

Network Calculus for Real Time Analysis of Embedded Systems with Cyclic Task Dependencies

Abstract

The paper addresses performance analysis associated with analysis and design of embedded real time systems with processor sharing among a number of processes. The main contribution concerns the application of Network Calculus (DNC) principles to the performance analysis of such systems, where complex data flow graphs prohibit the use of standard scheduling analysis techniques. Cyclic flow graphs take the problem into new grounds to which the theoretical basis for DNC is extended in this paper. An example is provided to support the practical relevance of the developed approach denoted CyNC and comparative results for CyNC and alternative methods are presented and commented.

Keywords: analysis of embedded real time systems, network calculus, scheduling, cyclic task dependencies, timed automata

1 Introduction

Embedded systems environments induce performance demands in the specifications for such systems. Therefore performance analysis is an indispensable part of the design process. Analysis of the time-related performance during design of embedded systems has been approached from various angles, such as scheduling analysis [1], completion time analysis [2] and model checking in timed automata [8, 9].

Completion time analysis assumes no or limited dependence between tasks. Task dependence is allowed through blocking in critical regions protecting shared resources. Results for fixed or dynamic priority scheduling, fixed cyclic schedules and round robin exist for that case. Automata-based model-checking in real time systems is based on timed automata modelling. Various externally observable time related properties can be expressed as timed logical propositions, which are then transformed into timed automata augmenting the system automata model.

Frequently task dependency models allowing only mutual exclusion by critical regions are not feasible. For example, in communication devices *producer-consumer* communica-

tion patterns between tasks are often met. Such dependencies bring completion time analysis off grounds and tends to enlarge to reachability sets in timed automata significantly. Within the field of communication systems a theoretical framework for non-stochastic performance analysis has emerged with the network calculus (DNC) of [4], [3]. DNC is based on specifications in the shape of lower and upper bounds on traffic as well as service. From specifications, upper bounds on backlogs and waiting in queues may be deduced.

This work explores how to bring DNC to use in the analysis of embedded real time systems. Earlier work including [5] introduced DNC for embedded systems with heterogeneous architectures and acyclic task dependencies. A related approach is presented in [6], where a *periodic with jitter* model is adopted. Results for cyclic flow graphs are provided under the *periodic with jitter* assumption in [7] including phase relations between events in the analysis. [3] provides results for cyclic dependencies for affine upper flow constraints and lower service constraints. Results for that approach (ADNC) are provided in this paper for comparison on a common example. This work provides theoretical results for cyclic flow graphs under the more general assumptions of minimum and maximum flow constraints and service guarantees.

Initially a short introduction to performance analysis for embedded systems is given through an example illustrating a typical situation. DNC is introduced and a suggestion for migrating this technique to the analysis of embedded systems is presented. A general theoretical basis for the presented approach (CyNC) is given. CyNC is illustrated for a relevant example and results are compared to ADNC and model checking. Algorithmic complexity issues are discussed before conclusive remarks are given along with suggestions for directions of future research.

2 Analysis and Design of Embedded Real Time Systems

The context of embedded real time systems is constituted by the set of external interfaces connected to the system [11]. Time analysis of external interfaces is typically ini-

- Flow *convergence* elements, where traffic flows are aggregated model message flows entering a common buffer.
- Flow *divergence* elements where traffic flows are split into separate subflows model the internal routing of messages in the system.

Migrating the DNC framework from network communication to the task communication in embedded systems involves re-interpretation of some of the element types above and modification of some as well. Network service elements naturally bound output flows R' below corresponding input flows R , i.e.

$$R'(t) \leq R(t) \quad (4)$$

This is not generally the case for tasks in an embedded system, since a single input message may stimulate the transmission of a variable number of output messages. We suggest scaling factors S associated to pairs of corresponding input and output flows. Consider a number of flows $\{f_i\}$ entering an ingress buffer of some task τ producing a number of output flows $\{g_j\}$ in return. Each message from f_i is then assumed to produce S_{ij}^τ messages to g_j . Scaling factors are allowed to be non-integer in which case the effect of packetizing can be accounted for by adding/subtracting the size of one packet to the upper/lower constraints.

