Balancing Message Criticality and Timeliness in IoT Networks

HOOTAN RASHTIAN1, SATHISH GOPALAKRISHNAN2
1Department of Electrical and Computer Engineering, The University of British Columbia
2The Peter Wall Institute for Advanced Studies and the Department of Electrical and Computer Engineering, The University of British Columbia
Corresponding author: Hootan Rashtian (e-mail: rhootan@ece.ubc.ca)

ABSTRACT We study the problem of balancing timeliness and criticality when gathering data from multiple sources using a two-level hierarchical approach. The devices that generate the data transmit them to a local hub. A central decision maker then has to decide which local hubs to allocate bandwidth to and the local hubs have to prioritize the messages they transmit when given the opportunity to do so. Whereas an optimal policy does exist for this problem such a policy would require global knowledge of messages at each local hub, rendering such a scheme impractical. We propose a distributed reinforcement-learning-based approach that accounts for both the timeliness requirements and criticality of messages. We evaluate our solution using a criticality-weighted deadline miss ratio as the performance metric. The performance analysis is done by simulating the behavior of the proposed policy as well as that of several natural policies under a wide range of system conditions. The results show that the proposed policy outperforms all the other policies – except for the optimal but impractical policy – under the range of system conditions studied and that in many cases it performs close (3% to 12% lower performance depending on the condition) to the optimal policy.

INDEX TERMS soft real-time systems, mixed-criticality streams, hierarchical scheduling, reinforcement learning, Q-learning

I. INTRODUCTION
With the proliferation of devices that can gather and transmit data about our world, we are confronted with the challenge of collecting and processing this data. More specifically, many such devices that we may deploy to observe and control aspects of our physical environment will use wireless communication links, and the available bandwidth for data transmissions can easily be saturated. To operate within these limits, we can perform a significant amount of data processing on the device that observes the data and we have to prioritize what data we choose to transmit. Some local processing is always possible, but certain decisions may require input from devices that are scattered quite widely in physical space, and therefore the data from many devices will need to be collected at some centralized (or semi-centralized) location for joint processing.

We discuss how we can prioritize and gather data from multiple devices when the data may have different timeliness and criticality requirements. In the system architecture we consider, data is collected in a two-level setup: devices may communicate with a local hub, and local hubs communicate with a global entity. Our approach uses observations of system behavior and reinforcement learning to identify suitable scheduling decisions. We present the mathematical analysis and simulation-based evaluation of scheduling policies for this problem.

A. CONTEXT
The Internet of Things is the term that is in vogue to refer to the connected system of devices (sensors and actuators) that may include a vast array of electronics-infused systems such as vehicles, home appliances, and control equipment in buildings used for heating and ventilation. The goal is to deploy applications that can identify changes in the environment or user preferences and adapt the system behavior appropriately. In the context of factory automation, we may have a critical message that indicates that a particular machine part may wear out in two hours. This message is in competition with other messages that are less critical but are essential for immediate operation. A similar situation may arise in a hospital when a network of sensors connected to a patient detects a significant event and will need to alert a nurse and, simultaneously, initiate a sequence of events to order relevant medication.

The decision-making process in a variety of applications – such as those described above – requires analyzing data
associated with timeliness requirements expressed as deadlines and semantic importance or criticality. Deadlines and criticality need not directly be related. Some data may be time-sensitive for optimal system operation (e.g., a message from a parking meter to a nearby car indicating that the spot is now free) but may not be critical. On the other hand, other data may be critical but have a more relaxed deadline (e.g., data that indicates wear-out of a machine part but the machine can operate correctly for a few days). There are, of course, situations that have associated data transmissions that are both time-sensitive and critical.

We overload the term criticality since the term has been used in the context of safety-critical systems. The notion of criticality is similar, but we are not suggesting the use of particular safety standards in this discussion.

From a system architecture perspective, when the number of communicating devices is large, a flat network is not desirable. To manage the network of devices, we consider a clustered architecture wherein a set of devices is associated with a hub or relay device. A global decision making entity communicates with the hubs to gather data. There is an element of centralization, but the hubs make tactical scheduling decisions concerning the data that needs to be forwarded to the central/global entity. The global entity has to make its own decisions about which hubs can transmit data during each decision-making epoch.

