ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUCPL06

VERIFICATION OF THE FAIR CONTROL SYSTEM USING DETERMINISTIC NETWORK CALCULUS

M. Schütze*, S. Bondorf†, DISCO Lab, University of Kaiserslautern, Kaiserslautern, Germany
M. Kreider‡, GSI, Darmstadt, Germany; Glyndŵr University, Wrexham, Wales

Abstract

The FAIR control system (CS) is an alarm-based design and employs White Rabbit time synchronization over a GbE network to issue commands executed accurate to 1 ns. In such a network based CS, graphs of possible machine command sequences are specified in advance by physics frameworks. The actual traffic pattern, however, is determined at runtime, depending on interlocks and beam requests from experiments and accelerators. In 'unlucky' combinations, large packet bursts can delay commands beyond their deadline, potentially causing emergency shutdowns. Thus, prior verification if any possible combination of given command sequences can be delivered on time is vital to guarantee deterministic behavior of the CS. Deterministic network calculus (DNC) can derive upper bounds on message delivery latencies. This paper presents an approach for calculating worst-case descriptors of runtime traffic patterns. These so-called arrival curves are deduced from specified partial traffic sequences and are used to calculate end-to-end traffic properties. With the arrival curves and a DNC model of the FAIR CS network, a worst-case latency for specific packet flows or the whole CS can be obtained.

INTRODUCTION

Non-functional aspects of large distributed systems often define the most safety-critical properties of such systems. For instance, the avionics sector employs x-by-wire applications with strict reliability and safety requirements [1]. Another area where formal verification is an important part of the development and operation are industrial facilities [2, 3]. Their uninterrupted operation is subject to fulfilment of predefined performance metrics, foremost w.r.t. to their control system (CS). This also holds true for the *GSI Helmholtz Centre for Heavy Ion Research*, a particle accelerator facility in Darmstadt, Germany. Its largest component, the *Facility for Antiproton and Ion Research (FAIR)*, will contain 3.5 kilometres of piping [4], several kilometres of cabling and more than 2000 endpoints [5]. Part of the development of FAIR has been the introduction of a highly accurate, low-latency CS system employing *White Rabbit* [6] time synchronization over Gigabit Ethernet [7]. The FAIR CS demands that, at any given time of operation, messages sent to any end point can reach their destination within 500 microseconds despite the presence of further messages in the network. While viable machine command sequences are specified in advance by physics frameworks, they might translate into a vast number of different traffic patterns. These are only determined at runtime and depend on interlocks and beam requests from experiments and accelerators. In the worst case, this might result in messages being delayed beyond their respective deadline, potentially causing emergency shutdowns of the entire system. To verify that the given timing constraint invariantly holds, all possible message sequences for an experiment need to be verified. For this, traffic specifications in a graph-based format as shown in Figure 1 were introduced.

The Deterministic Network Calculus (DNC) [8] has been used to verify deadlines in distributed avionics systems for quite some time [9]. DNC is an application-agnostic mathematical framework for worst-case modelling and analysis of distributed systems. More recently, it has also been applied to large industrial systems [10]. In this paper, we will provide its application to the FAIR CS. Among DNC's two parts, modelling and analysis, the latter has seen most attention. The literature set its focus on improving the accuracy of worst-case bounds on the end-to-end delay as well as computational aspects of deriving them (see [11] for recent and comprehensive results). In contrast, we extend the system modelling capabilities of DNC. To be precise, we contribute a method to convert the graph-based specification of possible machine command sequences to the deterministic upper bound on traffic flows at the location they enter the system. The DNC analysis takes the network topology, forwarding capabilities of the network, flows' path and these bounds, so-called arrival curves, to bound worst-case message delays. Arrival curves have been derived from input specifications before, however, such approaches often required explicit generation of compliant traffic traces that were then transformed into the arrival curves [12]. As these traces are finite, the domain of the resulting curves is finite, too. This may impact the validity of derived performance bounds and needs to be handled with care [13]. We avoid these problems by directly deriving arrival curves from the specification such that these are valid for indefinite length of operation of the system.

