Real-Time Syst (2012) 48:430–462
DOI 10.1007/s11241-012-9151-3

State-based scheduling with tree schedules: analysis and evaluation

Madhukar Anand · Sebastian Fischmeister · Insup Lee · Linh T.X. Phan

Published online: 28 April 2012

Abstract Distributed real-time systems require bounded communication delays and achieve them by means of a predictable and verifiable control mechanism for the communication medium. Real-time bus arbitration mechanisms control access to the medium and guarantee bounded communication delays. These arbitration mechanisms can be static dispatch tables or dynamic, algorithmic approaches.

In this work, we introduce a real-time bus arbitration mechanism called tree schedules that takes the best parts of both sides: It can be analyzed like static dispatch tables, and it provides a certain degree of flexibility similar to algorithmic approaches. We present tree schedules as a framework to specify real-time traffic and introduce mechanisms to analyze it. We discuss how tree schedules can capture application-specific behavior in a time-triggered state-based supply model by means of conditional branching built into the model. We present analysis results for this model specifically aiming at schedulability in fixed and dynamic priority schemes and waiting time analysis. Finally, we demonstrate the advantages of state-based supply over stateless supply by means of two case studies.

M. Anand
Cisco Systems, San Jose, USA
e-mail: anandm@seas.upenn.edu

S. Fischmeister (✉)
University of Waterloo, Waterloo, Canada
e-mail: sfischme@uwaterloo.ca

I. Lee · L.T.X. Phan
University of Pennsylvania, Philadelphia, USA

I. Lee
e-mail: lee@cis.upenn.edu

L.T.X. Phan
e-mail: linhphan@cis.upenn.edu

Keywords Real-time networking · Scheduling · Network code

1 Introduction

Modern real-time systems realize distributed applications with timeliness requirements. An intrinsic property of such a system is that the correctness of the system depends on the correctness of values and the correctness of timing. This implies that a correct value at an incorrect time can lead to an error. Consider a car with a brake-by-wire system, where the pedal communicates to the brakes when to apply force to the tires: In this system, a correct value means that the brakes apply force to the tires only when the driver hits the brake pedal, and correct timing means that the time between the two events of "hitting the pedal" and "applying force" should be bounded. It is obvious that the system is only useful, if both—correct timing and correct values—are guaranteed.

Distributed real-time systems add the complexity of decentralized control to a shared communication medium in the design process. In a decentralized system, all the connected network nodes act independently and could access the medium concurrently, resulting in message collisions. Retransmitting messages in the wake of collisions would resolve this problem, but could make it difficult to bound the end-to-end latency of messages. Another approach is to use dispatch tables in a TDMA fashion. However, these methods are designed for the worst-case and always execute this worst case, whereas our approach allows designing for the worst case but executing the worst case only if necessary and a better case otherwise. A primary research goal therefore is to design effective real-time message arbitration without all these shortcomings.

This goal has been the motivation to research elaborate mechanisms and led to this work. Here we introduce *tree communication schedules,* or in short, tree schedules, which provide a structured way to represent and program access control for time-triggered communication. Our model of programming communication is distinct from standard TDMA schemes in that our model makes use of application-specific state information at each network node to decide when to communicate. Such information is vital in increasing the operational range and flexibility of applications by, for instance, allowing retransmissions in specific cases or increasing throughput by reclaiming resources and turning off unused components. In our model, the state is kept by the use of shared variables. The variables are updated when messages are transmitted, and the communication schedule depends on the value of these variables.

Our approach differs from other approaches that it aims to use application-specific state information. Traditional scheduling techniques also maintain state such as a ready queue or priority information. Our approach tries to use application-specific information such as the contents of a transmitted message containing a sensor reading, the number of elements in a transmission or receive buffer, or the current temperature sensor reading. This application-specific information is then incorporated into the schedule to make decisions that improve the performance of the system.

In this work, we introduce tree schedules and provide analysis results with deadlines and with known path probabilities. Specifically, the contributions of this work include:

– We show how we can express time-triggered state-based supply models with tree schedules and show its advantages over stateless supply.
– In the presence of deadlines, we present results on the schedulability analysis of time-triggered state-based supply w.r.t. both fixed and dynamic priority schemes.
– In the presence of known path probabilities, we present calculations for waiting time analysis of messages serviced under state-based supply.
– We demonstrate the utility of the framework for control applications and jittery/bursty environments by performing a case study with an inverted pendulum system and a video streaming application.

The remainder of the article is structured as follows: Sect. 2 introduces tree schedules as means for time-triggered state-based supply and defines their structure and properties. Section 3 contains our results for schedulability analysis of message transmissions with tree schedules with fixed and dynamic priority schemes. Section 4 explains how to calculate average waiting times for messages scheduled with tree schedules. Section 5 shows a case study of an inverted pendulum and demonstrates the advantages of time-triggered state-based supply over stateless supply. Section 6 shows a case study of a jittery/bursty video streaming application which also shows the advantages of time-triggered state-based supply over stateless supply. Section 7 briefly summaries the implementation efforts around tree schedules. Finally, we close the paper in Sect. 8 with a brief summary of our work.

2 Tree schedule and definitions

Informally, a tree schedule is a structure consisting of locations and transitions between these locations such that its underlying graph is a directed tree. Each location of the tree schedule may specify a transmission on the shared network, and each transition is guarded by a condition that needs to be met before proceeding to the next location. An empty location indicates that no transmission is scheduled in the network.

2.1 Model overview

Figure 1 provides an overview of the model we use for our distributed real-time systems. A distributed real-time system consists of a set of *tasks* (τ_i). Tasks communicate via *messages* (m_i) which are encapsulated into packets. For this work we assume that every message fits into one packet. Messages are produced according to a specified arrival model such as periodic (Liu and Layland 1973), periodic with jitter (Baruah et al. 1997), recurring branching task (Baruah 1998), control flow graph (Pop et al. 2000). Each task enqueues its messages in an *output queue* (q_i). The system contains multiple such queues, and each task uses exactly one for its messages. Several tasks may share one queue. Messages in an output queue are ordered according to a specified policy. Such a policy could be earliest deadline first or rate monotonic.

Messages are communicated in slots according to a tree schedule (Ω). A slot has a start time and a length, and different slots may have different lengths. However, all slot assignments and slot lengths are specified offline. A slot assignment is the

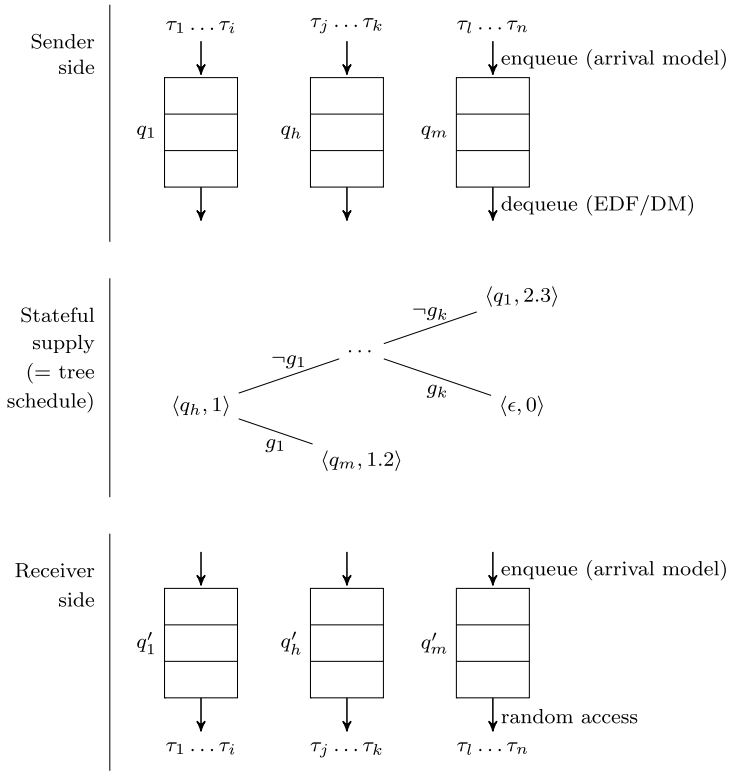


Fig. 1 Model overview

mapping of messages and queues to individual slots. Slots, with their assignments and lengths are specified as a state-based supply which is modelled as a tree schedule. During a slot, the sending network node has exclusive write access to the network and communicates as many messages from the specified queue as possible. If the queue is empty, then the slot will remain unused.

On the receiving side, the network node enqueues received messages into an *input queue* (q'_i). The system contains several input queues. Each input queue is mapped to one output queue, so messages from output queue q_1 will be received into input queue q'_1. Tasks dequeue messages from the input queue.

We assume that time is given in discrete units. Further, we assume the presence of a global clock and that all times are measured on this clock. The communication medium provides an atomic broadcast service (a common assumption for our target domain of embedded systems, which often use single segmented bus networks or can be achieved by special hardware (Kopetz 1997)); therefore, either all network nodes receive a message or none of them do. The system structure including the tasks, the queues, and the task to queue mapping is known in advance and static. All necessary data structures including the queues and the state-based supply are generated offline.

To ensure that the approach schedules, the developer can only include as many decisions in the schedule as resources are available for the scheduling system. So

for example in our implementation for switched Ethernet (Carvajal and Fischmeister 2010) on the NetFPGA platform, we operate the board at 125 MHz (62.5 for the cores) which is sufficient to communicate at line speed for 1 Gbps Ethernet. On the NetFPGA, we have 280 kB memory available for application-specific guards. On the ML403 implementation (Fischmeister et al. 2009) we operate at 100 MHz for 100 Mbps Ethernet and have about 8 kB memory available for guard.

2.2 Definition of a tree schedule

In our model of a tree schedule, a location is similar to a vertex. The tree schedule in Fig. 1 shows several locations (vertices), for example one being $\langle q_h, 1\rangle$.

Definition 1 (Tree Schedule) A *Tree Schedule* (TS) Ω is represented by a tuple $\langle V, \mathcal{V}, Q, sl, K, T\rangle$ where

- V is a finite set of locations,
- $\mathcal{V}$ is a finite set of variables,
- Q is a finite set of queues that hold messages to be transmitted,
- $sl : V \hookrightarrow Q \times B$ specifies the queue from which messages are transmitted in this location and a clock constraint,
- K is a finite set of clocks $|K| \geq 1$ relative to the shared global clock,
- $T \subseteq V \times G \times 2^K \times 2^V$ is a set of transitions such that the underlying graph (V, T) is a directed tree.

We denote the root location by $v^0 \in V$. The set of leaf locations is denoted by $V^F \subseteq V$. The mapping sl defines for each location a queue q (or ϵ) and a set of clock constraints on that location. The queue q contains messages communicating variables. Clocks are discrete time represented by $\mathbb{Q}$, and the clock constraint $b \in B$ is of the form $k = q$ where k is a clock and $q \in \mathbb{Z}^+$.

A transition $(v, g, s, v') \in T$, denoted by $v \xrightarrow{g,s} v'$, defines a source location v, an enabling condition g, a finite set of clocks s to be reset, and a target location v'. For reducible transitions, we will later expand v' to a set of alternative target locations. The enabling condition is any decidable function over the variables $\mathcal{V}$. The enabling conditions $g_v^1, \ldots, g_v^m$ of transitions leaving one location v must satisfy the following conditions: (1) any two enabling conditions g_v^i and g_v^j are mutually exclusive and (2) the set of enabling conditions is exhaustive, i.e., $\bigvee_{j=1}^m g_v^j = \text{true}$. These conditions ensure that the schedule always makes progress, and a reset to v^0 will always occur eventually when the schedule reaches a leaf. In practice, the enabling conditions are typically functions of the state of the schedule, local variables, and transmitted values.

We assume that the enabling conditions are evaluated instantaneously. In practice, however, evaluating the conditions will consume time but we treat this as the overhead of implementing tree schedules. The metrics for measuring such overheads are treated in our earlier work (Anand et al. 2006).

