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 degradation modes. 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-
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\textsuperscript{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, $\mu_i$ is a mixed-trust task with two execution segments, $\tau_i$ and $\kappa_i$, with period $T_i$ and deadline $D_i$. The segment $\tau_i$ is considered to be untrusted and scheduled preemptively in the untrusted OS kernel inside the VM. The

\textsuperscript{1}see [5] for the formalization of this scheme.
segment $\kappa_i$ is considered to be trusted code and scheduled non-privately within the trusted HV. To represent the fact that these segments are handled by different schedulers, they are considered to be tasks and call $\tau_i$ the guest task (GT) and $\kappa_i$ the hyper task (HT), as introduced before. These tasks are defined by: $\tau_i = (T_i, E_i, \mu_i), \kappa_i = (T_i, D_i, \kappa_C)$, where $T_i$ and $D_i$ are the same as in $\mu_i$. $C_i$ is the worst-case execution time (WCET) of $\tau_i$, $\kappa_C$ is the WCET of $\kappa_i$, and $E_i$ is the intermediate deadline of $\tau_i$ determined by analysis. Consider a particular job of $\mu_i$, $\tau_i,q$. 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 $\tau_i$ 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 $\mu_i$ 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 $\kappa_i$ is calculated by first calculating the maximum duration of a level-i active period:

$$t^*_i = \max_{j \in \kappa L_i} \kappa C_j + \sum_{j \in \kappa H_i} \left( \frac{t^*_j}{T_j} \kappa C_j \right),$$

(1)

where $\kappa L_i$ is the set of all HTs with lower priority than $\kappa_i$ and $\kappa H_i$ is the set of tasks with higher priority than $\kappa_i$.

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

$$w^*_i,q = \max_{j \in \kappa L_i} \kappa C_j + (q - 1)\kappa C_i + \sum_{j \in \kappa H_i} \left( \frac{w^*_i,q}{T_j} \right) + 1)\kappa C_j,$$

(2)

Finally, the response time of $\kappa_i$ is upper-bounded by:

$$R^*_i = \max_{q \in \{1, \ldots, \frac{t^*_i}{T_i}\}} \left( w^*_i,q + \kappa C_i - (q - 1)T_i \right).$$

(3)

Given the response time of a HT, $E_i$, is calculated as: $E_i = D_i - R^*_i$ 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 $\mu_i$ as presented in equation (4).

$$rbf(y, b) = \begin{cases} \left( \frac{t - (T_i - E_i)}{T_i} \right) \kappa C_i b + \left( \frac{1}{T_i} \right) \kappa C_i & \text{if } y = E, \\ \left( \frac{1}{T_i} \right) C_i b + \left( \frac{E_i}{T_i} \right) \kappa C_i & \text{if } y = A, \end{cases}$$

(4)

where $[x]^+ = \max(0, [x])$, $y \in \{E, A\}$ indicates if the time interval of duration $t$ starts with the arrival of GT of $\mu_i$ (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:

$$t^g_i,y = \left( \sum_{j \in L_i} rbf(E_j, t^g_j, 0) \right) + rbf(y, t^g_i, 1)
\quad + \sum_{j \in H_i} \max_{y \in \{E, A\}} rbf(y, t^g_j, 1),$$

(5)

where $L_i$ and $H_i$ contain the tasks with lower and higher priority (respectively) than $\tau_i$. The latest start time of the $q^{th}$ job of a GT $\tau_i$ in the level-i busy period, $w_i,q^g$, is computed as the smallest solution of:

$$w_i,q^g = \left( \sum_{j \in L_i} rbf(E_j, w_i,q^g, 0) \right) + qC_i
\quad + (q - 1 + I_{(x=E)})\kappa C_i + \sum_{j \in H_i} \max_{y \in \{E, A\}} rbf(y, w_i,q^g, 1),$$

(6)

where $I_\phi$ is an indicator function that returns 1 if $\phi$ is true and 0 otherwise.

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

$$R_i^g,q = w_i,q^g - ((q - 1)T_i + I_{(x=E)}(T_i - E_i)).$$

(7)

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

$$R_i^g = \max_{x \in \{E, A\}} R_i^g,x.$$  

(8)

Finally the response time of a GT $\tau_i$ is given by:

$$R_i^g = \max_{x \in \{E, A\}} R_i^g,x.$$  

(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/s². 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/s² 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 \( v \leq 20 \text{m/s} \) and the “last” distance from the obstacle \( d \) before the lidar broke was \( d \geq 20 \text{m} \).