The remainder of this paper is structured as follows: A background section presents the basics on DNC, the FAIR traffic specification and a formalization that we use for derivation of arrival curves. The derivation is contributed in the subsequent section. We aim for most accurate arrival curves, resulting in pseudo-periodic shapes. Yet, tools providing automated DNC analysis, foremost the DiscoDNC [14, 15], may be restricted to aperiodic arrival curves. Thus, we provide a concave hull algorithm to convert to arrival curves with a finite amount of piecewise affine segments. We present practical considerations concerning our generic algorithm and parameter ranges found in a FAIR traffic spec-

* m_schuetze13@cs.uni-kl.de
† bondorf@cs.uni-kl.de, supported by the Carl Zeiss Foundation
‡ m.kreider@gsi.de

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUCPL06

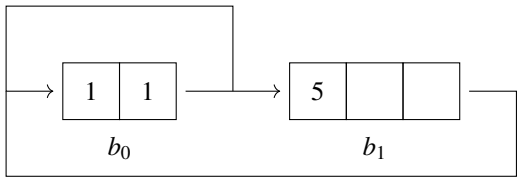


Figure 1: An example of a specification in graphical format. We assume that edges always leave blocks on the right and enter on the left.

ification before we evaluate our findings and conclude the paper.

BACKGROUND

Deterministic Network Calculus

DNC is a mathematical framework for deriving upper bounds on the end-to-end delay of messages in networks. Given a cumulative function describing the traffic entering the system at any given time, $A : \mathbb{R}_+ \to \mathbb{R}_+$, the arrival curve α describes an upper limit on the traffic entering the system in any given interval: $\alpha(s) \geq A(t+s) - A(t)$ for all $t \geq 0$. Combined with a service curve β, which is a lower bound on the forwarding capabilities of the system, this allows derivation of upper bounds on the end-to-end delay and backlog of messages [8].

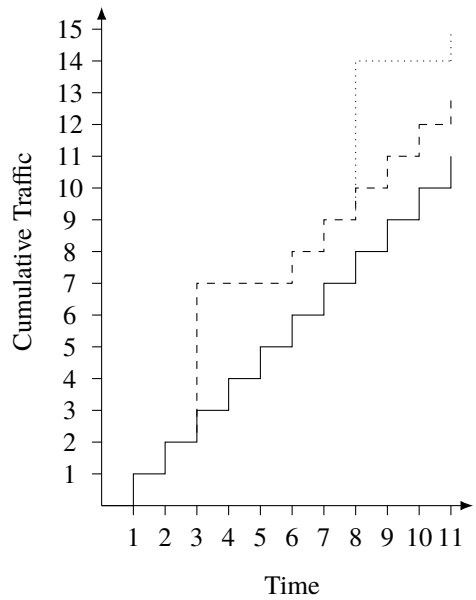
The FAIR Traffic Specification

The FAIR traffic specification represents several possible machine command sequences in a graph-based format. Each node in the graph represents a deterministic sequence of messages of some fixed length. Messages inside a block have a size, representing an abstract measure of size (for example the number of bytes) and an offset from the beginning of the block. Therefore, if a block is entered at time t, a message with offset τ is considered to be sent at time $t + \tau$. Machine command sequences start at the beginning of some predefined initial block. At the end of each block, several possible next blocks might exist as indicated by edges with multiple destinations in the graphical representation (see Figure 1). The choice is only made at runtime of the system. Overall the actually emitted command sequence can therefore be considered non-deterministic. For example, see the several possible cumulative models of Figure 2a. In this paper, we contribute a deterministic upper bound on the traffic, a DNC arrival curve α.

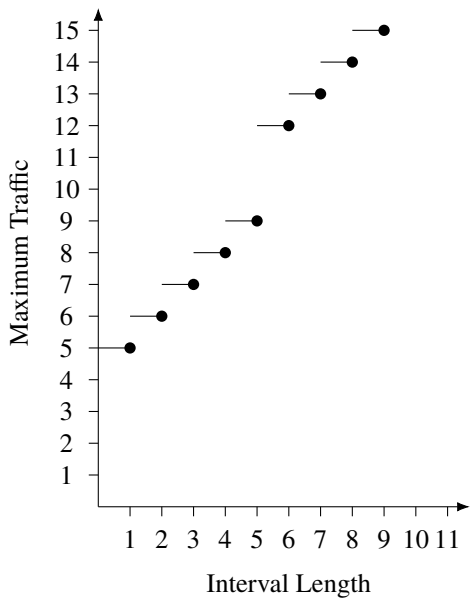
Problem Formalization

Specifications We model specifications as a directed graph (B, E), where B is the set of blocks of the specification and E is the successor relationship between blocks.

