Response Time for IEC 61499 over Ethernet

Per Lindgren, Johan Eriksson, Marcus Lindner and Andreas Lindner
Luleå University of Technology
Email:{per.lindgren, johan.eriksson, marcus.lindner, andreas.lindner}@ltu.se

David Pereira and Luís Miguel Pinho
CISTER / INESC TEC, ISEP
Email: {dmrpe, lmp}@isep.ipp.pt

***Abstract*—The IEC 61499 standard provides means to specify distributed control systems in terms of function blocks. The execution model is event driven (asynchronous), where triggering events may be associated with data (and seen as a message). In this paper we propose a low complexity implementation technique allowing to assess end-to-end response time of event chains spanning over a set of networked devices. In this paper we develop a method to provide safe end-to-end response time taking both intra- and inter-device delivery delays into account. As a use case we study the implementation onto (single-core) ARM-cortex based devices communicating over a switched Ethernet network. For the analysis we define a generic switch model and an experimental setup allowing us to study the impact of network topology as well as 802.1Q quality of service in a mixed critical setting. Our results indicate that safe sub millisecond end-to-end response times can be obtained using the proposed approach.**

I. Introduction

Traditional SCADA control systems typically imply the use of costly PLCs communicating over field buses, such as legacy Modbus[1]/PROFIBUS[2] and the more recent Ethernet based POWERLINK[3], EtherCAT[4] and PROFINET[2] buses. In order to establish end-to-end timing behaviour much attention has in general to be paid to the control system design, partitioning, and deployment. In this paper we take an alternative approach based on the recent IEC 61499 standard [5].

In contrast to the traditional scan (time triggered) based PLC behaviour (as implied by the IEC 61131 standard [6]), the IEC 61499 undertakes an event triggered (asynchronous) execution model. In the presented work we exploit the benefits of asynchronous execution both at device and network layer. In particular asynchronous execution allows at device level preemptive execution of the function block network, and at network layer, asynchronous communication over standard Ethernet (with optional 802.1Q quality of service) frames, as implemented in Commercial Off-The-Shelf (COTS) as well as rugged (industrial) Ethernet switches. In order to establish safe end-to-end timing properties of distributed IEC 61499 models, the presented work covers the execution model, requirements for the per device scheduling, a generic Ethernet switch model, and impact of general purpose traffic. This allows us to reason on timing properties under different network topologies, link speeds, and quality of service configurations.

Our results indicate that the proposed approach is capable of safe sub millisecond end-to-end response times even using cost efficient COTS components. In particular, we show that, given the assumption switches implement predictable 802.1Q quality of service, the impact of non-critical traffic can be bound, and safe response times obtained. This allows for flexible deployment, where the same physical network is shared in between the timing critical control applications and general purpose usage. Our results are safe to the worst case, while further improvements can be obtained if the Maximum Transmission Unit (MTU) of general purpose (best effort traffic) is controlled.

We can conclude that the presented approach reduces complexity of distribution (in comparison to the traditional SCADA approach), reduces device and network complexity (does neither require complex time triggered architectures like scan based PLCs, nor any dedicated network components/protocols), and is highly flexible (allowing network sharing between real-time and best-effort devices). Moreover, the presented approach opens up for the possibility to automate the generation of interfaces (SIFBs) interconnecting FB networks, for which correct-by-construction data delivery can be ensured. Moreover automation would allow the designer to focus on properties of the application at model level, independent on networking and deployment specifics.

II. Background

A. Task and Resource model

The task and resource model undertaken for this presentation follows that of the Stack Resource Policy (SRP) [7]. For this presentation we use task instance and task interchangeably. In short the SRP task and resource model is captured in the following:

- t_i is a *task* with run to completion semantics,
- $p(t_i)$ is the static priority for t_i used for scheduling,
- e_i is an event triggering the execution of the corresponding task t_i,
- r_j defines a resource, a task may claim resources in a LIFO nested manner.

SRP provides pre-emptive, deadlock free, single-core scheduling for systems with shared resources (in this context, claiming a resource can be seen as entering a critical section). Given WCETs for the tasks and their inner claims, response time analysis is readily available[7].

The RTFM-kernel has been developed from the outset of the SRP task and resource model [8]. The kernel API amounts to C code macros, with predicable and low overhead (typically a few machine instructions onto the ARM-Cortex family of processors). It extends the task model with the ability to emit events (hence task chains can be expressed). Under the common assumption that the inter-arrival of triggering events is larger than the associated task response time, each task can be associated with an interrupt handler, the underlying interrupt

hardware can be exploited as (a single unit) event buffer[1], and critical sections can be enforced through interrupt masking. In such a way the actual scheduling is performed by the interrupt hardware at a large.

