Building End-to-End IoT Applications with QoS Guarantees

Arne Hamann[1], Selma Saidi[2], David Ginthoer[1], Christian Wietfeld[2], and Dirk Ziegenbein[1]

[1]Corporate Research, Robert Bosch GmbH, Germany, firstname.lastname@de.bosch.com
[2]Technical University of Dortmund, Germany, firstname.lastname@tu-dortmund.de

***Abstract*—Many industrial players are currently challenged in building distributed CPS and IoT applications with stringent end-to-end QoS requirements. Examples are Vehicle-to-X applications, Advanced Driver-Assistance Systems (ADAS) or functionalities in the Industrial Internet of Things (IIoT). Currently, there is no comprehensive solution allowing to efficiently program, deploy, and operate such distributed applications. This paper will focus on real-time concerns, in building distributed CPS and IoT systems. Thereby, the focus lies, on the one hand, on mechanisms required inside of the IoT (compute) nodes, and, on the other hand, on communication protocols such as TSN and 5G connecting them. In the authors' view, the required building blocks for a first end-to-end technology stack are available. However, their integration into a holistic framework is missing.**

I. Introduction and Motivation

The product portfolios in various industrial domains currently evolve from classical non-connected cyber-physical system (CPS) applications (e.g. engine management) over cloud enhanced CPS that use networking and cloud computing for non-critical functional extensions (e.g. sending monitoring data to the cloud for predictive maintenance) to dynamic distributed CPS where time and safety critical parts of the functionality might get shifted to the (edge) cloud. Examples for these dynamic distributed CPS are Vehicle-to-X applications (e.g. platooning or intersection assistance), Advanced Driver-Assistance Systems (ADAS) where smart sensors/actuators communicate over in-vehicle networks with centralized vehicle computers, or adaptive and flexible manufacturing solutions in the Industrial Internet of Things (IIoT).

In this context, we witness a convergence of technologies from the embedded and IT domains which leads to an unprecedented level of heterogeneity on all levels of the technology stack. This includes classic "hard-wired" software architectures and real-time operating systems (e.g. AUTOSAR) as well as publish-subscribe and service-oriented middlewares (e.g. ROS2, MQTT) and POSIX operating systems (e.g. QNX, Linux), embedded micro-controllers with dedicated I/O units and on-chip SRAM as well as micro-processors with network on chips, GPUs and off-chip DRAM, and wired as well as wireless communication media with a plethora of different protocols.

On the application level, the consequence of dynamic distributed CPS with time-critical application parts being distributed to the (edge) cloud is that there are now cause-and-effect chains spanning multiple distributed compute and communication resources. The quality of service (QoS) requirements for these applications include safety, security, availability, reliability as well as timing. In this paper, we mainly focus on timing such as end-to-end (E2E) latencies along cause-and-effect chains in distributed IoT applications.

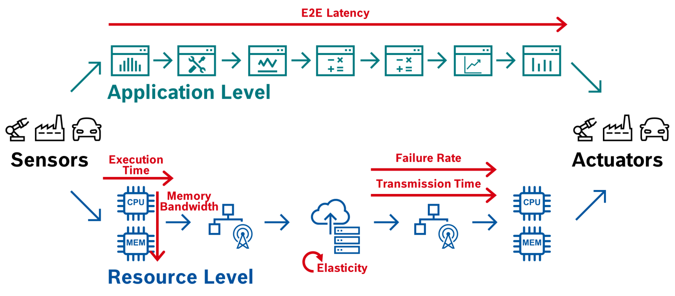

Fig. 1. System-wide QoS requirements (denoted by red arrows) on application level are broken down to resource specific QoS requirements

Figure 1 shows one application with an E2E latency requirement from sensors to actuators. When this application is deployed to a distributed system involving embedded compute, wired and wireless communication as well as cloud resources, the E2E latency requirement still has obviously to be fulfilled. However, as a consequence of the deployment the E2E latency requirement is broken down to different resource specific QoS requirements such as execution time and memory bandwidth requirements for the embedded compute and minimum transmission rates and failure rates for the communication network.