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



(a) Potential cumulative models for Figure 1.



(b) Arrival curve of Figure 1.

Figure 2: Curves derived from Figure 1.

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

$$\text{Duration } \delta\colon B \to \mathbb{N}_+$$
$$\text{Messages } \mu\colon B \to \mathcal{P}(M)$$

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

$$\text{size}\colon M \to \mathbb{N}_0$$
$$\text{offset}\colon M \to \mathbb{N}_0$$

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUCPL06

We define the traffic generated by some block b as

$$\tau(b) := \sum_{m \in \mu(b)} \text{size}(m).$$

Flows In order to capture the worst-case traffic generation deterministically, we extend our concept of blocks to arbitrary flows between blocks. The central part of a flow is uniquely characterized by the messages it contains[1], but we might need padding at the front and back when a flow starts or ends between two messages instead of borders of blocks. We will call the padding at the front of the flow pad_l and the padding at the end pad_r.

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

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

Note that this definition does not allow us to represent flows that end between two messages m, m' if $\text{offset}(m) = \text{offset}(m')$. However, this is not a crucial restriction since we are generally interested in the maximum traffic generated in any given interval. This worst case will not be achieved by excluding some concurrent messages from the flow.

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

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

$$\tau(f) := \sum_{m \in \sigma} \text{size}(m)$$

and the duration of f as

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

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

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

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

and the flows with the maximum traffic are

$$\text{flows}_\alpha(n) := \arg\max_{\delta(f) \leq n} \tau(f).$$

[1] Unless it passes through no messages at all – but in this case, all such flows have the same behaviour. They are also typically not of interest when trying to approximate worst-case traffic.

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

DERIVING ARRIVAL CURVES FROM MACHINE SEQUENCE SPECIFICATIONS

Pseudo-periodic Input Approximations

Given the restrictions of the DiscoDNC, we need to derive a concave hull of the arrival curve. This concave hull is particularly easy to calculate for ultimately pseudo-periodic functions. Let $d \in \mathbb{R}_+$ be the period, $c \in \mathbb{N}_0$ be the increment and $T \in \mathbb{N}_0$ be the offset of the periodic part. We call g an ultimately pseudo-periodic function [16] if

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

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

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

We pick some threshold k and define α_apx to be an ultimately pseudo-periodic function with initial offset 0, period k and increment $\alpha(k)$ with $\alpha_\text{apx}(i) = \alpha(i)$ for $i < k$.

Ultimately Affine Concave Hull

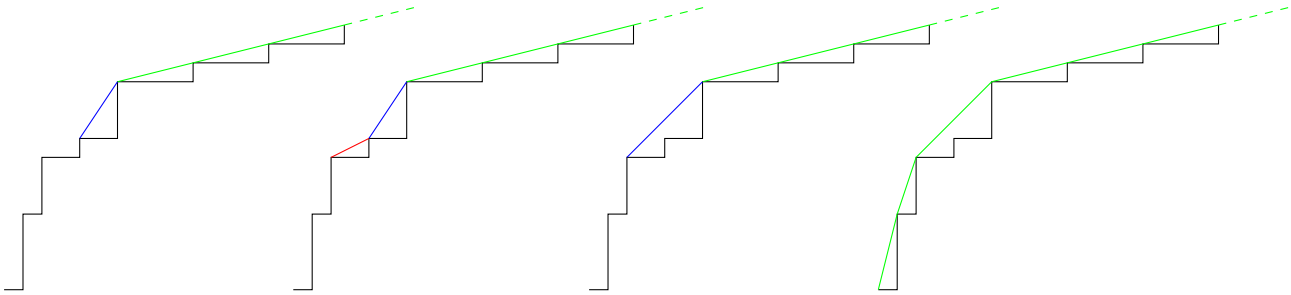
Based on the ultimately pseudo-periodic approximation of the arrival curve we next derive an arrival curve that is compliant with the DiscoDNC tool [14, 15]. The DiscoDNC demands concave arrival curves that are composed of a finite number of linear segments. We therefore need to further process our function. This step is based on the fact that every curve that is larger than the given arrival curve is also a valid arrival curve for the system (although model accuracy is decreased by over-approximation).

