A Distributed Communication Architecture for Dynamic Multiagent Systems

Kyle Hollins Wray
University of Massachusetts, Amherst
Amherst, MA 01003, USA
wray@cs.umass.edu

Benjamin B. Thompson
The Pennsylvania State University
State College, PA 16802 USA
bbt10@psu.edu

Abstract

We investigate the problem of creating a robust, rapidly converging, Distributed Communication Architecture (DCA) for the domain of low bandwidth, single channel Multiagent Systems (MAS) in which agents may drop in and out of communication without prior notification. There are only three capability-based assumptions made by the algorithm: 1) agents can classify a signal's message as either $noise$, $silence$, or $clarity$, 2) agents can classify their own messages, and 3) agents can understand one another to some degree. The final structure allows agents to communicate in a round-robin manner without any centralized or hierarchical control. We evaluate DCA's the convergence rate through four distinct experiments, including both a worst-case scenario that consists of all agents starting simultaneously and a more common-case scenario in which agents offset their starting times. We examine effective throughput as the average number of clearly sent messages in a cycle to determine the amount of information successfully communicated. Lastly, we emulate situations found in problems with moving agents to show that agents who incorporate local observations can improve both their convergence rates and throughput.

Introduction

Multiagent Systems (MAS) problems often benefit greatly from direct communication. MAS communication research is focused on both the higher-level structure of *what* is sent or their messages' internal representation (Parker 1998; Gerkey and Mataric 2002) and the lower-level structure for *how* agents can communicate, especially in communication-limited domains (Xuan, Lesser, and Zilberstein 2001; Yanco and Stein 1993). We focus on this lower-level, and solve a recurring problem. In a wide range of real-world scenarios, agents are limited low bandwidth communication on a single channel; e.g., the RoboCup domain (Kitano et al. 1997; Stone and Veloso 1999), cooperative task-based robotics (Yanco and Stein 1993; Vidal et al. 2002) and time slot allocation in sensor networks (Rhee et al. 2009; Lin et al. 2011). Designing these architectures for communication can often encounter problems given the constraint that agents cannot communicate simultaneously, the number of agents is not known *a priori*, and communication must be sustained over an extended period of time. The Distributed Communication Architecture (DCA) algorithm solves this problem without any centralized control.

The broadcast channel problem is a commonly used problem in evaluating algorithms to solve Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) (Bernstein et al. 2002). It consists of a set of agents seek to collaboratively share a single channel in a broadcast setting without any means of centralized control (Hansen, Bernstein, and Zilberstein 2004; Ooi and Wornell 1996). ALOHA and its variants are the most popular solution in an applied setting (Roberts 1975; Abramson 1977; Jones et al. 2001). It operates by broadcasting a message, checking if there was a collision, and rebroadcasting the message after waiting for a random amount of time. A related problem in anti-collision protocols for Radio-Frequency Identification (RFID) also explores varying the number of slots in a communication cycle and an unknown number of agents; however, the algorithms are centralized and do not support sustained communication (La Porta, Maselli, and Petrioli 2011; Cha and Kim 2006). DCA leverages the basic idea of ALOHA, but designs round-robin pattern in which agents reserve time slots for extended communication.

Our main contribution is an adaptive communication pattern be constructed in a decentralized manner with a group of agents, solving a kind of broadcast channel problem. Additionally, we show how to include high-level knowledge into this low-level process to boost performance. To demonstrate this, we look at a commonly found situation of agents with a limited field-of-view. We evaluate our algorithm with four experiments, each evaluating between 1 and 50 communicating agents. Our metrics include both the convergence rate to a round-robin pattern, in addition to the throughput (clear messages sent). We demonstrate that DCA is an effective solution to the broadcast channel problem, which brings ideas from the networking community into the artificial intelligence and multiagent communities.

The next section provides a brief overview of previous work found in communication formation, with the subsequent section stating the problem statement. We then delve into the full algorithms description with pseudocode followed by four experiments that each evaluate between 1 and 50 communicating agents. Finally, we discusses our results and conclusions.

