Resilient Mixed-Trust Scheduling

Dionisio de Niz[†], Bjorn Andersson[†], Hyoseung Kim*, Mark Klein[†], John Lehoczky[‡]
[†]Software Engineering Institute, Carnegie Mellon University
*Electrical and Computer Engineering, University of California, Riverside
[‡]Department of Statistics and Data Science, Carnegie Mellon University
{dionisio,baandersson}@sei.cmu.edu, hyoseung@ucr.edu, mk@sei.cmu.edu, jl16@andrew.cmu.edu

Abstract—**In this paper we present a new scheduling model for resilient real-time mixed trust systems. This model extends the previous Real-Time Mixed-Trust Computing framework RT−MTC to support degradation modes. Management of these modes has been identified in industrial documents as a key requirement for deploying trusted autonomous vehicles for safe autonomy. RT−MTC uses verified components (known as enforcers) to guarantee that the output of a system is safe by replacing it with a verified safe one if this output is deemed unsafe or is not produced on time. In this paper we extend RT−MTC and develop a scheduling model that uses the digraph scheduling model as a baseline but extends it in four critical ways: (1) it creates extensions for the mixed-preemptive scheduling required by RT−MTC, (2) it enables priority bands in order to separate trusted and untrusted components, (3) it uses these bands in order to calculate intermediate deadlines used by the RT−MTC framework for the scheduling of the trusted components, and (4) it defines system mode semantics to obtain two desirable properties of the new schedulability analysis: low pessimism and low time-complexity. This paper evaluates the new schedulability algorithm and shows that it is efficient in that it only needs to analyze one transition at a time. The new model supports the construction of a resilient autonomous system with provable guarantees protected by verified enforcers within the RT−MTC framework and, more importantly, preserves these guarantees even across failure-triggered mode changes.**

I. Introduction

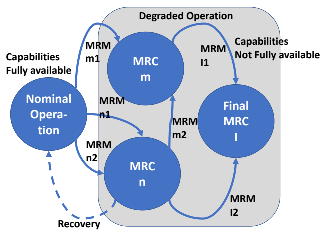
Certification authorities (e.g., FAA [17]) allow the validation of different parts of a system with different degrees of rigor depending on their level of criticality. Formal methods have been recognized as important methods to verify safety-critical components [1]. Unfortunately, a verified property can be easily compromised if the verified components are not protected from unverified (and untrusted) ones. Thus, the authors of [8] require that trusting the guarantee of verified components considers both verification and protection of components.

The work in [8] presented a *real-time mixed-trust computing* (RT−MTC) framework to achieve trust as previously defined. This framework enables the use of untrusted components even for CPS critical functionality. In this framework, untrusted components are monitored by verified components ensuring that the output of the untrusted components always leads to safe states (e.g., avoiding crashes). These monitoring components are known as *logical enforcers* [5], [9]. To ensure trust, these enforcers are protected by a verified micro-hypervisor [23]. To preserve the timing guarantees of the system, RT−MTC uses *temporal enforcers*, which are small, self-contained codeblocks that perform a default safety action (e.g., *hover* in a quadrotor) if the untrusted component has not produced a correct output by a specified time. Temporal enforcers are contained within the verified micro-hypervisor without jeopardizing the existing level of trust (e.g., using compositional verification offered by extensible micro-hypervisors [23]). Together the untrusted part and the temporal enforcers are scheduled as a combined task called a *mixed-trust task*. The untrusted part, known as a guest task (GT), is scheduled by a fixed-priority scheduler in an untrusted VM. The VM runs on top of a trusted hypervisor that executes the trusted temporal enforcer, known as a hypertask (HT). Furthermore, the authors presented a sample application for an autonomous drone vehicle showcasing how RT−MTC preserves the safety of the system under both permanent and temporal failures. The authors of [8] also presented the analysis to verify the schedulability of mixed-trust tasksets in a single operating mode. In this paper we extend this framework to support *degraded modes of operations* to adapt to faults and environmental changes in order to reach the appropriate level of resilience.

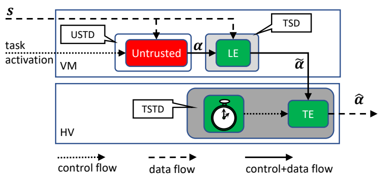
Achieving resilient operation across multiple degraded modes has been and continues to be a major goal for autonomous architectures. With the increasing push to deploy autonomous cars, such considerations made their way into industrial standards. For instance, in [12], Daimler et al. define an architecture for autonomous driving where degraded modes of operation are one of the key building blocks. In particular, [12] introduces the concept of a *Minimal Risk Condition* (MRC) as an operating mode to which an automated-driving system must transition upon a failure. Such a transition is performed by a *Minimal Risk Maneuver* (MRM). This approach is adopted from principles of ISO 26262 [13]. Figure 1a depicts an illustration of the modes and transitions proposed in [12].

While a number of real-time modal models have been proposed, they fail to address the challenges presented here in at least two important respects. First, previous models consider mode transitions as simple task parameter changes without taking into account the computation required by the transition and the synchronization between the modes and the transition. This is evidenced by the explicit description in [12] of the MRM. Second, previous work does not address the challenges imposed by the need to preserve safety guarantees during the transition. This paper addresses these issues by extending the RT−MTC framework to include degradation modes.

The extended RT−MTC framework supports trusted degra-



(a) MRC Modes [12]



(b) Mixed-Trust Task Runtime

Fig. 1: Behavior and Runtime Architecture

dation modes where for each mode there exists a number of potential failures that require a specific transition into a degraded mode. This continues until we reach a fail-stop mode where the system completely stops. Our RT-MTC extended framework supports degradation modes where we not only enforce trust during an operating mode but also during a mode transition. To analyze the schedulability of these systems we use an extension of the digraph model [19] in order to capture the nuances of the timing constraints of the transitioning tasks and the timing relationships with both their respective source and target mode GTs and HTs.

Four main limitations of the digraph model prevent its direct application to trusted mode changes:

1) **Mixed-Preemption**. While the digraph literature has been extended to support both preemptive and non-preemptive scheduling, the RT-MTC framework requires tasks with preemptive (in the VM) and non-preemptive (in the HV) segments executed by two separate schedulers.
2) **System Modes.** The flexibility of the digraph model allows tasks to change modes independently of each other. Unfortunately, this is a problem when the system requires these changes to be coordinated, e.g. when the entire system needs to adapt to a failure. This can cause a severe schedulability penalty.
3) **Intermediate Deadlines.** The intermediate deadline needed for the temporal enforcer of RT-MTC, denoted as E_i, must be calculated based on the schedulability analysis of the HTs. Moreover, these deadlines are linked to edge interarrival parameters in order to capture the semantics of the degradable RT-MTC. The digraph model does not have support for this, and it must be added.
4) **Priority Bands.** The RT-MTC framework requires HTs to be scheduled in a higher priority band than the GTs. This requires a subtle extension of the digraph model that must be integrated with the other extensions.

The contributions of this paper are three-fold. First, we created an extended mixed-trust framework that supports models of degradation required by industry without compromising the guarantees offered by the original mixed-trust framework. To the best of our knowledge this is the first framework that supports these modes. Secondly, we created a schedulability model based on the digraph model that supports this framework and overcomes the limitations cited above. Finally, we took advantage of the structure of our framework to create an analysis algorithm with low time-complexity and pessimism.

The rest of this paper is organized as follows. In Section II we provide a review of the mixed-trust framework. In Section III we show that the adaptation to failure can compromise the guarantees offered by the RT-MTC framework. We then present our semantics that are designed to preserve safety across adaptations. In Section IV we present the mixed-trust digraph scheduling model. We first introduce the standard digraph model and then present our generalizations. In Section V we then present the system modes and their corresponding system transitions to complete our model. With the generalized model, we present a procedure to calculate the E_i parameter for our modal tasks and the integrated system scheduling. In Section VI we then present the evaluation of our schedulability model. In Section VII we discuss our implementation. Section VIII presents related work, and Section IX offers concluding remarks.

II. Real-Time Mixed-Trust Background

We first provide some background on the RT-MTC framework. Figure 1b depicts the architecture of a mixed-trust task. The figure shows how the output of the untrusted component in red is validated by the logical enforcer (LE), in green, to make sure the value is safe[1]. However, because the untrusted component may delay the production of the output, an independent timer is programmed within the hypervisor (HV) that triggers the temporal enforcer (TE) if no output is produced by the LE. If the LE does produce an output before the timer interrupts, then this value is used as the output (actuation in a controller).

The architecture also presents three protection domains: An *Untrusted Space and Time Domain*, which has no memory or time protection, where the untrusted components reside; a *Trusted Space Domain*, which ensures the memory of the component is not compromised, where the LE resides; and the *Trusted Space and Temporal Domain*, which provides both memory protection and timing protection and it uses its own timer, where the TE lives.

