I. INTRODUCTION

The Real-Time Mixed-Trust (RTMT) Framework [2] enables the use of untrusted components in safety-critical CPS functions (e.g., driving a car) by monitoring their actions with verified and trusted components (called enforcers) that correct unsafe actions to guarantee critical safety properties (e.g., brake to prevent a crash). The enforcers are run within a verified hypervisor that protects them from security attacks or bugs and the untrusted components are run in an unverified virtual machine (VM) on top of the hypervisor. The untrusted and trusted components are executed as a single coordinated sporadic real-time task, called a mixed-trust task, where the untrusted part is known as the guest task (GT, because it runs in the guest VM) and the trusted part running in the hypervisor (HV) is known as the hypervisor task (HT). The GT is run by a preemptive fixed-priority scheduler in the VM and the HT by a non-preemptive fixed-priority scheduler in the HV. The non-preemptive scheduler prevents interleavings and simplifies the logical verification [4], [5]. From a timing point of view, the HT monitors that the GT produces a valid output before the deadline, and if not, the HT itself produces a safe output before the deadline elapses. A new set of schedulability equations to evaluate their schedulability were presented in [2] along with a full discussion of the framework.

This paper builds on the RTMT framework and takes a step toward the ultimate goal of the development of full verification support of autonomous systems. Specifically, the paper develops support for predictive task sets. Our discussions with autonomous systems researchers have indicated a need for flexible runtime assurance mechanisms that take into account predictive computation. This paper offers a flexible mechanism from a timing point of view and formalizes the associated timing analysis.

The original RTMT framework had two drawbacks: (1) the schedulability equations assumed that the HT always runs enforcement although we expect in reality that the HT only runs in emergencies, and (2) it was assumed that the reaction to emergencies must be handled within only one invocation of the HT. To address these drawbacks, in this paper, we propose an enhancement that allows HTs to run alternative computations when they do not need to perform enforcement computations. The results of the alternative computations can be used for future enforcement when it needs to obtain a refined assessment of the situation (e.g., to verify that no obstacles are nearby and less than full braking is needed) or build a more refined reaction to create an escape trajectory to deal with a GT deadline miss. Hence, we identify such additional work as predictive computation (or pre-computation) and the HT jobs executing pre-computation as predictive jobs, which are executed only when the enforcer jobs of the HT do not execute. An example of a trajectory predictor is depicted in Figure 1. This figure captures the notional concept of some pre-computation that can be stored and used when the enforcement needs to take place.

In our model, the pre-computation jobs need to produce some amount of data to be used by the enforcer job. We encode this as $I_i$, which defines the minimum number of pre-computing jobs required in order for the enforcer to produce a better quality output. It is worth noting that safety does not depend on the value of $I_i$. However, if $I_i$ pre-computation jobs are not completed, then the improved enforcement cannot be executed and the enforcement is executed as if no pre-computation jobs occurred (all or nothing semantics). In addition, we allow the enforcer and the pre-computing jobs to have different WCETs. In the following, we will discuss how we extend the original RTMT framework to use this parameter.

II. MIXED-TRUST SCHEDULING BACKGROUND

Let us start by presenting the original RTMT system scheduling followed by the extensions for predictive enforcement.

In the RTMT framework, a system is composed of a set of mixed-trust tasks $\Gamma = \{\mu_i | \mu_i = (T_i, D_i, \tau_i, \kappa_i)\}$ running on a single-core processor. Each mixed-trust task $\mu_i$ is modeled as two execution segments, $\tau_i$ and $\kappa_i$, with period $T_i$ and
deadline $D_i$. The segment $\tau_i$ is the untrusted component and runs in the untrusted OS kernel inside the VM. The segment $\kappa_i$ is the trusted component and runs within the trusted HV in a higher priority band than all $\tau_i$’s. To represent the fact that these segments are handled by different schedulers, [2] considers them to be tasks and calls $\tau_i$ the guest task (GT) and $\kappa_i$ the hypertask (HT). 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 $\mu_i$, $C_i$ is the WCET of $\tau_i$, $\kappa C_i$ is the WCET of $\kappa_i$, and $E_i$ is the deadline for $\tau_i$.

We begin by showing the schedulability equations in [2]; this is useful because we will, later in this paper, make straightforward changes to the equations in [2] so that pre-computation jobs are included in schedulability analysis. Later on, we will also present a much better approach that is not based on straightforward changes.

