What is Deterministic Network Calculus?

Tobias Wasner, Max Helm*, Dominik Scholz*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: mail@wasnertobias.me, helm@net.in.tum.de, scholzd@net.in.tum.de

***Abstract*—This paper gives a short introduction to Deterministic Network Calculus. Explains how service guarantees can be provided to error-free packet-switching networks and the limitations of this process. Discusses alternative mathematical approaches, their limitations and corresponding use cases.**

***Index Terms*—deterministic network calculus, worst-case performance, guaranteed service, packet-switching network, traffic shaping, schedulability**

1. Introduction

As shown by Cruz in [1] and [2], Deterministic Network Calculus (DNC) is used to calculate theoretical worst-case performance guarantees for error-free packet-switching networks of queues and schedulers.

DNC, other than Stochastic Network Calculus (SNC), does not use probabilistic terms to characterize performance guarantees. This means that performance guarantees calculated using DNC will hold in any case. This is why DNC is especially useful for hard real-time systems, as those systems assume that missing a single deadline results in a total system failure. [1]

In section 2 the required mathematical background for the application of DNC will be introduced.
In section 3 the application of DNC will be explained.
In section 4 examples for limitations of the application of DNC will be given.
In section 5 alternative frameworks will be briefly introduced and their limitations will be compared to those of DNC.
In section 6 the main results of this paper will be summarized.

2. Background

In order to be able to calculate performance bounds, certain mathematical models have to be applied to the network, which will be described in this section.

2.1. Flows

A comprehensive introduction of the term *flow* is shown by Boudec et al. in [3]. The notation of this section of the paper is derived from Geyer in [4]. The term *flow* is used to describe a unidirectional set of packets which are being sent from a single sender to a single receiver in a packet-switching network [4]. Cumulative arrival functions are used to mathematically model flows. They need to be a member of the following set: [4]

$$F = \{f : \mathbb{R}^+ \to \mathbb{R}^+ \mid \forall 0 \leq t \leq s : f(t) \leq f(s), f(0) = 0\}$$

In the formular s is the time of the end of the flow. This implies that every $A \in F$ is a non-decreasing strictly positive function. A represents the amount of data being sent by the flow in the time interval $[0, t)$.

Furthermore, a flow has a deterministic arrival curve $\alpha \in F$ if its cumulative arrival function A satisfies: [4]

$$\forall 0 \leq s \leq t : A(t) - A(s) \leq \alpha(t - s)$$

In the formular s is the time of the beginning of the constraint and t is the the time of the end of the flow. It is said that the flow A is constrained by α in that case [3]. The cumulative arrival function A can also be used to describe the sending rate as DNC assumes that no packet loss happens.

2.2. Traffic shaping

As shown by Tanenbaum in [5], traffic shaping is needed to constrain flows. Constraining of flows is the basis of all calculations of DNC. Traffic shaping is a technique which smooths out the traffic on the senders' side, rather than on the receivers' side [5]. Two different traffic shaping algorithms will be explained in the following.

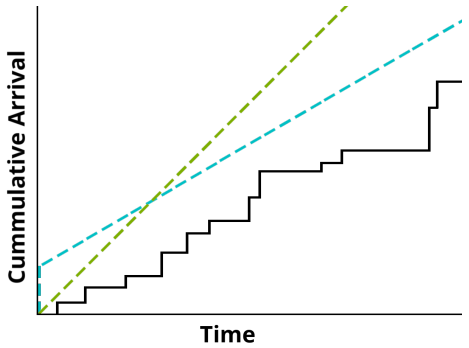

Figure 1: Cumulative byte arrival (solid black line) constrained by a token bucket arrival curve (dashed blue line) and a leaky bucket arrival curve (dashed green line).

doi: 10.2313/NET-2019-10-1_12

2.2.1. Leaky bucket algorithm [5]. This traffic shaping algorithm derives its name from the idea of a bucket being constantly filled with an irregular rate of water (water abstracts packets to be sent over the network) and with one little hole at the bottom of the bucket (packets actually being sent over the network).
The filling of the buffer, therefore, is expressed as the filling of the bucket. The outflow through the little hole is happening with a constant rate r, at least if the bucket is not empty. Once the bucket is fully filled, any further input will simply spill over and is therefore lost.
The rate r can either have the unit number of packets per time frame or data size per time frame. The second approach is useful in the case where not all packets have a fixed data size.

2.2.2. Token bucket algorithm [5]. The content of this bucket are tokens. A token is the allowance to send data in the form of a certain number of packets or bytes. If data has been sent over the network, the number of tokens in the bucket is reduced accordingly. Tokens are added to the bucket at a constant rate r, however, if the bucket is full, no more tokens can be added. If the bucket is empty, data cannot be sent out to the network, it is necessary to wait for tokens to be added in this case.

