CONCERT

Cooperative Networking and Control for Emergency Response Teams

A Thesis Presented in Partial Fulfillment of the Requirements
for the Master in Science in Forensic Computing
John Jay College of Criminal Justice
City University of New York

Carol A Dottin

February 2009

CONCERT

Cooperative Networking and Control for Emergency Response Teams

Carol A Dottin

This thesis has been presented to and accepted by the Office of Graduate Studies of the John Jay College of Criminal Justice of the City University of New York in partial fulfillment of the requirements for the Master of Science in Forensic Computing.

Bilal Khan

Thesis Advisor Signature Date

Samuel Graff

Second Reader Signature Date

Dr. Jannette Domingo

Dean of Graduate Studies Signature Date

Acknowledgments

I would like to express profound gratitude to my advisor, Professor Bilal Khan, for giving me the opportunity to work on this project and without whom this work would not have been complete. His invaluable support, encouragement, and expertise have been precious throughout my studies.

I am also especially indebted to my parents, Mr. George and Mrs. Anne Dottin for their love and support throughout my course of study.

Contents

List of Figures

1 Abstract

In the event of natural or man-made disasters, public emergency response services play a critical role in saving lives and reducing property losses. In such crisis situations, each emergency response team member must fuse data from their immediate vicinity with relevant information obtained from other rescuers and data warehouses, in order to maintain situational awareness to execute effective actions which would lead to mission objectives. This project concerns the design and development of components of a system that is capable of efficiently handling the tasking requirements of emergency first responders in disaster-recovery settings.

This project is part of a larger research initiative at John Jay College of Criminal Justice (City University of New York), known as the **Cooperative Networking for Emergency Response Teams** (CONCERT) Project. The CONCERT project is funded in part by National Security Agency grant H98230-07-1-0115, whose objective is to develop a distributed mobile phone application that will address the problem of context and interest-based data acquisition and delivery for teams of disaster recovery responders. The final operational CONCERT system will include a set of feature-rich mobile phones, and a central responder tasking server. The CONCERT hardware and software will ultimately be self-organizing, creating a scalable, dynamic wireless data network that is fast-forming, robust, and able to deliver appro-

priate information and response to wherever it is needed.

The research and development efforts reported in this thesis are a significant first step in the development of CONCERT. Specifically, this thesis concerns the development of the following two inter-related constituents of the final CONCERT system:

1. **A prototype distributed system** consisting of:

   (a) Mobile phones running custom software which enables each emergency response team member to dynamically maintain information regarding their technical capabilities and GPS location, at a central server.

   (b) A central server that stores this dynamically changing information in a MySQL database, and analyzes it continuously in order to compute optimal assignments of newly submitted tasks to available emergency personnel, then informs responders of assignments by sending details to them via their mobile phones.

2. **Effective algorithms for optimal agent scheduling**. In order for CONCERT to effectively manage its resources, an experimental framework is developed to allow simulations of a number of scheduling algorithms for the purpose of evaluating the relative merits of different scheduling schemes. The algorithms considered to date fall into two classes:

- "Greedy" algorithms in which the closest available qualified responder to the assignment is chosen.
- "Future-aware" algorithms that assign qualified responders to tasks in a way that accounts for the impact of the assignment on the system's ability to respond to future tasks efficiently.

The reader is advised that there is an index at the end of this document which lists, among other things, all mathematical symbols used in this document and the page on which these symbols are defined.

2 Problem Formulation

The system to be designed will be presented with a sequence of tasks and must manage the allocation of resources (a set of first responders) effectively towards the fulfilment of these tasks – to the extent that such an objective is attainable. In what follows, we will identify each first responder with the responder's mobile phone. This is not an unreasonable supposition, since we assume that the mobile phone is the only reliable way for the system to exchange information with the emergency response personnel.

Each task has a duration and a window of time during which the task needs to be performed. Clearly, the window is always greater than or equal to the duration. Each task in general will require one or more capabilities (on the part of the responder) in order for the task to be able to be executed.

Example 1. *A document needs to be translated from Spanish to English. The duration of this task is estimated at one hour, and the window in which the task must be performed is between 5pm tomorrow and 11pm tomorrow (because the Spanish document will not be available until 5pm tomorrow and the translation into English is required by 11pm the latest). The task must be carried out at John Jay College, and must be assigned to a bilingual first responder who is literate in both Spanish and English.*

The system may choose to reject a task (e.g. because it determines that it does not have any qualified available responders to perform it). If the system rejects the task, then the submitter is free to seek alternate means to meet their objectives (i.e. using resources outside the purview of the CONCERT system). The system must however, provide quality of service (QoS) assurance in the following sense: *If the system accepts a task, then the task must indeed be fulfilled.*

In order for a task to be fulfilled it must be assigned to a first responder (i.e. a phone) that is both available (i.e. is not already assigned), and that has all the capabilities necessary for the execution of the task. If a task is accepted, the system must (at the time of acceptance) assign the task to a specific phone, and moreover, specify the precise subinterval of the task window during which the phone will perform the task. In our example above, if the task is accepted it might be assigned to phone seven (which is known to be carried by a bilingual first responder), with the specific requirement that the task be done tomorrow between 7pm and 8pm, since first responder number seven is not presently allocated to any task during that time period.

At present the system does not support advanced features such as:

- Allocating sets of phones to a single task (i.e pooling capabilities from several first responders),
- Preempting existing assignments or shuffling tasks between first re-

sponders after their initial assignment–once an assignment is made, it stands until the task is completed.

- Specifying different priorities to tasks when they are submitted.

These simplifying restrictions will be lifted in later phases of the development of CONCERT, and are beyond the scope of the investigation reported on in this thesis.

How might we measure and compare the merits of one task assignment algorithm relative to another? For any specific sequence of tasks, the most natural measure is the percentage of tasks that the system successfully assigns. To further differentiate between systems that have comparable task acceptance rates, we will consider the distances moved by the first responders (i.e phones) in the process of fulfilling their assigned tasks. Minimizing this latter quantity is a reasonable secondary objective since it reflects a real systemic cost, both in terms of time and energy. In short, given two systems with similar tasks acceptance rates, the one which requires less movement by the first responders is preferable since it makes more efficient use of resources.

In what follows, we make the previous informal exposition mathematically rigorous.

2.1 Input

The System's Resources consist of a set of phones $\mathbb{P} = \{1, \cdots, n\}$ where each phone $i \in \mathbb{P}$ is specified by:

- A set of capabilities $C_i \subseteq \mathbb{C}$, where $\mathbb{C} = \{1, \cdots, m\}$.
- An initial location, given by $x_i(0), y_i(0)$, where

$$-180^\circ \leq x_i(0) \leq 180^\circ, \quad and$$

$$-90^\circ \leq y_i(0) \leq 90^\circ.$$

- A maximum speed $\upsilon_i > 0$.

While resources are fixed at the outset, the System's Tasks arrive in an "on-line" manner as a sequence of k tasks $\mathbb{T} = (T_1, \cdots, T_k)$, where each task T_j is specified by:

- A location (lat_j, lon_j)
- A time window $(start_j, end_j)$
- A time duration d_j
- A set of requirements $R_j \subseteq \mathbb{C}$, where $\mathbb{C} = \{1, \cdots, m\}$
- An arrival time a_j

Here the (lat_j, lon_j) values are standard geographic coordinates specifying as reported by the GPS receiver of each phone, satisfying:

$$\begin{aligned} -180^\circ \leq \quad & lon_j \quad \leq 180^\circ \\ -90^\circ \leq \quad & lat_j \quad \leq 90^\circ \end{aligned}$$

The earliest time a task can be started is denoted $start_j$ and the time by which the task must be completed (if the task is accepted) is denoted end_j. The amount of time the task will take to be executed is denoted d_j. The subset of capabilities which the chosen phone must possess in order to execute the task is denoted $R_j \subseteq \mathbb{C}$. The time at which the task is presented to the system (for assignment or rejection) is denoted a_j.

In order for a task T_j to be a **syntactically valid** part of sequence $\mathbb{T}$ it must satisfy these constraints:

- Arrival time precedes start time: $a_j \leq start_j$
- Duration cannot exceed the window: $start_j + d_j \leq end_j$
- Duration must be non-negative: $0 \leq d_j$
- Tasks arrive sequentially: $a_j \leq a_{j+1}$

Tasks must arrive (i.e. be submitted for scheduling) no later than the earliest time that they may be started. The duration of a task cannot be negative, nor can it be larger than the time interval between the earliest time a task can start and the time by which the task must be completed. Finally, the sequence $\mathbb{T}$ is, without loss of generality assumed to be sorted by task arrival time.

2.2 Algorithm Correctness

Given the resources $\mathbb{P}$ and task sequence $\mathbb{T}$, we seek to design algorithms for scheduling phones (resources) to tasks. Consider a hypothetical algorithm $\mathbb{N}$ being presented with a sequence of syntactically valid tasks, $\mathbb{T} = (T_1, T_2, \cdots, T_j, \cdots, T_k)$, one at a time. Upon receiving T_j at time a_j, algorithm $\mathbb{N}$ must decide to either **Accept** T_j *or* **Reject** T_j.

If it chooses to accept then it must assign task T_j to some phone, denoted $A_{\mathbb{N}}(T_j) \in \mathbb{P}$ and simultaneously specify the **execution interval**,

$$[\varepsilon_{\mathbb{N}}(T_j), \varepsilon_{\mathbb{N}}(T_j) + d_j]$$

during which the task T_j will actually be executed by phone $A_{\mathbb{N}}(T_j)$. If, however, algorithm $\mathbb{N}$ chooses to reject the task, then phone $A_{\mathbb{N}}(T_j)$ and $\varepsilon_{\mathbb{N}}$ are both undefined and set to a sentinel value of *"reject"*. Formally then, algorithm $\mathbb{N}$'s operation on a syntactically valid sequence of tasks $\mathbb{T}$ gives rise to two implicit maps:

$$\begin{aligned} A_{\mathbb{N}} : \mathbb{T} &\rightarrow \mathbb{P} \cup \{reject\} \\ \varepsilon_{\mathbb{N}} : \mathbb{T} &\rightarrow \mathbb{R}^{\geq 0} \cup \{reject\} \end{aligned}$$

An algorithm's assignment $(A_{\mathbb{N}}, \varepsilon_{\mathbb{N}})$ is said to be a **feasible solution** for a

sequence of tasks $\mathbb{T}$ given resources $\mathbb{P}$, if the following five conditions hold:

Feasibility Criteria

1. $A_{\mathbb{N}}$ and $\varepsilon_{\mathbb{N}}$ are mutually consistent, i.e. $A_{\mathbb{N}}$ rejects if and only if $\varepsilon_{\mathbb{N}}$ rejects:

$$\forall T_j \in \mathbb{T} : A_{\mathbb{N}}(T_j) = reject \quad \Leftrightarrow \quad \varepsilon_{\mathbb{N}}(T_j) = reject$$

2. The phone (responder) possesses the necessary capabilities to perform the task:

$$\forall T_j \in \mathbb{T} : R_j \subseteq C_{A_{\mathbb{N}}(T_j)}$$

3. The execution interval of the task is within the window defined for the task:

$$\forall T_j \in \mathbb{T} : start_j \leq \quad \varepsilon_{\mathbb{N}}(T_j) \quad \leq end_j - d_j$$

4. No phone is requested to do more than one task at any given time:

$$\begin{aligned} \forall T_j, \forall T_k \in \mathbb{T} : \qquad & (k < j \wedge A_{\mathbb{N}}(T_k) = A_{\mathbb{N}}(T_j)) \Rightarrow \\ & [\varepsilon_{\mathbb{N}}(T_j), \varepsilon_{\mathbb{N}}(T_j) + d_j] \cap [\varepsilon_{\mathbb{N}}(T_k), \varepsilon_{\mathbb{N}}(T_k) + d_k)] = \emptyset \end{aligned}$$

The final (fifth) condition is that no phone must be asked to move faster than its maximum speed. To make this mathematically precise, some intermediate definitions are needed.

Definition 1. *We define* $gaps : \mathbb{T} \longrightarrow 2^{\mathbb{R}}$ *as follows:*

$$gaps(T_j) \stackrel{def}{=} R^{\geq 0} \cap \{\varepsilon_{\mathbb{N}}(T_j) - (\varepsilon_{\mathbb{N}}(T_k) + d_k) \mid k < j \wedge A_{\mathbb{N}}(T_k) = A_{\mathbb{N}}(T_j)\}$$

Informally, $gaps(T_j)$ is the set of time differences between the time when the task T_j will begin and times at which tasks presently assigned to phone $A_{\mathbb{N}}(T_j)$ end.

Definition 2. *We define* $gap : \mathbb{T} \longrightarrow \mathbb{R}^{\geq 0}$ *as follows:*

$$gap(T_j) \stackrel{def}{=} \begin{cases} min(gaps) & if \ \ |gaps(T_j)| \neq 0 \\ \varepsilon_{\mathbb{N}}(T_j) & otherwise. \end{cases}$$

Informally, $gap(T_j)$ represents the time elapsed between the end of phone $A_{\mathbb{N}}(T_j)$'s last task prior to the start of task T_j.

Definition 3. *We define* $prev : \mathbb{T} \longrightarrow \{1, 2, \ldots, k\} \cup \{\infty\}$ *as follows:*

$$prev(T_j) \stackrel{def}{=} \begin{cases} \ell & if \ \ \varepsilon_{\mathbb{N}}(T_j) - [\varepsilon_{\mathbb{N}}(T_\ell) + d_\ell] = gap(T_j) \\ & \quad and \ A_{\mathbb{N}}(T_\ell) = A_{\mathbb{N}}(T_j) \\ & \quad and \ \ell \leq j \\ \infty & if \ \ otherwise. \end{cases}$$

Informally, $prev(T_j)$ is the last task completed by phone $A_{\mathbb{N}}(T_j)$ prior to its starting task T_j.

Definition 4. *We define* $prevloc : \mathbb{T} \longrightarrow [-180°, 180°] \times [-90°, 90°]$ *as follows:*

$$prevloc(T_j) \stackrel{def}{=} \begin{cases} (lat_{prev(T_j)}, lon_{prev(T_j)}) & if\ prev(T_j) \neq \infty \\ (x_j(0), y_j(0)) & otherwise. \end{cases}$$

Informally, $prevloc(T_j)$ is the location of the last task completed by phone $A_{\mathbb{N}}(T_j)$, prior to its starting task T_j.