The reasoning underlying the schedulability analysis in [2] is as follows. The response time of an HT $\kappa_i$ (following [1]) is obtained by first calculating the maximum duration of a level-i active period (denoted $t_\tau^i$) as the smallest solution of:

$$t_\tau^i = \max_{j \in L_i} \kappa C_j + \left\lfloor \frac{t_\tau^i T_j}{T_i} \right\rfloor \kappa C_i + \sum_{j \in H_i} \left\lfloor \frac{t_\tau^i T_j}{T_j} \right\rfloor \kappa C_j,$$

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 $\kappa_i$.

Then the latest start time of the $q^{th}$ job of $\kappa_i$ in the level-i active period ($w_{\kappa_i,q}$) is calculated as the smallest solution of:

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

and the response time of the HT $\kappa_i$ by

$$R_i^\kappa = \max_{q \in \{1, \ldots, \lfloor \frac{t_i T_i}{T_i} \rfloor \}} (w_{\kappa_i,q} + \kappa C_i - (q-1)T_i),$$

which basically adds the computation time to the latest start among all jobs in the active period.

Given this response time of an HT, the $E_i$ timer which serves as the deadline of the corresponding GT is obtained as follows:

$$E_i = D_i - R_i^\kappa.$$

To calculate the response time of a GT $\tau_i$, it is necessary to evaluate all the potential phasings of the interfering GTs (with higher priority) and HTs (all HTs except its own). To simplify this, [2] defines the request bound function (rbf) that captures the computation time of the mixed-trust task $\mu_i$ and all of the interfering tasks as in the equation below:

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

where $\lfloor x \rfloor = \max(0, \lfloor x \rfloor)$, $y \in \{E, A\}$ indicates if the interfering task is aligned with the HT ($E$) or the GT ($A$), 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 that starts with an HT job ($x = E$) or the GT job ($x = A$) of the mixed-trust task $\mu_i$:

$$R_i^\mu = \left\{ \sum_{j \in L_i} \text{rbf}_x^E(t_j^\mu, 0) + \text{rbf}_1^E(t_j^\mu, 1), \right\} + \max_{j \in H_i, y \in \{E, A\}} \text{rbf}_y^\mu(t_j^\mu, 1),$$

where $L_i$ and $H_i$ contain the tasks with lower and higher priority than $\tau_i$, respectively.

Let $w_{i,q}^g$ denote the latest finishing time of the $q^{th}$ job of the GT $\tau_i$, relative to the start of the maximum level-i busy period, such that this level-i busy period starts with a job of the HT or the GT of $\mu_i$. Then, the analysis in [2] computes $w_{i,q}^g$ as the smallest solution of:

$$w_{i,q}^g = \left\{ \sum_{j \in L_i} \text{rbf}_x^E(u_{i,q}^g, 0) + q C_i + (q-1 + B(x=E))\kappa C_i, \right\} + \max_{j \in H_i, y \in \{E, A\}} \text{rbf}_y^g(u_{i,q}^g, 1),$$

where $B_\phi$ 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 by:

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

Then the maximum response time among all the jobs in the busy period is calculated with

$$R_i^g = \max_{q \in \{1, \ldots, \lfloor \frac{t_i T_i}{T_i} \rfloor \}} \left\{ \max_{t_i} \left\{ \sum_{j \in L_i} \text{rbf}_x^E(u_{i,q}^g, 0) + q C_i + (q-1 + B(x=E))\kappa C_i, \right\} + \max_{j \in H_i, y \in \{E, A\}} \text{rbf}_y^g(u_{i,q}^g, 1) \right\}.$$

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

$$R_i^\tau = \max_{x \in \{E, A\}} R_{i,q}^g.$$

With these equations, the original RTMT framework first calculates the response time of all the HTs, 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 HT are less than or equal to their deadline and all the response times of the GT are less than or equal to their respective $E_i$.

III. PREDICTIVE MIXED-TRUST SCHEDULING

In our predictive mixed-trust model, we need to evaluate the schedulability of both the worst-case enforcement situation (that is basically the same as the original mixed-trust task) and the situation when the HTs do not need to enforce but perform some pre-computation.
A. System Model

Our new model for predictive scheduling can be captured as a modification to the real-time mixed-trust model as follows. First the $I_i$ parameter is added to the mixed-trust task to capture the number of predictive jobs; $\mu_i = (T_i, D_i, I_i, \tau_i, \kappa_i)$. Secondly, we extend the definition of the HT to include the WCETs of the enforcement ($\kappa C^p_i$) and the predictive ($\kappa C^p_i$) jobs, redefining the HT as: $\kappa_i = (T_i, D_i, \kappa C^e_i, \kappa C^p_i)$. Finally, we define $\kappa C^m_i = \max(\kappa C^e_i, \kappa C^p_i)$. It is worth noting that our model is based on concepts of the multiframe model [3] accommodating the mixed-preemption, priority bands, and intermediate deadlines required by the mixed-trust framework.