4 Performance Analysis

Considering a service element generating an output flow R' in return for an input flow R bounded as in 1. When service is bounded by β_L and β_U the output flow R' is bounded by α'_L and α'_U , where

$$\alpha'_L(t) = \inf_{0 \leq u \leq t} \{\alpha_L(u) + \beta_L(t-u)\} \quad (5)$$

$$\alpha'_U(t) = \inf_{0 \leq u \leq t} \{\sup_{v \geq 0} \{\alpha_U(u+v) - \beta_L(v)\} + \beta_U(t-u), \beta_U(t)\}$$

as given in [5]. Likewise remaining processing resources are bounded by β'_L and β'_U where

$$\beta'_L(t) = \sup_{0 \leq u \leq t} \{\beta_L(u) - \alpha_U(u)\} \quad (6)$$

$$\beta'_U(t) = \sup_{0 \leq u \leq t} \{\beta_U(u) - \alpha_L(u)\}$$

In [5] such bounds are propagated along job flows for systems with acyclic task dependencies, i.e. bounds may be propagated from one end to the other. It is readily verified that linear separation is invariant to the propagation formulae 5 and 6. Thus linear separation for external flows imply the same for internal flows.

Flow between tasks may in our model be scaled (by factors S_{ij}^τ) and aggregated so that flow constraints are computationally propagated through affine operators, i.e. $\alpha_U =$

$\mathbf{A}\alpha'_U + Ua$ and $\alpha_L = \mathbf{A}\alpha'_L + La$, where \mathbf{A} is a stable matrix of scaling factors defined by system data flow, vectors $\{\alpha_U, \alpha_L\}$ and $\{\alpha'_U, \alpha'_L\}$ comprise arrival bounds on flows entering and leaving service elements respectively and where $\{Ua, La\}$ account for external flow constraints and additive effects from packetizing.

Bounds on remaining processing resources are propagated from higher to lower priority tasks, i.e. $\beta_U = \mathbf{B}\beta'_U + Ub$ and $\beta_L = \mathbf{B}\beta'_L + Lb$, where \mathbf{B} is a stable matrix with boolean entries defined by priority order, vectors $\{\beta_U, \beta_L\}$ and $\{\beta'_U, \beta'_L\}$ comprise service bounds passed on and received by service elements respectively. $\{Ub, Lb\}$ account for externally given bounds on processing resources. Composing mappings 5 and 6 with the affine transformations above gives an overall system mapping Γ defined by 7

$$(\alpha'_U, \alpha'_L, \beta'_U, \beta'_L) = \Gamma(\alpha_U, \alpha_L, \beta_U, \beta_L, Ua, La, Ub, Lb) \quad (7)$$

For acyclic dependency a finite number of applications of the overall system mapping Γ produces flow and service constraints in an order defined by the dependency graph. When dataflows are cyclic or are directed from lower to higher priority tasks under processor sharing, cyclic dependencies between constraints appear. In that case constraints are given implicitly by 7 as a fix point to Γ .

5 Theoretical Basis

From a theoretical perspective we should ask for existence and uniqueness of solutions to 7 as well the interpretation w.r.t. arrival and service boundedness. Finally an algorithm solving 7 in some reasonable sense is desirable. We shall subsequently leave out the question of existence and establish results for uniqueness, interpretation and algorithm.