B. IEC 61499 to RTFM-core Task and Resource model

In previous work, a mapping from device/resource level IEC 61499 models have been proposed [9]. At the edges of the IEC 61499 resources, Service Interface Blocks (SIFBs) are required for inter resource communication. The IEC 61499 standard does not cover the implementation of SIFBs. For this work we propose RTFM-core as the means of implementation. This brings the advantage that the complete system (at device level) can be implemented in a uniform manner, allowing the SIFBs to be part of the device level scheduling without introducing any additional/external run-time system.

1) System model example:

a) IEC 61499 System model: Figure 1 depicts an IEC 61499 model developed in the 4DIAC IDE [10]. The SIFB instances `Ea1` and `Ec1` capture the external events and trigger the execution of actions associated to `a_1.i_1` and `c_1.i_1` respectively. Figure 2 depicts the ECC of `b1_b3`, showing the associated actions `taskb1` for input event `i1` and `taskb3` for input event `i3` respectively.

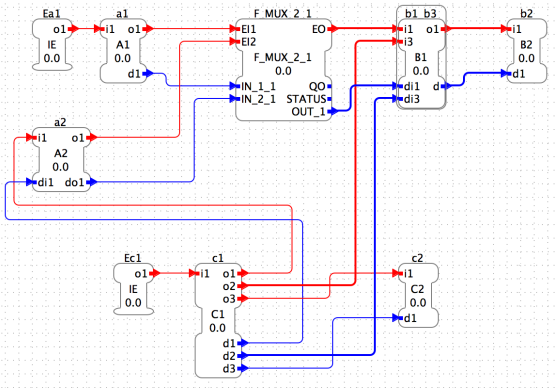

Figure 1. Example IEC 61499 system model.

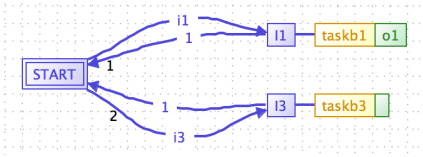

Figure 2. ECC of FB b1_b2.

b) RTFM System model: Figure 3 depicts an example RTFM system model expressing the same behaviour as the example model from Figure 1.

In RTFM-core the system is expressed in terms of event triggered tasks with named critical sections. The external event e_{a_1} triggers the execution of a_1, which in turn triggers the execution of b_1, which in turn trigger b_2. For part of the execution b_1 and b_3 enters a named critical section r in order to get exclusive access of the shared resource (in this case a data structure). This corresponds to the sequential behaviour of the ECC in b1_b3. While each link in -core is treated as message (i.e., event with associated data) there is no need for explicit data wiring, and multiplexing (F_MUX_2_1) rendering a succinct model.

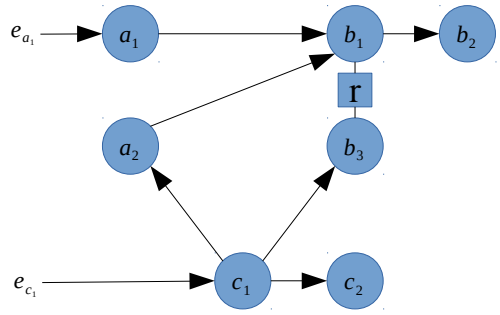

Figure 3. Example RTFM system model. We have the (external) events e_{a_1} and e_{c_1}, triggering the task chains headed by a_1 and c_1 respectively. Tasks b_1 and b_3 share a resource under the protection of a *critical section* r.

C. Monolithic single-core deployment

For deployment onto single-core platforms, the `rtfm-compiler` derives the task set from the input program and performs SRP analysis (i.e., derives the resource ceilings on basis of the per-task resource utilisation). The generated code compiled together with the (target specific) RTFM-kernel primitives render an executable for the deployment. The kernel primitives implement the SRP scheduling policy by exploiting the underlying interrupt hardware. (For the ARM-Cortex family of processors, the scheduling overhead is bound to a few clock cycles [8].) Given Worst Case Execution Time (WCET) for tasks (and their critical sections) we can deploy available SRP based methods for deriving per task response times [7].

1) Monolithic response times: The end-to-end response times for a task chain can be safely estimated by accumulating the response times along the chain. E.g., for the example system the response time of the task chain $e_{a_1} \rightarrow a_1 \rightarrow b_1 \rightarrow b_2$ is given as $rt(a_1) + rt(b_1) + rt(b_2)$.

D. Ethernet technology

Ethernet based buses are emerging in industrial applications as a cost and performance efficient alternative to proprietary field buses.

In the following we briefly review a set common topologies. (Messages in the figures relate to the distribution example later introduced, Section III.) Figure 4 depicts point to point topologies. In legacy systems, a single coaxial cable was used (left in Figure 4), using a collision detection and resend mechanism, for which bound transmission times are hard/impossible to guarantee. Alternatively, collision free communication is possible by using multiple interfaces and dedicated (point to point) cabling, however such solutions are costly and bulky (right in Figure 4).

