On Delay Bounds and Measurements: A COTS Testbed for Network Performance Experimentation

Bruno Cattelan
TU Kaiserslautern,
DISCO | Distributed Computer Systems Lab,
Kaiserslautern, Germany

Steffen Bondorf
Ruhr University Bochum,
Faculty of Mathematics, Center of Computer Science,
Bochum, Germany

***Abstract*—In many application domains such as the automotive and the avionics sector, networks need to fulfill different non-functional requirements. Among them is the strict demand to provide predictable and deterministic worst-case communication performance. Without formal verification of this property, an entire system embedding the network may not be certified and thus not be operated. A methodology for formal verification of communication performance is the Deterministic Network Calculus (DNC). DNC is under active development to better model real systems and to reduce pessimism required to provide verifiably correct bounds on communication delays. This previous work is mostly of theoretical nature. In our paper, we aim at providing a testbed design to benchmark bounds against measurements. Our design is composed of off-the-shelf components that we accompany with a custom software stack for taking measurements (network configuration and time stamping) as well as deriving delay bounds with DNC (system modeling and analysis). We also provide lessons-learned as well as first results that compare delay measurements and DNC-derived bounds.**

I. Introduction

Provably correct upper bounds on the transmission delay of time-sensitive data are required by a multitude of modern systems, for example in any x-by-wire application in the avionics industry [1]. For their networking backbone, the avionics sector started early to develop a standard based on the wide-spread IEEE Ethernet, the so-called Avionics Full-Duplex Ethernet (AFDX) [2]. In this process, Deterministic Network Calculus (DNC) was identified as the tool to provide the upper delay bounds required for certification [3]. This application of DNC has seen quite some attention, e.g., in order to improve the design of networks with delay bounds in mind [4, 5, 6, 7]. Independent of any application domain, DNC has seen many improvements. Most notably for our work, DNC can derive tight delay bounds [8, 9]. I.e., under the assumptions captured in the network model, there is a DNC analysis that can derive best bounds such that no smaller result can be a valid bound, too. On the more practical side, there has also been considerable progress in tool support [10], computational effort [11], the tradeoff between computational effort and tightness [12] (the quality measure for delay bounds, also called "accuracy" in its interpretation of a non-binary property), as well as the related aspect of scaling to growing models [12, 13].

This work was supported by the TU Kaiserslautern network for the promotion of young scientists, TU-Nachwuchsring, under research grant #2019-1.

Our work on a testbed is motivated by the observation that literature on the comparison between DNC-derived delay bounds and delay measurements is very scarce. An example to this stream of work can be found in the "Sensor Network Calculus", a variant of DNC for (wireless) sensor networks. Research in this area resulted in a medium access protocol based on DNC. Its implementation and experimental assessment revealed that DNC bounds can be very accurate [14]. However, this work employs DNC to design the network behavior that can therefore be modeled and analyzed perfectly with DNC. Efforts like AFDX and IEEE TSN, on the other hand, start as a standardization of behavior that needs to be modeled and analyzed with the capabilities of DNC in a subsequent step. When this cannot be achieved perfectly with DNC, it is to be expected that the modeling step as well as the analysis step both need to add pessimism in order to verifiably derive valid upper delay bounds. This, of course, leads to untightness (inaccuracy). A theoretical investigation on this subject can be found in [15].

Our paper aims to complement these efforts with a comparison between measurements and delay bounds in an Ethernet-based network. To achieve this goal, we have built a testbed for delay measurements. The first insight, leading to the first design decision, is that some current avionics networks seem to be relatively small [16, 17]. This makes it feasible to emulate them to an appropriate extent, and we opted for borrowing design concepts from AFDX. The decision also allows us to tap into the rich literature on modeling AFDX concepts in DNC, foremost [18]. However, we want to emphasize that we only provide a sample configuration in this paper. It is possible to extend and adapt our basic design to emulate other domains such as electrical substation networks [19, 20, 21].