The system model in [8] considers a uniprocessor system with a taskset $\Gamma = \{\mu_i | \mu_i = (T_i, D_i, \tau_i, \kappa_i)\}$ with unique priorities. In the taskset, μ_i is a mixed-trust task with two execution segments, τ_i and κ_i, with period T_i and deadline D_i. The segment τ_i is considered to be untrusted and scheduled preemptively in the untrusted OS kernel inside the VM. The

[1]see [5] for the formalization of this scheme.

segment κ_i is considered to be trusted code and scheduled non-preemptively within the trusted HV. To represent the fact that these segments are handled by different schedulers, they are considered to be tasks and call τ_i the *guest task* (*GT*) and κ_i the *hyper task* (*HT*), as introduced before. These tasks are defined by: $\tau_i = (T_i, E_i, C_i)$, $\kappa_i = (T_i, D_i, \kappa C_i)$, where T_i and D_i are the same as in μ_i, C_i is the worst-case execution time (WCET) of τ_i, κC_i is the WCET of κ_i, and E_i is the intermediate deadline of τ_i determined by analysis. Consider a particular job of μ_i, $(\tau_{i,q}, \kappa_{i,q})$. Ideally, $\tau_{i,q}$ will execute correctly taking no more than C_i time units and finishing within E_i time units after its arrival. In this case, the job $\kappa_{i,q}$ is not activated. The logical enforcer (LE) verifies the correctness of $\tau_{i,q}$, while the timing enforcer (TE) verifies the timing. If the logical enforcer (LE) does not notify the HV that $\tau_{i,q}$ finished correctly and on time, then the corresponding HT $\kappa_{i,q}$ is activated by a timer set to expire E_i time units after τ_i arrives running at a higher priority than any GT. The deadline for $\tau_{i,q}$, E_i, is chosen to ensure that $\kappa_{i,q}$ can finish by D_i, the μ_i deadline. Equations to verify schedulability were derived in [8].

We begin by showing the schedulability equations in [8]; this is useful because we will make changes to these equations to model and analyze degradation modes. In Section IV, we will also present an improved approach.

The reasoning of the schedulability analysis in [8] is as follows. The response time of κ_i is calculated by first calculating the maximum duration of a level-i active period:

$$t_i^{\kappa} = \max_{j \in \kappa L_i} \kappa C_j + \left\lceil \frac{t_i^{\kappa}}{T_i} \right\rceil \kappa C_i + \sum_{j \in \kappa H_i} \left\lceil \frac{t_i^{\kappa}}{T_j} \right\rceil \kappa C_j, \quad (1)$$

where κL_i is the set of all HTs with lower priority than κ_i and κH_i is the set of tasks with higher priority than κ_i.

The latest start time of the q^{th} job of κ_i in the active period is obtained by:

$$w_{i,q}^{\kappa} = \max_{j \in \kappa L_i} \kappa C_j + (q-1)\kappa C_i + \sum_{j \in \kappa H_i} (\left\lfloor \frac{w_{i,q}^{\kappa}}{T_j} \right\rfloor + 1)\kappa C_j, \quad (2)$$

Finally, the response time of κ_i is upper-bounded by:

$$R_i^{\kappa} = \max_{q \in \{1 \ldots \lceil \frac{t_i^{\kappa}}{T_i} \rceil\}} (w_{i,q}^{\kappa} + \kappa C_i - (q-1)T_i). \quad (3)$$

Given the response time of a HT, E_i is calculated as: $E_i = D_i - R_i^{\kappa}$, which serves as the deadline of the GT.

To calculate the response time of the GT, it is necessary to evaluate all the potential phasings of the interfering GTs (higher-priority) and HTs (all HTs except its own). To simplify this, [8] defines the request bound function (rbf) that captures the computation time of task μ_i as presented in equation (4).

$$\text{rbf}_i^y(t,b) = \begin{cases} \left\lceil \frac{t-(T_i-E_i)}{T_i} \right\rceil^+ C_i b + \left\lceil \frac{t}{T_i} \right\rceil \kappa C_i & \text{if } y = E, \\ \left\lceil \frac{t}{T_i} \right\rceil C_i b + \left\lceil \frac{t-E_i}{T_i} \right\rceil^+ \kappa C_i & \text{if } y = A, \end{cases} \quad (4)$$

where $\lceil x \rceil^+ = \max(0, \lceil x \rceil)$, $y \in \{E, A\}$ indicates if the time interval of duration t starts with the arrival of GT of μ_i (A) or its HT (E), and $b \in \{0, 1\}$ indicates if the GT execution should be included in the rbf.

The rbf is then used to calculate the maximum level-i busy period (where $x \in \{E, A\}$ indicates if it starts with HT or GT) with the smallest solution to:

$$\begin{aligned} t_i^{g,x} = & \left(\sum_{j \in L_i} \text{rbf}_j^E(t_i^{g,x}, 0) \right) + \text{rbf}_i^x(t_i^{g,x}, 1) \\ & + \sum_{j \in H_i} \max_{y \in \{E,A\}} \text{rbf}_j^y(t_i^{g,x}, 1), \end{aligned} \quad (5)$$

where L_i and H_i contain the tasks with lower and higher priority (respectively) than τ_i.

The latest start time of the q^{th} job of a GT τ_i in the level-i busy period, $w_{i,q}^{g,x}$, is computed as the smallest solution of:

$$\begin{aligned} w_{i,q}^{g,x} = & \left(\sum_{j \in L_i} \text{rbf}_j^E(w_{i,q}^{g,x}, 0) \right) + qC_i \\ & + (q - 1 + I_{(x=E)})\kappa C_i + \sum_{j \in H_i} \max_{y \in \{E,A\}} \text{rbf}_j^y(w_{i,q}^{g,x}, 1), \end{aligned} \quad (6)$$

where I_{ϕ} is an indicator function that returns 1 if ϕ is true and 0 otherwise.

The response time of a job for different phasings is computed using:

$$R_{i,q}^{g,x} = w_{i,q}^{g,x} - ((q-1)T_i + I_{(x=E)}(T_i - E_i)). \quad (7)$$

The maximum response time among all the jobs in the busy period is calculated using

$$R_i^{g,x} = \max_{q \in \left\{1 \ldots \left\lceil \frac{t_i^{g,x} - I_{x=E}(T_i - E_i)}{T_i} \right\rceil \right\}} R_{i,q}^{g,x}. \quad (8)$$

Finally the response time of a GT τ_i is given by:

$$R_i^g = \max_{x \in \{E,A\}} R_i^{g,x}. \quad (9)$$

With these equations, the response time of all the HTs is calculated first, then their respective E_i intervals, and finally the response time of the GTs. A taskset is schedulable if after all the calculated response times of the HTs are less than or equal to their deadline and all the response times of the GTs are less than or equal to their respective E_i.

III. Mode Transition Semantics

Ensuring a predictable mode transition behavior is critical for the preservation of guarantees across these transitions. In this section we first discuss how failures affect the guarantees in the mixed-trust framework, and then we present the semantics of the new transition protocol designed to support *transitioning* enforcers that run as HTs to prevent the violation of these guarantees.

A. Enforced Modes and Transition Guarantee Gaps

While the RT-MTC framework allows us to enforce unverified components to guarantee a safety property within an operating mode, it is insufficient to combine modes with mode transitions while still preserving the guarantee. To see this, consider the example of an autonomous car with two crash avoidance enforcers namely:

- a **high-speed enforcer** that uses a lidar to detect an object with a detection range of 20m and braking power of

$10m/s^2$. The detection range allows the car to reach a maximum speed of $20m/s$ and still brake in time to avoid a crash.

- a **low-speed enforcer** that uses a sonar sensor to detect objects with a detection range of 5m and the same braking power. The detection range allows the car to reach a maximum speed of $10m/s$ and still brake in time to avoid a crash.

While the car can be safely enforced with the high-speed or the low-speed enforcers if we ensure that the respective top speed is preserved, it is not possible to switch from one enforcer to another without further considerations. For instance, an autonomous car may want to use the high-speed enforcer if the lidar is working but if it breaks down (or it is too foggy to work properly), it may want to switch to the low-speed enforcer that uses the sonar instead. Unfortunately, an instantaneous switch-over may break the assumptions of the low-speed enforcer. Specifically, if the lidar breaks while the car is traveling at speed v such that $10 < v < 20$ and we try to switch to the low-speed enforcer, its assumed top speed of $10m/s$ would be violated, hence the guarantee would be invalidated. We call this situation a **guarantee gap**.

The guarantee gap between two enforcers can be closed if we add an intermediate *transitioning* enforcer that drives the system in a guaranteed manner from one environment where the first enforcer was valid (top speed of $20m/s$) to another where the second enforcer is valid (top speed of $10m/s$). In our example, a transitioning enforcer could be implemented to start braking at $10m/s^2$ at the instant the lidar stops working and stop braking when the car reaches the speed of $10m/s$. It is worth noting that such an enforcer will be consistent with the enforcer action and the point of failure of the high-speed enforcer, that is, the speed v would be $v \leqslant 20m/s$ and the "last" distance from the obstacle d before the lidar broke was $d \geqslant 20m$.

