# Stability-Aware Integrated Routing and Scheduling for Control Applications in Ethernet Networks

Rouhollah Mahfouzi[1], Amir Aminifar[2], Soheil Samii[1,3], Ahmed Rezine[1], Petru Eles[1], Zebo Peng[1]

[1]Embedded Systems Laboratory, Linköping University, Sweden
[2]Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
[3]General Motors R&D, Warren, MI, USA
{rohollah.mahfoozi, ahmed.rezine, petru.eles, zebo.peng}@liu.se, amir.aminifar@epfl.ch, soheil.samii@gm.com

***Abstract*—Real-time communication over Ethernet is becoming important in various application areas of cyber-physical systems such as industrial automation and control, avionics, and automotive networking. Since such applications are typically time critical, Ethernet technology has been enhanced to support time-driven communication through the IEEE 802.1 TSN standards. The performance and stability of control applications is strongly impacted by the timing of the network communication. Thus, in order to guarantee stability requirements, when synthesizing the communication schedule and routing, it is needed to consider the degree to which control applications can tolerate message delays and jitters. In this paper we jointly solve the message scheduling and routing problem for networked cyber-physical systems based on the time-triggered Ethernet TSN standards. Moreover, we consider this communication synthesis problem in the context of control applications and guarantee their worst-case stability, taking explicitly into consideration the impact of communication delay and jitter on control quality. Considering the inherent complexity of the network communication synthesis problem, we also propose new heuristics to improve synthesis efficiency without any major loss of quality. Experiments demonstrate the effectiveness of the proposed solutions.**

## I. Introduction

Due to growing bandwidth requirements, real-time communication over Ethernet is of increasing importance in embedded control and cyber-physical systems. Application domains include industrial automation and control, avionics, and active safety and automated driving applications [4], [6]. Several time-driven approaches like 802.1Qbv [12], developed by the IEEE 802.1 Time-Sensitive Networking (TSN) task group, and TTEthernet [20] exist to enhance standard Ethernet with real-time properties such as time determinism and packet delivery guarantees. The temporal properties of an Ethernet network depend significantly on the schedules and routes of the data flows along switches. Researchers have recently studied various design-space exploration problems to synthesize Ethernet schedules and routes in the context of hard deadlines and worst-case latencies. While this model is applicable to some real-time applications, it does not take into consideration temporal properties like average delay and jitter, which are known to impact control performance and stability in cyber-physical systems [1], [5], [6], [9], [13], [15], [16].

This research has been partially supported by the Swedish national strategic research area (project eLLIIT), the Swedish Research Council, the ONR-G Award Grant No. N62909-17-1-2006, and by the EC H2020 MANGO FETHPC project (Agreement No. 671668).

**Related Work.** In the context of TTEthernet networks with multiple hops, Steiner [18] proposed an approach based on Satisfiability Modulo Theory (SMT) to address the problem of schedule synthesis. This approach has been adapted in [7] to support the recent IEEE 802.1Q amendment for scheduled traffic (IEEE 802.1Qbv-2015 [12]). Time-triggered schedule synthesis has recently been extended in various directions, such as bandwidth optimization for best-effort traffic [21], mixed-criticality systems [19], resilience to link failures [3], traffic class assignment [11], and modeling of time-synchronization precision [22].

Tamas–Selicean et al. [21] considered scheduling, frame packing, and route selection separately in a Tabu Search heuristic. The proposed routing moves increase delay and jitter of time-triggered frames to meet their deadlines with as little time slack as possible, in order to increase schedulability and bandwidth, respectively, for rate-constrained and best-effort communication. The approach is customized to rate-constrained and best-effort traffic and is neither applicable nor generalizable to time-triggered Ethernet communication for hard real-time or control applications. Pop et al. [14] presented a design optimization framework for Ethernet with time-triggered (TT) and AVB traffic classes, with routing fixed and given by the shortest path for each TT data flow, leaving the entire TT routing optimization problem unsolved. Smirnov et al. [17] presented a 0–1 ILP formulation for routing and scheduling, where the network model is limited to bus-based, half-duplex communication with gateways. Their solution is limited to synthesis of collision-free schedules without any deadline constraints, and it merely determines whether or not pairs of messages share the same link.

