Pattern Guided Integrated Scheduling and Routing in Multi-Hop Control Networks

SUMANA GHOSH, SOUMYAJIT DEY, and PALLAB DASGUPTA, Indian Institute of Technology Kharagpur

Executing a set of control loops over a shared multi-hop (wireless) control network (MCN) requires careful co-scheduling of the control tasks and the routing of sensory/actuation messages over the MCN. In this work, we establish pattern guided aperiodic execution of control loops as a resource-aware alternative to traditional fully periodic executions of a set of embedded control loops sharing a computation and the communication infrastructure. We provide a satisfiability modulo theory–based co-design framework that synthesizes loop execution patterns having optimized control cost as the underlying scheduling scheme together with the associated routing solution over the MCN. The routing solution implements the timed movement of the sensory/actuation messages of the control loops, generated according to those loop execution patterns. From the given settling time requirement of the control loops, we compute a control theoretically sound model using matrix inequalities, which gives an upper bound to the number of loop drops within the finite length loop execution pattern. Next, we show how the proposed framework can be useful for evaluating the fault tolerance of a resource-constrained shared MCN subject to communication link failure.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; **Embedded software**; • **Networks → Cyber-physical networks**; • **Hardware →** *Functional verification*;

Additional Key Words and Phrases: Multi-hop control networks, schedulability, routing, control performance

ACM Reference format:
Sumana Ghosh, Soumyajit Dey, and Pallab Dasgupta. 2020. Pattern Guided Integrated Scheduling and Routing in Multi-Hop Control Networks. *ACM Trans. Embed. Comput. Syst.* 19, 2, Article 9 (February 2020), 28 pages.
https://doi.org/10.1145/3372134

1 INTRODUCTION

Multi-hop (wireless) control networks (MCNs) have emerged as a promising technology in the rapidly expanding family of wireless-based cyber-physical systems due to their increased flexibility, lower maintenance and installation cost, compositionality, and adaptability [1, 2]. As shown in Figure 1, such systems consist of a centralized control node C, responsible for controlling a set of plants $\{P_1, \ldots, P_n\}$, a set of sensor nodes V_S, a set of actuator nodes V_A, and several intermediate wireless nodes given by the set V_I. Each control loop must execute periodically, and for each execution of a control loop, the wireless network must carry the sensory message from sensor nodes

This work was partially supported by the IMPRINT grant FMSAFE.
Authors' address: S. Ghosh (corresponding author), S. Dey, and P. Dasgupta, Computer Science and Engineering, Indian Institute of Technology Kharagpur, West Bengal, 721302, India; emails: sumanaghosh@cse.iitkgp.ernet.in, {soumya, pallab}@cse.iitkgp.ac.in.

1539-9087/2020/02-ART9 $15.00
https://doi.org/10.1145/3372134

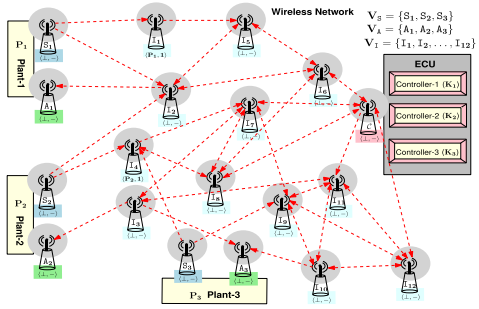


Fig. 1. A typical MCN.

to the control node, and actuation messages from the control node to actuator nodes inside a time budget bounded by the sampling period of the control loop.

Because of the limited bandwidth of the wireless networks, the execution of control loops has to be appropriately married with the scheduling of the sensory/actuation messages through the network. It is possible that the sampling rates of the control loops chosen by a designer based on control-theoretic considerations generate infeasible traffic on the network, which means that all sensory/actuation messages cannot be delivered on time by the network. This essentially mandates a proper balancing between the sampling rates of the control loops and real-time communication constraints to achieve the desired performance of the overall system.

In general, control designers assume fully periodic execution, and the standard practice for handling network bandwidth constraints is twofold. One possibility is to perform suitable network engineering and increase the network bandwidth.[1] The other possibility is choosing lower sampling rates and trading off control performance, thereby generating a control schedule and routing solution that works inside the available bandwidth. In this work, one of our key contributions is to propose a novel alternative to this common practice. Instead of using the inferior controller having a lower sampling rate, we propose to use the desirable sampling rates with some intentional drops of control loop executions in the form of well-defined pattern of loop executions. This helps in co-scheduling the control loops over the limited network bandwidth, which is often not possible otherwise (using standard periodic loop execution). We also provide a sound control-theoretic model for giving the upper bound to the number of drops in control loop executions, accounting for the given performance guarantees such as settling time. Although the concept of using pattern-based aperiodic scheduling to handle the load of a shared computation/communication platform is already well studied [15, 17, 27, 34, 42], exploring the same in the context of shared MCNs thereby considering network-induced constraints is still unexplored. We employ a novel satisfiability modulo theory (SMT)-based methodology to co-synthesize the patterns of loop execution together with the routing solution for the traffic generated by such patterns to be carried in real time by the wireless network. A technique similar to BMC is used as the underlying symbolic algorithm.

In the next part of this work, we present a prognostic approach that monitors the schedulability of the current MCN configuration considering faults (before they actually occur) and pre-computes schedulable solutions intelligently leveraging the SMT-based co-synthesis framework developed in the first part. Based on this, in an actual link failure scenario, the MCN can be reconfigured with alternate scheduling and routing. In addition, given the pre-computed schedulable solutions, the MCN user can be alerted at runtime about the vulnerability of the current MCN configuration

[1]This may be a difficult option since mostly the networking infrastructure already exists as a pre-installed physical asset.

based on availability of the schedulable solutions and current number of failed links so that necessary preventive measures can be taken. This helps in scheduling predictive maintenance for such systems. We summarize the novel contributions of this work as follows:

(1) We present a control-theoretic model based on matrix inequalities for calculating the lower bound on the number of loop executions over a finite length execution pattern, which guarantees the desired control performance (settling time). With such a derived constraint of provable performance, SMT-based MCN scheduling is a salient feature of this work.
(2) We include a quadratic cost metric within our SMT formulation exploiting the relationship of quadratic cost with loop drops and use an SMT optimizer to choose the loop execution pattern that optimizes the overall control cost. This significantly reduces the effort of finding the optimal solution as compared to the existing approach [16] of a brute-force method.
(3) At the heart of our co-synthesis, we propose new algorithms that judiciously allow drops in loop executions until the sensory and actuation messages are schedulable over the network.
(4) Finally, we show how our approach can be used to evaluate the fault tolerance of the MCN under link failures. We propose a heuristic approach that works with our SMT-based framework and an incremental SMT procedure for evaluating fault tolerance in a look-ahead fashion.

We believe this framework is the first attempt toward wireless control and scheduling co-design where complex interactions between control performance and real-time communication are formally modeled using suitable SMT clauses, and the optimal solution is obtained using an SMT optimizer.

The article is organized as follows. Section 2 discusses relevant related work in this direction, whereas Section 3 presents the system definitions formally. The detailed derivation of the performance metrics that we consider in this work is given in Section 4. Section 5 develops the SMT-based formalism to the co-synthesis problem in a given MCN, whereas the algorithmic framework for evaluating fault tolerance under communication link failure is described in Section 6. Section 7 presents the illustration of the key concepts with suitable case studies, followed by the experimental evidence to show superior quality of control (QoC) that is achieved using the proposed approach as compared to the traditional design techniques. Finally, Section 8 presents concluding remarks.

2 RELATED WORK

The problems of channel scheduling [14, 44], routing [28], and end-to-end delay analysis [18, 35, 37] for MCNs are well studied. However, other works [14, 35, 37, 44] do not consider the co-design problem of scheduling and routing as we do in this work. The problem of selecting optimal sampling rates for multiple feedback control loops sharing a wireless network has been addressed elsewhere [3, 4, 19, 36, 38]. Among them, the work in Saifullah et al. [36] is closer to our work. The problem of Bai et al. [4] and Saifullah et al. [36] is that they assume pre-defined routing paths and fully periodic sampling rates for all of the control loops, which therefore suffers from the sub-optimality problem that we address in this work. The limitation of Bai et al. [3], Li et al. [19], and Shu et al. [38] is that their analysis is not applicable for any multi-hop wireless network. A nice survey covering this line of research targeting wireless industrial cyber-physical systems can be found in Lu et al. [20]. An earlier effort similar to this work appears in Ghosh et al. [16]. But the major limitation of Ghosh et al. [16] is that it does not scale for large network configurations since

it computes all possible satisfiable solutions and evaluates their quadratic costs independently to get the optimal solution. Moreover, Ghosh et al. [16] focuses on stability only without considering any control performance requirement. In this work, we handle both of these issues in an efficient way.

Another line of work proposed in Mangharam and Pajic [25] and Pajic et al. [29] treats the wireless network itself as a controller, which is completely different from our assumption of an MCN. These works present an extension of their earlier work reported in Pajic et al. [30]. In Mangharam and Pajic [25] and Pajic et al. [29], the authors show that instead of assigning the task of control law computation to a particular node in the network, the entire network can be used as the controller. In their setup of MCN, which they referred to as *Wireless Control Network*, each node executes a procedure that updates its state as the linear combination of the states of its neighbors and thus stabilize the closed loop. Research developed in other works [1, 2, 40] provides a compositional method that derives a global switched system for an MCN and checks its stability under the assumption of a given routing path for each control loop. Based on this MCN model, the authors in D'Innocenzo et al. [8] synthesize the suitable routing paths for the control loops. However, as admitted by the authors in Fiore et al. [14], the construction of an automaton in Alur et al. [1, 2] for capturing the admissible patterns suffers from scalability issues. In contrast, inside our framework, we use the theory presented in Section 4 that provides a computable lower bound on loop execution count while guaranteeing the required performance. This enables us to choose the execution patterns without constructing any automaton. An SMT-based schedule synthesis methodology for multi-hop networks was developed in other works [7, 24, 32, 43]. All of the formal modeling and analysis of the work in Craciunas et al. [7] and Steiner [43] are mainly for multi-hop time-triggered Ethernet networks, whereas Ro et al. [32] presents the formal specification of the scheduling constraints for time-triggered wireless sensor networks with re-transmissions. However, all of these works focus on the real-time scheduling aspect only, ignoring the effect of scheduling on control metrics like stability and performance. However, the work in Mahfouzi et al. [24] addresses a joint routing and scheduling problem for periodic control loops communicating over time-triggered Ethernet networks, guaranteeing worst-case stability of the systems under the effect of delay and jitter. In contrast, our method synthesizes non-uniform patterns of loop execution guaranteeing not only the stability but also a given control performance. The constraints for Ethernet networks are different from our constraints of MCN for WirelessHART standard. In Mahfouzi et al. [24], an appropriate route is chosen by the SMT solver from a set of pre-defined routes for each loop, whereas our SMT solver synthesizes the route for each control loop according to the chosen execution pattern for that loop.

The works in Ning et al. [27] and Saha et al. [34] respectively find the optimal packet delivery sequence and dynamic scheduling for achieving optimal disturbance rejection in the presence of network packet drops. However, those authors [27, 34] do not consider wireless networks and the constraints that such networks induce. Control-scheduling co-design problems for low-power multi-hop wireless networks have been addressed in Mager et al. [23] and Zimmerling et al. [46]. The co-design problems addressing the joint optimization of control performance and energy consumption of the wireless network are reported in elsewhere [22, 31, 33]. However, none of the works reported in Ma et al. [22] and Park et al. [31] consider hard real-time requirements of control applications, whereas Sadi and Ergen [33] does not consider the challenges of a multi-hop network. Authors in D'Innocenzo et al. [9, 10] provide a fault-tolerant stabilization methodology where necessary and sufficient conditions are proposed on the plant dynamics and the communication protocol subject to permanent link failures and malicious intrusions. In contrast, our work focuses on the offline synthesis of feasible scheduling and/or routing solutions under different link

Table 1. Comparative Study of Related Work

Method	Scheduling Aspect?	Control Aspect?	Wireless Network?	Multi-Hop Network?	Aperiodic Sampling	Synthesize Routing?
[7, 14, 32, 35, 37, 43, 44]	Yes	No	Yes	Yes	No	No
[18, 28]	Yes	No	Yes	Yes	No	Yes
[4, 23, 24, 36, 40, 46]	Yes	Yes	Yes	Yes	No	No
[3, 19, 38]	Yes	Yes	Yes	No	No	No
[25, 29]	No	Yes	Yes	Yes	No	No
[1, 2]	Yes	Yes	Yes	Yes	Yes	No
[27, 34]	Yes	Yes	No	No	Yes	No
[22, 31, 33]	No	Yes	Yes	No	No	No
Proposed	Yes	Yes	Yes	Yes	Yes	Yes

failure configurations to make the system robust from the early design phase. Table 1 presents a comparative study of the existing methods in this context.

3 MODEL DEFINITION

An MCN is a tuple $\mathcal{N} = \langle \mathcal{P}, \mathcal{K}, \mathcal{G} \rangle$ consisting of a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n plants, a set $\mathcal{K} = \{K_1, \ldots, K_n\}$ of n controllers, and a network model $\mathcal{G}$. The plant-controller pairs are defined in standard terms using dynamic matrices and together constitute the control design of the MCN. The following sections formalize the plant-control model, aperiodic patterns of loop executions, and the network model.