We define *liveness* to mean that some locations will always eventually be reached. Note that liveness implies deadlock freedom. Tree schedules guarantee liveness by construction, because in each tree schedule, exactly one guard g_v^i is always enabled

and a reset will always eventually occur. Consequently, the schedule's root will always eventually be executed and all locations prior the first decision will always be executed (reachable with a probability of 1). The set T is partitioned into sets T_m and T_r, where T_m contains all transitions that have exactly one destination (i.e., for all $\langle v, g, s, v' \rangle \in T_m$ we have $|v'| = 1$). We call the transitions in T_m *minimum transitions* and the ones in T_r *reducible transitions*. Each network node can have multiple outgoing transitions, and each transition could lead to multiple locations; here, the system can choose to continue in one of the locations. This mechanism may encode alternative, equivalent schedules which are all acceptable to the application, which uses the schedule. Our notion of equivalence only considers the resulting resource supply. Hence, two schedules are considered equivalent if they both satisfy the application demand, although they might satisfy it differently or even overprovision it. For example, if an application requires nine slots out of ten, then the two schedules, where one provides ten out of ten and the other provides nine out of ten, are equivalent.

Definition 2 (Path) A path from location v^m to location v^{m+n}, denoted by $\text{path}_\Omega(v^m, v^{m+n})$, is a sequence of locations of a tree schedule Ω: $v^m \xrightarrow{g_{m+1}, s_{m+1}} \cdots \xrightarrow{g_{m+n}, s_{m+n}} v^{m+n}$, where $\langle v^{m+i-1}, g_{m+i}, s_{m+i}, v^{m+i} \rangle \in T$ for all i, $1 \leq i \leq n$. A *complete path* is $\text{path}_\Omega(v^0, v)$ with $v \in V^F$; the set of all complete paths is denoted by $\mathcal{P}$.

Two paths will be called *equivalent*, if both can generate the same sequence of transitions from the start and end location of the path. We denote two equivalent paths p_1 and p_2 by $p_1 \equiv p_2$. The *duration of a path*, $\text{dur}(p)$ with $p = \text{path}(v^m, v^{m+n})$, is the time it takes to transit from v^m to v^{m+n}. If all complete paths of Ω have the same duration, then we say that Ω is an *isochronous* tree schedule. Otherwise, it is said to be *anisochronous*. In an isochronous tree schedule, the duration of a complete path is defined to be the period of recurrence (P).

Finally, we note that the probability p_r of $\text{path}(v^m, v^{m+n})$ is the probability of reaching v^{m+n} starting from v^m. This probability is useful to reason about composition of anisochronous tree schedules where there is no fixed period of recurrence of the start location.

2.3 Execution semantics

The execution semantics of a tree schedule as it is executed in the network layer (see Fischmeister et al. 2007) for a complete architecture overview) is as follows: The tree schedule starts execution at v^0 with all clocks set to 0. The variables in $\mathcal{V}$ are assigned to some default values by the user. The network layer then transmits messages from queues as specified in schedule's mapping sl (or if this is ϵ, then it remains idle). The network layer remains in the current location v until it can make a suitable transition to one of v's children (recall that exactly one g will be enabled by definition). This happens when the clock constraint of the current location evaluates to true. The decision about which transition is taken is made by first evaluating all the enabling conditions and then making a transition to the one that is enabled. If this transition leads to multiple locations (representing equivalent schedules), then the system chooses the first one that is stored in the list of destination locations. Since

we use a distributed system, each node executes this independently. This requires distributing information relevant to making the decision to all nodes and assumes a reliable communication medium. These two problems together with verifying the tree schedule are extensively discussed here (Fischmeister et al. 2007).

The execution continues until a leaf location is reached. In the leaf location ($v \in V^F$), the schedule *resets* immediately after the clock constraint becomes true. This means that (1) the schedule starts again at the root v^0 and (2) all the clocks K are set to 0 during the reset. Variables get updated at run time through messages. If one network node transmits a new value for a variable, then all nodes use this new value for evaluating enabling conditions. For more details about this, we refer the reader to Fischmeister et al. (2006).

2.4 Example

Assume a distributed real-time system in which we have to communicate one sensor value. Data integrity requirements specify that the system must tolerate failures at the sensor reading hardware. We assume that the variation in the reading is bounded by δ unless a fault occurs, faults happen with a temporal distance of at least one communication round, fault occur in reading the value and are independent of faults in the controlling system and the tree schedule, and we assume that the communication medium between the units is reliable. Our approach requires three separate sensor reading units. Each unit uses its own sensor and produces a value. The units then need to agree on what the real sensor value is. To implement this, we need three values x_1 to x_3 communicated in queues q_1 to q_3. The units use these values to communicate each one's sensor reading. With our assumptions, a simple majority vote is sufficient. Figure 2 shows the resulting tree schedule for this example. In the figure, a tuple (a, b) represents the queue from which a message is being transmitted (a) and the time spent in that location (b). The two branches A and B only differ in the ordering of the initial two sensor readings. Note, that if two values already create a decisive vote, then it is unnecessary to communicate the third one.

We can encode the application by the TS $\langle V, \mathcal{V}, Q, sl, K, T\rangle$:

– $V = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$

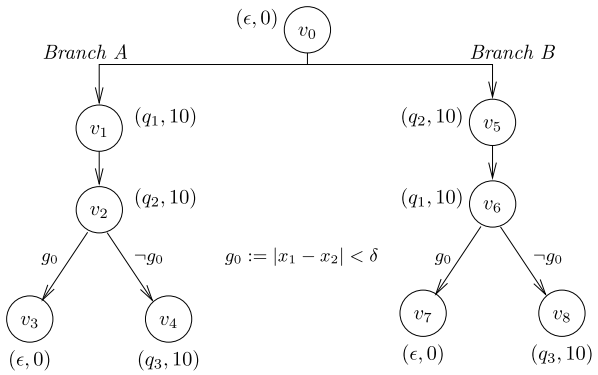


Fig. 2 Example tree schedule

– $\mathcal{V} := \{x_1, x_2, \delta\}$, $K = \{k\}$,
– $Q = \{q_1, q_2, q_3\}$ with one queue for each sensor,
– sl as shown in the figure for each location v,
– $T_r = \{\langle v_0, \emptyset, \{k\}, \{v_1, v_5\}\rangle\}$, and T_m contains all other transitions.

The guard $g_0 := |x_1 - x_2| < \delta$ checks whether the first two sensor readings are already within a bound δ. The clock k is reset on every transition.

The execution of the schedule is as follows: The system starts executing at the root location v_0. Since the clock constraint in location v_0 is $clk = 0$, the system immediately selects one of the branches and continues execution. Let us assume that it continues on Branch A: The system enters location v_1. There it transmits value x_1 and waits for the clock constraint ($clk = 10$) to become true. It then proceeds to location v_2 where it transmits value x_2 and waits again for 10 time units. Then, the system evaluates the guard g_0, and depending on result, it will enter location v_3 or v_4. In location v_3, the system will immediately reset and continue at the root location. In location v_4, the system will transmit the third value x_3 before it resets.

2.5 Relation to other state-based models

State-based methods have also been recently developed in the context of formal modeling and analysis of stream processing systems. For instance, event count automata (Chakraborty et al. 2005) and its hybrid techniques (Phan et al. 2007, 2008) have been used to model bursty characteristics of event streams and state-dependent scheduling policies. Similarly, timed-automata (Alur and Dill 1994) have also been used to describe complex task arrival patterns and processing semantics (see e.g., Abdeddaïm et al. 2003; Hendriks and Verhoef 2006). It has been shown for these models that, by capturing the state-information, not only are these models able to describe more complex practical systems, but also achieve a better analysis accuracy. Tree schedules can be considered as a special class of automata-theoretic methods, where the automata structure is limited to a tree structure. As a result, they allow for more efficient analysis techniques, which cannot be achieved using automata verification, while still being highly expressive.

On the implementation side, mode changes (Real and Crespo 2004) as implemented in, for instance, the Time Triggered Architecture (Kopetz 1997) permit some flexibility to adjust the current schedule to the application's demands. The developer can specify different schedules for different modes and then switch between modes at run time. Modern protocols such as FlexRay (FlexRay Consortium 2004) or PowerLink Ethernet (Ethernet 2003) can also adjust in how the application uses the dynamic segments. However, compared to tree schedules and their implementation (see Sect. 7) they offer less flexibility and oversight as tree schedules encapsulate all possible adaptations in one representation and tree schedules can make decisions after each communication slot.

2.6 Generation of tree schedules

It is important to understand how and when tree schedules occur in systems. While it is outside the scope of this work, we still want to refer to related work that addresses this question.

Tree schedules and other state-based schedules are generated from high-level specifications. Related work has shown to generate such schedules from control systems (Weiss et al. 2009) specifications, synchronous programs or Simulink (Potop-Butucaru et al. 2009), and from regular specifications (Alur and Weiss 2008). We also wrote a demonstrator to use state-based schedules in the Simulink TrueTime framework (Cervin et al. 2003). In generating these schedules is a topic of ongoing research.

3 Analysis with deadlines: schedulability

If deadlines for messages are known, then their schedulability analysis can be performed. In this section, we consider the schedulability analysis of messages given a periodic task model that generates these messages. We consider the cases when the messages are scheduled for transmission either with the Earliest Deadline First (EDF) or Rate Monotonic (RM) policy and the resource (network) is provided according to a tree schedule. We consider EDF and RM, because both are optimal algorithms in their class.

We assume that messages are generated by tasks, and that messages are released every time a task is executed. Each task has a fixed resource (network) requirement e that specifies how many network slots are needed for the messages of a task to be successfully communicated. The tasks also have a deadline d, by which the transmission should be completed. The released messages are enqueued into a shared queue, from which they are scheduled either according to EDF or RM scheduling policy. We associate all the tasks that share a queue with a workload W. We now present the schedulability analysis for such a system with periodic tasks. Note that preemption of messages cannot occur, because we assume that the whole message always fits into a single slot.

The network *demand bound function* (dbf) of a task that releases a message m provides an upper bound on the amount of resource required to meet the deadlines of all the released messages. For a time interval length t, $\mathrm{dbf}(t)$ gives the largest network demand in any time interval of length t. The $\mathrm{dbf}(t)$ includes all the task instances with messages that are both released and have their deadlines within the interval.

The *supply bound function* $\mathrm{sbf}^{Q}_{\Omega}$ of a tree schedule Ω lower bounds the amount of network provided (resource) to the messages in the queue Q. For a time interval length t, $\mathrm{sbf}^{Q}_{\Omega}(t)$ gives the smallest network supply in any time interval of length t. In the remainder of this section, we assume that the queue Q is apparent from the context, and represent the sbf as $\mathrm{sbf}_{\Omega}(t)$.

We also define the *service time* of a resource supply as the duration that it takes for the supply to provide the resource. Specifically, the service time function $\mathrm{tbf}^{Q}_{\Omega}(t)$ returns the time it takes for the supply Ω to provide t units of resource for the workload W that shares the queue Q.

In the remainder of this section, we consider that the tasks specified are periodic, without jitter. It is to be noted that the framework of tree schedules (which describes the resource supply) itself can be used in conjunction with other task models as well. For analyzing schedulability with other task models, the resource requirement (in

terms of demand bound function) would have to be computed, and checked against the supply provided by the tree schedules to ensure schedulability.

For a task set W consisting of periodic tasks $\tau_i \equiv (p_i, e_i, d_i)$, where p_i is the periodicity of the task and e_i and d_i are the network requirement and deadline of the message generated by τ_i, respectively,

$$\text{dbf}_{EDF}(W,t) = \sum_{\tau_i \in W} \left(\left\lfloor \frac{t-d_i}{p_i} \right\rfloor + 1 \right) \cdot e_i \tag{1}$$

This dbf has been proposed by Baruah et al. (1990) in the context of computational resources.