We propose and evaluate a scheduling mechanism for a two-level network architecture where messages that need to be transmitted have an associated deadline and criticality level. Each device in the system sends data to the local hub. We assume that this (local) communication is not severely constrained because one may have higher bandwidths for local communication. A hub has to assign a priority to each message. Each hub maintains a finite number of message queues, and messages can be assigned to any of these queues. These queues have fixed priorities associated with them, and data is transmitted to the global entity according to this prioritization; each hub will empty a high priority queue first before moving to a queue with lower priority data. The communication with the global entity is expensive (it may utilize, say, the cellular network) and the global decision maker will have to choose to poll local hubs based on some understanding of the value of data at the hubs.

The message scheduling problem can be solved efficiently if complete information about all the queues at each local hub is available to the central decision maker (Section III). On the other hand, the high overhead associated with gathering complete state information renders a fully centralized decision making impractical. We, therefore, propose a distributed decision making policy that is based on reinforcement learning.

B. CONTRIBUTIONS

Our contributions in this work are:

- Identifying, with proof, an optimal greedy algorithm for the hierarchical message scheduling problem (Section III).
- Development (Section IV-A and Section IV-B), and systematic quantitative evaluation (Section VI), of a reinforcement-learning-based distributed policy that is near-optimal (3% to 12% difference using the criticality-weighted deadline miss ratio).

II. SYSTEM MODEL

We consider a system with a hierarchical network architecture. At the lowest level are IoT devices such as sensors. These devices communicate with a local hub, which is a device that mediates communication with a central hub. We assume that the communication between the IoT devices and the local hub is not bandwidth-limited, and that the local hubs use any suitable multiplexing mechanism to send and receive messages from the IoT devices. Local hubs collect messages that need to be transmitted to the central hub, and need to decide on how to prioritize these messages for transmission to the central hub, which is at the top of this hierarchical system.

FIGURE 1: The hierarchical model where unit-length messages with deadlines \(d_i\) and criticalities \(\kappa_i\) arrive at local hubs and then transmitted to the central hub (based on a policy).

We consider messages of unit length (equal-sized messages). We use the following notation to describe a message and its characteristics: Message \(M_i\) has a relative deadline \(d_i\) and criticality \(\kappa_i\). If a message is created at time \(t\), then the absolute deadline for delivery of this message is \(t + d_i\).

At the local hub, messages are assigned priorities from a fixed set \(\{1, 2, \ldots, P\}\), with \(P\) representing the highest priority level.

At each scheduling epoch, the central hub selects one (or more) local hubs that can use the available bandwidth to transmit messages. A local hub will then transmit messages from its queues, starting with the highest priority queue that is not empty and moving to lower priority queues when higher priority queues are empty.

Performance Metric

In the model that we have described, the system goal is to minimize the criticality-weighted deadline miss ratio that is defined as follows. Let \(N\) be the total number of messages...
generated during the time interval of interest that also have deadlines within that time interval. Let \( x_i \) be an indicator variable for whether message \( M_i \) missed its deadline or not. The criticality-weighted deadline miss ratio is

\[
\rho := \frac{\sum_{i=1}^{N} x_i \kappa_i}{\sum_{j=1}^{N} \kappa_j}.
\]

With this performance metric, we can then state the problem that we want to solve.

**Problem Statement**

Determine a priority assignment policy for use at the local hubs and a bandwidth allocation policy at the central hub to minimize the criticality-weighted deadline miss ratio for the online scenario (we do not know what messages will arrive until they actually arrive and decisions need to be made for each new message).

### III. OPTIMAL OFFLINE POLICY

The problem of message prioritization can be solved efficiently, and optimally, when messages are all of equal length and information about all the messages is available at a central location. This offline and centralized policy is not realistic for two reasons: (i) In practice, messages arrive and need to be prioritized as they arrive, and (ii) the cost of centralizing decision making imposes a high overhead on system operation. Nevertheless, we present the scheduling algorithm for this case because the performance of this approach is an upper-bound on the performance that any online and decentralized approach can achieve. We refer to this policy as OP.

Consider an offline setting, where there is a list of messages that need to be scheduled for transmission. Message \( M_i \) has criticality value \( \kappa_i \), a relative deadline \( d_i \), and unit length. The goal is to develop a scheduling policy (or algorithm) that schedules\( \) selects messages to get processed while minimizing the metric \( \rho \) (Section II).

