Self-Optimized Scheduling of Software Updates in Positive Train Control Wayside Interface Units

Matthew Jablonski

George Mason University

mjablons@gmu.edu

December 7, 2015
Overview

1. Introduction to Positive Train Control (PTC) and Wayside Interface Units (WIUs)
2. Model Predictive Control (MPC) Approach
3. Supporting Future Communication Protocols
4. Implementation
   - Controller
   - WIU Internal CM Component
5. Future Work towards Implementation and Simulation
6. Conclusion
7. References
Contributions of This Paper

The goal is to determine an efficient and automated software update process for WIUs, while remaining always available when needed by passing locomotives

- Identification of "safe" time windows for update utilizing Model Predictive Control, while remaining always available to communicate with passing locomotives
- Support for future communication protocols that the WIU may not have yet installed, but passing locomotives require
- Identify the components required for a back office controller and internal to the WIU
  - Minimization of time required for updating software components
- Discussion of Future Work required for Building and Simulation
Background on PTC

Background on WIUs in PTC

Interface with Train Management Computer (TMC):
- As train approaches, TMC broadcasts the following messages:
  - **BeaconOn**
  - **GetWIUStatus**
- WIU responds to each per spec, and configures signaling equipment

Interface with Signaling Equipment:
- WIU interfaces with a set of signaling equipment, to include signal lamps, switches, and hazard detectors
- Onboard Train Management Computer (TMC) communicates anticipated time of crossing
- WIU acts on anticipated schedule by interfacing with connected signaling equipment

Interface with Back Office:
- Back office keeps locomotive position using GPS, can provide this information
- Controller used to manage WIUs with a number of knobs and settings
Real-World Complexity

One WIUs may support a signal lamp on a set of two-way tracks, surrounded by miles of openness.

Source: https://en.wikipedia.org/wiki/North_American_railroad_signals
Real-World Complexity

Dozens of WIUs supporting signaling lamps and switching in a large rail yard.

Source: https://en.wikipedia.org/wiki/Railroad_switch
Approach to Identifying Scheduling Model

Focus is prevention of *Timing Faults* by preventing service interruption:

- WIUs are not redundant
- Background downloading and processing of information occurs OOB - does not impact service
- Classification of system states: **operational states**, **mishap states**, **hazardous states**

**High level solution:**

- Identify time windows that WIU must be available to support passing locomotives
- Identify potential time windows for software upgrade
- Make coordinated decision as block of geographically similar WIUs to conduct software update

Autonomic Computing Self-Optimization Method from [Maggio, 2012]:

- Define a system model that represents the different operational states
- Determine the best approach for making decisions; Potential approaches include:
  - **Heuristic solutions**: designed for computational performance or simplicity
  - **Standard control-based solutions**: discrete-time linear models or discrete event systems
  - **Advanced control-based solutions**: require complex model that is estimated online to provide adaptive control
  - **Model-based machine learning solutions**: define framework to learn system behavior and adjust tuning online
  - **Model-free machine learning solutions**: does not require a model of the system
- Provide an evaluation of the approach
Model Predictive Control (MPC)

- A model of the system is available and control system keeps model updated
- The controller selects the next actions based on the predictive control of future system reactions
- At each step, the controller chooses the next action to perform so that the discrepancy between the desired behavior and the forecast behavior is minimized
- A loose hypothesis on the model accuracy at each step are enough to ensure the iterative process converges and drives the system to the desired behavior
- Provides performance guarantees and reaction to unseen
- Requires a model and does not have a low overhead
- Widely used for industrial plant operations and transportation
Approach to MPC for Update Scheduling

1. Each WIU predicts arrival of passing locomotive using GPS data provided from back office to solve for the model.

2. If update available, the WIU checks each future update slot for each possible connection made by a train at a WIU in that time slot.
   1. If there are no possible connections in that time slot, the controller polls each WIU in an area to vote for the time slot to conduct a software update.
   2. Each WIU waits until the time slot is within the required Beaconing interval, and then votes based on whether or not it must be servicing Beacons during that slot.
   3. If the vote is unanimous, the back office controller schedules the update.

2. If we have exceeded the prediction horizon and have not found a time slot to update, we reduce the size of the WIU set constrained to a smaller geographic area and return to step 2.