In this work in progress, we limit our focus to analyze the case where, after $I_i$ prediction/pre-computation jobs of the HT, there will be at least one enforcer job of the HT. This assumption will be lifted in our future work.

With this new system model, we need to calculate the response times of an HT in two different cases:

1) $R^e_i$ — the response time of the HT $\kappa_i$ for the case that it performs enforcement.

2) $R^p_i$ — response time of the HT $\kappa_i$ for the case that it performs prediction.

We need to compute these two versions of the response times considering that HTs are scheduled non-preemptively in the HV. Once we have computed them, we will use them to assign $E_i$ for each GT $\tau_i$, for example: $E_i = D_i - \max(R^e_i, R^p_i)$.

B. Enforcement

To calculate the response time of the HT $\kappa_i$ performing enforcement, $R^e_i$, we focus on the existing analysis [2] where the HTs are always assumed to run enforcer jobs. This is basically the same as equation (3) but replacing $\kappa C_i$ with $\kappa C^m_i$:

$$R^e_i = \max_{q \in \{1, \ldots, \frac{I_i}{T_i}\}} (w^e_{i,q} + \kappa C^m_i - (q - 1)T_i), \quad (10)$$

C. Prediction

For the case that the HT performs prediction, we need to develop new equations. In this case, the deadline encodes the time by which we need to complete the pre-computation in order to obtain the benefit of the improved enforcement.

Let us now discuss computation of the response time for the case that the HT $\kappa_i$ of a mixed-trust task $\mu_i$ performs prediction. Let $t^p_i$ denote the maximum duration of a level-$i$ active period. Following [1], we calculate $t^p_i$ as the smallest solution of:

$$t^p_i = \max_{j \in \kappa H_i} \kappa C^m_j + \left(\frac{t^p_j}{T_j}\right) \kappa C^p_j + CP(i, t^p_i) + \sum_{j \in \kappa H_i} \left(\frac{t^p_j}{T_j}\right) \kappa C^p_j + CP(j, t^p_j), \quad (11)$$

where $\kappa L_i$ is the set of HTs with lower priority than $\kappa_i$, $\kappa H_i$ is the set of HTs with higher priority than $\kappa_i$, $\kappa C^m_j = \max(\kappa C^e_j, \kappa C^p_j)$, and

$$CP(j, t^p_j) = \begin{cases} \left\lfloor \frac{t^p_j}{T_j}\right\rfloor (\kappa C^p_j - \kappa C^p_j) & \text{if } \kappa C^e_j > \kappa C^p_j \\ \left\lceil \frac{t^p_j}{T_j}\right\rceil (\kappa C^e_j - \kappa C^p_j) & \text{otherwise.} \end{cases}$$

Given that a lower-priority HT may be running when a higher-priority HT arrives, (11) takes into account the maximum interference from one job of a lower-priority task. Note that we do not make any assumption about whether the WCET of a prediction job ($\kappa C^p_j$) is larger than that of an enforcer job ($\kappa C^e_i$). Instead, (11) first accounts for the cumulative execution time considering $\kappa C^p_j$ (the second term) and then makes a compensation by using the function $CP$, which returns a positive value if $\kappa C^e_j > \kappa C^p_j$, and negative otherwise.

Let $w^p_{i,q}$ denote the latest starting time of the $q$th prediction job $\kappa^p_{i,q}$ in the level-$i$ active period. Then, from [1], we calculate $w^p_{i,q}$ as the smallest solution of:

$$w^p_{i,q} = \max_{j \in \kappa L_i} \kappa C^m_j + (q - 1)\kappa C^p_i + Q(i, q - 1) + \sum_{j \in \kappa H_i} \left(\left\lfloor \frac{w^p_{i,q}}{T_j}\right\rfloor + 1\right) \kappa C^p_j + CP_w(j, w^p_{i,q}), \quad (13)$$

with

$$Q(i, q) = \begin{cases} \left\lfloor \frac{q}{T_i}\right\rfloor (\kappa C^e_i - \kappa C^p_i) & \text{if } \kappa C^e_i > \kappa C^p_i \\ \left\lceil \frac{q}{T_i}\right\rceil (\kappa C^e_i - \kappa C^p_i) & \text{otherwise,} \end{cases}$$

and

$$CP_w(j, w^p_{i,q}) = \begin{cases} \left\lfloor \frac{w^p_{i,q}}{T_j}\right\rfloor + 1(\kappa C^e_j - \kappa C^p_j) & \text{if } \kappa C^e_j > \kappa C^p_j \\ \left\lceil \frac{w^p_{i,q}}{T_j}\right\rceil (\kappa C^e_j - \kappa C^p_j) & \text{otherwise.} \end{cases}$$