5.1 Uniqueness and Algorithm

Inspecting the mapping

$$(\alpha_U^{n+1}, \alpha_L^{n+1}, \beta_U^{n+1}, \beta_L^{n+1}) = \Gamma(\alpha_U^n, \alpha_L^n, \beta_U^n, \beta_L^n, Ua, La, Ub, Lb) \quad (8)$$

reveals that Γ is monotone in each of its components if relations ($\leq, \geq, =$) are defined elementwise for vectors and pointwise for real functions. If $+, -, 0$ denote non-decrease, non-increase and invariance, we may describe the monotonicity of Γ by the 4 by 4 matrix S

$$S = \begin{bmatrix} + & 0 & + & - & | & + & 0 & + & - \\ 0 & + & - & + & | & 0 & + & - & + \\ + & - & + & 0 & | & + & - & + & 0 \\ - & + & 0 & + & | & - & + & 0 & + \end{bmatrix} \quad (9)$$

indicating f.ex. that α_U^{n+1} does not decrease with α_U^n and that α_U^{n+1} is invariant to changes in α_L^n etc.

Assume the existence of one or more finite positive solutions to 7 and let

$$x^* = (\alpha_U^*, \alpha_L^*, \beta_U^*, \beta_L^*) = (\min(\alpha_U), \max(\alpha_L), \min(\beta_U), \max(\beta_L)) \quad (10)$$

where min and max are taken over all solutions of 7.

Also assume the existence of an initial point $x^0 = (\alpha_U^0, \alpha_L^0, \beta_U^0, \beta_L^0)$ so that

$$(\alpha_U^0 \leq \alpha_U^*, \alpha_L^0 \geq \alpha_L^*, \beta_U^0 \leq \beta_U^*, \beta_L^0 \geq \beta_L^*) \quad (11)$$

and

$$(\alpha_U^1 \geq \alpha_U^0, \alpha_L^1 \leq \alpha_L^0, \beta_U^1 \geq \beta_U^0, \beta_L^1 \leq \beta_L^0) \quad (12)$$

then the sequence $\{\alpha_U^n, \alpha_L^n, \beta_U^n, \beta_L^n\}$ fulfills

$$(\alpha_U^n \leq \alpha_U^*, \alpha_L^n \geq \alpha_L^*, \beta_U^n \leq \beta_U^*, \beta_L^n \geq \beta_L^*) \quad (13)$$

as well as

$$(\alpha_U^{n+1} \geq \alpha_U^n, \alpha_L^{n+1} \leq \alpha_L^n, \beta_U^{n+1} \leq \beta_U^n, \beta_L^{n+1} \geq \beta_L^n) \quad (14)$$

which means that the iterands of 8 converge to x^* which is therefore itself a uniquely determined *best* fixpoint of lowest upper bounds and highest lower bounds.

5.2 Initial point

Our result so far relies on the existence of some initial point x^0 fulfilling 11 and 12 as well as at least one positive finite solution to 7. Under linear separation (3) an appropriate initial point exists since Γ has fixpoints of linear constraints for linear external constraints. This fixpoint is found by solving flow equations for the vectors of upper and lower rates A_U, A_L, B_U, B_L defined in 3, i.e. $A_U = (\mathbf{I} - \mathbf{A})^{-1} r_U^A$, $A_L = (\mathbf{I} - \mathbf{A})^{-1} r_L^A$, $B_U = (\mathbf{I} - \mathbf{B})^{-1} (r_U^B - \mathbf{F} A_L)$ and $B_L = (\mathbf{I} - \mathbf{B})^{-1} (r_L^B - \mathbf{F} A_U)$, where vectors $r_U^A, r_L^A, r_U^B, r_L^B$ contain external upper and lower traffic and service rates and $\mathbf{F} = \{F_{ij} \text{ iff flow } j \text{ enters task } i\}$. Since $x^0 = (A_U t, A_L t, B_U t, B_L t)$ constitutes a fixpoint for Γ under linear external constraints, monotonicity properties 9 of Γ implies that x^0 fulfills 11 and 12.

We have in this way established the uniqueness of a best fixpoint x^* and the existence of an iterative algorithm 8 for finding it.

5.3 Interpretation of Fix Point Solution

DNC establishes for some solution $x^* = (\alpha_U^*, \alpha_L^*, \beta_U^*, \beta_L^*)$ to 7 that $\alpha_L^* \preceq R' \preceq \alpha_U^*$ and $\beta_L^* \preceq S' \preceq \beta_U^* \Rightarrow \alpha_L^* \preceq R' \preceq \alpha_U^*$ and $\beta_L^* \preceq S' \preceq \beta_U^*$, which however is a tautology not sufficient to prove compliance of flows and services to the obtained constraints x^* .