This algorithm is more flexible in comparison to the leaky bucket algorithm as the output rate is not fixed and the token bucket is fully filled at the time of the beginning of the constraint. Therefore this algorithm allows bigger bursts to happen in comparision with the leaky bucket algorithm. This can also be seen in Figure 1, where different minimum rates r (slopes) are required to constrain the given flow, due to the offset of the token bucket algorithm.

2.3. Servers and service curves

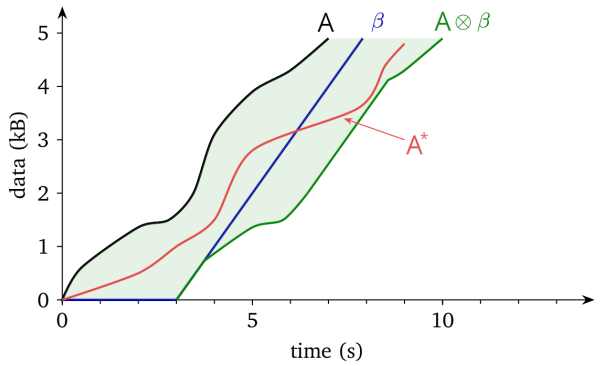

Figure 2: Visualization of the service curve concept [6].

Every single node in a network has a queue and a scheduler, which is an algorithm which decides which packet will be sent next, in case there are multiple packets in the queue waiting to be transferred. The term *server* is used to describe a whole network or certain parts of a network, e.g. a link, a scheduler or a traffic shaper. [4]

Given a deterministic arrival curve A, a server is characterized by its deterministic service curve β, such that the output curve A^* of a flow after traversing the server is defined as: [4]

$$A^*(t) \geq \inf_{t \geq s \geq 0} \{A(s) + \beta(t-s)\} = A \otimes \beta$$

This definition is illustrated by Bemten et al. in [6], correspondingly Figure 2.
As the departure of some data cannot occur before its arrival it is implied that $\forall t \geq 0 : A(t) \geq A^*(t)$.

3. Application

In this section the application of DNC will be explained.

3.1. Service guarantees

DNC provides two different kinds of service guarantees, namely delay bounds and backlog bounds. Both will be introduced in the following.

3.1.1. Delay bounds. The virtual delay d at time t is defined by the following equation: [3]

$$d(t) = \inf_{\tau \geq 0} \{A(t) \leq A^*(t+\tau)\}$$

The term delay bound corresponds to the maximum time that incoming data has to wait before being processed by the server. In mathematical terms this can be expressed using the following equation: [4]

$$A^*(t) - A(t-s) \leq \sup_{t \geq 0} \{d(t)\}$$

3.1.2. Backlog bounds. The backlog b at time t is defined by the following equation: [3]

$$b(t) = A(t) - A^*(t)$$

The term backlog bound corresponds to the maximum amount of data that will have to wait before being processed by the server. In mathematical terms this can be expressed using the following equation: [4]

$$A(t) - A^*(t) \leq \sup_{t \geq 0} \{b(t)\}$$

3.2. Generalized Processor Sharing

Generalized Processor Sharing (GPS) is the ideal form of per flow queuing. Per flow queuing provides isolation of flows and therefore service guarantees differentiated per flow. Numerous practical implementations of GPS have been proposed in the literature. Each differs in their provided service guarantees and their implementation complexity. Practical implementations of GPS will be introduced in the following. [3]

3.2.1. Practical Generalized Processor Sharing. Practical Generalized Processor Sharing (PGPS) implements GPS using one First In First Out (FIFO) queue per flow [3]. Each queue is assigned a priority [4]. Based on the assigned priority the available bandwidth is shared accordingly [4].

doi: 10.2313/NET-2019-10-1_12

3.2.2. Guaranteed Rate Schedulers. All practical implementations of GPS fit in the framework Guaranteed Rate Schedulers (GRS). This approach considers a server with FIFO-scheduling and a constant bit rate r. [3]
Furthermore, T_i is defined as the arrival time, T_i' as the departure time and l_i as the length in bits of the ith packet, ordered by arrival time [3].
Assuming $T_1 \geq 0$, where T_1 is the arrival time of the packet which arrived earliest, the definition of T_i' is the following: [3]

$$T_i' = \begin{cases} 0 & \text{if } i = 0 \\ max\{T_i, T_{i-1}'\} + \frac{l_i}{r} & \text{if i > 0} \end{cases}$$

This means that packet i starts its service at $max\{T_i, T_{i-1}'\}$ and ends at $max\{T_i, T_{i-1}'\} + \frac{l_i}{r}$ [3].

