Combined Scheduling of Time-Triggered and Priority-Based Task Sets in Ravenscar

Jorge Real, Sergio Sáez, Alfons Crespo

Universitat Politècnica de València, Spain

23rd International Conference on Reliable Software Technologies - Ada-Europe 2018 - Lisbon, Portugal, June 18-22, 2018

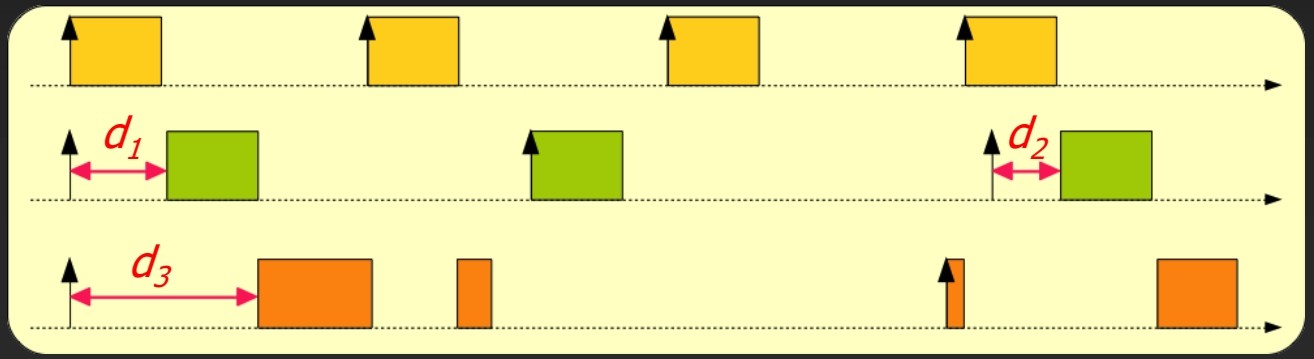
Outline

- Introduction
- System Model
- TT Scheduler
- TT Patterns
- Experimental Results
- Conclusion

Introduction

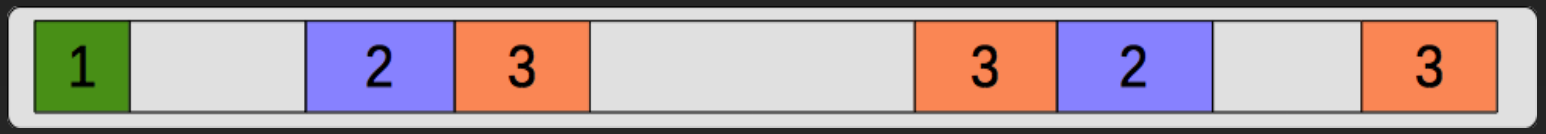
Introduction

- RT control & communication tasks need precise timing – execute close to their intended start time
- In practice, *release delays* occur
  - *Actual_Start_Time - Intended_Start_Time*
  - Degrade performance of digital controllers
  - Hinder synchronisation of communication tasks
- e.g. under Deadline Monotonic scheduling:



Introduction

- Priority-based scheduling
  - ✓ Time/logic separation, flexibility
  - ◎ Potentially large release delays (interference, blocking)
    - Techniques exist for PB scheduled systems
      - Reduce periods/deadlines to gain priority
      - Task decomposition (e.g., initial-final parts at high prio)
- Time-triggered scheduling
  - ✓ Predictable timing, release delays determined a priori



  - ◎ Building a schedule is complex (NP-complete)
    - ◎ Problem gets worse with more/longer tasks

Introduction

- Our proposal (2016) was to combine
  - a TT plan including only delay-sensitive tasks
    - Typically, control and communication tasks
  - a PB schedule for delay-tolerant tasks
    - HMI, logging, sporadics…
- Highest prio. of PB scheduler reserved for TT tasks
- Best of both worlds
  - Fewer tasks lead to simpler TT plans
  - PB benefits for the rest of tasks
    - No need for off-line plan, no task splitting
- Proposal included TT model, Ada implementation and TT task patterns

Introduction

- Our proposal (2018) is to make the approach compatible with the Ravenscar tasking profile
  - Take it closer to certifiable, embedded, HI systems, natural niches of Ravenscar and TT technology
- We have dropped soft real-time features that required non-Ravenscar mechanisms
  - Overrun *avoidance* at the task level (ATC)
  - Overrun *tolerance* with priority demotion (dynamic priorities)
- But we have also added new features

