit{ndf})\ \mathit{set}.\ (\forall \mathit{in}.\ \exists \mathit{out}.\ (\mathit{in}, \mathit{out}) \in s) \wedge (\forall (\mathit{in}, \mathit{out}) \in s.\ \mathit{out} \leq \mathit{in})\,\}$$

and we define what it means for a flow to be constrained by an arrival curve α and for a server to provide minimum service β:

$$R \preceq \alpha \ \equiv\ R \leq R * \alpha \qquad S \trianglerighteq \beta \ \equiv\ \forall (\mathit{in}, \mathit{out}) \in S : \mathit{in} * \beta \leq \mathit{out}.$$

Again, we prove results relating these constraints to bounds on delays and backlogs. For example, the following theorem provides a bound on the delay of a simple server:

theorem *d-h-bound*:
  assumes $\mathit{in} \preceq \alpha$ **and** $S \trianglerighteq \beta$
  shows $\mathit{worst\text{-}delay\text{-}server}\ \mathit{in}\ S \leq \mathit{h\text{-}dev}\ \alpha\ \beta$

where the horizontal deviation is defined in (Eq. 7) and *worst-delay-server in S* denotes the maximal delay incurred by input flow *in* at server S.

Building on these results about simple servers, we derive theorems about sequences of servers. We also formalize concepts such as *packetization*, which refers to servers that group individual bits into larger packets, introducing extra delays. Finally, these concepts are extended to multiple-input multiple-output servers that takes vectors of flows as input and output. We do not describe these concepts in detail, as they are not used in the following example.

[4] $f \cdot x$ denotes the result of applying $f :: \mathit{ndf}$ to x.

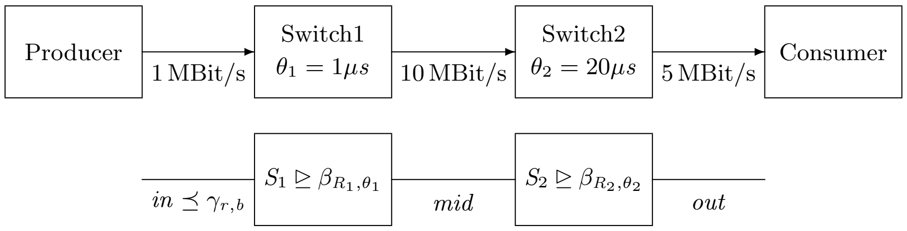
Producer
1 MBit/s
Switch1
$\theta_1 = 1\mu s$
10 MBit/s
Switch2
$\theta_2 = 20\mu s$
5 MBit/s
Consumer
$in \preceq \gamma_{r,b}$
$S_1 \trianglerighteq \beta_{R_1,\theta_1}$
mid
$S_2 \trianglerighteq \beta_{R_2,\theta_2}$
out

Fig. 3. A simple system and its Network Calculus representation.

4 Certifying a Simple Network Computation

In order to illustrate the use of our theories on a simple example, let us consider the producer-consumer setup shown in Fig. 3. The producer is assumed to send at most one frame every $T = 20\,\text{ms}$. We further assume that the maximum frame size is $MFS = 8000\,\text{bits}$. This flow is sent to a consumer via two switches with switching delays $\theta_1 = 1\,\mu s$ and $\theta_2 = 20\,\mu s$. The physical links between the producer, the switches, and the consumer are assumed to have bandwidths of 1, 10 and 5 MBit/s, respectively.

The NC model appears in the lower part of Fig. 3. Flow *in* is constrained by the arrival curve $\alpha_{in} = \gamma_{r,b}$ where b equals MFS and $r = \frac{MFS}{T} = \frac{8000}{20\times10^3} = \frac{2}{5}$.

The service curves are given by the function β_{R_i,θ_i} where the bandwidths are $R_1 = 10\,\text{bit}/\mu s$ and $R_2 = 5\,\text{bit}/\mu s$, and the delays are θ_1 and θ_2.