We are now finally ready to state the 5th condition precisely as follows.

5. No phone can be asked to move faster than its maximum speed, i.e.

$$\forall T_j \in \mathbb{T} : A_{\mathbb{N}}(T_j) \neq rejected \Rightarrow \frac{d((lat_j, lon_j), prevloc(T_j))}{gap(T_j)} \leq v_{A_{\mathbb{N}}}(T_j)$$

where

$$d : ([-180°, 180°] \times [-90°, 90°]) \times ([-180°, 180°] \times [-90°, 90°]) \longrightarrow \mathbb{R}^{\geq 0}$$

is the function which assigns to each pair of points (both specified in terms of their latitude and longitude) the geodesic distance between the two points on the sphere defined by the Earth's surface.

2.3 Performance Measures

Suppose algorithm $\mathbb{N}$, given resources $\mathbb{P}$ and a syntactically valid task sequence $\mathbb{T}$ acts via feasible maps $A_{\mathbb{N}}, \varepsilon_{\mathbb{N}}$. To evaluate the effectiveness of an algorithm, we consider the set of accepted tasks:

$$\Gamma_{\mathbb{N}}(\mathbb{T}) \stackrel{def}{=} \{T_j \in \mathbb{T} \mid A_{\mathbb{N}}(T_j) \neq rejected\}$$

and use this as the basis upon which to define two performance measures:

1. *Acceptance Rate* of $\mathbb{N}$ on $\mathbb{T}$ is defined as:

$$\gamma_{\mathbb{N}}(\mathbb{T}) \stackrel{def}{=} \frac{|\Gamma_{\mathbb{N}}(\mathbb{T})|}{|\mathbb{T}|}$$

   where the numerator is the cardinality of the set of tasks which were accepted, and the denominator is the cardinality of the set of all tasks.

2. *Average Movement Cost* of $\mathbb{N}$ on $\mathbb{T}$ is given by:

$$\beta_{\mathbb{N}}(\mathbb{T}) \stackrel{def}{=} \frac{\sum_{T_j \in \Gamma_{\mathbb{N}}(\mathbb{T})} d((lat_j, lon_j), prevloc(T_j))}{|\mathbb{T}| \cdot \gamma_{\mathbb{N}}(\mathbb{T})}$$

   where the numerator is the total distance moved to fulfil each accepted task, and the denominator is the cardinality of the set of all accepted tasks.

3 Scheduling Algorithms

In what follows I describe the operation of three concrete algorithms: Random (R), Greedy (G) and Future (F), all of which produce feasible solutions (as defined in the previous section). In the Section 4, I will present the design and implementation of a simulation framework that is capable of evaluating the relative merits of these scheduling algorithms (and indeed any other future algorithms that may be developed). Finally, in Section 5, I will use the simulation framework to evaluate algorithms R, G, and F, and present the outcomes of these experiments together with an analysis and interpretation.

3.1 Algorithm R

The first algorithm R considers a submitted task T_j and seeks to find any available phone i that is capable of fulfilling T_j via an assignment that fulfils the feasibility criteria 1-5 described in Section 2. If more than one phone fulfils this criteria, the random algorithm simply chooses one of the phones arbitrarily.

In practice this algorithm is implemented by first having the task controller search through the (unordered) set of phones until it finds a phone i meeting

feasibility criteria (2):

$$R_j \subseteq C_{A_{\mathbb{N}}(T_j)}. \tag{1}$$

If each phone stores its capabilities in a balanced binary tree, this search can be completed in worst-case time $O(|R_j| \sum_{i=1}^{n} \log |C_i|)$. Let us denote the resulting subset of phones capable of performing task T_j – with respect to feasibility criterion (2) (page 10), as $f_2(T_j)$.

We can get a sense of the average size of $f_2(T_j)$ using probabilistic analysis. Suppose each phone i has a set of capabilities of cardinality χ, taken at random from the set of all capabilities $\mathbb{C} = \{1, 2, \ldots, m\}$, and each task T_j has a set of requirements of cardinality $\tau \leq \chi$, also taken at random from the set of all capabilities $\mathbb{C}$. Then the previous time complexity $O(|R_j| \sum_{i=1}^{n} \log |C_i|)$ becomes $O(\tau n \log \chi)$. We refer to χ as the **mean phone capabilities** and τ as the **mean task complexity**. Moreover, the probability that a random phone is able to do a random task is:

$$\frac{\begin{pmatrix} \chi \\ \tau \end{pmatrix}}{\begin{pmatrix} m \\ \tau \end{pmatrix}} = \frac{\chi!(m-\tau)!}{m!(\chi-\tau)!}.$$

and so the expected size of

$$E[|f_2(T_j)|] \;=\; n \cdot \frac{\chi!(m-\tau)!}{m!(\chi-\tau)!}.$$

In the special case where $\tau = \chi$, this quantity becomes

$$\begin{aligned} E[|f_2(T_j)|] &= n \cdot \frac{\tau!(m-\tau)!}{m!0!} = \frac{n}{\left(\begin{array}{c} m \\ \tau \end{array}\right)} \\ &\leq \frac{n}{\left(\frac{m}{\tau}\right)^{\tau}} = n(\tau/m)^{\tau}. \end{aligned}$$

Now the algorithm assesses the phones in $f_2(T_j)$ as it constructs this set, and returns any phone for which it is possible to make an assignment $\epsilon_{\mathbb{N}}$ satisfying criteria (3), (4) and (5). We denote the set of phones satisfying criteria 2-5 (page 10) as $f_{2-5}(T_j)$.

The worst case cost of finding an element in $f_{2-5}(T_j)$ is $O(j - 1 + |f_2(T_j)|)$ time, since it could require examining all tasks that have arrived prior and examining every phone in $f_2(T_j)$; thus the total worst-case cost of assigning task T_j is

$$O(j + |R_j| \sum_{i=1}^{n} \log |C_i|).$$

Since tasks are of finite duration, we may extend the previous probabilistic formulation by assuming that on average δ tasks are active in the system at any given time. We refer to δ as the **mean task intensity**. This yields an *expected time complexity* of task assignment

$$O(\delta + \tau n \log \chi).$$

We see that Algorithm R has expected cost that depends linearly on mean task intensity, mean task complexity, and the number of phones. Having chosen a phone satisfying the feasibility criteria, the controller specifies the execution interval of the task to be the earliest available contiguous subinterval of duration d within the task window that does not overlap with the execution intervals of the phone's other assignments. The algorithm R is the simplest possible algorithm in that it always assigns phones to tasks if it is possible to do so, and when there is more than one feasible assignment the algorithm chooses randomly from among its options; this is by virtue of the fact that R returns the first phone it finds that is able to meet the feasibility criteria.

3.2 Algorithm G

The behavior of the second algorithm G differs from the first algorithm R, only in the case where the set of phones meeting the feasibility criteria $f_{2-5}(T_j)$ has cardinality greater than one. In this case, the controller implementing the G algorithm assigns the task to that phone $i \in f_{2-5}(T_j)$ that would have to move the shortest distance in order to execute the task. Formally stated, $A_G(T_j)$ is taken to be the phone $i \in f_{2-5}(T_j)$ which would minimize

$$d((lat_j, lon_j), prevloc(T_j)).$$

In practice, this minimization requires the controller to consider every phone in $f_{2-5}(T_j)$, where Algorithm R would have considered, on average only half the elements of $f_{2-5}(T_j)$. While this factor of 2 increase in time complexity does not change the asymptotic cost of assigning a task (since big-O notation is immune to constant factors), it does produce different assignment choices. We will explore the nature and implications of these different choices shortly.

3.3 Algorithm F

While algorithms R and G operate in a greedy way, which is to say, they do not consider future implications of their assignment decisions, algorithm F seeks to take the future impact of its present decisions into consideration. To do so, it must make certain assumptions about the nature of tasks it anticipates seeing in the future. In the worst case, one might consider that the future is determined by a **omniscient malicious adversary** which seeks to present us with tasks that are maximally costly given our past choices. In practice, however, this is needlessly pessimistic. In this study, I considered the case where future tasks are determined by a **probabilistic non-malicious adversary**, which creates tasks in a fixed interval in advance of their starting time, where the tasks generated are distributed uniformly and randomly in space, and always require a randomly selected subset of cardinality τ of the m capabilities. The window of a task is a fixed size W, and the duration of the task is always a constant fraction ϵ, $0 \leq \epsilon \leq 1$, of its task window. We refer to ϵ as *slack*.

Given these assumptions about the behavior of the probabilistic adversary's construction of tasks, algorithm F operates as follows. When presented with a new task T_j, algorithm F computes the statistically *expected cost* in fulfilling the probabilistic adversary's hypothetical future task T_{j+1} under the assumption $A_F(T_j) = i$, for each feasible assignment $i \in f_{2-5}(T_j)$. It then

makes the assignment of the current task T_j to the phone that minimizes the expected cost of fulfilling the future task. It should be noted that in computing the expected cost of fulfilling the future task T_{j+1}, algorithm F assumes that the future task is going to be fulfilled by the algorithm G, even though (somewhat paradoxically) when the future task T_{j+1} actually arrives, it will be processed using algorithm F (in the same manner as T_j was) and not by algorithm G. There is no way to avoid this reference to G, without leading to a self-referential definition of algorithm F.

We make this assignment algorithm more explicit: Given a task T_j, algorithm F proceeds as follows:

Step 1: Find all phones in $f_{2-5}(T_j)$.

Step 2: For each phone $i \in f_{2-5}(T_j)$ compute the expected cost of algorithm G to fulfil the future task T_{j+1} generated by the probabilistic adversary, as follows:

(a) For each possible location (x, y) and each possible cardinality τ subset of capabilities z, compute the cost of fulfilling a hypothetical task at location (x, y) requiring capabilities z using algorithm G, under the assumption that T_j is assigned to phone i.

(b) Compute the average value of the previously computed set of costs and call this the expected future cost of a task if T_j is assigned to phone i, denoted $E[fc(j, i)]$.

Step 3: Choose for T_j the phone i for which $E[fc(j,i)]$ is minimal.

Note that the actual computation in steps 2a and 2b must in practice be an approximation of the expected value of the future cost. This is necessary since an exact execution of the directive "For each possible location (x, y)" would require the construction of an infinite set in step 2a and computationally infeasible minimization in step 2b. Thus, in practice, step 2a is implemented as "For each location (x, y), taken from a finite set L of candidate placements of the future task T_{j+1}..." In this work, the finite set of candidate placements $L \subset [-180°, 180°] \times [-90°, 90°]$ is taken to be a Cartesian lattice of uniformly spaced points at intervals of 1 degree spacing.

As in the first two algorithms R and G, the algorithm F assigns the execution interval to be the earliest available contiguous subinterval of duration d within the task window that does not overlap with the execution intervals of the phone's other assignments.

The time complexity of Algorithm F requires simulating algorithm G on each of the $|L|$ possible placements of the hypothetical future task T_{j+1}. Unraveling the above three steps, we see that this takes expected time

$$O(n(\tau/m)^\tau \cdot |L|(me/\tau)^\tau \cdot (\delta + \tau n \log \chi)),$$

The first term $n(\tau/m)^\tau$ is an upper bound on expected size of $E[|f_{2-5}(T_j)|]$:

$$\begin{aligned} E[|f_{2-5}(T_j)|] &\leq E[|f_2(T_j)|] \\ &\leq n(\tau/m)^\tau, \end{aligned}$$

$|L|$ is the number of possible positions of task T_{j+1}; $(me/\tau)^\tau$ is an upper bound on the number of different capability requirements that may be found in T_{j+1}, since

$$\begin{pmatrix} m \\ \tau \end{pmatrix} \leq (me/\tau)^\tau;$$

and $\delta + \tau n \log \chi$ is the cost of running Algorithm G on the hypothetical task T_{j+1}. Combining we see that the cost of assignment of a task is:

$$O(n|L|e^\tau(\delta + \tau n \log \chi)),$$

which is exponential in the task complexity, quadratic in the number of phones, linear in the grid size and linear in the task intensity.

3.4 Examples of differences in algorithms

Recall Example 1, in which we have a document that needs to be translated from Spanish to English with a window between 5pm and 11pm tomorrow and has an estimated duration of one hour. To understand the operation of algorithm R in this setting, suppose we have three phones:

- Phone 1 has a contiguous segment of one hour within the window from 6pm to 7pm to perform the task, however it does not have the capability to translate documents from Spanish to English.
- Phone 2 can translate documents from Spanish to English. Between 5pm and 11pm tomorrow, it has tasks assigned from 5pm to 7pm, 7:30pm to 9pm and 9:30pm to 11:30pm.
- Phone 3 on on the other hand can translate documents from Spanish to English. It has a previously assigned task to perform from 5pm to 7pm.

Given the description of algorithm R, only phone 3 meets the feasibility criteria and so will be assigned the task, with the requirement that the task be done between 7pm to 8pm since this is the first contiguous hour interval within the window during which phone 3 is free.

To see the difference between algorithm G and algorithm R, suppose we have two phones:

- The responder with phone 1 has the capability to translate documents from Spanish to English but has a previously assigned task with an execution interval from 3pm to 4pm and is 500 meters away from John Jay College where the new task must be performed.
- Responder 2 can also translate documents from Spanish to English and has a previously assigned task from 4pm to 5pm that will cause it to be 200 meters away from John Jay College where the new task must be performed.

Neither phones 1 nor 2 have other tasks assigned prior to the task to be assigned. Since both phones fulfil conditions 2-5 (page 10), the algorithm G considers the location of both phones just prior to the start of the submitted task. In this situation, phone 2 is closer to the submitted task as it is only 200 meters away. Algorithm G will therefore assign the new task to phone 2.

To demonstrate that the decisions reached by algorithm F differ markedly from the decisions reached by algorithm G, consider the following one-dimensional example. Task T needs to be completed at the location indicated on the graph. Phone p and q are both completely unassigned and capable of fulfilling task T; the locations of p and q are identified on the graph as well.