For a periodic task set W under RM scheduling, Lehoczky et al. (1987) proposed a $\text{dbf}_{RM}(W,t,i)$ as,

$$\text{dbf}_{RM}(W,t,i) = e_i + \sum_{\tau_k \in HP_W(i)} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \tag{2}$$

where $HP_W(i)$ is the set of higher priority tasks than τ_i in W. For a task τ_i over a resource supply model Ω, the worst case response time $r_i(\Omega)$ of τ_i can be computed as given in Shin and Lee (2004) and based on Joseph and Pandya (1986) for periodic resource models:

$$r_i(\Omega) = \min\{t\} \quad \text{s.t.} \quad \text{dbf}_{RM}(W,t,i) \le \text{sbf}_{\Omega}(t) \tag{3}$$

For isochronous TS, we present the following scheme to compute its sbf. The procedure is similar to that used by Baruah (1998) to compute the dbf of a recurring branching tasks (RBT) which is a task model with branches.

To compute the sbf_{Ω}^{Q} of a tree schedule Ω, we need to compute the lower bound on the amount of network provided (resource) to the messages in the queue Q. This is computed for two cases based on the period of recurrence P. The first case considers the supply provided in any interval t of duration $< 2P$ for that queue, and the second case calculates the minimum supply for that queue amongst all intervals t ($\geq 2P$). In the first case, the supply in any interval t is computed by enumerating all possible paths of the tree schedule of duration t, and noting the minimum possible supply in these paths. To compute the worst case supply in the second case, we break down the supply in interval $t > 2P$ into multiple intervals of duration P, which provide minimum possible supply in duration P, and the remainder interval, which is going to be of duration $< 2P$. The idea is that, by breaking it down in this fashion, we can compute the demand for any interval $t > 2P$ based on the demand calculated in the first step. This technique is elaborated below.

1. *Case* $t < 2P$: Consider the TS Ω with a period P and a time interval of length $t < 2P$ where P is the period of Ω. In this case, we enumerate all the paths and build a table tabulating the different t and corresponding minimum supply, $\text{sbf}_{\Omega}(t)$ for every queue. Observe that in this case, the initial location v^0 occurs at most once. There are two subcases. The first subcase is a path where v^0 does not occur at all, and the second subcase where v^0 occurs exactly once. If v^0 does not

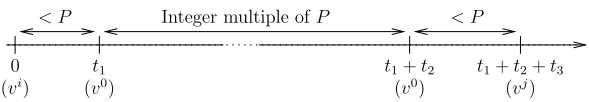


Fig. 3 Critical run for $t \geq 2P$

appear in a path, then, the path originates at a internal location v_a, and terminates at another internal location v_b. The number of distinct runs in this subcase is, therefore, $\mathcal{O}(|V|^2)$ corresponding to the number of different combinations of v_a and v_b. Now consider the second subcase. In case v^0 appears exactly once, we can describe the path (in general) to consist of two components: a path starting from some location v_a leading to v^0 via a leaf v_b and a path from v^0 to some location v_c. The number of distinct runs in this subcase is, therefore, $\mathcal{O}(|V|^3)$ corresponding to the number of different combinations of v_a, v_b and v_c. Based on this, we can state that the enumeration procedure in either subcase is polynomial in $|V|$.[1]

2. *Case* $t \geq 2P$: For $t \geq 2P$, we first define the worst case supply path for a queue Q given a tree schedule Ω as a path from v^0 back to v^0 with the minimum network supply (for a queue Q). We denote the supply along this path by ws^Q.

   In general, a path with minimum supply in Ω for queue Q and duration t ($\geq 2P$), can be seen to consist of the three sub-paths. The first part of the run consists of a path from some internal location v^i to the initial location v^0 through some leaf location. The second part of the run consists of some (non-zero) paths from location v^0 back to v^0. The third part of the run consists of a path from v^0 to some internal location v^j. The three sub-paths of path of duration t are illustrated in Fig. 3. Formally, a path(v^i, v^j) of duration t ($\geq 2P$) can be shown to consist of three sub-paths:

   – a path(v^i, v^0) of duration t_1 s.t. $t_1 < P$,
   – a path(v^0, v^0) of duration t_2 s.t. $t_2 = kP, \exists k \in \mathbb{N}$,
   – a path(v^0, v^j) of duration t_3 s.t. $t_3 < P$, and
   – $t_1 + t_2 + t_3 = t$.

   A few observations:

   (a) Given a path with minimum supply for a queue Q of duration $t \geq 2P$, there are following possibilities. (a) The path starts at v^0 (i.e., $v^i = v^0$, $t_1 = 0$), or (b) ends at v^0 (i.e., $v^j = v^0$, $t_3 = 0$), or (c), begins and ends at v^0, or (d) comprises of all the three sub-paths, i.e., begins and ends at a location different from v^0.
   (b) The minimum supply for a queue Q during any path that begins at v^0 and ends at v^0 has to be equal to ws^Q. Therefore, the minimum supply in the second part of the run is simply $k \cdot ws^Q$.
   (c) Consider the first and the last sub-paths. Since path(v^i, v^0) of duration t_1 ends in v^0 and path(v^0, v^j) of duration t_3 starts from v^0, for the purposes of sbf computation, we can concatenate them into a single path of duration $t_1 + t_3$, as the supply of the new path is the same as that of the two paths individually.

[1] However, it must be noted that the enumeration complexity is actually pseudopolynomial as number of vertices $|V|$ itself is exponential in terms of the input.

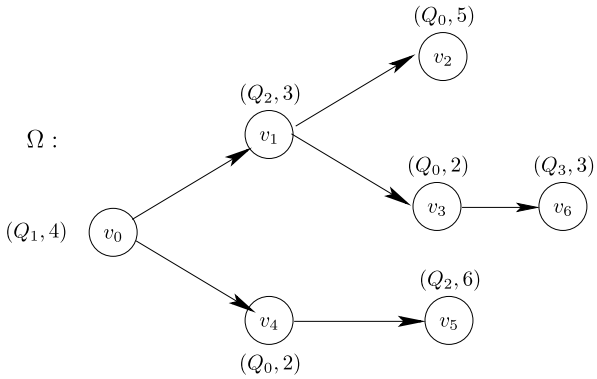


Fig. 4 Example tree schedule for computing the sbf

Based on the above observations, we proceed to compute the minimum supply in Ω for queue Q and duration t ($\geq 2P$) as follows. We have to consider all the four possibilities highlighted in the first observation. In case such a path begins and ends at v^0, then t is a multiple of P, and the minimum supply for a queue Q is simply $\lfloor \frac{t}{P} \rfloor ws^Q$. If the path either begins at v^0, or ends at v^0 (but not both), then again, the minimum resource supply for queue Q for the phase from v^0 back to v^0 is $\lfloor \frac{t}{P} \rfloor ws^Q$. The remainder of the run has a duration of less than P, and its supply can be looked up the enumerated table from the first case as $\text{sbf}_\Omega(t - \lfloor \frac{t}{P} \rfloor P)$.

If the path of minimum duration neither begins, nor ends, at v^0, then there are two cases to consider: $t_1 + t_3 < P$, and $P \leq t_1 + t_3 < 2P$. If the first and third sub-paths combined have a duration less than P, the minimum supply can be looked up the enumerated table from the first case ($t < 2P$) as $\text{sbf}_\Omega(t - \lfloor \frac{t}{P} \rfloor P)$. The minimum supply contribution of the sub-path from v^0 to v^0 would be $\lfloor \frac{t}{P} \rfloor ws^Q$. Otherwise, if the duration of the first and third sub-paths combined is at least P and less than $2P$, then, the supply would be $\text{sbf}_\Omega(t - \lfloor \frac{t}{P} \rfloor P + P)$, and the contribution of the middle portion would be $(\lfloor \frac{t}{P} \rfloor - 1) ws^Q$.

Combining all these different possibilities, we can state the $\text{sbf}_\Omega(t)$ for a queue Q and $t \geq 2P$ compactly as,

$$\min\left\{\left\lfloor \frac{t}{P} \right\rfloor ws^Q + \text{sbf}_\Omega\left(t - \left\lfloor \frac{t}{P} \right\rfloor P\right), \left(\left\lfloor \frac{t}{P} \right\rfloor - 1\right) ws^Q + \text{sbf}_\Omega\left(t - \left\lfloor \frac{t}{P} \right\rfloor P + P\right)\right\} \tag{4}$$

Example 1 Consider the tree schedule Ω as shown in Fig. 4. There are 4 queues in the system, labeled Q_0, Q_1, Q_2 and Q_3. The tuple (Q, n) indicates the queue being serviced (network supply) for n units of time. The tree schedule presented is isochronous with a period of recurrence 12.

sbf_Ω is tabulated for Q_0 and Q_1 in the Fig. 5.

To elaborate on the procedure outlined above, consider queue Q_0. Observe that Q_0 receives a supply in locations v_2 and v_3, and v_4. In any path from v_0 to v_0, the least amount of supply that queue Q_0 receives is 2. One such path is $v_0 \rightarrow v_4 \rightarrow v_5 \rightarrow v_0$.

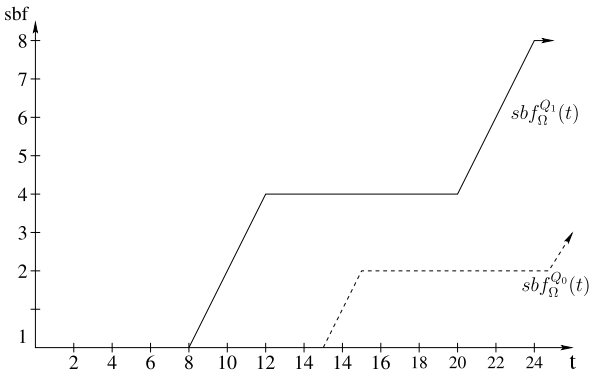


Fig. 5 The sbf values for $t \leq 2P$ for TS in Fig. 4

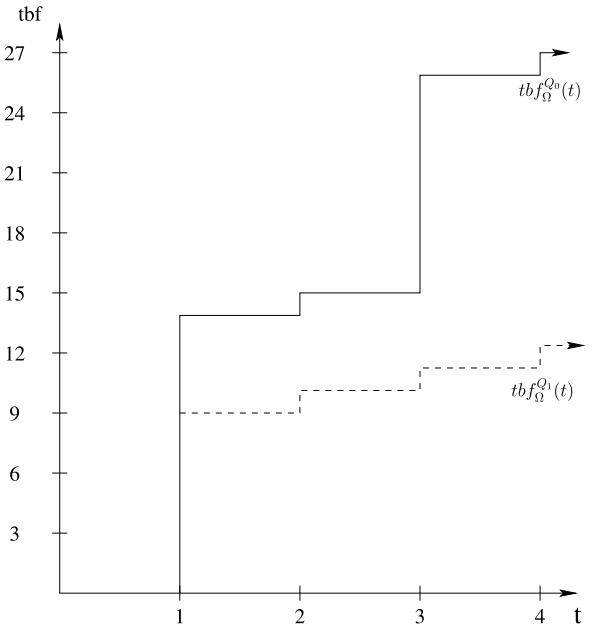


Fig. 6 The tbf values for TS in Fig. 4

Therefore, we have $ws^{Q_0} = 2$. Similarly, it can be seen that and $ws^{Q_1} = 4$, on any path from v_0 back to v_0. For any path of duration < 10 that visits v_0 at most once, the worst case supply for Q_0 is 0. One such path is $v_5 \rightarrow v_0$. Therefore $\text{sbf}_{\Omega}^{Q_0}(t)$ for $t \leq 10$ is 0. From the path $v_5 \rightarrow v_0 \rightarrow v_1 \rightarrow v_3$, we can also see that $\text{sbf}_{\Omega}^{Q_0}(14) = 1$. Any other path, for instance, one involving location v_2 will involve a greater supply, and is therefore, ignored in the computation. After we have computed sbf values in Fig. 5, we can easily compute the tbf values. For example, the first (minimum) duration where queue Q_0 is guaranteed of a supply of 1 unit is 14, a supply of 2 units is 15, a supply of 3 units is 26, and so on. The numbers for other queues can be computed similarly. The computed tbf values are shown in Fig. 6.