3.1 Plant-Controller Model

We use the vector $x = [x_p^\mathsf{T}, x_c^\mathsf{T}]^\mathsf{T}$ to represent the state vector of the closed-loop control system, where the vector x_p represents the state of the plant variables and the vector x_c represents the state of the control variables. The dynamics of the physical plant, when sampled with some fixed interval h, can be expressed as a discrete-time linear time-invariant (LTI) system with equations:

$$x_p[k+1] = A_p x_p[k] + B_p u[k], \quad y[k] = C_p x_p[k],$$

where the vectors $x_p[k]$, $y[k]$, and $u[k]$ define the plant state, the output, and the control input, respectively, at time $t = kh$, for some $k \in \mathbb{N}$ (that means, t is the real time at the k-th sample). The matrices A_p, B_p, and C_p describe the transition matrix, the input map, and the output map for the plant model, respectively. The feedback control software senses the plant output y and decides the control action by adjusting the control variables in u. The feedback control law is usually represented as an LTI system given as

$$x_c[k+1] = A_c x_c[k] + B_c y[k], \quad u[k] = C_c x_c[k],$$

where $x_c[k]$ represents the state of the controller at the time $t = kh$, and A_c, B_c, and C_c are the state transition matrix, the input map, and the output map for the controller, respectively. Thus, for an LTI plant $P = (A_p, B_p, C_p)$ and its stabilizing controller $K = (A_c, B_c, C_c)$, the dynamics of the resulting closed loop, $\Sigma = \langle P, K \rangle$, is given as

$$x[k+1] = \begin{bmatrix} A_p & B_p C_c \\ B_c C_p & A_c \end{bmatrix} x[k] = A_1\, x[k]. \tag{1}$$

The notion of *loop drop* is formally incorporated as follows. If the controller does not execute in a sampling interval $[k, k+1)$, then $x_c[k+1] = x_c[k]$. In this interval, the closed-loop dynamics is

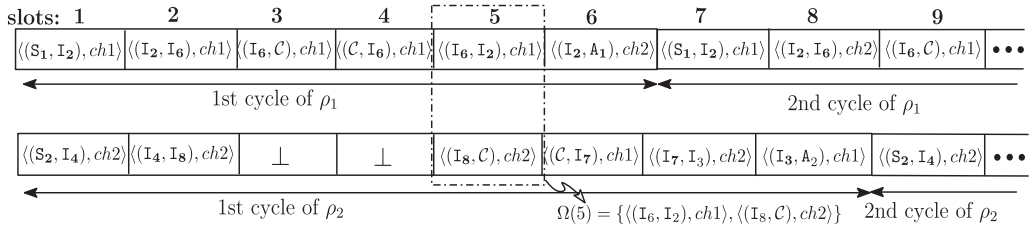


Fig. 2. Portion of a routing solution.

therefore $x[k+1] = A_0\, x[k]$, where A_0 is same as A_1, except A_c is replaced by the identity matrix, I_c, and B_c is replaced by the null matrix, O—that is, $A_0 = \begin{bmatrix} A_p & B_p C_c \\ O & I_c \end{bmatrix}$.

For a given control loop $\Sigma = \langle P, K \rangle$ together with its dynamic matrices $\{A_0, A_1\}$, a *loop execution pattern* is defined as an infinite sequence, s^ω, which is an infinitely repeating finite length string $s \in \{A_0, A_1\}^*$. Therefore, $\forall k \in \mathbb{N}$, the dynamics of the closed-loop system, $\langle P, K \rangle$, is defined as $x[k+1] = s[k\%l]x[k]$, where l is the length of s. Starting from an initial state $x[0]$, according to $s = s[0]s[1]\cdots s[l-2]s[l-1]$, the system evolves as $x[l] = s[l-1]s[l-2]\cdots s[1]s[0]x[0]$, where $x[l]$ is the state valuation after l-th iteration. For example, according to the loop execution pattern $s = A_1A_1A_1A_0A_1A_0$, we have the system evolving as $x[6] = A_0x[5] = \cdots = A_0A_1A_0A_1A_1A_1x[0]$. Note that in the sampling intervals corresponding to positions where we have A_0, the network does not have to carry the sensory and actuation messages for that control loop. For the sake of simplicity, a loop execution pattern may be expressed as a binary string. As an example, for $A_1A_1A_1A_0A_1A_0A_1$ we denote this loop execution pattern by $s = 1110101$.

3.2 Network Model

The network model $\mathcal{G}$ consists of a directed connectivity graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the number of *channels* (frequency bands) M. $\mathcal{V}$ represents the set of vertices defined as $\mathcal{V} = V_\mathtt{S} \cup V_\mathtt{A} \cup V_\mathtt{I} \cup \mathcal{C}$, where $V_\mathtt{S}$ denotes the set of sensor nodes, $V_\mathtt{A}$ the set of actuator nodes, $V_\mathtt{I}$ the set of intermediate nodes that follow store and forward policy to route messages over the network, and $\mathcal{C}$ the control node that is responsible for executing control laws for all of the plants. The set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ models the radio connectivity. An edge $(v_i, v_j) \in \mathcal{E}$, from node v_i to node v_j exists if v_j lies within the transmission range of v_i, and v_i transmits messages to v_j. The sensor nodes can only transmit, and actuator nodes can only receive. Our modeling of MCN is well equipped for capturing standard industrial wireless control protocols such as WirelessHART [41].

Each channel is divided into time slots of length δ (in WirelessHART, $\delta = 10$ ms). Each time slot on each channel allows exactly one transmission and its acknowledgment between a pair of nodes. The sensory and actuation messages generated following the loop execution patterns have to be routed across the network by using these time slots. A *routing solution* defines the use of time slots on each channel by pairs of communicating nodes. In our case, the routing solution is recurrent because the loop execution patterns generate recurrent traffic on the network. Thus, a routing solution can be visualized as a collection of *recurrent transmission schedules*, one for each plant.

Figure 2 shows a part of the routing solution of two plants corresponding to the MCN given in Figure 1. Conservatively assuming interference between any two transmissions using the same frequency, two channels ($ch1$ and $ch2$) are used for simultaneous communications between pairs of nodes. The recurrent transmission schedules, ρ_1 and ρ_2 (for plant-1 and plant-2, respectively), are shown in Figure 2. Note that in Figure 2, $ch1$ is used by $\mathtt{S}_1$ in time slot 1 to transmit to node

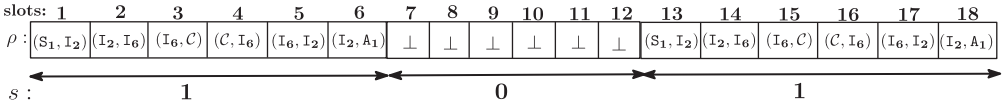


Fig. 3. Routing following a loop execution pattern.

I_2 (denoted by $\langle(S_1, I_2), ch1\rangle$ in slot 1 of ρ_1), whereas $ch2$ is used by S_2 to transmit to node I_4 (denoted by $\langle(S_2, I_4), ch2\rangle$ in slot 1 of ρ_2). Empty time slots are shown as $\perp$—for example, the sensory message from S_2 waits at I_8 as the node C is busy in communicating with I_6 in slot 3. Note that ρ_1 captures one complete end-to-end communication—for instance, transmission of the sensory message from plant-1 to controller and respective actuation message back from the controller to plant-1 within 6 time slots. Similarly, ρ_2 represents end-to-end communication for plant-2 within 8 time slots. Since ρ_1 and ρ_2 have lengths 6 and 8, respectively, the global state of the network will recur after $lcm(6, 8) = 24$ slots. Formally, we define the terms, recurrent transmission schedules and routing solution, as follows.

Definition 1 (Recurrent Transmission Schedule). Given an MCN $\mathcal{N} = \langle \mathcal{P}, \mathcal{K}, \mathcal{G} \rangle$, where the plant $P_i \in \mathcal{P} = \{P_1, \ldots, P_n\}$ has the sampling period h_i, and for $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of M available channels, $CH = \{ch1, \ldots, chM\}$, each channel has time slot of length δ. In case of plant P_i, the recurrent transmission schedule, ρ_i, of length π_i, is a function defined as $\rho_i : [1, 2, \ldots, \pi_i] \rightarrow (\mathcal{E} \cup \{\perp\}) \times CH$, where $\pi_i \times \delta \leq h_i$.

The condition, $\pi_i \times \delta \leq h_i$, in Definition 1 ensures that end-to-end communication (sense-actuation cycle) must be within the sampling period h_i of that plant P_i, enabling its stability guarantee.

Definition 2 (Routing Solution). Given n recurrent transmission schedules, $\rho_1, \ldots, \rho_n$, of lengths $\pi_1, \ldots, \pi_n$, for n plants of an MCN $\mathcal{N} = \langle \mathcal{P}, \mathcal{K}, \mathcal{G} \rangle$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with CH as the set of M channels, we define the routing solution as a synchronous parallel composition of those recurrent transmission schedules, denoted as $\Omega = \rho_1 || \ldots || \rho_n$ having period $\Pi = lcm(\pi_1, \ldots, \pi_n)$, where $\Omega : [1, \ldots, \Pi] \rightarrow 2^{(\mathcal{E} \cup \{\perp\}) \times CH}$ with $\Omega(i)$ defined as $\Omega(i) = \{\rho_1(i), \ldots, \rho_n(i)\}$.

For example, according to Figure 2, we get $\rho_1 = \{\langle(S_1, I_2), ch1\rangle, \langle(I_2, I_6), ch1\rangle, \langle(I_6, C), ch1\rangle, \langle(C, I_6), ch1\rangle, \langle(I_6, I_2), ch1\rangle, \langle(I_2, A_1), ch1\rangle\}$ and $\rho_2 = \{\langle(S_2, I_4), ch2\rangle, \langle(I_4, I_8), ch2\rangle, \langle\perp, ch2\rangle, \langle\perp, ch2\rangle, \langle(I_8, C), ch2\rangle, \langle(C, I_7), ch2\rangle, \langle(I_7, I_3), ch2\rangle, \langle(I_3, A_2), ch2\rangle\}$, thereby $\Omega(1) = \{\langle(S_1, I_2), ch1\rangle, \langle(S_2, I_4), ch2\rangle\}$, $\Omega(3) = \{\langle(I_6, C), ch1\rangle\}$, $\Omega(5) = \{\langle(I_6, I_2), ch1\rangle, \langle(I_8, C), ch2\rangle\}$ (highlighted in Figure 2), and so on.

Routing following a loop execution pattern. As mentioned in Section 3.1, the network does not carry the sensory and actuation messages in a sampling interval of a control loop, if there is a loop drop at that interval. We now illustrate this fact in terms of recurrent transmission schedule of that loop in a continuation with the previous example. Let the loop execution pattern be $s = 101$ and the entire routing for that loop be done using one channel. Then the corresponding recurrent transmission schedule, ρ, is shown in Figure 3 (for the sake of clarity of the figure and the assumption of using only one channel, the channel id is removed from all tuples). Therefore, for a set of loop execution patterns, $\{s_1, \ldots, s_n\}$, for n control loops, we find the period (Π) of the routing solution, Ω, satisfying the condition $\Pi\delta \leq lcm(|s_1| \times h_1, \ldots, |s_n| \times h_n)$. Note that within the bound, $lcm(|s_1| \times h_1, \ldots, |s_n| \times h_n)$, all of the loop execution patterns repeat an integer number of times.

4 DERIVING MINIMUM LOOP EXECUTIONS FROM PERFORMANCE METRIC

Consider that starting from k-th sample, the loop is dropped in next consecutive β_1 sampling intervals, then in consecutive α_1 sampling intervals it is executed, and so on, for $j = 1, 2, \ldots, a$. In this way, we have the system evolution over the sampling window, $[k, k + l)$ as

$$x[k + l] = A_1^{\alpha_a} A_0^{\beta_a} \ldots A_1^{\alpha_2} A_0^{\beta_2} A_1^{\alpha_1} A_0^{\beta_1} x[k], \tag{2}$$

where $\sum_{j=1}^{a} \alpha_j = \kappa$, $\sum_{j=1}^{a} \beta_j = \theta$, and $l = \kappa + \theta$. Note that for any l-length sampling window $[k, k + l)$, $k \geq 0$, the corresponding loop execution pattern, $1^{\alpha_1} 0^{\beta_1} 1^{\alpha_2} 0^{\beta_2} \ldots 1^{\alpha_a} 0^{\beta_a}$, in Equation (2) varies with the variation of α_j, β_j, for $j = 1, 2, \ldots, a$ and for any $a \in \mathbb{N}$. However, for any such loop execution pattern, the closed-loop stability of this system may not be guaranteed. Following Soudbakhsh et al. [42], we introduce the control-theoretic analysis of stability and performance for the system in Equation (2) exploiting the following theorem of matrix inequalities.

Theorem 1. *Dowler [12]. Given a matrix $A \in \mathbb{C}^{n \times n}$ with spectral radius $\mathscr{R}(A) = sup\{|\lambda| : \lambda \in \Lambda(A)\}$, where $\Lambda(A)$ is the set of eigenvalues of A, for any $\gamma \geq \mathscr{R}(A)$, there exists a $c \geq 1$ such that for each non-negative integer j, $||A^j|| \leq c\gamma^j$.*

Note that since A_0 is a block upper triangular matrix (see Section 3.1), $|A_0| = |I_c|.|A_p|$ (using *Schur complement* of matrix [39]), which means that eigenvalues of A_0 are the combined eigenvalues of A_p and I_c. Therefore, A_0 always has eigenvalue 1 with some multiplicity m since all of the eigenvalues of I_c are equal to 1. Other eigenvalues of A_0 are the eigenvalues of A_p. If the eigenvalues of A_p are less or equal to 1, then $\mathscr{R}(A_0) = 1$, else $\mathscr{R}(A_0) > 1$. Hence, $\mathscr{R}(A_0) \geq 1$.

Therefore, in our setting, for the stable closed-loop dynamic matrix A_1, we can get Equation (3), and similarly, for the open-loop dynamic matrix A_0, since $\mathscr{R}(A_0) \geq 1$, we can have Equation (4).

$$||A_1^j|| \leq c_1 \gamma_1^j, \quad c_1 \geq 1, \quad 0 \leq \gamma_1 < 1 \tag{3}$$

$$||A_0^j|| \leq c_0 \gamma_0^j, \quad c_0 \geq 1, \quad \gamma_0 \geq 1 \tag{4}$$

In this work, the control performance requirement is captured in terms of (l, ϵ)-exponential stability criterion [45] of the system, which is defined as follows.

Definition 3 ((l, ϵ)-Exponential Stability). Given the dynamical system of Equation (1), this criterion requires the system to reduce the error by at least a factor of ϵ in every l-length sampling window (i.e., to meet $\frac{||x[k+l]||}{||x[k]||} \leq \epsilon$ for every $k \in \mathbb{N}$ and $x[k] \in \mathbb{R}^n$), with $l \in \mathbb{N}$ and $\epsilon \in [0, 1)$.

This (l, ϵ)-exponential stability criterion can be easily gleaned from the standard performance index, *settling time*, and the desired system norm [15]. The settling time, τ_s, requires the controller to move the system back to its reference value of χ from the perturbed value of $\chi + \sigma$ within τ_s. To meet the settling time criterion, the number of sampling intervals needed is $L = \lceil \frac{\tau_s}{h} \rceil$, where h is the sampling period. The respective damping factor becomes $\xi = \frac{\chi}{\chi + \sigma}$. We can always use a stricter criterion $(\frac{L}{q}, \xi^{\frac{1}{q}})$, for some $q > 1$, with a lower number of sampling intervals, such that satisfaction of $(\frac{L}{q}, \xi^{\frac{1}{q}})$ always implies satisfaction of (L, ξ). We put $l = \frac{L}{q}$ and $\epsilon = \xi^{\frac{1}{q}}$, and consider this stricter (l, ϵ)-exponential stability criterion as the performance requirement.

Accordingly, the stability of the system in Equation (2) can be addressed by the following theorem.