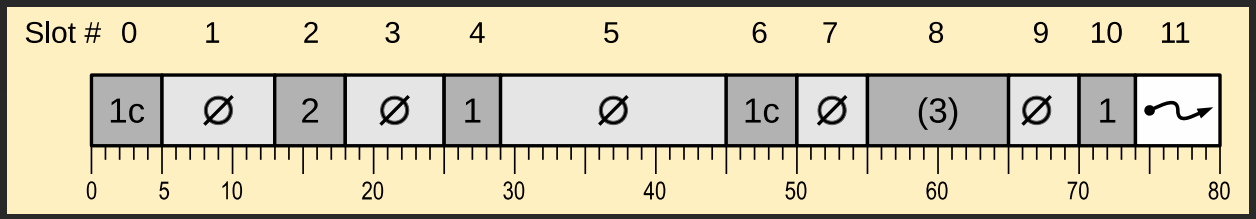
System Model

System Model

- Two task subsets
  - Delay-sensitive tasks: run according to a TT plan, using the top priority of a PB scheduler
  - Delay-tolerant tasks: PB scheduled using rest of priorities
- TT plan = ordered sequence of time slots
  - Each slot has a duration and starts right after previous
  - Sequence repeated cyclically
  - Actions during slot duration depend on slot type
    - Run one TT task, under different regimes
    - Other scheduler actions
- Schedulability: TT plan regarded as a high priority flow of tasks with offsets

System Model

- Slot types
  - Regular [1,2]
    - Run TT task *n,* with overrun check (fail → Program_Error)
  - Continuation [1c]
    - Run TT task *n,* Hold/Resume for sliced execution of long tasks
  - Optional [(3)]
    - Run TT task *n,* if the task so requires
  - Empty [∅]
    - Spare time available for PB tasks
  - Mode change [↝]
    - Process pending mode change request



TT Scheduler

TT Scheduler

- Arbitrates execution of the TT plan
  - Release TT workload at slot switch
  - Check for overruns
  - Apply Hold/Resume
- Driven by a Timing Event set to next slot switch
- Scheduler actions depend on type of slot
  - Empty slot: nothing to do – slot available for PB
  - Mode change slot: empty + enforce new plan at slot end
  - Regular, continuation and optional: apply the model

TT Scheduler

- Generic package on `Nr_Of_Work_IDs`
- Visible types for defining slots and building plans

```
type Kind_Of_Slot is (TT_Work_Slot, Empty_Slot, Mode_Change_Slot);

type Time_Slot (Kind : Kind_Of_Slot := TT_Work_Slot) is record
   Slot_Duration : Ada.Real_Time.Time_Span;
   case Kind is
      when TT_Work_Slot =>
         Work_Id         : TT_Work_Id;   -- Range 1..Nr_Of_Work_IDs
         Is_Continuation : Boolean := False;
         Is_Optional     : Boolean := False;
      when others =>
         null;
   end case;
end record;

type Time_Triggered_Plan is array (Natural range <>) of Time_Slot;
type Time_Triggered_Plan_Access is access all Time_Triggered_Plan;
```