Formally, we can state the problem as follows:

- **Input**: \( (d_1, \kappa_1), (d_2, \kappa_2), ..., (d_n, \kappa_n) \)
- **Output**: Schedule \( S = \{S(1), S(2), \ldots, S(i), \ldots, S(n)\} \)

where \( |S| \leq n \):

\( S(i) = j \) means that the message \( M_i \) is scheduled in time slot \( j \);

Any message is scheduled at most once;

If \( x_i = 1 \) when \( M_i \) misses its deadline and \( x_i = 0 \) otherwise then \( \rho := \frac{\sum_{i=1}^{N} x_i \kappa_i}{\sum_{j=1}^{N} \kappa_j} \) is minimized.

In the offline scenario, minimizing \( \rho \) is equivalent to maximizing the criticality sum of the messages that meet their deadlines.

The proposed greedy algorithm (i.e., Algorithm 1) solves this problem optimally.

**Theorem 1.** Algorithm 1 maximizes the sum of criticalities of messages that meet their deadlines.

The proof is presented in the appendix (Appendix A).

### Algorithm 1 Optimal Greedy Policy

1: sort messages in non-increasing order of criticality values: \( \kappa_1 \geq \ldots \geq \kappa_n \)
2: \( d \leftarrow \max \{d_i\} \)
3: for \( t \leftarrow 1 \) to \( n \) do
4: \( S(t) \leftarrow 0 \)
5: \( t \leftarrow 0 \)
6: for \( i \leftarrow 1 \) to \( n \) do
7: if \( S(t) = 0 \) and \( t \leq d_i \) then
8: \( S(t) \leftarrow i \)
9: \( t \leftarrow t + 1 \)
10: else
11: skip message \( M_i \)

**Running Time**
The initial sorting can be done in \( \Theta(n \log n) \) time in the worst case, and the remaining two loops take \( \Theta(n) \) time in the worst case. Therefore, Algorithm 1 runs in \( \Theta(n \log n) \) time in the worst case.

Although this greedy algorithm is an optimal polynomial time algorithm, it requires complete and centralized information about all messages in the system. This centralization will impose a high overhead and therefore we seek decentralized solutions to the problem at hand.

### IV. USING REINFORCEMENT LEARNING IN A DECENTRALIZED POLICY

Given the two-level model that was described in Section II, the central hub and local hubs have to make decisions on message transmission. Therefore, all hubs need to have policies for making such decisions. Here we elaborate on a policy that uses reinforcement learning to achieve near-optimal performance.

**A. AT LOCAL HUBS**

For each message, the local hub has to decide which queue to place the message in. We assume that there are \( P \) priority levels \( \{1, 2, \ldots, P\} \), each of which is associated with a queue. Messages in the highest priority queue, associated with priority level \( P \), are emptied first before messages at priority level \( P - 1 \) are transmitted, and so on. The local hub needs to decide on an action \( a \in \{1, 2, \ldots, P\} \) for each message where \( a \) represents the queue a message is assigned to.

The state \( s_t \) of a local hub is the state of each of the queues at time \( t \). We will use \( n(s_t) \) to represent the number of messages queued at time \( t \). A decision made to assign a message to a queue changes the state of the system.

We can represent the value of a specific message \( M \) as a function of its deadline and criticality:

\[
v_M = \kappa_M^\beta \left( \frac{1}{d_M} \right)^\alpha.
\]
The value of a message is related to its criticality (higher the criticality higher the value) and its relative deadline (shorter the relative deadline higher the value).

\( \alpha \) and \( \beta \) are parameters that help a system architect achieve a balance between criticality and timeliness. These may be adapted on a per-local-hub basis but in this discussion all local hubs use the same settings for these two parameters.

We define the cost of taking action \( a_t \) at time \( t \) for new message \( M^* \), when the system is in state \( s_t \), as follows:

\[
c(a_t, s_t) = \frac{\sum_{M \in s_{t+1}} w(M) v(M)}{n(s_{t+1})}. \tag{3}
\]

We use \( s_{t+1} \) to indicate the state we reach after taking action \( a_t \) at state \( s_t \). \( w(M) \) is the weight associated with each message which represents the absolute priority of \( M \) at the local hub. For the highest priority message in the system (the message at the head of the highest priority queue with any message in it), \( w(M) = n(s_t) \) at state \( s_t \) and \( w(M) \) for the lowest priority message (at the end of the lowest priority queue with any message in it) is 1.

\( c(a_t, s_t) \) can be interpreted as the (weighted) average cost of a message queued at a local hub after taking action \( a_t \).