From the RT-MTC scheduling point of view, this is reflected as two task modes where each task mode has a GT and HT and an intermediate transitioning mode that only has a HT, since this transition needs to be trusted and cannot contain an untrusted component. It is worth noting that the transitioning HT also needs to execute periodically since it is expected to set a setpoint to the (brake) controller and periodically monitor its progress until it reaches the desired condition (speed).

B. Synchronized Degradation

Following [12], we define a *synchronized modal transition* that (1) ensures that the first activation of the transitioning HT occurs at the time when the source-mode HT would have occurred; and (2) the first activation of the GT of the target mode occurs when the last period of the transitioning HT ends. This is depicted in Figure 2.

Note that special care must be taken when assigning values to E-parameters. Let us consider a mode m. Let $R_i^{\kappa,m}$ denote the response time of the HT of a mixed-trust task μ_i in mode

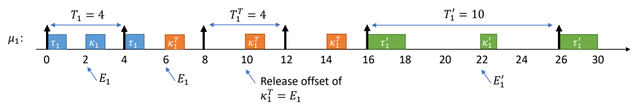


Fig. 2: Synchronized Degradation Timeline.

m and let E_i^m denote the E-value of the GT of μ_i in mode m. Then we require that:

$$E_i^m \leqslant D_i^m - R_i^{\kappa,m} \tag{10}$$

Extra care must be exercised, because the E-value also depends on the transition. Let $R_i^{\kappa,m,m'}$ denote the response time of the HT of a mixed-trust task μ_i in the transition from mode m to mode m'. Then we require that:

$$E_i^m \leqslant D_i^m - R_i^{\kappa,m,m'}. \tag{11}$$

The synchronized transition provides a design abstraction in line with the desire to preserve a timing guarantee across mode changes. Specifically, in our system, a mode degradation can degrade the mission performance (e.g., car speed) but not the safety property (e.g., crash avoidance). This means that the timing requirements of the first activation of the transitioning HT should be the same as the timing requirements of the source mode HT. This is the most stringent requirement for the transitioning HT, since it will transition to a degraded mode with more relaxed timing requirements (e.g., due to the slower speed of $5m/s$). The synchronized activation of the guest task of the target mode is just a natural consequence of the activation.

When recovering from a degraded mode (e.g., when the lidar becomes operational again), we use a synchronized transition protocol that involves a transitioning HT and a period alignment with the target mode guest task. In this case we assume that the more stringent restrictions (e.g., faster speed—$20m/s$) will not become operational until the target mode is fully operational.

C. Trusted Transition Initiation

The preservation of the guarantees across mode transitions requires the transition of the HT within the trusted HV and the execution of the transition in a trusted manner. To align this with the RT-MTC model we design our transition initiation to be part of the HT triggering mechanism. More specifically:

1) We use the trusted HT-triggering timer to trigger the detection of the transition initiation event (e.g., lidar failure).
2) When the HT timer elapses we decide whether to execute the current mode HT or the transitioning HT. This leads to a replacement effect where the transitioning HT replaces the source mode HT when the E_i timer elapses as shown in Figure 2 at time instant 6.

D. System Modes

It is worth noting that a single event (e.g., lidar failure) can cause multiple tasks to switch modes (e.g., collision avoidance and pedestrian detection). This has two important effects that

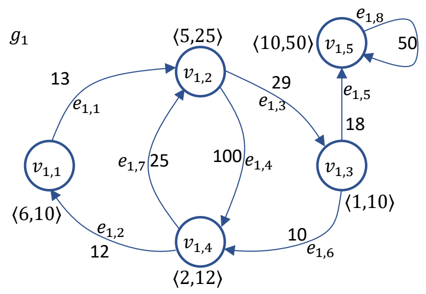


Fig. 3: Sample Digraph Model

must be considered. First, when testing the schedulability of a task in one mode it should not consider tasks not active in that mode. Secondly, because the detection of the triggering mechanism occurs in the HT of each task, this detection can occur at different times in different tasks. This enables the specification of the timing requirements of each task transitioning need independent of each other. To consider these effects we create system modes that group tasks that are active at the same time and that react to the same transitioning events. We discuss this in Section V.

Now that we have established the critical design features for our mode transition protocol, we will discuss the mapping of the mixed-trust tasks into the digraph scheduling model.

IV. Mixed-Trust Digraph Scheduling

In this section we present first, how the digraph model allows us to accommodate the complex timing dependencies between the GT and HT of a mixed-trust task in both source and target modes along with the transitioning HT. And secondly, the limitation of the digraph model and our extensions to accommodate (1) both preemptive and non-preemptive scheduling of GT and HT, respectively, (2) the system-mode partitioning that disables tasks from one mode when another is active, (3) intermediate deadlines to support the HT activation, and (4) priority bands to accommodate the HT's higher-priorities. In this section, we present these extensions. We also introduce a new approximation for low-priority blocking. Based on these, a detailed modeling of system modes will be discussed in Section V.

A. Digraph Model Background

We first introduce the basic digraph model as originally presented in [22], with altered notation that is compatible with the RT-MTC model notation used in this paper. The authors in [22] describe a digraph taskset as $\{g_1, g_2, \ldots\}$ where each task is represented by a directed graph $g_i = (V_i, \mathcal{E}_i)$ with a set of vertices $V_i = \{v_{i,1}, v_{i,2}, \ldots\}$ and a set of edges $\mathcal{E}_i = \{e_{i,1}, e_{i,2}, \ldots\}$. Each vertex is labelled as $v_{i,k} = (C_{i,k}, D_{i,k})$ with $C_{i,k}$ representing the worst-case execution time and $D_{i,k}$ the relative deadline. Each edge is labelled as $e_{i,k} = (v_{i,s}, v_{i,t}, T_{i,k})$ where $v_{i,s}$ is the source vertex, $v_{i,t}$ is the target vertex, and the $T_{i,k}$ is the interarrival time between two vertices. A vertex in the model encodes a type of execution requirement of a job and its deadline, and each edge represent an interarrival time between two types (or the same type, if self loops are possible) of possible job instances. Figure 3 depicts a sample digraph with labelled vertices and edges and annotations $\langle C_{i,k}, D_{i,k}\rangle$ for each vertex $v_{i,k}$ and $T_{i,k}$ for each edge $e_{i,k}$.

A task graph in this model generates jobs by visiting each vertex (generating a job with that vertex's timing parameters), then selects non-deterministically an outgoing vertex, waits at least for the vertex's interarrival time and moves to the target vertex. We say that a task graph is schedulable using a particular scheduling protocol if all possible jobs that the task graph can generate can always finish by their corresponding deadline even when experiencing the worst case interference from all the other task graphs with higher priority.

The digraph scheduling model was originally defined for the Earliest Deadline First scheduling policy [19] but was later extended for fixed-priority scheduling [22]. The fixed-priority model will be used as the basis of our analysis and we describe it next.

1) Fixed-Priority Digraph Scheduling: In [22], the authors evaluate schedulability with the help of a request function (rf) for a path $\pi_j = (v_{j,0}, \ldots, v_{j,l})$ of a task graph g_j.

$$rf_{\pi_j}(t) := max\{e(\pi'_j) | \pi'_j \text{ is a prefix of } \pi_j \text{ and } p(\pi'_j) < t\}, \tag{12}$$

where

- $e(\pi_j) := \sum_{k=0}^{l} C_{j,k}$ and
- $p(\pi_j) := \sum\{T_{j,k} | e_{j,k} = (v_{j,r}, v_{j,r+1}, T_{j,k}) \wedge 0 \leqslant r \leqslant l-1\}$.

For instance, for a path $\pi = (v_{1,1}, v_{1,2}, v_{1,3})$ from the example in Figure 3 its $rf_\pi(50) = 6 + 5 + 1 = 12$ with a $p(\pi) = 13 + 29 = 42$. Note that a smaller t, say 40, would not allow the inclusion of $v_{1,3}$ as part of the request giving $rf_\pi(40) = 11$.

We say that a vertex $v_{i,k}$ is schedulable with an interference set $hp(i)$ if and only if

$$\forall \bar{\pi} \in \Pi(hp(i)) : \exists t \leqslant D_{i,k} : C_{i,k} + \sum_{g_j \in hp(i)} rf_{\pi^{(g_j)}}(t) \leqslant t, \tag{13}$$

where $\Pi(g_i)$ is the set of paths of g_i, $\Pi(\Gamma) := \Pi(g_1) \times \ldots \times \Pi(g_N) | \Gamma = \{g_1, \ldots, g_N\}$ denotes all combinations of paths from all tasks, and $\bar{\pi} = (\pi^{(g_1)}, \ldots, \pi^{(g_N)})$ denotes a single combination of paths in $\Pi(\Gamma)$.