TT Scheduler

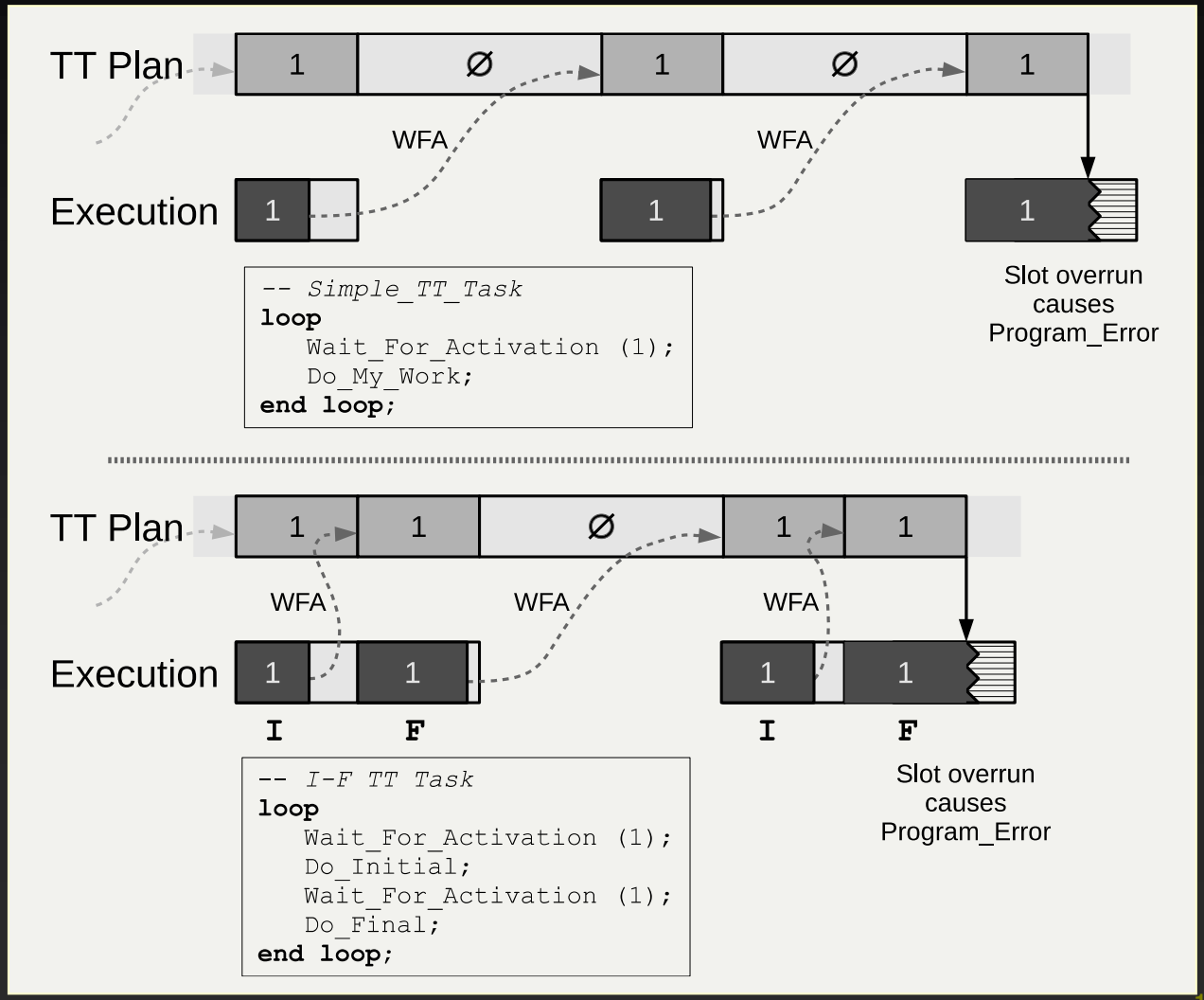
- Generic package on `Nr_Of_Work_IDs`
- Visible types for defining slots and building plans
- Visible procedures
  - `Set_Plan (TTP)` – Set the next TT plan to run
  - `Wait_For_Activation (Work_Id)` – For TT tasks to wait for own slot
  - `Continue_Sliced`, `Leave_TT_Level` (later)
- Internal data structure `Work_Control_Block`
  - `Has_Completed`, `Is_Waiting`, `Work_Thread_ID`,…
- Array of suspension objects for TT tasks to wait
- PO with TE handler