Theorem 2. *Given the system in Equation (2) with the associated closed-loop matrix A_1 being Schur stable and $\theta \leq \theta_{max}$ being the number of loop drops over any sampling window of length l, the system remains (l, ϵ)-exponentially stable if there are $\kappa \geq \kappa_{min}$ successful executions of the*

loop over that l-length sampling window with $\kappa_{min} = \lceil \frac{(\theta_{max}+1)log_e(c_0c_1)+\theta_{max}log_e(\gamma_0)+|log_e(\epsilon)|}{|log_e(\gamma_1)|} \rceil$, *where* $c_0 \geq 1, c_1 \geq 1, \gamma_0 > 1$, *and* $0 \leq \gamma_1 < 1$.

Proof. Using the *sub-multiplicative* property of the matrix norm (i.e., $||AB|| \leq \|A\|.\|B\|$) from Equation (2), we get the following:

$$\|x[k+l]\| \leq \left\|A_1^{\alpha_a}\right\|.\left\|A_0^{\beta_a}\right\| \ldots \left\|A_1^{\alpha_2}\right\|.\left\|A_0^{\beta_2}\right\|.\left\|A_1^{\alpha_1}\right\|.\left\|A_0^{\beta_1}\right\|.\|x[k]\|.$$

For being (l,ϵ)-exponential stable, $\frac{||x[k+l]||}{||x[k]||} < \epsilon$ should hold. This condition can be guaranteed if $||A_1^{\alpha_a}||.||A_0^{\beta_a}|| \ldots ||A_1^{\alpha_2}||.||A_0^{\beta_2}||.||A_1^{\alpha_1}||.||A_0^{\beta_1}|| < \epsilon$ becomes true. Now,

$$\left\|A_1^{\alpha_a}\right\|.\left\|A_0^{\beta_a}\right\|..\left\|A_1^{\alpha_2}\right\|.\left\|A_0^{\beta_2}\right\|.\left\|A_1^{\alpha_1}\right\|.\left\|A_0^{\beta_1}\right\| < \epsilon \Rightarrow \sum_{j=1}^{a} log_e\left(\left\|A_1^{\alpha_j}\right\|\right) + \sum_{j=1}^{a} log_e\left(\left\|A_0^{\beta_j}\right\|\right) < log_e(\epsilon). \quad (5)$$

From the inequalities given in (3) and (4), we respectively get $\sum_{j=1}^{a} log_e(||A_1^{\alpha_j}||) \leq \sum_{j=1}^{a} log_e(c_1\gamma_1^{\alpha_j})$ and $\sum_{j=1}^{a} log_e(||A_0^{\beta_j}||) \leq \sum_{j=1}^{a} log_e(c_0\gamma_0^{\beta_j})$. Since $\sum_{j=1}^{a} \alpha_j = \kappa$ and $\sum_{j=1}^{a} \beta_j = \theta$, the inequality in (5) is satisfied if the following holds:

$$alog_e(c_1) + \kappa log_e(\gamma_1) + alog_e(c_0) + \theta log_e(\gamma_0) < log_e(\epsilon) < 0 \quad [\because \epsilon < 1]. \quad (6)$$

With $a \geq 1$ and $log_e(c_1), log_e(c_0) \geq 0$ as $c_1, c_0 \geq 1$, the preceding inequality can be written as

$$alog_e(c_1c_0) + \kappa log_e(\gamma_1) + \theta log_e(\gamma_0) < log_e(\epsilon) \Leftarrow alog_e(c_0c_1) + \theta log_e(\gamma_0) < log_e(\epsilon) - \kappa log_e(\gamma_1)$$
$$\Leftarrow \kappa|log_e(\gamma_1)| - |log_e(\epsilon)| > log_e(c_0c_1) + \theta log_e(\gamma_0) \quad [\because log_e(\epsilon) < 0, log_e(\gamma_1) < 0 \; as \; (\gamma_1 < 1)]$$
$$\Leftarrow \kappa > \frac{alog_e(c_0c_1) + \theta log_e(\gamma_0) + |log_e(\epsilon)|}{|log_e(\gamma_1)|} \Leftarrow \kappa > \frac{(\theta+1)log_e(c_0c_1) + \theta log_e(\gamma_0) + |log_e(\epsilon)|}{|log_e(\gamma_1)|}$$

since the number of switching $a \leq \theta + 1$. Thus, for an l-length sampling window (or l-length loop execution pattern) without compromising the given settling time requirement, if we allow up to θ_{max} number of loop drops, then the minimum number of loop executions κ_{min} within that l-length window can be lower bounded as

$$\kappa_{min} = \left\lceil \frac{(\theta_{max}+1)log_e(c_0c_1) + \theta_{max}log_e(\gamma_0) + |log_e(\epsilon)|}{|log_e(\gamma_1)|} \right\rceil. \quad (7)$$

□

4.1 Cost Functions for Loop Execution Patterns

In general, the QoC obtained following the loop execution patterns vary with the pattern definition due to the presence of loop drops. Therefore, in case of two loop execution patterns having the same number of loop drops, although both of them satisfy the performance requirement due to the drop bound provided by Theorem 2, their QoC may vary with their distribution of drops. We use the linear quadratic cost function of Equation (8) as the measure of QoC.

$$J = \int_0^\infty \left(x^T[t]Qx[t] + u^T[t]Ru[t]\right) dt \quad (8)$$

$$\left\lceil (j+1).\frac{\kappa}{l} \right\rceil - \left\lceil j.\frac{\kappa}{l} \right\rceil, \quad 0 \leq j \leq l \quad (9)$$

The quadratic weight matrices Q and R in J represent the relative importance of the deviation of state valuation $x[t]$ and control effort $u[t]$, respectively. Standard linear quadratic regulator (LQR)-based control design minimizes J over infinite horizon and provides a least cost optimal controller subject to perfectly periodic execution. Since in case of loop execution patterns we consider recurrent control execution with drops instead of perfectly periodic execution, the QoC becomes

dependent on the relative positions of loop drops [27]. Among such patterns having the same loop drops, it can be shown following Ning et al. [27] that the pattern with the most uniform drop exhibits the best QoC in terms of LQR cost. The definition of uniformity follows uniform distribution of 0s in binary words known as upper mechanical word [6]. Formally, a binary word is upper mechanical if and only if there exists $0 \leq \mu \leq 1$ such that the j-th letter of the word is given by $\lceil (j+1).\mu \rceil - \lceil j.\mu \rceil, \forall j \geq 0$, where μ denotes the fraction of 1 second in the word. Hence, among l-length loop execution patterns with number of loop executions as κ, the best choice of pattern with respect to LQR cost can be obtained following the expression given in D'Innocenzo et al. (9). Note that the best choice of the pattern is only with respect to the given controller in use and the cost function.

5 PATTERN GUIDED SCHEDULING AND ROUTING IN AN MCN

This section describes the pattern guided co-synthesis problem of scheduling-routing for an MCN and its SMT-based solution mechanism. Given the plant-controller models $\langle \mathcal{P}, \mathcal{K} \rangle$ with h_i as the sampling period of the plant $P_i \in \mathcal{P}$, the network model, $\mathcal{G}$, and the performance metric (settling time) of all of the control loops as inputs to the co-synthesis problem, our goal is to find the following:

(1) A recurrent loop execution pattern for each control loop (i.e., the set $\mathscr{S} = \langle s_1, \ldots, s_n \rangle$).
(2) A routing solution, Ω, that realizes the timed movement of the sensory and actuation messages on the network as generated by the synthesized loop execution patterns, $\mathscr{S} = \langle s_1, \ldots, s_n \rangle$. The existence of a routing solution ensures schedulability of control loops over the network.

The solution to this co-synthesis problem needs to guarantee that all of the control loops are stable and meet the given settling time constraints on the MCN.

5.1 Formal Modeling of the Network

This section starts by presenting the formal modeling of the network that is essential for building the SMT-based formalism. The *state* of a node v_j in the network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as follows.

Definition 4 (State of a Node). Given an MCN $\mathcal{N} = \langle \mathcal{P}, \mathcal{K}, \mathcal{G} \rangle$, where $\mathcal{P} = \{P_1, \ldots, P_n\}$, $\mathcal{K} = \{K_1, \ldots, K_n\}$, and $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a node $v_j \in \mathcal{V}$ is in state $v_j(T)$ at time T, with $v_j(T) = \langle P_i, \tau \rangle$, if it contains a sensory message generated by the plant P_i at some time $\tau \leq T$. Likewise, the state of v_j is defined as $v_j(T) = \langle K_i, \tau \rangle$ if it contains an actuation message of the controller K_i, computed using the sensory message $\langle P_i, \tau \rangle$. The state of v_j may also be defined as $v_j(T) = \langle \perp, - \rangle$ if the node has no messages at time T.

Following Definition 4, for any intermediate node $v_j \in V_{\mathrm{I}} \subset \mathcal{V}$, $v_j(T) = \langle P_i, \tau \rangle / \langle K_i, \tau \rangle / \langle \perp, - \rangle$. Since sensor nodes can only transmit and actuator nodes can only receive, a sensor node $v_j \in V_{\mathrm{S}} \subset \mathcal{V}$ can have state as $v_j(T) = \langle P_i, \tau \rangle$ or $v_j(T) = \langle \perp, - \rangle$. Similarly, for an actuator node $v_j \in V_{\mathrm{A}} \subset \mathcal{V}$, either $v_j(T) = \langle K_i, \tau \rangle$ or $v_j(T) = \langle \perp, - \rangle$. Next, we define the state of an MCN as follows.

Definition 5 (State of an MCN). The state of an MCN, $\mathcal{N} = \langle \mathcal{P}, \mathcal{K}, \mathcal{G} = (\mathcal{V}, \mathcal{E}) \rangle$, at any time T, is collection of the states of all nodes and is defined as $z(T) = \langle v_1(T), v_2(T), \ldots, v_{|\mathcal{V}|}(T) \rangle$. The projection of the global state to some node $v_j \in \mathcal{V}$ at time T is defined as $z(T)[v_j] = v_j(T)$.

Assuming the transmissions have not been started in the initial state $z(0)$, the states of all nodes at $T = 0$ are assigned by the value, $\langle \perp, - \rangle$, reflecting their emptiness. We define the following set of state update rules (U1 to U3) for the MCN states, which describes the situations when different

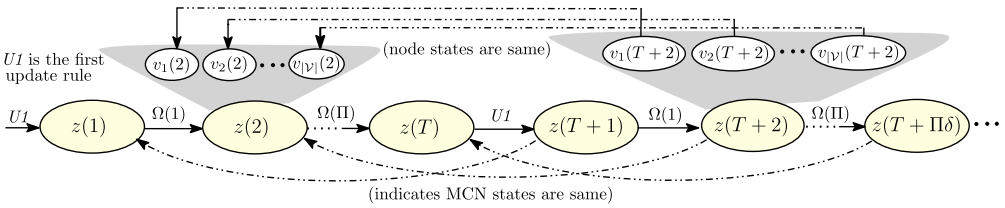


Fig. 4. Recurrent nature of the states of MCN.

nodes in the MCN change their states and thereby update the current state of the MCN. Let h_i be the sampling period of the i-th control loop, $\langle P_i, K_i \rangle$.

U1: Update due to sensory message generation. Following this rule, state update of the MCN mainly happens due to the state changes of the sensors nodes. According to the chosen l_i-length pattern s_i, a sensor node $v_i \in V_S$ of the i-th control loop, updates its state by containing the sensory message, $\langle P_i, \tau \rangle$, at time $\tau = (ql_i + m)h_i,\ \forall q = 0, 1, 2, \ldots, \forall m \leq l_i$, whenever $s_i[m] = 1$. Formally,

$$\forall i = 1, \ldots, n,\ if\ s_i[m] = 1,\ then\, z((ql_i + m)h_i)[v_i] = \langle P_i, (ql_i + m)h_i \rangle, q \in \{0, 1, 2, \ldots\}.$$

U2: Update due to actuation message generation. In this case, the control node, C, updates its state. If C receives the sensory message, $\langle P_i, \tau \rangle$, at some real time T, such that $t\delta \leq T < (t + 1)\delta$ (i.e., the t-th time slot), then it updates its state by generating the corresponding actuation message, $\langle K_i, \tau \rangle$, at the next time slot—that is, $(t + 1)$-th slot or at the real time $(t + 1)\delta$. Formally,

$$if\ z(T)[C] = \langle P_i, \tau \rangle,\ then\ z((t + 1)\delta)[C] = \langle K_i, \tau \rangle,\ t\delta \leq T < (t + 1)\delta.$$

U3: Update due to message forwarding. According to the routing solution Ω, a pair of nodes change their states by transmitting data and thereby update the MCN state. For an edge $(v_j, v_p) \in \Omega(t)$, where $\Omega(t)$ represents the set of transmissions that occurred at t-th time slot of Ω, we have

$$z((t + 1)\delta)[v_j] = \langle \perp, \_ \rangle\ and\ z((t + 1)\delta)[v_p] = z(t\delta)[v_j].$$

For an MCN, a potential routing solution Ω with period Π specifies a sequence of state transitions of the MCN such that there exists a state $z(T)$ *reachable* from $z(0)$ using the non-recurrent sequence of transmissions defined in Ω, and there exists a state $z(T + \Pi\delta)$ with $z(T) \overset{\Omega}{\Rightarrow} z(T + \Pi\delta)$ so that in the MCN states $z(T)$ and $z(T + \Pi\delta)$, the sensory/actuation messages at node v_j are the same for any $v_j \in \mathcal{V}$—that is, states of all nodes are the same in $z(T)$ and $z(T + \Pi\delta)$ as depicted in Figure 4. A routing solution is recurrent due to the recurrent traffic generated by corresponding loop execution patterns. For the chosen set of patterns, $\langle s_1, \ldots, s_n \rangle$, we define the *recurring time bound*, η, as $\eta = lcm(|s_1| \times h_1, \ldots, |s_n| \times h_n)$. Note that inside a time interval $[T, T + \eta]$, the loop execution patterns for all control loops repeat an integer number of times.

5.2 The SMT-Based Solution Framework

An SMT-based bounded model checking (BMC) framework along with the overall methodology of the co-synthesis process is presented in this section. The BMC approach generates the clauses by unwinding the states of MCN (at each time slot) and searches for a routing solution Ω along with the respective loop execution patterns, $\mathscr{S} = \langle s_1, \ldots, s_n, \rangle$ of n control loops.