This schedulability test can be applied directly to task graphs that have graph-wide priorities by simply calculating the interfering set $hp(i)$ for a graph g_i and testing all its vertices for schedulability.

2) Static Job-Type Priorities: The digraph model was extended in [22] to accommodate job-type priorities, i.e., where each vertex can have its own priority, by using the busy window concept. This busy window concept was needed to take into account the fact that a high-priority job from a vertex $v_{i,k}$ can delay a medium priority job from a vertex $v_{j,r}$ that causes a preemption on a job from a vertex $v_{i,k+1}$ (a successor of $v_{i,k}$). This delay shortens the interarrival time between two consecutive jobs of the vertex $v_{j,r}$ increasing the interference on the $v_{i,k+1}$ job. To take into account this delay effect, Stigge et al. [22] define the suffix request function (denoted $\mathrm{rf}_{\pi}^{\mathrm{sfx}}(t)$) that traverses previous nodes going backwards from the node of interest. This is defined as:

$$\mathrm{rf}_{\pi}^{\mathrm{sfx}}(t) := max\{e(\pi')|\pi' \text{ is suffix of } \pi \text{ and } p(\pi') \leqslant t\}. \quad (14)$$

It is worth noting that (14) uses $\leqslant t$ to include computations that arrive at the end of the time interval of duration t.

The following schedulability condition for $v_{i,q}$ can be applied

$$\forall \mathrm{rf}^{\mathrm{sfx}} \in \mathrm{RF}^{\mathrm{sfx}}(g_i[\leqslant v_{i,q}], v_{i,q}), \bar{\mathrm{rf}} \in \mathrm{CRF}(\Gamma[\leqslant v_{i,q}]), x \leqslant L : \\ \exists t \leqslant D_{i,q} : \mathrm{rf}^{\mathrm{sfx}}(x) + \sum_{g_j \in \Gamma} \mathrm{rf}^{(g_i)}(x+t) \leqslant x + t, \quad (15)$$

where this condition is quantified over all critical[2] suffix request functions ($\mathrm{RF}^{\mathrm{sfx}}$) for g_i which represent paths ending in $v_{i,q}$ (captured with the notation $[\leqslant v_{i,q}]$) and all combinations of critical request functions (CRF) from all other tasks. This is tested over all time interval durations x up to maximum duration L. If in this test there is an interval of duration t where all of the workload from the other task and this vertex finish then the condition is satisfied. A bound on L can be obtained by using the most abstract request function (mrf) given in [22].

3) Non-Preemptive Digraphs: Non-preemptive scheduling can be modeled in digraphs by first calculating the blocking term due to lower-priority tasks that start executing before the vertex of interest. This is captured by

$$B(lp(i)) = max\{C_{j,q}|v_{j,q} \in g_j, g_j \in lp(i)\}, \quad (16)$$

where $lp(i)$ is the set of lower priority tasks. Then an inclusive request function (to give preference to lower-priority tasks when they arrive simultaneously with higher-priority — the worst case) is defined as:

$$\mathrm{rf}_{\pi}^{\bullet}(t) := max\{e(\pi')|\pi' \text{ is prefix of } \pi \text{ and } p(\pi') \leqslant t\}, \quad (17)$$

and the schedulability condition is modified to

$$\forall \mathrm{rf}^{\mathrm{sfx}} \in \mathrm{RF}^{\mathrm{sfx}}(g_i, v_{i,q}), \bar{\mathrm{rf}} \in \mathrm{CRF}^{\bullet}(hp(I)), x \leqslant L : \\ \exists t \leqslant D_{i,q} - C_{i,q} : B(lp(i)) + \mathrm{rf}^{\mathrm{sfx}}(x) - C_{i,q} + \\ \sum_{g_j \in hp(i)} \mathrm{rf}^{(g_j)}(x+t) \leqslant x + t. \quad (18)$$

It it worth noting that (18) basically calculates the starting instant of the task and verifies that this instant leaves enough time for $C_{i,q}$ to finish by its deadline.

The non-preemptive scheduling analysis presented in [22] was not for job-type priorities. Job type priorities are actually needed to capture the different priorities of HT and GT. However, this is a straightforward extension where the execution time of a vertex is considered to be zero if its priority is lower than that of the vertex we are testing. This approach was implemented in our extension.

[2]critical functions are defined in [22] but not discussed here for brevity.

B. Mixed-Preemption Digraphs

Having presented the digraph background, we next present the extensions needed to accommodate our resilient mixed-trust multi-modal model. The first challenge we address for mixed-trust digraphs is the mixture of preemptive and non-preemptive scheduling. To simplify the discussion, we use the function $\chi(v_{i,k})$ that returns true if the vertex is non-preemptive and false otherwise.

The general strategy to enable mixed preemption is to develop a conditional schedulability test where if the vertex of interest is non-preemptive we calculate the starting time and if not, we calculate the completion time.

1) Suffix Path Approximation: One of the big challenges for digraph scheduling is the complexity of its algorithms. The mixed-trust scheme can potentially bring additional complexity that starts with the fact that the suffix path calculation needs to accommodate the additional digraph nodes and transitions of the mixed-trust task model (GT and HT). Hence, we decided to start with a simpler over-approximation of the suffix path calculation.

Let us first observe that the delay effect of the suffix path can cause at most one additional task preemption from a higher-priority vertex. This is the traditional carry-in effect of a self-suspending task. Hence, we decided to take advantage of previous results in this area [6] and replace the suffix path calculation with a task-wide calculation of the additional exposure to preemptions. This is captured by the maximum jitter that an interfering task can have.

$$J(g_j) = \max_{v_{j,q} \in V_j} (D_{j,q} - C_{j,q}). \quad (19)$$

This allows us to build a request function that does not require the path suffix transversal.

We create a conditional request function based on (12) that either includes computations that arrive at the end of the interval t (if the vertex $v_{i,k}$ being tested is non-preemptive) or excludes them (if the vertex is preemptive). We define

$$rf_{\pi_j}^{v_{i,k}}(t) := max\{e(\pi'_j)|\pi'_j \text{ is a prefix of } \pi_j \text{ and } end(\pi'_j, v_{i,k})\}, \quad (20)$$

where

$$end(\pi, v_{i,k}) = \begin{cases} p(\pi) \leqslant t & \text{if } \chi(v_{i,k}), \\ p(\pi) < t & \text{otherwise.} \end{cases} \quad (21)$$

We can calculate the maximum interfering interval (where other tasks can interfere) for a vertex $v_{i,k}$ as the minimum solution to (22)

$$MI(v_{i,k}) = P(v_{i,k}) + \sum_{g_j \in hp(i)} rf_{\pi^{(g_j)}}^{v_{i,k}}(MI(v_{i,k}) + J(g_j)), \quad (22)$$

where $rf_{\pi^{(g_j)}}^{v_{i,k}}$ is defined in a similar fashion to (13) but using (20) instead of (12). In addition, $P(v_{i,k})$ defines the pending work that is conditionally defined as:

$$P(v_{i,k}) = \begin{cases} B(lp(i)) & \text{if } \chi(v_{i,q}), \\ C_{i,k} & \text{otherwise,} \end{cases} \quad (23)$$

and $N(v_{i,k})$ is defined as the non-preemptible work that the task is allowed to do once it starts to execute. This is defined as:

$$N(v_{i,k}) = \begin{cases} C_{i,k} & \text{if } \chi(v_{i,k}), \\ 0 & \text{otherwise.} \end{cases} \tag{24}$$

We next create the schedulability condition as follows.

$$\forall \bar{\pi} \in \Pi(hp(i)) : MI(v_{i,k}) \leqslant D_{i,k} - N(v_{i,k}). \tag{25}$$

Note that (25) verifies that a vertex finishes before its deadline if it is preemptible or starts $C_{i,k}$ units before its deadline if it is non-preemptible.

C. Mixed-Trust Digraph Mapping

Now that we have the basic building blocks to test schedulability of a mixed-trust task with a digraph extension model, we discuss the mapping of GTs and HTs of a mixed-trust task to a digraph model and the transitioning HT.

Specifically, we will build a modal system using digraph in three stages. First, we will create a *uni-modal mixed-trust task graph (UMMT)* that models a mixed-trust task in one mode as a digraph. Secondly, we will create a *multi-modal mixed-trust task graph (MMMT)* that models modes with mixed-trust tasks graphs that are active in different modes with a transitioning graph that captures the execution of the transitioning HT as explained in Section III. These two stages allow us to evaluate the schedulability of tasks with the modal behavior from Section III that forms the basis of our schedulability. In the final stage, we build a *system-level meta graph* that combines the MMMTs of all mixed-trust tasks and their transitioning HTs and group them into system-level modes. This stage is elaborated in the next section in order to describe the model transformation required to verify schedulability of system transitions.