**Contributions.** We address the joint Ethernet schedule–route synthesis problem for control applications communicating on a time-triggered Ethernet network. The novelty of our contribution lies in (1) an SMT formulation for simultaneous routing and scheduling, (2) consideration of real-time, control, and worst-case stability requirements, and (3) heuristics to improve synthesis efficiency based on route subsets and time slices. We demonstrate the effectiveness and efficiency of our proposed technique experimentally, in addition to its applicability in a real-life example from the automotive domain. To our knowledge, this is the first paper to consider control stability in a design-space exploration problem for Ethernet.

978-3-9819263-0-9/DATE18/©2018 EDAA

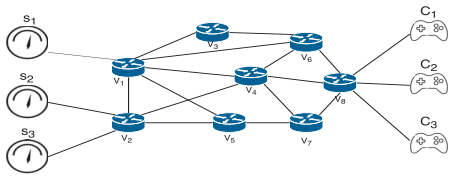


Fig. 1. An example System

## II. System Model

Our system comprises a network of Ethernet switches that is used as an infrastructure to connect sensors and controllers of $n$ control applications.

### A. Network and Ethernet Switches

We model the network and its topology as a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ where nodes $v_i \in \boldsymbol{V}$ are Ethernet switches, sensors, or controllers and the edges $[v_i - v_j] \in \boldsymbol{E}$ are full-duplex physical links between these nodes. Figure 1 shows an example of a network with 14 nodes consisting of 8 Ethernet switches that connects 3 sensors to 3 controllers.

Ethernet switches in the network implement the IEEE 802.1Qbv-2015 TSN standard [12]. According to this standard, each egress (output) port in a switch has up to 8 associated queues with strict-priority arbitration. Figure 2 shows an overview of an Ethernet TSN switch with 4 ports. In each output port up to 8 queues are dedicated for time-triggered, scheduled traffic. The rest of the queues (lower priority queues) are used for non-scheduled traffic [4]. All the incoming messages go through a *Forwarding Engine*, where the output port is decided based on a look-up table that captures the routing. In the output port, a dispatcher assigns the message to a proper queue based on its priority (the priority code point embedded in an IEEE 802.1Q Ethernet frame). Finally, for queues with scheduled traffic, the switch opens the timed-gate at the end of the queue, based on the predefined schedule, to transfer the message. If the time-gates for multiple queues are open at the same time, then the message in the higher priority queue is sent first. The output port and release time of messages is determined based on the forwarding and scheduling variables stored in the switch. Therefore, inside a switch we have two stored variables for each message.

- **Output port** for message $m_{i,j}$ arriving at switch $v_k$ is denoted by $\eta_{ijk}$.
- **Release time** of message $m_{i,j}$ arriving at switch $v_k$ is denoted by $\gamma_{ijk}$.

These variables are determined at design time using our routing and scheduling algorithm (Section V).

We highlight that the messages $m_{i,j}$ considered here are not the only ones traveling over the Ethernet network. Several other messages of lower criticality level are handled by the switches corresponding to their traffic classes. The time-triggered messages, which are the only ones of interest in this paper, are handled at the highest priority level, and their communication is not impacted by messages belonging to any of the other traffic classes.

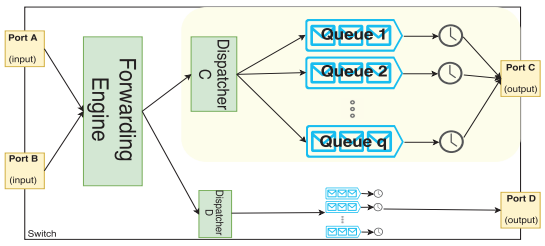


Fig. 2. Overview of an Ethernet TSN switch, showing the ingress portion of ports A and B, and the egress portion of ports C and D.

### B. Delay, Latency, and Jitter

A message experiences three types of delays in a switch:

- **Forwarding delay** is the time from receiving a message in a switch until the message is put in the outgoing queue[1]. Forwarding delay for a switch is denoted by $sd$.
- **Scheduling delay** is the time between putting a message in the proper queue of its outgoing port until the switch decides to start message transmission. This delay depends on the release time variable of the message on that specific switch, as determined at schedule synthesis.
- **Transmission delay** is the amount of time it takes to transmit the message on a link. The transmission delay is denoted by $ld$ and depends on the size of the message and the data-rate of the link[1].

The end-to-end delay for a message going from a sensor to a controller is the sum of forwarding, scheduling, and transmission delays along all switches in the route.