Problem Statement

To remain as general as possible we enforce only minimal requirements. The problem domain is as follows:

1. A single channel, low bandwidth communication
2. Agents require fair round-robin communication
3. The number of agents is unknown *a priori*
4. Agents may leave without prior notification

Given this particular problem domain, we assume that agents have the following capabilities:

1. Agents can classify a message as *noise*, *silence*, or *clarity*
2. Agents can classify their own messages
3. Agents are able to understand one another

With these simple assumptions which are easily found in many problem domains, we describe an algorithm to create an ad hoc round-robin communication architecture.

Communication Algorithm

The general idea is to make a system for the unknown quantity of n agents to each settle on a time slot index that is exclusively held by them. An agent probabilistically selects slots until it finds one without noise, at which point the agent settles on that slot. Internal to all agents, this new settled slot is shifted down to the settled slots region.

We begin by defining some terms. **States** are the current function an agent is running as part of listening in the communication architecture. **Slots** refer to the time-delimited location on the single channel in which an agent may speak. Also a component of an agent's recognition of the signal's $noise$, $silence$, or $clarity$. **Varying slots**, denoted as the list (ordered multiset) $\mathcal{S}_V$, are the slots that do not have a definitive agent. Agents attempt to send their message on these slots. **Settled slots**, denoted as the list (ordered multiset) $\mathcal{S}_S$, are the slots agents have successfully claimed. Agents who have settled may send a message on their respective settled slot. **Cycles**, denoted as the list (ordered multiset) $\mathcal{S} = \mathcal{S}_S \cup \mathcal{S}_V$, are the concatenation of the settled slots followed by the varying slots. Therefore, the full cycle length varies in proportion to the size of $\mathcal{S}_S$ and $\mathcal{S}_V$.

The algorithm is structured into three states (see Figure 1) that each agent goes through individually. The first **learning** state only focuses on listening to the current communications until it fully comprehends the pattern. Next, the **slot selection** state attempts to find a slot without another agent speaking. Once that is found, the **settled** state allows this agent to continue to send messages during the slot it found.

Base Functions

To allow this to work for an arbitrary MAS, we need to define the two interface functions for listening and speaking (Algorithms 1 and 2). They handle the state function calls and decide whether the agent should attempt to speak or not. We state them here for completeness, but are fairly straight forward. We define the $\mathcal{X}$ variable as the current internal state of an agent, visually described in Figure 1. Agents maintain the current slot $s_c \in \mathcal{S}$ (with index c), the slot that

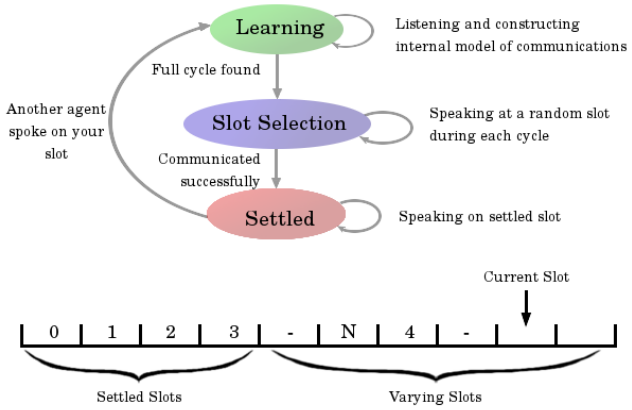

Figure 1: The high level state transition diagram for the communication algorithm (top). A visual example of the communication cycle structure (bottom) with numeric values showing the settled slot ID's, "N" representing a noisy signal, and "-" being a silent signal.

the agent has selected $s_\alpha \in \mathcal{S}$ (with index α), and the number of previously settled agents that dropped out of communication during any given cycle n_d. Each agent is also managing the slots (cycle) list $\mathcal{S}$. This list describes both what was previously heard and actual communication structure. Therefore, its elements are the actual signal messages, i.e., indices communicated by other agents.

We assume lists are zero-indexed and any use of a slot index refers to the index in the full slots list, e.g., $s_i \in \mathcal{S}$ is the ith position, so if $s_i \in \mathcal{S}_V$, then i is still the index in $\mathcal{S}$. This notation allowed us to condense the algorithm into a succinct representation. We also use the function $type(s)$, $s \in \mathcal{S}$, as the type of message. Finally, we let the function $initialize()$ set all variables to 0 and assign all lists to $\emptyset$.