Moreover, we set the objective to create a testbed design that resembles the core AFDX network architecture by using off-the-shelf Ethernet components instead of expensive AFDX switches or clock-synchronized network interfaces [19]. For reference, other Ethernet-based solutions have already been evaluated for their potential to replace AFDX in the avionics sector [22, 23]. This decision might add to the untightness of our delay bounds as we need to work with a model that was simplified by pessimistic assumptions. But it also prevents us from the inherent complexity in configuration of sophisticated hardware – e.g., priority classes at AFDX switches' output ports. Instead, we revert to the assumption of a single shared queue and FIFO multiplexing, a modeling assumption that, in turn, can be analyzed perfectly with DNC [9].

We run our testbed with a custom software stack that automates the crucial steps towards derivation and comparison of delay measurements and bounds:

1) configuration of systems, in particular data flows, routes;
2) conducting a simulation run, including the collection of packets' time stamps and the computation of end-to-end delay observations (measurements);
3) DNC-modeling of the testbed setup and computation of delay bounds;
4) centralized data collection and visualization of the results.

Comparing both results, the theoretical upper bounds and the measurements allows to investigate how conservative these bounds are under our model assumptions and simplifications – we show that it is worthwhile to further develop [9] to incorporate more system characteristics into the analysis. These insights from our testbed can potentially help in further optimizing the design [4, 5, 6, 7], too. Last, we also present our lessons learned from building and using our testbed.

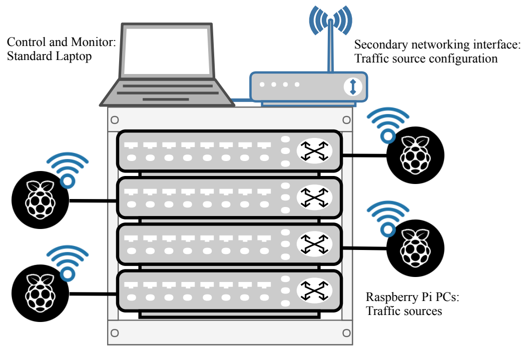
The remainder of the paper is organized as follows: Section II provides the background theory on network characteristics in avionics and on DNC for deriving delay bounds. Section III presents our testbed design, mapping the above concepts to hardware, their configuration and their DNC model. This section also presents our software stack that is used to derive the numerical results shown in Section IV. Section V concludes the paper with an outlook on improvements and lessons learned.

II. Background

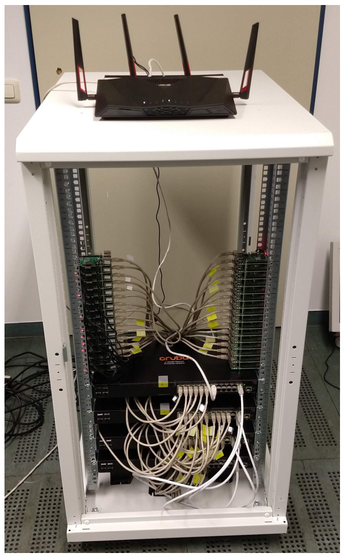
A. Avionics Networks

Avionic Full Duplex Ethernet (AFDX) is a network standard [2] developed by Airbus and employed also by other aircraft manufacturers. The main characteristic of these networks is their adaptation of standard Ethernet for real-time systems. Therefore, they require formal certification. Deterministic Network Calculus has been used for such certification, as in the example of the Airbus' A380 backbone [3] .

The actual network topologies in aircraft are, however, company secrets and we need to rely on a limited amount of literature for insights on them [16, 17]. In general, these topologies are composed of an edge and a core. Systems that generate and send data are located at the edge – these systems are called End-Systems (ESs) – and they are connected to each other through a dense core of switches. They generate data flows that can sometimes be multicast. These are denominated Virtual Links (VLs). Each VL is described by two values: Bandwidth Allocation Gap (BAG), and the Max Frame size (MaxFrame). The BAG describes the minimum interval between transmissions, values are in powers of two and can range from 1ms to 128ms. MaxFrame defines the maximal size of a packet.