Let $N(v_j)$ be the set of neighbor nodes of a node $v_j \in \mathcal{V}$. Consider the following representations:

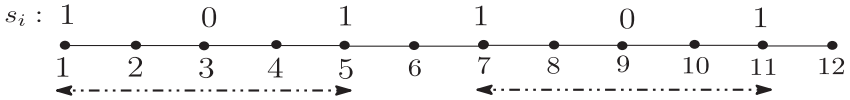
1: We define $v^t_{j,i,\tau}$ as the Boolean variable corresponding to $v_j \in \mathcal{V}$ such that $(v^t_{j,i,\tau} = True)$, when at the t-th time slot, v_j contains the message $\langle P_i, \tau \rangle$ or $\langle K_i, \tau \rangle$.

2: Any transmission $(v_j, v_p) \in \Omega(t)$ (when v_j transmits to v_p at t-th slot) can be equivalently represented by the constraint $(v^t_{j,i,\tau} = True) \Rightarrow (v^{t+1}_{j,i,\tau} = False) \wedge (v^{t+1}_{p,i,\tau} = True)$.
3: For the i-th control loop, corresponding to the l_i-length loop execution pattern, s_i, we set the Boolean variable, $s_{i,m}$, as $(s_{i,m} = True)$ if $s_i[m] = 1$, $0 \leq m \leq l_i - 1$.

Constraints for the initial condition. Assuming that initially at the first time slot (i.e., $t = 1$) all intermediate, control, and actuator nodes do not contain any valid message, we set *False* to all associated Boolean variables of these nodes. Recalling that $\eta = lcm(|s_1| \times h_1, \ldots, |s_n| \times h_n)$, we formally get

$$C1 : \forall v_j \in V_\mathsf{A} \cup V_\mathsf{I} \cup C,\ \forall i = 1, 2, \ldots, n,\ \tau = qh_i + 1, q = 0, 1, \ldots, \text{and } \tau \leq \eta,\ \left(v^1_{j,i,\tau} = False\right).$$

In case of a sensor node $\mathsf{S}_i \in V_\mathsf{S}$ of the i-th control loop, we initialize it according to the pattern s_i. For example, let $h_i = 2ms$, $s_i = 101$ and messages are generated up to the bound $\eta = 12ms$. Within η, s_i repeats twice and dispatches sensory messages at $1ms, 7ms$ (for $s_i[0]$), and $5ms$, *and* $11ms$ (for $s_i[2]$).



Therefore, we set the Boolean variable, S^t_i, corresponding to S_i, as $(\mathsf{S}^t_i = True)$ for the time slot t such that $t\delta = 1ms, 5ms, 7ms$, *and* $11ms$, and set *False* for all other time slots within the time of $12ms$. Generalizing this, if $s_i[m] = 1$, we set $(\mathsf{S}^t_i = True)$ for $t = \lfloor((ql_i + m)h_i)/\delta\rfloor$, $q = 0, 1, 2, \ldots$, and $t \leq \lfloor\eta/\delta\rfloor$. Since we synthesize the loop execution patterns, the clause for initializing the sensor node, S_i, according to $s_i, \forall i = 1, \ldots, n$ are of the following form:

$$C2 : \bigwedge_{i=1}^{n} \bigwedge_{m=0}^{l_i-1} \left[\left((s_{i,m} = True) \Rightarrow \bigwedge_{\substack{t=\lfloor((ql_i+m)h_i)/\delta\rfloor,\\ t\leq\lfloor\eta/\delta\rfloor,\ q=0,1,\ldots,}} \left(\mathsf{S}^t_i = True\right)\right) \wedge \left((s_{i,m} = False) \Rightarrow \bigwedge_{\substack{t=\lfloor((ql_i+m)h_i)/\delta\rfloor,\\ t\leq\lfloor\eta/\delta\rfloor,\ q=0,1,\ldots}} \left(\mathsf{S}^t_i = False\right)\right)\right]$$

For ensuring network schedulability of n loop execution patterns, $\mathscr{S} = \langle s_1, \ldots, s_n, \rangle$, the following six constraints as modeled in the form of suitable clauses have to be satisfied:

(1) *Existential constraint*: A message can exist only in one node at a particular time slot. If $z(T)[v_j] = \langle P_i, \tau\rangle$ *or* $\langle K_i, \tau\rangle$, then $\forall T, \forall v_p \in \mathcal{V}, p \neq j$, $z(T)[v_p] \neq \langle P_i, \tau\rangle$ *or* $\langle K_i, \tau\rangle$. Since messages for plant P_i are generated following the pattern, s_i, we have the clause

$$C3 : \bigwedge_{i=1}^{n} \left[\left((s_{i,m} == True) \Rightarrow \bigwedge_{\substack{\tau=(ql_i+m)h_i,\\ \tau\leq\eta,\ q=0,1,2,\ldots}} \bigwedge_{t=\lfloor\tau/\delta\rfloor}^{\lfloor(\tau+h_i)/\delta\rfloor-1} \left(\left(\sum_{v_j\in\mathcal{V}} Int\left(v^t_{j,i,\tau}\right)\right) == 1\right)\right)^2 \right.$$

$$\left. \wedge \left((s_{i,m} = False) \Rightarrow \bigwedge_{\substack{\tau=(ql_i+m)h_i,\\ \tau\leq\eta,\ q=0,1,2,\ldots}} \bigwedge_{t=\lfloor\tau/\delta\rfloor}^{\lfloor(\tau+h_i)/\delta\rfloor-1} \left(\bigwedge_{v_j\in\mathcal{V}} \neg v^t_{j,i,\tau}\right)\right)\right]$$

(2) *Validity constraint*: A message is valid in the network only for its sampling interval, and any node always contains such valid messages. Following Ω, if $z(T)[v_j] = \langle P_i, \tau\rangle$ *or* $\langle K_i, \tau\rangle$,

[2]For a Boolean variable b, *Int()* gives integer encoding of b as *Int(b) = 1, if b = True and Int(b) = 0, if b = False.*

for any $v_j \in \mathcal{V}$ and any loop i, $1 \le i \le n$, then $T - \tau \le h_i$. Formally,

$$C4: \bigwedge_{v_j \in \mathcal{V}} \bigwedge_{t < \lfloor \tau/\delta \rfloor \vee t > \lfloor (\tau + h_i)/\delta \rfloor} \left(v_{j,i,\tau}^{t} = False\right).$$

(3) *Transmission constraint*: A node can send only to its neighbor at any time slot t—that is, $(v_j, v_p) \in \Omega(t) \Rightarrow v_p \in N(v_j)$. Formally,

$$C5: \bigwedge_{v_j \in \mathcal{V}} \bigwedge_{i=1}^{n} \bigwedge_{\substack{\tau = (ql_i + m)h_i, \\ \tau \le \eta,\ q = 0,1,2,\ldots}} \left[\left(v_{j,i,\tau}^{t} = True\right) \Rightarrow \left(v_{j,i,\tau}^{t+1} = True\right) \vee \bigvee_{v_p \in N(v_j)} \left(v_{p,i,\tau}^{t+1} = True\right)\right]$$

Note that because of the existential constraint given in C3, for any node $v_o \notin N(v_j)$, the associated Boolean variables $v_{o,i,\tau}^{t+1}$ will be set to *False* by the SMT solver.

(4) *Mutual exclusion constraint*: A node can contain only one message at a time. If v_j contains $\langle P_i, \tau \rangle$ or $\langle K_i, \tau \rangle$ at the t-th slot, it cannot contain any other message at that slot. This is achieved by falsifying all other Boolean variables associated with v_j at the same time slot:

$$C6: \bigwedge_{v_j \in \mathcal{V}} \bigwedge_{i=1}^{n} \bigwedge_{\substack{\tau = (ql_i + m)h_i, \\ \tau \le \eta,\ q = 0,1,2,\ldots}} \left[\left(v_{j,i,\tau}^{t} = True\right) \Rightarrow \bigwedge_{d=1}^{n} \bigwedge_{\substack{\tau' = (ql_d + m)h_d,\ \tau \ne \tau' \\ \tau' \le \eta,\ q = 0,1,2,\ldots}} \left(v_{j,d,\tau'}^{t} = False\right)\right]$$

(5) *Conflict constraint*: A node cannot act as sender and receiver at the same time slot. If $(v_j, v_p) \in \Omega(t)$, then for every $v_o \ne v_j$, $(v_o, v_j) \notin \Omega(t)$. Equivalently this can be ensured by falsifying all Boolean variables associated with v_j at $(t+1)$-th slot, if v_j sends at t-th slot. This is because if v_j receives at the t-th slot, then following C5, we get $(v_{j,i,\tau}^{t+1} = True)$ at the $(t+1)$-th slot, but as it also sends at the t-th slot, then following C5 and C3, we get $(v_{j,i,\tau}^{t+1} = False)$ (representing v_j as empty). This becomes a contradiction, and hence it cannot receive any message at the same slot.

$$C7: \bigwedge_{v_j \in \mathcal{V}} \bigwedge_{i=1}^{n} \bigwedge_{\substack{\tau = (ql_i + m)h_i, \\ \tau \le \eta,\ q = 0,1,2,\ldots}} \left[\left(v_{j,i,\tau}^{t} = True\right) \Rightarrow \bigwedge_{d=1}^{n} \bigwedge_{\substack{\tau' = (ql_d + m)h_d,\ \tau \ne \tau' \\ \tau' \le \eta,\ q = 0,1,2,\ldots}} \left(v_{j,d,\tau'}^{t+1} = False\right)\right]$$

Note that if at the t-th slot v_j retains the message, then at $(t+1)$-th slot we find $(v_{j,i,\tau}^{t+1} = True)$. In that case, all other associated variables of v_j will be *False* due to the constraint in C6.

(6) *Deadline constraint*: For each control loop, the end-to-end communication (sense-actuation cycle) must be completed within its sampling period. Let $\mathtt{A}_{i,\tau}^{t}$ and $\mathtt{C}_{i,\tau}^{t}$ be the Boolean variables corresponding to an actuator node $\mathtt{A}_i \in V_{\mathtt{A}}$ of the i-th loop and the control node C, respectively, such that $(\mathtt{C}_{i,\tau}^{t} = True)$ and $(\mathtt{A}_{i,\tau}^{t} = True)$ if at the t-th time slot, C and $\mathtt{A}_i$ contain $\langle P_i, \tau \rangle$ and $\langle K_i, \tau \rangle$, respectively. Therefore, deadline constraint can be ensured as follows. If a sensory message for plant P_i, $i \in \{1, \ldots, n\}$, is generated at the t-th slot, then before the time slot $(t + \lfloor h_i/\delta \rfloor)$, the variables $\mathtt{C}_{i,\tau}^{t}$ and $\mathtt{A}_{i,\tau}^{t}$ must get a *True* value indicating their reception of $\langle P_i, \tau \rangle$ and $\langle K_i, \tau \rangle$.

$$C8: \bigwedge_{i=1}^{n} \left(\left(\mathtt{S}_i^{t} = True\right) \Rightarrow \bigvee_{d=1}^{\lfloor h_i/\delta \rfloor - 1} \left(\mathtt{C}_{i,\tau}^{t+d} = True\right) \wedge \bigvee_{d=1}^{\lfloor h_i/\delta \rfloor - 1} \left(\mathtt{A}_{i,\tau}^{t+d} = True\right) \right)$$

Note that C8 is the property of plant-controller model and becomes the specification to be verified by the model checker over the MCN state space. In contrast, C3 through C7 are properties of the network model used to progressively generate the MCN state space over time.

Performance constraint. Together with these six constraints mentioned previously, we add the following constraint for the desired performance requirement. For an l_i-length pattern, s_i, following Theorem 2 (see Section 4), we find $\kappa_{min,i}$ as the minimum number of loop executions—that is, the number of 1's in the pattern s_i is lower bounded by $\kappa_{min,i}$. Since in our synthesis methodology (presented in next section) we try to find the suitable choice of $\kappa_i \geq \kappa_{min,i}$ as the number of loop executions, therefore, to specify the number of 1's in s_i , $\forall i$, we add the following clause:

$$C9 : \bigwedge_{i=1}^{n} \left(\left(\sum_{m=0}^{l_i-1} Int(s_{i,m}) \right) == \kappa_i \right)^2$$

Note that satisfaction of this clause yields a loop execution pattern s_i for the i-th loop, ensuring the given settling time requirement of that loop.

Constraint for best choice of pattern. As discussed in Section 4.1, the best choice of loop execution pattern (i.e., with high QoC) relies on the uniform distribution of loop drops over the pattern. Let $s_{uni,i}$ be the l_i-length uniform pattern for the i-th control loop computed following Equation (9). Due to the schedulability constraints of all control loops over the network, it may not be possible to synthesize the uniform patterns for all of the loops. In that case, our target is to synthesize the pattern s_i for $i = 1, 2, \ldots, n$, that matches the respective uniform pattern $s_{uni,i}$ maximally. For this reason, we define the following penalty function that serves as the measure of equality of s_i with respect to $s_{uni,i}$. To check the equality of s_i and $s_{uni,i}$ bit by bit, first we define the function, $match(s_i, j)$, as follows:

$$match(s_i, j) = \begin{cases} 0 & if\ s_i[j] = s_{uni,i}[j] \\ -1 & if\ s_i[j] \neq s_{uni,i}[j] \end{cases} \tag{10}$$

Hence, we quantify the mismatch of s_i with respect to $s_{uni,i}$ by the function $MATCH(s_i) = \sum_{j=1}^{l_i} match(s_i, j)$. Our objective is to synthesize patterns for n loops, which match their respective uniform patterns maximally. Hence, we set the objective function $Maximize\ \sum_{i=1}^{n} MATCH(s_i)$, subject to the consolidated constraint $C1 \wedge C2 \wedge C3 \wedge C4 \wedge \bigwedge_{t=1}^{\lfloor \eta/\delta \rfloor} (C5 \wedge C6 \wedge C7) \wedge C8 \wedge C9$. We use an SMT optimizer [5] for the optimization. Among all of the satisfying assignments, the SMT optimizer returns the optimal assignment with respect to the objective function. We add suitable clauses for modeling this objective function by adding new Boolean variables for each bit of the uniform patterns of all of the control loops.

Our model checking procedure `SolveSMTOPT()` takes as arguments the MCN $\mathcal{N} = \langle \mathcal{P}, \mathcal{K}, \mathcal{G} \rangle$, the pattern lengths $\{l_1, \ldots, l_n\}$, execution counts within the pattern lengths (i.e., $\{\kappa_1, \ldots, \kappa_n\}$), and the recurring time bound η (see Section 5.1). The procedure automatically generates clauses $C_1, \ldots, C_9$ starting from the initial condition and by unrolling the MCN states up to η. Next, it checks for the satisfiability of the consolidated formula. If a satisfiable solution is found, it returns $True$ and the solution $\langle \mathcal{S}, \Omega \rangle$ with $\mathcal{S} = \langle s_1, \ldots, s_n \rangle$ as the schedulable combination of best choice of loop execution patterns for n loops and Ω as the associated routing solution of $\mathcal{S}$. This solution is extracted from the satisfiable assignment of the Boolean variables. Otherwise, it returns $False$. The next section describes the co-synthesis methodology where this procedure is employed.