A switched (full duplex) star (single layer) network mapping of the system is given in Figure 9. Figure 11, depicts the

[1]The 1-buffer assumption can be relaxed by user defined buffers, for which error handling semantics can be tailored to the application.

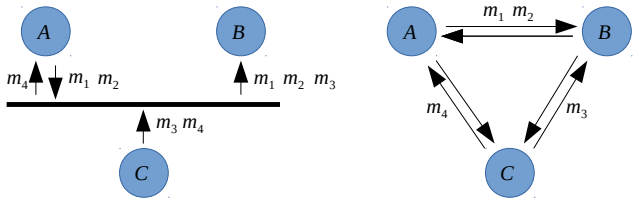

Figure 4. Point to point topologies, CSMA/CD collision detect and resend (left), collision free full duplex (right). Each link is marked with the associated messages m.

scenario of a layered network, with local switches S_1 and S_2 and a backbone switch S_3. In this case the network is shared between real-time traffic and best effort traffic.

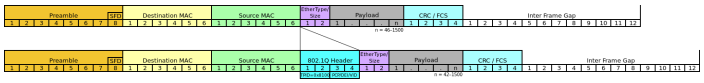

Figure 5. Embedding of 802.1Q (VLAN) into an Ethernet frame (source Wikipedia, author Bill Stafford).

1) Ethernet framing: An Ethernet frame (Figure 5, top) has a payload of 46 to 1500 bytes (octets). A 802.1Q (VLAN) frame (Figure 5, bottom) has a payload of 42 to 1500. In both cases this yields a minimum packet size of 64 bytes (including the header and CRC/FCS). The large minimal size packages is a legacy from non-switched (shared media) networks, for which the collision detection (CSMA/CD) mechanism is dependent on the propagation delay of the signal.

Including Preamble (8 bytes) and Inter Frame Gap (12 bytes), the total size of a frame size is 84 bytes. The transmission time for a complete minimal frame over a link is hence $84 * 8/ls$ (where ls is the link speed), amounting to $67.2/6.72/0.672$µs over 10Mbit/100Mbit/1Gbit/s links respectively. This gives the minimum period for transmissions over a link (and hence the maximum blocking time inferred by tramission of a minimal sized packet). Considering only packet delivery we can exclude the Inter Frame Gap (12 bytes), $57.6/5.76/0.576$µs over 10Mbit/100Mbit/1Gbit/s links respectively.

E. Ethernet based field buses

The term Industrial Ethernet (IE) is commonly used, referring to rugged components relying on Ethernet technology.

A gamut of Ethernet based field buses (such as POWERLINK [3] and EtherCAT [4]) have been introduced, for an overview see e.g., [11]. In common they have a focus on predictable timing, hence they are collectively referred to as Real-Time Ethernet.

III. Distributed systems in RTFM-core

In the following we discuss distribution aspects of RTFM-core models from a generic perspective, allowing -core models to be derived from corresponding IEC 61499 models, written directly in the RTFM-core language[12], or any mix thereof.

A. Distributed single-core deployment

In a distributed setting, the task set (and associated) resources are partitioned onto a set of devices. For this presentation we make the assumption that resources cannot be shared among tasks executing on different devices.

Figure 6 depicts a deployment scenario associating the tasks $\{a_1, a_2\}$ to device A, $\{b_1, b_2, b_3\}$ to B and $\{c_1, c_2\}$ to C. The messages (events with optional payload) crossing devices borders are marked m_i, e.g., $m_1 = a_1 \rightarrow b_1$ indicates an event (message) from task a_1 (on device A) triggering b_1 (on device B).

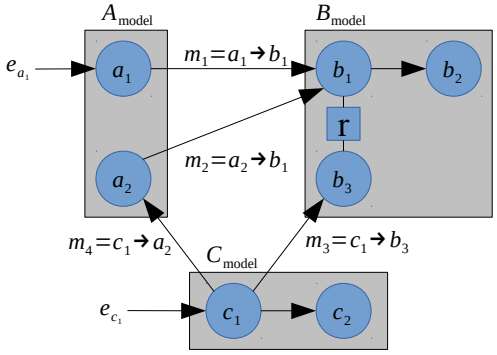

Figure 6. Distributed system model.

B. End-to-End Response Time, a holistic approach

A safe end-to-end response time estimate of a task chain spanning multiple devices can be computed by the accumulated sum of intra-device response time and the inter-device delivery times.

As a use case, we study Switched Ethernet, the impact of network topologies, 802.1Q quality of service configurations and the sharing of physical links between real-time and best effort domains. For the intra-device response time we rely on the previous discussion (Section II-C).

C. Holistic device level response times

In our setting we consider device drivers as a part of the application at device level, and thus allow us to assess intra device response times in a holistic manner.