(a) Testbed Design.



(b) Assembled Testbed.

Figure 1: Testbed design. Planned device composition and the assembled testbed.

B. Deterministic Network Calculus

A comprehensive background on DNC can be found in [24, 25]. The two crucial concepts for resource description are:

- *Arrival Curve.* A non-negative, monotonically increasing function upper bounding the amount of data that a flow can send into the network over a period of time.
- *Service Curve.* A non-negative, monotonically increasing function that lower bounds the forwarding service that a server offers to the aggregate of flows crossing it.

A network is represented as a so-called server graph in DNC. I.e., an unidirectional graph of servers forwarding data. The arrival curves of flows are only known at their ingress locations. Moreover, the paths of flows are assumed to be static, as are the multiplexing locations. At these locations, we assume FIFO multiplexing, followed by FIFO service. Given the FIFO assumptions, we will employ the tight FIFO analysis for computation of tight bounds [9].

III. Testbed Design and Setup

In this section, we present the design, implementation, configuration and modeling of our testbed. It is shown in Figure 1, from an early design sketch to its eventual assembly.

A. The Network

Figure 2 shows the DNC models of our testbed. Figure 2a depicts the so-called device graph, a model that is essentially the same as the design of Figure 1a (switches are numbered top to bottom as shown in Figure 1). It is the precursor of the server graph introduced in Section II-B. For the conversion [7], output ports of the modeled Ethernet switches are assumed to be the only locations of potential queueing to consider (i.e., we assume input ports operate at least at line speed). Therefore, they are converted to servers. (Outgoing) Links in the server graph represent turns over devices, from the current output port to a subsequent one. Therefore, links are unidirectional and servers can have multiples input and output links.

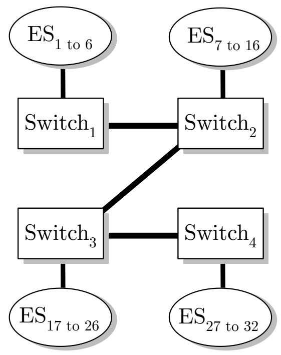
For the sample network configuration under investigation, we show the server graph in Figure 2b. After the conversion, it is considerably more complex than the small network graph. Despite the small device graph, it is already larger than the networks found in exemplary evaluations in the literature on avionics networks [26, 7]. On the other hand, it seems not as large as potentially realistic avionics networks [16, 17].

B. Raspberry Pi Cluster: The End-Systems

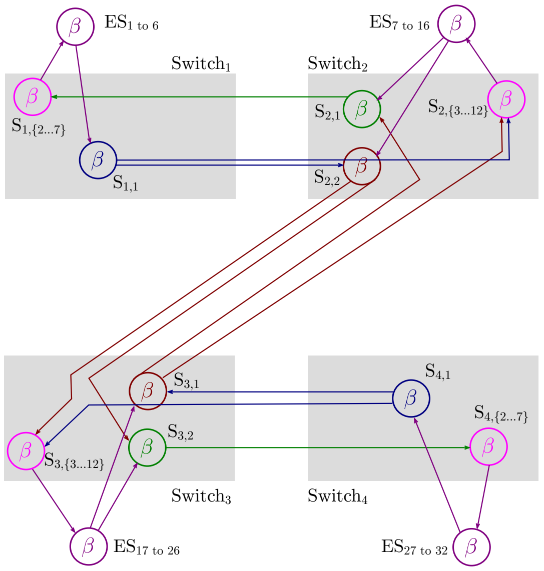
The cluster has 32 Raspberry Pis (RPis) Model 3B+ connected through 4 switches (HP Aruba 2930F). The disposition of the devices mimics the topology of AFDX networks. That is, End-Systems (ESs, the RPis) are connected to each other through a heavily connected center of switches. The distributions of the devices is shown in Figure 2a. The RPis boot Raspian, which runs the ptp daemon (ptpd) such that their clocks are synchronized with the Precision Time Protocol. It also decides on a master (whoever has the best internal clock according to the PTP standard) and synchronizes all other devices based on it. In this way, we can use the timestamps given by the system in order to compute the delays of the packets and avoid the cost of in-network measurements [27].