Algorithm 1 listen: Given a signal (message), update the internal state of an agent's communications architecture.

Require: ω: the new signal
1: **if** $\mathcal{X}$ = "learning" **then**
2: $learning(\omega)$
3: **else if** $\mathcal{X}$ = "slot selection" **then**
4: $slot\_selection(\omega)$
5: **else if** $\mathcal{X}$ = "settled" **then**
6: $settled(\omega)$
7: **end if**

Note that an additional feature of the algorithm is that it automatically generates unique indices for agents in a distributed manner.

The Learning State

The learning state's primary purpose is to accurately find the number of settled slots and varying slots (Algorithm 3). An agent that is in the learning state holds two primary functions. The first purpose is to check if anyone is even com-

Algorithm 2 speak: Based on the current internal state, determine whether this agent should speak. If so, return their slot index to be sent as a message in a signal.

1: **if** $c = \alpha$ **and** $\mathcal{X} \neq$ "learning" **then**
2: **return** $\alpha + 1 - n_d$
3: **else**
4: **return** $\emptyset$
5: **end if**

municating at all (lines 3-6). The algorithm waits an initial predefined constant iterations for some form of communication denoted by T_{max}, which we assume is $T_{max} > n$. If nothing is heard, then it moves into the slot selection state.

The second purpose for the learning state is to develop an internal model of the communication pattern. The algorithm waits until it hears the lowest communicated ID twice with only clarity and silence in between. Once that interval is heard, the number of clear slots (minus the repeated ID) is a settled size. From this the current slot and remaining information can be inferred. Finally, the first communication attempt after switching states will follow Equation 1.

This waiting behavior for settled slots benefits the system as a whole because it prevents the addition of more agents while a group of agents in the slot selection state are vying for a settled slot. Adding more agents would increase time not only from having more agents competing, but because the number of varying slots would have to grow to accommodate the new ones. Each increase to the number of varying slots means a full cycle must occur with only noise and clear signals, as is described in the next section.

The Slot Selection State

In the slot selection state, the agent randomly attempts some of the varying slots to see if one slot might be free (Algorithm 4). The random selection of a new varying slot is done by Equation 1. Let $\mathcal{U}(a, b)$, $a > b$, $a, b \in \mathbb{N}$, refer to a discrete uniform random variable on the interval $[a, b)$.

$$\alpha \leftarrow |\mathcal{S}_S| + \mathcal{U}(0, |\mathcal{S}_V|) \tag{1}$$

The slot selection state has the agent attempt to communicate its message without any other agent doing so at the same time. Each cycle now has the settled agents speaking during the settled slots section followed by the varying slots section whereby this and other agents try communicating on one of the slots. On success, it changes its state to settled with the $update\_slots$ function managing the shift to the settled slots section. On failure, this state randomly assigns a new slot when the cycle ends following Equation 1.

The Settled State

When an agent is in the settled state, the only feature it must look for are message collisions on its slot, and also keeps track of any new signals (Algorithm 5).

Once an agent has settled, it needs to communicate its messages on its turn. Additionally, if another agent sends a message when this agent does, causing a collision, the default behavior is to go back into the learning state. This

Algorithm 3 learning: Learn the communication architecture that other agents are using.