To prove compliance w.r.t. a fixpoint x^* we shall impose the following assumption on upper and lower bounds

α_U^*, α_L^* associated to some flow R : for any flow R where $\alpha_L^* \preceq_T R \preceq_T \alpha_U^*$ for some non-negative time T there is a continuation $\alpha_L^* \preceq R^T \preceq \alpha_U^*$ defined for all $t \geq 0$, so that $R^T(t) = R(t)$ for all $0 \leq t \leq T$. And likewise for service constraints β_U^*, β_L^* . Such an assumption is fulfilled by e.g. convex (concave) constraints. We have however not so far been able to characterize exactly the loss of generality introduced in this way and leave that question for future research.

We introduce the imaginary network *shaper* elements $C_{\alpha_U^*, \alpha_L^*}$ for each flow in a given system, so that for $\tau = \sup\{T \geq 0 \mid \alpha_L^* \preceq_T R \preceq_T \alpha_U^*\}$ we define $R' = C_{\alpha_U^*, \alpha_L^*}(R) = R^\tau$. Then $\alpha_L^* \preceq R' \preceq \alpha_U^*$ and $\alpha_L^* \preceq R \preceq \alpha_U^* \Rightarrow R' = R$. $C_{\beta_U^*, \beta_L^*}$ is defined equivalently for services.

Any flow and service propagation is then imagined to happen through corresponding shapers. Since all inputs to service elements in this case are constrained according to x^* , so are outputs and in turn all inputs to shapers. Since inputs to shapers are constrained according to x^* , shapers have no effect on traffic flows and services. Therefore flows and services are provably constrained by x^* .

6 Example

Performance analysis is illustrated with a simplified abstraction of the information retrieval transaction described above as an example. As shown in figure 2 two tasks are

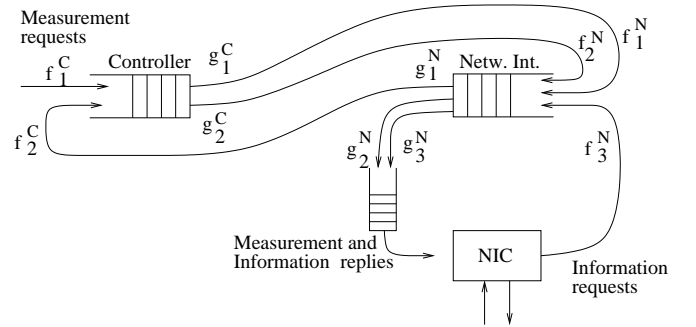


Figure 2. Simplified architecture for LAN accessible controller unit.

defined; *controller* (τ^C) and *network interface* (τ^N) along with corresponding message buffers. Two external flows are defined; *measurement requests* and *information requests* from the LAN. An independent network interface controller (NIC) with FIFO acts as message sink.

6.1 Network modelling

τ^C serves two input flows; the measurement flow f_1^C and a flow of information retrieval messages $f_2^C = g_1^N$ from τ^N .

(Notation f and g is used to distinguish input and output flows respectively. Subscripts denote flow indexing, whereas superscripts indicate task association.) Information retrieval messages flow through f_3^N to τ^N and are forwarded to τ^C through $f_2^C = g_1^N$. Reply messages from τ^C flow across $g_2^C = f_2^N$ to τ^N and produce a direct flow g_2^N to the NIC FIFO. Every 30th. measurement is sent to the network interface, through the flow $g_1^C = f_1^N$ to τ^N and finally by g_3^N to the NIC.

All together non-zero scaling factors include: $S_{1,1}^C = 1/30$, $S_{2,2}^C = 1$, $S_{1,3}^N = 1$, $S_{2,2}^N = 1$ and $S_{3,1}^N = 1$.