### C. Control Applications

We consider periodic control applications that are sensitive to jitter and latency. A control application $\Lambda_i$ consists of sensor $S_i$, controller $C_i$, and plant $P_i$. The sensor $S_i$ and the controller $C_i$ communicate using a network of Ethernet switches[2]. The plant $P_i$ is modeled by a continuous-time system of differential equation [2],

$$\dot{\boldsymbol{x}}_i = \boldsymbol{A}_i \boldsymbol{x}_i + \boldsymbol{B}_i \boldsymbol{u}_i, \tag{1}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and the control signal, respectively. The sampling is done periodically and the control signal is kept constant between two updates. Each sensor $S_i$ periodically samples, and sends a message to controller $C_i$. The controller uses this data to compute the control signal that is immediately applied to the plant by the actuator.

The sampling period of application $\Lambda_i$ is denoted by $h_i$. A series of message instances released in successive periods by the same sensor is called a *flow*. Flow $f_i$ originates from $S_i$ to controller $C_i$. The $j$-th message in this flow is denoted by $m_{i,j}$. Since all the control applications are periodic, exactly

[1] We consider that this delay is constant for all messages in all links and switches. Note that this assumption is only for simplifying the discussion in the rest of the paper and our approach and experimental results consider different delays depending on switch/message/link.

[2] We are considering this one-to-one sensor-controller correspondence only for the discussion in the paper, in order to keep the notation and equations readable. Our implementation does not impose this restriction.

the same events occur in each hyper-period. Therefore, we only need to consider the message instances inside one hyper-period. Consequently, the set $\mathcal{M}$ of messages considered for routing and scheduling in the following sections consists of all messages $m_{i,j}$ generated by the sensors inside a hyper-period (the LCM of periods).

## III. Problem Formulation

Given $n$ control applications and a network of Ethernet switches as described in Section II. We consider the following as input to our problem:

- the topology of the network, given by graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$;
- the switch forwarding delay, $sd$, and link transmission delay, $ld$; and
- for each control application $\Lambda_i$:
  - controller $C_i$ with period $h_i$, which is also the period of messages received by $C_i$ from sensor $S_i$.
  - model of the plant $P_i$;

Our goal is to schedule and route all messages in the network so that the control applications are guaranteed to be stable. Therefore, the outputs of our routing and scheduling algorithm are the release time (schedule) and output port (route) for each message in each Ethernet switch. These are captured by the values assigned to the following variables:

- output port $\eta_{ijk}$; and
- release time $\gamma_{ijk}$,

for each message $m_{i,j} \in \mathcal{M}$ arriving at switch $v_k$. These values are produced at design time and stored in switches.

## IV. Stability Analysis

In the context of networked systems, due to the shared resources, control applications may experience time-varying delays which may lead to instability, if not properly considered during design. The worst-case control performance of a system can be quantified by an upper bound on the gain from the uncertainty input (here, time-varying delays) to the plant output. In order to measure the worst-case control performance of a system in the presence of time-varying delays, we use the Jitter Margin toolbox [5]. The Jitter Margin toolbox provides sufficient conditions for the worst-case stability of a closed-loop system with a linear continuous-time plant and a linear discrete-time controller. These conditions are described using the latency and jitter experienced by the controller.

Considering a control application $\Lambda_i$, the latency $L_i$ is defined as the constant part of the delay experienced by the controller (in our case this is the total delay of the message received from the sensor). The worst-case jitter $J_i$ is defined as the variation in the delay. In the following, we elaborate the dependency between the stability of a control application and the latency and jitter experienced by its controller.

The Jitter Margin toolbox provides the stability curve that determines the maximum tolerable response-time jitter $J_i$ based on the latency $L_i$. The green curve in Figure 3 is an example of the *stability curves* generated by the Jitter Margin toolbox, where the area below the curve is the stable area. This solid curve is generated for a DC servo process with

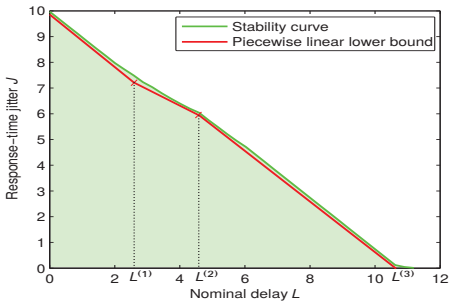


Fig. 3. The stability curve generated by Jitter Margin (below the curve, highlighted in green, is the stable area) and the piecewise linear lower bound.