Resources are typically shared by multiple applications with heterogeneous QoS requirements. Some require hard guarantees of latency bounds while others are more concerned with a best effort average throughput. Thus, adequate resource provisioning to applications and isolating the applications against each other are of utmost importance to be able to guarantee QoS requirements. This is even more relevant, since most of the available technologies are built with best effort in mind and do not support hard QoS guarantees out of the box.

In this paper, we will have a look at the different technology domains, namely the embedded compute and wired/wireless communication, and discuss promising solution approaches to guarantee end-to-end QoS properties for distributed CPS and IoT applications.

II. Embedded Compute

Heterogeneous System-on-Chip (SoC) platforms are currently being increasingly used in embedded real-time systems to satisfy the tremendous compute power demands of new dynamic distributed CPS applications in various domains including automated driving, the factory of the future, and augmented/extended reality. These SoCs are usually µP-based featuring a variety of integrated special purpose accelerators including GPUs and FPGAs. Examples for theses class of SoCs include NXP's S32V vision processor family, or the Tegra series developed by Nvidia.

Figure 2 shows a schematic example highlighting the characteristics of importance for this paper: the memory hierarchy (caches and DRAM) shared between different execution engines, as well as the interconnect fabric connecting different master components such as CPU clusters and special purpose accelerators. The architecture of these SoCs is usually driven by the goal of achieving high throughput and optimized average performance. As a result, they only provide "best-effort" performance from real-time systems point-of-view.

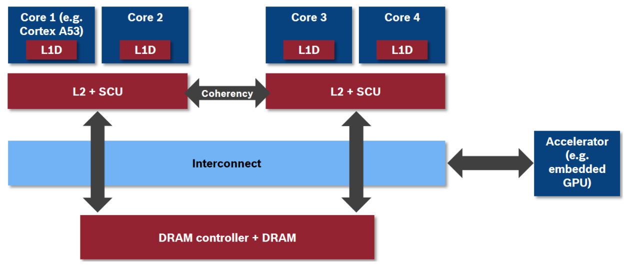

Fig. 2. Schematic example of a μP-based compute platform considered in this paper

One big challenge that engineers in different industrial domains are currently facing, is that heterogeneous applications following different models of computation (e.g. perception, planning, control, (deep) learning, etc.) must be integrated onto these powerful SoC platforms, while ensuring that individual real-time and safety requirements are met.

In order to tackle this highly complex integration task, engineers currently resort to richer and more complex software stacks borrowed from the IT world including POSIX-based operating systems, and middleware systems. While these IT technologies are based on powerful abstractions hiding the underlying system complexity and offer (highly needed) flexibility, scalability, and dynamic adaptability to implement and integrate complex applications, they do not support the engineer in catering for real-time requirements of the applications built on top.

This is not astonishing since the individual technologies were originally not intended to be used for timing-critical applications where predictable real-time behavior is key. The task at hand is, thus, to extend the utilized technology stack with mechanisms enabling engineers to handle real-time requirements (where needed) constructively. Thereby, two key requirements need to be satisfied: 1) efficiency of the resulting system in terms of power and cost, and 2) preservation of the level of abstraction as well as ease-of-use to shield engineers from the complexity of the underlying technology stack.

Figure 3 visualizes the technology stack of µP-based embedded compute platforms, as they are currently deployed in various industrial domains such as automotive and the factory of the future, along with required mechanisms for ensuring end-to-end real-time guarantees.

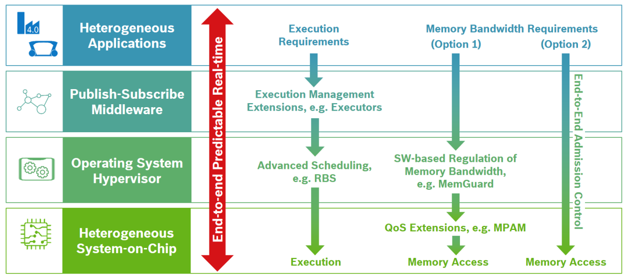
Heterogeneous Applications
Publish-Subscribe Middleware
Operating System Hypervisor
Heterogeneous System-on-Chip
End-to-end Predictable Real-time
Execution Requirements
Memory Bandwidth Requirements
(Option 1)
(Option 2)
Execution Management Extensions, e.g. Executors
Advanced Scheduling, e.g. RBS
SW-based Regulation of Memory Bandwidth, e.g. MemGuard
QoS Extensions, e.g. MPAM
End-to-End Admission Control
Execution
Memory Access
Memory Access