**Priority assignment.** An incoming message is assigned a priority \( p^* \) based on the immediately perceived cost and the future cost associated with the decision:

\[
p^* = \arg \min_{p \in P} E[c(s_t, a(p)) + \gamma Q(s_{t+1})]. \tag{4}
\]

\( Q(s_t) \) is the value function for the local hub in state \( s_t \) and we use \( a(p) \) to denote the action of assignment priority \( p \) to the message.

**Updating the value function.** As each local hub evolves by the arrival and priority assignment of new messages, the value function (i.e., \( Q \)) gets updated:

\[
Q(s_{t+1}) = Q(s_t) + \delta[c(s_t, a_t) + \gamma Q(s_{t+1}) - Q(s_t)], \tag{5}
\]

where \( \gamma \) is the discounting factor and \( \delta \) is the learning rate.

**B. AT THE CENTRAL HUB**

We assume that at the start of each scheduling epoch the central hub gathers a snapshot of the state of the local hubs. This snapshot, for local hub \( L \) at time \( t \), is the value \( Q(s^L_t) \) and the number of queued messages, \( n(s^L_t) \) (as explained in Section IV-A). This information is relatively small in size compared to the actual messages.

The central hub computes a weight for local hub \( L \) as follows:

\[
w_L(s_t) = Q(s^L_t)n(s^L_t) \sum_{M_i \in T_L} \kappa_i, \tag{6}
\]

where \( T_L \) is the set of messages that \( L \) can transmit in one time slot. If \( m \) messages can be transmitted in one time slot then \( T_L \) is the set of the \( m \) highest priority messages at \( L \). For the rest of this discussion, we assume that a local hub transmits only one message in a time slot but our work can be extended to situations when multiple messages can be transmitted in the same time slot.

The central hub selects the local hub with the highest weight. We call this policy at the central hub the Value-Weighted Policy (VWP).

**V. ALTERNATIVE POLICIES AT THE CENTRAL HUB**

Having described the policy, based on reinforcement learning, that we have proposed, we discuss other heuristics that we can use to compare with our proposal.

Recall that the performance goal is to reduce the criticality-weighted deadline miss ratio. To this end, we can consider four other decentralized policies. In each of these policies, the local hubs assign priorities to messages using reinforcement learning (Section IV-A) but the central hub uses a simple heuristic.

1) Deadline-Greedy (DG): The central hub then selects a local hub using the highest deadline among the highest priority messages at each local hub. Ties can be broken arbitrarily.

2) Criticality-Greedy (CG): The central hub then selects a local hub using the highest criticality among the highest priority messages at each local hub. Ties can be broken arbitrarily.

3) Criticality-Density-Greedy (CDG): We define the criticality density of message \( M_i \) as \( \frac{c_i}{p_i} \). The central hub selects the local hub that has the message of highest criticality density.

4) Random (RA): The central hub selects a local hub at random.

A fifth alternative policy that we consider, which is completely centralized, is the Global-Random (GRA) policy. We assume that the central hub has information about all messages at all the local hubs and selects a message to be transmitted at random from a uniform distribution. This policy is impractical because it is centralized but we mention it here as another policy to compare our proposed policy with.

**VI. QUANTITATIVE EVALUATION**

We evaluate the policies via simulation. We set up a system with eight local hubs and a central hub, and each local hub has three priority queues \( q = 3 \) representing low, medium, and high priority levels. We assume each queue has a capacity of \( p = 50 \) messages. We model message arrivals according to a Poisson process with rate \( \lambda = 0.15 \).

We assume that the processing time to calculate the weight \( w_i \) as shown in Section IV-A for each local hub is negligible.

**A. EXPERIMENTAL PARAMETERS**

There are four parameters involved in the system model. Two parameters are related to the value function (the \( Q \) function): the discounting factor \( (\gamma) \) and the learning rate \( (\delta) \). The other two are local hub balancing parameters for the deadline \( (\alpha) \) and the criticality \( (\beta) \).

We assume, in our evaluation, that each local hub uses the same choice of parameter values to balance timeliness and...
criticality. For all four parameters, we assigned one of three values: low (0.1), medium (0.5) and high (0.9). This resulted in $3^4 = 81$ possible settings, and we simulated each setting for 1000 time steps.

**B. EVALUATION SCENARIOS**

We have identified an offline optimal policy (Section III) that provides an upper-bound on the performance of any of the decentralized policies. We compare the proposed policy (Section IV-A) and Section IV-B with the five alternative heuristics (Section V) as well as the optimal offline policy.