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

Let $h_1, \ldots h_n$ be the segments of the concave hull with intervals $[x_0, x_1), [x_1, x_2), \ldots [x_{n-1}, \infty)$. By concavity we know that $\text{slope}(h_i) \geq \text{slope}(h_{i+1})$. The last affine segment of the concave hull h_f has a slope of $\frac{c}{d}$, an a y-offset of $y_0 = \max \{\ f(x) - x \cdot \frac{c}{d} \mid x < T + d\ \}$ and begins at $x_{n-1} = \min \left\{ \arg\max \left\{\ f(x) - x \cdot \frac{c}{d} \mid x \leq T + d\ \right\} \right\}$.

If $h_f = (h_i)_{i \leq n}$ is a tightest concave hull over f, then $h_i(x_i) = f(x_i)$. From this, we develop Algorithm 1 (Consider also Figure 3 for a visualization of the steps of the algorithm).

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUCPL06



(a) Initial step. (b) Monotonicity violated. (c) Skipping a segment. (d) Final hull.

Figure 3: Concave hull algorithm.

Algorithm 1 Concave Hull for Pseudo-Periodic Functions

```
1: function ConcaveHull(seg_end, min_slope) → segment offsets
2: # Compare Figure 3a
3:     seg_begin = seg_end − 1
4:     if seg_begin = 0 then
5:         return [0]
6:     end if
7:     max_slope = ∞
8: # Abort when no valid segments left
9:     while slope(seg_begin, seg_end) ≥ min_slope do
10:        slope = slope(seg_begin, seg_end)
11:        if slope ≤ max_slope then # Check interception
12:            hull = ConcaveHull(seg_begin, slope)
13:            if hull ≠ "concave hull not found" then
14: # Found a valid hull, compare Figure 3d
15:                return hull + [seg_begin]
16:            end if
17: # Cannot have higher slope than this for the current segment without intercepting at seg_begin
18:            max_slope = slope
19:        end if
20: # Consider the next segment, compare Figure 3c
21:        seg_begin = seg_begin − 1
22:    end while
23: # Compare Figure 3b
24:    return "concave hull not found"
25: end function
```

PRACTICAL CONSIDERATIONS

Realistic machine command sequences for the FAIR CS reveal that brute-forcing the calculations will cause infeasible computational effort. Typical values for the length of blocks for the GSI control system are in the range of 10^9 nanoseconds. However there are usually only 10^2 to 10^3 messages in a block. We can therefore cache the times at which the curve increments and the corresponding value at these points in time.

Listing 1: Step Function

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

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

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

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

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

We now extend present an algorithm to compute the maximum prefix in a single block that makes use of these optimizations.

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

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

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUCPL06

Algorithm 2 Calculating MAXPREFIX

```
1:  function BLOCK::FIRSTTIMEEXCEEDING(traffic) → time
2:      while maxprefix.maxValue ≤ traffic do # Find the
        next step of the function
3:          ttr = maxprefix.maxValue − totalBlockTraffic
4:          min = min_next b.FIRSTTIMEEXCEEDING(ttr)
5:          generatedTraffic = max_next b.MAXPREFIX(min)
6:
7:          time = period + min
8:          value = totalBlockTraffic + generatedTraffic
9:          ADD STEP TO maxprefix AT TIME time TO VALUE
        value
10:     end while
11:     return maxprefix.FIRSTTIMEEXCEEDING(traffic)
12: end function
13:
14: function BLOCK::MAXPREFIX(time) → traffic
15:     while maxprefix.validTo IS NOT DEFINED UP TO time
        do
16:         FIRSTTIMEEXCEEDING(maxprefix.maxValue)
17:     end while
18:     return maxprefix.VALUEAT(time)
19: end function
```

Each call of FIRSTTIMEEXCEEDING results in up to $|B|$ calls of FIRSTTIMEEXCEEDING on other blocks. Each of these may result in another $|B|$ calls of FIRSTTIMEEXCEEDING. These calls however will return immediately in $\log S$, resulting in a runtime of $O(|B|^2 \cdot \log S)$:

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

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

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

EVALUATION

We now evaluate the feasibility of our proposed approach. For this, we consider two dimensions: How accurate the algorithm is, and how fast it terminates, in both cases with respect to a given evaluation threshold. We leave dimensions that are potentially of interest, for instance density of messages, the number of blocks in the specification or the average branching factor of blocks, for future work.