The sbf computation procedure for isochronous models cannot be directly applied to anisochronous tree schedule models. This is because, the minimum duration between successive invocations of the root location v_0 in the anisochronous case, depends on the particular run. In fact, a reduction from the integer knapsack problem can be used to prove that sbf computation for this case is NP-hard. It is possible to get around this limitation, by resorting to approximations to lower bound the supply. One possibility is to use approximations to the integer knapsack problem to compute the approximate minimum supply. The other possibility is to convert the anisochronous tree schedule into an isochronous tree schedule, such that the minimum supply is preserved (i.e., it does not increase) across the transformation. A similar approach has been proposed for computing the demand of anisochronous tasks (Anand et al. 2008), and we leave the possibility of pursuing this approach as future work.

We are now in a position to state the schedulability conditions under two scheduling algorithms—EDF and RM. EDF is a dynamic scheduling algorithm where the scheduler schedules tasks that are closest to their deadline with priority, i.e., the task with earliest deadline is scheduled for execution over tasks that have later deadlines. RM scheduling is a static priority scheduling algorithm, where task priorities are assigned based on the period of execution of the job: the shorter the task period, the higher is the task priority. The results we present below for EDF and RM are similar to Theorem 1 and 2 (Shin and Lee 2003) stated in the context of periodic resource models.

Theorem 1 *Consider a workload W consisting of periodic tasks $\tau_i \equiv (p_i, e_i, d_i)$, $(d_i \leq p_i)$ that generate messages m_i with resource (network) requirement e_i and a deadline d_i every p_i units of time. Let us also denote by rl_i the release time of task τ_i. W is schedulable under EDF, if and only if,*

$$\forall 0 < t < 2LCM_W + rl_{max}, \quad \text{dbf}_{EDF}(W, t) \leq \text{sbf}_{\Omega}(t) \tag{5}$$

where LCM_W is the least common multiple of all p_i, and $rl_{max} = \max_{\tau_i \in W} rl_i$.

Proof If sbf_{Ω} is less than dbf_W, then the required resources are insufficient, and the workload W is not schedulable. This implies the necessity. To show sufficiency, we prove that if all tasks in W are not schedulable by EDF, then Eq. (5) does not hold. Let t_2 be the first instant at which a message generated by some task τ_i in W misses its deadline. Let t_1 be the latest instant at which the resource supplied was idle or was supplied to a message whose deadline is after t_2. Therefore, for the time interval $t' = t_2 - t_1$, the total demand is greater than the supply provided by Ω, which implies $\text{dbf}_{EDF}(W, t') > \text{sbf}_{\Omega}(t')$. The condition should be tested for all intervals $0 < t < 2LCM_W + rl_{max}$. This is because each of the periodic tasks could be asynchronously released, and for the asynchronous tasks, it has been shown by Leung and Merrill (1980) that the task set is feasible if and only if all deadlines in the interval $0 < t < 2LCM_W + rl_{max}$ are met. □

Theorem 2 *Consider a workload W consisting of periodic tasks $\tau_i \equiv (p_i, e_i, d_i)$, $(d_i = p_i)$ that generate messages m_i with resource (network) requirement e_i and a*

deadline d_i. *W is schedulable under RM, if and only if,*

$$\forall \tau_i \in W, \exists 0 < t \leq p_i, \quad \text{dbf}_{RM}(W, t, i) \leq \text{sbf}_{\Omega}(t) \tag{6}$$

Proof A workload W is schedulable with Ω if and only if the maximum service duration for all the messages of W is no greater than their relative deadlines. For a task $\tau_i \in W$, the maximum response time of a workload occurs when it experiences the worst-case interference I_i from other higher priority tasks that share the same queue. The maximum service duration of Ω for I_i is given by $\text{tbf}(I_i)$, which is the maximum response time $r_i(\Omega)$ of τ_i. Therefore, a necessary and sufficient condition for τ_i to meet its deadline with supply Ω is $r_i(\Omega) \leq p_i$. We get the desired result from Equation 3 and the observation that all messages generated by tasks in the workload is schedulable with Ω if and only if each of them is individually schedulable with Ω. □

Discussion Although we consider a conditional resource supply, we have not specifically considered the conditions in the branching in our analysis above. By ignoring the conditions, we were able to derive a schedulability condition under both dynamic, and fixed priority scheduling. Viewed in the context of conditional task and supply models, these results may be too conservative and pessimistic. Exact schedulability analysis, which models the conditions explicitly, can be performed by timed model checking. We refer the reader to some approaches involving timed model checking (Fersman et al. 2002; Fischmeister et al. 2007) which have performed such analyses. We plan on expanding our results using some of the methods used in literature.

In this work, we have focused exclusively on message schedulability, i.e., the schedulability of the system with respect to network resource. This analysis makes the assumption that computational resources to enqueue/dequeue messages from the queues, and other execution requirements of the tasks that generate messages are met. Although separating computational and communication requirements keeps the analysis simple, to be completely safe, what we need to do is to analyze both computational and communication requirements in conjunction. This type of schedulability analysis is called *holistic*, which has been extensively explored for systems with stateless supply (see e.g., Tindell and Clark 1994). We note that model checking techniques discussed above can be used to perform such holistic schedulability analysis for tree schedules. The disadvantage of the model checking approach is the lack of closed form schedulability conditions, which may be desirable for quick checking. We leave the prospect of extending the analysis to consider computational requirement of tasks as future work.

4 Analysis with probabilistic choices: average waiting time

If path probabilities (p_r as mentioned in Sect. 2) are known, then we can analyze how long a message waits in a queue before it is transmitted. This is an important metric both from the perspectives of quality of service and estimating the error, for instance,

in control applications. The average waiting time specifies how long a specified message has to wait from the instant it arrives, to the instant it is serviced (i.e., when it gets a communication slot in the network).

We start with the assumption that we are given an application that generates different events. These events, in turn, generate messages which need to be sent on the network. We assume that there are two types of events: independent and dependent ones. For instance, in a fault tolerant application, there could be a provision to transmit a message again in case the original message could not be sent. The original event constitutes an independent event whereas the transmission of the backup message is a dependent event. We assume that independent events arrive according to a known distribution. We also assume that the conditional probabilities of a dependent event given the events it depends on are known. For instance, such a model can be presented in the form of a dependency graph. Such a dependency graph can be generated from the knowledge of dependencies and probabilities of different events.

In addition to the message arrival at network nodes, we assume a Tree Schedule Ω, which specifies the CPU schedule where the application runs. We also assume that there is a known mapping from messages to slots in the communication schedule and the probability of taking a transition in the schedule. Further, in our calculations, we assume that each of the reducible transitions is equally likely to be taken.

We now introduce some additional terminology for this section.

Terminology. When messages arrive, we assume that they will be enqueued in the queue q_{in}. The message at the head of the queue q_{in} on a network node will be serviced in the slot assigned to the node holding the message. We refer to the time spent in the queue until the message reaches the head of the queue as the *waiting time* (T_W). The *effective service time* (T_{ES}) is the total time spent at the head of the queue q_{in} waiting for the slot plus the time in the slot (T_S). The *effective waiting time* (T_{EW}) can be defined as $T_W + T_{ES} - T_S$. For the analysis here, we do not consider the waiting time in the queue q_{out} of the receiving node, as we are more interested in the queueing delay related to the communication schedule.

In our analysis, we consider event-triggered systems, in which messages are generated as a result of events. For the analysis, the system is assumed to consist of a network of queues and we consider the service and waiting times once the system has reached a steady state. We note that the waiting time may be different for the first few messages, but not so in a stabilized system. For event-triggered systems, independent events are considered to arrive as a Poisson process, i.e., the number of arrivals $N(\mathcal{E}, t)$ of an event $\mathcal{E}$ in a finite interval t is given by $P\{N(\mathcal{E}, t) = m\} = \frac{(\lambda t)^m}{m!} e^{-\lambda t}$. We choose the Poisson process as it represents the sequence of events that are randomly spaced in time. We defer the proposition of considering more complicated models (e.g., Lehoczky 1996) as future work. Further, to simplify the analysis, we consider infinite buffers.

Waiting time analysis Before we present the distribution of effective service times given a tree schedule, we introduce some terms and functions to be used in that result.

Let us assume that messages are generated corresponding to an event $\mathcal{E}_i$ that arrives according to a Poisson distribution, and that they are serviced at multiple locations $V_i = \{v_i^1, \ldots, v_i^m\}$ as specified per a Tree Schedule Ω. For example, given the

TS in Fig. 4, if we assume an event of interest gets serviced by queue Q_0, then, $V_i = \{v_2, v_3, v_4\}$. As in Sect. 2.2, $p_r(v^i, v^j)$ of path(v^i, v^j) is the probability of reaching v^j starting from v^i. For every location $v_i^j \in V_i$, we define set $F(v_i^j)$ as the set of leaf locations that are reachable from v_i^j. In the TS specified in Fig. 4, for an event that is serviced by queue Q_0, $F(v_1) = \{v_2, v_6\}$ and $F(v_4) = \{v_5\}$. As there could be multiple locations in V_i servicing a particular event, we define a contribution of every location $v_i^j \in V_i$ towards the service time of event $\mathcal{E}_i$ by a function $w(v_i^j) = p_r(v^0, v_i^j) \cdot \sum_{v \in F(v_i^j)} p_r(v_i^j, v) \frac{1}{m(v)}$, where $m(v)$ is the number of service locations of $\mathcal{E}_i$ from v^0 up to, and including, v. Informally, the weight function $w(v_i^j)$ computes how likely a particular event is serviced by location v_i^j considering the probability of reaching v_i^j. We denote by $S(v_i^j) \subset V_i$, the set of descendant schedule locations servicing event $\mathcal{E}_i$ that are reachable from location v_i^j. Lastly, we denote by $NS_i \subset V^F$, the set of leaf locations such that the path from v^0 to any location in NS_i contains no service location for event $\mathcal{E}_i$. For example, given the TS in Fig. 4 and an event that is serviced by queue Q_3, then, $NS_i = \{v_2, v_5\}$.

The following theorem gives a bound on the average waiting time to service all these messages.

Theorem 3 *Given a TS Ω, if the messages corresponding to an arrival event $\mathcal{E}_i$ are serviced at schedule locations $V_i = \{v_i^1, \ldots, v_i^m\}$ as specified in Ω, then the moment* (*i.e., distribution of a random variable.*) *s of the effective service times is given by,*

$$T_{1_n}(v_i^j) = \sum_{v_i^k \in S(v_i^j)} p_r(v_i^j, v_i^k) \cdot \text{dur}(v_i^j, v_i^k)^n \tag{7}$$

$$T_{2_n}(v_i^j) = \sum_{v_i^k \in F(v_i^j)} p_r(v_i^j, v_i^k) \cdot \left(\text{dur}(v_i^j, v^0) + \mu_i^0\right)^n \tag{8}$$

$$s_n(\mathcal{E}_i) = \sum_{v_i^j \in V_i} w(v_i^j)\left(T_{1_n}(v_i^j) + T_{2_n}(v_i^j)\right) \tag{9}$$

$\mu_i^0 = \frac{\sum_{v_i^k \in S(v^0)} p_r(v^0, v_i^k) \cdot \text{dur}(v^0, v_i^k) + \sum_{v \in NS_i} p_r(v^0, v) \cdot \text{dur}(v^0, v)}{1 - \sum_{v \in NS_i} p_r(v^0, v)}$ *where n is the order of the moment, and the terms are as defined above.*

The mean and variance of effective service times (T_{ES}) for an event $\mathcal{E}_i$ are then given by $s_1(\mathcal{E}_i)$ and $s_2(\mathcal{E}_i) - (s_1(\mathcal{E}_i))^2$, respectively.