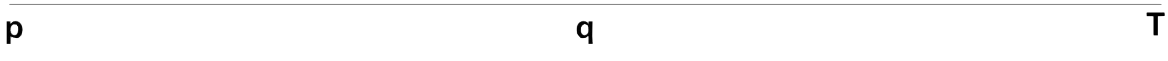


Figure 1: One-dimensional task simulation

To determine which phone algorithm F will assign to task T, it must compute the expected cost of fulfilling a future task if T were to be assigned to q and p respectively. It then chooses phone p if $E[fc(T,p)] < E[fc(T,q)]$, and chooses phone q otherwise.

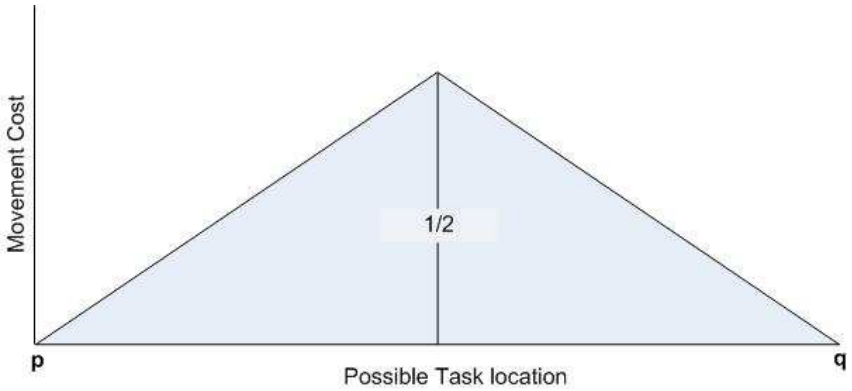


Figure 2: Distance moved for hypothetical task locations of task T

In Figure 2, we contemplate what happens if the task was assigned to phone q. For all possible locations (x, y), the distance to be traveled by the phone closest to the task as determined by algorithm G is plotted and the average future cost is calculated as follows:

Total area $= \frac{1}{2} \times (1 \times \frac{1}{2}) = \frac{1}{4}$

So, average future cost $E[fc(T,q)] = \frac{1}{4}/1 = \frac{1}{4}$

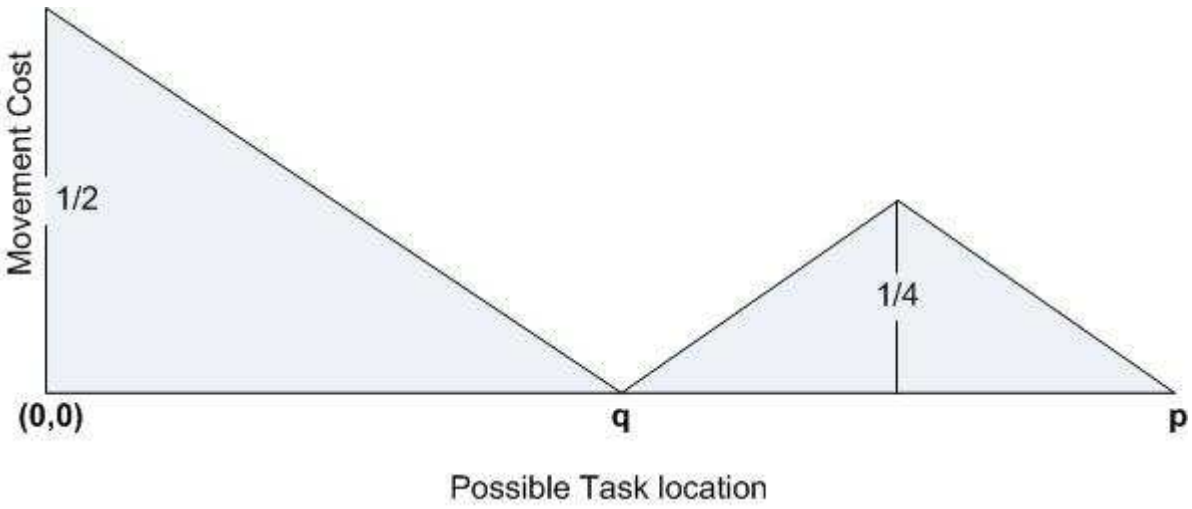


Figure 3: Distance moved for hypothetical task locations of task T

In Figure 3, we contemplate what happens the task was assigned to phone p. For all possible locations (x, y), the distance to be traveled by the phone closest to the task as determined by algorithm G is plotted and the average future cost is calculated as follows:

Total area $= \frac{1}{8} + \frac{1}{16} = \frac{3}{16}$
So, average future cost $E[fc(T, p)] = \frac{3}{16}/1 = \frac{3}{16}$

Based on the evidence in Figures 2 and 3, it is clear that Algorithm F will choose to assign task T to phone p. Clearly, this differs from the decision reached by algorithm G, which would choose the closest phone to T, namely phone q.

3.5 Note on Riemann Approximations

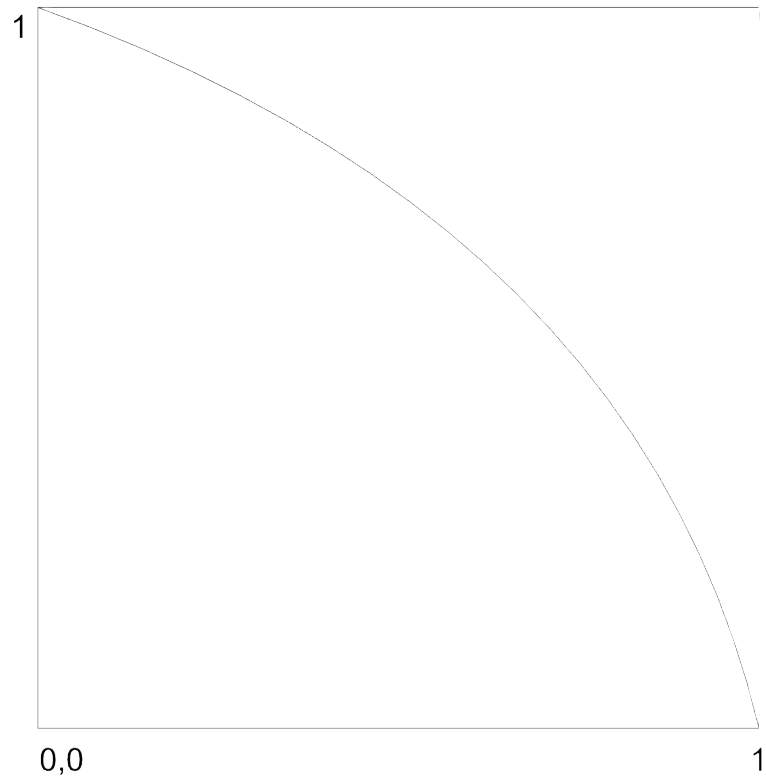
The true value of the future cost can be computed by evaluating an integral and what I am doing in using a Cartesian lattice L is effectively replacing the integral with a finite Riemann sum. Since it is not the absolute values of the expected future costs that are important but only their relative values that are used to assess the merits of different assignments, the approximation introduced by the Riemann sum is not a cause for concern. For completeness I give two examples of the exact computation of expected future cost using integral calculus.

Example 1: Suppose we have a phone at $(0, 0)$ and a random task appears in the unit square. What is the expected movement cost to fulfil the task?

$$\text{E(cost)} = \frac{\int_{y=0}^{1}\int_{x=0}^{1}(x^2+y^2)\,dx\,dy}{1*1}$$

$$\int_{y=0}^{1}(x^2+y^2)\,dy$$
$$= [x^2y + \tfrac{y^3}{3}]|_{y=0}^{y=1}$$
$$= x^2 + \tfrac{1}{3}$$

$$\int_{x=0}^{1}(x^2+\tfrac{1}{3})\,dx$$
$$= [\tfrac{x^3}{3} + \tfrac{x}{3}]|_{x=0}^{x=1}$$
$$= \tfrac{1}{3} + \tfrac{1}{3}$$
$$= \tfrac{2}{3}$$



Average value $= \frac{2}{3} \div 1$

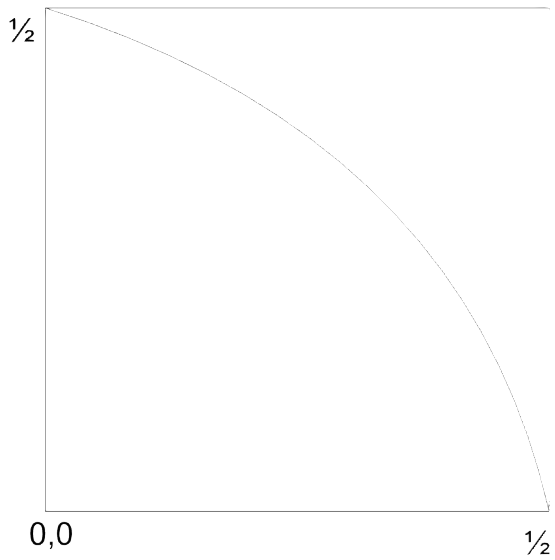
$= \frac{2}{3}$

Example 2: Suppose we have a phone at $(0.5, 0.5)$ and a random task appears in the unit square. What is the expected movement cost to fulfil the task?

E(cost) $= \int_{x=0}^{\frac{1}{2}} \int_{y=0}^{\frac{1}{2}} \frac{(x^2+y^2)\,dx\,dy}{1*1}$

$\int_{y=0}^{\frac{1}{2}} = (x^2 + y^2)\,dy$

$= [x^2 y + \frac{y^3}{3}]|_{y=0}^{y=\frac{1}{2}}$



$= \frac{1}{2}x^2 + \frac{1}{24}$

$\int_{x=0}^{\frac{1}{2}} (2x^2 + \frac{1}{24})dx$

$= [\frac{x^3}{6} + \frac{x}{24}]|_{x=0}^{x=\frac{1}{2}}$

$= \frac{1}{48} + \frac{1}{48}$

$= \frac{1}{24}$

Average value $= \frac{1}{24} \div \frac{1}{4}$

$= \frac{1}{6}$

From these two examples ($1/6 < 2/3$), we see (not surprisingly) that it is better to place a phone in the middle of a unit square, than at one of its corners. The Reimann sum approach would reach the same conclusion, provided the lattice L was sufficiently fine.

4 Simulation Framework

We now go on to detail the framework used to evaluate the relative merits of the scheduling algorithms described in Section 3. The framework can also be used to evaluate the relative merits of any future algorithms developed as an extension of this project. The simulation framework falls into two distinct layers, (1) the Discrete event simulator (FW) and (2) the CONCERT simulator, which is built using the Discrete event simulator as its foundation.

4.1 The Discrete event simulator

The Discrete event simulator (FW) presents the operation of the system as a chronological sequence of events modeling the real world. It is used to understand the operation of the real system and evaluate the performance measures described in the previous section. The Discrete event simulator provides the foundation for the concrete classes of the CONCERT simulator and specifies how inputs to the simulator by CONCERT are handled. The Discrete event simulator contains two base classes, SimEntity and Event, and a Scheduler.

1. SimEntity

   A SimEntity represents an object which (a) is capable of generating

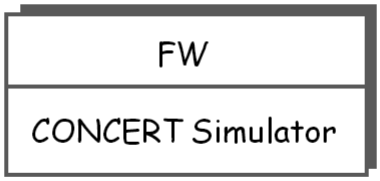


Figure 4: Architecture of the Simulation Framework

events to be delivered to other SimEntities, and (b) can respond to the arrival of events.

2. Event

   An Event is a message that flows between SimEntities.

3. Scheduler

   The Scheduler manages a queue of events, and delivers them to the appropriate Entities at the specified times, allowing them to respond by changing their state and perhaps creating new events.

4.2 Concrete classes in the CONCERT Simulator

1. SimEntities

   (a) Task Generator

   This is responsible for generating new tasks at a set rate and send-

ing them to each Controller for scheduling. The Task Generator continuously sends Make Task events to itself at a set time interval, which triggers the generation of new tasks. On receiving this event, a Submit Task event is made and sent to each Controller it has registered in the simulation.

The internals of the Submit Task event specify a new task as follows:

$$
\begin{aligned}
a_i &= \Delta \cdot i, \\
&\quad \textit{where } \Delta \textit{ is the inter} - \textit{arrival time of each task} \\
lat_i &= 90 \cdot rand(-1, 1) \\
lon_i &= 180 \cdot rand(-1, 1) \\
R_i &= \textit{a random cardinality } \tau \textit{ subset } k, \textit{ of } \mathbb{C} \\
start_i &= \textit{randomly sampled value from } N(a_i + L, 1hr), \\
&\quad \textit{where } L \textit{ is the look ahead time, and } N \textit{ is a normal distribution} \\
end_i &= \textit{randomly sampled value from } N(start_i + W, 1hr), \\
&\quad \textit{where } W \textit{ is the window of a task} \\
d_i &= (end_i - start_i) \times \epsilon, \\
&\quad \textit{where } \epsilon \textit{ is slack value between } 0 \textit{ and } 1
\end{aligned}
$$

Recall from Section 3, on average δ tasks are active in the system at any given time. Since the duration of a task is on average $(W \times \epsilon)$ in length, then

$$\begin{aligned} \delta &= \frac{W \times \epsilon}{\Delta} \\ \therefore \Delta &= \frac{W \times \epsilon}{\delta} \end{aligned}$$

Thus we can say each new task T_i has an arrival time a_i which is equivalent to $\frac{W \times \epsilon}{\delta} \cdot i$.

It is possible that end_i generates a value which is less than $start_i$ since a normal distribution about $start_i$ can produce a negative number. To ensure this does not occur, the value of end_i is checked against $start_i$ and a new value is calculated continuously until it is greater than $start_i$.

The Submit Task is sent to each Controller entity. If the task is accepted, the Controller seeks to assign the task to a phone that meets the feasibility criteria described in Section 2 (page 10) using the scheduling algorithm implemented by the Controller.

(b) Controller

The Controller represents a resource scheduling algorithm. It manages a set of phones which it assigns to incoming tasks. The main activity of the controller is to receive Submit Task events from the Task Generator, analyze them, reach a scheduling deci-

sion and send an Assign Task event to the phone it determines is most suitable to perform the task. The Controller is also responsible for receiving updates from phones regarding their status.