We assume in this example a non preemptive fixed priority scheduling discipline, where tasks are defined to be ready, whenever ingress buffers are nonempty. In the presented example it is assumed that τ^C has priority over τ^N . Locally tasks serve inflows from a common buffer in a FIFO discipline.

6.2 Numerical results

Numerical results are given for the example in figure 2. Message processing times are set to 5 generally. External flows are assumed to have affine constraints, i.e. $\alpha_U^{sensor} = 1/10t + 1$, $\alpha_L^{sensor} = 1/10t - 1$ and $\alpha_U^{network} = 1/40t + 5$, $\alpha_L^{network} = 1/40t - 5$ given in units of messages of length 5. Note that upper and lower rates are assumed equal. Summing over all rates gives a system utilization $5(1/10 + 1/40 + 1/10/30 + 2/40) = 0.9$, which may be characterized as heavy load.

A coarse solution through ADNC is provided by assuming affine internal traffic and service bounds and using only upper traffic and lower service bounds. More information on ADNC is found in [3] pp. 104. ADNC yields a solution (in units of CPU time): [105 280] for backlogs in τ_C and [31 396 191] in τ_N . Delay estimates for τ_C and τ_N on 200 and 1330 are obtained. Transaction delays are found by adding individual task delays along transaction paths. For *Information retrieval* a maximum delay of $1330 + 200 + 1330 = 2860$ is found. CyNC provides backlog estimates [5.5 5.5] and [12 65 63]. Delay estimates for τ_C and τ_N on 6 and 337 are obtained, which are all significant improvements to ADNC. Aggregating obtained upper bounds for flows to τ^N leads to an upper backlog bound in this buffer on 113. Transaction delay for *Information retrieval* is 680. Iterates of the fix point iteration 8 are shown in figure 3 illustrating how upper and lower bounds diverge monotonously from the initial linear estimate as predicted in section 5. A timed automata model of the presented example is constructed and UPPAAL [10] is used for simulation and model checking. The UPPAAL diagram of a token bucket filter modelling affine flow constraints is shown in figure 4. Simulation results show backlogs in the controller queue ≤ 2 and ≤ 4 messages in the network queue, which

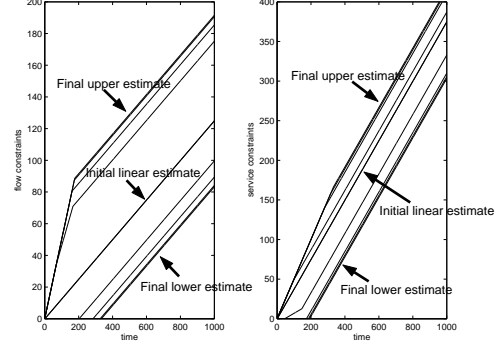


Figure 3. Iterates of fix point iteration for flow and service constraints.

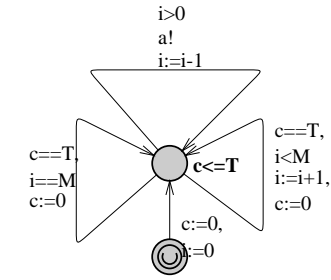


Figure 4. Timed Automata Model of Token Bucket Filter

may indicate that the obtained bounds are somewhat conservative. Model checking has been applied to timed automata model and exact backlog bounds 2 and 7 messages have been found for the controller queue and network queue respectively. The exact result indicates how extreme states are not likely to be revealed through simulation as well as the quality of the CyNC results. CyNC results for τ^C are seen to be tight, whereas for τ^N CyNC results are reasonable tight when comparing individual backlog results with the overall model checking bound. However the aggregated result is more than 3 times higher than for model checking. A possible reason is the lack of phase information in the CyNC model preventing it from accounting for the obvious phase relations between flows f_2^N and f_3^N in the presented example.

7 Complexity Issues

ADNC requires solving a system of an order matching the number of flows n defined in the system. Linear systems are solved in $n^3 + n^2$ floating points operations (flops).