Proof Let the messages corresponding to an arrival event $\mathcal{E}_i$ get serviced in slots at locations $V_i = \{v_i^1, \ldots, v_i^m\}$ in the TS Ω. Consider one such location, say v_i^j. The effective service times experienced starting from this location depends on the next slot available for $\mathcal{E}_i$ starting from v_i^j. Now, it is possible that one of the descendants of v_i^j in Ω, say v_i^k will be one servicing $\mathcal{E}_i$. If this is the case, then the effective service time will be dur(v_i^j, v_i^k). When this path involves reducible transitions (there

are multiple paths to v_i^k from v_i^j), we consider dur to be the average time taken over all the reducible transitions. Therefore, the nth moment of effective service time in this case is, $\sum_{v_i^k \in S(v_i^j)} p_r(v_i^j, v_i^k) \cdot \text{dur}(v_i^j, v_i^k)^n$.

If there is no direct descendant slot that services $\mathcal{E}_i$, then the schedule Ω will consist of a return to the root location from some leaf location that is a descendant of v_i^j and the service slot would be reached in the next cycle at any of the locations servicing $\mathcal{E}_i$ from v^0, say v_i^k. The probability of reaching v_i^k is $p_r(v^0, v^k)$ and the service time here is simply $\text{dur}(v^0, v^k)$. On the other hand, the probability that no service slot would be reached is $(1 - \sum_{v_i^j \in S(v^0)} p_r(v^0, v_i^j))$ where $S(v^0) \subset V_i$ is the set of next schedule locations servicing event $\mathcal{E}$ (from location v^0). The average effective service time starting from v^0 can be derived as, $\mu_i^0 = \sum_{v_i^k \in S(v^0)} p_r(v^0, v_i^j) \cdot \text{dur}(v^0, v_i^k) + \sum_{v \in NS_i} p_r(v^0, v)(\text{dur}(v^0, v) + \mu_i^0)$. The average cycle duration for schedules without any slot for $\mathcal{E}_i$ is then obtained by solving the above equation to get, $\mu_i^0 = \frac{\sum_{v_i^k \in S(v^0)} p_r(v^0, v_i^k) \cdot \text{dur}(v^0, v_i^k) + \sum_{v \in NS_i} p_r(v^0, v) \cdot \text{dur}(v^0, v)}{1 - \sum_{v \in NS_i} p_r(v^0, v)}$. Therefore, $\sum_{v_i^k \in F(v_i^j)} p_r(v_i^j, v_i^k) \cdot (\text{dur}(v_i^j, v^0) + \mu_i^0)^n$ is the nth moment of effective service time in this case, where the initial location v^0 is reached through v_i^k.

From the above observations, we get that, the nth moment of the total effective service time $s_n(\mathcal{E}_i) = \sum_{v_i^j \in V_i} w(v_i^j) \cdot (T1_n(v_i^j) + T2_n(v_i^j))$, where $w(v_i^j)$ is the weight function associating the contribution of location v_i^j towards the servicing time of an event such that $\sum_{v_i^j} w(v_i^j) = 1$ along a path. □

Returning to the analysis of waiting times, the arrival of independent events is Poisson (and hence Markovian), the service times are general distribution function, we can model them as the queueing model $M/G/1$ for the independent events. Therefore, the variance in arrival times is $\sigma_T^2(\mathcal{E}_i) = \frac{1}{\lambda_i}$. Now consider dependent events. Let us say that the dependent event $\mathcal{E}_j$ depends on an independent event $\mathcal{E}_i$. Since the arrival process of $\mathcal{E}_i$ is Poisson, its inter-arrival times are exponentially distributed. The inter-arrival times (T_I) of dependent event $\mathcal{E}_j$ is given by the function, $P(T_I = t) = p_{ij} \cdot \lambda_i v^{-\lambda_i t}$. We can consider this as a $G/D/1$ queueing scheme. For a $G/G/1$, we have the following inequality on the waiting time, $W_q \leq \lambda \frac{\sigma_X^2 + \sigma_T^2}{2(1 - \lambda \mu_X)}$ (Bose 2001), where λ is the average rate of arrival, μ_X, σ_X are the mean and variance of the service times, and σ_T is the variance of the arrival time. For a dependent event, the mean arrival time is $\sigma_T^2(\mathcal{E}_j) = \frac{1}{p_{ij}\lambda_i}$.

Definition 3 Given an event-triggered system comprising of events $\mathcal{E}_i, i = 1, \ldots, n$ such that the independent events arrive as a Poisson process, and these events are serviced by a TS Ω, the *average waiting time* (T_W) of an event $\mathcal{E}_i$ satisfies,

$$T_W(\mathcal{E}_i) \leq \frac{\lambda_i^2 s_2(\mathcal{E}_i)^2 + 1}{2\lambda_i(1 - \lambda_i s_1(\mathcal{E}_i))} \tag{10}$$

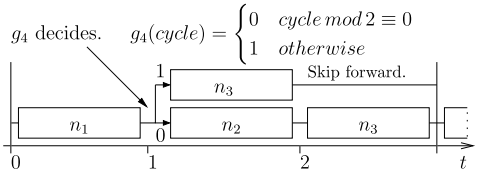


Fig. 7 A tree schedule with one on-the-fly choice

where $\mathcal{E}_i$ is an independent event with arrival rate of λ_i, and

$$T_W(\mathcal{E}_j) \leq \frac{p_j^2 \lambda_i^2 s_2(\mathcal{E}_i)^2 + 1}{2p_j\lambda_i(1 - p_j\lambda_i s_1(\mathcal{E}_i))} \tag{11}$$

where $\mathcal{E}_j$ is a dependent event with interarrival times distributed as $p_j \cdot \lambda_k v^{-\lambda_k t}$ for some k. $s_1(\mathcal{E})$ and $s_2(\mathcal{E})$ are the average and variance of the effective service times as given in Theorem 3. The average waiting time for the whole system can be defined as, $\overline{T_W}(\mathcal{E}) = \frac{1}{|\mathcal{E}|}\sum_{\mathcal{E}_i \in \mathcal{E}} T'_W(\mathcal{E}_i)$ where $T_W{}'$ represents the upper bounds of effective waiting times.

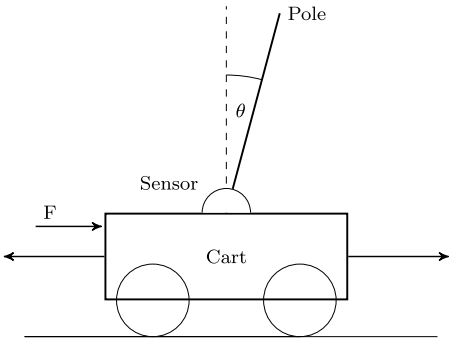
Example 2 Consider the schedule in Fig. 7 with one on-the-fly choice at time $t = 1$. Let the probability of guard transition to n_3 be $\frac{3}{4}$ and n_2 be $\frac{1}{4}$. Assume that the slot for n_3 in the upper path is single slot in duration. Let us assume that the arrivals at n_1 is Poisson with rate λ_1. Then, the variance in arrival time for n_1 is $\frac{1}{\lambda_1^2}$. The mean service time for n_1 is $\frac{9}{4}$, and the variance is $(\frac{3}{4})2^2 + (\frac{1}{4})3^2 - (\frac{9}{4})^2 = \frac{3}{16}$. Let us analyze the waiting time for a dependent event n_3. Its mean arrival rate is $\frac{1}{\lambda}$, the mean service time would have to be calculated for positions of n_3 (on the path above and the path below). In both these cases, this is given by $(\frac{3}{4})1 + (\frac{1}{4})2$. Therefore the mean time is $\frac{5}{4}$. Its variance is then $(\frac{3}{4})1 + (\frac{1}{4})4 - (\frac{5}{4})^2 = \frac{3}{16}$. For n_2, the mean arrival time is $\frac{1}{4\lambda_i}$ and the mean service time is $\sum_{n\geq 0}(\frac{3}{4})^n \frac{1}{4}(2 + 2 \cdot n) = 8$. The average waiting time for the entire schedule can now be calculated.

5 Case study 1: inverted pendulum control

The aim of the case study is to evaluate the idea of tree schedules. For this, we implement a control system for an inverted pendulum, which tolerates one independent, transient value fault using triple modular redundancy (Kopetz 1997). In the case study we show benefits of state-based communication schedules as expressed in tree schedules and analyse them using the presented framework.

An inverted pendulum is essentially a pole mounted on a cart. The pole is free to rotate round on an axis, and the cart can move horizontally. The objective is to maintain the inverted pendulum in the upright position (see Fig. 8). The control parameter is the linear acceleration F of the pivot. This parameter bases on the pendulum's angle θ to the car and its angular speed ω. We assume the linear acceleration F to saturate around a maximum, meaning that the motor only has finite strength specified by a saturation point.

Fig. 8 Inverted pendulum



In our system, we want the system to tolerate one fault of the reading of variable θ. We use standard triple modular redundancy to mask this fault. In our particular case, we use three independent units reading θ's value to vote. These units are labeled u_1 to u_3. All units share a broadcast network which they use to communicate their readings to a controller following a communication schedule. The control unit receives all readings and computes a new value for F. To mask a single measuring fault, the controller receives three readings and uses a majority vote to identify the correct value. A majority of at least two votes for one reading value will decide the voting. Note that we assume faults will only occur when reading the variables. We exclude faults in the controller or the system executing the tree schedules.

5.1 The simulation model

Figure 9 shows the Simulink model for the inverted pendulum using a PID controller (the inverted pendulum block was taken from CMU (last visited 12/2007)). We model the packetized broadcast network shared by the controller and the measurement units by an enabled, triggered subsystem. A pulse generator (labeled "Network slotting") specifies the slot length by triggering the subsystem and the communication schedule enables communication of a new value θ to the controller (representing a successful voting). For example, in the standard TDMA system a successful voting happens every third slot: Given a step size of 0.02 s, a new value of θ is available every 0.06 s ($= 3 \cdot 0.02$ s), In the implementation, we ignore overhead introduced by clock synchronization or computation time for reading values, adjusting values, and the voting. This can be incorporated as additional overhead to the cycle duration.

The blocks between the inverted pendulum and the PID controller (i.e., PACKETSIM and memory) simulate a packetized network and tree schedules. The network schedules are implemented in a state machine inside the block called "Tree Schedule". This block is triggered by the pulse generator in the block "Network slotting" that specifies the slot length. The output of the tree schedule block enables or disables the network (i.e., PACKETSIM and memory) and thus controls whether the PID controller receives a fresh value of θ. Additionally, the Tree Schedule block receives an randomized input parameter to simulate error.

This error rate is used to determine the likelihood of a decisive vote after receiving $\gamma + x$ slots with γ as the minimum number of required readings and x as the

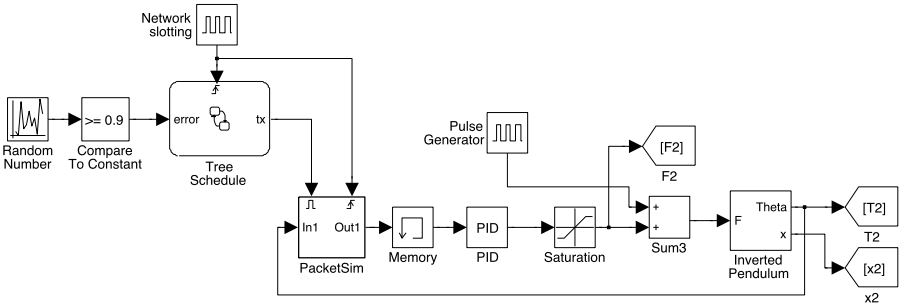


Fig. 9 PID control with pendulum and Tree Schedules

extra readings. For example: after receiving three readings, the voting is always decisive given that at most one error occurs. We use this in our case study to determine the likelihood that a voting is already decisive after receiving two readings (i.e., the first reading equals the second reading). Thus, when a system tolerates n faults, the probability $P(x)$ of requiring $\gamma + x$ slots is:

$$P(x) = \binom{\gamma}{x} p^x (1-p)^{(\gamma - x)} \tag{12}$$

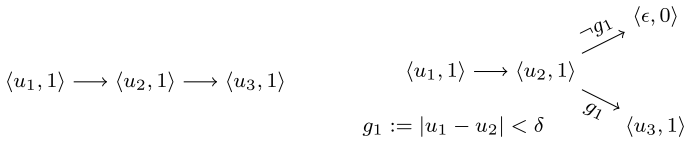
with $\gamma = \lceil \frac{2n+1}{2} \rceil$ defining the minimal number of slots, p defining the probability of any network node failing, and $1 \leq x \leq \lfloor \frac{2n+1}{2} \rfloor$ defining the number of additionally required slots.