(c) Phone

The phone is one of the main participants in the CONCERT simulators. The phone receives a number of events from itself and other SimEntities and responds to them as follows:

- Assign Task - when a phone receives an Assign Task event it sends notification to its controller that it has begun the assignment. It then creates a Move event and send to itself. On completion of the task, it creates a Task Complete event and then sends this to itself.
- Move - on receiving a move event, the phone updates it latitude and longitude and notifies the controller of its new location.
- Task Complete - on receiving this event the phone notifies its controller that it has completed the task and is available for future task.

2. Events

(a) Make Task

This event is used to periodically create additional tasks at the Task Generator.

(b) Submit Task

A Submit Task event is used to provide each controller with information regarding the task to be assigned. It contains the location of the task, the capabilities required, the duration of the task and the earliest and latest time the task can be scheduled.

(c) Assign Task

The Assign Task event is used by Controllers to convey assignments to a phone. It provides the phone with the location of the task and its start time and duration.

(d) Move

The Move event occurs at the beginning of a task execution interval and notifies the receiving phone to change its location to the coordinates given by the event.

(e) Task Complete

A Task Complete event is used to signal the completion of an assigned task by a phone.

4.3 The Design of the CONCERT Simulator

The CONCERT Simulator operates as follows:

- A Task Generator is created.

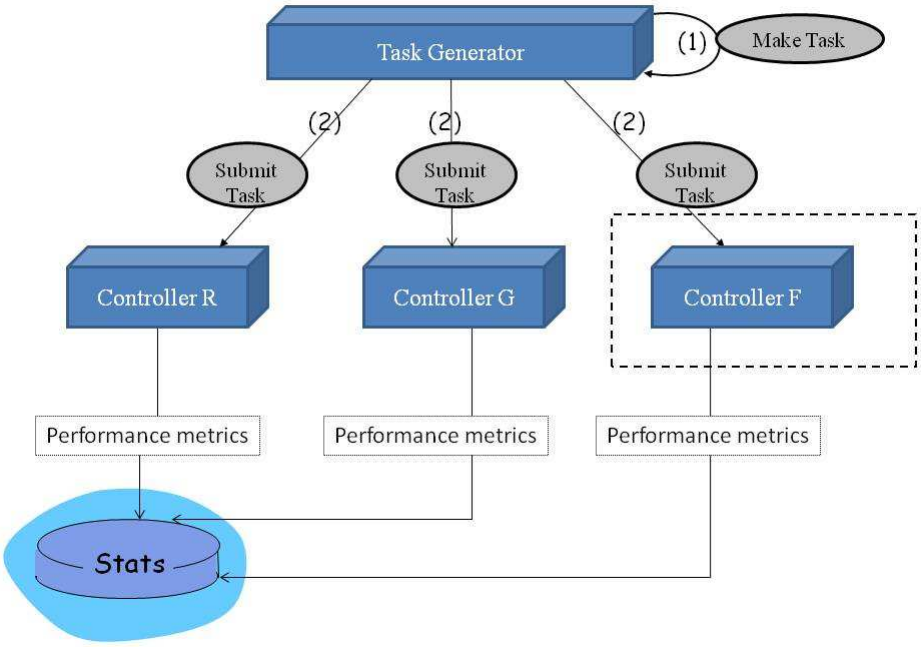


Figure 5: Design of the CONCERT Simulator

- A set of random phones is created.
- A set of Controllers is created, one for each scheduling algorithm as described in Section 3.
- Each Controller is registered with the Task Generator.
- For each Controller, an exact copy of the set of phones is made and each such set is registered with their respective Controller.

The CONCERT simulator, as shown in Figures 5 and 6, operates as follows:

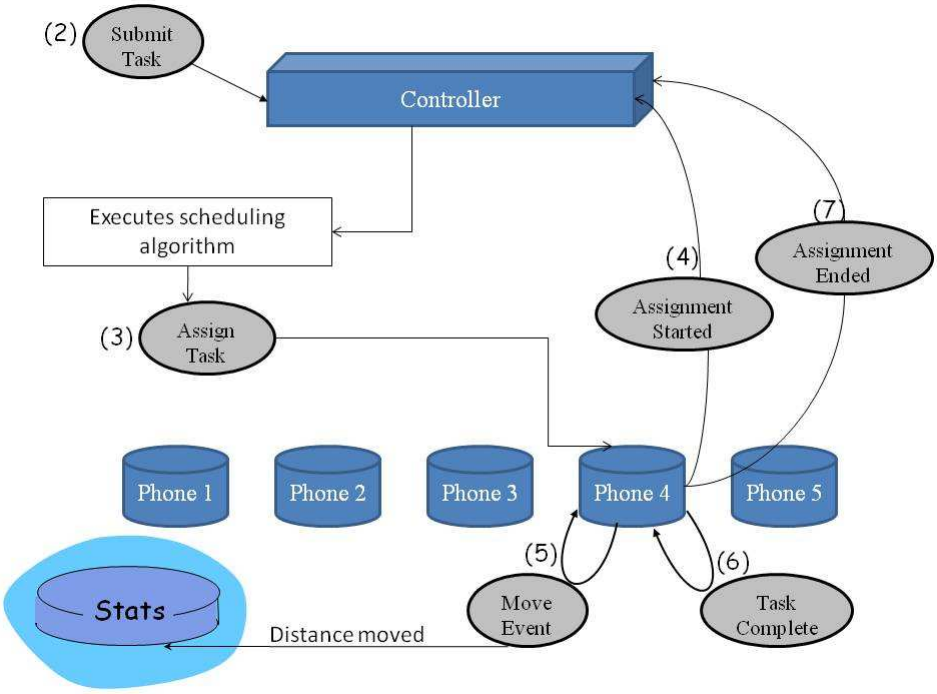


Figure 6: Design of the CONCERT Simulator cont'd

The Task Generator generates a Make Task event [Figure 5 (1)] and sends the event to itself. On receipt of the Make Task event, the Task Generator makes a Submit Task event. An exact duplicate of the Submit Task event is made for each controller and this copy is sent to each controller [Figure 5 (2)].

When each controller receives the Submit Task event, it executes its scheduling algorithm using its set of phones to determine which phone meets the feasibility criteria required to execute the task. If a suitable phone is available, an Assign Task event is created and the Assign Task event is added to a list of tasks for that phone. The event is sent to the chosen phone at the

time when the task is suppose to start [Figure 6 (3)]. The task is recorded as being assigned. If no suitable phone is available to complete the task, the task is recorded as unassigned.

On receipt of the assign task, the phone chosen to complete the task sends notification to its controller that it has started the assignment [Figure 6 (4)]. The Controller on receipt of this notification removes the handle it stored for that task.

The phone then creates a Move event with the location of the task and sends it to itself immediately [Figure 6 (5)]. When the phone receives the move event, it calculates the distance between its current location and the location of the task to be completed. The phone then updates it location to the location of the task and notifies the Stats Collector of the distance traveled.

The phone makes a Task Complete event and sends it to itself on completion of the task [Figure 6 (6)]. When it is received by the phone, the controller is notified that the assignment has ended [Figure 6 (7)]. When the controller receives the notification, it removes the assignment from the list of tasks assigned to that phone.

The Make Task event is sent to the task generator every δ time unit and the process is repeated until the simulation is ended.

5 Experimental Findings

To better evaluate the effectiveness of the three algorithms described in Section 3, I analyzed several key input parameters over a range of values using the simulation framework. The performance measures of Acceptance Rate and Average Movement Cost and the movement of the controllers relative to each other was considered and shown in the following graphs.

5.1 Capabilities

The set of capabilities $\mathbb{C}$ from which phones and tasks choose their requirements was analyzed with m ranging from 1 to 20 with increases in increments of 2, while the other parameters to the simulator were kept at the following values:

The set of phones $\mathbb{P}$ has a cardinality of $n = 2$. Each phone $i \in \mathbb{P}$ has the following finite values:

$$\begin{aligned} |C_i| &= 1 \\ v_i &= 1.0 \div 3600 \end{aligned}$$

Recall that v_i is the maximum speed a phone can travel (page 7).

Each task T_i has the following finite values:

$$\begin{aligned} a_i &= 3600 \cdot i \\ start_i &= N(a_i + L, 1hr),\ where\ L\ is\ (24 \times 3600) \\ end_i &= N(start_i + W, 1hr),\ where\ W\ is\ (4 \times 3600) \\ slack\ (\epsilon) &= 0.8 \\ |R_i| &= 1 \end{aligned}$$

The results generated by algorithms G (Controller 1), R (Controller 2), and F (Controller 3) are shown in Figures 7 and 8.

The trend of the Acceptance Rate graph in Figure 7 shows that Controller 1 (Algorithm G) and Controller 2 (Algorithm R) have the same Acceptance Rate for all values of the capabilities set $\mathbb{C}$. It should be noted that the graphs for Controller 1 and Controller 2 follow the same path on the graph. This is expected, since both Algorithm G and Algorithm R make their assignments from phones using the same feasibility criterion and only differ in their choices when more than one phone meets the criterion by choosing phones based on distance or randomness respectively.

While there is fluctuation in the Acceptance Rate as the number of capabilities increases, there is a general downward movement in the number of tasks that are successfully assigned as the capabilities set increases. This is

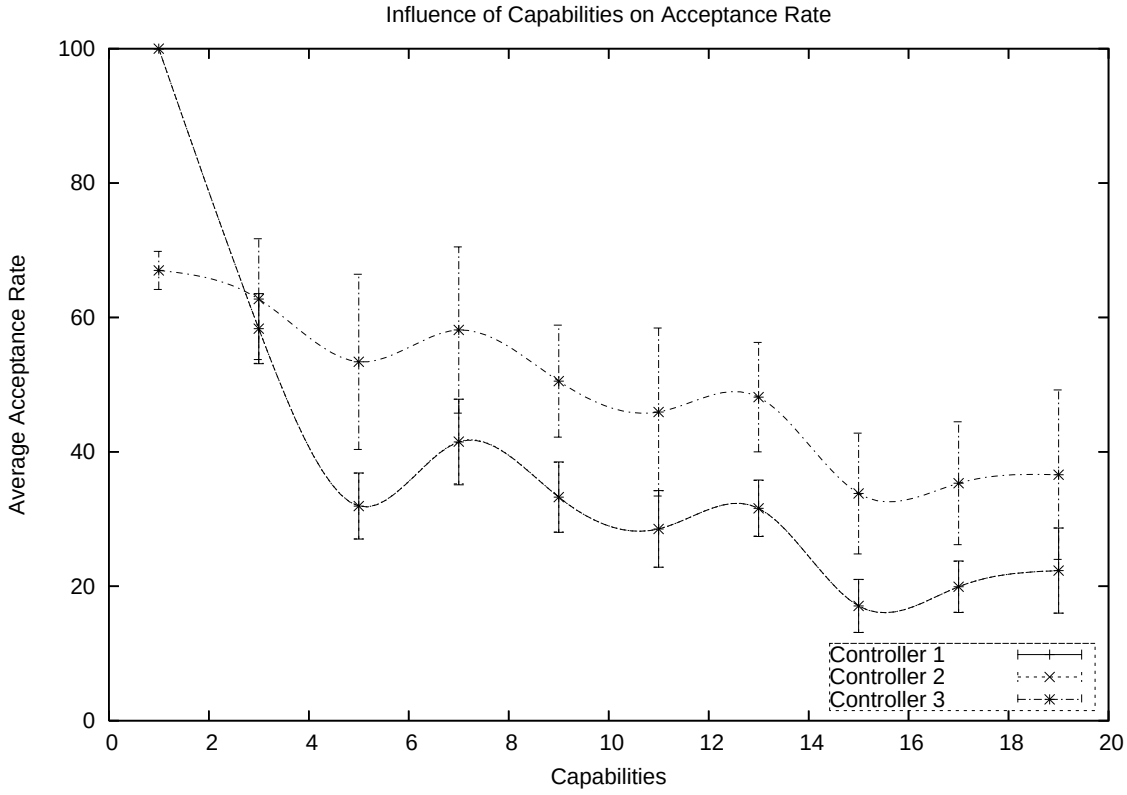


Figure 7: Acceptance Rate Performance Measure on Capabilities

plausible since a random phone is less likely to fulfil a random task as the set of capabilities increases.

Controller 3 (Algorithm F), on the other hand, while showing the same downward trend in its Acceptance Rate, the fluctuation which it also displays is steadier and gradual. This is unlike Controllers 1 and 2 which show a sharp plunge once the capabilities set has size greater then 1.

In addition, other than when the capabilities set has size one, Algorithm F

has a higher rate of assigning tasks than both Algorithm G and Algorithm R. This shows that Algorithm F is more efficient in assigning tasks when the set of capabilities has size greater than one .