transfer function $\frac{1000}{s^2+s}$ and a discrete-time Linear-Quadratic-Gaussian (LQG) controller, with a sampling period of 6 $ms$. The stability curve can safely be approximated by a piecewise linear (lower-bound) function of the latency and jitter, see the red piecewise linear function in Figure 3. The *stability condition*, hence, can be formulated as,

$$\text{stablility} = \begin{cases} L_i + \alpha_i^{(1)} \cdot J_i \leq \beta_i^{(1)} & \text{if } 0 \leq L_i \leq L_i^{(1)} \\ L_i + \alpha_i^{(2)} \cdot J_i \leq \beta_i^{(2)} & \text{if } L_i^{(1)} \leq L_i \leq L_i^{(2)} \\ \dots & \dots \\ L_i + \alpha_i^{(m)} \cdot J_i \leq \beta_i^{(m)} & \text{if } L_i^{(m-1)} \leq L_i \leq L_i^{(m)} \\ \textit{false}\text{: not stable} & \text{otherwise} \end{cases} \tag{2}$$

where $\alpha_i^{(j)}$, $\beta_i^{(j)}$, and $L_i^{(j)}$ are non-negative constants for the $j$-th linear segment of the piecewise linear lower-bound function, for control application $\Lambda_i$. Note that $L_i^{(m)}$ is the constant associated with the last segment of this piecewise linear function ($m = 3$ in Figure 3). For each control application $\Lambda_i$, we define the stability margin $\delta_i$ as follows:

$$\delta_i = \begin{cases} \beta_i^{(1)} - (L_i + \alpha_i^{(1)} \cdot J_i) & \text{if } 0 \leq L_i \leq L_i^{(1)} \\ \beta_i^{(2)} - (L_i + \alpha_i^{(2)} \cdot J_i) & \text{if } L_i^{(1)} \leq L_i \leq L_i^{(2)} \\ \dots & \dots \\ \beta_i^{(m)} - (L_i + \alpha_i^{(m)} \cdot J_i) & \text{if } L_i^{(m-1)} \leq L_i \leq L_i^{(m)} \\ -\infty & \text{otherwise} \end{cases} \tag{3}$$

A non-negative $\delta_i$ guarantees the stability of the system.

## V. Stability-Aware Routing and Scheduling

This section presents our approach for routing and scheduling of control messages with stability constraints. In particular, we present a stability-aware SMT formulation and efficient heuristics to find port assignments and release times for all messages and network switches. In the following, the problem constraints to be formulated using SMT are described.

### A. Constraints with respect to routing and scheduling

- **Topology constraint:** The selected output node should be among the nodes that are connected to the current node.

$$\begin{aligned} &\forall i,j | m_{i,j} \in \mathcal{M}, \forall k | v_k \in \boldsymbol{V} : \\ &\quad if \quad \eta_{ijk} = v_l \quad then \quad [v_k - v_l] \in \boldsymbol{E}. \end{aligned} \tag{4}$$

- **Contention-free constraint:** If two messages are waiting in the queues of the same port of a switch, then one

of them should wait until the other one is released into the link. We consider this waiting time equal to the transmission delay ($ld$) of that message.

$$\begin{aligned} &\forall i,j,r,s | m_{i,j}, m_{r,s} \in \mathcal{M}, \forall k | v_k \in \boldsymbol{V} : \\ &\quad if \quad \eta_{ijk} = \eta_{rsk} \quad and \quad (i \neq r \quad or \quad j \neq s) \\ &\quad then \quad |\gamma_{ijk} - \gamma_{rsk}| \geq ld. \end{aligned} \tag{5}$$

- **Transposition constraint:** The release time of a message should be after release time of the same message from the predecessor node plus the link transmission delay ($ld$) plus the switch forwarding delay ($sd$).

$$\begin{aligned} &\forall i,j | m_{i,j} \in \mathcal{M}, \forall k,l | v_k, v_l \in \boldsymbol{V} : \\ &\quad if \quad \eta_{ijk} = v_l \quad then \quad \gamma_{ijl} \geq \gamma_{ijk} + sd + ld. \end{aligned} \tag{6}$$

- **No-loop constraint:** For each message there should not be any two switches which forward the message to the same destination.