To understand the performance differences, we consider five different scenarios concerning message deadlines and message criticalities:

- **Scenario I**: Majority of messages have higher criticality values and shorter deadlines.
- **Scenario II**: Majority of messages have high criticality values. On the other hand, those messages with lower criticality values have shorter deadlines.
- **Scenario III**: Majority of the messages have low criticality values. On the other hand, those messages with high criticality values have shorter deadlines.
- **Scenario IV**: Majority of the messages have low criticality values. On the other hand, those messages with low criticality values have shorter deadlines.
- **Scenario V**: For each message, the deadline and criticality values are chosen from a uniform distribution. Therefore, we chose deadlines of messages from the uniform distribution $U_d(1, 50)$, and criticality values from the uniform distribution $U_c(1, 25)$.

In these scenarios, we assumed $d_{short} = 25$ as the shorter deadline, and $d_{long} = 100$ as the longer deadline. Similarly, we considered $k_{low} = 1$, and $k_{high} = 2$ as the lower and higher criticality values, respectively. In Scenarios I to IV, we used a triangle distribution to generate 75% and 25% of messages from the specified majority and minority groups, respectively.

**C. RESULTS**

Our observations, based on the results from the simulations (Figure 3 and Table 1), can be broken down across the five scenarios we considered. In each scenario and for each policy, the aggregate results of the 81 settings is considered for the evaluations. Across all scenarios, we find that the proposed policy performs competitively when compared with the optimal performance and outperforms all other policies. The proposed approach offers within 88% and 97% of the upper-bound on performance.

We note that in Scenario II, the GRA and RA policies that pick local hubs uniformly at random perform relatively well because a majority of messages are of high criticality and with long deadlines, which means that selecting messages at random also leads to reasonable performance.

In Scenario III, CDG has similar performance to our proposed policy. This result may reflect the fact that most low criticality messages have a long deadline so it may be useful to prioritize messages with a high criticality-density.

In Scenario IV, our proposed policy is significantly better than most other policies because there are more low criticality messages with short deadlines and reinforcement learning adapts to this scenario but other heuristics do not.

**VII. RELATED WORK**

Our research brings multiple areas such as Reinforcement Learning (RL), performance analysis, and job scheduling together. Therefore, the related work section is mostly about the area of scheduling and its relation to RL techniques, especially in performance analysis of distributed systems applications.

In distributed environments such as grid and cloud, job scheduling is considered an NP-hard problem given the problem setup [14]. Therefore, the optimization targets are around metrics such as makespan and load balance.

Lucas-Estan and Gozalvez have investigated load balancing for industrial IoT networks, but without specific attention to messages with deadlines and criticalities [6]. Maguluri et al. [7] assumed to have unknown job sizes with the same criticality levels and optimized throughput using load balancing/scheduling algorithm. Shi et al. [11] developed an algorithm for provisioning and scheduling jobs under deadline constraints to address unpredictable issues in large-scale scientific computing. Seno et al. have developed soft real-time scheduling algorithms for industrial wireless networks with timeliness constraints alone [10]. Our work and these efforts relate to timing constraints but we focus on striking criticality and overall timeliness in a distributed IoT-like setup.

Tordsson et al. proposed a scheduling and resource allocation method for cloud jobs based on particle swarm optimization, taking the scheduling deadline, and scheduling budget into account [13]. Classic heuristic optimization techniques...
FIGURE 3: Range of criticality-weighted miss ratio ($\rho$) values for different evaluation scenarios. The box plots indicate the range of miss ratio values using the same data reported in Table IV. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy.
such as parallel genetic algorithms have been often employed to solve the job scheduling problems in cloud computing [7] and particle swarm optimization algorithm [8, 15]. Although there has been a large amount of research done regarding job scheduling in the distributed computing environment such work does not directly apply to the message scheduling problem in hierarchical networks.

The performance analysis of infrastructure-as-a-service (IaaS) cloud platforms has been studied extensively under various configurations and use cases. Although the environment that we target is different regarding resource constraints, we share similarities such as preserving service level agreement (SLA) that we refer to as criticality-weighted miss ratio (Section II). For instance, Salah et al. proposed an analytic model based on Markov chains to predict the number of cloud instances or VMs needed to satisfy a given SLA performance requirement such as response time, throughput, or request loss probability [9]. Khazaei et al. proposed a general analytic model for end-to-end performance analysis of a cloud service [5]. They illustrated their approach using IaaS cloud with three pools of servers: hot, warm, and cold, using service availability and provisioning response delays as the key QoS metrics. The proposed approach reduces the complexity of performance analysis of cloud centers by dividing the overall model into sub-models and then obtaining the overall solution by iteration over individual sub-model solutions. Our work in this paper is similar to that of Khazaei et al. [5] in the sense of having a hierarchy such that local hubs partially handle prioritization of messages.