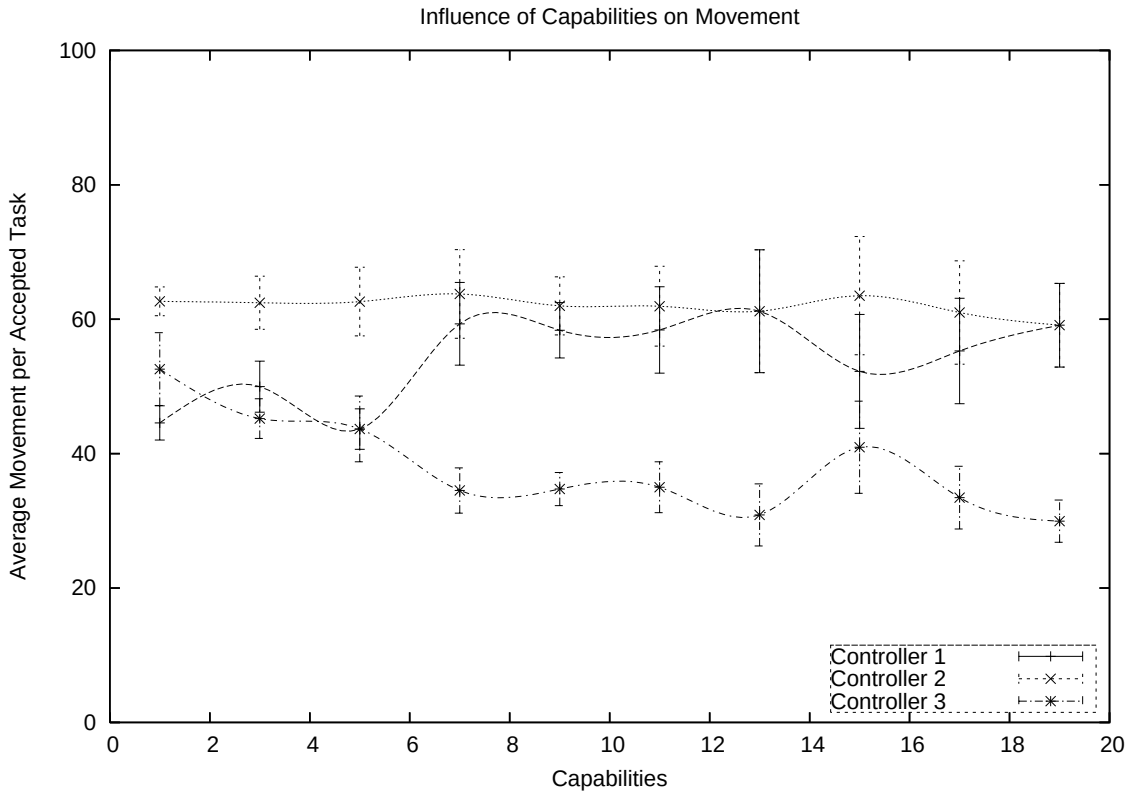


Figure 8: Average Movement Cost Performance Measure on Capabilities

While the Acceptance Rate of tasks presented to the Controllers decreased as the number of capabilities increased, the Average Movement Cost varied among each Controller as shown in Figure 8. Algorithm's R choice of an arbitrary phone resulted in little change in movement across the range of capabilities, however, it had the highest movement rate among all the

Controllers.

Algorithm G, which choose the closest phone available for the current task without consideration of any other factors, resulted in a general increase in the movement cost as the number of capabilities increased. This can be explained by the fact that as the capabilities set increased, the probability of finding a phone which met the feasibility criteria (page 10) of the arriving task decreased. If a phone was found that did meet the criteria, the phone may have had to travel a far distance to execute the task since the subset of phones from which to choose the closest one was reduced.

Controller 3 proved to be also the most efficient of the three Controllers, since as the number of capabilities increased, Controller 3 (Algorithm F) was the only one to show a general decline in the movement cost. While there was some fluctuation, this was not the trend but an exception in certain intervals. By looking towards future cost assignments, Controller 3 was able to successfully assign phones to tasks in a manner that reduced the distance traveled for future tasks as they arrived.

5.2 Task Arrival Time

The inter-task time Δ at which tasks are presented to the system for possible acceptance and assignment was varied with values ranging from every

3600 seconds to every (10×3600) seconds, with increases in increments of 3600 seconds, while the other parameters to the simulator were kept at the following values:

$$m = |\mathbb{C}| = 2$$

The set of phones $\mathbb{P}$ has a cardinality $n = 2$. Each phone $i \in \mathbb{P}$ has the following finite values:

$$\begin{aligned} |C_i| &= 1 \\ \upsilon_i &= 1.0 \div 3600 \end{aligned}$$

Each task T_i has the following finite values:

$$\begin{aligned} start_i &= N(a_i + L, 1hr),\ where\ L\ is\ (24 \times 3600) \\ end_i &= N(start_i + W, 1hr),\ where\ W\ is\ (4 \times 3600) \\ slack\ (\epsilon) &= 0.8 \\ |R_i| &= 1 \end{aligned}$$

As the time between arriving tasks increases, the rate at which tasks are accepted and successfully assigned to resources (i.e phones) increases. This

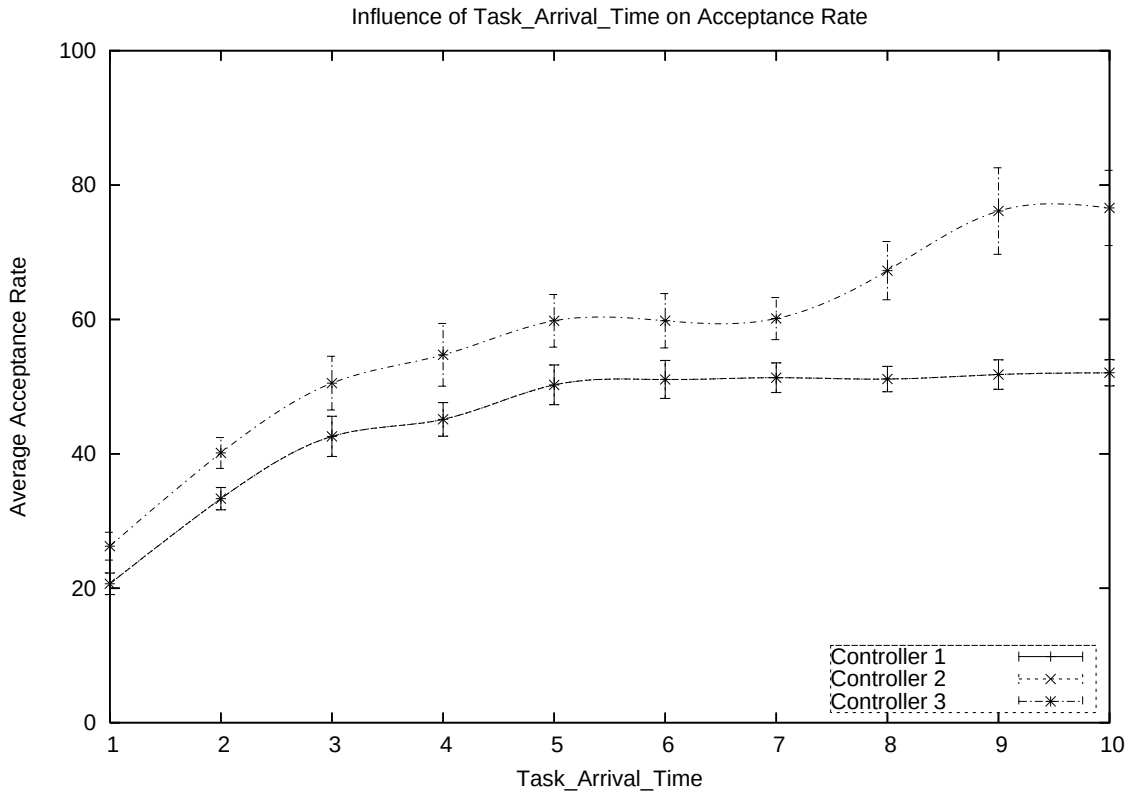


Figure 9: Acceptance Rate Performance Measure on Task Arrival Times

is true for each Controller as Figure 9 shows.

However, Controller 3 (Algorithm F) has a higher Acceptance Rate than Controller 1 (Algorithm G) and Controller 2 (Algorithm R). This signifies that Controller 3 manages its resources (i.e phones) more efficiently than Controllers 1 and 2, thus ensuring it has more phones available that meet the feasibility criteria of each arriving task.

Analysis of the Movement Cost of each Controller against changes in the

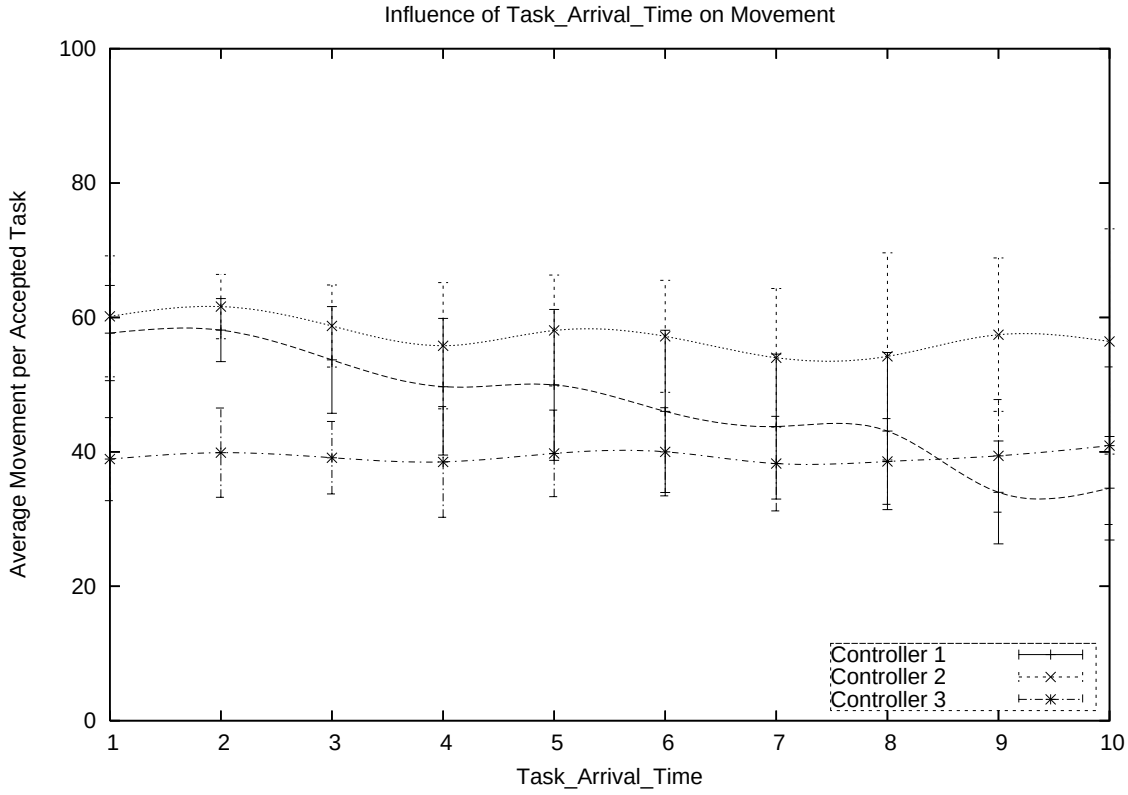


Figure 10: Average Movement Cost Performance Measure on Task Arrival Times

arrival times of tasks (Figure 10) shows a different trend for each Controller.

Controller 2, which implemented Algorithm R for its scheduling algorithm to determine if it accepted or rejected a task has the greatest movement throughout all values of task arrival times, with movement costs somewhat constant at about 60.

Controller 1 (Algorithm G) was the controller with the next greatest movement cost. However this controller had a downward trend in movement cost

as the time between arriving tasks increased. When arriving time between tasks was greatest, Controller 1 had the lowest movement cost of the three Controllers. This implies that Algorithm G operates at optimal efficiency in terms of minimizing movement when the time between which new tasks are presented for acceptance or rejection is high. This can be further explained by the fact that at higher arrival times, all phones available for assignment by the Controller have already completed their tasks, since the duration of any tasks is now less than the time when new tasks arrive. Thus the Controller has its full complement of resources (i.e phones) available, so that the closest phone is always available.

Controller 3 (Algorithm F) displayed a fairly constant movement cost regardless of when a new task arrived and also had the lowest movement cost of the three Controllers for the majority of the arrival times. Again this confirmed that Algorithm F displayed optimal efficiency in managing its resources.

5.3 Task Duration

In analyzing task duration, we look at how changes to slack (ϵ) affected the Acceptance Rate and the Average Movement Cost of the three algorithms. We studied a range of $0.1 \leq \epsilon \leq 1.0$. The other parameters to the simulator were kept at the following values:

$$m = |\mathbb{C}| = 2$$

The set of phones $\mathbb{P}$ has a cardinality of $n = 2$. Each phone $i \in \mathbb{P}$ has the following finite values:

$$\begin{aligned} |C_i| &= 1 \\ \upsilon_i &= 1.0 \div 3600 \end{aligned}$$

Each task T_i has the following finite values:

$$\begin{aligned} a_i &= 3600 \cdot i \\ start_i &= N(a_i + L, 1hr),\ where\ L\ is\ (24 \times 3600) \\ end_i &= N(start_i + W, 1hr),\ where\ W\ is\ (4 \times 3600) \\ |R_i| &= 1 \end{aligned}$$

Since the duration of each task equals ϵW, as ϵ increases, it is expected that the number of tasks accepted by each Controller will decline (Figure 11). This is due to resources being unavailable for longer periods of time.

While all three controllers show a decline in the Acceptance Rates, Controller 3 (Algorithm F) maintains a higher Acceptance Rate over Controller

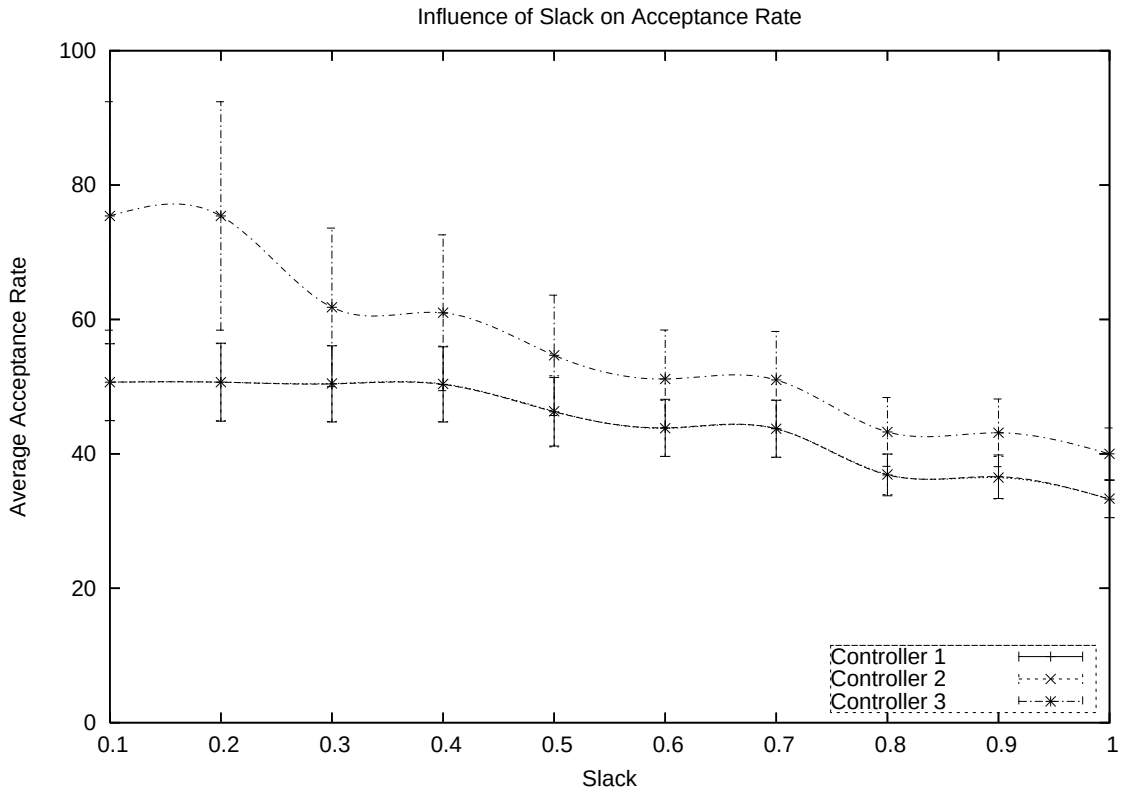


Figure 11: Acceptance Rate Performance Measure on Slack

1 (Algorithm G) and Controller 2 (Algorithm R) by 10%.