$$\begin{aligned} &\forall i,j | m_{i,j} \in \mathcal{M}, \forall k | v_k \in \boldsymbol{V}, \forall l | v_l \in \boldsymbol{V} : \\ &\quad if \quad k \neq l \quad then \quad \eta_{ijk} \neq \eta_{ijl}. \end{aligned} \tag{7}$$

- **Route constraint:** For each message there should be a route between the source (sensor) and destination (controller) of the control application that message relates to. $\boldsymbol{R}_i$ is the set of all possible routes from $S_i$ to $C_i$.

$$\begin{aligned} &\forall i,j | m_{i,j} \in \mathcal{M}, \exists \{S_i, v_{l_1}, v_{l_2}, \ldots, v_{l_p}, C_i\} \in \boldsymbol{R}_i : \\ &\quad (\eta_{ijS_i} = v_{l_1}) \wedge (\eta_{ijl_1} = v_{l_2}) \wedge \cdots \wedge (\eta_{ijl_p} = C_i). \end{aligned} \tag{8}$$

### B. Stability Constraints

The two parameters impacting stability of control applications are latency and jitter, as discussed in Section IV. In order to compute these two parameters, we consider the end-to-end delay of message $m_{i,j}$ from sensor $S_i$ to controller $C_i$, denoted by $e2e_{ij}$. This end-to-end delay might vary for different messages in the flow from $S_i$ to $C_i$ because of two reasons: (1) the route for each individual message might be different; (2) the scheduling and thus the scheduling delay for each individual message might be different. The end-to-end delay $e2e_{ij}$ is equal to the release time of that message from the last switch in the route, plus the $ld$ of the link between this switch and the controller minus the initial release time from the sensor. If we consider the best-case delay for the controller $C_i$ as $\min_j\{e2e_{ij}\}$ and the worst-case delay as $\max_j\{e2e_{ij}\}$, then the latency $L_i$ and jitter $J_i$ for control application $\Lambda_i$ in Equation 3 is computed as follows,

$$L_i = \min_j\{e2e_{ij}\}, \qquad J_i = \max_j\{e2e_{ij}\} - \min_j\{e2e_{ij}\}. \tag{9}$$

Given the model of the plant $P_i$ and controller $C_i$, we use the Jitter Margin toolbox to generate the piecewise linear lower bounds. Then, for each linear piecewise segment, we compute the $\alpha_i^{(j)}$ and $\beta_i^{(j)}$ parameters in Equation 3.

- **Stability constraint:** All control applications should be stable. Hence, The stability margins defined by Equation 3 should hold for all control applications.

$$\delta_i \geq 0, \quad \forall \Lambda_i. \tag{10}$$

We use Satisfiability Modulo Theory (SMT) to solve the problem captured by the above constraints. If the conjunction of these constraints is satisfiable, the solver associates a value to all $\eta_{ijk}$ and $\gamma_{ijk}$ for each message $m_{i,j}$ at each node $v_k$ of the network.

### C. Improving scalability

Routing and static scheduling for time-triggered messages are known to be NP-complete problems [7], [18], hence the scalability issue. Considering this inherent complexity of the network communication synthesis problem, we propose techniques to make a trade-off between the quality of synthesized solutions and the complexity of the synthesis process. Concretely, we propose two heuristics to address this scalability issue and to improve synthesis efficiency without major loss of quality.

*1) Route subset:* This heuristic is proposed to reduce the routing complexity. In our basic solution, we consider all possible routes for each message and calculate the optimal one. Here, we use the designers' knowledge of the network and control applications to reduce the size of the problem by eliminating the routes unlikely to be used. For example, the designer might consider only the first $K$ shortest routes for each control application. In order to apply this approach to the SMT formulation, we need to only consider the route subsets provided by the designer when formulating the *route constraint* (Equation 8).

*2) Incremental Synthesis:* For the second method we divide the hyper-period into several slices. Thus, the algorithm solves several problems of smaller size in an incremental fashion. In the first stage, the messages instantiated inside the first time slice are scheduled/routed solving the SMT formulation for the decision variables corresponding to this subset of messages. In the subsequent stages, schedules and routes are synthesized by running the SMT solver for the decision variables corresponding to the messages instantiated in the corresponding time slice and considering the decision variables determined in the previous stages as fixed. The incremental algorithm terminates after all message instances have been considered. The above incremental technique reduces the number of messages to be scheduled in each stage of the algorithm, resulting in reduced synthesis time.