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 \( R_{\kappa,m}^\mu \) denote the response time of the HT of a mixed-trust task \( \mu \) in mode \( m \) and let \( E_{\kappa,m}^i \) denote the E-value of the GT of \( \mu \) in mode \( m \). Then we require that:

\[
E_{\kappa,m}^i \leq D_{\kappa,m}^i - R_{\kappa,m}^\mu
\]

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

\[
E_{\kappa,m,m'}^{\mu} \leq D_{\kappa,m,m'}^{\mu} - R_{\kappa,m,m'}^{\mu}.
\]

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
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 \( g_i = (V_i, E_i) \) with a set of vertices \( V_i = \{v_{i,1}, V_{i,2}, \ldots \} \) and a set of edges \( E_i = \{e_{i,1}, e_{i,2}, \ldots \} \). 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 \( (C_{i,k}, D_{i,k}) \) 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 deadlines 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 \( g_j \).

\[
rf_{\pi_j}(t) := \max \{e(\pi_j')|\pi_j' \text{ is a prefix of } \pi_j \text{ and } p(\pi_j') < t\},
\]

where

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

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 \( p(\pi) = 13 + 29 = 42 \). Note that a smaller, 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

\[
\forall \pi \in \Pi(hp(i)) : \exists t \leq D_{i,k} : C_{i,k} + \sum_{j \in hp(i)} rf_{\pi_j}(t) \leq t,
\]

where \( \Pi(g_i) \) is the set of paths of \( g_i \), \( \Pi(\Gamma) := \Pi(g_1) \times \ldots \times \Pi(g_N) \) 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 \( v_{k,k+1} \) (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 $\text{RF}^\text{sfx}(t)$) that traverses previous nodes going backwards from the node of interest. This is defined as:

$$\text{RF}^\text{sfx}_i(t) := \max\{e(\pi')|\pi' \text{ is suffix of } \pi \text{ and } p(\pi') \leq t\}. \quad (14)$$

It is worth noting that (14) uses $\leq t$ 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

$$\forall \text{RF}^{\text{sfx}} \in \text{RF}^{\text{sfx}}(g_i[\leq v_{i,q}], v_{i,q}), \exists t \leq D_{i,q} : \text{RF}^{\text{sfx}}(x) + \sum_{g_j \in \Gamma} \text{RF}^{(g_j)}(x + t) \leq x + t,$$

where this condition is quantified over all critical suffix request functions (RF$^{\text{sfx}}$) for $g_i$ which represent paths ending in $v_{i,q}$ (captured with the notation $[\leq v_{i,q}]$) 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:

$$\text{RF}^{\text{sfx}}_i(t) := \max\{e(\pi')|\pi' \text{ is prefix of } \pi \text{ and } p(\pi') \leq t\}, \quad (17)$$

and the schedulability condition is modified to

$$\forall \text{RF}^{\text{sfx}} \in \text{RF}^{\text{sfx}}(g_i, v_{i,q}), \exists t \leq D_{i,q} : B(lp(i)) + \sum_{g_j \in \Gamma} \text{RF}^{(g_j)}(x + t) \leq x + t.$$

It is 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.

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

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

where

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

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

$$MI(v_{i,k}) = P(v_{i,k}) + \sum_{g_j \in \Gamma} rf^{\text{rf}}_{v_{i,k}}(MI(v_{i,k}) + J(g_j)), \quad (22)$$

where $rf^{\text{rf}}_{v_{i,k}}$ 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:

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

$^2$critical functions are defined in [22] but not discussed here for brevity.
task is allowed to do once it starts to execute. This is defined as:

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

We next create the schedulability condition as follows.

$$ \forall \bar{\pi} \in \Pi(h(p)) : MI(v_{i,k}) \leq D_{i,k} - N(v_{i,k}). \hfill (25) $$

Note that (25) verifies that a vertex finishes before its deadline if it is preemtible 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 $\mu g_i = (V_i, E_i)$ 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). $\mu g_i$ has three edges: $E_i = \{e_{i,1} = (v_{i,1}, v_{i,2}, T_{i,1}), e_{i,2} = (v_{i,2}, v_{i,1}, T_{i,2}), e_{i,3} = (v_{i,1}, v_{i,1}, T_{i,3})\}$ 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 $\mu_i$, $E_i$ is calculated during the analysis process. We will show how this is done with (27).