Require: ω: the new signal
1: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\omega\}$
2: $\mathcal{S}^- \leftarrow \{s \in \mathcal{S} : type(s) =$ "silence"$\}$
3: **if** $|\mathcal{S}| > T_{max}$ **and** $|\mathcal{S}| = |\mathcal{S}^-|$ **then**
4: $initialize()$
5: $\mathcal{S}_V \leftarrow \{0\}$
6: $\mathcal{X} \leftarrow$ "slot selection"
7: **else if** $type(\omega) =$ "noise" **then**
8: $initialize()$
9: **else if** $type(\omega) =$ "clarity" **and** $|\mathcal{S}| > 1$ **then**
10: $x \leftarrow \min\{i \in \{1, \ldots, |\mathcal{S}|\} : s_i = \omega, s_i \in \mathcal{S}\} \cup \{\infty\}$
11: **if** $x = |\mathcal{S}| - 1$ **or** $x = \infty$ **then**
12: **return**
13: **end if**
14: $\mathcal{S}^- \leftarrow \{s_i \in \mathcal{S} : i \geq x$ **and** $type(s) =$ "silence"$\}$
15: $\mathcal{S}^N \leftarrow \{s_j \in \mathcal{S} : j \geq x$ **and** $type(s) =$ "noise"$\}$
16: **if** $|\mathcal{S}^N| \neq 0$ **or** $|\mathcal{S}^-| \neq 1$ **then**
17: $initialize()$
18: **return**
19: **end if**
20: $\mathcal{S} \leftarrow \{s_x, \ldots, s_{|\mathcal{S}|-2}\}$
21: $i \leftarrow \min\{i \in \{1, \ldots, |\mathcal{S}|\} : s_i = 1, s_i \in \mathcal{S}\}$
22: $\mathcal{S} \leftarrow \{s_i, \ldots, s_{|\mathcal{S}|-1}, s_0, \ldots, s_{i-1}\}$
23: $n_d \leftarrow |\{s_i \in \mathcal{S} : i \leq |\mathcal{S}_S|$ **and** $type(s_i) =$ "silence"$\}|$
24: $\mathcal{X} \leftarrow$ "slot selection"
25: $\alpha \leftarrow |\mathcal{S}| - 1$
26: $update(\omega)$
27: **end if**

simple check allows for two groups of agents to merge their communications together through resetting their internal states. It serves as a kind of error handler for agents who might be subject to environmental noise and need to relearn the group's pattern. One might also remove the check in a scenario such as robot soccer, given the opposing team could communicate and throw off the algorithm.

Slot Organization

We must also include the slot management algorithm for the internal slot model of an agent (Algorithm 6). It performs the necessary adding/removing from the slots list $\mathcal{S}$.

During normal time increments, this handles adding mes-

Algorithm 4 slot_selection: Try to find a slot that no one else is using.

Require: ω: the new signal
1: **if** $c = \alpha$ **and** $type(\omega) =$ "clear" **then**
2: $\mathcal{X} \leftarrow$ "settled"
3: **end if**
4: $update\_slots(\omega)$
5: **if** $c = 0$ **and** $\mathcal{X} \neq$ "settled" **then**
6: $\alpha \leftarrow |\mathcal{S}_S| + \mathcal{U}(0, |\mathcal{S}_V|)$
7: **end if**

Algorithm 5 settled: This agent is settled so do not adjust anything unless another agent selects the same slot.

Require: ω: the new signal
if $c = \alpha$ **and** $type(\omega) =$ "noise" **then**
  $initialize()$
  $\mathcal{X} \leftarrow$ "learning"
else
  $update\_slots(signal)$
end if

sage signals to a list and maintaining the current slot. However, at the end of a cycle it realigns the internal slots list by removing agents who have stopped communicating, places newly settled agents to the left side's settled slots, and reevaluates the internal state. It is also responsible for expanding the number of slots, but only when there were no silences in the past cycle.

There is an underlying reason for resizing with no silences. It comes from a combination of the respective states' actions, namely that the slot selection state only attempts to communicate once each cycle. From this one can infer that the maximum number of varying slots allowable is exactly half the number of agents in the slot selection state rounded up, i.e., $\lceil \frac{n}{2} \rceil$. Since it requires a minimum of two agents to cause a message collision and there can be no silences to add a new slot slot. This maximum number of slots prevents the number of varying slots from probabilistically growing to infinity. More importantly, it aids in the learning state's design, which enables new agents to learn a communication pattern.

Inclusion of Local Agent Knowledge

To test how local agent knowledge might be included in a communication method, we present a sample scenario often encountered in MAS. The particular situation is easily found in the RoboCup (Kitano et al. 1997), predator-prey models (Benda, Jagannathan, and Dodhiawalla 1986), and in pursuit-evasion problems (Vidal et al. 2002). For simplicity, we only model a random placement of n agents in a $n \times n$ grid, each with a 180 degree field-of-view in all cells above it. Thus, the size of the grid grows with the number of agents at a simple linear rate. All agents face the same direction and are assumed to be traveling together at the same speed. We use the number of fellow teammates an agent can directly observe as a variable to adjust the varying slot it selects during the slot selection state. For Equation 2, let $m < n$ be the number of visible agents (agents don't observe themselves). Finally, let $\mathcal{N}(\mu, \sigma^2)$ be the normal distribution with mean μ and variance σ^2.

$$\alpha \leftarrow |\mathcal{S}_S| + \left\lfloor \max\left(0, \min\left[|\mathcal{S}_V|, \mathcal{N}(\min(m, |\mathcal{S}_V| - 1), \sqrt{|\mathcal{S}_V|})\right]\right)\right\rfloor \quad (2)$$

This attempts to try a slot that is past the settled slot indices and is forced to reside within the varying slots (via the min/max). The inner part of the equation effectively places

Algorithm 6 update_slots: Update a slot given a signal message and increment the current slot number.