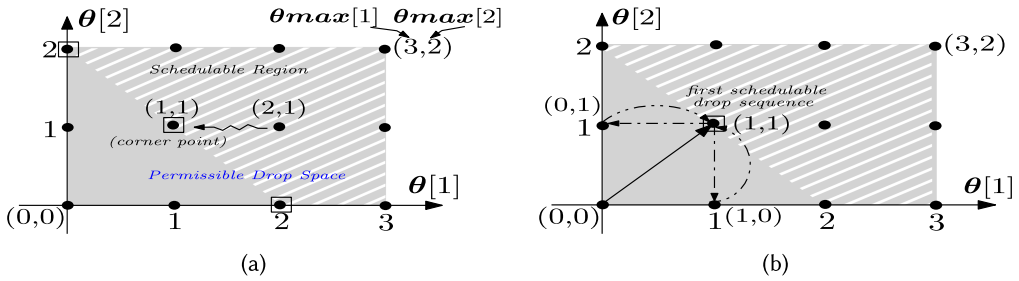


Fig. 5. Illustration of `Find_Corner_Point()`.

5.3 The Overall Co-Synthesis Methodology

Algorithm 1 outlines the proposed scheduling-routing co-synthesis approach. It takes the MCN specification together with the sampling periods, pattern lengths, and maximum drop bounds (see Section 4) of n loops as inputs and returns the solution $\langle \mathscr{S}, \Omega \rangle$ as output. It first computes the recurring time bound η (line 1), which is required for unrolling the MCN state. The vectors $\boldsymbol{l}$ and $\boldsymbol{\theta_{max}}$ contain the pattern lengths and the maximum drop count within those pattern lengths (obtained from Theorem 2), respectively, for n loops (line 2). The vector $\boldsymbol{\theta}$ represents the current drop count for n loops initialized by the value $\mathbf{0}$ in line 3. This essentially means that initially no drop is induced for all of the loops. We refer to $\boldsymbol{\theta}$ as *drop sequence* thereafter. In line 4, the algorithm checks the existence of any schedulable solution against the periodic execution of all of the loops (since $\boldsymbol{\theta} = \mathbf{0}$) by invoking `SolveSMTOPT`$(\mathcal{N}, \boldsymbol{l} - \boldsymbol{\theta}, \boldsymbol{l}, \eta)$. If a solution is found, the algorithm terminates there. This is because for being schedulable, there is no need to induce drops through a pattern guided approach, as schedulable solution exists for periodic execution of the loops. Otherwise, in line 6, it invokes another procedure, `Find_Corner_Point()`, for obtaining a pattern guided solution.

ALGORITHM 1: `Gen_Pattern_Guided_Scheduling_Routing`

Input: MCN: $\mathcal{N} = \langle \mathcal{P}, \mathcal{K}, \mathcal{G} \rangle$, Periods: $\{h_1, \ldots, h_n\}$, Lengths: $\{l_1, \ldots, l_n\}$, Max. Drops: $\{\theta_{max,1}, \ldots, \theta_{max,n}\}$
Output: $\langle \mathscr{S}, \Omega \rangle$, where $\mathscr{S} = \langle s_1, \ldots, s_n \rangle$

1 Compute $\eta = lcm(l_1 \times h_1, \ldots, l_n \times h_n)$; `// Computing the recurring time bound`
2 $\boldsymbol{l} = [l_1, \ldots, l_n]$; $\boldsymbol{\theta_{max}} = [\theta_{max,1}, \ldots, \theta_{max,1}]$; `// Initial Valuation`
3 $\boldsymbol{\theta} = \mathbf{0}$; `// Start with no drop situation for all of the loops`
4 **if** `SolveSMTOPT`$(\mathcal{N}, \boldsymbol{l} - \boldsymbol{\theta}, \boldsymbol{l}, \eta) ==$ *True* **then**
5 **return** NULL; `// Periodic solution exists; no need of pattern-based scheduling`
6 $[\boldsymbol{\theta_c}, \langle \mathscr{S}, \Omega \rangle]$ = `Find_Corner_Point` $(\mathcal{N}, \boldsymbol{\theta}, \boldsymbol{l}, \eta)$; `// Find the corner point - one schedulable option`
7 **return** $\langle \mathscr{S}, \Omega \rangle$;

Details about `Find_Corner_Point()`. This procedure provides a search mechanism that judiciously selects a drop sequence, $\boldsymbol{\theta}_c$, such that for n control loops, the n loop execution patterns synthesized with the drop count as selected in $\boldsymbol{\theta}_c$ provide a schedulable solution over the network without violating the desired performance requirement. We refer to $\boldsymbol{\theta}_c$ as the *corner point* relying on the fact that it contains a *minimal choice of drop count* of all of the loops for which schedulable solution exists. Let us elucidate this using an example.

Consider an MCN with two control loops where we are given with $\boldsymbol{l} = [10, 7]$ and $\boldsymbol{\theta_{max}} = [3, 2]$. In Figure 5(a), the blocked shaded region that we refer to as *permissible drop space* depicts

all possible valuation of the drop sequence, $\boldsymbol{\theta}$, such that $\boldsymbol{\theta}[i] \leq \boldsymbol{\theta_{max}}[i]$ for $i = 1, 2$. Within this region, the set of drop sequences for which a schedulable solution exists forms a *schedulable region*, $\mathbb{SR}$, as marked by the striped shaded region in Figure 5(a). It may be noted that in region $\mathbb{SR}$, the schedulability answer for some choices of drop sequences can be deduced from the schedulability answer of some other drop sequences. We formalize this aspect using the *dominance relation* as defined next.

Definition 6 (Dominance Relation). For an MCN with n control loops, a drop sequence $\boldsymbol{\theta'} \in \mathbb{SR}$ is dominated by another drop sequence $\boldsymbol{\theta} \in \mathbb{SR}$ if and only if $\boldsymbol{\theta'}[i] \geq \boldsymbol{\theta}[i]$, for $i = 1, 2, \ldots, n$ and $\boldsymbol{\theta'} \neq \boldsymbol{\theta}$.

This essentially signifies that if for $\boldsymbol{\theta}$ a schedulable solution exists, for the same MCN configuration a schedulable solution also exists for $\boldsymbol{\theta'}$ since $\boldsymbol{\theta'}$ allows more drops than $\boldsymbol{\theta}$. In Figure 5(a), $(1, 1)$ dominates $(2, 1)$ as marked by an arrow from $(2, 1)$ to $(1, 1)$. Based on this, we define the notion of corner point as follows.

Definition 7 (Corner Point). A drop sequence, $\boldsymbol{\theta}_c \in \mathbb{SR}$, is said to be a corner point if and only if no other drop sequence in $\mathbb{SR}$ dominates it. Therefore, each corner point contains a minimal choice of drop count for all of the loops such that further reduction in the drop count of any loop leads to unschedulability.

For the preceding example, $(0, 2)$, $(1, 1)$, and $(2, 0)$ are the corner points as marked by boxes in Figure 5(a). The procedure `Find_Corner_Point()` as outlined in Algorithm 2 searches for such a corner point.

At the start of Algorithm 2, the input drop sequence $\boldsymbol{\theta}$ has the value $\mathbf{0}$ as updated in its caller procedure, Algorithm 1. Starting with this value $\boldsymbol{\theta}$, Algorithm 2 updates $\boldsymbol{\theta}$ by allowing one drop for each loop (lines 2–5) while ensuring that the drop count never violates its upper bound as given in $\boldsymbol{\theta_{max}}$, and then it invokes `SolveSMTOPT`$(\mathcal{N}, \boldsymbol{l} - \boldsymbol{\theta}, \boldsymbol{l}, \eta)$ for checking the schedulability with this current value of $\boldsymbol{\theta}$. Note that in `SolveSMTOPT()`, the drop count for the loops in $\boldsymbol{\theta}$ are used for the formation of performance constraints as discussed in Section 5.2. This process continues until it gets a value of $\boldsymbol{\theta}$ for which a schedulable solution is found. This value of $\boldsymbol{\theta}$ resembles a drop sequence in the schedulable region $\mathbb{SR}$. In the preceding example, as shown in Figure 5(b), following this process we reach at $(1, 1)$ starting from $(0, 0)$. Next, it checks whether this $\boldsymbol{\theta}$ is a corner point (lines 7–18). If $\boldsymbol{\theta}$ is indeed a corner point, the procedure returns. If not, then it finds the corner point nearby θ. This search process is discussed next.

Corner point search. Let $\boldsymbol{\theta}$ is the current *start point* of this search mechanism. In the *for* loop in line 8, it iteratively selects a loop id i and reduces its drops count $\boldsymbol{\theta}[i]$ by one if the drop count does not reach its minimum bound 0 (lines 9 and 10). In this way, it picks up each drop sequence in the neighborhood of $\boldsymbol{\theta}$ that dominates the current $\boldsymbol{\theta}$ and checks schedulability for it in line 11. If no schedulable solution is found for any value of i, it declares this start point $\boldsymbol{\theta}$ as the corner point (lines 15 and 16), and the witness for satisfiability yields the solution pair $\langle \mathscr{S}, \Omega \rangle$ (line 17). Finally, it terminates in line 18. However, if schedulability holds for a value of i in line 11, then this current value of $\boldsymbol{\theta}$ (picked up from the neighborhood of start point) becomes the *new start point*, and the search process for the corner point gets reoriented based on it, reiterating the preceding steps. Algorithm 2 is developed following the corner point generation approach presented in Dixit et al. [11].

In the preceding example, to check if $\boldsymbol{\theta} = (1, 1)$ is a corner point or not, the nearby dominating drop sequences, $(0, 1)$ and $(1, 0)$, are picked up as indicated by the dashed arrows from $(1, 1)$ to $(0, 1)$ and $(1, 0)$ in Figure 5(b). For both of them, the schedulability check fails. Hence, $(1, 1)$ is a

ALGORITHM 2: `Find_Corner_Point`

Input: MCN: $\mathcal{N} = \langle \mathcal{P}, \mathcal{K}, \mathcal{G} \rangle$, A drop sequence: $\boldsymbol{\theta}$, Maximum drop bound: $\boldsymbol{\theta_{max}}$, Lengths: $\boldsymbol{l}$, Recurring bound: η

Output: Another drop sequence, the corner point: $\boldsymbol{\theta_c}$, The solution pair for this corner point: $\langle \mathscr{S}, \Omega \rangle$

```
1  N = {1, 2, ..., n};                                   // It contains set of loop ids
2  repeat
       // Allow one drop for each loop without violating maximum drop bound
3      for each i ∈ N do
4          if θ[i] ≠ θ_max[i] then
5              θ[i]=θ[i]+1;
6  until SolveSMTOPT(N, l − θ, l, η)== True;
   // At this point θ is a drop sequence in schedulable region. Now, check if it is a
      corner point using dominance relation
7  while True do
8      for each i ∈ N do
9          if θ[i] ≠ 0 then
10             θ[i]=θ[i]-1;    // Reduce one drop for the i-th loop and thus get a new drop
               sequence
               // Check schdulability for the current value of θ
11             if SolveSMTOPT(N, l − θ, l, η)== True then
12                 i=0;     // θ is schedulable. Hence, reorient search and check if it is
                   corner point
13                 break;
14             else θ[i]=θ[i]+1;    // Reverse back to the last schedulable valuation of θ
15     if i==n then
16         θ_c = θ;          //  This θ becomes the corner point, and hence store it in θ_c
           // Extract loop execution patterns and associated routing solution from the satisfiable
              assignments
17         ⟨S, Ω⟩ ← Report_Solution();  return (θ_c, ⟨S, Ω⟩);
```

corner point, and accordingly the best choice of loop execution patterns with a single drop in both of them are returned in $\mathscr{S} = \langle s_1, s_2 \rangle$ together with its associated routing solution in Ω.

6 EVALUATING FAULT TOLERANCE USING THE PROPOSED CO-SYNTHESIS FRAMEWORK

In this work, we propose an offline event-driven approach for monitoring MCN configurations and evaluating its fault-tolerance capability under potential future link failure situations by leveraging the proposed SMT-based co-synthesis framework as an oracle. The proposed approach checks the schedulability of the MCN anticipating various link failure situations. If a schedulable solution exists for an anticipated failure, it stores that solution in a look-up table. When the failure situation actually arises in real time, the contingency schedulable solution is already available for that failure situation, and then the next run of the approach starts over this current faulty MCN configuration. If no schedulable solution exists for an anticipated failure without violating the desired system performance, then this fact is reported to the system designer a priori so that necessary preventive measures can be taken to avoid this failure situation in real time.

6.1 Finding a Schedulable Solution against Simultaneous Link Failures

This section presents a methodology using the proposed SMT-based framework as an oracle to find a schedulable solution $\langle \mathscr{S}_d, \Omega_d \rangle$ if a set of an arbitrary d number of links fail simultaneously in a given MCN. Let $\langle \mathscr{S}_I, \Omega_I \rangle$ is the initial solution when there is no faulty link in the MCN and $E_d \subseteq \mathcal{E}$ is the set of d anticipated faulty links. We define R_i^I as the *routing path* of the i-th control loop, Σ_i, where $\forall\ t = 1, 2, \ldots, \lfloor \eta/\delta \rfloor$, $R_i^I(t) = (v_j, v_p) \subset \Omega_I(t)$ such that (v_j, v_p) transmits $\langle P_i, \tau \rangle$ or $\langle K_i, \tau \rangle$ at the t-th slot or $R_i^I(t) = \perp$. For each such faulty link, (v_j, v_p), we add clauses to falsify all of the transmissions following that link at any time slot t. Thus, the solver avoids those links for finding a new routing solution.

The complexity of the synthesis process can be reduced by refraining the SMT solver from exploring the entire search space, as it is highly likely that a solution may be found just by changing the routing paths of the loops while keeping the patterns the same. If a routing solution Ω_d is found, then we are done. Otherwise, new patterns are explored along with routing solutions. Algorithm 3 outlines this approach using the idea of *incremental SMT solving*. Let Θ be the set of clauses defined in the SMT formalism of `SolveSMTOPT()` (see Section 5.2), which is one input to Algorithm 3, together with the initial set of n patterns $\mathscr{S}_I$, n routing paths, $\{R_i^I\}_{i=1}^n$, and set of faulty links, E_d. The procedure starts by adding clauses to the original formula Θ augmenting suitable conditions for demarcating the set of faulty links. This is achieved by falsifying all Boolean variables associated with the node v_p if the node v_j currently contains any message (lines 1–3). Next, suitable clauses are added to Θ for considering as constant the available set of schedulable loop execution patterns (lines 4 and 5). If a routing solution, Ω_d, is found (lines 6–8), then this new solution is reported; otherwise, in line 10, we slacken the constraints in Θ by removing the previously added clauses for the original set of patterns and go for fresh co-synthesis by invoking Algorithm 1 in lines 11 and 12. A satisfying assignment corresponding to both $\mathscr{S}_d$ (patterns) and Ω_d (routing paths) returned by Algorithm 1 yields the final solution at line 12. Otherwise, Algorithm 3 terminates by returning NULL in line 13.