Fig. 3. Required technology stack for real-time guarantees on µP-based embedded compute platforms. For satisfying memory bandwidth requirements two different options are discussed: 1) regulation-based techniques, and 2) centralized end-to-end admission control.

From the application's point-of-view two different concerns need to be taken care of by the underlying platform. First of all, it must be guaranteed that there is enough computational capacity available (and possibly reserved) on the required execution engines (e.g. processors and special purpose accelerators). Secondly, it must be guaranteed that the computations are fed timely with the required data by the memory hierarchy to prevent computations to get stalled. In the following we will discuss both concerns separately.

A. *Execution Requirements*

Classically, real-time systems are built with precise knowledge about the activation patterns of concurrently executed applications. A large part of real-time theory is, for instance, based on the assumption that applications are activated strictly periodically. This periodic model can easily be extended with the notion of activation jitter, indicating that the individual points of execution may vary to a certain extent. For many applications (e.g. control) assumptions like periodic execution make perfectly sense, and most real-time operating systems come along with built-in support for realizing such behavior (e.g. through *periodic tasks*).

However, when looking at middleware systems such as ROS2 [1] or MQTT [2] which heavily use the *publish-subscribe messaging pattern*, controlling and understanding the activation patterns of applications is far more involved.

Publish-subscribe is a messaging pattern implemented by most popular middleware systems. Messages are sent asynchronously by so-called publishers without the knowledge how many (if any) so-called subscribers receive the messages. This decoupling of senders and receivers provides great flexibility since dependencies are not explicitly programmed, and can even be added dynamically during runtime. Messages are usually "delivered" by the execution of callback-functions giving the subscriber(s) the possibility to react and process the received data. In addition to this data-driven activation, most middleware system also support time-driven activation of callback-function through timers.

When it comes to building real-time systems, the publish-subscribe abstraction can cause problems. For instance, since activated callback-functions can publish messages themselves, complex behavior can emerge making it potentially very hard to understand the dynamic behavior of an application. Moreover, in most implementations, the user has only little control over how messages are queued and in which order callback-functions of an application are executed by the underlying operating system. For instance, in ROS2 callback-functions triggered by timers are executed first, afterwards callback-functions triggered by messages are executed in the order they have been registered in the source code [3]. Obviously, with this semantics, messages triggering real-time critical functionalities might get heavily delayed, or, depending on the message queue size, even dropped.

In order to make middleware systems ”real-time capable”, a layer providing fine-grained control over the mapping of callback-functions to the scheduling primitives and mechanisms of the underlying operating system based on high-level real-time requirements is needed. The European research project OFERA [4], for instance, is working on a so-called *Real-time Executor* [5] for ROS2 providing such control.

On the operating system level mostly fixed priority preemptive and TDMA scheduling are currently used in practical systems. Like it has been pointed out in [6], both scheduling strategies are inadequate for efficiently integrating heterogeneous applications with different real-time requirements and models of computation.

A well studied alternative scheduling approach for efficient integration of heterogeneous applications, which surprisingly has found only little attention in commercial products, is *Reservation Based Scheduling (RBS)*. In RBS a fractional part of a CPU is assigned to a server (usually through a periodically replenished budget) that is responsible for executing associated applications. An associated mechanism guarantees that the promised capacity is indeed delivered to the server (e.g. [7], [8]). These guarantees coupled with budget enforcements provide temporal isolation for applications assigned to different servers. In RBS budget accounting is performed at server level allowing for composability. Support for heterogeneous applications can be obtained by combining different reservation types on the same platform. Moreover, with well known and safe techniques (e.g. [9]), any unconsumed server budget can be redistributed guaranteeing work conservation and, thus, resource efficiency.