Both Controller 1 and Controller 2 exhibit the same Acceptance Rates, due to the same reasoning used to analyze phone Capabilities (Section 5.1). When the duration of arriving tasks is half the window of the task or greater, all three Controllers follow the same path in terms of Acceptance Rate, the difference being Controller 3 has an Acceptance Rate of approximately 10% greater than Controller 1 and Controller 2.

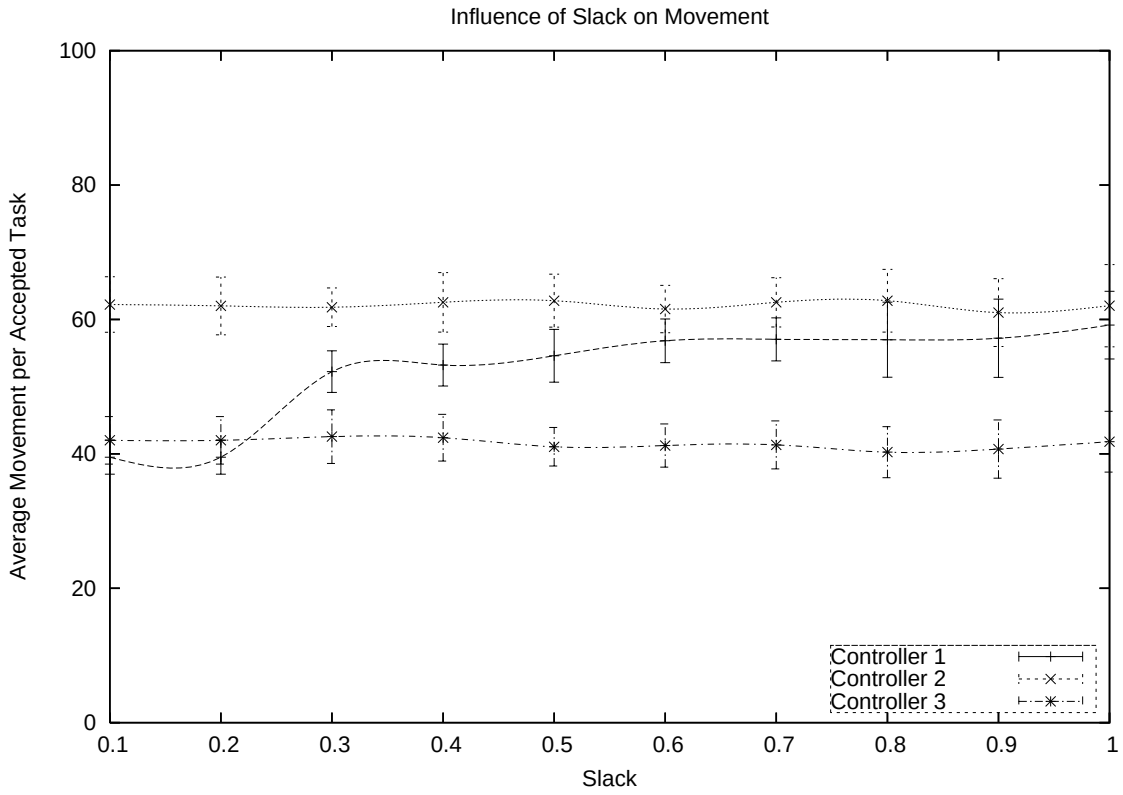


Figure 12: Average Movement Cost Performance Measure on Slack

The trend we expected for the Average Movement Cost was that as the duration of tasks increased, the movement cost would also increase. However as shown in (Figure 12), the only controller that exhibited this trend was Controller 1; both Controller 2 and Controller 3 showed relatively constant rates of movement throughout all changes in slack.

Controller 2 has the highest movement cost of all three Controllers with Controller 3 having the lowest average cost. While slack had little effect on the movement of resources under Controller 3, it was somewhat surprising that

Controller 2 (Algorithm R) which makes its assignment of tasks by randomly choosing a phone which meets the feasibility criteria for the task, had a fairly constant movement cost, while Controller 1 (Algorithm G) which chooses the closest available phone, had a higher movement cost as the duration of arriving tasks increased.

5.4 Resources

In analyzing resources (i.e phones), we looked at how increases in the number of phones available for assignment, affected the Acceptance Rate and the Average Movement Cost of the three algorithms. We studied a range of $1 \leq |\mathbb{P}| \leq 10$. The other parameters to the simulator were kept at the following values:

$$m = |\mathbb{C}| = 2$$

Each phone $i \in \mathbb{P}$ has the following finite values:

$$\begin{aligned} |C_i| &= 1 \\ v_i &= 1.0 \div 3600 \end{aligned}$$

Each task T_i has the following finite values:

$$
\begin{aligned}
a_i &= 3600 \cdot i \\
start_i &= N(a_i + L, 1hr),\ where\ L\ is\ (24 \times 3600) \\
end_i &= N(start_i + W, 1hr),\ where\ W\ is\ (4 \times 3600) \\
slack\ (\epsilon) &= 0.8 \\
|R_i| &= 1
\end{aligned}
$$

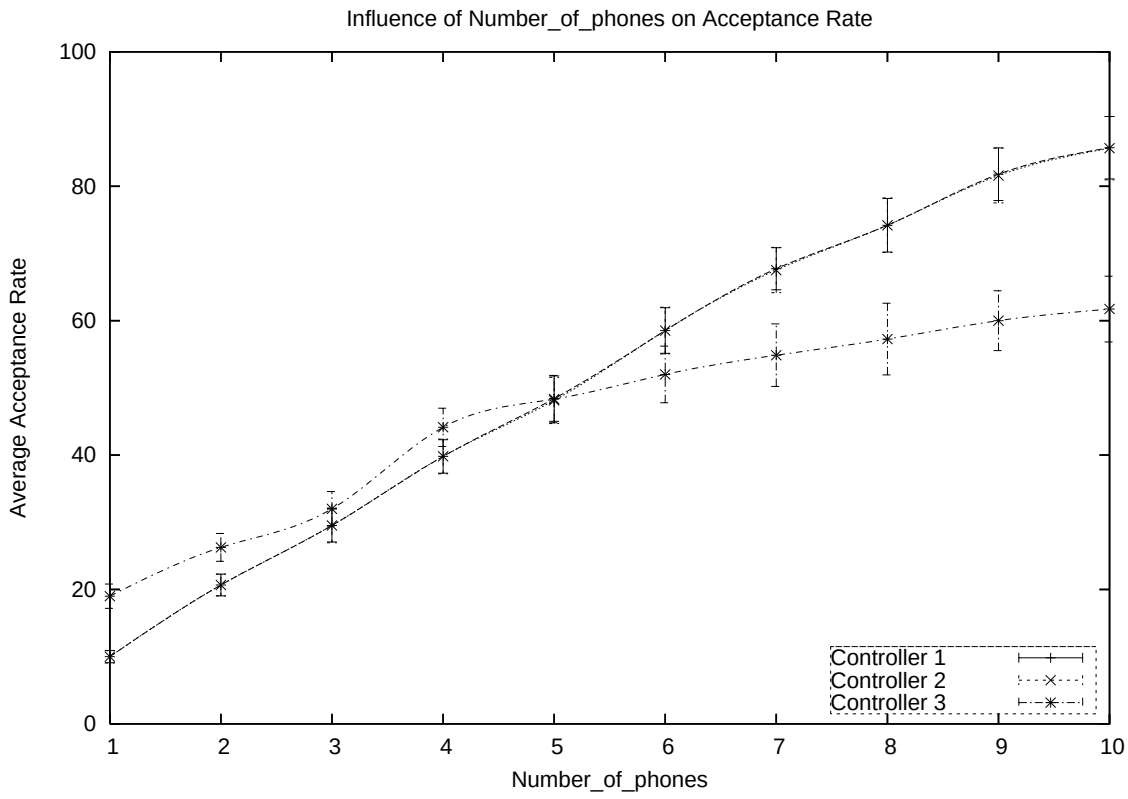


Figure 13: Acceptance Rate Performance Measure on Available Resources

The expected general trend of the Acceptance Rate due to increases in the number of phones would dictate that the rate will increase as the number of phones increases. The results graphed in Figure 13 support this trend.

A interesting point on the graph is where the number of phones available for assignment is 5. At that point all three Controllers accepted approximately 50% of the tasks presented to them. As the number of capabilities available was two and any phone had a 50% probability of having one of the 2 features and arriving tasks also had a 50% probability of requiring capability, we can conclude that of the 5 phones available, half met the feasibility criteria to execute any task that arrived.

As the number of phones increased, Controller 3 (Algorithm F) was more conservative in its acceptance of tasks and thus had a lower Acceptance Rate than Controller 1 and Controller 2. However, for smaller number of phones, Controller 3 was more effective at assigning a greater number of tasks.

While having more phones resulted in higher Acceptance Rates among the controllers, it would logically follow that the Average Movement Cost would behave inversely, with lower movement as the number of phones increased. However, as Figure 14 shows, the opposite occurred.

The only controller which exhibited any signs of decline was Controller 1, although it was not a significant decline. Controller 3 had the lowest Average

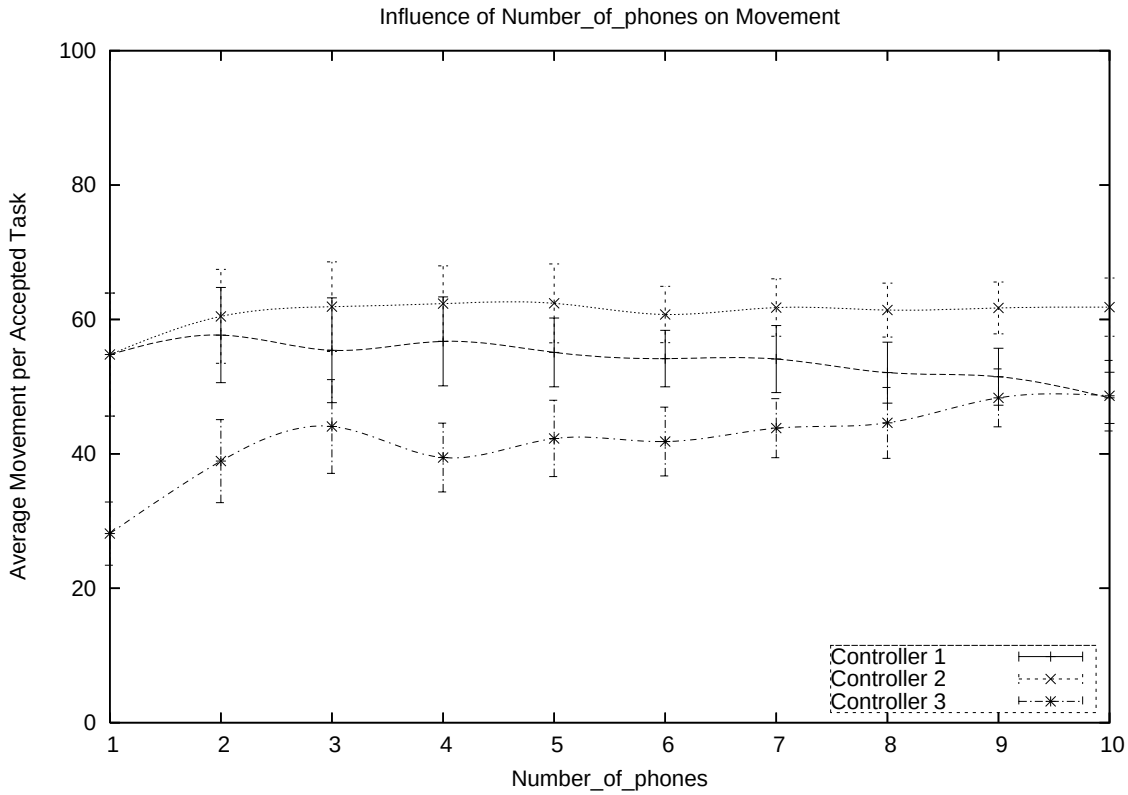


Figure 14: Average Movement Cost Performance Measure on Available Resources

Movement Cost overall but its cost increased as the number of phones also increased. Controller 2 showed a constant movement cost throughout for all values of n.