Figure 7, depicts the device level task set for node A including the Ethernet drivers A_{RX} and A_{TX} with associated resources (buffers) r_{RX} and r_{TX}. Figure, 8 depicts the task set for device B, having only a receiving Ethernet driver B_{RX} with the associated receiving resource (buffer) r_{RX}.

The ARM-Cortex M3 based LPC1769 micro-controller provides an on chip Ethernet MAC (EMAC) with DMA functionality triggering the event $e_{a_{RX}}$ when a buffer has been received. The LPC1769 bus matrix allows the EMAC DMA to read and write package data using a dedicated SRAM allowing interference to be isolated to reading out/setting up package payload. This amounts for task a_{RX} and function fa_{TX} to a few shared memory accesses, for which worst case latency can be deduced from the data sheets. Under the assumption that messages will be consumed before reoccurring, a single (non-protected) buffer for each unique message is sufficient,

hence accesses to r_{RX} and r_{TX} do not require critical sections (hence marked white in figure). However, since the function fa_{TX} can be executed (preemptively) on behalf of both tasks a_1 and a_2 it's operations on the EMAC must be protected in a critical section (marked blue).

The EMAC transmit and receive functionality operates on circular description tables using producer and consumer indexes.

a) Transmit: The operation of fa_{TX} amounts to writing the message payload into the dedicated send buffer (in the shared SRAM), allocating a descriptor (round robin), update its packet pointer and size fields, and increasing the `TxProducerIndex` modulo `TxBufferNumber`.

b) Receive: The operation of task b_{RX} (triggered by the EMAC when `RxConsumerIndex` != `RxProducerIndex`) amounts to reading the destination port number and triggering either task b_1 (messages m_1 and m_2) or task b_3 (message m_3) with the corresponding payload as argument.

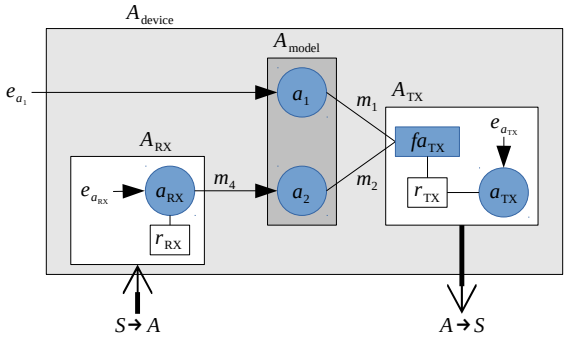

Figure 7. Device A with A_{RX} and A_{TX}. S is the network, typically a switch.

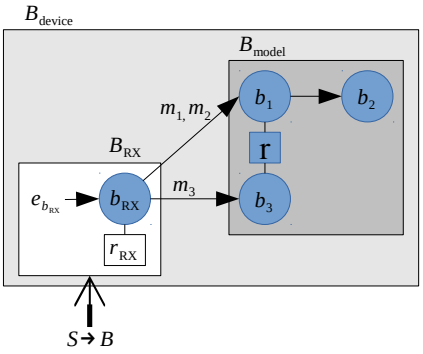

Figure 8. Device B with B_{RX} driver.

1) Device Level Analysis: Assuming that the size (in bytes) of payloads are given as $|m_1| = 4$, $|m_2| = 4$, $|m_3| = 8$ and $|m_4| = 4$ in the following, we have for device A the device level response times:

$$\begin{aligned} D_{rp}(e_{a_1}) &= rp(a_1), \\ D_{rp}(e_{a_{RX}^{m_4}}) &= rp(a_{RX}^{m_4}) + rp(a_2) \end{aligned}$$

Notice that tasks a_1 and a_2 includes the execution for the queuing function fa_{TX}. For device B we have the following:

$$\begin{aligned} D_{rp}(e_{b_{RX}^{m_1}}) &= rp(b_{RX}^{m_1}) + rp(b_1^{m_1}) + rp(b_2) \\ D_{rp}(e_{b_{RX}^{m_2}}) &= rp(b_{RX}^{m_2}) + rp(b_1^{m_2}) + rp(b_2) \\ D_{rp}(e_{b_{RX}^{m_3}}) &= rp(b_{RX}^{m_3}) + rp(b_3) \end{aligned}$$

For the scheduling analysis, we have the task instances $b_1^{m_1}$ and $b_1^{m_2}$ and $b_{RX}^{m_1}$, $b_{RX}^{m_2}$ and $b_{RX}^{m_3}$. In this way we can take into account payload size WCET dependencies for the receiver task b_{RX}. In this case $WCET(b_{RX}^{m_1}) = WCET(b_{RX}^{m_2})$ since the massages are of same length, while $WCET(b_{RX}^{m_3})$ has a different (larger) WCET due to the copying of the larger payload.

D. Link Layer Model

In order to assess the impact of networking we first study a simple (directly connected) point-to-point topology, that allow us to isolate the link-layer, Figure 4 (right). For the discussion we consider only nodes A and B (hence for each device a single network interface suffices).