In summary, RBS provides a comprehensible and composable abstraction for handling computing resources and controlling real-time properties. RBS mechanisms are backed-up with a large body of work proving its real-time properties and making it an analytically well understood and predictable technology.

B. Memory Bandwidth Requirements

Modern SoCs are optimized for average-case performance and extensively make use of shared resources (e.g. the memory hierarchy), hence introducing:

- a strong correlation between the execution of a program and the access to data it manipulates, as well as
- high degree of interference imposing a strong timing correlation between concurrently running applications on the same chip.

In [10] it was shown that these effects can be drastic. For instance, on the Nvidia Tegra X1 SoC the average read latency per word in a sequential read scenario from a single core (in isolation) is below 10 ns, whereas it increases to more than 50 ns when simultaneous memory accesses from 3 other cores are interfering. As a result, the execution time of an application can vary easily by several hundreds percent depending on its memory access patterns and execution context.

One important building block to ensure predictable performance is, therefore, to ensure that applications are fed with the required data in a timely fashion. In the following, we will describe two possible approaches: memory bandwidth regulation and admission control. While the first approach focuses on the coarse grain shaping of memory traffic to reduce the worst-case impact of contention scenarios, the second approach is more constructive in a way that it dynamically reserves all shared resources on the data access path for required transactions, thus enabling stronger guarantees at the cost of higher overall complexity.

Additionally, we briefly discuss system partitioning techniques that can be used to complement both approaches by providing dedicated access to parts of the shared system resources.

1) Memory Bandwidth Regulation: SoCs currently used in system development have very little on-chip support for regulating and guaranteeing memory bandwidth for time-critical applications. However, for future SoCs this might change, since new QoS architecture extensions are currently proposed and developed by major IP vendors, ARM’s MPAM[1] Extension Architecture Specification [11] being one prominent example.

Nevertheless, the goal of achieving application level memory bandwidth regulation on the core clusters can be achieved by adding software techniques such as MemGuard [12] to the overall picture. Or in other words, the lack of hardware support on current SoCs can be compensated by software techniques resulting in a dual-layer memory bandwidth regulation scheme:

- Mechanisms using basic QoS features of the interconnect to perform traffic shaping of the memory bandwidth between all masters (e.g. configuring average/maximum number of outstanding requests on the interconnect). This can in particular be used to ”protect” core clusters from other master components (such as accelerators) and, thus, to guarantee a minimum memory bandwidth that can be safely distributed among executed applications.
- SW-based mechanisms (implemented for instance at hypervisor level) for distributing memory bandwidth among applications executed within core clusters.

The required memory bandwidth per application can be determined with profiling tools. One idea, for instance, is to initially execute an application in isolation without memory bandwidth restrictions, and then gradually restrict the bandwidth until the execution time starts increasing significantly. By this means, sweet spots for memory bandwidth budgets yielding ”near optimal performance” per application can be determined. Obviously, this simple scheme can be extended to account for varying bandwidth requirements based on different phases or modes of an application.

When executing multiple applications in parallel, the sum of all memory bandwidth budgets (corresponding to ”optimal performance”) might be superior to the memory bandwidth that can be sustained by the shared platform. For such cases,

[1]Memory System Resource Partitioning and Monitoring

support for dynamic adjustment of memory bandwidth budgets considering a notion of system-wide "application criticality" is required. For instance, in an "overload scenario" applications with high criticality are guaranteed their budgets at the expense of applications with lower criticalities, whose budgets are dynamically (transiently) reduced.

2) Centralized E2E Admission Control: In order to conduct memory accesses, an application in MPSoCs must generally acquire several shared (interconnect and memory) resources with independent arbiters and often provided by different vendors. Each shared resource may be further divided into sub-resources (i.e., sub-arbiters). For instance, many modern MP-SoCs are equipped with Networks-on-Chips (NoCs) featuring wormhole-switching and multi-stage arbitration (e.g. iSLIP). NoCs resources are not reserved in advance, i.e. packets are switched as soon as they arrive and ongoing transmissions compete for link bandwidth (output ports) and buffer space (virtual channels).