If $x = 1$, then we need one additional slot in our communication schedule; therefore, one of γ slots contains a different value than the others, which happens when a fault occurred. For $\gamma = 3$ this is: $(1-p)(1-p)p + (1-p)p(1-p) + p(1-p)(1-p) = 3p(1-p)^2$. For an arbitrary γ, this is: $\gamma \cdot p \cdot (1-p)^{(\gamma-1)}$. Now for an arbitrary x, the probability is the sum of $p^x(1-p)^{(\gamma-x)}$ for all possible subsets of length x. Equation (12) follows.

For the case study, we consider a slot length of 0.02 time units and we use a simulation fixed-step size of 10^{-4}. The physical values are taken from CMU (last visited 12/2007): The mass of the cart is 0.5 kg, the mass of the pendulum is 0.2 kg, the friction of the cart is 0.1 N/m, the length to the pendulum center of mass is 0.3 m, the inertia of the pendulum is 0.006 kg, and gravity is approximated by 9.8 m/s^2. The PID controller has a setting of $K_p = 100$, $K_i = 1$, and $K_d = 20$. Finally, we assume an error rate of $p = 0.005$, which means that one out of two hundred measurements shows an error. Although, this results in a mean time to fault which is several magnitudes higher than empirical results (Kopetz 2004), a lower fault rate only further improves our results.

5.2 Case study schedules

Figure 10 shows the schedules for the two scenarios. The schedule in Fig. 10(a) shows the standard table-based TDMA schedule in which all three values are communicated each round before a decision is made and a new value is supplied to the PID controller.



(a) Schedule using table TDMA. (b) Schedule using Tree Schedules.

Fig. 10 Schedules for the case study

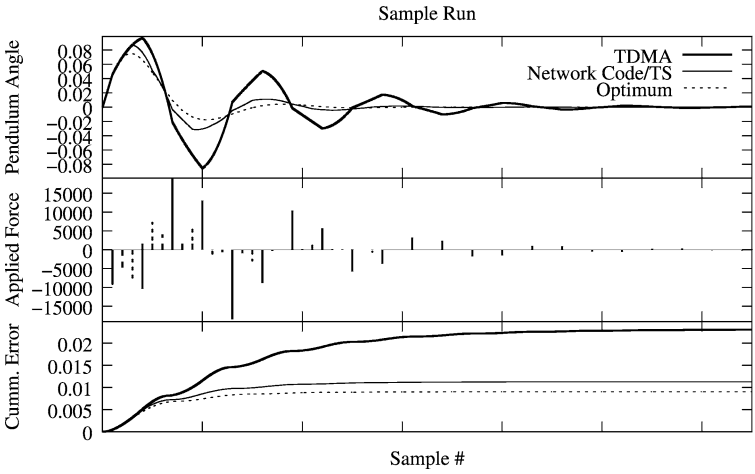


Fig. 11 Simulation sample run with a force saturation of 19 000 Newton

Figure 10(b) shows the tree schedules for the same scenario; however, in this system the PID controller may get a new value after already two slots if the voting is decisive.

For comparison purposes, we also consider the optimal schedule, which assumes no faults. This schedule consists of only one slot in which one measurement unit reports a new value of θ to the PID controller.

5.3 Hypothesis, experiment, and results

In our case study, we would like to the test the following hypothesis: *Using tree schedules, the required motor strength to stabilize the pendulum is significantly lower than using table-based TDMA.* The intuition is that the tree schedule will provide values more frequently (a ratio of 2:3 for the system using the tree schedule compared to the one using table-based TDMA), and this increase in update frequency allows us to use a weaker motor.

Figure 11 shows an example run of the simulation using a motor force limit of 19 000 Newton. Note that we used aggressive PID settings. In practice, the settings would be tuned to permit lower motor forces. The top part of the chart shows the angle θ of the pendulum, the middle part shows the force applied to the cart, and the bottom part shows the cumulative error with respect to the stable position of $\theta = 0$.

To test the hypothesis, we gradually lowered the force limit of the motor as applied by the PID controller. We started with a force limit of 25 000 which allowed

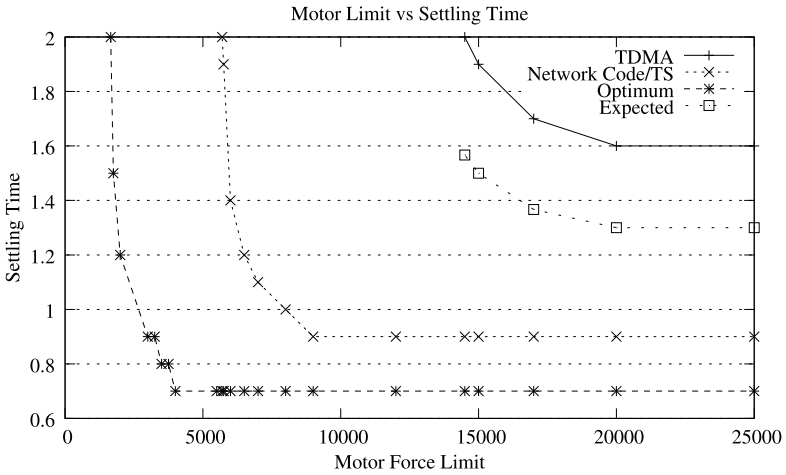


Fig. 12 Simulation results of motor force limits

all systems to stabilize the pendulum, down to the point when no system was able to stabilize any longer. Figure 12 shows the results of this experiment by comparing the settling time of the pendulum with the force limit of the motor.

For the statistical analysis, we use the Kruskal-Wallis Rank Sum Test to compare the results of the algorithms, because our resulting data follows no normal distribution (tested with the Shapiro-Wilk normality test). Since our model had no other disturbances other than the induced faults in the sensor, only the tree schedule data differs between runs. We ran 100 runs with varying random seeds to collect data for the statistical analysis. From these results we can make the following observations:

- *The hypothesis is correct*. Using tree schedules allow us to use a weaker motor without compromising the requirement of tolerating one fault. In addition to the presented scenario, we also checked with different sets of values for pendulum length, cart weight, etc. and the system with the tree schedule consistently outperformed the system using table-based TDMA schedules (with a p of usually less than 0.0003 in the statistical test).
- *The system settles faster than expected*. Our expectation was that tree schedules reduce settling time by about $2/3$: for example, given a force limit of 20 000, TDMA has a settling time of 1.6 seconds, so tree schedules should offer an approximate settling time of 1.3 seconds. However, as can be seen in Fig. 12, the system using tree schedules settles much faster than the estimated value; in this case nearly twice as fast.
- *The operational range increases more than expected*. Similar to the previous observation, using tree schedules exceeded our estimates. Using tree schedules, we can use a motor that is approximately twice as weak and achieve the same performance than the system using table-based TDMA.

6 Case study 2: scheduling of video streaming systems

This case study aims to illustrate the efficacy of tree schedules in practical real-time applications. Specifically, we employ tree schedules to manage the video transmission over a non-reliable network in a video-streaming system. We show that, by capturing the state-information, state-based tree schedules enable better system performance compared to the standard round robin, in terms of server buffer requirement, user waiting time, and user I/O buffer requirement. At the same time, we demonstrate that tree schedules adapt well to bursty traffic conditions and effectively utilize the system's resources while ensuring stable system behavior.

6.1 System architecture

Figure 13 depicts the high-level architecture of the system. As shown in the figure, the MPEG-2 encoder running at the server first encodes the raw video stream from the camera and stores its output in the queue ENC_Q. Note that we assume a constant bit rate encorder (CBR), which is commonly used for real-time video (Whitaker and Benson 2001). The network scheduler reads the encoded bitstream from the queue ENC_Q and transmits to the user across a bursty network. The transmitted video stream, upon arriving at from the network interface, is stored at the input queue DEC_Q while waiting to be decoded by the MPEG-2 decoder running at the user's machine. The decoded video stream will finally be displayed at the output device.

In this system, we assume that the encoder encodes the raw video at a constant bit rate of 15 Mbps. The encoded bitstream consists of three types of frames: intra (I), non-intra predicted (P) and bidirectional (B) frames, where possible patterns of I, B and P are determined by a transition system that determines the implementation of the encoder application as described in Wandeler et al. (2004). Here, the size of an I-frame is taken to be 2 times the size of a P-frame and 6 times the size of a B-frame. The decoder reads data from its input queue DEC_Q at the same speed as the encoder (15 Mbps), after an initial delay of 20 seconds waiting for the initial frames buffering in the queue DEC_Q.

The encoded video stream is transmitted through a non-reliable network that exhibits bursty behavior, which is caused by other background applications that share the same network with the video streaming application (not shown in Fig. 13). We assume that these background applications are network intensive, and they take as much network bandwidth as are allocated by the network scheduler. In this case study, we

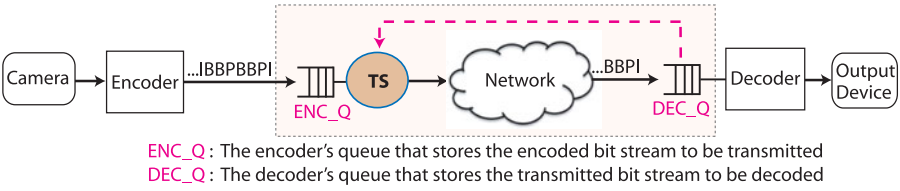


Fig. 13 Architecture of a video streaming system

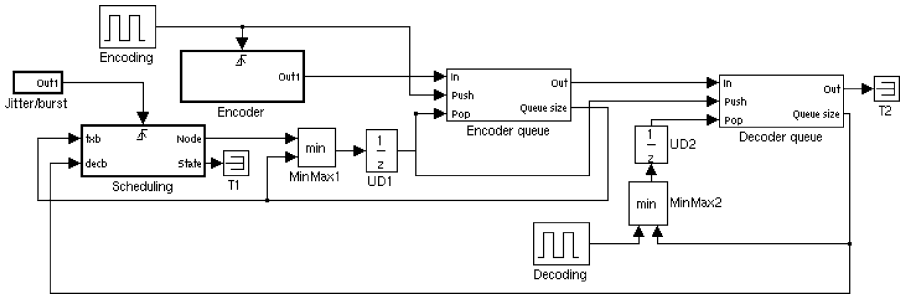


Fig. 14 Simulink model for the video streaming case study

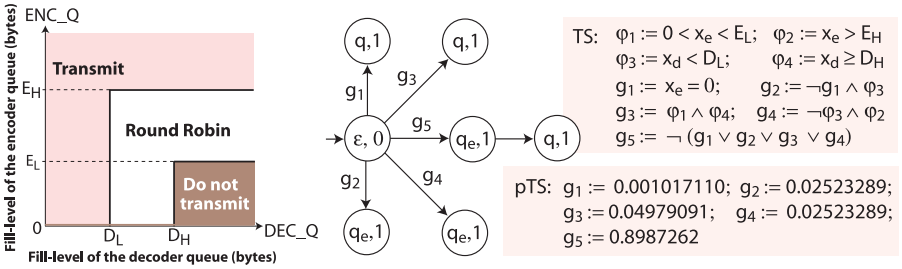
assume that the quality of service for these background applications is not our concern. Note that, if the quality of service of the other applications were a concern, then they would need to be encoded in a tree schedule as well.