Require: ω: the new signal
$\mathcal{S} \leftarrow \{s_1, \ldots, s_{c-1}, \omega, s_{c+1}, \ldots, s_{|\mathcal{S}|-1}\}$
$c \leftarrow c + 1$
if $c < |\mathcal{S}_S|$ **and** $type(\omega) =$ "silence" **then**
  $n_d \leftarrow n_d + 1$
end if
if $c = |\mathcal{S}|$ **then**
  $A \leftarrow \{s_i \in \mathcal{S}_S : type(s_i) \neq$ "clarity"$\}$
  $\alpha \leftarrow \alpha - |\{s_i \in \mathcal{S} : s_i \in A$ **and** $i \leq \alpha\}|$
  $\mathcal{S}_S \leftarrow \mathcal{S}_S \setminus A$
  $B \leftarrow \{s_j \in \mathcal{S}_V : type(s_j) =$ "clarity"$\}$
  $C \leftarrow \{s_k \in \mathcal{S}_V : type(s_k) \neq$ "clarity"$\}$
  for $s_j \in B$ **do**
    for $s_k \in C$ **and** $k < j$ **do**
      $\mathcal{S} \leftarrow \{s_1, \ldots, s_k, \ldots, s_j, \ldots, s_{|\mathcal{S}|-1}\}$
      if $\alpha = j$ **then**
        $\alpha \leftarrow k$
      else if $\alpha = k$ **then**
        $\alpha \leftarrow j$
      end if
    end for
  end for
  $\mathcal{S}_S \leftarrow \mathcal{S}_S \cup B$
  $\mathcal{S}_V \leftarrow \mathcal{S}_V \setminus B$
  $n_d \leftarrow 0$
  $\mathcal{S}^- \leftarrow \{s \in \mathcal{S} : type(s) =$ "silence"$\}$
  if $|\mathcal{S}^-| = 0$ **then**
    $\mathcal{S}_V \leftarrow \mathcal{S}_V \cup \{0\}$
  else
    $c \leftarrow 0$
  end if
end if

the mean of normal random variable at the number of visible agents m. To ensure this m is not beyond the number of varying slots, we limit m's influence on the mean by the number of varying slots. When communications first begin, there are very few varying slots in existence. Even if an agent observes a large number of agents in front of it, it will restrict the mean to be at the last index of the varying slot. Lastly, the normal distribution's standard deviation is the number of varying slots, hence the $\sqrt{|\mathcal{S}_V|}$ component.

This modification seems quite small but will be shown to greatly improve performance. The main idea is to select slots so agents in front of the team have a predilection towards lower-indexed varying slots and agents in the back tend to select higher-index varying slots. When there are a large number of agents in the system, but a small number of total slots in the communication architecture, the vast majority of agents will center their means at the last varying slot index. This behavior results in a large quantity of collisions around the later varying slot indices, but few collisions towards the initial varying slots. Thus, agents move to settled slots more rapidly when they do not observe as many other agents. This produces an emergent sub-system in which agents proba-

bilistically "queuing" themselves, helping agents to more rapidly converge. It also lends itself nicely for realistic applications since agents in front of the group tend to have a better view of the environment than the rest.

Experimentation

We include two sets of experiments. The first focuses primarily on exploring the convergence rates of the algorithm. The second shows that by incorporating local agent knowledge into the equation, we can vastly improve the convergence times and the effective throughput, i.e., the average percentage of clearly sent messages per cycle. For each of these sets, we consider the worst-case scenario in which all of the agents start simultaneously in the slot selection state, and a staggered scenario in which an agent enters the learning state at a random time-step offset apart from the previously indexed one following a discrete uniform distribution of $U(0, 10)$ slots. This staggered scenario's indexing is allowed because agents are all homogeneous with respect to communication, so any permutation of them is equivalent.