6.2 Monitoring Schedulability for $\mathscr{K}$-Look-Ahead to Link Failures

A brute-force technique to leverage the proposed SMT-based framework and develop an offline monitoring algorithm that periodically monitors the schedulability of the MCN configuration anticipating a finite number of link failure situations would be as follows. For resolving *$\mathscr{K}$-look-ahead to link failures* for a small constant $\mathscr{K}$, we can simply pre-compute all of the schedulable solutions anticipating various link failure scenarios (i.e., 1,2,3,.. up to $\mathscr{K}$ simultaneous failures in all possible subsets). However, this strategy is inefficient. The computation can be done efficiently by leveraging memoization of sub-problems. In this case, a sub-problem instance means to find a schedulable solution for some d links failures. When a sub-problem instance is first encountered during the execution of the proposed algorithm, its solution is computed and stored in the table. In the future, when the sub-problem is encountered, the stored solution is simply looked up and returned, thus avoiding an SMT call. In summary, we exploit the following properties of our co-synthesis:

(1) The co-synthesis problem for the set of d faulty links, E_d, becomes a sub-problem of all co-synthesis problems having $d' > d$ faulty links with E_d as its subset.
(2) The existence of a schedulable solution for the set of d faulty links, E_d, guarantees existence of a schedulable solution of the sub-problem with any $E \subseteq E_d$.

Let us consider the co-synthesis problem for monitoring $\mathscr{K}$-look-ahead to link failures in an MCN with topology given by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for a given $\mathscr{K}$. We enumerate all of the non-empty subsets of

$\mathcal{E}$ having maximum $\mathcal{K}$ elements as $E_1, E_2, \ldots, E_N$, whereas the empty subset is represented by E_0. Let $\langle \mathcal{S}_{E_f}, \Omega_{E_f} \rangle$ be the solution returned by Algorithm 3 on simultaneous failure of the links given by the set $E_f \subset \mathcal{E}$, with $|E_f| \leq \mathcal{K}$, and $R_i^{E_f}$ be the routing path for the i-th control loop obtained from Ω_{E_f} (see Section 6.1). To store the solution of N different faulty configurations, we use a list, $sol[0..N]$, where $sol[f]$ stores solution of the co-synthesis problems with E_f as the set of faulty links. We recursively store solutions to $sol[0..N]$ following the method outlined in Algorithm 4.

ALGORITHM 3: Solve_d_Link_Failure

Input: Initial Sch.: $\mathcal{S}_I = \langle s_1, \ldots, s_n \rangle$, Initial Routing: $\{R_i^I\}_{i=1}^n$, SMT-Clauses: Θ, Faulty Links: E_d
Output: $\langle \mathcal{S}_d, \Omega_d \rangle$

// Add following new clauses to SMT-Clause Set Θ
// For each faulty link falsify its transmissions
1 **for** *each* $(v_j, v_p) \in E_d$ **do**
2 $\quad \Theta = \Theta \wedge \bigwedge_{i=1}^{n} \bigwedge_{\substack{\tau=(ql_i+m)h_i, \\ q=0,1,2,\ldots}} \bigwedge_{m=1}^{l_i} \bigwedge_{t=1}^{\lfloor \eta/\delta \rfloor - 1}$
3 $\quad \Big((v_{j,i,\tau}^t == True) \Rightarrow (v_{p,i,\tau}^{t+1} == False) \Big)$
// For each loop add constraints for fixing pattern
4 **for** *each* i, *where* $i = 1, 2, \ldots, n$ **do**
5 $\quad \Theta = \Theta \wedge \bigwedge_{m=1}^{l_i} (s_{i,m} == val)$, such that $val = True/False$ if $s_i[m] = 1/0$;
// Search for routing solution, Ω_d
6 **if** SolveSMTOPT() *finds* Ω_d **then**
7 $\quad \mathcal{S}_d = \mathcal{S}_I$;
8 $\quad$ **return** $\langle \mathcal{S}_d, \Omega_d \rangle$
// Call Algorithm 1 for re-synthesis
9 **else**
10 $\quad$ Remove clauses from Θ that are added in line 4;
11 $\quad$ **if** Gen_Pattern_Guided_Scheduling_Routing() $\neq NULL$ **then**
12 $\quad\quad$ **return** $\langle \mathcal{S}_d, \Omega_d \rangle$;
13 $\quad$ **else return** NULL;

ALGORITHM 4: CoSynthesis_for_Link_Failure

Input: Initial Sch.: $\langle \mathcal{S}_I, \Omega_I \rangle$, Initial Routing: $\{R_i^I\}_{i=1}^n$, SMT-Clauses: Θ, Network Topology: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Output: $sol[0, .., N]$ storing solution for N non-empty subsets of $\mathcal{E}$ having maximum $\mathcal{K}$ elements

1 $sol[0] = \langle \mathcal{S}_I, \Omega_I \rangle$;
2 **for** *each* $E_f \subset \mathcal{E}$ *such that* $|E_f| = 1$ **do**
3 $\quad$ **if** $E_f \notin R_i^I, \ \forall i = 1, \ldots, n$ **then**
4 $\quad\quad sol[f] = \langle \mathcal{S}_I, \Omega_I \rangle$;
5 $\quad$ **else** $sol[f] =$ Solve_d_Link_Failures$(\mathcal{S}_I, \{R_i^I\}_{i=1}^n, \Theta, E_f)$;
6 **for** *each* $E_f \subset \mathcal{E}$ *such that* $1 < |E_f| \leq \mathcal{K}$ **do**
7 $\quad flag = 0$;
8 $\quad$ **for** *each* m *such that* $1 \leq m \leq |E_f|$ **do**
9 $\quad\quad E_{f_1} = E_f[1, .., m]$;
$\quad\quad E_{f_2} = E_f[m+1, .., |E_f|]$;
10 $\quad\quad$ **if** $sol[f_1] == NULL$ *or* $sol[f_2] == NULL$ **then**
11 $\quad\quad\quad sol[f] = NULL; \quad flag = 1; \quad$ break;
12 $\quad\quad$ **if** $E_{f_2} \notin R_i^{E_{f_1}}, \ \forall i = 1, \ldots, n$ **then**
13 $\quad\quad\quad sol[f] = sol[f_1]; \quad flag = 1;$ break;
14 $\quad\quad$ **if** $E_{f_1} \notin R_i^{E_{f_2}}, \ \forall i = 1, \ldots, n$ **then**
15 $\quad\quad\quad sol[f] = sol[f_2]); \quad flag = 1;$ break;
16 $\quad$ **if** $flag == 0$ **then**
17 $\quad\quad sol[f] =$ Solve_d_Link_Failure$(\mathcal{S}_I, \{R_i^I\}_{i=1}^n, \Theta, E_f)$;
18 $\quad$ **else** break;

If $f = 0$, no link has failed, and $sol[0]$ is updated with the initial solution $\langle \mathcal{S}_I, \Omega_I \rangle$ in line 1. For each single link failure, if the faulty link does not belong to the initial routing paths, $R_i^I \, \forall i = 1, \ldots, n$, it updates $sol[f]$ by $\langle \mathcal{S}_I, \Omega_I \rangle$ in lines 2 through 4. Otherwise, in line 5, it invokes Algorithm 3 for the solution. It computes $sol[f]$ for $1 < f \leq N$ in lines 6 through 18 by

taking advantage of the sub-structure of a schedulable solution. While computing E_f, we assume that solutions for all sets $E \subset E_f$ are available. It splits E_f into two disjoint subsets E_{f_1}, $E_{f_2} \subset \mathcal{E}$ such that $E_{f_1} = E_f[1, .., m]$ and $E_{f_2} = E_f[m+1, .., |E_f|]$ taking the m-th element as the pivot, for some $m \in \{1, 2, \ldots, |E_f|\}$ (lines 8-9). Now, $sol[f]$ can be obtained from the solution of the sub-problems for E_{f_1} and E_{f_2} as follows. If any one of these sub-problems has a *NULL* solution, then it updates $sol[f]$ by *NULL* (lines 10 and 11). It sets $sol[f] = sol[f_1]$ if none of the routing paths, $R_1^{E_{f_1}}, \ldots, R_n^{E_{f_1}}$, overlap with E_{f_2} (lines 12 and 13). Otherwise, it sets $sol[f] = sol[f_2]$ when none of the routing paths, $R_1^{E_{f_2}}, \ldots, R_n^{E_{f_2}}$ overlap with E_{f_1} (lines 14 and 15). It increments m until $|E_f|$ to check if any of the preceding cases occur. In that case, $sol[f]$ gets updated accordingly either by $sol[f_1]$ or by $sol[f_2]$. This is because once we get $sol[f]$ for some choice of m, there is no need to explore all other combinations of sub-problems further for other choices of m. If the preceding procedure does not find a solution using the solutions of the sub-problems, then it invokes Algorithm 3 (`Solve_d_Link_Failures()`) in lines 16 through 18.

6.3 Handling Link Failures

There exist several techniques for identifying faulty links in networks (see Ma et al. [21] and the references therein). It is better to use a hardware-based simple deterministic feedback considering the short period-based control often required in real-time wireless CPS. We consider Algorithm 4 to be an event-driven offline approach. Initially, Algorithm 4 is used to pre-compute contingency solutions for all possible combinations of up to $\mathscr{K}$ link failures. When a link failure occurs, the corresponding pre-computed solution is used to re-configure the communication schedule. By virtue of maintaining the contingency provision, the system is prepared to re-configure itself readily up to the first $\mathscr{K}$ failures. To avoid a downtime (i.e., reaching a state where no contingency schedule is available on time) in the network, Algorithm 4 must be run again so as to complete before the first $\mathscr{K}$ failures occur. When Algorithm 4 is run on a network that already has q faults, where $q \leq \mathscr{K}$, it aims to create contingency schedules for at most $\mathscr{K}$ new failures on the existing state of the network. It is to be noted that by virtue of its first iteration, it already has contingency schedules for up to $\mathscr{K}$ failures. If the second iteration is invoked after q faults have occurred, then it must determine schedules for between $\mathscr{K}+1$ to $\mathscr{K}+q$ faults.

It is important to guarantee stability when a controller switches between two loop execution patterns (and associated scheduling solutions) based on a link failure scenario. For this purpose, we can use the notion of a bridge between two loop execution patterns following the theory presented in Ghosh et al. [15]. A bridge from one pattern s_1 to another pattern s_2 is a stable transition if such a switching satisfies the minimum required execution count κ (obtained from our stability analysis) for all possible transitions starting from any potential position in s_1 and landing up in s_2. Note that such a notion of bridge always maintains the desired exponential stability guarantee.

7 EXPERIMENTAL EVALUATION

To demonstrate our method, we consider MCN examples constituting multiple control loops with the plants being different cart-inverted pendulums sharing a multi-hop network.

7.1 Experimental Setup

In our experiments, the pendulums differ by their parameters such as mass and length. We use LQR-based optimal control technique for designing controllers. For generating different network topologies, we follow the Erdős-Rényi random graph model [13], where we vary the number of loops, edges, nodes, and the total number of messages transmitted over the network. We have used Z3Opt [5], an SMT optimizer, as the underlying SMT solver and Z3py, a Z3 API in Python

Table 2. Inverted Pendulum Parameters

ID	$M(kg)$	$m(kg)$	$L(m)$	$b(N/m/sec)$	$I(kg.m^2)$
Pendulum-1	0.5	0.2	0.3	0.1	0.06
Pendulum-2	0.6	0.2	0.3	0.3	0.06
Pendulum-3	0.4	0.1	0.26	0.25	0.03

for front-end modeling of the co-synthesis framework. All experiments have been performed on a 2.40-GHz Intel Xeon E5620 processor with 16 cores and 50 GB of memory, running 64-bit Ubuntu 16.04.

Plant specification. The linearized continuous time state space model of the cart inverted pendulum is adapted from Messner and Tilbury [26]. The state variables are $x = [x_1^\mathsf{T}, x_2^\mathsf{T}, x_3^\mathsf{T}, x_4^\mathsf{T}]^\mathsf{T}$, with x_1 being the horizontal displacement of the cart, x_2 the linear velocity of the cart, x_3 the angular displacement of the pendulum, and x_4 the angular velocity of the pendulum. The control input u is the force that moves the cart horizontally, and the outputs are the horizontal displacement of the cart (x_1) and angular displacement of the pendulum (x_3). For our first experiment where we demonstrate the efficiency of our co-synthesis framework (Algorithm 1) in terms of control performance and control cost, we consider three pendulums sharing the multi-hop network of Figure 1. The loops differ from each other by their parameter values as provided in Table 2, where M and m denote the cart and pendulum mass, respectively, L the pendulum length, $g = 9.8m/s^2$ the gravitational constant, b the friction coefficient for the link where the pendulum is attached to the cart, and I the mass moment of inertia of the pendulum. As the performance requirement, settling time and the desired system norm of all pendulums are taken as $\tau_s = 5$ seconds and $\chi = 0.005$ rad, respectively.

Design of controllers. Given the preceding performance criteria and in the absence of any limitation on the network bandwidth, possible assignments of sampling periods for which suitable discrete controller design can ensure satisfactory closed-loop response are 90 ms, 80 ms, and 100 ms for the pendulums, respectively. We design LQR controllers for the pendulums discretized with the preceding sampling periods.

7.2 Experimental Results

We now provide the description of the performed experiments and the obtained results.

Finding minimum loop execution counts. Given the maximum perturbed value of 0.35 rad, we compute the pattern lengths for these three pendulums as $l_1 = 10, l_2 = 5$, and $l_3 = 9$, respectively, using the settling time, τ_s, and desired system norm, χ, as discussed in Section 4. Next, we use Theorem 2 to calculate the minimum loop execution count $\kappa_{min,i}$ for $i = 1, 2, 3$ as 7,4, and 6, respectively. Consequently, we get the maximum loop drops as $\theta_{max,1} = 3, \theta_{max,2} = 1$, and $\theta_{max,3} = 3$.