DRAMs feature as well complex internal hierarchical structure as they are composed of multiple modules which are further structured in a number of banks used to store data. Each bank contains a matrix-like structure where data is located along with a row buffer. The matrix-like structure is not visible to the memory controller and all data exchanges are performed through the corresponding row buffer. Commercial off-the-shelf memory controllers are optimized for the average-case performance and for this they rely on the open-row policy. First-ready first-come-first-serve (FR-FCFS) scheduling policy is often used to prioritize memory requests accessing the same neighboring memory region (i.e. same row) over other requests to maximize row-hit rate, and thereby performance.

Conventional network and memory resources do not take into account interference between different threads/applications when making scheduling decisions. Each router is conducting its arbitration locally and independently from other routers and memory accesses are translated by the memory controller into internal DRAM commands used to access data and read/write from row buffers. Applications requests granularity is often different from shared resources granularity of arbitration. While applications issue data transmissions (cache lines or DMA), routers arbitrate among data flits and packets, and memory controllers schedule internal DRAM commands[2].

Centralized E2E admission control can be used as an alternative method to provide applications with a global (physical) resources arbitration. Admission control allows to decouple the data transmission layer, where data flows, from the control layer. Transmissions are therefore established and scheduled at a higher logical level before applications acquire access to physical resources. Arbitration between multiple applications is then shifted from individual (sub) resources to a centralized control unit which has a global view of the system (i.e. both applications and resources). The idea of admission control is not new, it is often used in the IT domain in combination with Software-Defined Networking (SDN) to implement routing processes that are more dynamic and efficient than physical ones implemented in network switches [13].

[2]An application data transmission is decomposed into a number of smaller flits or packets and internal DRAM commands.

In [14], admission control concepts were borrowed from the IT domain to be applied in MPSoCs and provide real-time guarantees for (mixed) critical communication and memory traffic. The proposed approach provides an overlay network built on top of existing NoC architectures. Whenever an applications is granted admission, temporal isolation is achieved by an E2E exclusive access allocation of a sequence of shared network and memory resources during the entire duration of the transmission (i.e. one access or bundle of accesses). This control layer has a global view of current traffic in the network and can dynamically adapt admission control to perform optimization, e.g., assign dynamic TDM slot sizes and change priorities dynamically to handle mixed-criticality traffic. Interestingly enough, when used in combination with DRAMs [15] to provide E2E admission control for memory traffic traversing the network, the proposed admission control proves to provide performance on general-purpose MPSoCs platforms, comparable with custom designed predictable memory controllers like the PRET DRAM controller [16]. One of the advantages of the provided E2E resources allocation is the preservation of the locality of accesses from the same application without, for instance, the need for static resources partitioning schemes like bank privatization.

E2E admission control allows additionally to simplify analytical timing analysis models used to bound interference effects and compute timing guarantees on the the E2E latency of individual transmissions. Bounding the timing effects of shared resources requires a careful analysis of requests arrival (that determine interference) at every resource and its corresponding scheduling/arbitration policy. With admission control, interference analysis can account for applications requests arrival at the centralized control unit instead of individual flits/packets/commands arrival at every (sub) resource. Thus, reducing the complexity of coupling different resources timing analysis which usually leads to pessimistic formal guarantees or decreased performance and utilization.

3) System Partitioning Approaches: Both aforementioned approaches to regulate and control memory bandwidths can be complemented with approaches to partition shared resources along the memory hierarchy. The goal of such approaches is to guarantee freedom-from-interference for concurrently executed applications on the respective shared resources, leading to the decoupling of performance effects and the simplification of the overall memory bandwidth regulation problem. Examples for such partitioning approaches are cache partitioning through page coloring [17] or DRAM bank privatization [16].

Whether or not it make sense to add partitioning approaches to the overall solution, heavily depends on the hardware platform and the executed application mix. Cache partitioning, for instance, privatizes parts of the cache to a set of applications, reducing interference effects from other parts of the system. However, at the same time the usable cache size for those applications is reduced. In case the possible cache interference on system level is anyhow only very low, the overall performance and efficiency could be severely reduced by cache partitioning. For this reason, the use of partitioning approaches must be assessed precisely on a case-by-case basis.