To control the RPi devices at the edge, we use the message passing interface (MPI). Both MPI and ptpd use the WiFi interfaces of the RPis, leaving the Ethernet interfaces free of this traffic. I.e., data traffic required to configure and control the devices does not impact the performance measurements taken on the wired Ethernet network.

To control the RPi devices according to our selection of parts of the ADFX standard, there was no existing off-the-shelf solution available. Therefore, we developed a set of software from scratch. First, a Python script controls each ES. This program takes as input a configuration file, which describes the VLs (virtual link connections between RPis) in each ES. This design choice means that all ESs have the same configuration file, but only the pertinent information for each device is used.



(a) Device Graph.



(b) Server Graph (data flows not shown).

Figure 2: Testbed modeling. DNC device graph for the testbed design of Figure 1 and its conversion to a DNC server graph.

For the packet creation and capture, we employed the Scapy library[1].

The controller also aimed at a soft synchronization between the ESs, for which MPI was used. That is, all programs start their execution at approximately the same time. This was necessary due to the necessity of custom packets creation at each ES. Depending on the BAG values and the number of VLs the time necessary to create them could vary, and therefore a MPI barrier was implemented so that all programs would be ready when the experiment started. These custom packets are used to identify individual packets at different machines.

Secondly, we implemented a random configuration generator that creates the file used by the above program. It follows a similar pattern as the AFDX generator shown in [18]. It receives as input minimal and maximal values for the number of VLs per ESs and the size of the packets generated. Additionally, it is also given a list of possible BAG values (see

[1]www.scapy.net

Section II-A). This list also gives the probabilities of each BAG value to be used in a given VL. The position in the list is the determining factor, with probability p being defined as $p = \frac{index+1}{finalTotalValue}$ where $finalTotalValue$ is the final value from the addition of all $index + 1$. This design aims at avoiding some ESs to be overly stressed due to the relatively weak performance offered by RPi devices.

There is also a Java program extending the DNC software tool [10][2]. In order to model the network as a server graph, it reads the configuration file created in the previous program and it convert the VLs into DNC flows.

C. Component Configuration

For bounding the delays, we required an optimization software. We employed IBM CPLEX and an experimental addition to the DNC software tool that uses the CPLEX Java API. The optimization's result constitutes the best-known delay bound that can be computed with DNC for FIFO networks, considering the modeling assumptions. The code is an implementation of the work presented in [9]. However, a drawback is that the background theory only works with unicast flows whereas VLs may be multicast. The unicast transformation that we used can convert the network in a way that guarantees validity of worst-cast bounds [28].

Since two different types of devices were used, we modeled two different service curves. Similar devices share the same service curve.

RPis: The Ethernet port of the RPis is limited to 300Mbps. Therefore, the rate of its service curve is set to this value. Regarding other sources of latency, we can report that it needs to be estimated, since its datasheet [29] does not specify them. It does, however, say that the Ethernet is connected through an USB 2.0 interface. According to its specification [30], two latencies are possible. For a full/slow speed bus we have 1ms, and for high speed 125μs. Since some RPis were in fact overloaded, and this is a worst case analysis, the latency assumed was 1ms.

Switches: The Aruba switches from HP have Gigabit Ethernet ports, and therefore their service curve rate was of 1Gbps. Following the documentation provided by HP, the latency of the switch is in fact related to the link rates and packet sizes. Since we work always with the worst-case, we used the largest latency that could happen: This was of 16.2μs.

The arrival curves are modeled as follows: The rate is given by $\frac{maxFrame}{BAG}$, and the burst is the $maxFrame$ value itself. It means that each VL is upper bounded by the maximal size of the packets sent at predetermined intervals. Since this is a well behaved system, the burst is simply one maximal packet size, $maxFrame$.