TT Patterns

TT Patterns

- Patterns using Regular slots

Simple TT Task

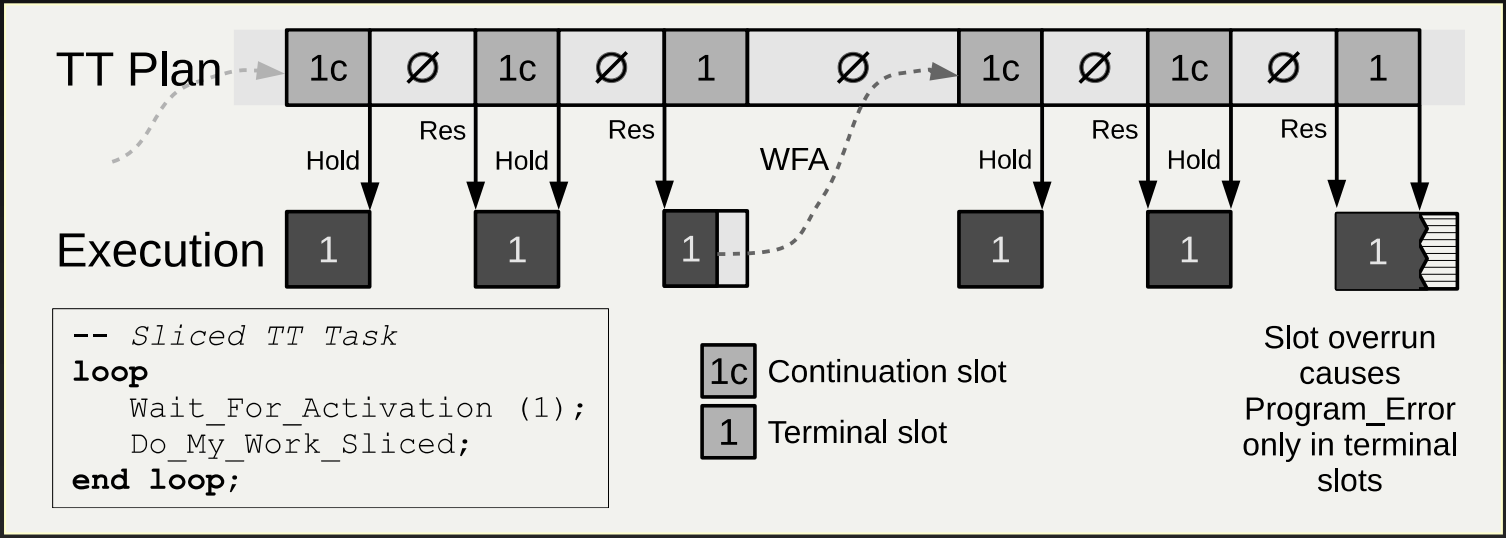


Initial - Final
I-F Task

TT Patterns

- Patterns using Continuation slots

Sliced TT Task

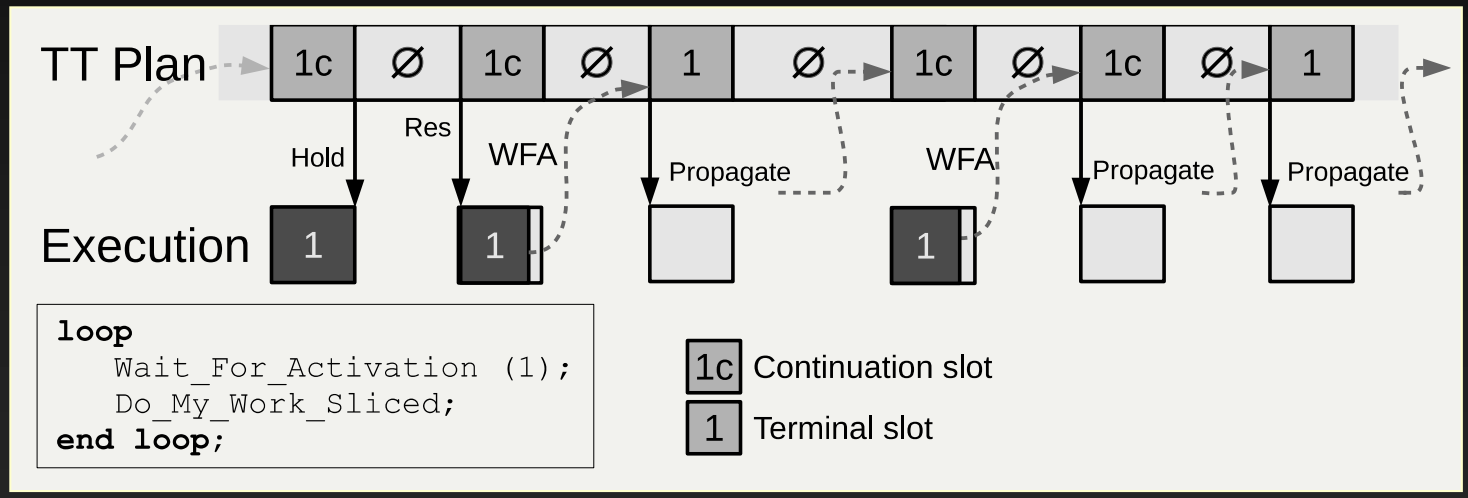


- Pattern looks like Simple TT Task, but plan is different
  - One or more cont. slots, ending with a terminal regular slot
- Hold/Resume implemented using runtime thread ops

TT Patterns

- Early completion of sliced sequence

Sliced TT Task



- `Work_Control_Block` has all needed flags for the TT scheduler to handle propagation

TT Patterns

- Hold deserves special consideration if sliced task is running a protected action (PA)
- Deferred Hold – blocking time charged to next slot
  - Use Ceiling at scheduler priority (Interrupt_Priority'Last)
    - Only acceotable with granted very short PAs (not checkable)
  - Annotate pending Hold and do it at end of PA – runtime
  - In both cases, blocking can be absorbed with empty slot
    - Need to check that empty follows continuation
    - Need to account potential interference to PB tasks
- Alternatively, forbid PAs while running sliced
  - Not trivial to detect statically
  - Can be detected at runtime (Program_Error)