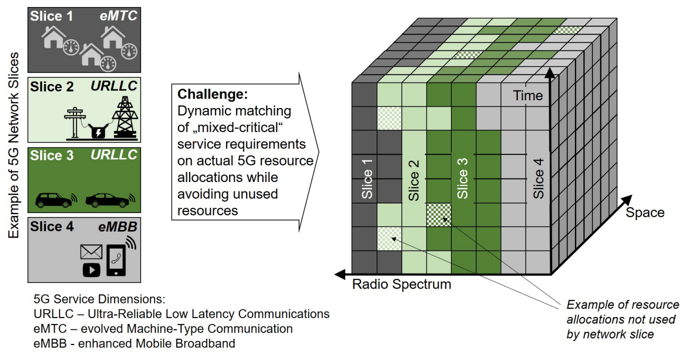
Example of 5G Network Slices
Slice 1 eMTC
Slice 2 URLLC
Slice 3 URLLC
Slice 4 eMBB
Challenge: Dynamic matching of „mixed-critical" service requirements on actual 5G resource allocations while avoiding unused resources
Time
Space
Radio Spectrum
Slice 1
Slice 2
Slice 3
Slice 4
Example of resource allocations not used by network slice
5G Service Dimensions:
URLLC – Ultra-Reliable Low Latency Communications
eMTC – evolved Machine-Type Communication
eMBB - enhanced Mobile Broadband

Fig. 4. Mixed-critical 5G network slices dynamically matched on spatio-temporal resource grid

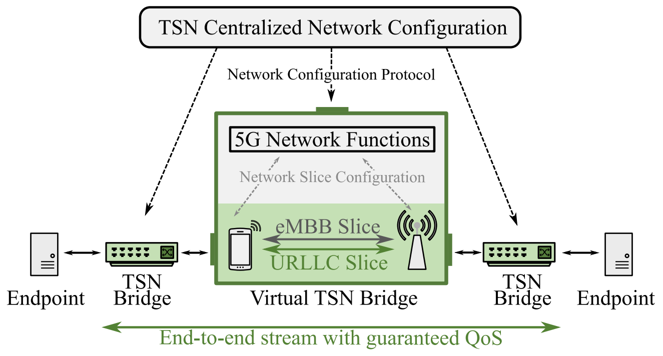
TSN Centralized Network Configuration
Network Configuration Protocol
5G Network Functions
Network Slice Configuration
eMBB Slice
URLLC Slice
Endpoint
TSN Bridge
Virtual TSN Bridge
TSN Bridge
Endpoint
End-to-end stream with guaranteed QoS

Fig. 5. TSN Centralized configuration model with integrated 5G virtual TSN bridge.

III. Wired and Wireless Communication

A. QoS-support in 5G

The development of 5G has very much emphasized the goal to support guaranteed quality of service through the Ultra-Reliable Low Latency Communication (URLLC) design track. The ”ultra low latency” part of the URLLC goal was achieved by breaking up the established 4G resource grid, making it more flexible and capable of supporting short round-trip-times, for example by introducing so called mini-slot scheduling. Also a grant-free transmission enables fast access to radio resources without prior dedicated scheduling. Apart from the optimization of the radio link, leveraging edge computing in the radio access network is essential to realize low latency communication. While those measures (and some more not discussed here for lack of space) lead to a reduction of round-trip times ”on average”, the control of the associated distribution function of latencies is still an open question. Traditionally, radio communications faces packet losses due to mobility and dynamically changing channels. Such interference is mitigated with measures, which often contradict the low latency goal, such as repeated or duplicated transmissions and many forms of redundancy, which comes along with increased processing effort and delay. So while lower latencies and high reliability can be achieved individually by different measures, the combination of both is most challenging and demands eventually large network resources.