1) Output queueing: The actual scheduling of the output link is performed by the function fa_{TX}. For our implementation we deploy a simple FIFO mechanism, where packets are queued in order of arrival to the task fa_{TX}. I.e. in the worst case, a message will be blocked/interfered by all other outgoing messages. Assume that we have n outgoing messages, and $tr(m_j)$ is the transmission time for a message m_j, then the blocking time is given as:

$$b(m_i) = \sum_{j=1}^{n} tr(m_j), i \neq j$$

The total time for delivery includes the transmission time of m_i, giving us a total delivery time (or network delay) over a link:

$$l(m_i) = b(m_i) + tr(m_i) = \sum_{j=1}^{n} tr(m_j)$$

In the case all messages fit the minimum Ethernet payload (46/42 bytes for Ethernet II/802.1Q respectively), we have a common $d(m) = tr(m) * n$, which as seen in Section II-D amounts to $n * 67.2/6.72/0.672$µs over a 10 Mbit/100 Mbit/1Gbit/s link respectively.

One could think of improvements to the above queuing mechanism, either by allowing non standard framing (reducing the transmission time), or by priority queuing.

2) Input queueing: On the receiver side, packets arrive in the order defined by the output queueing discussed above. The propagation delay over the link l be neglected for most practical cases.

E. Response time in a point-to-point network

We can now derive a safe response time for task chains spanning multiple devices over a point-to-point Ethernet network.

Let T be a global set of tasks, and $|T|$ the size of the set, M be a global set of messages, and $|M|$ its size. Let $TC(e)$ be a set of tasks triggered by the event e (i.e., tasks along the task chain headed by the event e). Let $MC(e)$ be a set of messages along the task chain headed by the event e.

For a triggering event e passing the devices D and the links L we have:

$$\begin{aligned} D_{rp}(e) &= \sum_{i=1}^{|T|} rp(t_i), t_i \in TC(e) \\ L_{rp}(e) &= \sum_{i=1}^{|M|} l(m_i), m_i \in MC(e) \\ E_{rp}(e) &= D_{rp}(e) + L_{rp}(e) \end{aligned}$$

$D_{rp}(e)$ being the accumulated response times over all devices, $L_{rp}(e)$ the accumulated link delays and $E_{rp}(e)$ the total end-to-end response time (delay).

Taken the running example, we have the chain:

$$e_{a_1} \rightarrow a_1 \rightarrow m_1 \rightarrow e_{TX} \rightarrow b_1^{m_1} \rightarrow b_2$$

with:

$$\begin{aligned} D_{rp}(e) &= rp(a_1) + rp(b_1^{m_1}) + rp(b_2) \\ L_{rp}(e) &= l(m_1) \\ E_{rp}(e) &= rp(a_1) + rp(b_1^{m_1}) + rp(b_2) + l(m_1) \end{aligned}$$

Given the assumption that the inter-arrival time of e is larger than $E_{rp}(e)$, we can exclude the blocking and interference by tasks t_j along the same chain ($t_j \in TC(e)$) for the computation of $rp(t_i)$ (at device level). The same reasoning goes for the link delays, where messages belonging to the same chain cannot block each other.

However, for the given example, the result would be the same, but for cases when the task chain traverses the same nodes (and links) multiple times a tighter, yet safe, estimate can be obtained.

F. Switch Model

In order to derive the response time over switched network, we first have to define a model of the switch at hand. We take the outset that switches are non-blocking at link speed, and able to store and forward all packages as long as the output bandwidth (for each port) is sufficient.

Figure 10 gives an architecture outline; incoming frames are buffered and inserted (by reference) into the corresponding output queue (or queues in case of multi-cast). The frame (if any) referenced at the head of each output queue is transmitted from the input queue, through the switching fabric, to the output port.

The output queuing mechanism may also vary: FIFO queuing, i.e., merging streams in order of arrival; 802.1Q quality of service, i.e., merging streams according of QoS/PCP priority. (Actual implementations may feature extended QoS, based on VLAN priority groups, and/or stream management using double tagging inside service provider networks).

G. Response time in a switched star network

Taking the outset of our example, Figure 6, and the topology Figure 9, we take a closer look at the end-to-end response time for the task chains headed by e_{c_1}.

(a) $e_{c_1} \rightarrow c_1 \rightarrow m_4 \rightarrow a_{RX} \rightarrow a_1 \rightarrow m_4 \rightarrow b_{RX} \rightarrow b_1 \rightarrow b_2$
(b) $e_{c_1} \rightarrow c_1 \rightarrow m_3 \rightarrow b_{RX} \rightarrow b_3$
(c) $e_{c_1} \rightarrow c_1 \rightarrow c_2$

For chain (a), we find that the messages m_2 and m_4 passes two links (indexes above show the order of delivery). Each link delay can be derived analogously to Section III-D, taking into account the queueing characteristics and scheduling overhead of the switch as discussed in previous section. The end-to-end response time can then be derived analogously to Section III-E.