The above sharing of network resource might result in a delay in the transmission of the video stream. However, to ensure that the system works fine in the standard round robin scheme, the network bandwidth is assumed to be two times faster than the encoder's bitrate in average. Given the above specification, we are interested in the behavior of the part of the system enclosed in the dashed rectangle box. In particular, we would like to compute the memory requirements for both the encoder and the decoder, as well as the end-to-end delay of the encoded frames (i.e., the instant they are written into the queue ENC_Q till the instant they are read by the decoder). We consider two cases, where the network scheduler follows (i) a state-based tree schedule, and (ii) the standard round robin scheduling policy. Figure 14 shows the Simulink model for the case study.

6.2 Case study tree schedule

In this video streaming system, we assume that the network scheduler has feedback information on the status of the user's input queue. The tree schedule decides whether to transmit the encoded frames based on the current network condition, the fill-level (amount of data) in the encoder queue ENC_Q, and the fill-level of the decoder's queue DEC_Q.

As illustrated in Fig. 15(a), the tree schedule always transmits a frame when (i) the encoder's queue ENC_Q is non-empty and the fill-level of the decoder's queue is below a threshold of D_L bytes, or (ii) the encoder's queue is more than a threshold of E_H bytes. Our aims are to guarantee that there are no underflows at the decoder's queue and no overflows at the encoder's queue. However, when the transmission of the current frame is not urgent (i.e., the decoder's queue has buffered enough to allow for a smooth decoding process and the encoder's queue is not close to being full), it is desirable the allocate the network resource to other background applications. As a result, the tree schedule always delays its transmission of the current frame in ENC_Q if ENC_Q contains less than E_L bytes and DEC_Q contains at least D_H bytes. In all other cases, the scheduler distributes the resource evenly to the video stream and other applications by following the standard round robin policy.



(a) The tree schedule's decision. (b) The tree schedule specification.

Fig. 15 The deterministic (TS) and probabilistic (pTS) tree schedules for the video stream

Figure 15(b) sketches the tree schedule for the encoded video stream for the deterministic (TS) and probabilistic version. In the figure, q_e refers to the encoder's queue (ENC_Q) whereas q refers to the queue of another application that shares the same network with the video application. The tree schedule has two variables x_e and x_d, which denote the current fill-level of the encoder's queue and decoder's queue, respectively. One can verify that the guard associated with each transition in the tree schedule corresponds to the scheduling decision shown in Fig. 15(a). Note that x_d is communicated back to the encoder as part of the background traffic.

For the probabilistic tree schedule shown in Fig. 15(b), we can apply the analysis developed in Sect. 4. The arrival of B, P, and I frames follows the frame pattern generation process described by the FSM given in Wandeler et al. (2004), which can generate three commonly used patterns: IPB, IPBB and IPBBPBB. Here, we assume if there are more than one outgoing transitions from a state of the FSM, the transition that generates an I frame is taken one third of the time, whereas the transitions that generate B and P frames are taken with equal probability. Based on this generation process, the probability of arrival is based to be 0.335 (B), 0.278 (P), and 0.385 (I), respectively. The mean service time for the encoded data can be calculated as per Theorem 3 as $2 \cdot 0.898 + 0.025 + 0.025 = 1.847$ slots of the tree schedule. The variance is $4 \cdot 0.898 + 0.025 + 0.025 - 1.84 \cdot 1.847 = 0.230$. Now, applying the Definition 3, we get the bound on expected waiting times as 3.942, 3.714, and 4.552 slots for B, P, and I frames. The expected queue size can then be found using Little's theorem (Sect. 5.2, Cooper 1981) as, $3.942 \cdot 0.335 + 3.714 \cdot 0.278 \cdot 3 + 4.552 \cdot 0.385 \cdot 6 = 14.971$. Note that the queue size calculated is the expected size. From a practical standpoint however, the maximum queue size may be more interesting. We estimate this using simulations in the next section.

6.3 Hypothesis, experiment, and results

The video streaming system as built can exhibit jittery and bursty behavior: the communication can be jittery, and this in turn leads to bursty arrival patterns at the decoder side. In addition, as the encoder can also switch between different encoding sequences upon a change in the video scene, the number of frames that are generated is also varied with time. In this case study we would like to test the hypothesis: *state-based (e.g., tree scheduled) systems can handle bursty/irregular and jittery behaviour better than schedules that don't consider state (e.g., round robin) systems.*

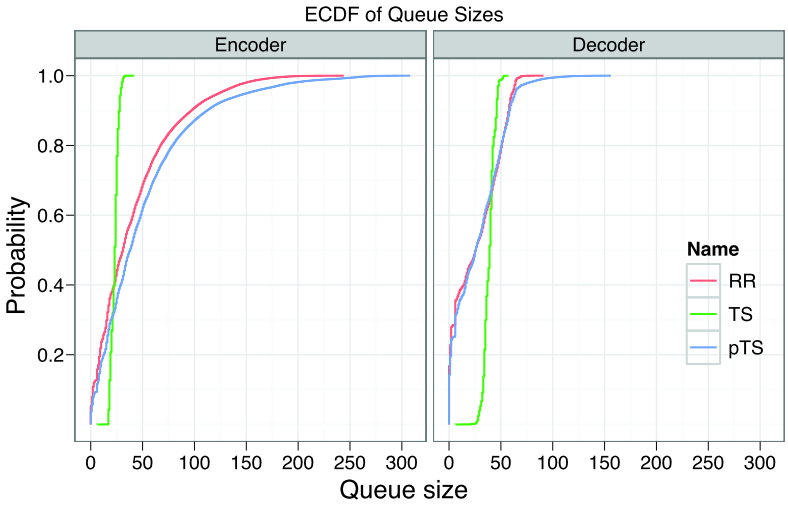


Fig. 16 Buffer size in the data queues of the encoder and decoder

We ran the Simulink model 100 times for a duration of 5 000 produced frames. The resulting data contains 997 936 observations per variable. Our data collection process is as follows: We run the Simulink model in one of the three configurations—round robin, tree schedule, probabilistic tree schedule—from the command line and store the resulting data with a run identifier in a data file. The data files act as input to our statistical analysis.

Our data passed our integrity checks: (1) all queues always have byte count in $\mathbb{N}_{\geq 0}$ (2) all queues always have frame count in $\mathbb{N}_{\geq 0}$, (3) the encoder always produces data at integer values following the specification (t DIV $1 = 0$), (4) the queue byte counts always change by a value between 0 and 6, which corresponds to a combined push and pop action of a frame.

6.3.1 Queue size analysis

Both the encoder and the decoder contain queues to buffer data packets in case of sporadic packet drops or bursty events on the decoder side.

Figure 16 shows the empirical cumulative distribution function (ECDF) for the queue sizes of the different scheduling scenarios. The x-axis shows the buffer size normalized to B-frames; so a value of one means that it has the size of one B-frame. The y-axis shows the probability of a specific buffer size being required at run time. For example, in 80 percent of the simulation cases, the tree schedule requires a queue of at most 25 at the encoder side while round robin requires at least 67 units.

This figure is useful to compare how the different strategies use queues in the system. Due to the large amount of collected data, the confidence bands on the ECDF are minimal and can be ignored. For example the confidence band on 80 percent for round robin is precisely 67. From the data, we can observe the following:

– *Hypothesis holds: Deterministic tree schedules can handle bursts and jitter better than stateless schedules*. The tree scheduled system has a steeper increase in its

probability than the round robin scheduled system. This means that the queue sizes vary less between the different runs. In fact, the round robin scheduled system has a long tail close to the probability of one, which means that building a reliable system with round robin will require much over-provisioning of buffers. The difference between the two systems comes form the fact that tree schedules can react to bursts in the system as they can encode a separate state how to handle a burst.
– *Tree schedules can require smaller queue sizes in bursty and jittery systems*. The cross over point where the tree schedule and the round robin schedule perform equally well is low for both the encoder and the decoder queue. With a high reliability—this equals a low frame drop rate which will occur, if the queue cannot contain all frames—the tree schedule clearly outperforms round robin.

We finally note that, based on the simulations, we found that the mean queue size with probabilistic tree schedules was 13.85041 B frames. As can be seen, this number is close to the theoretical bound (14.971) that we computed in Sect. 6.2 based on the analysis developed in Sect. 4.

6.3.2 *Queue waiting times*

We also investigated the waiting time for packets in the queues. The results are equivalent to the ones for the queue size behavior shown before. The data for tree schedules shows less variation than round robin and this means that the tree schedules provide more robust and predictable timing patterns. Due to space constraints, we cannot display the figures.

6.3.3 *Frame underruns*

The two queues in the system buffer data in case of bursts. A queue underrun occurs, if an entity such as the network or the decoder tries to read from the queue, but the queue is empty. Queue underruns are of particular interest at the decoder side, because when the decoder wants to read the next frame, but the queue is empty, the resulting video will be discontinuous. If many underruns happen at run time, then the video will be choppy.

Table 1 shows the percent of occurred underruns for the different queues. We calculate the percentage as the ration of the total number of underruns over the total number of read requests for the queues. From the data, we can observe the following:

Table 1 Queue underruns of the round robin (RR), the tree schedule (TS), and the probabilistic tree scheduled (pTS) system

	Queue	Schedule	Underruns [%]
1	Encoder	RR	0.0201
2		TS	0.0000
3		pTS	0.0153
4	Decoder	RR	0.1688
5		TS	0.0000
6		pTS	0.1458

– *State-based schedules can be used to enforce boundary conditions*. The tree-schedule-based system cannot experience underruns in the encoder queue caused by the network scheduler, because the schedule at run time explicitly checks for this behavior. A surprising result is that this also results in zero underruns at the decoder side. Note that underruns can occur due to other problems such as loss of the network link.
– *Probabilistic conditions can replace deterministic decisions in a state-based schedule*. The probabilistic schedule works as well as the round robin schedule. The difference between the two systems is statistically insignificant although the probabilistic schedules seems to outperform the round robin one.

7 Implementation of tree schedules

We successfully implemented tree schedules in several prototypes. Due to the space constraints, we provide only a quick overview and cite relevant work.

We can translate a tree schedule into a Network Code programs. Network Code (Fischmeister et al. 2007) is a verifiable, executable specification language for programming access control of shared communication media in a distributed real-time system. It allows encoding application-specific behavior in the programmable media access layer. A Network Code program is thereby basically a small program, specifying, when and what is going to be transmitted and received.

The translation of a tree schedules into Network Code program is straightforward (Anand et al. 2006). Once we generated the program, we can either run it on the software prototype (Fischmeister et al. 2007), or on a hardware-accelerated version programmed on FPGA technology (Fischmeister et al. 2009). We also successfully used tree schedules on a case study with medical devices (Arney et al. 2009) and also have an implementation for switched Ethernet (Carvajal and Fischmeister 2010).

One particular problem that tree schedules create is that on partial distributed consensus. At any choice point where the two different stations in the network may communicate depending on the decision, these two stations must agree on the result of the choice. All other stations can reconstruct the decision based on the data they receive; hence only partial distributed consensus. With the right communication medium, such consensus is easy to achieve. For example, when using the Controller Area Network, then one of the choices uses a higher communication priority than the other. For other communication media, for example Ethernet, which do not provide intrinsic arbitration of concurrent media access, the problem of partial distributed consensus is a topic of ongoing research. Naive solutions like mini-slotting as found in Berwanger et al. (2000) together with state consistency protocols work (Kopetz et al. 2001).

8 Conclusions

Distributed real-time systems require bounded communication delays and achieve them by means of a predictable and verifiable control mechanism for the communication medium often expressed as schedules. Tree schedules provide an expressive

framework for modeling and analysing time-triggered communication schedules. The concept of tree schedules is to find a balance between flexibility and analyzability. For example, tree schedules allow conditional branching based on state information in the communication schedule, however, the control-flow is restricted to tree structures to keep the system analyzable.