One important step in achieving URLLC services in an efficient way is to separate other types of 5G services in network slices, which provide independent, virtualized network resources with specialized services characteristics [18]. One network slice may for example serve high data rate streaming for consumers, while another network slide supports URLLC-type of service with ultra-high QoS demands (see figure 4) . While the basic service characteristics of a network slices can be easily described, the network operator faces the challenge to allocate the appropriate amount of network resources. A very simple approach of allocating static resources in terms of spectrum and time to each network slice which may accommodate any possible worst case may prove to be highly inefficient and not feasible, given the limited and highly expensive spectrum resources. Therefore the network operator needs to carefully explore the opportunities for allocating network resources to network slices in a flexible way, in order to balance out the demands: while the ”emergency lane” (i.e. URLLC slice) can be used by ”normal users” (streaming consumer slice) in ”normal” operation mode, it needs to be cleared instantaneously in case it is needed. To manage such extremely time-critical reallocation of resources, some resources may need to be reserved even if they are not used. On the other hand, will the immediate need for URLLC resources often be limited in terms of time and space. Therefore, highly flexible, spatio-temporal resource allocations are required to implement a network slice resource-efficiently [19]. Within the German collaborative research project 5GAIN[3], it is currently explored, how such resource allocations can be supported by leveraging external data sources and machine-learning. For example: in case the weather forecast predicts very strong winds, a network slice dedicated to the control of wind farms may be dimensioned more generously in order to be well prepared for transmission of time-critical control messages.

B. Network Convergence

A common trend observed likewise in the wired and wireless communication domain is the shift towards highly flexible network architectures suitable for a broad variety of applications with highly diverse requirements. Until now, network deployments have followed a rather hierarchical structure with dedicated networks for different classes of applications. Especially in the wired case, a considerable number of networking technologies exist for various purposes, ranging from real-time communication system with tight synchronization to throughput-optimized networks for high-bandwidth applications. The goal is to enable network convergence by sharing a single networking resource for all the various types of applications. This flexibility however comes at the price of increased configuration and management complexity. Furthermore, guaranteeing service requirements for different applications, especially those with tight latency bounds, is a demanding task. A prominent example technology enabling network convergence is Time-sensitive Networking (TSN). TSN is a set of standards mostly defined in the IEEE 802.1Q [20] designed to support various configurations models, traffic shapers, network synchronization and filtering mechanisms, which can be configured depending on the system requirements. The most challenging part is the isolation

[3]https://url.tu-dortmund.de/5Gain

of critical and non-critical traffic, as all applications have to share the same transmission resources. Unlike in 5G, where sophisticated spatio-temporal sharing over multiple domains can be exploited, the Ethernet-based TSN system is restricted to TDMA techniques. Guarantees are achieved by employing reservation-based methods where users are bounded by e.g. specific time windows (time-aware shaper) or coarser bandwidth limits (credit-based shaper).

This network convergence trend can be also observed in the wireless domain for the 5G technology, as shown in the previous section. 5G relies on the novel concept of network slicing to enable the concurrent support of latency-sensitive and elastic traffic over the same network. A logical next step is to extent the network convergence concept also across the wired and wireless domain. Recently, there have been strong efforts to combine TSN and 5G in a joint framework that has been partially standardized in the 3GPP [21]. This step can be motivated from mainly two perspectives. On the one hand, both technologies are inherently different and very challenging to configure and manage on their own to support all applications with their respective service level agreements. To feasibly operate such a complex network, the integration into a single framework eases operation and configuration. On the other hand, a transparent integration architecture is required in order achieve end-to-end services guarantees over the entire network infrastructure. In such an integrated framework, the centralized TSN configuration model (Figure 5) can be leveraged to jointly manage the wired and wireless network in a true end-to-end fashion, which is a prerequisite to enable end-to-end IoT applications.

IV. Conclusion

In this paper, we provided an overview of some of the approaches used for guaranteeing QoS in the embedded and wired/wireless communication domains. Several similarities are observed in the different domains, mainly on the need of providing efficient and predictable mechanisms for providing guaranteed latencies for critical applications, as well as the ability to handle different classes of (mixed criticality) traffic. We also deem further cross-pollination of concepts between the domains extremely helpful in order to advance the state of the art.