Our evaluation is based on a fictional yet representative specification for the FAIR CS and its CRYRING component, a heavy ion storage ring (see Figure 4 and [17]). For it, the sequence with the highest average traffic emits `B_CRY_HALT` 1 message per $5 \cdot 10^5$ ns, and the longest block has a length of $2.75 \cdot 10^9$ ns. The shown error is calculated based on the approximated slope of the final segment and compared to the "true" slope of the final segment ($2 \cdot 10^{-6}$).

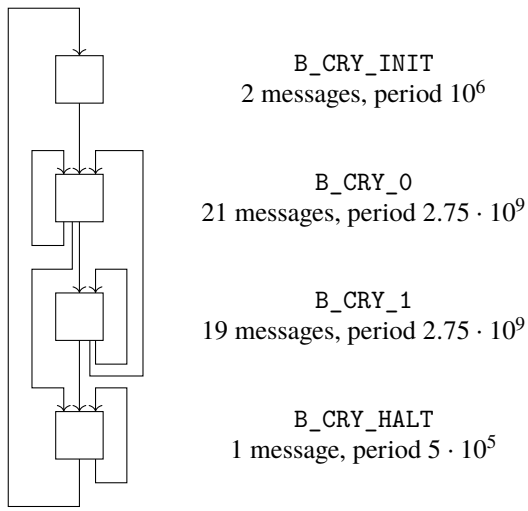


Figure 4: The CRYRING specification.

Accuracy

Table 1 and Figure 5 show that our sub-additive approximation lets the error quickly approach zero.

Table 1: Accuracy

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

Error (abs.) derived slope minus ground truth
Error (rel.) calculated slope divided by ground truth

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUCPL06

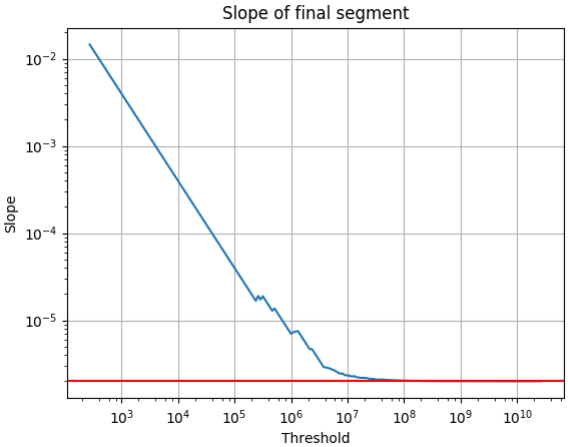


Figure 5: Accuracy of sub-additive approximation with respect to thresholds. Red line is "true" slope given by the concave hull over α at $2 \cdot 10^{-6}$.

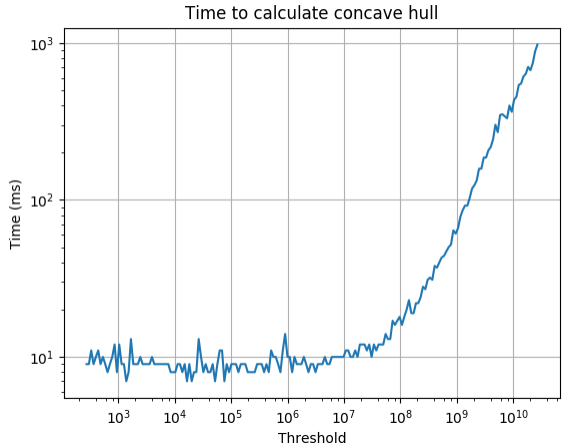


Figure 6: Runtime of sub-additive approximation with respect to thresholds.

However, the error will never reach zero as the sub-additive approximation pays bursts for each multiple of the threshold instead of only once.

Execution Time

The following tests were run on a Windows 10 computer with a quad-core Intel i7-4710HQ CPU. The processor has a clock speed of 2.5GHz and made use of 16GB RAM running at 1600MHz. While the presented results are not meant to be authoritative, they provide insight on the speed of our algorithm. Results are shown in Table 2 and Figure 6.