Multi-Modal Mixed-Trust Task Graph (MMMT). An MMMT $\mu g_m = (G_m, R_m)$ is composed of a set of mixed-trust task graphs $G_m = \{\mu g_1, \ldots, \mu g_N\}$ connected through a set of transitioning task graphs $R_m = \{g_{N+1}, \ldots, g_{N+M}\}$. Each transitioning task graph $g_t \in R_m$ has a single vertex $V_t = \{v_{t,1}\}$ and three edges $E_t = \{e_{t,1}, e_{t,2}, e_{t,3}\}$, 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 $D_{t,1} = D_{t,2}$ and $\chi(v_{t,1}) = T$, and those of three edges by $E_t = \{e_{t,1} = (v_{t,1}, v_{t,1}, T_{t,1}), e_{t,2} = (v_{t,1}, v_{t,1}, T_{t,2}), e_{t,3} = (v_{t,1}, v_{t,1}, T_{t,3})\}$ where $T_{t,1} = T_t - E_t \wedge T_{t,2} = T_t \wedge T_{t,3} = T_t - E_t$ and $v_{t,1} \in V_i \wedge v_{t,1} \in V_j \wedge \mu g_i = (V_i, E_i) \wedge \mu g_j = (V_j, E_j) \wedge \mu g_i, \mu g_j \in G_m$. We also define $src(g_i) = v_{i,1}$ as the shorthand for the source vertex of the transitioning graph and $tgt(g_i) = v_{j,1}$ as the shorthand for the target vertex. Figure 4 depicts a multi-modal mixed-trust task graph $\mu g_1 = (G_1, R_1)$ with $G_1 = \{\mu g_1, \mu g_2\}, R_1 = \{g_3, g_4\}$. 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:

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

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

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

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

$$S = (\mathcal{MG} = \{\mu g_1, \mu g_2, \ldots\}, \mathcal{M} = \{M_1, M_2, \ldots\})$$

where, given a set of MMMTs $\mathcal{MG} = \{\mu g_s = (G_s, R_s), \mu g_t = (G_t, R_t), \ldots\}$, each mode $M_i$ is built as a set of UMMTs as $M_i = \{\mu g_k = (V_k, E_k), \mu g_l = (V_l, E_l), \ldots\}$. Each transition $T_{i,j}$ between two modes $M_i$ and $M_j$ is defined as the set of transitioning task graphs $T_{i,j} = \{g_a = (V_a, E_a), g_b = (V_b, E_b), \ldots\}$. Each transition $T_{r,s}$ to a mode $M_t$ is built as $T_{r,t} = \{g_a = (V_a, E_a), g_b = (V_b, E_b), \ldots\}$. Each mode is 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 $\mu g_1$ and $\mu g_2$ 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 $T_{s,t}$ from a mode $M_s$ to a mode $M_t$, we define four sets a mixed-trust task graph can belong to.

**Definition 1.** $O_{s,t} = \{\mu g_l | \mu g_l \in M_s - M_t\}$. These are the old UMMTs that belong to the source mode but do not belong to the target mode.

**Definition 2.** $N_{s,t} = \{\mu g_l | \mu g_l \in M_t - M_s\}$. These are the new UMMTs that belong to the target mode but not to the source mode.

**Definition 3.** $P_{s,t} = \{\mu g_l | \mu g_l \in M_s \cap M_t\}$. These are the persistent UMMTs that belong to both the source and the target mode.

**Definition 4.** $C_{s,t} = \{\mu g_l | G_l \cap M_t \neq \emptyset \land G_l \cap M_s \neq \emptyset\}$. 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 counterpart 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 $T_{s,t}$. We call this the transitioning task-graph set $ST_{s,t}$. $ST_{s,t}$ calculated as:

$$ST_{s,t} = O_{s,t} \cup N_{s,t} \cup P_{s,t} \cup C_{s,t}$$

where

$$O_{s,t} = \{\mu g_l = (V_l, E_l) | \mu g_l = (V_l, E_l) \in O_{s,t} \land T_{s,t} = E_{s,t} \setminus \{(v_{i,1}, v_{i,1}, T_{i,1}), (v_{i,1}, v_{i,2}, T_{i,2})\}\}$$

$$N_{s,t} = \{\mu g_x = (G_x, R_x) | \mu g_x = (G_x, R_x) \in N_{s,t} \land T_{s,t} = E_{s,t} \setminus \{(v_{i,1}, v_{i,1}, T_{i,1}), (v_{i,1}, v_{i,2}, T_{i,2})\}\}$$

$$P_{s,t} = \{\mu g_y = (G_y, R_y) | \mu g_y = (G_y, R_y) \in P_{s,t} \land T_{s,t} = E_{s,t} \setminus \{(v_{i,1}, v_{i,1}, T_{i,1}), (v_{i,1}, v_{i,2}, T_{i,2})\}\}$$

$$C_{s,t} = \{\mu g_z = (G_z, R_z) | \mu g_z = (G_z, R_z) \in C_{s,t} \land T_{s,t} = E_{s,t} \setminus \{(v_{i,1}, v_{i,1}, T_{i,1}), (v_{i,1}, v_{i,2}, T_{i,2})\}\}$$