D. Simulation Workflow

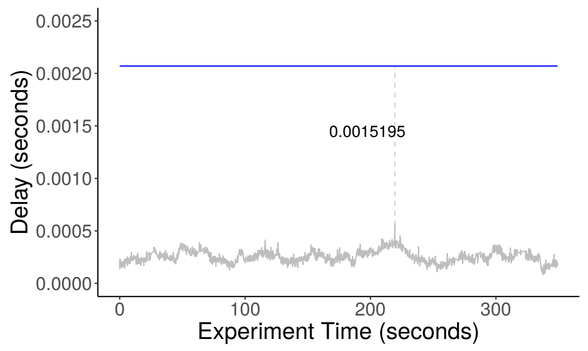
1 - Configuration Phase: The Java code is used to create a configuration file. This file describes the VLs; their source, destination, BAG and packet size. This file (and other modifications to the Python code) is sent to each RPis, so that they all have the same configuration. A simple Shell script is used for the file transfers. After it, the RPis have all the information necessary to start the execution, and so a command is sent through MPI. This starts the next phase.

2 - Preparation Phase: First, the controller uses MPI in order to start the execution of the ESs scripts. Each ES then reads the configuration file and filters out the relevant part based on its own IP address. Based on this, each ES will create a separate software thread. This thread creates a list of packets, which will be used in the next phase. This is done in order to achieve the necessary inter arrival times (BAG) required, while also having custom packets. That is, instead of creating packets "on the fly" they are created in this initial step. The weak hardware of the RPis does not allow the necessary speed otherwise. Each packet will have its own IP ID and port source. This allows us to compute the delay of any packet sent, since they are all unique. However, it also increases the runtime of the experiment, and the memory requirements of the scripts. Once the ES has finished creating the lists, new threads are created (but not executed yet). These threads will handle the actual data sending in the next step. Once they are created, the ESs reach a MPI barrier, and wait for the other ESs to finish creating their own lists. We do not intend to perfectly synchronize each ES (inheriting the potential but small offset from the PTP protocol), but instead to have some control on when the experiment actually starts, since the list creation can take some time, and there is a great variability depending on how many VLs an ES has, or their BAGs. Another thread is also created (but not executed immediately either) that handles the packet capture in the next phase.

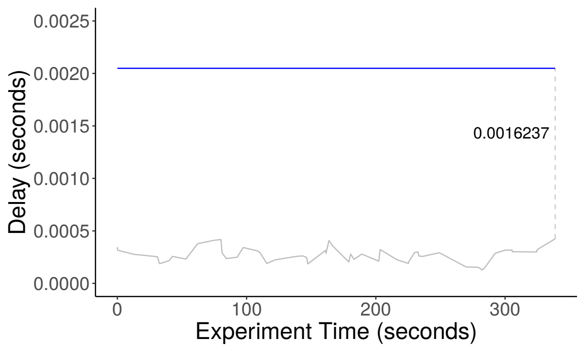
3 - Execution Phase: Once all ESs reach the MPI barrier, all threads are started. The experiment is executed for as long as it was configured to (we have to add limits on how long it can execute due to the memory limitations of the RPis). As packets are captured, they are written into a temporary file. The pertinent information is the time of capture, the IP header identifier, the source port, the size of the packet and the IP addresses (source and destination). There is also a flag called "sent" – a safety mechanism to save resources. This flag is set by the ES as it captures the packets. If it identifies the packets as having itself as source, it means that it sent the packet. Otherwise, is a captured packet from another ES.

4 - Data Extraction and Visualization: Once the experiment is completed, a Shell script is used to collect the data from the temporary files in each RPi to a central computer (in this case, the laptop on top of the server rack in Figure 1a). Another Java program is then executed. It uses the configuration file of step 1, and takes care of the entire DNC analysis (from modeling to execution). The aggregate file and the output from the previous program are both fed into a R script, which computes the delays of each packet, and displays the largest delay experienced in each VL. This script also creates the visualizations presented next.