The response time can then be calculated as follows. For the jobs in the level-$i$ active period, we can move the arrival times of the $\kappa_i$ jobs to be as early as possible; this may change the schedule but neither the duration of the level-$i$ active period nor the starting time of each $\kappa_i$ job decreases. Hence, it holds that each $\kappa_{i,q}$ in the active period arrives $(q - 1)T_i$ time units after the level-$i$ active period starts.

For each job of $\kappa_i$, we can add $\kappa C_i$ to its starting time and then subtract the arrival time of this job, which yields the response time of the job. Then we upper-bound the response time of $\kappa_i$ as:

$$R^p_i = \max_{q \in \{1, \ldots, \frac{I_i}{T_i}\}} (w^p_{i,q} + \kappa C^p_i - (q - 1)T_i). \quad (16)$$

IV. GUEST-TASK SCHEDULING

It is worth remembering that safety does not depend on the GT. Hence, the GT scheduling guarantee is aimed at ensuring that the system is able to make progress (e.g., the car navigates the road and does not brake constantly) with the GT computation even if we cannot trust it with safety. Following
the same spirit, there are two ways to think about guest task scheduling: (i) considering the worst-case enforcement 
$I_i$ interarrival, i.e., we will always use $\kappa C_i$, and (ii) assume that the steady state is to have one enforcement job every $I_i$ pre-computing jobs. Since we expect that for many tasksets 
$I_i > \kappa C_i$ using a fixed $I_i$ can give us better schedulability. 

In the next two subsections we present these two sets of schedulability equations.

A. Worst HT Execution

In this case the only difference will be the replacement of 
$I_i$ by $\kappa C_i$ in all the GT schedulability equations of Section II.

B. Fixed $I_i$ parameter

This case involves modifying the interference equations as follows.

The re-definition of the request-bound function for our model is given by:

$$
\text{rbf}_i^q(t, b) = \begin{cases}
\frac{\ell - (T_i - E_i)\bigg)}{T_i} & \text{if } y = E, \\
\frac{\ell - (T_i - E_i)\bigg)}}{T_i} - \kappa C_i^p + CP(i, t) & \text{if } y = A,
\end{cases}
$$

where $[x] = \max(0, [x])$ and $(x)^+ = \max(0, x)$.

Then, similar to (5), we compute $t_i^q$ as the smallest solution of:

$$
t_i^q = \left(\sum_{j \in L_i} \text{rbf}_j^E(t_i^q, 0)\right) + \text{rbf}_i^q(t_i^q, 1)
$$

where $L_i$ and $H_i$ contain the tasks with lower and higher priority (respectively) than $\tau_i$. Given $\tau_i$ and level-$i$ busy period, we refer to job $q$ as the $q^{th}$ job with a GT arrival in the level-$i$ busy period. For each $\tau_i$, and $x \in \{E, A\}$, let $w_i^q$ denote the maximum finishing time of job $q$ of task $\tau_i$, relative to the start of the maximum level-$i$ busy period, such that this level-$i$ busy period starts with a job of the HT or the GT of $\tau_i$ arriving ($x$ indicates which). Then, similar to (6), we compute $w_i^q$ as the smallest solution of:

$$
w_i^q = \left(\sum_{j \in L_i} \text{rbf}_j^E(w_i^q, 0)\right) + q C_i + (q - 1 + B_{(x=E)})\kappa C_i^p
$$

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

For each $\tau_i$ and for each $x \in \{E, A\}$, let $R_i^q$ denote the maximum response time of job $q$ of $\tau_i$ such that this level-$i$ busy period starts with the arrival of a job of the HT or the GT of $\tau_i$ ($x$ indicates which).

V. Acknowledgements

This paper presents our work in progress on the extension of the real-time mixed-trust scheduling model to support predictive tasksets. Our discussions with autonomous systems researchers have indicated a need for flexible runtime assurance mechanisms that take into account predictive computation. This paper offers a flexible mechanism from a timing point of view and formalizes the associated timing analysis. Consequently, the paper represents an important step toward the development of full verification of autonomous systems.

VI. References