Reinforcement learning has been used for scheduling for several applications of distributed computing. Peng et al. proposed an effective RL-based scheduler scheme for cloud computing under SLA constraint. Quan et al. [8] developed a two-layered RL method offload task. In this method, the first layer is in charge of selecting the appropriate cluster of machines, and the second layer selects a physical machine to execute the task. Chang et al. [3] tackled the data forwarding problem in Under Water Wireless Sensor Networks (UWSN) using an RL-based method that factors in the challenge of timeliness and energy constraints.

The model we have studied is different from the work on mixed-criticality scheduling that has been studied in the context of traditional real-time systems [1]. In the mixed-criticality real-time scheduling problems, the goal is to ensure that high criticality jobs always meet their deadline and the criticality-weighted deadline miss ratio is not of the metric of importance.

To the best of our knowledge, we believe that ours is the first attempt at applying reinforcement learning to the soft real-time scheduling problem with message deadlines and criticalities, and with the goal of minimizing the criticality-weighted deadline miss ratio.

### VIII. CONCLUSIONS AND FUTURE WORK

We have demonstrated that a reinforcement learning approach to message scheduling in a hierarchical system with many nodes can help us strike a balance between timeliness and criticality. The approach we have proposed outperforms many other heuristics that one could use in this context. We believe that these ideas can be of value in the context of the Internet of Things, with particular relevance to factory automation and medical systems.

We have tackled this problem in the specific setting where all messages are of equal (or unit) length. **We have shown that our solution has performance that is near-optimal by determining an upper-bound using an optimal offline algorithm.** The optimal offline algorithm for the problem we studied is a greedy centralized algorithm, which is not practical because the centralization will come with significant overhead.

In the near future, we would like to understand the problem when messages can be of varying lengths. This change to the problem is significant because the offline problem (when we know all messages ahead of time) is NP-Hard and can be solved by a pseudo-polynomial time dynamic program formulation.

### IX. ACKNOWLEDGEMENT

This work was supported by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) as well as a research grant from the Peter Wall Institute for Advanced Studies at The University of British Columbia.

### REFERENCES


### TABLE 1: The table shows the average Missed Criticality Ratio for the policies in each of the scenarios. As shown in green colored cells, the proposed policy (VWP) consistently performs as the second best policy after the optimal policy.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>OP</th>
<th>VWP</th>
<th>CDG</th>
<th>DG</th>
<th>CG</th>
<th>RA</th>
<th>GRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario I</td>
<td>0.1046</td>
<td>0.2250</td>
<td>0.2679</td>
<td>0.7699</td>
<td>0.6840</td>
<td>0.5762</td>
<td>0.3239</td>
</tr>
<tr>
<td>Scenario II</td>
<td>0.1148</td>
<td>0.1475</td>
<td>0.1810</td>
<td>0.7004</td>
<td>0.8452</td>
<td>0.2972</td>
<td>0.1879</td>
</tr>
<tr>
<td>Scenario III</td>
<td>0.1460</td>
<td>0.2263</td>
<td>0.2467</td>
<td>0.6505</td>
<td>0.8094</td>
<td>0.4079</td>
<td>0.2938</td>
</tr>
<tr>
<td>Scenario IV</td>
<td>0.1443</td>
<td>0.1865</td>
<td>0.2559</td>
<td>0.8029</td>
<td>0.5943</td>
<td>0.4820</td>
<td>0.2680</td>
</tr>
<tr>
<td>Scenario V</td>
<td>0.0413</td>
<td>0.1488</td>
<td>0.2565</td>
<td>0.2171</td>
<td>0.6932</td>
<td>0.3721</td>
<td>0.2725</td>
</tr>
</tbody>
</table>
This lemma implies that the result of the Greedy algorithm is optimal. This suggestion is because \( P(n) \) tells us that the result of Greedy can be extended to an optimal schedule using only messages from \( \emptyset \). Therefore the result of Greedy must be an optimal schedule.