5.5 Phone Velocity

The maximum velocity v a phone could travel was analyzed over a range $(1 \div 3600) \leq v \leq (5.0 \div 3600)$. The other parameters to the simulator were kept at the following values:

$$m = |\mathbb{C}| = 2$$

The set of phones $\mathbb{P}$ has a cardinality of $n = 2$. Each phone $i \in \mathbb{P}$ has the following finite values:

$$|C_i| = 1$$

Each task T_i has the following finite values:

$$\begin{aligned} a_i &= 3600 \cdot i \\ start_i &= N(a_i + L, 1hr),\ where\ L\ is\ (24 \times 3600) \\ end_i &= N(start_i + W, 1hr),\ where\ W\ is\ (4 \times 3600) \\ slack\ (\epsilon) &= 0.8 \\ |R_i| &= 1 \end{aligned}$$

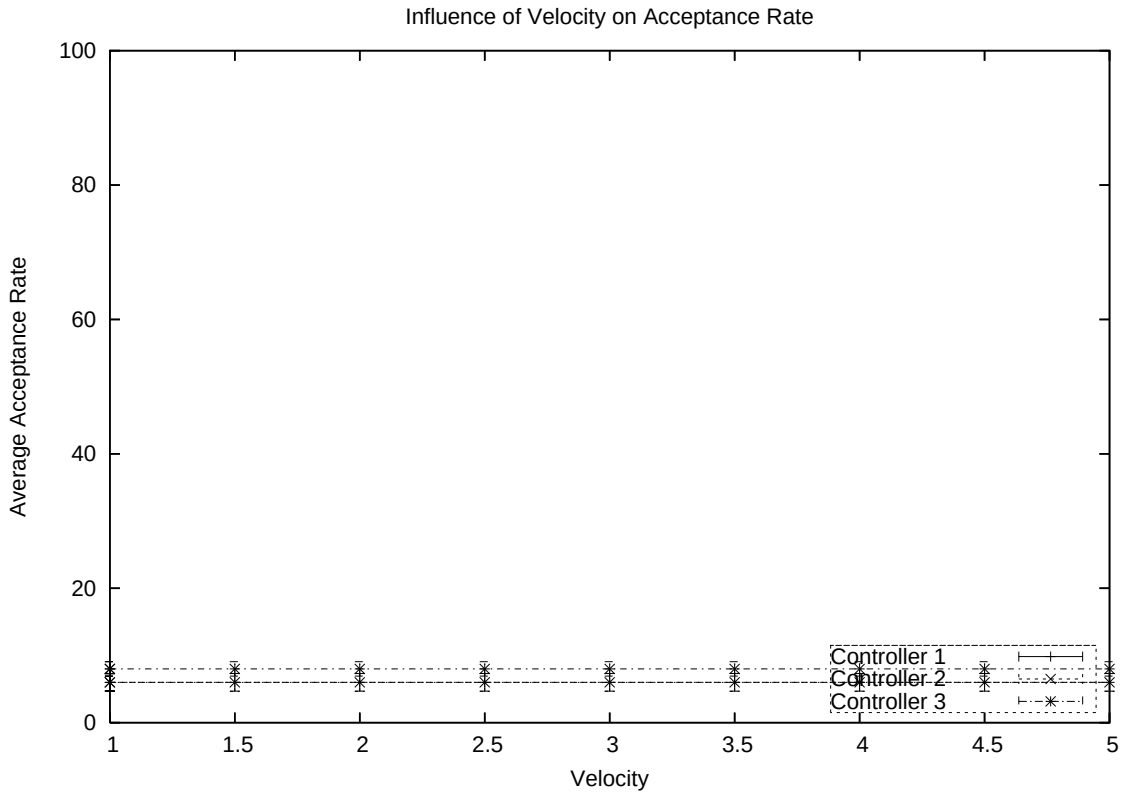


Figure 15: Acceptance Rate Performance Measure on Phone Velocity

As shown in Figure 15, the maximum velocity a phone could have had little effect on the Acceptance Rate of the three Controllers. In addition, it also limited the rate to below 20% with Controller 3 having a slightly higher Acceptance Rate over Controller 2 and Controller 3.

Similar to results for Acceptance Rate in Figure 15, the Average Movement Costs are relatively constant for all tested values of velocity (Figure 16).

Controller 3 had the lowest movement cost, while Controller 2 (Algorithm

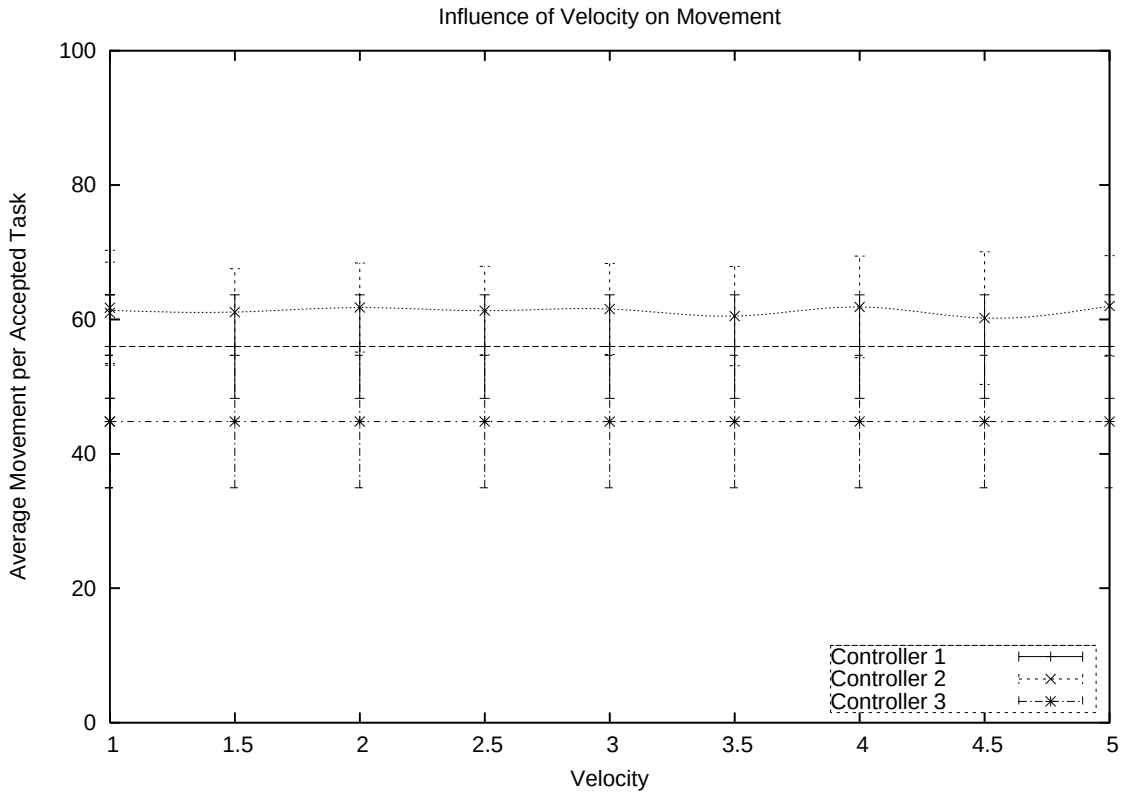


Figure 16: Average Movement Cost Performance Measure on Phone Velocity

R) had the highest movement cost and was the only controller that showed some deviation as values of velocity changed.

6 Real System Implementation

6.1 The Architectural Overview

The architecture of the Real System is shown in Figure 17. It consists of several components, each implemented using different software packages.

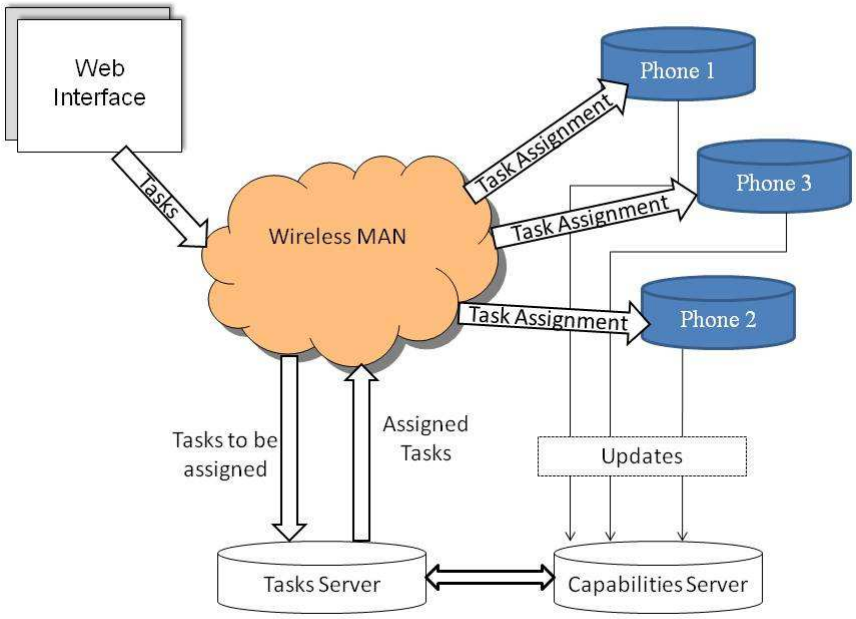


Figure 17: Architecture of the Real System

1. Phones

   The phone application is designed using a Mobile Information Device Profile (MIDP), which is a part of the Java 2 Platform, Mobile Edition

(J2ME). The services provided by the phone are:

(a) Register team members

Prior to receiving assignments from the Controller, each response team member must submit to the Capabilities server user information and technical expertise. From within the application, the member selects the Register phone option and enters pertinent data such as first and last name, phone number and their current location. The register module presents to the team member a list of the capabilities required by the system, which is retrieved dynamically from the Capabilities Server. The team member selects all options which they are capable of executing and submits the form to the server.

The phone uses the cellular network to make connection to the Capabilities Server where it makes a new entry for each new registrant and stores the information submitted. A team member can also update their information by accessing the same form and resubmitting updated data.

(b) Update team member's location

On a periodic basis, the application makes a connection to the Controller and transmits its location coordinates via GPS.

(c) Receive task assignments

The application periodically polls the Tasks Server for any assign-

ments the Controller allocated to the team. If any new assignments are found on the Tasks Server, a copy is delivered to the phone and stored in a queue. When a team has completed a prior task, the team member selects the retrieved tasks option which displays the first task in the queue and deletes it from the queue. If the phone is turned off or the application is aborted, the queue is re-populated the next time the phone makes a connection to the server. Tasks are marked as completed on the server once the team indicates it has been completed.

2. Capabilities Server

   A MySQL database is the interface for the server side of the application. The capabilities server stores a list of required functions that a response team can register and indicate their ability to execute, as well as the capabilities of each registered team. The two tables that make up the capabilities server are:

   (a) Capability

   A one-to-many relational table whose fields are the phoneID of the team and the functionID populate the capability table. For every capability submitted via the phone by the team, an entry is made in the table using the primary key assigned to the phone and the unique identifier of each function.

   (b) Function

This stores all available functions which users can register to perform. The ID of the function is used as the primary key of the table.

3. Tasks Server

   The task server handles the submission of tasks to be completed and the assigned tasks and the phone assigned to each task. The tables used to store this information are:

   (a) Task

   This stores the tasks submitted to the system. An automatically generated ID number is used as the primary key.

   (b) Assignment

   This stores the TaskID from the task table along with the PhoneID of the phone assigned to the task by the scheduling algorithm. This table contains the primary keys from the phone and task tables as primary keys.

   The relationship between the tables in the MySQL database is shown in Figure 18.

4. Controller

   The controller is responsible for scheduling tasks by submitting them to the most efficient resource and assigning the task to relevant phone.

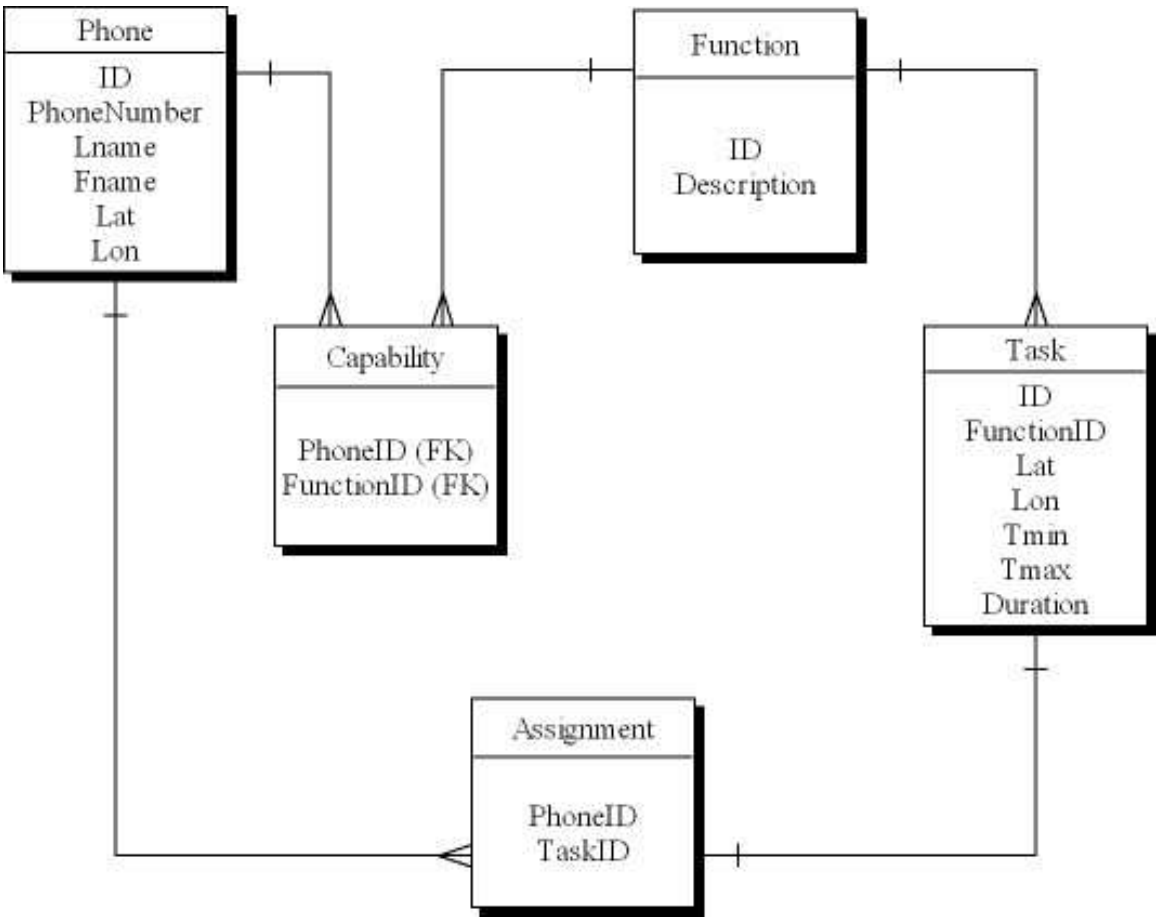


Figure 18: Relationship Diagram of MySQL Database

6.2 Data Flow

1. Capability Updates

   (a) Phone

   A phone can register new capabilities with the server or delete already registered capabilities. To do this the phone makes a connection to the capabilities server and queries the capabilities table

for all entries in the database tied to the phoneID of the phone. This information is shown to the user in a graphical interface and the user is given the option to add or delete functions. The information submitted to the server is saved in the table. Updates can be made to information already stored as well as adding new features which creates new entries in the table. On successful updating of the information, the server sends an acknowledgment to the phone.