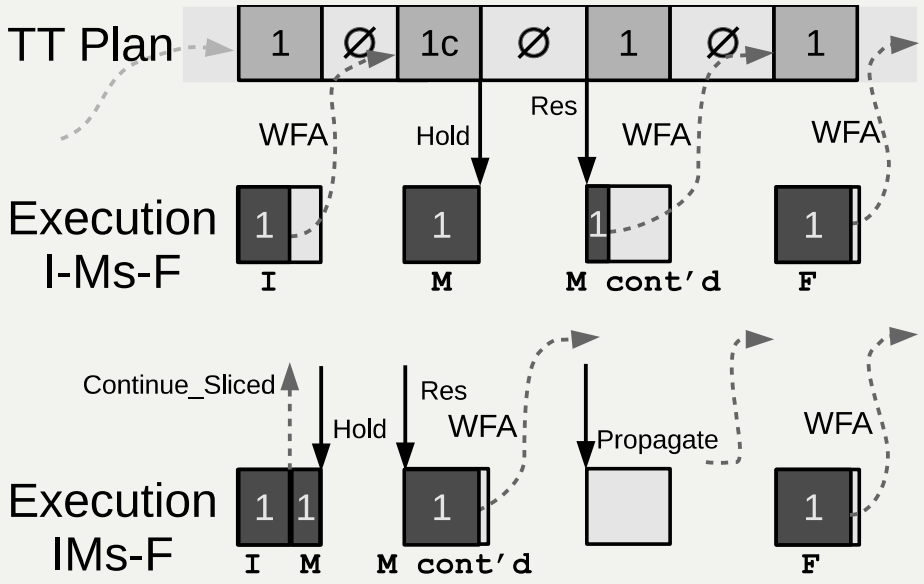
TT Patterns

- Other patterns with sliced parts and motivation for `Continue_Sliced`

I-Ms-F and IMs-F

```
loop  -- I-Ms-F Pattern
   Wait_For_Activation (1);
   Do_Initial_Part;
   Wait_For_Activation (1);
   Do_Mandatory_Sliced;
   Wait_For_Activation (1);
   Do_Final_Part;
end loop;
```

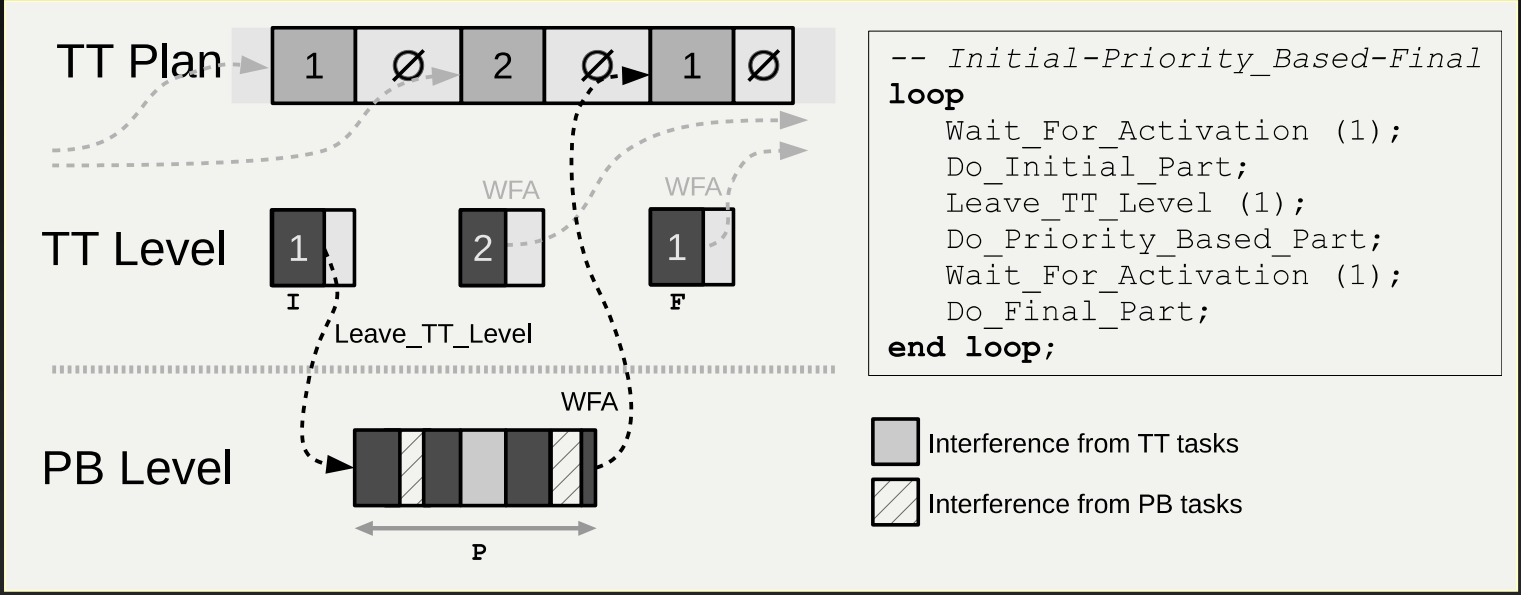
```
loop  -- IMs-F Pattern
   Wait_For_Activation (1);
   Do_Initial_Part;
   Continue_Sliced;
   Do_Mandatory_Sliced;
   Wait_For_Activation (1);
   Do_Final_Part;
end loop;
```



TT Patterns

- Patterns with non-TT parts

  Initial - Priority_Based – Final (I-P-F)