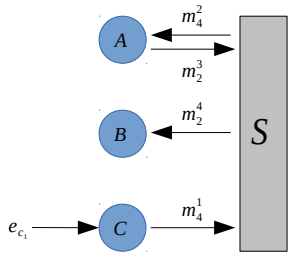

Figure 9. e_{c_1} chain (s) implies 4 transmissions, m_4^1, m_4^2, m_2^3, and m_2^4.

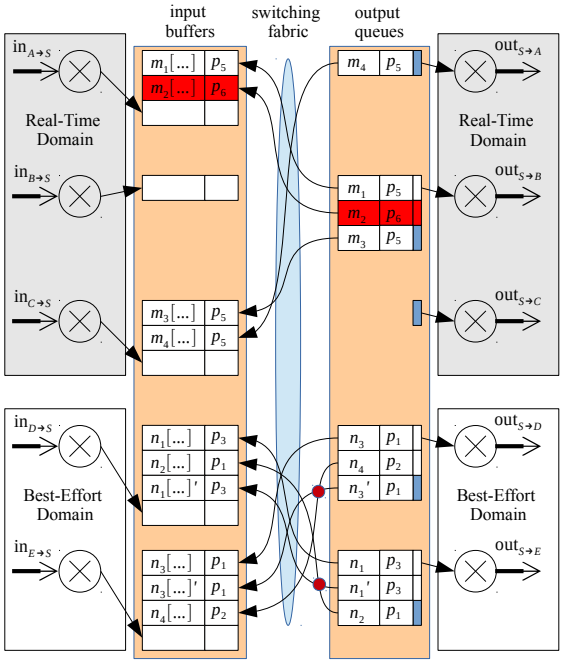

Figure 10. Generic Switch model. Reordering due to QoS is marked red/filled.

H. Switched tree model

Figure 11 depicts a tree network topology with edge switches S_1 and S_2 and a backbone switch S_3. The ports connecting to the backbone switch are so called *trunks*, merging traffic from different streams. In this case the links to/from S_3 convey mixed traffic from the real-time and and best effort domains, exposing the messages m_4^2 an m_4^3 to (potentially unbound) blocking and queueing overhead.

I. Response time in a switched tree network

Taking the outset of our running example, we take a closer look at the end-to-end response time for the task chains headed

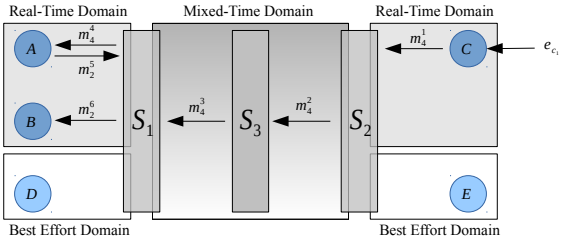

Figure 11. e_{c_1} (a) implies a transmission chain $m_4^{1,2,3,4}$, and $m_2^{5,6}$.

by e_{c_1}. The communication for (a) is depicted in Figure 11, indexes above show the order of delivery:

$$e_{c_1} \rightarrow c_1 \rightarrow m_4^{1,2,3,4} \rightarrow a_{RX} \rightarrow a_2 \rightarrow m_2^{5,6} \rightarrow b_{RX} \rightarrow b_1 \rightarrow b_2$$

The response time can be derived analogously to Section III-G, however we will need to take into account blocking and queuing overhead implied by best-effort traffic for the backbone (trunk) links.

IV. Experiments

A. Experimental Setup

1) Devices: Experiments have been conducted on the Embedded Artist (EA) LPCXpresso1769 development board running at 96MHz. The board features a LAN8720 PHY, connected to a 10/100 Base-Transformer (RJ45 jack) featured on the EA base board. (100Mbit per default for the experiments.)

2) Switches: The experiments were conducted using a set of modern Cisco Catalyst 2960 S switches (running OS version 12.2) [13]. The switch provides 4 output (egress) queues, where queue 1 has priority (expedited unless empty), other queues are expedited to share or shape remaining bandwidth under configurable Weighted Tail Drop (WTD) parameters.

3) Basic Setup: Figure 12 depicts the generic experimental setup, allowing us to observe device/link layer and switching overhead, as well as implications of network topology and interference by best effort traffic. An initial message (tagged with an index of 1) is produced by a_1. Upon receiving a correct package (w.r.t destination and index), device A generates a new message to device B with incremented index, while device B simply forward any incoming package (as received) to device A. Task a_2 toggles an digital output on each invocation (allowing external monitoring of the roundtrip time). The minimal Ethernet frame size (60+CRC/FCS) was deployed, out of which only 16 bytes (including MAC, EtherType/size, and index) were actually used.