**Proof.** We use induction for the proof as follows:

- **Base Case:** To see that \( P(0) \) holds, consider any optimal schedule \( S_{\text{opt}} \). Clearly, \( S_{\text{opt}} \) extends the empty schedule using only messages from \( \{1, \ldots, n\} \). Let \( 0 \leq i < n \) and assume \( P(i) \) holds. We want to show \( P(i+1) \). By assumption, \( S_i \) can be extended to some optimal schedule \( S_{\text{opt}} \) using only messages from \( \{i+1, \ldots, n\} \).

- **Induction Step:** Suppose that \( S \) is promising, and let \( S_{\text{opt}} \) be some optimal schedule that extends \( S \). Let \( S_{i+1} \) be the result of one more iteration through the loop where message \( M_{i+1} \) is considered. We must prove that \( S_{i+1} \) continues to be promising, and therefore the goal is to show there is an optimal schedule solution that extends \( S_{i+1} \). Hence, we consider the following two cases:

  - **Case 1:** Message \( M_{i+1} \) cannot be scheduled, so \( S_{i+1} = S_i \). Since \( S_{\text{opt}} \) extends \( S_i \), we know that \( S_{\text{opt}} \) does not schedule message \( M_{i+1} \). Therefore, \( S_{\text{opt}} \) extends \( S_{i+1} \) using only messages from \( \{i+2, \ldots, n\} \).

  - **Case 2:** Message \( M_{i+1} \) is scheduled by the algorithm, say at time \( t_0 \) (so \( S_{i+1}(t_0) = i+1 \) and \( t_0 \) is the latest free slot in \( S_i \) that is \( \leq d_{i+1} \)).

* **Case 2-I:** Message \( M_{i+1} \) occurs in \( S_{\text{opt}} \) at some time \( t_1 \) (where \( t_1 \) may or may not be equal to \( t_0 \)). Then \( t_1 \leq t_0 \) (because \( S_{\text{opt}} \) extends \( S_i \) and \( t_0 \) is as large as possible) and \( S_{\text{opt}}(t_1) = i+1 = S_{i+1}(t_0) \). If \( t_0 = t_1 \), we are finished with this case, since \( S_{\text{opt}} \) extends \( S_{i+1} \) using only messages from \( \{i+2, \ldots, n\} \). Otherwise, we have \( t_1 < t_0 \). Say that \( S_{\text{opt}}(t_0) = j \neq i+1 \). Form \( S'_{\text{opt}} \) by interchanging the values in slots \( t_1 \) and \( t_0 \) in \( S_{\text{opt}} \). Thus, \( S'_{\text{opt}}(t_1) = S_{\text{opt}}(t_0) = j \), and \( S'_{\text{opt}}(t_0) = S_{\text{opt}}(t_1) = i+1 \). The new schedule \( S'_{\text{opt}} \) is feasible (since if \( j \neq 0 \), we have moved message \( M_j \) to an earlier slot), and \( S'_{\text{opt}} \) extends \( S_{i+1} \) using only messages from \( \{i+2, \ldots, n\} \). We also have \( P(S_{\text{opt}}) = P(S'_{\text{opt}}) \), and therefore \( S'_{\text{opt}} \) is also optimal.

* **Case 2-II:** Message \( M_{i+1} \) does not occur in \( S_{\text{opt}} \). Define a new schedule \( S'_{\text{opt}} \) to be the same as \( S_{\text{opt}} \) except for time \( t_0 \), where we define \( S'_{\text{opt}}(t_0) = i+1 \). Then \( S'_{\text{opt}} \) is feasible and extends \( S_{i+1} \) using only messages from \( \{i+2, \ldots, n\} \). To finish the proof for this case, we must show that \( S'_{\text{opt}} \) is optimal.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.
timal, we must have $P(S'_{\text{opt}}) = P(S_{\text{opt}})$ and $S'_{\text{opt}}$ is optimal. So say that $S_{\text{opt}}(t_0) = j$, $j > 0, j \neq i + 1$. Recall that $S_{\text{opt}}$ extends $S_i$ using only messages from $\{i + 1, \ldots, n\}$. So $j > i + 1$, so $g_j \leq g_{i+1}$. We have $P(S'_{\text{opt}}) = P(S_{\text{opt}}) + g_{i+1} - g_j \geq P(S_{\text{opt}})$. As above, this implies that $S'_{\text{opt}}$ is optimal.

\[ \square \]