Uni-Modal Mixed-Trust Task Graph (UMMT). A UMMT $\mu g_i = (V_i, \mathcal{E}_i)$ corresponding to mixed-trust task $\mu_i = (\tau_i, \kappa_i)$ is built with a pair of vertices $V_i = \{(v_{i,1} = (C_{i,1}, D_{i,1}), v_{i,2} = (C_{i,2}, D_{i,2}))\}$. The vertices $v_{i,1}$ and $v_{i,2}$ represent the GT $\tau_i = (T_i, E_i, C_i)$ and the HT $\kappa_i = (T_i, D_i, \kappa C_i)$, respectively, with $C_{i,1} = C_i$, $C_{i,2} = \kappa C_i$, $D_{i,1} = E_i$ and $D_{i,2} = D_i - E_i$. We use $\chi(v_{i,j})$ to tell if $v_{i,j}$ corresponds to a HT or not, i.e., $\chi(v_{i,1}) = F$ (false) and $\chi(v_{i,2}) = T$ (true). μg_i has three edges: $\mathcal{E}_i = \{e_{i,1} = (v_{i,1}, v_{i,2}, T_{i,1}),\ e_{i,2} = (v_{i,2}, v_{i,1}, T_{i,2}),\ e_{i,3} = (v_{i,1}, v_{i,1}, T_{i,3})\}$ with $T_{i,1} = E_i$, $T_{i,2} = T_i - E_i$, and $T_{i,3} = T_i$. While in this mapping C_i, T_i, D_i are the given parameters of the task μ_i, E_i is calculated during the analysis process. We will show how this is done with (27).

Multi-Modal Mixed-Trust Task Graph (MMMT). An MMMT $\hat{\mu g}_m = (\mathcal{G}_m, \mathcal{R}_m)$ is composed of a set of mixed-trust task graphs $\mathcal{G}_m = \{\mu g_1, \ldots, \mu g_N\}$ connected through a set of transitioning task graphs $\mathcal{R}_m = \{g_{N+1} \ldots, g_{N+M}\}$. Each transitioning task graph $g_t \in \mathcal{R}_m$ has a single vertex $V_t = \{v_{t,1}\}$ and three edges $\mathcal{E}_t = \{e_{t,1}, e_{t,2}, e_{t,3}\}$, where $e_{t,1}$ connects from the source GT vertex to the transitioning HT one, $e_{t,2}$ is a self-loop, and $e_{t,3}$ is from the transitioning HT to the target GT. Then, in g_t, the parameters of vertices are characterized by $V_t = \{v_{t,1} = (C_{t,1}, D_{t,1})\}$ with $D_{t,1} = D_{i,2}$

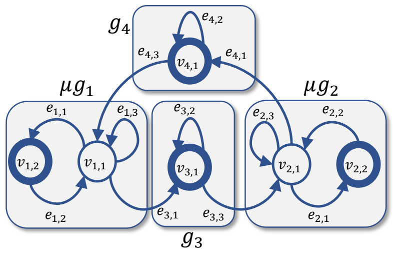


Fig. 4: Digraph $\hat{\mu g}_1$ (MMMT)

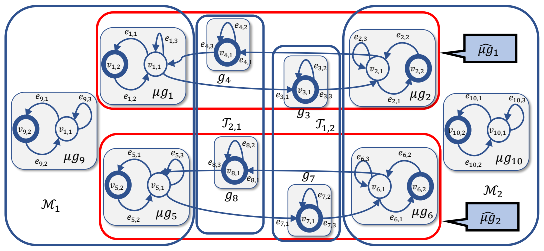


Fig. 5: System Modes

and $\chi(v_{t,1}) = T$, and those of three edges by $\mathcal{E}_t = \{e_{t,1} = (v_{i,1}, v_{t,1}, T_{t,1}), e_{t,2} = (v_{t,1}, v_{t,1}, T_{t,2}), e_{t,3} = (v_{t,1}, v_{j,1}, T_{t,3}\}$ where $T_{t,1} = T_i - E_i \wedge T_{t,2} = T_i \wedge T_{t,3} = T_i - E_i$ and $v_{i,1} \in V_i \wedge v_{j,1} \in V_j \wedge \mu g_i = (V_i, \mathcal{E}_i) \wedge \mu g_j = (V_j, \mathcal{E}_j) \wedge \mu g_i, \mu g_j \in \mathcal{G}_m$.

We also define $src(g_t) = v_{i,1}$ as the shorthand for the source vertex of the transitioning graph and $tgt(g_t) = v_{j,1}$ as the shorthand for the target vertex. Figure 4 depicts a multi-modal mixed-trust task graph $\hat{\mu g}_1 = (\mathcal{G}_1, \mathcal{R}_1)$ with $\mathcal{G}_1 = \{\mu g_1, \mu g_2\}, \mathcal{R}_1 = \{g_3, g_4\}$. Thick circles on a vertex indicates it is non-preemptive.

The mapping shown in Figure 4 allows us to calculate the E_i parameter as follows. First, we verify the schedulability of all non-preemptive nodes (corresponding to HTs) assuming the other node's (GTs) WCET is zero. This is possible because in the mixed-trust model all HTs have higher priority than GTs. That is, they are scheduled in a higher-priority band. This also allows us to calculate the response time of each HT vertex with:

$$R(v_{i,k}) = MI(v_{i,k}) + C_{i,k}, \tag{26}$$

The E_i parameter is then calculated in a similar manner to [8]:

$$E_i = D_i - R(v_{i,k}). \tag{27}$$

Finally, we verify the schedulability of the GT vertex to test if they can finish by their corresponding E_i parameter.

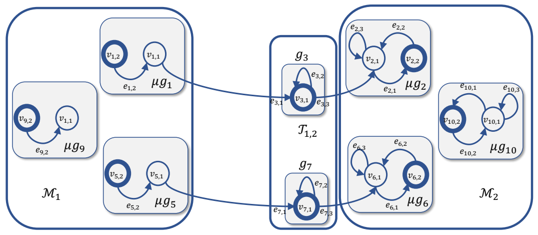


Fig. 6: System Transition

In the next section, the calculation of the E_i parameter and the system schedulability will be revisited once we introduce systems modes.

V. Modeling System Modes

In this section, we build the model that defines system modes with multiple MMMTs and systems transitions that allows these MMMTs to switch modes together. This model is a meta-graph composed of graph tasks that are enabled in different modes and graph edges and transitioning vertex that are executed during the mode transition. Specifically, we define a modal system $\mathcal{S}$ as

$$\begin{aligned}\mathcal{S} = (\mathcal{MG} = \{\hat{\mu g}_1, \hat{\mu g}_2, \ldots\}, \mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots\}, \\ \mathcal{T} = \{\mathcal{T}_{i,j}, \mathcal{T}_{r,s}, \ldots\})\end{aligned} \quad (28)$$

where, given a set of MMMTs $\mathcal{MG} = \{\hat{\mu g}_s = (\mathcal{G}_s, \mathcal{R}_s), \hat{\mu g}_t = (\mathcal{G}_t, \mathcal{R}_t), \ldots\}$, each mode $\mathcal{M}_i$ is built as a set of UMMTs as $\mathcal{M}_i = \{\mu g_k = (V_k, \mathcal{E}_k), \mu g_l = (V_l, \mathcal{E}_l), \ldots | \mu g_k \in \mathcal{G}_s, \mu g_l \in \mathcal{G}_t, \ldots\}$. Each transition $\mathcal{T}_{i,j}$ between two modes $\mathcal{M}_i$ and $\mathcal{M}_j$ is defined as the set of transitioning task graphs $\mathcal{T}_{i,j} = \{g_a = (V_a, \mathcal{E}_a), g_b = (V_b, \mathcal{E}_b), \ldots | g_a \in \mathcal{R}_s, src(g_a) \in V_k, g_b \in \mathcal{R}_t, src(g_b) \in V_l, \ldots\}$, and $\mathcal{M}_j = \{\mu g_c = (V_c, \mathcal{E}_c), \mu g_d = (V_d, \mathcal{E}_d), \ldots | \mu g_c \in \mathcal{G}_s, tgt(g_a) \in V_c, \mu g_d \in \mathcal{G}_t, tgt(g_b) \in V_d, \ldots\}$.

Figure 5 depicts an example of a system with two system modes, $\mathcal{M}_1$ and $\mathcal{M}_2$, and two transitions, $\mathcal{T}_{1,2}$ and $\mathcal{T}_{2,1}$, built with two multi-modal mixed-trust task graphs plus an additional single-modal mixed-trust task graph in each mode. The multi-modal task graphs $\hat{\mu g}_1$ and $\hat{\mu g}_2$ have identical structure to the one presented in Figure 4.

It is important to note that a mixed-trust task graph can belong to any number of modes. This allows us to classify task graphs with respect to mode transitions as follows.

Given a transition $\mathcal{T}_{s,t}$ from a mode $\mathcal{M}_s$ to a mode $\mathcal{M}_t$ we define four sets a mixed-trust task graph can belong to.