4) 802.1Q Setup: For the experiments involving 802.1Q, frames from the real-time domain were tagged with the EtherType $0x8100$, PCP/CoS 7 (network), DEI 0 (non-drop), and VLAN 2, giving a frame size of 64[2].

The best effort traffic was generated (and received) on ordinary PCs (devices C and D) using the tools Ostinato 0.6 (for package generation) and Wireshark (for package analysis).

[2]According to Cisco interpretation of minimal frame size.

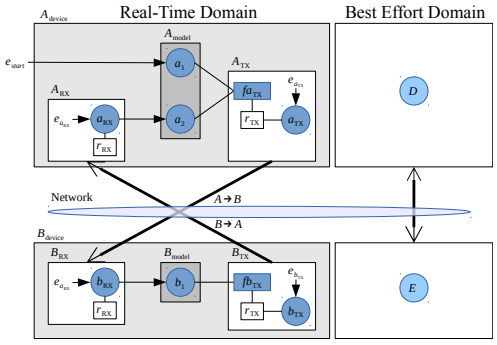

Figure 12. Generic experimental setup.

B. Experiment 1: Point-to-point network

For this experiment, devices A and B were directly connected. This shows the overhead of the link layer transmission time, EMAC hardware and device driver overhead. The round-trip, A (receiving a frame, extracting payload, verifying index, sending a frame) and B (receiving a frame, extracting payload, sending a frame with same payload), time was measured to 33µs (with a jitter of < 1µs). This experiment confirms the low link layer software and hardware overhead, indicating end-to-end response times in the range of 15µs.

C. Switched star network

For this experiment, devices A and B were connected over a (single) switch. This experiment focuses the Real-Time Domain (and isolates the effect of the switch on real time traffic). No additional devices were connected. Again the round-trip time was measured with the logic analyser, and measured to 55µs (with a measured jitter of < 1µs).

D. Switched single-layer network with best effort traffic

For this experiment, real-time devices (A and B) and best effort devices (D and E) were connected over a (single) switch. Real-time and best-effort traffic were separated by port (i.e., all real-time traffic is transmitted between devices A and B, while all best effort traffic is transmitted between devices D and E. For the experiment an average of 5000 (MTU 1518 bytes) packages/s was sent in between devices D and E (emulating a generic and heavy IP traffic load). To stress buffering effects, packages were sent in burst of 10 at the rate of 500 bursts/s). The measured round-trip time was still 55µs (with a measured jitter of < 1µs).

E. Switched tree network

For these experiments we tunnelled all traffic over a common backbone (trunk) with link speed set to 100Mbit (Ex4, ..., Ex6) and 1Gbit for Ex7. Two edge switches were used, connecting (A, D) and (B, E) respectively. In the first experiment (Ex4) only real-time traffic was emitted. A roundtrip delay of 103µs with a jitter of < 1µs was observed.

The second experiment (Ex5) contains both real-time and best-effort streams. The observed best case was 103µs, while the worst case 340µs was less than the theoretical bound $103 + 2 * tr(MTU) = 103 + 2 * 122 = 347$µs.

In the third experiment (Ex6) we reduce the best effort MTU to 64 bytes. Results show that the overhead of best effort traffic can be efficiently mitigated by controlling the MTU.

In the final experiment (Ex7) we observe the effect of a Gigabit backbone. This indicates that even without controlling the MTU of best effort traffic, Gigabit backbone efficiently mitigates the blocking overhead.

Ex1	Ex2	Ex3	Ex4	Ex5	Ex6	Ex7
P2P	Star	Star + BE 1518	Tree	Tree + BE 1518	Tree + BE 64	GB Tree + BE 1518
33µs	55µs	55µs	103µs	103:340µs	103:103µs	77:78µs

Table I. BE INDICATES THE PRESENCE (AND FRAME SIZE) OF BEST EFFORT TRAFFIC OVER THE SHARED LINKS. (ALL FRAME SIZES INCLUDE FCS/CRC.) GB INDICATES A GIGBIT BACKBONE.

V. EVALUATION AND DISCUSSION

Guaranteed real-time services over standard Ethernet has been studied from a modelling and simulation perspective (e.g., in [14]). Response time of IEC 61499 has been studied at device level e.g. [15], and over switched Ethernet e.g. [16]. Our models extends on [14], [15], [16] and our experiments indicate their validity. The problem of shared output queuing to QoS has been identified and discussed in [17]. In our case we need only two priority classes supported by the switch in order to provide real-time guarantees.

Our results are safe to the worst case, assessing end-to-end response in a holistic manner including the overhead of receive/transmit hardware and software. Our experiments show that the adversative effects of best effort traffic can be mitigated by controlling the MTU and/or faster backbone.

In cases when absolute performance requirements cannot be met by the presented approach, specialised solutions and technologies (e.g., EtherCAT) may be combined with the presented work, allowing to further reduce end-to-end response times, with (potentially) tighter bounds to the analysis.