Note that by applying these heuristics the solver searches within a subset of all possible solutions and, therefore, in some cases it might end up without a solution, even if such a solution exists and could be reached solving the initial, complete SMT formulation (See experimental results in Section VI.).

## VI. Experimental Setup and Results

In this section, we evaluate the efficiency of our proposed stability-aware routing and scheduling technique experimentally. We perform three experiments followed by a real-life example. Our experiments are carried out using Z3 [8], a state-of-the-art SMT solver. They were run on a regular desktop computer with 2.67 GHz Xeon CPU and 6 GB of RAM. For the three experiments, we randomly choose control applications from a database with inverted pendulums, ball

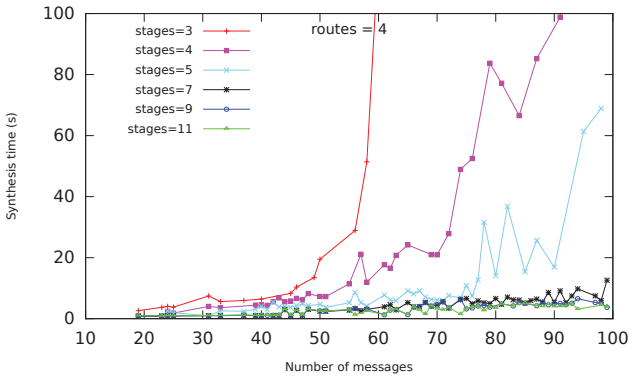


Fig. 4. Scalability of the incremental synthesis heuristic

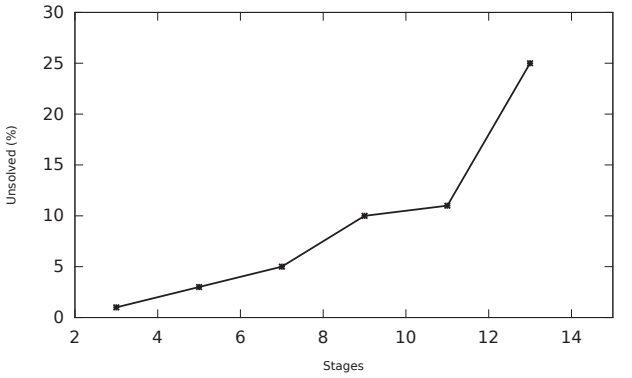


Fig. 5. Percentage of unsatisfied problems with incremental synthesis

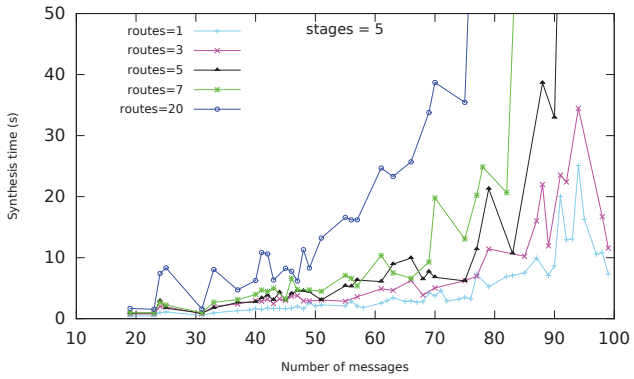


Fig. 6. Scalability of the route subset heuristic

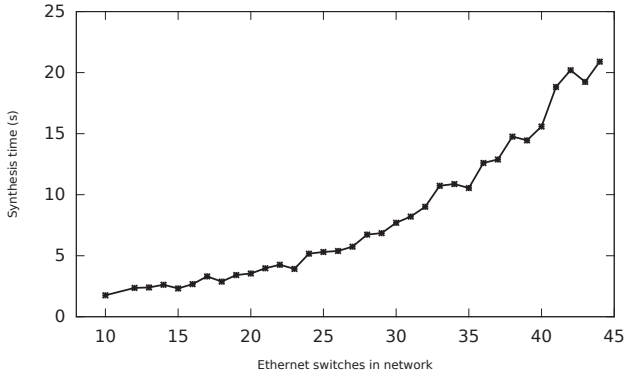


Fig. 7. Scalability of the proposed approach for big networks

and beam processes, DC servos, and harmonic oscillators. These plants are considered to be representative for realistic control applications and are extensively used for experimental evaluation in the literature [2]. The first two experiments show how our approaches scale in terms of synthesis time. In both experiments, we run 60 different synthesis problems on a network of 35 nodes (10 sensors, 10 controllers, and 15 Ethernet switches). For each synthesis problem, these 10 control applications are chosen randomly, as discussed previously. To show the impact of our incremental synthesis heuristic we run these 60 different problems 6 times changing the number of stages considered in the synthesis process. The result is shown in Figure 4. In this figure each point represents the synthesis time of a particular problem. From the graph we can see that increasing the number of stages dramatically reduces synthesis time. In our experiments, without the incremental approach (i.e., stages=1), the solver did not produce any results after a whole day for the problems with more than 80 messages, while the same problems are solved in less than a minute when we increase the number of stages to 5. Note that all problems have been run considering 4 alternative routes for each message. As discussed in Section V, with the incremental approach we pay for the speed-up with only a partial exploration of the solution space. Figure 5 shows the percentage of cases in which a solution (all controllers are guaranteed to be stable) has been found, depending on the number of time slices employed. Observe that, while execution times are already dramatically reduced for five stages compared to three (see Figure 4), the quality of exploration with five or seven stages is still very good.