3. When the update is complete, the success or failure is communicated to the controller and its database is updated.

Note: Software updates are thoroughly and rigorously tested prior to selection. We assume the update time slot provides time for fault detection and recovery.
Part of a railway network from [De Schutter, 2001].
Railroad Notation and Start of Model

- \( T \): period of time schedule
- \( t_{update} \): time update is available
- \( t_{complete} \): time all WIUs are updated
- \( n \): number of tracks in block
- \( W^{block} \): set of WIUs in a block
- \( W_j \): WIU at beginning of track \( j \)
- (virtual) train \( j \): physical train on track \( j \)
- \( k \): \( k^{th} \) train to pass \( WIU_j \)
- \( x_j(k) \): time instant train \( j \) departs \( WIU_j \) for the \( k^{th} \) time
- \( d_j(k) \): scheduled departure time for train \( j \)
- \( y_j(k) \): earliest time instant train \( j \) could depart \( WIU_j \) for the \( k^{th} \) time
- \( a_{ij}(k) \): travel time from \( WIU_i \) to \( WIU_j \) for each \( i \in C_j(k) \)
- \( t_{ij}^{min}(k) \): connection time for train \( i \) to stop at \( WIU_i \) before proceeding to \( WIU_j \); can equal 0 if no stop

- \( C^{block} \): set of trains in a block
- \( C_j(k) \): set of trains to which the \( k^{th} \) train on track \( j \) gives a connection
- \( C_j^{hard}(k) \): set of trains with hard connections, where the train on track \( i \) and on track \( j \) are physically the same train or if it is a very important connection that should be guaranteed at all costs
- \( C_j^{soft}(k) \): set of trains with soft connections, represents local trains to which the train \( j \) should give connection but if local train \( i \in C_j^{soft}(k) \) has too large a delay, then we wait

\[ C_j^{hard}(k) \cap C_j^{soft}(k) = \emptyset \]

\[ C_j^{hard}(k) \cup C_j^{soft}(k) = C_j(k) \]

Notation adapted from [De Schutter, 2001].
The time schedule constraint:

\[ x_j(k) \geq d_j(k) \geq y_j(k) \]

Hard synchronization constraint:

\[ x_j(k), y_j(k) \geq x_i(k - 1_{ij}^*) + a_{ij}(k) + t_{ij}^{\min}(k) \]
for each \( i \in C^\text{hard}_j(k) \)

Soft synchronization constraint:
If the connection takes place,

\[ x_j(k), y_j(k) \geq x_i(k - 1_{ij}^*) + a_{ij}(k) + t_{ij}^{\min}(k) \]

If the connection does not take place,

\[ x_j(k), y_j(k) < x_i(k - 1_{ij}^*) + a_{ij}(k) + t_{ij}^{\min}(k) \]

Note: some control variable \( u_{ij}(k) \) could be used to collapse these functions into one:

\[ x_j(k), y_j(k) \geq x_i(k - 1_{ij}^*) + a_{ij}(k) + t_{ij}^{\min}(k) - u_{ij}(k) \]
Predicting Arrival Time (Adapted from [De Schutter, 2001])

Early arrival time:

\[ y_j(k) = \min(d_j(k)), \]
\[ \max_{i \in C_{j}^{\text{hard}}(k)} (x_i(k - 1^*_{ij}) + a_{ij}(k) + t_{ij}^{\min}(k)), \]
\[ \max_{i \in C_{j}^{\text{soft}}(k)} (x_i(k - 1^*_{ij}) + a_{ij}(k) + t_{ij}^{\min}(k) - u_{ij}(k)) \]

Worst arrival time:

\[ x_j(k) = \max(d_j(k)), \]
\[ \max_{i \in C_{j}^{\text{hard}}(k)} (x_i(k - 1^*_{ij}) + a_{ij}(k) + t_{ij}^{\min}(k)), \]
\[ \max_{i \in C_{j}^{\text{soft}}(k)} (x_i(k - 1^*_{ij}) + a_{ij}(k) + t_{ij}^{\min}(k) - u_{ij}(k)) \]

Both approaches converge on actual as we get closer to present.