[2]An open-source software hosted at: https://github.com/NetCal/dnc



(a) Example VL 1: from 29 to 2.



(b) Example VL 2: from 6 to 31.

Figure 3: Delay bound and measurements over time for virtual links from ES 29 to 2 and from 6 to 31.

IV. Evaluation Results

For an example run of the testbed, configured as shown in Figures 1 and 2, we show the results.

The first result presented is a barplot that depicts all VLs between the ESs on the very edge of the network (see Figure 4). I.e., the end-systems connected to Switch 1 and Switch 4, as shown in Figure 2a. The ID of the flows is on the x-axis. VLs are labeled as $X\text{to}Y$, where X is the source ES and Y is the destination ES.

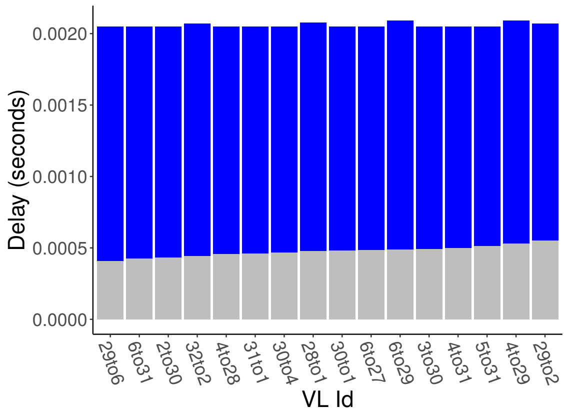


Figure 4: Delay bounds (entire bars) and maximum measurement during the test run (fraction of bars) for all virtual links.

The bars show the delays for the respective VL. As the theoretical result is an upper bound on the worst-case end-to-end delay, it must be always larger than the biggest measured delay. Therefore, the bars are bicolors; blue depicting the theoretical DNC bound, and the gray part showing the maximum measured delay. It can be seen that the theoretical bound is not close to the measurement - it cannot be guaranteed that we saw the actual worst-case system behavior while measuring. In particular, as our PTP time synchronization is not perfect, we might not see the worst-case simultaneous arrival of bursts that easily. Yet, this might not explain the entire gap. The tight FIFO analysis we use has inherent modeling restrictions. We already mentioned the unicast flows, but it also operates on a fluid model that over-approximates packetized traffic that would be better modeled with staircase functions [31]. Moreover, the shaping nature of a link, e.g., only a single packet can be sent at once and thus the burstiness propagation should be restricted by the link [3, 19], is lacking. Last, the link exact speed is used as a lower bound on the data transmission rate, i.e., in a service curve. It is also a maximum rate that can be modeled by a maximum service curve not yet part of the analysis of [9]. We hope the currently observed gap between measurements and bounds sparks research in this directions.

The second visualization alternative for our results is dedicated to individual VLs over the simulation time. End-to-end transmission delays tend to vary over time, an aspect not visible on the barplot. Therefore, the second plot alternative has the experiment time on the x-axis (Figures 3a and 3b). The time-invariant DNC delay is also included in the plot. We can observe a single measurement, that defines the experienced maximum delay each (this distance is shown in seconds). Moreover, this measurement happens at different times of the simulation run, showing that different flows experience their worst case at distinct times. Last, it is possible to see the effect of the BAG values when comparing the two plots. Figure 3a has a smaller BAG value, and therefore the number of packets per second is much larger than in Figure 3b.

V. Conclusion

In this paper, we presented the design of a COTS testbed built from off-the-shelf components. It is designed to emulate a small avionics-style network. We achieve this by creating a custom software stack that provides an easy way to configure the network with parameters known from AFDX, as well as an automated workflow to measure delays, model the network, bound delays with DNC, and visualize results.