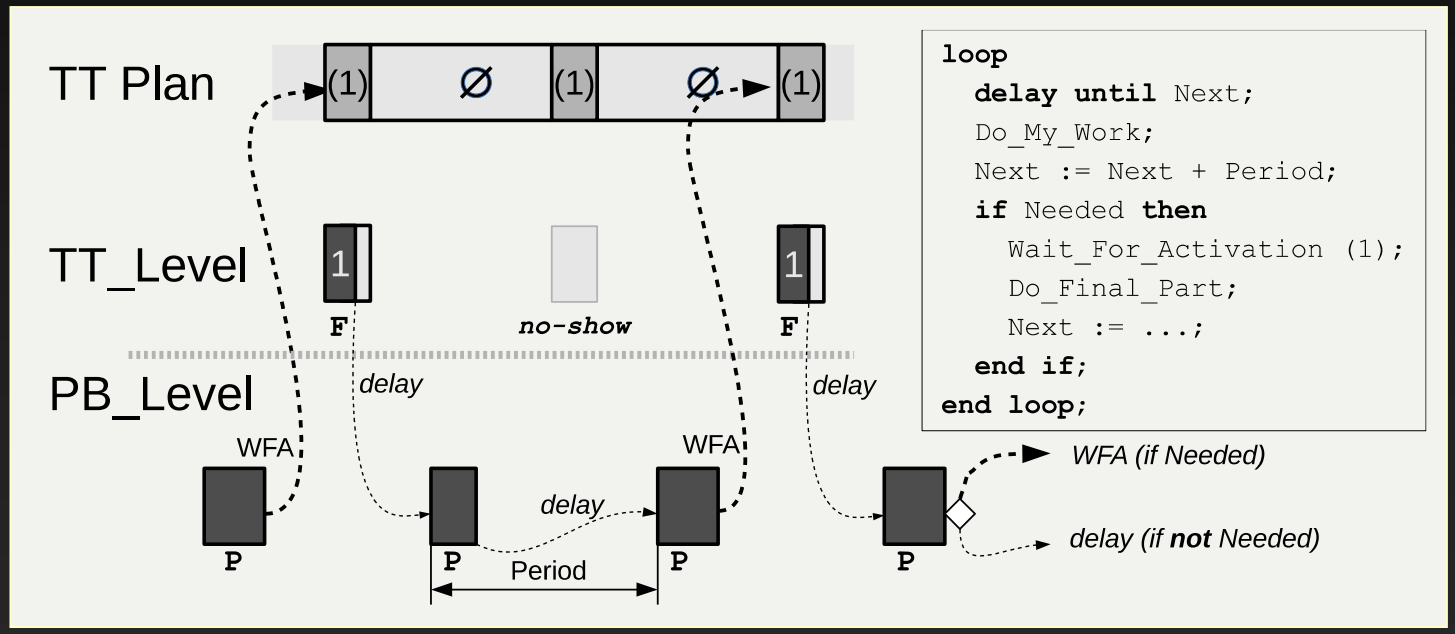


- Dynamic priorities, but in a restricted manner
  - Changes only at instance of affected task
  - Changes only between base and TT priorities
  - Conceptually, like a ceiling inherited when running the TT parts

TT Patterns

- Pattern using Optional slot

  Priority_Based - Optional_Final (P-[F])