3.3. Schedulability

Networks are not usually built in the way that a single piece of hardware is being exclusively used by one single flow. Quite the opposite, it is extremely common that a single node in a network has to handle transfers for multiple flows in parallel. Schedulability enables that service guarantees can still be made in that case. This is done in the following way: [3]

When a node is affected by a new flow it has to reserve two kinds of resources locally: bandwidth and buffer size. In order to be able to reserve those resources the quantity has to be determined. To calculate the needed bandwidth and buffer size we have to keep the service curve and the arrival curve constraints of the flow in mind. The most general framework which is making those calculations possible is named Service Curve Earliest Deadline First. [3]

3.3.1. Earliest Deadline First. The concept of Earliest Deadline First (EDF) schedulers assumes that there is a list for every corresponding flow which contains the arrived packets which are waiting to be transferred further. Furthermore, a deadline D_i^n is allocated to every nth packet of every ith flow. [3]
At every time slot the scheduler picks one packet with the earliest deadline out of all packets independent of the flow. This general concept contains no further assumption of how the deadline is mathematically allocated. For that reason multiple concepts of other scheduler types - e.g. FIFO - can be fit in this concept as well. [3]

3.3.2. Service Curve Earliest Deadline First Schedulers. As shown in subsection 3.3.1 EDF schedulers assume that there is a deadline allocated to every packet. Service Curve Earliest Deadline First (SCEDF) schedulers allocate the deadlines for every packet in that way that every ith flow does have β_i as a service curve. [3]

3.4. Time Analysis Methods

It is explained in section 3.1 how service guarantees can be calculated for individual servers. The Time Analysis Model enables to calculate delay bounds for flows which traverse multiple servers.

3.4.1. Total Flow Analysis. As shown by Heidinger in [7], this method of calculating delay bounds is done in the following way: The delay bounds are calculated per traversing node and then added up. For that reason this method is also known as *node-by-node analysis*.
The problem with this method is that the delay bounds will be calculated overly pessimistic, because bursts are not only paid at the first traversing node, but at every traversing node. [7]

3.4.2. Separate Flow Analysis. This method of calculating delay bounds is done in the following way: The service curves are calculated per traversing edge, added up and then the horizontal deviation is used as a delay bound. [7]
In that way bursts will be only paid at the first traversing node. The problem with this method is that the delay bounds will be calculated overly pessimistic, if a flow is multiplexed several times. [7]

3.4.3. Pay Multiplexing Only Once. As shown by Schmitt et al. in [8], with this method overly pessimistic calculations of delay bounds do not happen, if a flow is multiplexed several times. This method is based on separate flow analysis as shown in section 3.4.2.

4. Limitations

In this section the limitations of the application of DNC will be explained.

4.1. Cyclic dependencies

As shown by Schiøler et al. in [9], cyclic dependencies in dataflows can not be modeled using DNC without any modifications. This problem is still under active research and there is already an approach named Cyclic Network Calculus (CyNC) which aims to support that use case, but has not yet produced correct results in every case [9].
CyNC is based on DNC but extends the theoretical basis. Several modifications of DNC are already proposed and even more are needed in the future to fully support this application area. [9]

4.2. Feedback loops

One might think that protocols which include feedback loops - e.g. TCP - cannot be modeled using DNC. In fact, this assumption has been proven wrong, as Baccelli et al. have shown in [10] that TCP can be modeled using max-plus algebra. However this is not a common use case for DNC, because the used traffic shaping in DNC is about regulating the average rate and burstiness of data transmission whereas the sliding window protocols, such as TCP, only limit the amount of data in transit at once [5].

4.3. Overprovisioning

DNC aims to determine the actual worst case which can be seen in Figure 3. Practically the calculation of overly pessimistic upper bounds can be observed frequently using DNC as shown by Fidler in [11]. However,

doi: 10.2313/NET-2019-10-1_12

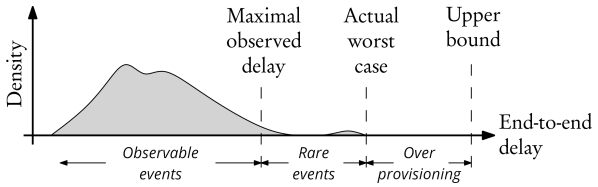

Figure 3: Visualization and naming of delay bounds. [4]

determining the exact actual worst-case is NP-hard [4].
For systems which are not of the type hard real-time - e.g. best-effort or soft real-time systems - those overly pessimistic calculations of the actual worst case can easily make the calculations of DNC worthless. Even if the actual worst case is not calculated overly pessimistic it still may be attained rarely and may not even have the effect of breaking the whole system as silently assumed by DNC. [11]

5. Alternative approaches

In this section alternative approaches to calculate performance guarantees will be briefly introduced and their limitations will be compared to those of DNC.
Figure 4 provides a broad overview of the different use cases of several alternative approaches. Some approaches will be explained further in the following subsections.