Due to the pessimistic nature of DNC, some gap between measurements and the formal bounds will exist. The worst case modeling always assume the worst possible system trajectory plus some pessimism as a security margin. Such a trajectory, however, is usually a rare sight in systems. For example, in the

DNC for sensor networks, it was found that the system needs to become very unsynchronized to see a small gap between measurements and bounds [14]. This cannot be enforced in our avionics-emulating testbed. We still expect the DNC analysis to have room for improvements (see Section IV) and we hope our results have sparked interest in that research.

Pitfalls and Lessons Learned

We conclude the paper with insights into the problems we encountered in this hands-on work.

a) Effort of the theoretical analysis: For this small example (32 ESs), the optimization-based DNC analysis (using IBM CPLEX) imposed execution times of more than 1 hour wall-clock time, with over 5GB memory usage. The machine used was a Lenovo Thinkpad E495 laptop, with 12Gb ram and Ryzen 7 processor. Alternatively, it is possible to execute an algebraic DNC FIFO analysis [32] or a hybrid one [33]. We expect the analysis times to become considerably smaller but this is paid for by increased bounds.

b) Effort of running the testbed: Effects of memory limitations and computational power of the RPi 3B+ devices have already been mentioned above.

Another problem that remains in the rather well-synchronized testbed is that the computational effort at RPis synchronizes, too. This lead to a power surges that cannot always be handled well by our COTS testbed design where we used a small number of wall outlets and extension cords. During surges, individual RPi devices were temporarily operated with electrical voltages below the specified safe range. This corrupted SD cards with the Linux operating system (potentially during an experiment) and enforced a time-consuming new installation. This problem can only be solved by a more expensive, dedicated power supply unit for RPi clusters.

Moreover, the memory limitation also put a limit on the total time of the simulation run – and thus lead to an overall low confidence for bad system trajectories to be seen, which could cause measured delays close to the worst-case. In order to avoid this, a focus on only one flow (instead of all flows at once) could be implemented. In this way, only the packets for that flow of interest would be stored, which can greatly improve the number of custom packets for one execution. This might also counteract the power supply problem.

References

[1] F. Geyer and G. Carle, “Network engineering for real-time networks: comparison of automotive and aeronautic industries approaches,” *IEEE Comm. Mag.*, vol. 54, no. 2, pp. 106–112, 2016.

[2] Aeronautical Radio, Incorporated (ARINC), “Avionics Full-Duplex Switched Ethernet (AFDX),” Specification 664 Part 7, Tech. Rep., 2009.

[3] J. Grieu, “Analyse et évaluation de techniques de commutation ethernet pour l’interconnexion des systèmes avioniques,” Ph.D. dissertation, INPT, 2004.

[4] F. Frances, C. Fraboul, and J. Grieu, “network calculus to optimize the AFDX,” in *Proc. of ERTS*, 2006.

[5] A. Al Sheikh, O. Brun, M. Chéramy, and P.-E. Hladik, “Optimal design of virtual links in AFDX networks,” *Real-Time Syst.*, vol. 49, no. 3, pp. 308–336, 2013.

[6] X. Zheng, N. Huang, Y. Zhang, and X. Li, “Performability optimization design of virtual links in AFDX networks,” in *Proc. of RAMS*, 2016.

[7] B. Cattelan and S. Bondorf, “Iterative design space exploration for networks requiring performance guarantees,” in *Proc. of IEEE/AIAA DASC*, 2017.

[8] A. Bouillard, L. Jouhet, and É. Thierry, “Tight performance bounds in the worst-case analysis of feed-forward networks,” in *Proc. of IEEE INFOCOM*, 2010.

[9] A. Bouillard and G. Stea, “Exact worst-case delay in FIFO-multiplexing feed-forward networks,” *IEEE/ACM Trans. Net.*, vol. 23, no. 5, pp. 1387–1400, 2015.

[10] S. Bondorf and J. B. Schmitt, “The DiscoDNC v2 – a comprehensive tool for deterministic network calculus,” in *Proc. of EAI ValueTools*, 2014.