High-level definition of an optimization problem for finding a time slot for polling (needs refinement):

\[ \min (t_{\text{complete}} - t_{\text{update}}) \]
\[ \text{wrt } y_{ij}(k + l) \text{ (1), } x_{ij}(k + l) \text{ (2), GetUpdateTime}(W_{1-n}, t, t+1) \]
\[ \text{for all } i, j, l = 0, ..., N_p - 1, \text{ and } t = 0, ..., N_t - 1 \]
Algorithm 1: GetUpdateTime

Input: Set $C_{block}$ of trains in block
Input: Set $W_{block}$ of WIUs in block
Input: $t_{start}$ time update is posted
Input: $N_t$ time prediction horizon
Input: $N_p$ connection prediction horizon

1 slot ← −1;
2 for $t \in t_{start}$ to $t_{start} + N_t - 1$ do
3     if IsUpdateOK($C_{block}(k+l)$, $W_{block}$, $N_p$, $t$, $t+1$) == 1
4         and PollWIUs($W_{block}$, $t$) == size($W_{block}$) then
5             slot ← $t$;
6             break;
7     end
8 end
9 if slot ≠ −1 then
10     return (W block, slot);
11 end
12 return GetUpdateTime ($C_{block}$, $W_{block}$, $i \in \{1, \text{size}(W_{block})/2 - 1\}$) +
13     GetUpdateTime ($C_{block}$, $W_{block}$, $i \in \{\text{size}(W_{block})/2, n\}$);

Algorithm 2: IsUpdateOK

Input: Set $C_{block}$ of trains in block
Input: Set $W_{block}$ of WIUs in block
Input: $N_p$ connection prediction horizon
Input: $t_{start}$ time of proposed update start
Input: $t_{end}$ time of proposed update end

1 updateWIUOK ← 0; // tally for guaranteed available slot
2 updateTrainOK ← 0; // all trains verified for WIU
3 for $i \in 0$ to $N_p - 1$ do
4     updateWIUOK ← 0;
5     for $j \in W_{block}$ do
6         updateTrainOK ← 0;
7         for $i \in C_{block}$ do
8             if $i.y(j(k+l)) \neq -1$ and $i.x(j(k+l)) \neq -1$ then
9                 $y \leftarrow i.y(j(k+l));$
10                $x \leftarrow i.x(j(k+l));$
11                if ($y < t_{start}$ and ($x < t_{start}$) or ($y > t_{end}$)
12                    and ($x > t_{end}$)) then
13                    updateTrainOK ← updateTrainOK + 1;
14                else
15                    break;
16                end
17            end
18            if updateTrainOK == size($C_{block}$) then
19                updateWIUOK ← updateWIUOK + 1;
20            else
21                break;
22            end
23        end
24        if updateWIUOK ≠ size($W_{block}$) then
25            return 0;
26        end
27    end
28 return 1;
Controller Algorithm:

**Algorithm 3: PollWIUs**

**Input:** Set $W^{\text{block}}$ of WIUs in block  
**Input:** $t_{\text{start}}$ time of proposed update start

1. $\text{tally} \leftarrow 0$;  
2. for $j \in W^{\text{block}}$ do  
3. $\quad \text{tally} \leftarrow \text{tally} + j.WIUVote(t_{\text{start}})$;  
4. end
5. return $\text{tally}$;

WIU Algorithm:

**Algorithm 4: WIUVote**

**Input:** $t_{\text{start}}$ time of proposed update start

1. $t_{\text{now}} \leftarrow \text{time}()$;  
2. while $t_{\text{start}} - t_{\text{now}} < \text{BeaconEndTime}$ do  
3. $\quad \text{sleep}(1\text{ sec})$;  
4. $\quad t_{\text{now}} \leftarrow \text{time}()$;  
5. end
6. while $t_{\text{start}} - t_{\text{now}} \geq 0$ do  
7. $\quad$ if $\text{BeaconEndTime} > t_{\text{start}}$ then  
8. $\quad \quad$ return 0;  
9. $\quad$ else if $\text{BeaconTTL} == 0$ and $\text{UpdateReady} == 1$ then  
10. $\quad \quad$ return 1;  
11. end
12. $\quad$ sleep($1\text{ sec}$);  
13. $\quad t_{\text{now}} \leftarrow \text{time}()$;  
14. end
15. return 0;
**Additional Problem:** If finding an update window takes too long or occurs too quickly for the WIUs, and a locomotive traverses the rail using a different PTC communications protocol, communications in these circumstances must be handled.