The purpose for randomly offsetting an agent's starting time may not be directly apparent, but it is essential to demonstrate the effect each of the algorithms' states has on the system. If the worst-case (all starting simultaneously) was the only one considered, it would not show the computational benefits in having a collection of agents "waiting" in the learning state. Recall that the learning state waits until the system is settled before joining. This changes the number of agents vying for a settled slot, improving performance.

Monte-Carlo Convergence Rate Modeling

To get a sense of how efficient the algorithm is, we look at a collection of trials to chart the performance. Our performance measure has two components. The main measurement we use is the total number of steps until the entire group of agents have entered the settled state. This is then averaged over the collection of trials to gain an understanding of the number of steps for the particular size of the system. Similarly, we also investigated the average number of clear slots, which gives us one measurement pertaining to the "quality" of the system. This helps to infer additional communication estimates once the actual message size is determined, but is somewhat convoluted for comparative evaluation. To this end, we include one other point of comparison: the percentage of successfully sent messages per cycle (throughput). This provides a common basis for comparing the overall information that can be communicated.

Discussion

A common similarity across all of the results seems to be the apparent exponential growth of the convergence rates as a function of the number of agents in the system. This is seen in all figures and hints at the underlying structure of the entire algorithm. Notably, including knowledge resulted in a 25% improvement for convergence rate with 50 agents.

Comparing the convergence rates between offsetting agents' start times and not doing so does not seem to provide

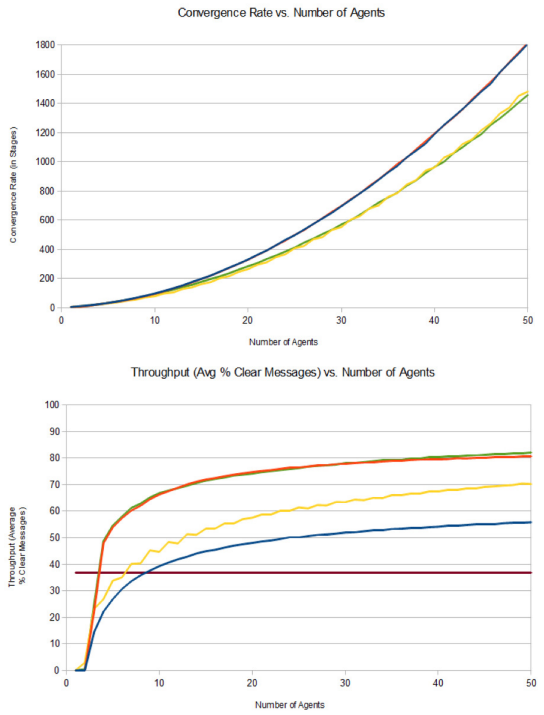

Figure 2: Convergence rates (top) and throughput (i.e., average number of clear messages) (bottom). These results summarize 1000 trials for each configuration of 1 to 50 agents in the system. Each line denotes an initialization: simultaneous (blue), random offset (orange), simultaneous with knowledge (yellow), and random offset with knowledge (green). The throughput of slotted ALOHA is shown in red.

a drastic difference at all. This is not too surprising because any offset trades off pushing the convergence time back with having fewer agents competing for variable slots. The interesting results from this are found in the average percentage of clear signals during a cycle. There is a significant increase in the number of messages that can be sent in configurations with agents that are delayed from entering the communication system. The overall result seems to encourage more settled slots and fewer varying slots during a cycle.

Another interesting point from the figures pertains to the throughput estimation computed by the average percentage of clear signals in a given cycle, which appears to grow at a logarithmic rate. From the results one might expect a similar number of successful messages sent by a MAS of any reasonable size (capping at around 80-85%). We see that in the worst-case scenario (simultaneously initializing agents) the throughput remains at about 55% at 50 agents. Including knowledge improves this to around 70%. By placing agents at random initialization times, we see that both the non-knowledge and knowledge cases are approximately the same, reaching 80-85% throughput. This similarity is caused by the fewer number of agents vying for a clear varying slot, reducing the need for customized slot selection equations.