[11] A. Scheffler, M. Fögen, and S. Bondorf, “The deterministic network calculus analysis: Reliability insights and performance improvements,” in *Proc. of IEEE CAMAD*, 2018.

[12] S. Bondorf, P. Nikolaus, and J. B. Schmitt, “Quality and cost of deterministic network calculus – design and evaluation of an accurate and fast analysis,” *Proc. of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, vol. 1, no. 1, pp. 16:1–16:34, 2017.

[13] F. Geyer and S. Bondorf, “On the robustness of deep learning-predicted contention models for network calculus,” 2019, arxiv:1911.10522.

[14] U. Roedig, N. Gollan, and J. Schmitt, “Validating the sensor network calculus by simulations,” in *Proc. of WICON*, 2007.

[15] A. Soni, X. Li, J.-L. Scharbarg, and C. Fraboul, “Pessimism analysis of network calculus approach on AFDX networks,” in *Proc. of IEEE SIES Work-in-Progress Session*, 2017.

[16] O. Hotescu, K. Jaffrès-Runser, J.-L. Scharbarg, and C. Fraboul, “Towards quality of service provision with avionics full duplex switching,” in *Proc. of ECRTS Work-in-Progress Session*, 2017.

[17] N. Tobeck, “Enforcing domain segregation in unified cabin data networks,” in *Proc. of IEEE/AIAA DASC*, 2017.

[18] M. Boyer, N. Navet, and M. Fumey, “Experimental assessment of timing verification techniques for AFDX,” in *Proc. of ERTSS*, 2012.

[19] H. Yang, L. Cheng, and X. Ma, “Analyzing worst-case delay performance of IEC 61850-9-2 process bus networks using measurements and network calculus,” in *Proc. of ACM e-Energy*, 2017.

[20] ——, “Bounding network-induced delays for time-critical services in avionic systems using measurements and network calculus,” in *Proc. of ACM/IEEE ICCPS*, 2019.

[21] ——, “Combining measurements and network calculus in worst-case delay analyses for networked cyber-physical systems,” in *Proc. of IEEE INFOCOM Workshops*, 2019.

[22] P. Heise, F. Geyer, and R. Obermaisser, “Deterministic OpenFlow: Performance evaluation of SDN hardware for avionic networks,” in *Proc. of CNSM*, 2015.

[23] L. Zhao, F. He, E. Li, and J. Lu, “Comparison of time sensitive networking (TSN) and TTEthernet,” in *Proc. of IEEE/AIAA DASC*, 2018.

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

[25] A. Bouillard, M. Boyer, and E. Le Corronc, *Deterministic Network Calculus: From Theory to Practical Implementation*. John Wiley & Sons, Ltd, 2018.

[26] G. Kemayo, N. Benammar, F. Ridouard, H. Bauer, and P. Richard, “Improving AFDX end-to-end delays analysis,” in *Proc. of IEEE ETFA*, 2015.

[27] L. Linguaglossa, F. Geyer, W. Shao, F. Brockners, and G. Carle, “Demonstrating the cost of collecting in-network measurements for high-speed VNFs,” in *Proc. of IFIP TMA*, 2019.

[28] S. Bondorf and F. Geyer, “Generalizing network calculus analysis to derive performance guarantees for multicast flows,” in *Proc. of EAI ValueTools*, 2016.

[29] Raspberry Pi Foundation, *Raspberry Pi 3 Specification*.

[30] *Universal Serial Bus Specification*, April 2000, rev. 2.0.

[31] M. Boyer, J. Migge, and N. Navet, “An efficient and simple class of functions to model arrival curve of packetised flows,” in *Proc. of WCTT*, 2011.

[32] M. Fidler, “Extending the network calculus pay bursts only once principle to aggregate scheduling,” in *Proc. of QoS-IP*, 2003.

[33] L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea, “Estimating the worst-case delay in FIFO tandems using network calculus,” in *Proc. of ICST ValueTools*, 2008.