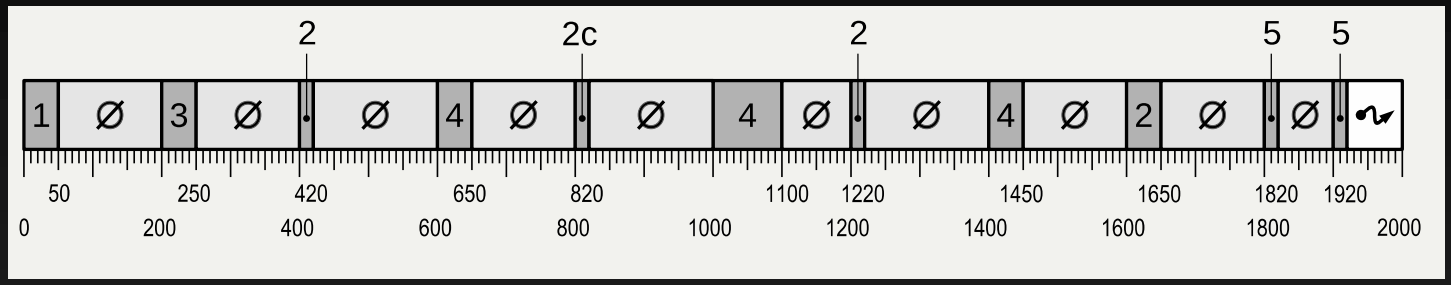


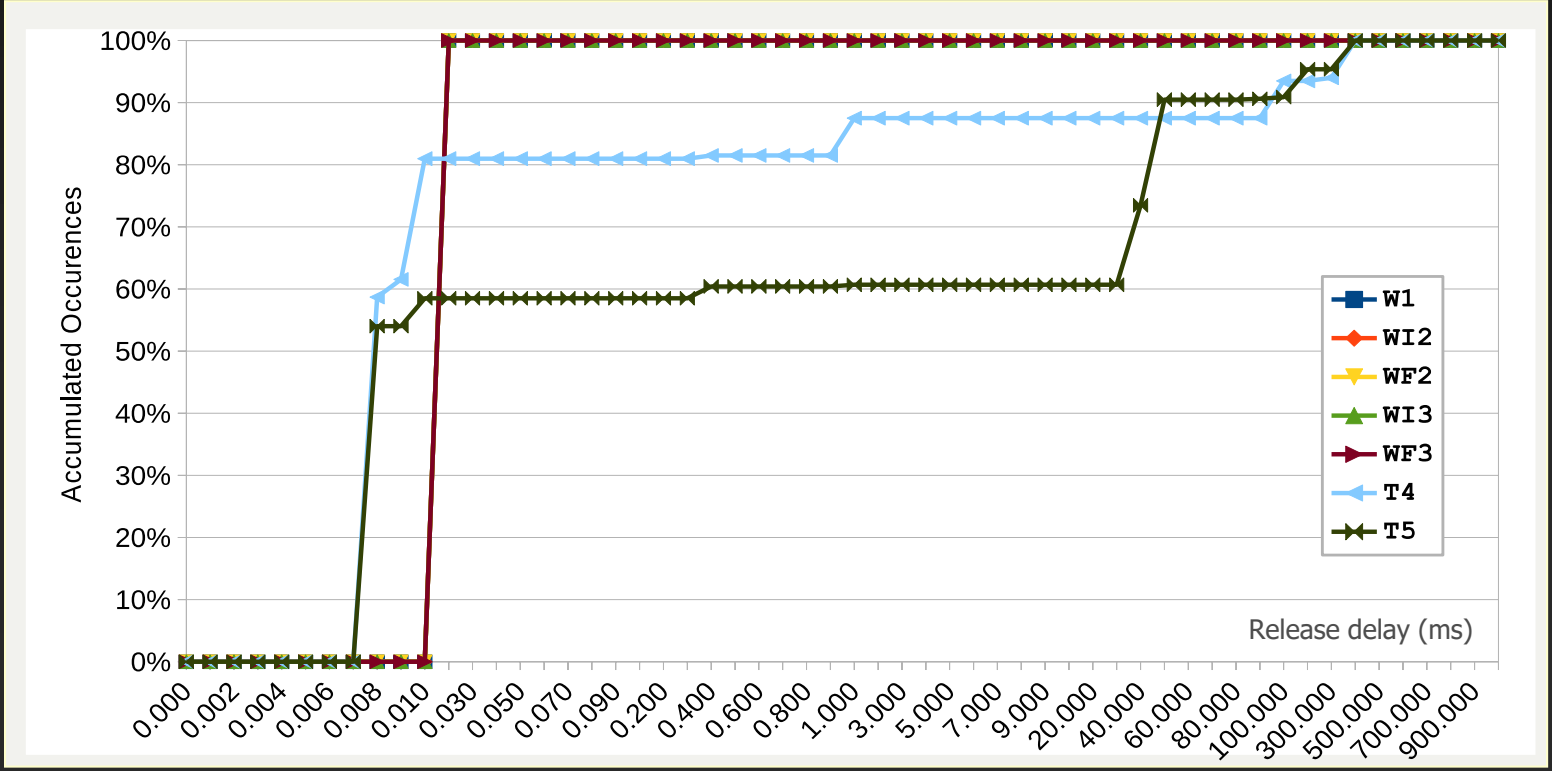
  - TT part is optional
  - "No-show" is not an error with optional slots