In order to show the impact of providing a subset of alternative routes we run the same 60 problems 5 times changing the number of alternative routes while keeping the number of stages to 5. The results of this experiment are shown in Figure 6. The trend in the graph shows that decreasing the number of possible routes between sensors and controllers in the network reduces the synthesis time. On the other hand, our experiments show that reducing the set of alternative routes per application to 1 or 2 will result in more than 90% of unsolved problems, while for 3 and more sets of routes less than 10% remained unsolved.

The third experiment shows how synthesis time scales with the number of nodes in the network. In this experiment we choose 10 control applications that generate 45 messages in one hyper-period. Then for each experiment we change the number of Ethernet switches in the underlying network. Given the number of switches in the network, we generate randomly the topology based on the Erdős-Rényi graph model [10] and connect randomly the 10 sensors and 10 controllers to this topology. Figure 7 shows the synthesis time for different numbers of switches in the network.

Finally, we have also applied our approach to a real-life example from General Motors. The example consists of 20 sensors (camera, radar, and lidar) and electronic control units for surround view perception, object tracking, active safety functions, and autonomous vehicle control. These 20 control applications are communicating through a network of 8

TABLE I
PARAMETERS OF THE EXAMPLE CONTROL APPLICATIONS AND ROUTING AND SCHEDULING RESULTS

| | | | | Stability-Aware | | | Deadline | | |
|---|---|---|---|---|---|---|---|---|---|
| Control App. | Period (ms) | $\alpha$ | $\beta$ | $\max\{e2e\}$ (ms) | Latency (ms) | Jitter (ms) | $\max\{e2e\}$ (ms) | Latency (ms) | Jitter (ms) |
| 1 | 20 | 1.53 | 27.78 | 19.99 | 19.98 | 0.01 | 19.91 | 4.81 | 15.10 |
| 2 | 40 | 2.27 | 15.70 | 15.68 | 15.68 | 0 | 38.14 | 16.02 | 22.12 |
| 3 | 50 | 1.07 | 80.71 | 49.99 | 49.99 | 0 | 47.35 | 17.22 | 30.13 |
| 4 | 40 | 2.27 | 15.70 | 15.68 | 15.68 | 0 | 38.53 | 30.83 | 7.70 |
| 5 | 50 | 1.07 | 80.71 | 49.99 | 49.99 | 0 | 49.91 | 13.57 | 36.34 |

Ethernet switches, which are connected based on the topology depicted in Figure 1. The parameters (period, $\alpha$, and $\beta$) for five of these control applications are shown in Table I. Note that the stability curve for each control application is estimated by one line (see Equation 2), hence, one $\alpha$ and $\beta$ for each control application.

The maximum packet size for all these control applications is 1500 bytes and the Ethernet switches and links are assumed to be $10Mbit/s$. Therefore, the transmission delay for each link and each message is $ld = 1.2ms$. The forwarding delay for all switches is considered $sd = 5\mu s$.