Due to the similarity with slotted ALOHA, we plotted

Experiment	10 Agents	20 Agents	30 Agents	40 Agents	50 Agents
Simultaneous Start	97.6 ± 19.2	330.6 ± 45.2	695.3 ± 82.5	1190.2 ± 111.3	1809.5 ± 155.6
Random Offset	94.9 ± 18.7	328.6 ± 47.3	697.4 ± 80.8	1194.8 ± 117.3	1816.6 ± 157.3
Simultaneous Start (Knowledge)	78.9 ± 21.7	264.8 ± 54.3	552.2 ± 89.5	961.5 ± 132.6	1481.5 ± 189.8
Random Offset (Knowledge)	94.6 ± 20.0	284.6 ± 50.5	570.5 ± 85.8	965.1 ± 126.5	1458.6 ± 173.1

Table 1: For each of the four types of experiments and number of agents, this table shows the convergence rates (in number of steps) and its corresponding standard deviation.

Experiment	10 Agents	20 Agents	30 Agents	40 Agents	50 Agents
Simultaneous Start	$39.3\% \pm 5.5\%$	$48.0\% \pm 3.7\%$	$51.9\% \pm 3.0\%$	$54.1\% \pm 2.5\%$	$55.8\% \pm 2.2\%$
Random Offset	$66.3\% \pm 5.4\%$	$74.7\% \pm 3.5\%$	$77.9\% \pm 2.7\%$	$79.6\% \pm 2.2\%$	$80.7\% \pm 2.2\%$
Simultaneous Start (Knowledge)	$44.7\% \pm 7.9\%$	$57.5\% \pm 5.0\%$	$63.3\% \pm 3.6\%$	$67.4\% \pm 2.8\%$	$70.2\% \pm 2.5\%$
Random Offset (Knowledge)	$66.8\% \pm 5.0\%$	$74.1\% \pm 3.7\%$	$78.2\% \pm 2.8\%$	$80.3\% \pm 2.4\%$	$82.1\% \pm 1.9\%$

Table 2: For each of the four types of experiments and number of agents, this table shows the throughput (i.e., average percentages of clear messages sent for a cycle) and its corresponding standard deviation.

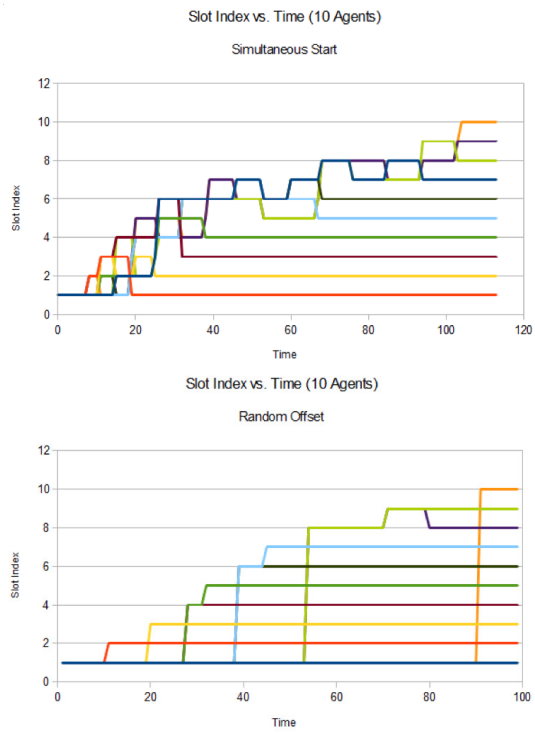

Figure 3: Examples of 10 agents starting simultaneously (top) and at random offsets between 0-10 steps (bottom).

its theoretical (and empirically verified) maximum throughput of $1/e$, which is approximately 36.79%. We see that DCA isn't as efficient during the time it takes for our agents to converge between about 1-8 agents. After that point, our algorithm will provide a higher throughput than slotted ALOHA. Note that once this algorithm converges to a set pattern, the throughput goes to 100% as the stages $t \to \infty$, which is always better than ALOHA.

One commonly observable behavior can be found in Figure 3 when Agent 10 waits until all other agents have settled before it jumps up to the available silent varying slot. Agents tend to silently wait in groups since agents remain in the learning state until no noise occurs in a cycle. Once the current set of agents settle, the waiting agents will begin.