Definition 1. $\mathcal{O}_{s,t} = \{\mu g_i | \mu g_i \in \mathcal{M}_s - \mathcal{M}_t\}$. *These are the old UMMTs that belong to the source mode but do not belong to the target mode.*

Definition 2. $\mathcal{N}_{s,t} = \{\mu g_i | \mu g_i \in \mathcal{M}_t - \mathcal{M}_s\}$. *These are the new UMMTs that belong to the target mode but not to the source mode.*

Definition 3. $\mathcal{P}^p_{s,t} = \{\mu g_i | \mu g_i \in \mathcal{M}_s \cap \mathcal{M}_t\}$. *These are the persistent UMMTs that belong to both the source and the target mode.*

Definition 4. $\mathcal{C}_{s,t} = \{\hat{\mu g}_i | \mathcal{G}_i \cap \mathcal{M}_s \neq \varnothing \wedge \mathcal{G}_i \cap \mathcal{M}_t \neq \varnothing\}$ *These are the set of MMMTs that change mode.*

To verify the schedulability of a modal system we need to analyse the schedulability of each mode and of each transition. A mode is analyzed by disabling all the vertices outside of the mode and the edges that are either outside or have an end outside the mode.

To analyze a transition we need to enable both modes of a transition and disable some transitions in the source mode that are not part of the possible mode transition path. Specifically, the mode-triggering event (e.g., lidar failure) can only be discovered at the beginning of the execution of a HT as explained in Section III. This means that a task may be starting to run the GT vertex when the mode-triggering event occurs. This means that it will need to wait until it finished executing its GT and the HT arrives to discover that it needs to switch mode and starts executing the mode-transition HT. This is in fact the worst case that we evaluate and is reflected in Figure 6 where the GT self-loop edges are removed along with the edges between the GTs and the HTs (except to or from the transitioning HT).

It is worth noting that the edges of the tasks of the target mode are all enabled when the modal transition starts. This is because, it may so happen that one task may transition much faster to the target mode than another and the edges of the former may be traversed multiple times before the latter finishes transitioning to the new mode. Similarly, **new** mixed-trust tasks that do not have a counter part in the source mode are assumed to start as soon as the transition is activated.

We now formalized the task-graph set transformation required to obtain the task-graph set to be analyzed during a transition $\mathcal{T}_{s,t}$. We call this the transitioning task-graph set $\mathcal{ST}_{s.t}$. $\mathcal{ST}_{s.t}$ calculated as:

$$\mathcal{ST}_{s,t} = \overline{\mathcal{O}}_{s,t} \cup \mathcal{N}_{s,t} \cup \mathcal{P}_{s,t} \cup \overline{\mathcal{C}}_{s,t} \quad (29)$$

where

$$\begin{aligned}\overline{\mathcal{O}}_{s,t} =& \{\overline{\mu g_i} = (V_i, \overline{\mathcal{E}}_i) | \mu g_i = (V_i, \mathcal{E}_i) \in \mathcal{O}_{s,t} \\ & \wedge \overline{\mathcal{E}}_i = \mathcal{E}_i \backslash \{(v_{i,1}, v_{i,1}, T_{i,1}), (v_{i,1}, v_{i,2}, T_{i,2})\}\}\end{aligned} \quad (30)$$

$$\begin{aligned}\overline{\mathcal{C}}_{s,t} =& \{\overline{\mu g_i} = (V_i, \overline{\mathcal{E}}_i) | \mu g_i = (V_i, \mathcal{E}_i) \in \mathcal{G}_x \cap \mathcal{M}_s \\ & \wedge \hat{\mu g}_x = (\mathcal{G}_x, \mathcal{R}_x) \in \mathcal{C}_{s,t} \\ & \wedge \overline{\mathcal{E}}_i = \mathcal{E}_i \backslash \{(v_{i,1}, v_{i,1}, T_{i,1}), (v_{i,1}, v_{i,2}, T_{i,2})\}\} \\ & \cup \{\mathcal{G}_y \cap \mathcal{M}_t | \hat{\mu g}_y = (\mathcal{G}_y, \mathcal{R}_y) \in \mathcal{C}_{s,t}\}\end{aligned} \quad (31)$$

Figure 6 depicts $\mathcal{ST}_{1,2}$ of the system presented in Figure 5. For convenience we use the shorthand function $\mathcal{ST}(\mathcal{T}_{s,t})$ to obtain the transitioning task graph from a transition graph $\mathcal{T}_{s,t}$. The definitions presented in this section will be used to calculate the E_i parameter and test the general schedulability of our modal system in Sections V-A and V-B respectively.

A. Mode-Aware E Calculation

In order to calculate the E parameters we first focus on all the hypertasks during the different modes and the different transitions assuming that the guest tasks do not run. That is, first we test all the modes, then all the transitions only considering the HT. We will keep a variable $RS^q_{i,x}$ for each mode or transition q (remember a vertex may belong to multiple modes and transitions). Each of these test will proceed as follows:

1) Assume that each $v_{i,x} | \chi(v_{i,x}) = T$ corresponding to a HT has D_i to complete (including transitioning nodes if evaluating transition)
2) Assign deadline monotonic priorities to all these vertices
3) Calculate the response time $R_{i,x}$ of each vertex.

4) if $R_{i,x} \leqslant D_i$ then $RS^q_{i,x} = RS^q_{i,x} \cup \{R_{i,x}\}$ otherwise we return unschedulable.
5) Then we calculate $E_{i,x} = D_i - \max RS^q_{i,x}$.
6) Finally, we select the minimum between the $E_{i,x}$ of the vertex in the source mode representing the HT $E_{i,1}$ and the $E_{i,y}$ of the transitioning vertex (see Fig 6) to be used for both the source mode HT and the transitioning HT in order to honor the transition semantics presented in Section III.

Once the schedulability of the HT is verified and the E_i parameters are calculated then we integrate the schedulability of the system as presented next.

Algorithm 1: IsSchedulable($\mathcal{S}$)

```
$\forall RS_{i,x} \leftarrow \varnothing$ ;
for $\mathcal{M}_i \in \mathcal{M}$ do
    SetLayeredDMPrior($\{v_{k,t} \in V_k | \mu g_k = (V_k, E_k) \in \mathcal{M}_i\}$) ;
    for $v_{k,t} \in V_k | \chi(v_{k,t}) \wedge \mu g_k = (V_k, E_k) \in \mathcal{M}_i$ do
        if $R(v_{k,t}) \leqslant D_{k,t}$ then
            $RS_{k,t} \leftarrow RS_{k,t} \cup \{R(v_{k,t})\}$ ;
        else
            return FALSE ;
        end
    end
end
for $\mathcal{T}_{s,t} \in \mathcal{T}$ do
    SetLayaredDMPrior($\{v_{k,t} \in V_k | \mu g_k = (V_k, E_k) \in \mathcal{ST}_{s,t}\}$) ;
    for $v_{k,t} \in V_k | \chi(v_{k,t}) \wedge \mu g_k = (V_k, E_k) \in \mathcal{ST}_{s,t}$ do
        if $R(v_{k,t}) \leqslant D_{k,t}$ then
            $RS_{k,t} \leftarrow RS_{k,t} \cup \{R(v_{k,t})$ ;
        else
            return FALSE ;
        end
    end
end
for $v_{k,t} \in$ allvertex($\mathcal{S}$) do
    $E_{k,t} \leftarrow D_{k,t} - \max(RS_{k,t})$ ;
end
for $\hat{\mu g}_i = (\mathcal{G}_i, \mathcal{R}_i) \in \mathcal{MG}$ do
    for $v_{r,2} \in V_r, v_{t,1} \in V_t | \mu g_r = (V_r, \mathcal{E}_r) \in \mathcal{G}_i \wedge g_t =$
      $(V_t, \mathcal{E}_t) \in \mathcal{R} \wedge \mu g_r = src(g_t)$ do
        $E_{r,2} = \max(E_{r,2}, E_{t,1})$ ;
        $E_{t,1} = \max(E_{r,2}, E_{t,1})$ ;
    end
end
for $\mathcal{M}_i \in \mathcal{M}$ do
    SetLayeredDMPrior($\{v_{k,t} \in V_k | \mu g_k = (V_k, E_k) \in \mathcal{M}_i\}$) ;
    for $v_{k,t} \in V_k | \mu g_k = (V_k, E_k) \in \mathcal{M}_i$ do
        if $R(v_{k,t}) > D_{k,t}$ then
            return FALSE ;
        end
    end
end
for $\mathcal{T}_{s,t} \in \mathcal{T}$ do
    SetLayeredDMPrior($\{v_{k,t} \in V_k | \mu g_k = (V_k, E_k) \in \mathcal{ST}_{s,t}\}$) ;
    for $v_{k,t} \in V_k | \mu g_k = (V_k, E_k) \in \mathcal{ST}_{s,t}$ do
        if $R(v_{k,t}) > D_{k,t}$ then
            return FALSE ;
        end
    end
end
return TRUE ;
```

B. Modal Schedulability

We consider that a modal mixed-trust digraph system is schedulable if (1) all of its modes are schedulable and (2) all its transitions are schedulable.