The synthesis problem for our set of 20 applications comprises the scheduling and routing of 106 messages (generated in the hyper-period of length $200ms$). We have applied our synthesis approach providing 3 alternative routes for each message and considering 5 stages. The synthesis time was 112 seconds and all 20 applications satisfied the worst-case stability constraints. We show the resulted maximal end-to-end delay, latency, and jitter for five out of 20 control applications in Table I under *Stability-Aware* column. In order to highlight the importance of control-aware communication synthesis with explicitly considering the stability conditions, we run an additional experiment with the identical setting as before. However, instead of imposing the stability constraints (Equation 10 and 3), we imposed a constraint on the worst-case delay of each message to not exceed its period. This approach represents the state of the art (i.e., scheduling and routing with implicit hard deadlines). The results are shown in Table I under *Deadline* column. Analyzing the latencies and jitters, out of the five applications, three (highlighted in the table) turn out to be potentially unstable in the worst case. From the whole set of 20 applications, only 14 satisfied the worst-case stability conditions, further demonstrating the importance of our stability-aware routing and scheduling approach.

## VII. CONCLUSIONS

In this paper, we have addressed the joint message routing and scheduling problem for control applications communicating over time-triggered Ethernet networks. Our goal is to guarantee worst-case stability of the networked control applications considering the impact of communication delay and jitter on control quality. We have also proposed new techniques to cope with the inherent complexity of this synthesis problem. Our extensive experimental evaluation demonstrates the efficiency and applicability of the proposed technique in solving the joint routing/scheduling problem in networked cyber-physical systems to provide stability guarantees.

## REFERENCES


[1] A. Aminifar, P. Eles, Z. Peng, and A. Cervin. Control-quality driven design of cyber-physical systems with robustness guarantees. In *DATE*, 2013.

[2] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems*. Prentice Hall, 3rd edition, 1997.

[3] G. Avni. Synthesizing Time-Triggered Schedules for Switched Networks with Faulty Links. In *EMSOFT*, 2016.

[4] L. L. Bello. The case for ethernet in automotive communications. *ACM SIGBED Review*, 2011.

[5] A. Cervin. Stability and worst-case performance analysis of sampled-data control systems with input and output jitter. In *Proceedings of the 2012 American Control Conference (ACC)*, 2012.

[6] S. Chakraborty, M. A. Al Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu. Automotive cyber-physical systems: A tutorial introduction. *IEEE Design & Test*, 2016.

[7] S. S. Craciunas, R. S. Oliver, and W. Steiner. Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks. In *24th International Conference on Real-Time Networks and Systems (RTNS)*, 2016.

[8] L. De Moura and N. Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[9] P. Deng, Q. Zhu, A. Davare, A. Mourikis, X. Liu, and M. Di Natale. An efficient control-driven period optimization algorithm for distributed real-time systems. *IEEE Transactions on Computers*, 2016.

[10] P. Erdős and A. Rényi. On random graphs I. *Publ. Math. Debrecen*, 1959.

[11] V. Gavrilut and P. Pop. Traffic class assignment for mixed-criticality frames in TTEthernet. *ACM SIGBED Review*, 2016.

[12] LAN/MAN Standards Committee of the IEEE Computer Society. *IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks Amendment 25 : Enhancements for Scheduled Traffic, IEEE Std. 802.1Qbv-2015*. 2015.

[13] B. Lincoln and A. Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Proceedings of the* $41^{st}$ *IEEE Conference on Decision and Control*, 2002.

[14] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner. Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks. *IET Cyber-Physical Systems: Theory & Applications*, 2016.

[15] D. Quaglia, R. Muradore, R. Bragantini, and P. Fiorini. A SystemC/Matlab co-simulation tool for networked control systems. *Simulation Modelling Practice and Theory*, 2012.

[16] S. Samii, A. Cervin, P. Eles, and Z. Peng. Integrated scheduling and synthesis of control applications on distributed embedded systems. In *DATE*, 2009.

[17] F. Smirnov, M. Glaß, F. Reimann, and J. Teich. Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks. In *DAC*, 2017.

[18] W. Steiner. An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks. In *Real-Time Systems Symposium*, 2010.

[19] W. Steiner. Synthesis of Static Communication Schedules for TTEthernet-Based Mixed-Criticality Systems. In *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, 2011.

[20] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch. Time-Triggered Ethernet: TTEthernet. *Time-Triggered Communication*, 2011.

[21] D. Tamas-Selicean, P. Pop, and W. Steiner. Design optimization of TTEthernet-based distributed real-time systems. *Real-Time Systems*, 2014.

[22] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty. Task- and Network-level Schedule Co-Synthesis of Ethernet-based Time-triggered Systems. In *ASP-DAC*, 2014.