Table 2: Execution Time of Sub-additive Approximation for Different Thresholds, Average over 10 Repetitions

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

The (optimized) sub-additive approximation can be computed very fast, returning very accurate results within a couple of milliseconds. While the worst-case analysis of our algorithm's computational effort show that it might suffer from quadratic scaling, we observed fast evaluation times for realistic specifications for machine command sequences. In our example, execution time increases roughly linear with the threshold though at low terms a fixed overhead dominates.

CONCLUSION

In this paper, we presented an approach to deterministically bound the traffic generated by the FAIR control system. We developed an algorithm to convert the graph-based representation of all attainable machine command sequences possibly emitted during runtime as this representation entails non-determinism. Giving a worst-case bound on the generated traffic in terms of a Deterministic Network Calculus arrival curve, we enable worst-case analysis of the entire CS in a timing verification step prior to system runtime. Results of this step, namely bounds on the end-to-end delay of message transmissions, can be compared to command deadlines and thus help to avoid potential emergency shutdowns of the system.

REFERENCES

[1] F. Geyer and G. Carle, "Network engineering for real-time networks: Comparison of automotive and aeronautic industries approaches," *IEEE Communications Magazine*, Feb. 2016.

[2] J. Yoo, S. Cha, and E. Jee, "A verification framework for fbd based software in nuclear power plants," in *Proc. of the 15th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2008.

[3] C. J. Tomlin, I. Mitchell, A. M. Bayen, and M. Oishi, "Computational techniques for the verification of hybrid systems," *Proceedings of the IEEE*, vol. 91, no. 7, Jul. 2003.

[4] *How FAIR is being built*. `http://www.fair-center.eu/construction/how-fair-is-being-built.html`

[5] R. Huhmann *et al.*, "The Fair Control System - System Architecture and First Implementations," in *Proc. of the 14th International Conference on Accelerator & Large Experimental Physics Control Systems (ICALEPCS)*, Oct. 2013.

[6] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, and G. Gaderer, "White rabbit: Sub-nanosecond timing distribution over ethernet," in *Proc. of the International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS)*, Oct. 2009.

[7] M. Kreider, "On Time, in Style: Nanosecond Accuracy in Network Control Systems," PhD thesis, Glyndŵr University, Wrexham, Wales, Aug. 2017.

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

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUCPL06

[9] J. Grieu, "Analyse et évaluation de techniques de commutation ethernet pour l'interconnexion des systèmes avioniques," PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, France, Sep. 2004.

[10] X. Jin, N. Guan, J. Wang, and P. Zeng, "Analyzing multi-mode wireless sensor networks using the network calculus," *Journal of Sensors*, vol. 2015, Feb. 2015.

[11] S. Bondorf, P. Nikolaus, and J. Schmitt, "Quality and Cost of Deterministic Network Calculus – Design and Evaluation of an Accurate and Fast Analysis," in *Proc. of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Jun. 2017.

[12] S. Künzli and L. Thiele, "Generating event traces based on arrival curves," in *Proc. of the 13th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, Mar. 2006.

[13] K. Lampka, S. Bondorf, and J. Schmitt, "Achieving efficiency without sacrificing model accuracy: Network calculus on compact domains," in *Proc. of IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sep. 2016.

[14] J. Schmitt and F. A. Zdarsky, "The DISCO Network Calculator - A Toolbox for Worst Case Analysis," in *Proc. of the 1st ICST International Conference on Performance Evaluation Methodologies and Tools (ValueTools)*, ACM, Nov. 2006.

[15] S. Bondorf and J. Schmitt, "The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus," in *Proc. of the 8th EAI International Conference on Performance Evaluation Methodologies and Tools (ValueTools)*, Dec. 2014.

[16] A. Bouillard and É. Thierry, "An algorithmic toolbox for network calculus," *Discrete Event Dynamic Systems*, vol. 18, no. 1, Mar. 2008.

[17] M. Schütze, *Modelling and Analysis of Timing Constraints of an Industrial Control System*, Bachelor thesis, University of Kaiserslautern, Kaiserslautern, Germany, Oct. 2017.