However, there is a need for an overarching framework combining different approaches and providing a cross-domain unified solution. One major challenge is the integration of the different technology stacks covering different resources, protocols, etc. On the other hand, even more needed is a common abstraction which allows designers to declaratively specify their application requirements and support the end-to-end configuration of the different QoS mechanisms without requiring the designer to have expert knowledge across all technologies. We see that considerable research effort will have to be spent in order to enable industry to build distributed IoT applications with end-to-end QoS guarantees.

Acknowledgement

This work has been supported by the European Union's Horizon 2020 research and innovation program under grant agreement No 871669.

This work was also partially supported by the Ministry of Economic Affairs, Innovation, Digitalisation and Energy of the state of North Rhine–Westphalia in the course of the Competence Center 5G.NRW under grant number 005–01903–0047, as well as by the "5GAIN" project, funded by the Federal Ministry for Economic Affairs and Energy (BMWi) under grant agreement no. 03EI6018C.

References

[1] *ROS2 - Robotic Operating System 2*, last access April 2020. Available at https://index.ros.org/doc/ros2/.

[2] *MQTT - Message Queuing Telemetry Transport*, last access April 2020. Available at http://mqtt.org/.

[3] D. Casini, T. Blaß, I. Lütkebohle, and B.B. Brandenburg. Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, 2019.

[4] *OFERA Project: Open Framework for Embedded Robot Applications*, last access April 2020. Available at http://ofera.eu/.

[5] *OFERA Deliverable D4.4 - Real-time Executor Software Release Y1*, last access April 2020. Available at http://ofera.eu/storage/deliverables/M12/OFERA_40_D4.4_Real-time_executor_software_release_Y1.pdf.

[6] S. Saidi, S. Steinhorst, A. Hamann, D. Ziegenbein, and M. Wolf. Special session: Future automotive systems design: Research challenges and opportunities. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–7, 2018.

[7] J. K. Strosnider, J. P. Lehoczky, and Lui Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[8] L. Abeni, G. Lipari, and J. Lelli. Constant bandwidth server revisited. *SIGBED Rev.*, 11(4):19–24, January 2015.

[9] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proceedings 21st IEEE Real-Time Systems Symposium*, pages 295–304, 2000.

[10] R. Cavicchioli, N. Capodieci, and M. Bertogna. Memory interference characterization between CPU cores and integrated GPUs in mixed-criticality platforms. In *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–10, 2017.

[11] *ARM Architecture Reference Manual Supplement: Memory System Resource Partitioning and Monitoring (MPAM) for Armv8-A*, last access April 2020. Available at https://static.docs.arm.com/ddi0598/a/DDI0598_MPAM_supp_armv8a.pdf.

[12] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 55–64, 2013.

[13] J. Leguay, L. Maggi, M. Draief, S. Paris, and S. Chouvardas. Admission control with online algorithms in sdn. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 718–721, 2016.

[14] A. Kostrzewa, S. Saidi, L. Ecco, and R. Ernst. Dynamic admission control for real-time networks-on-chips. In *21st Asia and South Pacific Design Automation Conference, ASP-DAC 2016*, pages 719–724. IEEE, 2016.

[15] Adam Kostrzewa, Selma Saidi, Leonardo Ecco, and Rolf Ernst. Ensuring safety and efficiency in networks-on-chip. *Integr.*, 58:571–582, 2017.

[16] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee. PRET DRAM controller: Bank privatization for predictability and temporal isolation. In *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 99–108, 2011.

[17] Y. Ye, R. West, Z. Cheng, and Y. Li. COLORIS: A dynamic cache partitioning system using page coloring. In *23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 381–392, 2014.

[18] L. U. Khan, I. Yaqoob, N. H. Tran, Z. Han, and C. S. Hong. Network slicing: Recent advances, taxonomy, requirements, and open research challenges. *IEEE Access*, 8:36009–36028, 2020.

[19] C. Bektas, S. Bocker, F. Kurtz, and C. Wietfeld. Reliable software-defined ran network slicing for mission-critical 5g communication networks. In *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019.

[20] IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks. *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pages 1–1993, July 2018.

[21] 3GPP TS 23.501. System Architecture for the 5G System (Release 16). December 2019.