We are interested in the maximal delays that frames may incur. Using theorem *d-h-bound*, the delay at server 1 is bounded by $h(\alpha_{in}, \beta_{10,1})$, which evaluates to $801\,\mu\text{s}$. As explained in Section 2, the arrival curve of flow *mid* can be computed as

$$\alpha_{mid} = (\alpha_{in} \oslash \delta_{801}) \sqcap \delta_0 = (\gamma_{\frac{2}{5},8000} \oslash \delta_{801}) \sqcap \delta_0 = \gamma_{\frac{2}{5},\frac{41602}{5}}.$$

Continuing for the second server, its delay is at most $h(\alpha_{mid}, \beta_{5,20}) = \frac{42102}{25}\,\mu\text{s}$. Consequently, the overall delay incurred by frames equals

$$801\,\mu\text{s} + \frac{42102}{25}\,\mu\text{s} = \frac{62127}{25}\,\mu\text{s}.$$

These computations are performed by the PEGASE Network Calculus tool [2] and certified using Isabelle.

5 Conclusion

We have presented preliminary work aiming at ensuring the correctness of embedded network designs by certifying the result of standard NC tools within a theory developed in the proof assistant Isabelle/HOL. A prototype has been developed and it can handle a realistic industrial configuration, with 8 switches and more than 5.000 flows, in 8 hours on a standard laptop computer. Much remains to be done: the proofs of many theorems of the NC formalization are still incomplete. Moreover, we only support simple arrival curves and therefore obtain worse bounds than state-of-the-art tools for NC analysis. Nevertheless, we believe that our work demonstrates the feasibility and the interest of the approach.

Developing a Network Calculus engine that is able to handle an AFDX configuration requires about one or two years of implementation. The effort for developing a qualified version of such an engine, using state-of-the-art techniques (documentations, testing, peer-review, etc.) is higher by a factor of 5 or 10.

Although one should not confuse result certification with the development of a qualified NC tool, the approach that we suggest here promises to reduce the overhead while increasing the confidence in the results produced by the software. We have so far invested less than 1 development year for encoding some fundamental concepts of Network Calculus in Isabelle/HOL, and for instrumenting an existing tool so that it produces a trace that can be checked in Isabelle. We estimate that the overall effort for producing the proof for a realistic network should be between 2 and 3 years. This includes effort to complete the formalization of the basic concepts, extensions to more complicated types of servers, and developing special-purpose proof methods for checking the proof traces.

In other words, we believe that result certification could reduce the overhead for developing a trustworthy version of a Network Calculus tool to a factor of 2 or 3, while significantly improving its quality.

References

1. AEEC. Arinc 664p7-1 aircraft data network, part 7, avionics full-duplex switched ethernet network. Technical report, Airlines Electronic Engineering Committee, september 2009.
2. M. Boyer, N. Navet, X. Olive, and E. Thierry. The PEGASE project: Precise and scalable temporal analysis for aerospace communication systems with network calculus. In T. Margaria and B. Steffen, editors, *4th Intl. Symp. Leveraging Applications (ISoLA 2010)*, volume 6415 of *LNCS*, pages 122–136, Heraklion, Greece, 2010. Springer. http://www.realtimeatwork.com/software/rtaw-pegase/.
3. F. Frances, C. Fraboul, and J. Grieu. Using network calculus to optimize AFDX network. In *Proc. 3thd Europ. Cong. Embedded Real Time Software (ERTS'06)*, Toulouse, January 2006.
4. J. Grieu. *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques.* PhD thesis, Institut National Polytechnique de Toulouse (INPT), Toulouse, Juin 2004.
5. J.-Y. Le Boudec and P. Thiran. *Network Calculus.* Springer, 2001.
6. T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL. A Proof Assistant for Higher-Order Logic.* Number 2283 in Lecture Notes in Computer Science. Springer Verlag, 2002.