In this work, we introduced the notion of tree schedules and provide a framework to model and analyze application-specific communication schedules with tree schedules. We specifically presented results in the context of two message models: (1) one with explicit message deadlines and (2) one with probabilistic information about enabling conditions for transitions in the tree schedule. For the former, we presented results for checking schedulability for fixed and dynamic priority schemes. For the latter, we presented results for analyzing the average waiting time.

In the case study, we have shown the utility of our framework for application-specific schedules and thereby showed the utility of the introduced concept of tree schedules. The case study provides the unexpected result that a simple optimization can yield a high overall improvement of the system. In the case study, we showed that by reducing the network traffic approximately $1/3$, the system settles faster by a factor of 2 and increases its operational range by 2. This makes a strong case that such optimization techniques are valuable for applications such as distributed control systems. The presented framework provides the tools and analysis that allow building complex distributed control systems using such optimization techniques.

While our framework is essential in weighing the costs versus the benefits under different metrics for an application developer in considering application-specific schedules, the framework also serves as an important step towards automatic generation of application-specific media access control based on higher-level requirements. In the future, we will explore the problem of composing different tree schedules which is significant in the context of large scale systems.

Acknowledgements This research was supported in part by NSERC DG 357121-2008, ORF RE03-045, ORE RE04-036, ORF-RE04-039, ISOP IS09-06-037, APCPJ 386797-09, CFI 20314 with CMC, ARO W911NF-11-1-0403, NSF CNS-1117185 and NSF CPS-1135630.

References

Abdeddaïm Y, Kerbaa A, Maler O (2003) Task graph scheduling using timed automata. In: Proc of the 17th international symposium on parallel and distributed processing (IPDPS), Nice, France

Alur R, Dill D (1994) A theory of timed automata. Theor Comput Sci 126(2):183–235

Alur R, Weiss G (2008) Regular specifications of resource requirements for embedded control software. In: RTAS '08: proceedings of the 2008 IEEE real-time and embedded technology and applications symposium. IEEE Computer Society, Washington, pp 159–168. doi:10.1109/RTAS.2008.13

Anand M, Fischmeister S, Lee I (2006) An analysis framework for network-code programs. In: Proceedings of the 6th annual ACM conference on embedded software (EMSOFT), Seoul, South Korea, pp 122–131

Anand M, Easwaran A, Fischmeister S, Lee I (2008) Compositional feasibility analysis for conditional task models. In: Proceedings of the eleventh IEEE international symposium on object-oriented real-time distributed computing (ISORC). IEEE Computer Society, Washington

Arney D, Trausmuth R, Fischmeister S, Goldman JM, Lee I (2009) Plug-and-play for medical devices: experiences from a case study. Biomed Instrum Technol 43:313–317

Baruah SK (1998) Feasibility analysis of recurring branching tasks. In: ECRTS, pp 138–145

Baruah SK, Mok AK, Rosier LE (1990) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: IEEE real-time systems symposium, pp 182–190. URL citeseer.ist.psu.edu/baruah90preemptively.html

Baruah SK, Chen D, Mok AK (1997) Jitter concerns in periodic task systems. In: RTSS '97: proceedings of the 18th IEEE real-time systems symposium (RTSS '97). IEEE Computer Society, Washington, p 68

Berwanger J, Peller M, Griessbach R (2000) ByteFlight—a new high-performance data bus system for safety-related applications. Tech Rep EE-211, BMW AG

Bose SK (2001) Introduction to queueing systems. Kluwer Academic/Plenum Publishers, New York

Carvajal G, Fischmeister S (2010) A TDMA Ethernet switch for dynamic real-time communication. In: Proc of the 18th IEEE symposium on field-programmable custom computing machines (FCCM), Charlotte, United States, pp 119–126

Cervin A, Henriksson D, Lincoln B, Eker J, Arzen KE (2003) How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. IEEE Control Syst Mag 23(3):16–30. doi:10.1109/MCS.2003.1200240

Chakraborty S, Phan LTX, Thiagarajan PS (2005) Event count automata: a state-based model for stream processing systems. In: Proc of the 26th IEEE real-time systems symposium (RTSS), Miami, Florida, USA

CMU (last visited 12/2007) Control tutorials for Mathlab and simulink. Web site, http://www.library.cmu.edu/ctms/ctms/index.htm

Cooper R (1981) Introduction to queueing theory. Edward Arnold, Sevenoaks

Ethernet powerlink V2.0—communication profile specification (2003) Ethernet Powerlink Standadisation Group (EPSG)

Fersman E, Pettersson P, Yi W (2002) Timed automata with asynchronous processes: schedulability and decidability. In: TACAS, pp 67–82

Fischmeister S, Sokolsky O, Lee I (2006) Network-code machine: programmable real-time communication schedules. In: Proc of the 12th IEEE real-time and embedded technology and applications symposium (RTAS), San Jose, United States, pp 311–324

Fischmeister S, Sokolsky O, Lee I (2007) A verifiable language for programming communication schedules. IEEE Trans Comput 56(11):1505–1519

Fischmeister S, Trausmuth R, Lee I (2009) Hardware acceleration for conditional state-based communication scheduling on real-time Ethernet. IEEE Trans Ind Inform 5(3):325–337

FlexRay Consortium (2004) FlexRay communications system—protocol specification. Version 2.0

Hendriks M, Verhoef M (2006) Timed automata based analysis of embedded system architectures. In: Proc of the 20th international workshop on parallel and distributed processing symposium (IPDPS), Rhodes Island, USA

Joseph M, Pandya PK (1986) Finding response times in a real-time system. Comput J 29(5):390–395

Kopetz H (1997) Real-time systems: design principles for distributed embedded applications. Kluwer Academic, Dordrecht

Kopetz H (2004) The fault hypothesis for the time-triggered architecture. In: Proc of the IFIP world computer congress

Kopetz H, Bauer G, Poledna S (2001) Tolerating arbitrary node failures in the time-triggered architecture. In: SAE 2001 world congress, March 2001, Detroit, MI, USA

Lehoczky JP (1996) Real-time queueing theory. In: RTSS '96: proceedings of the 17th IEEE real-time systems symposium (RTSS '96). IEEE Computer Society, Washington, p 186

Lehoczky JP, Sha L, Strosnider JK (1987) Enhanced aperiodic responsiveness in hard real-time environments. In: IEEE real-time systems symposium, pp 261–270

Leung JT, Merrill M (1980) A note on preemptive scheduling of periodic, real-time tasks. Inf Process Lett 11(3):115–118

Liu C, Layland J (1973) Scheduling algorithms for multi-programming in a hard-real-time environment. J ACM 20(1):46–61

Phan LTX, Chakraborty S, Thiagarajan PS, Thiele L (2007) Composing functional and state-based performance models for analyzing heterogeneous real-time systems. In: Proc of the 28th IEEE real-time systems symposium (RTSS), Tucson, Arizona, USA

Phan LTX, Chakraborty S, Thiagarajan PS (2008) A multi-mode real-time calculus. In: Proc of the 29th IEEE real-time systems symposium (RTSS), Barcelona, Spain

Pop P, Eles P, Peng Z (2000) Schedulability analysis for systems with data and control dependencies. In: Euromicro conference on real-time systems, pp 201–208. URL http://www2.imm.dtu.dk/pubdb/p.php?4632

Potop-Butucaru D, de Simone R, Sorel Y, Talpin JP (2009) Clock-driven distributed real-time implementation of endochronous synchronous programs. In: EMSOFT '09: proceedings of the seventh ACM international conference on embedded software. ACM, New York, pp 147–156. doi:10.1145/1629335.1629356

Real J, Crespo A (2004) Mode change protocols for real-time systems: a survey and a new proposal. Real-Time Syst 26(2):161–197

Shin I, Lee I (2003) Periodic resource model for compositional real-time guarantees. In: RTSS, pp 2–13

Shin I, Lee I (2004) Compositional real-time scheduling framework. In: RTSS, pp 57–67

Tindell K, Clark J (1994) Holistic schedulability analysis for distributed hard real-time systems. Microprocess Microprogram 40(2–3):117–134. doi:10.1016/0165-6074(94)90080-9

Wandeler E, Maxiaguine A, Thiele L (2004) Quantitative characterization of event streams in analysis of hard real-time applications. In: Proceedings of the 10th IEEE real-time and embedded technology and applications symposium (RTAS), Toronto, Canada, pp 450–461

Weiss G, Fischmeister S, Anand M, Alur R (2009) Specification and analysis of network resource requirements of control systems. In: Proc of the 12th international conference on hybrid systems: computation and control (HSCC), San Fransisco, United States, pp 381–395

Whitaker J, Benson B (2001) Standard handbook of audio and radio engineering. McGraw-Hill, New York

Madhukar Anand received a B.S and M.S degrees in Mathematics and Computing from the Indian Institute of Technology (IIT), Kharagpur, and a Ph.D. in Computer and Information Science from the University of Pennsylvania in 2008. From Fall of 2008, he is working with the Data Center Routing Team at Cisco Systems. His research interests include, Real-Time and Embedded Systems, Networked Embedded Systems, Data Center Networking, Formal Methods, Hybrid Systems, and Wireless Sensor Networks. Amongst other awards, he has won the Institute Silver Medal for academic excellence from IIT Kharagpur.

Sebastian Fischmeister is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Waterloo, Canada. He received his MASc in Computer Science at the Vienna University of Technology, Austria, and his Ph.D. degree at the University of Salzburg, Austria. He was awarded the APART stipend in 2005 and worked as a research associate at the University of Pennsylvania, USA, until 2008. He performs systems research at the intersection of software technology, distributed systems, and formal methods.

Insup Lee is Cecilia Fitler Moore Professor of Computer and Information Science and Director of PRECISE Center at the University of Pennsylvania. He also holds a secondary appointment in the Department of Electrical and Systems Engineering. He received the B.S. in Mathematics from the University of North Carolina, Chapel Hill and the Ph.D. in Computer Science from the University of Wisconsin, Madison.

His research interests include cyber physical systems (CPS), real-time embedded systems, formal methods and tools, high-confidence medical device systems, and software engineering. The theme of his research activities has been to assure and improve the correctness, safety, and timeliness of life-critical embedded systems. Recently, he has been working in the area of medical cyber physical systems.

He has served on many program committees and chaired many international conferences and workshops. He has also served on various steering and advisory committees of technical societies, including CPSWeek, ESWeek, ACM SIGBED, IEEE TC-RTS, RV, ATVA. He has served on the editorial boards of the several scientific journals and is a founding co-Editor-in-Chief of KIISE Journal of Computing Science and Engineering (JCSE). He was Chair of IEEE Computer Society Technical Committee on Real-Time Systems (2003–2004) and an IEEE CS Distinguished Visitor Speaker (2004–2006). He with his student received the best paper award in RTSS 2003. He was a member of Technical Advisory Group (TAG) of President's Council of Advisors on Science and Technology (PCAST) Networking and Information Technology (NIT), 2006–2007. He is IEEE fellow and received IEEE TC-RTS Outstanding Technical Achievement and Leadership Award in 2008.

Linh T.X. Phan is a Postdoctoral Researcher in the PRECISE Center at the University of Pennsylvania. She received the B.S. degree in Computer Science in 2003 and the Ph.D. degree in Computer Science in 2009 from the National University of Singapore (NUS). Her research interests include formal modeling and analysis methods, system-level design techniques, and compositional analysis methods for real-time embedded systems, cyber-physical systems and multi-mode systems. Some of the application domains she works in include automotive electronics and software, avionics, real-time multimedia, body-area sensor networks and cloud computing. She was a recipient of the Singapore Scholarship (1999–2003) and NUS Graduate Scholarship (2003–2007). For her Ph.D. dissertation, she received the Graduate Research Excellence Award from NUS (2009). She also received the Best Paper Award nomination at EMSOFT 2010. She has served as a co-chair of APRES 2011 and CRTS 2011, and a PC member of ETFA 2010–2012, APRES 2012, RTAS WiP 2010–2012, RTSS-At-Work 2011, CPSNA 2011–2012, EMC 2010–2011, CRTS 2010, WTR 2011, and WCTT 2011.