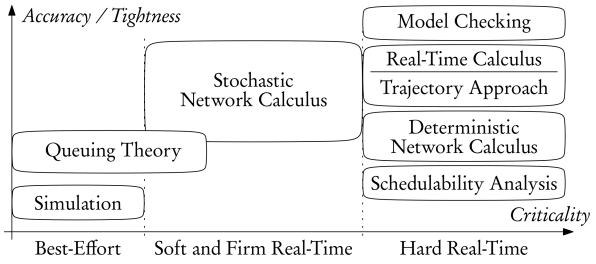

Figure 4: An overview of the use cases of several alternative approaches [4].

5.1. Stochastic Network Calculus

As described in section 4.3, DNC is not designed to model systems which are not of the hard real-time type. SNC characterizes performance bounds in probabilistic terms [1], which makes it more suitable to model firm or soft real-time systems [4].

5.2. Queuing theory

As shown by Lipsky in [12], this approach uses probabilistic terms to characterize performance bounds, as well as SNC. Furthermore, it does not analyze the worst-case, as DNC or SNC does, but the average-case [12], which makes it more suitable to model best-effort systems [4].

6. Conclusion and future work

In this paper we have shown how Deterministic Network Calculus can be used to calculate theoretical worst-case performance guarantees for error-free packet-switching networks of queues and schedulers.

In section 2, we started to define the term flow, which is used to describe a unidirectional set of packets which are being sent from a single sender to a single receiver. We showed what it means that a flow is constrained and how constrains of flows can be calculated.
In section 3, the concept of servers and service curves, delay bounds and backlog bounds have been defined and explained. We put all concepts together and showed how service guarantees can be calculated even when multiple traversed servers and active flows are involved.
In section 4, we have shown what the limitations of the application of DNC are.
In section 5, we briefly introduced alternative approaches to calculate performance bounds and compare their limitations and corresponding use cases.

Overall we have seen that the analysis of the network is bound to the exact requirements of each individual flow. Therefore the exact requirements have to be known in beforehand to be able to calculate performance bounds using DNC. Therefore the constructed network is bound to the initial planned use case, also in terms of hardware. That fact makes the use cases of DNC inflexible. The effort of defining those exact requirements and making the calculations is only worth it in special real-world applications.
On one hand, the calculated performance guarantees of DNC can be valuable whenever the criticality is high, e.g. when even lives are at stake. On the other hand, the calculations of DNC assume that the network is error-free, which means that a single point of failure could break calculated performance guarantees partly or even fully. This problem still has to be taken into mind. One real-world application is the validation of embedded networks inside the Airbus A380 and A350 [4].

References

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

[2] ——, "A calculus for network delay. ii. network analysis," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 132–141, Jan 1991.

[3] J. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003.

[4] F. Geyer, "Quality-of-service and network calculus," 2019, [accessed 20-May-2019]. [Online]. Available: https://acn.net.in.tum.de/slides/190205_chap12_QoS_Network_Calculus.pdf

[5] A. Tanenbaum, *Computer Networks*, ser. Computer Networks. Prentice Hall PTR, 2003, no. S. 3.

[6] A. V. Bemten and W. Kellerer, "Network calculus: A comprehensive guide," 2016, [accessed 23-June-2019]. [Online]. Available: https://mediatum.ub.tum.de/doc/1328613/1328613.pdf

[7] E. Heidinger, "Worst case analysis - network calculus," 2012, [accessed 20-May-2019]. [Online]. Available: https://www.net.in.tum.de/pub/systemperformanz/ss2012/skript/networkcalculus.pdf

[8] J. B. Schmitt, F. A. Zdarsky, and I. Martinovic, "Improving performance bounds in feed-forward networks by paying multiplexing only once," in *14th GI/ITG Conference - Measurement, Modelling and Evalutation of Computer and Communication Systems*, March 2008, pp. 1–15.

doi: 10.2313/NET-2019-10-1_12

[9] H. Schiøler, J. J. Jessen, J. D. Nielsen, and K. G. Larsen, "Network calculus for real time analysis of embedded systems with cyclic task dependencies," in *Computers and Their Applications*. Citeseer, 2005, pp. 326–332.

[10] F. Baccelli and D. Hong, "Tcp is max-plus linear and what it tells us on its throughput," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 219–230, Aug. 2000, [accessed 23-May-2019]. [Online]. Available: https://doi.acm.org/10.1145/347057.347548

[11] M. Fidler, "Survey of deterministic and stochastic service curve models in the network calculus," *IEEE Communications Surveys Tutorials*, vol. 12, no. 1, pp. 59–86, First 2010.

[12] L. Lipsky, *Queueing Theory - A Linear Algebraic Approach*, 2nd ed. Berlin Heidelberg: Springer Science & Business Media, 2008.