**Solution:** Ajmani, Liskov, and Shrira describe Simulation Objects in [Ajmani, 2003].

From [Ajmani, 2003].
Simulation Objects (SOs)

- SOs support communication protocols in both directions (past and future) that can be chained together to support multiple versions in both directions.
- SOs are wrappers - they delegate most of their behavior to other objects.
- SOs are slower to implement than full versions, as they are simulations.
- When $O_i$ is updated to $O_{i+1}$, a mapping function $MF_{i+1}$ maps the abstract state of $O_i$ to $O_{i+1}$ and some functionality or data may be lost.
- The $MF_{i+1}$ separates $O_i$’s functionality into an independent part $I_i$ and a dependent part $D_i$.
- The $MF_{i+1}$ transition ignores $I_i$, but transitions $D_i$ to $D_{i+1}$.
- Calls to $SO_{f}^{i+1}$ that modify $D_{i+1}$ must be implemented by calling methods of $O_i$.
- Calls to $SO_{f}^{i+1}$ that use $D_{i+1}$ but cannot be implemented by calling methods of $O_i$ must fail.
- [Ajmani, 2003] also describes necessary failure states and formal correctness criteria for this method.
Components Include:

- Administrative Web Interface
- Back End
  - Software Update Management
    - Update Advertisement
    - Time Slot Search
  - Train Tracking Engine
  - Train Modeling Engine
  - Polling Engine
- Database Requirements:
  - Train Location Database
  - WIU Software Versioning Database


High-level update server design from [Ajmani, 2003].
Figure shows relationship between controller and WIUs

- WIU Upgrade Manager includes:
  - Software Update Download
  - Access to PTC beaconing status
  - Ability to conduct WIUVote
  - Fault Identification and Recovery (further research needed)

- WIU internal execution environment should be designed so that component updates are minimalized and piecemeal
  - Real-time system design best practices
  - Further research needed to minimize update time
  - Further research needed for Fault Detection and Correction
MPC Optimization and Future Work Needed

- Obvious: the algorithms presented in the MPC discussion need to be further refined into the model.
- The controller algorithm represents a "brute force" search, so we need to refine the search methodology to optimize the cost.
- Determine if it is possible to use our update controller to manage the control variable for the soft synchronization constraint.
- Refine and define a cost function based on $t_{\text{complete}}$ and variances when the scheduled departure times slip.
Additional Discussion on Future Work

- Address the issues discussed wrt. the MPC approach
- Refine the optimization problem
- Determine a more efficient way to search for \( t_{\text{complete}} - t_{\text{update}} \)
- Obtain simulation software
- Within the simulation, implement the Controller and WIU functionality
  - Evaluate the search method with voting against the search method without voting, and randomized time slot selection with voting
  - Determine if one approach is significantly better
  - Determine at what point the system is too saturated to support WIU software updates and generate a model
- Within the simulation, add different protocol versions for the WIUs and trains
  - Evaluate the ability to support future and past comm. protocols
- Identify method for minimalizing updated software components to limit downtime
- Identify method for fault detection and recovery
We believe automated software update scheduling of WIUs is a problem that is novel in research.

We presented an approach to upgrade WIUs that:

- Provides a ”starter” model to allow the software update controller to anticipate the hard and soft time constraint associated with a passing locomotive
- Provides a centralized voting method to ensure that locomotives are not geographically nearby during the proposed software update time slot
- Includes a model for upgrading the WIU’s communications protocol to support short-term requirements prior to an identified upgrade window
- Includes a design for a back-office software update controller and internal WIU controller
- Provides next steps for future research and simulation
References

Maggio, M. et. al. (2012)  
Comparison of Decision Making Strategies for Self-Optimization in Autonomic Computing Systems  

Evolving Dependable Real-Time Systems  

Model Predictive Control for Railway Networks  
*Proceedings of the 2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM’01).*

Scheduling and Simulation: How to Upgrade Distributed Systems  
*Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9.*

Interoperable Train Control Wayside Interface Unit Requirements  
*Railway Electronics AAR S-9220-0200.*