Mode schedulabilty is performed applying the new mixed-trust digraph schedulability presented in Section IV while filtering the vertex that do not belong to the current mode.

Transitioning graph schedulability is performed in a similar fashion to the modes. It is worth reminding the reader that the graph will include parts of the source and target mode as presented in Figure 6. Algorithm 1 presents the overall algorithm. In this algorithm `SetLayeredDMPrio()` assigns deadline monotonic priorities in a layered fashion, with all $\chi(v_{i,j}) = T$ (HT) nodes with higher priority than $\chi(v_{r,s}) = F$ (GT) nodes.

VI. Evaluation

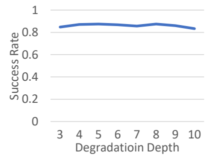
It is tempting to believe that since our algorithm that performs schedulability testing uses enumeration of paths, the algorithm would have high time-complexity and hence its running time would be so large that it cannot be used on realistically sized systems. Note, however, that we consider a run-time model where mode change must finish before a new mode change can begin. This reduces the number of possible paths to explore. Indeed, we will see, in this section, that the running time of our schedulability test is reasonable.

In order to evaluate the performance of our algorithm we conduct five experiments to measure both the schedulability success rate (percentage of tasksets successfully scheduled) and the time it takes to execute the schedulability. While there are no other mixed-trust scheduling schemes for modal systems, we wanted to provide a sense of the schedulability loss if one only takes the worst-case parameters across all modes and transitions and use the original mixed-trust scheduling [8]. This is presented in Figure 7b.

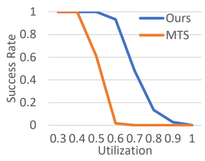
The tasksets are generated starting with the following default parameters: utilization of 70%, 10 tasks per mode[3], two modes, minimum period of 100, maximum period of 1000 and a utilization of the HT of 10% of the GT. Then these parameters are modified according to each of the experiments. The tasksets are generated by first generating at random the period of the task and then calculating their C_i and κC_i by multiplying the period by the utilization allocated to the GT and the HT respectively. Each point in the plots is the average of 1000 experiments. The analysis algorithms were implemented in Java running on Intel i7-1065G7 at 1.3GHz (max-turbo: 3.9GHz) with 32GB RAM.

Figure 7a shows the success rate as a function of the increasing degradation depth. This degradation is calculated as a random enlargement of the period in the previous mode and recalculating the computation time as explained above. As you can observe in the plot there is basically no effect of the number of degradation modes to the schedulability. However, there is an effect in the analysis execution time (as expected) which is presented in Figure 8a. This figure in fact presents a super-linear tendency.

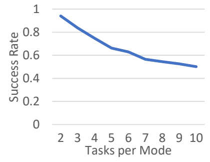
[3]This number is motivated by the number of tasks used in recent autonomous vehicle research; see [24] for example.



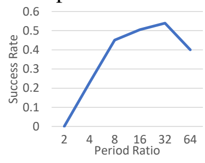
(a) Degrad. Depth



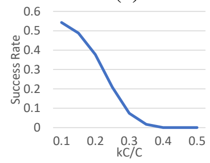
(b) Utilization



(c) Tasks Per Mode

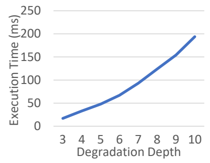


(d) Period Ratio

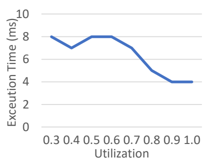


(e) HT Utilization

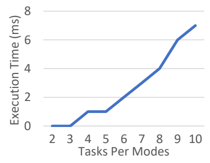
Fig. 7: Success Rate



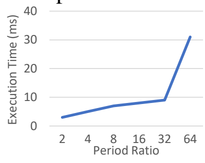
(a) Degrad. Depth



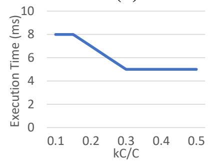
(b) Utilization



(c) Tasks Per Mode



(d) Period Ratio



(e) HT Utilization

Fig. 8: Analysis Execution Time

Figure 7b shows the success rate as a function of the increasing utilization. As expected we can see that the schedulability start to decrease between 60 and 70%. The average execution time of these experiments is presented in Figure 8b. In this case, the execution time starts decreasing as the utilization grows due to the fact that we need to traverse shorter paths to explore longer interference. The slight bump between 0.4 and 0.7 can be explained by the fact that at lower utilizations the response time over which we need to explore paths is small, then transitions into a longer paths for tasks that are still schedulable, and finally transitioning to shorter paths that lead to a large number of unschedulable tasks. As expected, the original mixed-trust scheduling scheme degrades rapidly after only 40% utilization down to zero at 60%.

Figure 7c shows the success rate as a function of the increasing number of tasks per mode. In this case the schedulability decreases a bit initially and stays at around 50%. This is a lower schedulable utilization than the regular RM but is expected due to the 10% default utlization of HTs. The average execution time of these experiments is presented in Figure 8c. In this case the execution time increases with the number of tasks per mode and because we do not see a decrease in schedulability beyond 50% even for the larger number of tasks per mode, the execution time does not decrease.

Figure 7d shows the success rate as a function of the increasing period ratio (between minimum and maximum period). We can see that when the periods are very similar there is a lot of interference that later improves with a larger ratio but decreases again due to the larger effect of the preemption of large computation of HT with long period over GT of shorter period. The average execution time of these experiments is presented in Figure 8d. In this case, even though we see a decrease in schedulability with large period ratios, the algorithm still needs to evaluate large paths to determine schedulability.

Figure 7e shows the success rate as a function of the increasing percentage of HT utilization. Clearly, as the utilization of HTs increases their preemption on GTs and other HTs (due to non-preemptive nature) decreases the utilization sharply. The average execution time of these experiments is presented in Figure 8e. In this case, we see a decrease in execution time of the analysis that matches the decrease in schedulability given that with a larger $\frac{\kappa C_i}{C_i}$ the schedulability can be resolved with much shorter paths.

VII. Implementation

We implemented our resilient mixed-trust framework on a Raspberry Pi 3-B board running Linux raspberrypi 4.4.50-v7+ and the uber-XMHF hypervisor (https://uberspark.org/). In order to preserve trust during mode changes, our implementation locates the mode change logic within the trusted HV. The modes are implemented as an array of CPU reservations keeping track of the index of the current mode. When the mode change logic in the HV decides to change modes, the new mode index is sent to the VM as the return code of the `guestjobstart()` hypercall. This hypercall is used to inform the HV when the reservation periodic activation starts. This hypercall is used to check whether or not the previous guest job has completed or is expected to continue running during the current activation. If it continues to run, the HV then will block the output to prevent the output from a job from a previous period and execute the HT to compensate for it. It is worth noting that using the `guestjobstart()` allows us to avoid the use of an upcall from the HV into the VM.

The mode-change protocol implementation is depicted in Figure 9. The figure shows a sequence diagram with the mode change messages. This sequence starts with the HV timeout. Then the mode changer (not shown separately) within the HV detects that it needs to change the mode. This then create a reprogramming of the timers with the first period timer of the new mode. Then it can either execute the HT of the new mode or the last execution of the HT of the old mode. The execution of the new or old HT is an option in the implementation that allows us to capture the need to quickly react to a (e.g., sensor) failure, or detect that the transitioning mode has completed its corrective action (e.g., has reduced the speed to 10 m/s in the Lidar failure of Section III-A) with its last execution. Then, when the kernel periodic timer elapses, and call the `guestjobstart()` hypercall returning the new mode, a mode-change signal is sent to the application before waking it up and programming the period and budget timers with the

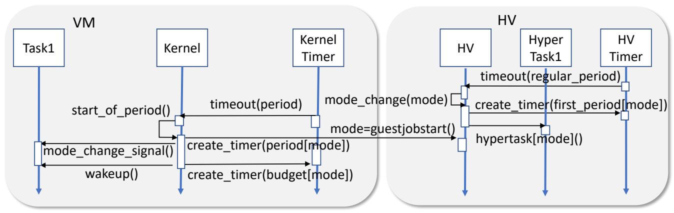


Fig. 9: Mode Change Protocol Implementation

Action	Average	Worst-Case
HV mode change	5,985	12,211
VM mode change	19,578	22,894
Hypercall	11	14

TABLE I: Mode Change Overhead Execution Time (ns)

new mode parameters.

We measured the implementation overhead in the HV and kernel module mode change routines and the hypercall to communicate the mode change. The results are presented in Table I the figure is the average and worst-case of 100 samples.

We measured the implementation overhead of (i) the `mode_change()` function in the HV, (ii) the `mode_change()` routing in the kernel module and (iii) the `guestjobstart()` hypercall that obtains the new mode.