Set target of future work is to further automate the design process for distribution, allowing networking components to be generated directly from 61499/RTFM models, and to integrate the proposed methods for analysis in the RTFM-4-FUN tool suit. In order to ensure correctness of the presented analysis and future automation features, rigorous formalisation and extraction of certified code is possible through e.g., Coq[18]. Another area of interest is deriving a generic test bed, allowing fast and accurate characterisation of switch configurations, facilitating the deployment into real-life environments.

VI. CONCLUSIONS

In this paper we have discussed end-to-end deadlines for distributed IEC 61499 applications communicating over switched Ethernet networks. The end-to-end analysis takes device level response time, network link layer transmission times, network topology as well as switch specifics (up to layer 2) into consideration. For the analysis and implementation, the system is translated into concurrent tasks and resources (critical sections) in RTFM-core language. The static nature of the RTFM-core model allows device level response times to be assessed using available schedulability analysis. For each link the number of (outstanding) messages in the system is precisely bound, thus allowing the blocking and queuing time to be safely estimated. To assess the impact of switches in the network a generic model is presented. Our experiments conducted on commercially available micro-controllers and Ethernet switches indicate that safe end-to-end response times well below 1ms are feasible, even over tree network topologies and in the presence of best-effort traffic.

ACKNOWLEDGEMENTS

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology), and the EU ARTEMIS JU funding, within project ARTEMIS/0001/2013, JU grant nr. 621429 (EMC2) and VINNOVA (Swedish Governmental Agency for Innovation Systems) and Svenska Kraftnät (Swedish national grid).

REFERENCES

[1] Modbus Organization. (webpage) Last accessed 2015-02-17. [Online]. Available: http://www.modbus.org

[2] PROFIBUS and PROFINET International (PI). (webpage) Last accessed 2015-02-17. [Online]. Available: http://www.profibus.com

[3] POWERLINK - Organization. (webpage) Last accessed 2015-02-17. [Online]. Available: http://www.ethernet-powerlink.org

[4] EtherCAT - The Ethernet Fieldbus. (webpage) Last accessed 2015-02-17. [Online]. Available: http://www.ethercat.org

[5] "International Standard IEC 61499: Function Blocks - Part 1, Architecture," Geneva, Switzerland: Int. Electrotech. Commission, 2012.

[6] "International Standard IEC 61131-3: Programmable Controller - Part 3, Programming Languages," Geneva, Switzerland: Int. Electrotech. Commission, p. 230, 1993.

[7] T. Baker, "A stack-based resource allocation policy for realtime processes," in *Real-Time Systems Symposium, 1990. Proceedings., 11th*, Dec. 1990, pp. 191 –200.

[8] J. Eriksson, F. Häggstrom, S. Aittamaa, A. Kruglyak, and P. Lindgren, "Real-Time For the Masses, Step 1: Programming API and static priority SRP kernel primitives." in *SIES*. IEEE, 2013, pp. 110–113.

[9] P. Lindgren, M. Lindner, V. Vyatkin, and et. al., "Real-Time Execution of Function Blocks for Internet of Things using the RTFM-kernel," in *Proceedings of 2014 IEEE 19th International Conference on Emerging Technologies & Factory Automation (ETFA 2014)*, 2014.

[10] 4DIAC - Organization. (webpage) Last accessed 2015-02-17. [Online]. Available: http://www.fordiac.org

[11] M. Felser, "Real-time ethernet - industry prospective," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1118–1129, June 2005.

[12] RTFM: Real-Time For the Masses. (webpage) Last accessed 2015-02-17. [Online]. Available: http://www.rtfm-lang.org

[13] Catalyst 2960 - Product Web. (webpage) Last accessed 2015-02-17. [Online]. Available: http://www.cisco.com/c/en/us/products/switches/catalyst-2960-series-switches/index.html

[14] X. Fan and M. Jonsson, "Guaranteed Real-Time Services over Standard Switched Ethernet," in *30th Annual IEEE Conference on Local Computer Networks (LCN 2005), 15-17 November 2005, Sydney, Australia, Proceedings*. IEEE Computer Society, 2005, pp. 490–492.

[15] L. Lednicki, J. Carlson, and K. Sandstrom, "Device utilization analysis for IEC 61499 systems in early stages of development," in *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*, Sept 2013, pp. 1–8.

[16] A. Schimmel and A. Zoitl, "Real-time communication for IEC 61499 in switched Ethernet networks," in *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on*, Oct 2010, pp. 406–411.

[17] R. Janowski, P. Krawiec, and W. Burakowski, "On assuring QoS in Ethernet access network," in *ICNS*. IEEE Computer Society, 2007, p. 17.

[18] The Coq Proof Assistant. (webpage) Last accessed 2014-08-06. [Online]. Available: http://www.lix.polytechnique.fr/coq/