2. Task Notifications Updates

   The phone is updated with new tasks asynchronously unknown to the user. The phone periodically makes a connection to the task server and queries the assignment table for any tasks for the phone which have not been completed. If any tasks are found, a copy is placed in a queue on the phone. When the user retrieves a task, the next assigned task is shown on the display of the phone indicating the location of the task, the time it should start and its duration along with the capabilities the task requires.

3. Task Assignments

   The tasks are entered via a web interface by the operator "at the control." All the information related to each task is entered. Importantly the time to start, deadline for completion along with the capabilities required for the task are entered directly into the task server.

The assigned tasks are then placed in the "Scheduler" which then tries to assign a task to a "phone." If the task is successfully scheduled the operator will be notified and the phone that is assigned to the task will be shown. The operator is notified when tasks are not successfully assigned to a phone.

Once the schedule selects a phone for the job, the information will be sent to the "queue" for the phone. When the phone polls the server, it will be notified of the assigned task.

6.3 Implementation

1. Capabilities/Tasks Server

   A MySQL database with the schema shown in Figure 18 was created to store the dynamic information submitted by the mobile phones within the system as well as submitted tasks and task assignments.

2. Controller

   The Controller is incorporated within PHP scripts used to manage tasks submitted to the Tasks Server which then computes optimal assignments of the tasks submitted based on the data stored in the Capabilities Server.

3. Phones

   The Java J2ME MIDP application written to implement the real system

was done using the Netbeans 6.1 Platform and the Mobility Pack and tested using the Sun Wireless Toolkit and the Nokia Java SDK S60 emulator for Nokia mobile devices.

- MIDP

  MIDP is part of the Java Platform, Micro Edition (J2ME) framework, and sits on top of the Connected Limited Device Configuration which is a set of lower level programming interfaces that allows applications to be developed for devices with limited resources such as mobile phones.

- J2ME

  The Java Platform, Micro Edition (J2ME) is a specification of a subset of the Java platform aimed at providing a collection of Java APIs for the development of software for extremely small and resource-constrained devices such as cell phones and PDAs

- Netbeans

  Netbeans 6.1 is a reusable framework that simplifies the development of Java and other applications. It allows applications to be developed from a set of Java modules that contain Java classes that interact with the Open APIs in Netbeans.

- Nokia Java SDK S60 Emulator

  The Nokia Java SDK S60 emulator is the primary development and debugging tool for S60 Symbian OS application development.

It provides PC keyboard combinations for accessing special testing and logging functionality and also for mimicking the hardware keys of an actual mobile device.

6.4 Operation

The execution of the model shown in Figure 17 is shown in the following screen shots, specifically the web interface and the emulated phone application.

1. Web Interface

   - Submitting Capabilities

     Figure 19 shows the web interface module which is used to submit new capabilities to the Capabilities server. The Capabilities Server is queried and a list of the existing capabilities are dynamically retrieved and presented to the user. While editing or deleting of existing capabilities is not part of the current framework, it is part of future enhancements to be made.

   - Submitting Tasks

     Submission of tasks is made through the web form shown in Figure 20. The user enters the location of the tasks as well as the start and end times - which constitute the window W - for the tasks and

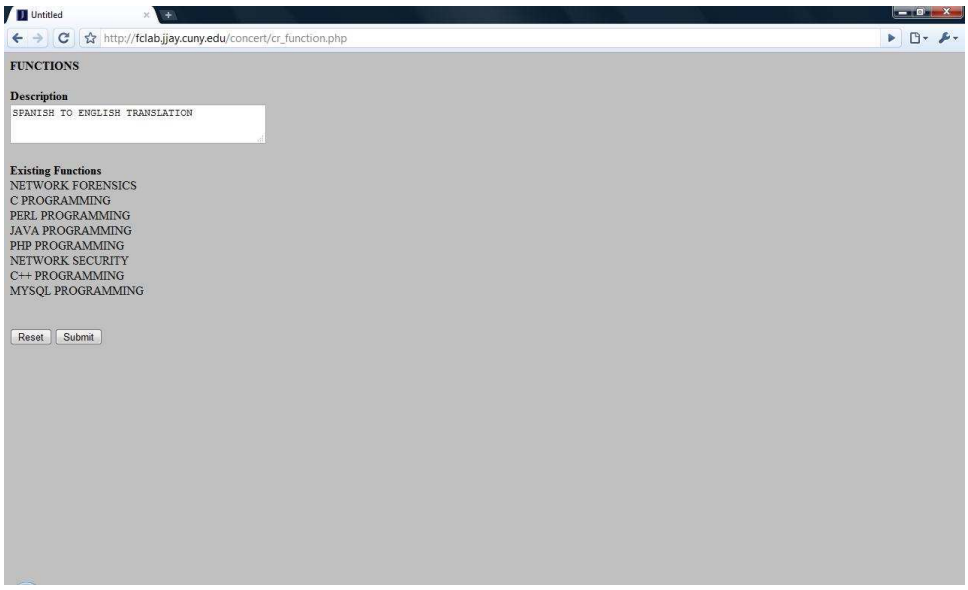


Figure 19: Web Interface to submit capabilities to the system

selects the requirements needed to perform the task from the capabilities set. The estimated duration of the tasks is also entered. The completed form is sent to the Tasks Server where the Controller computes the optimal assignment for the task and sends notification to the resource (i.e phone) that meets the feasibility criteria of the task.

2. Emulation of Phone Application

   The following screen shots (Figures 21 - 26) display the working prototype of the mobile phone application which will be used by response teams in the field. An emulator has been used to simulate a real working mobile phone which will produce similar output.

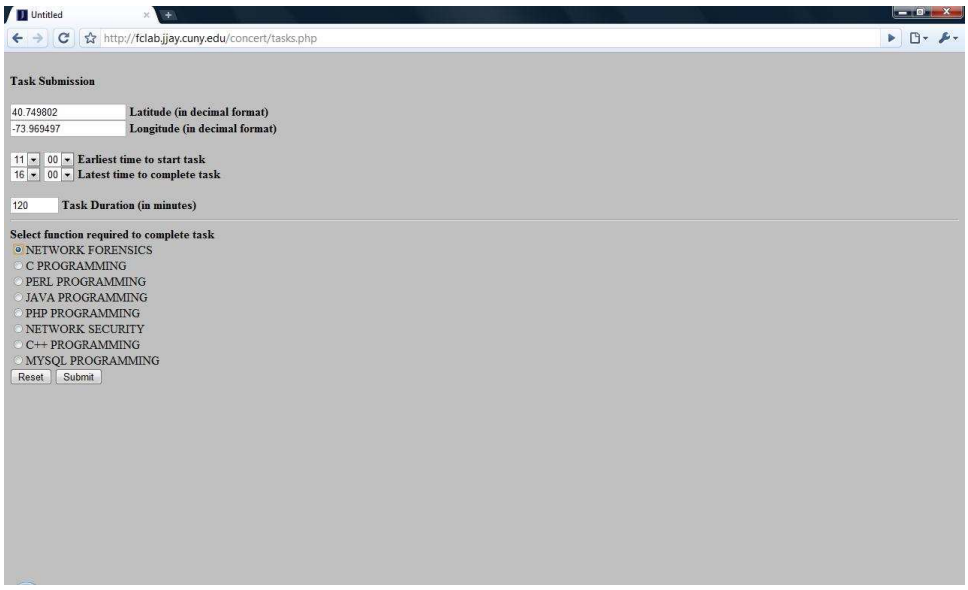


Figure 20: Web Interface used to submit tasks

- Start Up

  When a first responder first logs into the application, they are required to enter a unique phone ID, which identifies them in the system. This ID is the only method of identifying a responder within the network. Figures 21 and 22 show the screens the responder is presented on logging into the system.

- Registration

  After successfully logging into the system, the first responder must register their information with the Capabilities server if they are new members to the network. The Registration module, shown in Figures 23 and 24 can also be used by a responder to update their capabilities set, should they gain additional capabilities through

Figure 21: Start-up Screen of Phone Application

training or otherwise, or if the network creates new capabilities it requires for new tasks which the responder may possess. The current prototype does not allow a responder to delete any capabilities which they have already registered with the Capabilities

Figure 22: Start-up Screen of Phone Application

server but will be a part of future models.

- Tasks Notification

  The mobile application continuously polls the Tasks Server unknown to the responder for any tasks which the Controller has

Figure 23: Registering Interface

assigned to the ID of the phone. If any tasks have been assigned to the phone, the information regarding the tasks is saved in a queue on the phone. The responder uses the Retrieve Task option on the phone to check the queue for any existing tasks. If a task

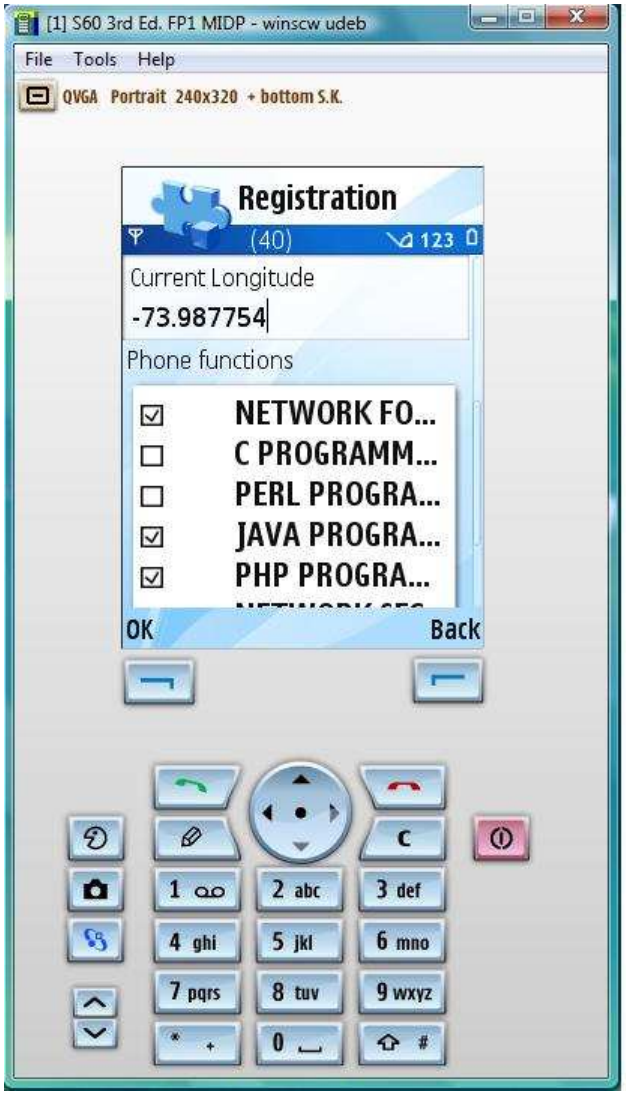


Figure 24: Registering Interface cont'd

is waiting to be completed by the responder, the information regarding the task is displayed on the phone, as shown in Figures 25 and 26.

Figure 25: Option to retrieve tasks

Figure 26: Information on task assigned by Controller

7 Future

CONCERT is still in its infancy and as such, future enhancements are planned to improve its functionality and make it more secure. These enhancements will eventually be distributed throughout the entire CONCERT system.

7.1 Simulation Framework

The simulation framework is an integral part of the CONCERT system as it allows testing of prototypes before actual implementation in the real system. The following features will be added to the framework:

- Include the ability to have cooperation of phones on tasks by combining their capabilities to meet the feasibility criteria of a task.
- Assign incoming tasks based on a rank placed on the task. This would result in higher priority tasks being assigned ahead of other tasks.
- Ability to instruct phones to stop tasks they are currently executing or put them on hold if tasks of a higher priority needs to be completed.
- Allow tasks to be reassigned. This would be the case if there are other phones available that meet the feasibility criteria.

- Ability to have tasks that are unassigned to be checked for possible assignment provided the window has not passed for assignment.

7.2 Scheduling Algorithms

- Develop additional scheduling algorithms taking into consideration the priority of tasks.
- Implement the use of Geocoding, the process of finding associated latitude and longitude coordinates from street addresses, to notify response teams of the accurate location of a crisis.
- Account for movement of phones when not assigned to tasks so that their location can be periodically updated since they may have changed from where they were the last time they performed a task.

7.3 Web Interface

Although meeting the needs of the current system, I intend to enhance the web interface to include the following features:

- Allow capabilities which have already been submitted to the Capabilities Server to be modified or deleted from the system. This however involves some logistical planning since only those capabilities which

have not been part of the requirements for tasks will be allowed to be modified or deleted to ensure integrity of the stored data.

- Allow tasks to be ranked according to priority.

7.4 Phone Application

The MIDP phone application currently in use will see a number of enhancements to its operation:

- Automatically update responders when new capabilities are added to the system via the web interface.
- Allow responders to delete capabilities from their capabilities set.
- Automatically notify responders when new tasks have been assigned to them, either via an alert or overriding any current module on the phone and displaying the task information. This is subject to research on the feasibility and capabilities of the software.
- Use General Packet Radio Services (GPRS) Core Network, Wi-Fi, and satellite communication to ensure that there is constant communication between the phone and the central server.
- Enhance the security of the application to ensure only authorized personnel are able to communicate with the servers.

Index

References

[1] Jonathan Knudsen. *Kicking Butt with MIDP and MSA: Creating Great Mobile Applications (Java Series)*. Prentice Hall PTR, 2008.

[2] Jonathan Knudsen Sing Li. *Beginning J2ME: From Novice to Professional*. Apress, 2005.

8 Appendix 1: The Simulation Framework Code Listing

9 Appendix 2: The CONCERT Simulator Code Listing

10 Appendix 3: The CONCERT Server Code Listing

11 Appendix 4: CONCERT Phone Software Code Listing