Figure 10 shows a trace of a task reconstructed from timestamps collected during execution. The task starts with a period of 1 sec (with execution time of 10ms) repeating for four periods at zero, one, two, and three secs. Then the task switches to one period of 4 sec and then repeats the four-one pattern a second time.

VIII. Related work

In [8] de Niz et al. developed the mixed-trust computing framework which allows a component to be designed with trusted and untrusted components such that they are enforced from both logical and timing perspectives. Unfortunately, this framework neither considered degraded modes nor the expressiveness of the digraph task model.

Mixed-criticality scheduling [7] allows designers to specify different assurance levels for different tasks and also specify possible behaviors of a task with different parameters. In [10] de Niz and Phan presented a multi-modal mixed-criticality scheduling scheme on multiprocessors. Unfortunately, these works do not consider the need to protect trusted components from untrusted ones.

Stigge et al. [19] introduced the digraph task model for single-core scheduling with EDF. The authors provided a schedulability test based on an abstraction of paths, dynamic programming, and path dominance relations. In [21] Stigge and Yi presented a fixed-priority schedulability analysis for diagraph on unicore showing that the exact schedulability

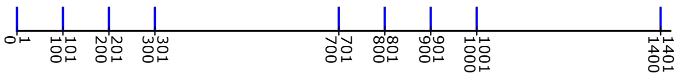


Fig. 10: Mode Switch Trace (10's of ms)

analysis is co-NP-hard. Stigge and Yi [22] generalized the request function overapproximation presented in [20] that works also for non-preemptive tasksets and for the case that the priority of a job is specified by its corresponding node. Unfortunately, this does not address mixtures of preemptive and non-preemptive jobs nor system modes. Abdullah et al. [14] studied a digraph fixed-priority scheduling on single core with mixture of preemptive and non-preemptive vertices. Unfortunately, they do not consider same-task vertices with different priorities or system modes.

Mode change for periodic and sporadic tasks on a single processor has been studied for fixed-priority preemptive scheduling [16], [15] and EDF [4]. Ekberg and Yi [11] studied EDF scheduling of digraph tasks scheduled on a single processor. They considered a model for mixed criticality and this was achieved with mode change. However, all tasks make the mode switch simultaneously by replacing old with new parameters. This solution cannot capture our needs for fixed-priority, and mixed-preemption.

Simplex [18] is an architecture comprising a complex controller, a simple controller, and two sets of states. The first set describes safe states; the second set describes when there is a need to transition between controllers. The complex controller is allowed to operate when the plant is in the second set. If the plant leaves this set, then the simple controller takes over. With this architecture, the complex controller can be optimized for performance and does not need to be verified; the simple controller, however, is verified to make sure that the plant is always in a safe state. One can think of the simple controller in Simplex as somewhat analogous to our HT. Other frameworks (e.g., [2], [3]) mitigate the impact of attackers by rebooting, assuming that attacks do not happen instantaneously, but do not protect against bugs in unverified code.

IX. Conclusions

In this paper we present a new scheduling model for resilient real-time mixed trust systems. This model extends the previous Real-Time Mixed-Trust Computing framework `RT-MTC` to be able to model degradation modes that had been required by autonomous vehicles over the years. The `RT-MTC` uses verified enforcers to monitor the output of a system and replaces it with a safe one if the output is deemed unsafe or is not produced on time. The extended model presented in this paper uses the digraph scheduling model as a base line but extends it in four critical ways: (1) it creates extensions for the mixed-preemption scheduling required by `RT-MTC`, (2) it enables priority bands in order to separate trusted and untrusted components, (3) it uses these bands to calculate intermediate deadlines used by the `RT-MTC` framework in the scheduling of the trusted components, and (4) it defines system mode semantics to obtain two desirable properties of our schedulability analysis: low pessimism and low time-complexity. We evaluated our schedulability algorithm with experiments that vary different system parameters concluding that while there is room for optimization the algorithm is reasonably efficient (e.g., even for a degradation depth of

10). This is explained by the fact that at any given time, the algorithm only needs to analyze one transition, because in our model transitions are required to be completed before another is enabled. We also discussed our implementation on Raspberry Pi. The new model presented here allows the construction of resilient autonomous systems with provable guarantees protected by verified enforcers within the `RT-MTC` framework and, more importantly, preserve these guarantees even across failure-trigger mode changes.

X. Acknowledgment

Copyright 2021 IEEE. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM21-0739

References

[1] RTCA Special Committee 205. Formal methods supplement to DO-178C and DO-278A, 2011.

[2] Fardin Abdi, Chien-Ying Chen, Monowar Hasan, Songran Liu, Sibin Mohan, and Marco Caccamo. Guaranteed physical security with restart-based design for cyber-physical systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '18, pages 10–21, Piscataway, NJ, USA, 2018. IEEE Press. URL: https://doi.org/10.1109/ICCPS.2018.00010, http://dx.doi.org/10.1109/ICCPS.2018.00010 doi:10.1109/ICCPS.2018.00010.

[3] Fardin Abdi, Rohan Tabish, Matthias Rungger, Majid Zamani, and Marco Caccamo. Application and system-level software fault tolerance through full system restarts. In *Proceedings of the 8th International Conference on Cyber-Physical Systems*, ICCPS '17, pages 197–206, New York, NY, USA, 2017. ACM. URL: http://doi.acm.org/10.1145/3055004.3055012, http://dx.doi.org/10.1145/3055004.3055012 doi:10.1145/3055004.3055012.

[4] B. Andersson. Uniprocessor EDF scheduling with mode change. In *OPODIS*, 2008.

[5] B. Andersson, S. Chaki, and D. de Niz. Combining symbolic runtime enforcers for cyber-physical systems. In *RV*, 2017.

[6] Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil Audsley, et al. Many suspensions, many problems: a review of self-suspending tasks in real-time systems. *Real-Time Systems*, 55(1):144–207, 2019.

[7] R. Davis and A. Burns. Mixed-criticality systems—a review. In *Technical Report, University of York, Available at https://www-users.cs.york.ac.uk/burns/review.pdf*, 2019.

[8] D. de Niz, B. Andersson, M. Klein, J. Lehoczky, A. Vasudevan, H. Kim, and G. Moreno. Mixed-trust computing for real-time systems. In *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–11, Aug 2019. http://dx.doi.org/10.1109/RTCSA.2019.8864566 doi:10.1109/RTCSA.2019.8864566.

[9] D. de Niz, B. Andersson, and G. Moreno. Safety enforcement for the verification of autonomous systems. In *Proceedings of SPIE*, 2018.

[10] Dionisio de Niz and Linh T. X. Phan. Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*, pages 111–122. IEEE Computer Society, 2014. URL: https://doi.org/10.1109/RTAS.2014.6925995, http://dx.doi.org/10.1109/RTAS.2014.6925995 doi:10.1109/RTAS.2014.6925995.

[11] P. Ekberg and W. Yi. Schedulability analysis of a graph-based task model formixed-criticality systems. *Real-Time Systems Journal*, 52:1–37, 2018.

[12] Daimler et al. Safety First for Automated Driving. https://www.daimler.com/documents/innovation/other/safety-first-for-automated-driving.pdf, 2019.

[13] ISO. ISO 26262 Road Vehicle Functional Safety , 2018.

[14] M. Mohaqeqi J. Abdullah, G. Dai and W. Yi. Schedulability analysis and software synthesis for graph-based task models with resource sharing. In *2018 24th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2018.

[15] A. Burns K. Tindell and A. J. Wellings. Mode changes in priority preemptively scheduled systems. In *RTSS*, 1992.

[16] J. P. Lehoczky K. Ramamritham L. Sha, R. Rajkumar. Mode change protocols for priority-driven preemptive scheduling. *Journal of Real-Time Systems*, 1989.

[17] Special C. of RTCA. DO-178C, software considerations in airborne systems and equipment certification, 2011.

[18] L. Sha. Using simplicity to control complexity. *IEEE Software*, 2001.

[19] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 71–80, 2011.

[20] M. Stigge and W. Yi. Combinatorial abstraction refinement for feasibility analysis. In *2013 IEEE 34th Real-Time Systems Symposium*, 2012.

[21] M. Stigge and W. Yi. Hardness results for static priority real-time scheduling. In *2012 24th Euromicro Conference on Real-Time Systems*, 2012.

[22] M. Stigge and W. Yi. Combinatorial abstraction refinement for feasibility analysis of static priorities. *Real-Time Systems Journal*, 51:639–674, 2015.

[23] A. Vasudevan, S. Chaki, P. Maniatis, L. Jia, and A. Datta. überspark: Enforcing verifiable object abstractions for automated compositional security analysis of a hypervisor. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[24] Bo Yu, Wei Hu, Leimeng Xu, Jie Tang, Shaoshan Liu, and Yuhao Zhu. Building the computing system for autonomous micromobility vehicles: Design constraints and architectural optimizations. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1067–1081, 2020. http://dx.doi.org/10.1109/MICRO50266.2020.00089 doi:10.1109/MICRO50266.2020.00089.