Exploring options for network schedulability. The suitable choice of sampling periods of 90 ms, 80 ms, and 100 ms cannot be implemented on our MCN since this choice of sampling rates generates message traffic in a pattern for which no communication schedule exists (as determined in line 4 of Algorithm 1). Following the traditional periodic approach [36], one option in such situations would be to reduce the periodic sampling rates. For this, we implement two existing methods of optimal rate selection through greedy heuristic and convex optimization as given in Saifullah et al. [36] by calculating the end-to-end network delay [35] under a certain choice of routing path for the loops. Both of these methods report the schedulable optimal choice of sampling periods for the loops as 90 ms, 100 ms, and 110 ms, respectively, for which we compute the optimal periodic controllers.

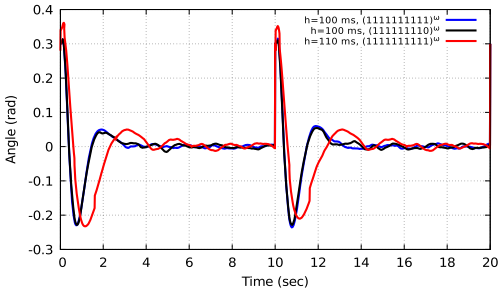


Fig. 6. Responses of Pendulum-3: proposed versus Saifullah et al. [36].

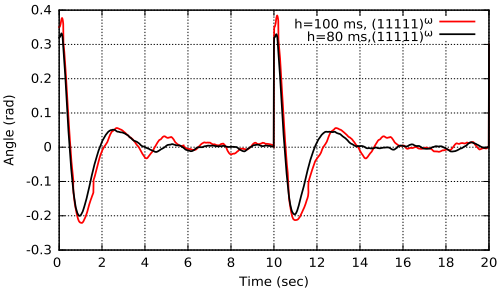


Fig. 7. Responses of Pendulum-2: proposed versus Saifullah et al. [36].

Another option is our pattern guided co-synthesis framework. With the original choices of sampling periods of 90 ms, 80 ms, and 100 ms, Algorithm 1 synthesizes a schedulable combination of best choice of loop execution patterns for these three loops by selecting a drop sequence $(1, 0, 1)$. This essentially means the best choice of loop execution patterns of loop 1 and loop 3 have one drop, whereas loop 2 avails fully periodic execution without any loop drop. The optimizer (see Section 5.2 for best choice of patterns) in the co-synthesis framework selects $s_1 = 0111111111$, $s_2 = 11111$, and $s_3 = 111111110$ as the resulting best choice of patterns for these three loops.

Improvement in output response. The advantage of our pattern guided approach over the traditional periodic approach in terms of a plant's response is given in Figures 6 through 8. Figure 6 compares the output response of Pendulum-3 for (i) the superior but infeasible periodic controller with a period of 100 ms (in blue), (ii) the inferior but schedulable periodic controller with a period of 110 ms obtained following Saifullah et al. [36] (in red), and (iii) a controller having a period of 100 ms but scheduled following the pattern $s_3 = 111111110$ as generated by Algorithm 1 (in black). Note that in Figure 6, the inferior but schedulable solution of 110 ms (option (ii)) is significantly worse than the superior but infeasible periodic solution of 100 ms (option (i)) in terms of settling time and peak overshoot. The comparison of the response of Pendulum-3 following option (iii), with the response for the infeasible superior controller of option (i) in Figure 6, establishes that the schedule generated by Algorithm 1 with instrumented loop drops performs just as well as the superior controller, which was unschedulable.

Similarly, for Pendulum-2, Figure 7 compares the responses obtained following the inferior periodic execution of 100 ms (in red) with the pattern guided execution of 80 ms (in black). Here, our SMT-based analysis does not put any loop drop for this loop, and thus Algorithm 1 produces $s_2 = 11111$. The degradation in peak overshoot and settling time for the inferior but feasible periodic solution of 100 ms is clearly evident when compared to the pattern guided solution of 80 ms. Observe that in this case, the pattern guided approach is able to preserve fully periodic execution with the period of 80 ms only because of the band width relinquished by the patterns of other loops. For Pendulum-1, Figure 8 compares the responses of pattern guided solution of 90 ms having the pattern $s_1 = 0111111111$ (in black) with the superior but infeasible periodic solution of 90 ms (in red). It is clear from the figure that there is not much difference in responses due to the occasional loop drops as instrumented by the SMT solver to fit in the network bandwidth. In Figures 6 through 8, all responses are taken by running a simulation of 20 seconds and tested under a disturbance scenario comprising a Gaussian state noise with covariance $R = 0.4 \times B_p B_p^{\mathsf{T}}$ overlapping with a periodic spike of 0.3 rad with a period of 10 seconds. Simulations are carried out using MATLAB version R2017b.

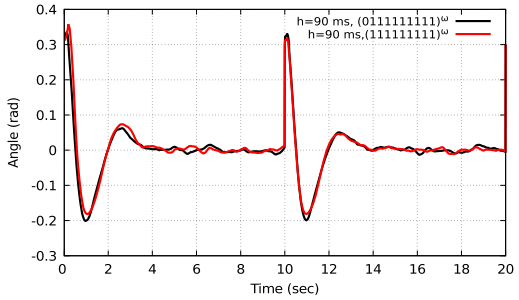


Fig. 8. Responses of Pendulum-1: Proposed versus Saifullah et al. [36].

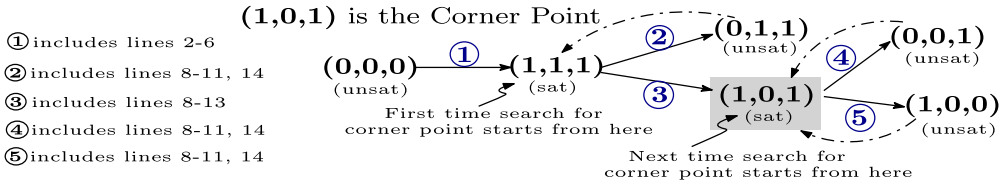


Fig. 9. Execution flow of Algorithm 2.

Improvement in control cost. To prove the superiority of our approach further, we compare the actual quadratic LQR cost (see Equation (8)) of both the design options. A high value of J indicates either a high deviation of the desired state or a high control effort needed to bring the state to its reference value. For the traditional periodic schedulable solution (i.e., 90 ms, 100 ms, and 110 ms), loop-1 and loop-2 exhibit LQR cost of 3970.2 and 6443.1, respectively, whereas for loop-3 it is 6745.5, manifesting a cumulative LQR cost of 3970.2 + 6443.1 + 6745.5 = 17158.8. In contrast, our approach exhibits a reduced cumulative LQR cost of 5344.6 + 3371.5 + 4702.3 = 13418.4, where 5344.6 for loop-1, 3371.5 for loop-2, and 4702.3 for loop-3 corresponding to the preceding choice of patterns. Thus, the solution generated using Algorithm 1 shows nearly 21.7% improvement in LQR cost when compared to the LQR cost of 17158.8 for the traditional approach.

Working details of Algorithm 1. To generate the solution, Algorithm 1 works as follows. Upon detecting unschedulability of the periodic execution of three pendulums in line 4 of Algorithm 1, it invokes Algorithm 2 in line 6. The stepwise execution flow of Algorithm 2 is shown in Figure 9. In this example, the underlying network of Figure 1 has 19 nodes and 54 edges. Within the recurring time bound of $\eta = 3.6$ seconds, 124 messages are transmitted over the network due to the execution of three pendulums for their choice of sampling periods as 90 ms, 80 ms, and 100 ms, respectively. The time taken by `SolveSMTOPT()` to check the satisfiability for each of these six different drop sequences, as shown in Figure 9, is on average 49 seconds. Note that the total time of 49 seconds includes SMT model generation time (t_{mod_gen}) of 45.5 seconds on average and model checking time (t_{mod_chk}) of 3.5 seconds on average. Hence, the total time (on average) taken by Algorithm 1 to generate the solution for this case is $6 \times (45.5 + 3.5) = 294$ seconds. We run this experiment for multiple MCN specifications as given in Table 3. As mentioned earlier, the specifications are obtained by varying the number of loops, edges, nodes, and messages transmitted within the respective recurring time. In Table 3, column 5 reports the total number of drop sequences (#DrpSeq) explored to reach the corner point, whereas column 6 and column 7 report the average time taken by `SolveSMTOPT()` for generating (t_{mod_gen}) and checking (t_{mod_chk}) the SMT model, respectively,

Table 3. Synthesis Time for Different MCN Specifications

ID	#Loops	#(Nodes, Edges)	#Msgs	#DrpSeq	t_{mod_gen} (s)	t_{mod_chk} (s)	Time (s)
MCN-1	2	(13, 30)	17	4	1.55	0.15	6.8
MCN-2	2	(13, 46)	17	4	1.75	0.15	7.6
MCN-3	3	(19, 54)	124	6	45.5	3.5	294
MCN-4	3	(19, 80)	124	6	52	4.5	339
MCN-5	5	(31, 95)	109	11	66.5	6	797.5
MCN-6	5	(31, 128)	109	11	83.5	7.5	1001

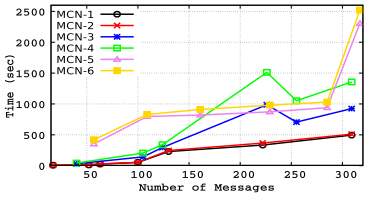


Fig. 10. Syn. Time versus #Messages.

Table 4. Syn. Time (in seconds) versus #Nodes-Edges

(N,E)	MCN1	MCN2	(N,E)	MCN3	MCN4	(N,E)	MCN5	MCN6
(13,30)	6.8	7.6	(19,54)	294	312	(31,95)	797	810
(20,80)	4.67	4.46	(22,80)	399	339	(31,128)	805	1001
(35,150)	8.81	8.57	(37,150)	675	792	(41,170)	812	1089
(45,200)	8.94	8.91	(47,200)	810	816	(51,230)	890	1120
(50,250)	9.27	9.23	(52,250)	798	870	(56,270)	1080	1225
(70,250)	9.8	10	(75,250)	810	890	(80,270)	1102	1240

Here, (N,E) denotes number of nodes (N) and edges (E). Columns 2 and 3 report synthesis time for MCN-1 and MCN-2, respectively, with respect to the varying node-edges pair as given in column 1. Similarly, we report time for MCN-3 (column 5) and MCN-4 (column 6) with respect to the varying node-edges pair in column 4, and for MCN-5 (column. 8) and MCN-6 (column 9) with respect to the node-edges pair in column 7.

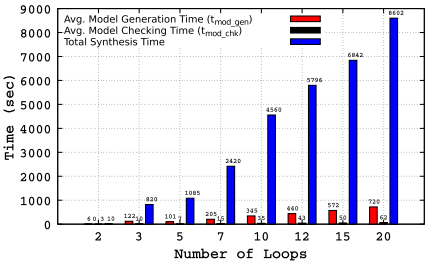


Fig. 11. Syn. Time versus #Loops.

for each of these drop sequences. Finally, column 8 shows the total synthesis time of Algorithm 1, which is obtained using the formula #DrpSeq×$(t_{mod\_gen} + t_{mod\_chk})$.

Scalability of Algorithm 1. To show the scalability of the proposed approach, we perform a series of experiments by increasing the number of messages, nodes, and edges for each of these preceding six MCN configurations: {MCN-1, MCN-2, ..., MCN-6}. We vary one parameter (e.g., (nodes, edges)) at a time while keeping fixed other parameters (e.g., number of loops and messages). For all of these cases, we report the average synthesis time of Algorithm 1 in Figure 10 and Table 4. Moreover, for further scalability testing, we perform experiments by varying the number of control loops on an MCN configuration having 50 nodes and 310 edges. We vary the number of loops choosing sampling periods from the range [60 ms, 240 ms]. Note that with the changes in loop count and the sampling period, the total number of messages in the network is changed (e.g., lower sampling period and a higher number of loops increase total messages). Figure 11 reports the time t_{mod_gen}, t_{mod_chk}, and total synthesis time taken by Algorithm 1 for this case. Clearly, our approach scales for each case of increasing the number of edges, nodes, messages, and loops with the synthesis time remaining reasonable. Given that the synthesis of schedules is an offline step before system deployment, the synthesis time seem acceptable. In retrospect, if we consider example domains like the multi-building wireless sensor network testbed of Saifullah et al. [36] and Ro et al. [32], the Ethernet-based control example of Mahfouzi et al. [24], and the wireless CPS testbed of Mager et al. [23], our experiments have derived synthesis results with CPS networks having comparable, if not higher, scalability parameters.

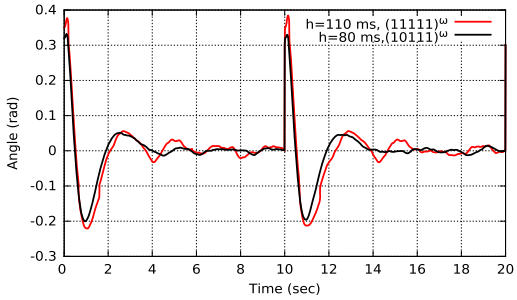


Fig. 12. Periodic versus Pattern when the link (S_2, I_4) fails.

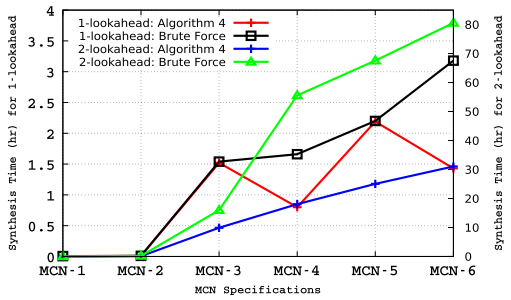


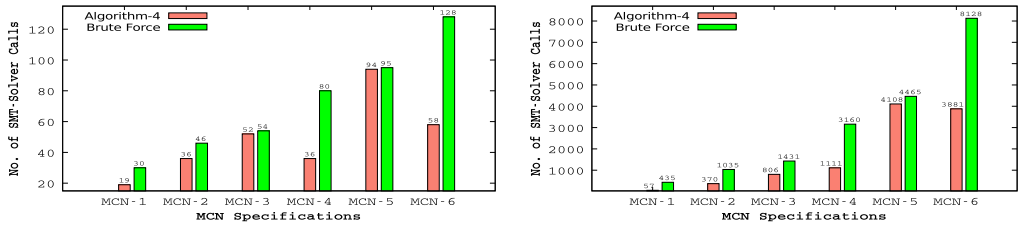
Fig. 13. Synthesis time: Algorithm 4 versus brute force.