In the presented example this amounts to 150 flops. CyNC requires numeric iteration in a space of constraint functions. It is believed that the required number of iterations for a sufficiently precise solution does not depend on the number of constraint functions but rather on the overall load of the analysed system. If so the computational complexity of the presented solution is linear in the number of constraint functions as well as the size of their numerical representation. For the presented example 10 iterations amounting to 1E6 flops are required. The implemented algorithms apply a naive table based representation of constraint functions and it is expected that possibilities for efficiency improvement are large.

Exact backlog bounds 2 and 7 have been found through checking existence of paths in the timed automata model leading to backlogs 2 and 3 for the controller queue and 7 and 8 for the network queue this requires traversing the entire reachability set. Thus exact bounds may potentially require exponential complexity. In the presented case the reachability set includes 1E6 states. We conclude that for the presented example CyNC and model checking are computationally comparable.

8 Conclusion

The application of network calculus (DNC) to the analysis and design of embedded systems has been suggested as well as several modifications of DNC needed for this application area.

A theoretical basis for the application of DNC for cyclic task dependencies and fairly general constraint assumptions is provided. The presented theory extends previous work of [3] and [5].

An example is provided for comparison between 3 different approaches; a coarse DNC based approach (ADNC) inspired by [3], the developed CyNC approach and model checking in a timed automata model. Model checking provides exact backlog bounds, whereas the ADNC yields rather conservative bounds for backlog and message delay. CyNC yields significantly less conservative bounds than ADNC. CyNC bounds are close to exact bounds except when aggregating flows with phase relations.

ADNC is by far the lightest from a complexity viewpoint. Complexities for CyNC and model checking are comparable for the presented example. This balance is however expected to point in favour of CyNC for larger system models, since CyNC is believed to be polynomial whereas model checking is exponential.

CyNC is illustrated by an example assuming fixed priority and FIFO scheduling disciplines, which despite their popularity in industrial designs are only two among a number of applied principles like *Round Robin*, *Earliest Deadline First* and *Weighted Fair Queuing*. Generalization of the

presented approach to other scheduling disciplines is postponed to future work.

References

- [1] C. L. Liu and J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment., *JACM*, pp.46-61,1973.
- [2] J. P. Lehoczky, L. Sha and Y. Ding, The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour, *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pp. 166-171, 1989.
- [3] J. L. Boudec and P. Thiran *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet.*, Springer-Verlag, 2000.
- [4] R. L. Cruz, *A Calculus for Network Delay Part I: Network Elements in Isolation*, IEEE trans. on Information, 1991.
- [5] Samarjit Chakraborty, Simon Kuenzli, Lothar Thiele *A General Framework for Analysing System Properties in Platform-Based Embedded System Designs*. Design Automation and Test in Europe (DATE), IEEE Press, Munich, Germany, pages 190-195, March, 2003
- [6] Marek Jersak and Rolf Ernst, *Enabling Scheduling Analysis of Heterogeneous Systems with Multi-Rate Data Dependencies and Rate Intervals*. 40th Design Automation Conference (DAC), Anaheim, USA, June, 2003
- [7] Marek Jersak, Kai Richter and Rolf Ernst, *Performance Analysis for Complex Embedded Applications*, (to appear in) the International Journal of Embedded Systems, Interscience Publishers, 2004
- [8] Larsen, Kim G. and Steffen, Bernhard and Weise, Carsten, *Continuous Modelling of Real Time and Hybrid Systems: From Concepts to Tools*, International Journal on Software Tools for Technology Transfer, pp. 64-85, Vol. 1, No. 1, 1997.
- [9] R. Alur and D.L. Dill, *A Theory of Timed Automata*, Theoretical Computer Science, pp. 183-235, 1994.
- [10] K. G. Larsen, W. P. Pettersson and W. Li, *UPPAAL in a Nutshell*, Int. Journal on Software Tools for Technology Transfer, Vol. 1, 1997.
- [11] H. Gomma, *Software Design Methods for Concurrent and Real-Time Systems.*, Addison Wesley, 1993.