To summarize, the most desirable situation seems to be a combination of random offsets to agent start times and incorporating additional agent knowledge into the system. Comparing just the convergence rates of each experiment type, it appears that randomly offsetting agents does not have much of an affect in comparison to simultaneously starting agents. However, there is a large effect on the percentage of clear messages, demonstrating that random offsets allow for more communication among agents. Lastly, the inclusion of high-level knowledge into this lower-level communication architecture algorithm always improves the overall behavior of the agent, and therefore the system as a whole.

Conclusion

In this paper we have introduced a new algorithm to dynamically form a round-robin communication architecture. This algorithm attempts to be as general as possible, and states a few extensions to tailor it to more specific situations. We demonstrated a method to add local agent knowledge into the system that greatly improves performance, as high as a $\sim$25% increase in convergence rates. It is our hope that these results show that including an agent's local knowledge into mid-to-low level algorithms (such as communications) can have a noticeable effect on overall performance. To test this, we created a simple scenario commonly found in the MAS research community in which agents can only observe a subset of the total number of agents. This number was then used as a parameter in the communication algorithm, demonstrating strong performance increases. We evaluated the algorithm using several metrics, primarily focusing on convergence rates and throughput. Our four experiments show that including local knowledge of an agent to the system can vastly improve both the both the convergence rates and percentage of clear messages. Future work might include a mathematical model to compute the convergence rate, and expand experimentation to a robotic domain.

Acknowledgments

This material is based upon work supported by The Office of Naval Research (ONR) through The Naval Sea Systems Command under Contract No. N00024-02-D-6604. The authors would like to thank the anonymous reviewers for their feedback, as well as both ONR and The Pennsylvania State University's Applied Research Laboratory for their support.

References

Abramson, N. 1977. The throughput of packet broadcasting channels. *IEEE Transactions on Communications* 25(1):128.

Benda, M.; Jagannathan, V.; and Dodhiawalla, R. 1986. On optimal cooperation of knowledge sources - an emperical investigation. Technical Report BCS-G2010-28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27(4):819–840.

Cha, J.-R., and Kim, J.-H. 2006. Dynamic framed slotted aloha algorithms using fast tag estimation method for rfid system. volume 2, 768–772.

Gerkey, B., and Mataric, M. 2002. Sold!: auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on* 18(5):758–768.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 709–715.

Jones, C.; Sivalingam, K.; Agrawal, P.; and Chen, J. 2001. A survey of energy efficient network protocols for wireless networks. *Wireless Networks* 7(4):343–358.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; and Osawa, E. 1997. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, Agents '97, 340–347. New York, NY, USA: ACM.

La Porta, T.; Maselli, G.; and Petrioli, C. 2011. Anticollision protocols for single-reader rfid systems: Temporal analysis and optimization. *IEEE Transactions on Mobile Computing* 10(2):267–279.

Lin, C.; Zadorozhny, V.; Krishnamurthy, P.; Park, H.; and Lee, C. 2011. A distributed and scalable time slot allocation protocol for wireless sensor networks. *IEEE Transactions on Mobile Computing* 10(4):505–518.

Ooi, J., and Wornell, G. W. 1996. Decentralized control of a multiple access broadcast channel: performance bounds. In *Decision and Control, 1996., Proceedings of the 35th IEEE Conference on*, volume 1, 293–298.

Parker, L. 1998. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation* 14(2):220–240.

Rhee, I.; Warrier, A.; Min, J.; and Xu, L. 2009. Drand: Distributed randomized tdma scheduling for wireless ad hoc networks. *IEEE Transactions on Mobile Computing* 8(10):1384–1396.

Roberts, L. 1975. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.* 5:28–42.

Stone, P., and Veloso, M. 1999. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence* 110(2):241–273.

Vidal, R.; Shakernia, O.; Kim, H.; Shim, D.; and Sastry, S. 2002. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation* 18(5):662–669.

Xuan, P.; Lesser, V.; and Zilberstein, S. 2001. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 616–623. ACM Press.

Yanco, H., and Stein, L. 1993. An adaptive communication protocol for cooperating mobile robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, 478–485. MIT Press.