Performance under link failure. The advantage of our approach over the traditional periodic approach is also evident under link failure situations. We choose three single link failure scenarios and discuss the results as follows. The solution, $\mathscr{S} = \langle 0111111111, 11111, 111111110\rangle$, together with its associated routing scheme,Ω, is the initial solution, $\langle \mathscr{S}_I, \Omega_I\rangle$, when no link failure is considered. *Scenario-1.* When the link (S_2, I_4) fails, Algorithm 3 returns a new combination of patterns, $\mathscr{S}_d = \langle 0111111111, 10111, 111111110\rangle$, incorporating a loop drop in the pattern s_2, whereas the patterns s_1 and s_3 are the same as in $\mathscr{S}_I$ for the original choice of sampling periods of 90 ms, 80 ms, and 110 ms, respectively. The cumulative LQR cost thus obtained is 5344.6 + 4215.9 + 4702.3 = 14262.8. In this faulty scenario, the traditional periodic approach generates a feasible solution for the choice of sampling periods as 90 ms, 110 ms, and 110 ms, respectively. In that case, we would have the cumulative control cost as 3970.2 + 7364.7 + 6745.5 = 18080.4, which is much larger than 14262.8. Figure 12 compares the output response of loop-2 on simulating our pattern-based solution of 10111 (in black) for the sampling period of 80 ms with the periodic execution for the period of 110 ms (in red). The benefit of our pattern-based scheduling is clearly manifested in the output responses. Note that for s_2 during the switching from 11111 to 10111, the stability remains guaranteed since every intermediate pattern obtained by taking the transition from any position of 11111 to starting position of 10111 (i.e., the initial 1), maintains the minimum execution count of $\kappa_{min,2} = 1$.

Scenario-2. Suppose the link (S_2, I_2) has failed. Algorithm 3 finds the solution just by changing the routing paths of all of the loops while keeping the patterns the same as $\mathscr{S}_I$.

Scenario-3. When the link (I_6, I_2) fails, no pattern guided solution is found by Algorithm 3 for the choice of periods of 90 ms, 80 ms, and 110 ms, respectively, without violating the minimum required execution counts.

We next present the experimental results for evaluating fault tolerance of the MCN specifications given in Table 3, up to 2-lookahead to link failures. Note that for each link failure scenario of an MCN specification, we solve an instance of the co-synthesis problem. But following Algorithm 4, we can get the solution of an instance of the co-synthesis problem from the solutions of its sub-problems without invoking the underlying SMT solver of Algorithm 3, thus reducing the total synthesis time. In contrast, the brute-force method uses the SMT solver of Algorithm 3 for the solution of each such instance of the co-synthesis problem. A comparative analysis between Algorithm 4 and the brute force method in terms of total synthesis time needed for all possible single (1-lookahead) and double links (2-lookahead) failures of each MCN specification are reported in Figure 13. For the same experiment, Figure 14 compares the number of instances of the co-synthesis problem for which the SMT-solver is called for the solution by Algorithm 4, and by



(a) For 1-lookahead to Link Failures

(b) For 2-lookahead to Link Failures

Fig. 14. Total number of times the SMT solver is called: Algorithm 4 versus brute force.

the brute force method. In case of Algorithm 4, for the rest of the co-synthesis problem instances, solution of the sub-problems are used. For example, as shown in Figure 14(a), for MCN-4, the SMT solver is called 36 times and the solutions of the sub-problems are used for the rest $(80 - 36) = 44$ cases.

8 CONCLUSION AND FUTURE WORK

In this work, we present an SMT-based framework for fine-grained scheduling using loop execution patterns for MCNs. The use of such patterns instead of fully periodic control widens the search space for schedulable solutions without compromising the stability and QoC. We devise heuristics working on the top of our SMT formulation and synthesize fault-tolerant, performance, and resource-aware MCN schedules. Our future research directions are as follows. We aim to develop an analytical framework for trading off control performance with respect to multi-hop wireless network power consumption. In this regard, we plan to extend the current framework with support for dynamic features such as runtime change in topology and adaptive transmission power control. Moreover, in the context of link failure analysis, we have established the usefulness of our monitoring approach from a synthesis perspective. In the future, we plan to extend this with a network simulation framework like OMNeT++ from a deployment perspective. Such network simulators may be used to generate packet drop and attempted re-transmission traces, which can be used to create temporal traces of injured MCNs. Using the monitoring framework periodically in that context will provide a network-aware CPS simulation framework for fault-tolerant MCN design.

REFERENCES

[1] Rajeev Alur, Alessandro D'Innocenzo, Karl H. Johansson, George J. Pappas, and Gera Weiss. 2009. Modeling and analysis of multi-hop control networks. In *Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'09)*. 223–232.

[2] R. Alur, A. D'Innocenzo, K. H. Johansson, and G. J. Pappas, and G. Weiss. 2011. Compositional modeling and analysis of multi-hop control networks. *IEEE Transactions on Automatic Control* 56, 10 (Oct. 2011), 2345–2357.

[3] Jia Bai, Emeka P. Eyisi, Yuan Xue, and Xenofon D. Koutsoukos. 2011. Distributed sampling rate adaptation for networked control systems. In *Proceedings of the 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS'11)*. 768–773.

[4] J. Bai, E. P. Eyisi, F. Qiu, Y. Xue, and X. D. Koutsoukos. 2012. Optimal cross-layer design of sampling rate adaptation and network scheduling for wireless networked control systems. In *Proceedings of the 2012 IEEE/ACM 3rd International Conference on Cyber-Physical Systems (ICCPS'12)*. 107–116.

[5] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. *νZ*—An optimizing SMT solver. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*. 194–199.

[6] Christian Choffrut and Juhani Karhumäki. 1997. *Combinatorics of words. In Handbook of Formal Languages—Volume 1: Word, Language, Grammar, G. Rozenberg and A. Salomaa (Eds)*. Springer, 329–438.

[7] Silviu S. Craciunas and Ramon Serna Oliver. 2016. Combined task- and network-level scheduling for distributed time-triggered systems. *Real-Time Systems* 52, 2 (March 2016), 161–200.

[8] A. D'Innocenzo, G. Weiss, R. Alur, A. J. Isaksson, K. H. Johansson, and G. J. Pappas. 2009. Scalable scheduling algorithms for wireless networked control systems. In *Proceedings of the 2009 IEEE International Conference on Automation Science and Engineering (CASE'09)*. 409–414.

[9] A. D'Innocenzo, M. D. Di Benedetto, and E. Serra. 2013. Fault tolerant control of multi-hop control networks. *IEEE Transactions on Automatic Control* 58, 6 (2013), 1377–1389.

[10] A. D'Innocenzo, M. D. Di Benedetto, and E. Serra. 2011. Link failure detection in multi-hop control networks. In *Proceedings of the 2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC'11)*. 5248–5253.

[11] Manoj G. Dixit, S. Ramesh, and Pallab Dasgupta. 2014. Time-budgeting: A component based development methodology for real-time embedded systems. *Formal Aspects of Computing* 26, 3 (May 2014), 591–621.

[12] D. A. Dowler. 2013. Bounding the norm of matrix powers. *Master's Thesis*. Brigham Young University.

[13] P. Erdös and A. Rényi. 1959. On random graphs, I. *Publicationes Mathematicae (Debrecen)* 6 (1959), 290–297.

[14] G. Fiore, V. Ercoli, A. J. Isaksson, K. Landernas, and M. D. Di Benedetto. 2009. Multihop multi-channel scheduling for wireless control in wirelessHART networks. In *Proceedings of the 2009 IEEE Conference on Emerging Technologies and Factory Automation (ETFA'09)*. 1–8.

[15] S. Ghosh, S. Dutta, S. Dey, and P. Dasgupta. 2017. A structured methodology for pattern based adaptive scheduling in embedded control. *ACM Transactions on Embedded Computing Systems* 16, 5s (Sept. 2017), Article 189, 22 pages.

[16] S. Ghosh, S. Dey, and P. Dasgupta. 2018. Co-synthesis of loop execution patterns for multihop control networks. *IEEE Embedded Systems Letters* 10, 4 (2018), 111–114.

[17] S. Ghosh, S. Dey, and P. Dasgupta. 2019. Performance and energy aware robust specification of control execution patterns under dropped samples. *IET Computers and Digital Techniques* 13, 6 (Nov. 2019), 493–504.

[18] R. Jacob, M. Zimmerling, P. Huang, J. Beutel, and L. Thiele. 2016. End-to-end real-time guarantees in wireless cyber-physical systems. In *Proceedings of the 2016 IEEE Real-Time Systems Symposium (RTSS'16)*. 167–178.

[19] B. Li, Y. Ma, T. Westenbroek, C. Wu, H. Gonzalez, and C. Lu. 2016. Wireless routing and control: A cyber-physical case study. In *Proceedings of the 2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS'16)*. 1–10.

[20] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen. 2016. Real-time wireless sensor-actuator networks for industrial cyber-physical systems. *Proceedings of the IEEE* 104, 5 (May 2016), 1013–1024.

[21] Q. Ma, K. Liu, Z. Cao, T. Zhu, and Y. Liu. 2015. Link scanner: Faulty link detection for wireless sensor networks. *IEEE Transactions on Wireless Communications* 14, 8 (Aug. 2015), 4428–4438.

[22] Yehan Ma, Dolvara Gunatilaka, Bo Li, Humberto Gonzalez, and Chenyang Lu. 2018. Holistic cyber-physical management for dependable wireless control systems. *ACM Transactions on Cyber-Physical Systems* 3, 1 (Sept. 2018), Article 3, 25 pages.

[23] F. Mager, D. Baumann, R. Jacob, L. Thiele, S. Trimpe, and M. Zimmerling. 2019. Feedback control goes wireless: Guaranteed stability over low-power multi-hop networks. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'19)*. 97–108.

[24] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng. 2018. Stability-aware integrated routing and scheduling for control applications in Ethernet networks. In *Proceedings of the 2018 Design, Automation, and Test in Europe Conference and Exhibition (DATE'18)*. 682–687.

[25] Rahul Mangharam and Miroslav Pajic. 2013. Distributed control for cyber-physical systems. *Journal of the Indian Institute of Science* 93, 3 (2013), 353–387.

[26] W. C. Messner and D. M. Tilbury. [n.d.]. Control Tutorials for MATLAB and Simulink: A Web-Based Approach. Retrieved December 30, 2019 from http://ctms.engin.umich.edu/CTMS.

[27] Jia Ning, Song YeQiong, and Simonot-Lion Francoise. 2007. Graceful degradation of the quality of control through data drop policy. In *Proceedings of the 2007 European Control Conference (ECC'07)*. 4324–4331.

[28] Marcelo Nobre, Ivanovitch Silva, and Luiz Affonso Guedes. 2015. Routing and scheduling algorithms for wirelessHART networks: A survey. *Sensors (Basel)* 15, 5 (2015), 9703–9740.

[29] M. Pajic, S. Sundaram, G. J. Pappas, and R. Mangharam. 2011. The wireless control network: A new approach for control over networks. *IEEE Transactions on Automatic Control* 56, 10 (Oct. 2011), 2305–2318.

[30] M. Pajic, S. Sundaram, J. Le Ny, G. J. Pappas, and R. Mangharam. 2010. The wireless control network: Synthesis and robustness. In *Proceedings of the 49th IEEE Conference on Decision and Control (CDC'10)*. 7576–7581.

[31] P. Park, J. Araújo, and K. H. Johansson. 2011. Wireless networked control system co-design. In *Proceedings of the 2011 Conference on Networking, Sensing, and Control*. 486–491.

[32] J. W. Ro, P. Roop, and A. Malik. 2015. Schedule synthesis for time-triggered multi-hop wireless networks with retransmissions. In *Proceedings of the 2015 IEEE 18th International Symposium on Real-Time Distributed Computing (ISORC'15)*. 94–101.

[33] Y. Sadi and S. Coleri Ergen. 2015. Joint optimization of communication and controller components of wireless networked control systems. In *Proceedings of the 2015 IEEE International Conference on Communications (ICC'15)*. 6487–6493.

[34] Indranil Saha, Sanjoy Baruah, and Rupak Majumdar. 2015. Dynamic scheduling for networked control systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control (HSCC'15)*. ACM, New York, NY, 98–107.

[35] Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. 2011. End-to-end communication delay analysis in wirelessHART networks. In *Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'11)*.

[36] Abusayeed Saifullah, Chengjie Wu, Paras Babu Tiwari, You Xu, Yong Fu, Chenyang Lu, and Yixin Chen. 2014. Near optimal rate selection for wireless control systems. *ACM Transactions on Embedded Computing Systems* 13, 4s (2014), Article 128, 25 pages.

[37] Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. 2010. Real-time scheduling for WirelessHART networks. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium (RTSS'10)*. 150–159.

[38] Weihuan Shu, Xue Liu, Zhonghua Gu, and Sathish Gopalkrishnan. 2008. Optimal sampling rate assignment with dynamic route selection for real-time wireless sensor networks. In *Proceedings of the 2008 Real-Time Systems Symposium (RTSS'08)*. 431–441.

[39] John R. Silvester. 2000. Determinants of block matrices. *Mathematical Gazette* 84, 501 (Nov. 2000), 460–467.

[40] F. Smarra, A. D'Innocenzo, and M. D. Di Benedetto. 2012. Optimal co-design of control, scheduling and routing in multi-hop control networks. In *Proceedings of the 2012 IEEE 51st Conference on Decision and Control (CDC'12)*. 1960–1965.

[41] Jianping Song, Song Han, Al Mok, Deji Chen, Mike Lucas, Mark Nixon, and Wally Pratt. 2008. WirelessHART: Applying wireless technology in real-time industrial process control. In *Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*. 377–386.

[42] Damoon Soudbakhsh, Linh T. X. Phan, Oleg Sokolsky, Insup Lee, and Anuradha Annaswamy. 2013. Co-design of control and platform with dropped signals. In *Proceedings of the 2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'13)*. 129–140.

[43] W. Steiner. 2010. An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium (RTSS'10)*. 375–384.

[44] E. Toscano and L. L. Bello. 2012. Multichannel superframe scheduling for IEEE 802.15.4 industrial wireless sensor networks. *IEEE Transactions on Industrial Informatics* 8, 2 (May 2012), 337–350.

[45] Gera Weiss and Rajeev Alur. 2007. Automata based interfaces for control and scheduling. In *Proceedings of the 2007 International Workshop on Hybrid Systems: Computation and Control (HSCC'07)*. 601–613.

[46] Marco Zimmerling, Luca Mottola, Prathush Kumar, Federico Ferrari, and Lothar Thiele. 2017. Adaptive real-time communication for wireless cyber-physical systems. *ACM Transaction on Cyber-Physical Systems* 1, 2 (Feb. 2017), Article 8, 29 pages.

Received January 2019; revised September 2019; accepted November 2019