Experimental Results

Experimental Results

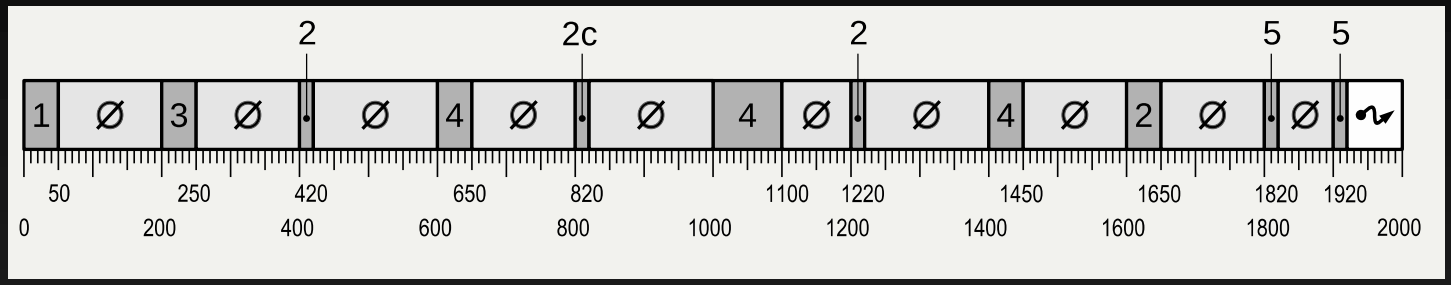
Measured release delays

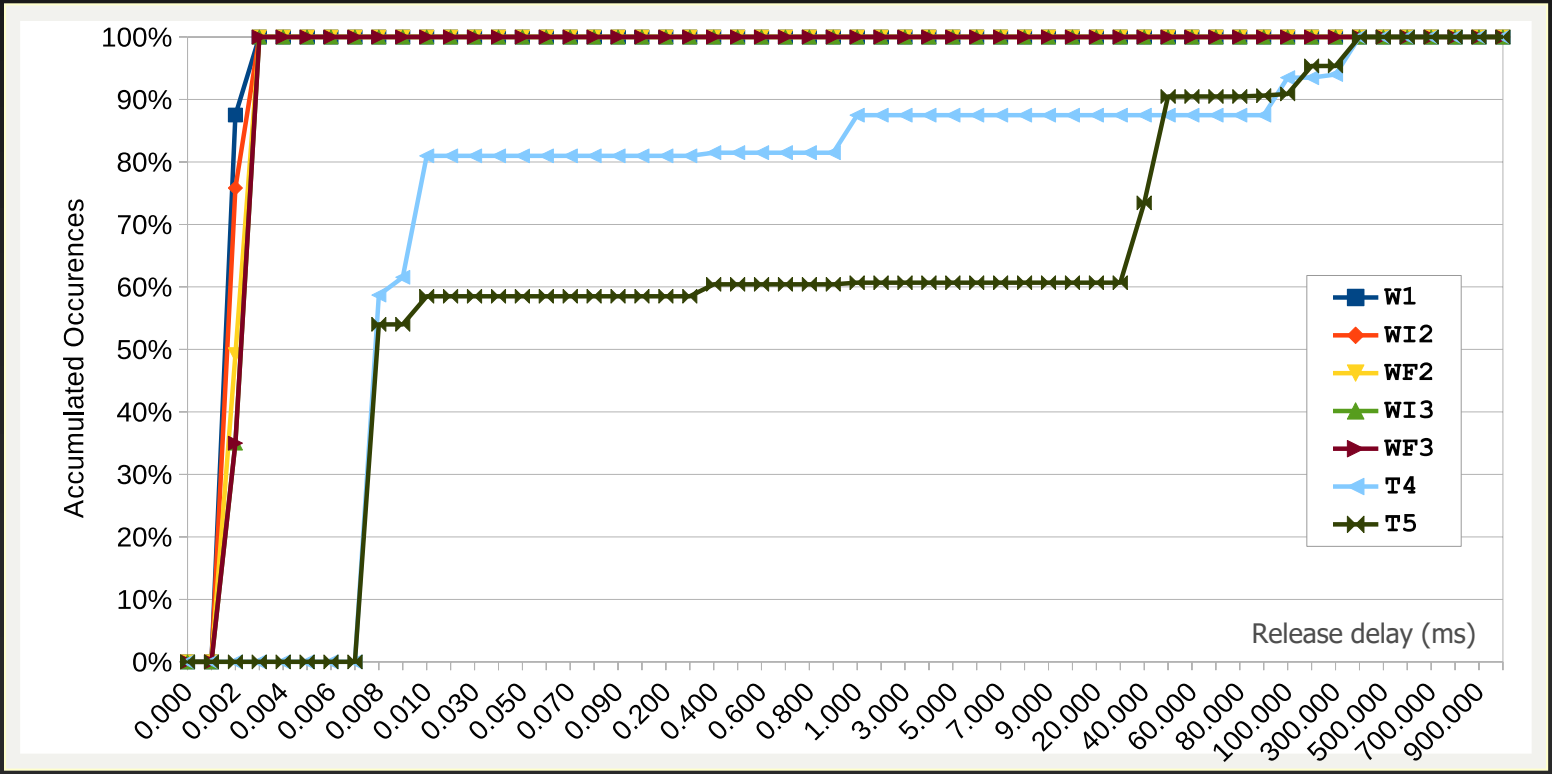




Experimental Results

Measured release delays – with overhead correction





Conclusion

- TT scheduling transported to Ravenscar
- Dropped non-Ravenscar features (ATC, dyn prio)
- Added optional and continuation slots
- Pursue standardisation
- All code available in GitHub