Figure 6 depicts $ST_{1,2}$ of the system presented in Figure 5. For convenience we use the shorthand function $ST(T_{s,t})$ to obtain the transitioning task graph from a transition graph $T_{s,t}$. 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 $E_i$ parameters we first focus on all the hypertexts 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 $RST_{i,x}$ 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 $v_{i,x}|x(v_{i,x}) = T$ 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.
if $R_{i,x} \leq D_i$ then $RS_{i,x}^q = RS_{i,x} \cup \{R_{i,x}\}$ otherwise we return unschedulable.

5) Then we calculate $E_{i,x} = D_i - \max RS_{i,x}^q$.

6) Finally, we select the minimum between the $E_{i,x}$ of the

vertex in the source mode representing the HT $E_{s,1}$ and

the $E_{t,g}$ 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($S$)

∀$RS_{i,x} \neq \emptyset$ ;

for $M_j \in M$ do

| SetLayeredDMPrior($\{v_k,t \in V_h | \mu g_k = (V_h, E_h) \in M_j\}$) ;

for $v_k,t \in V_h | \mu g_k = (V_h, E_h) \in M_j$ do

| if $R(v_k,t) \in D_k$ then

| $RS_{k,t} = RS_{k,t} \cup \{R(v_k,t)\}$ ;

| else

| return FALSE ;

end

end

end

for $T_{s,t} \in T$ do

| SetLayeredDMPrior($\{v_k,t \in V_h | \mu g_k = (V_h, E_h) \in ST_{s,t}\}$) ;

for $v_k,t \in V_h | \mu g_k = (V_h, E_h) \in ST_{s,t}$ do

| if $R(v_k,t) \leq D_k$ then

| $RS_{k,t} = RS_{k,t} \cup \{R(v_k,t)\}$ ;

| else

| return FALSE ;

end

end

end

for $v_{k,t} \in all\imath e r t e(S)$ do

| $E_{k,t} = D_{k,t} - \max (RS_{k,t})$ ;

end

for $\mu g_i = (G_i, R_i) \in MG$ do

| for $v_r,t \in V_r, v_t \in V_t | \mu g_r = (V_r, E_r) \in G_i$ \& $gt = (V_t, E_t) \in R \land \mu g_t = src(g_t)$ do

| $E_{r,2} = \max (E_{r,2}, E_{t,2})$ ;

| $E_{t,3} = \max (E_{r,2}, E_{t,3})$ ;

end

end

for $M_j \in M$ do

| SetLayeredDMPrior($\{v_k,t \in V_h | \mu g_k = (V_h, E_h) \in M_j\}$) ;

for $v_k,t \in V_h | \mu g_k = (V_h, E_h) \in M_j$ do

| if $R(v_k,t) > D_{k,t}$ then

| return FALSE ;

end

end

end

for $T_{s,t} \in T$ do

| SetLayeredDMPrior($\{v_k,t \in V_h | \mu g_k = (V_h, E_h) \in ST_{s,t}\}$) ;

for $v_k,t \in V_h | \mu g_k = (V_h, E_h) \in ST_{s,t}$ 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 schedulability is performed applying the new mixed-

dependable 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

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 SetLayeredDMPrior() assigns

deadline monotonic priorities in a layered fashion, with all

$\chi(v_{r,t}) = T$ (HT) nodes with higher priority than $\chi(v_{r,t}) = F$

(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 sys-

tems, 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 def-

fault parameters: utilization of 70%, 10 tasks per mode3, 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 $\kappa C_i$

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.

3This number is motivated by the number of tasks used in recent au-

tonomous vehicle research; see [24] for example.
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 utilization 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 \( \frac{\kappa G}{C} \) 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
TABLE I: Mode Change Overhead Execution Time (ns)

<table>
<thead>
<tr>
<th>Action</th>
<th>Average</th>
<th>Worst-Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>HV mode change</td>
<td>5.985</td>
<td>12,211</td>
</tr>
<tr>
<td>VM mode change</td>
<td>19.578</td>
<td>22,894</td>
</tr>
<tr>
<td>Hypercall</td>
<td>11</td>
<td>14</td>
</tr>
</tbody>
</table>

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 digraph on unicore showing that the exact schedulability 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 uses 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...
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, or service by trade name, trade mark, manufacturer, or other-

REFERENCES


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, or service by trade name, trade mark, manufacturer, or other-

REFERENCES