ADDRESSING THE CONSENSUS PROBLEM IN REAL-TIME USING LIGHTWEIGHT MIDDLEWARE ON DISTRIBUTED DEVICES

A Thesis

by

KEITH ANTON HALL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2011

Major Subject: Industrial Engineering

Addressing the Consensus Problem in Real-time Using

Lightweight Middleware on Distributed Devices

ADDRESSING THE CONSENSUS PROBLEM IN REAL-TIME USING LIGHTWEIGHT MIDDLEWARE ON DISTRIBUTED DEVICES

A Thesis

by

KEITH ANTON HALL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTERS OF SCIENCE

Approved by:

Chair of Committee,	Abhijit Deshmukh
Committee Members,	Guy Curry
	Salih Yurttas
Head of Department,	Brett Peters

August 2011

Major Subject: Industrial Engineering

ABSTRACT

Addressing the Consensus Problem in Real-time Using Lightweight Middleware on Distributed Devices. (August 2011)

Keith Anton Hall, B.S., Texas A&M University

Chair of Advisory Committee: Dr. Abhijit Deshmukh

With the advent of the modern technological age, a plethora of electronic tools and devices are available in numbers as never before. While beneficial and exceedingly useful, these electronic devices require users to operate them. When designing systems capable of observing and acting upon an environment, the number of devices can become unmanageable. Previously, middleware systems were designed for large-scale computational systems. However, by applying similar concepts and distributing logic to autonomous agents residing on the devices, a new paradigm in distributed systems research on lightweight devices is conceivable. Therefore, this research focuses upon the development of a lightweight middleware that can reside on small devices enabling the capability for these devices to act autonomously.

In this research, analyses determine the most advantageous methods for solving this problem by defining a set of requirements for the necessary middleware as well as assumptions for the environment and system in which it would operate achieved a proper research focus. By utilizing concepts already in existence

such as peer-to-peer networking and distributed hash tables, devices in this system could communicate effectively and efficiently. Furthermore, creating custom algorithms for communicating with other devices and collaborating on task assignments achieves an approach to solving the consensus problem in real time.

The resulting middleware solution allows a demonstration to prove the efficacy. Using three devices capable of observing the environment and acting upon it, two tests highlight the capabilities of the consensus-finding mechanism as well as the ability of the devices to respond to changes in the environment autonomously.

To my friends and family who have been my source of encouragement and inspiration

ACKNOWLEDGEMENTS

I would sincerely like to thank my advisor, Dr. Abhijit Deshmukh, for his support and guidance as well as for sparking my interest in research. I would also like to thank Dr. Salih Yurttas for his support as a member of my committee and for his advice and knowledge throughout the last year of my studies. Additionally, I would like to thank Dr. Guy Curry for his invaluable advice and guidance throughout my graduate career. I would also like to express my gratitude to Dr. Gunnar Feldmann whose friendship and mentorship are deeply appreciated.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

TRADEMARKS AND SPECIAL TERMS

Author's Note: The following terms are acknowledged as trademarks, registered trademarks, or external material. In the body of the thesis, they are identified by italics.

- Apache®
- Apple® iPad®
- Apple® iPhone®
- Apple® Mac OS® X
- Chord
- Entangled
- HyperText Markup Language (HTML)
- JavaScript®
- jQuery
- Kademlia
- Lighttpd
- Linux®
- Microsoft® Internet Information Services (IIS)
- Microsoft® Windows® XP
- Nginx™
- Pastry
- PHP Hypertext Preprocessor (PHP)

- Python®
- SQLite®
- Synaptec
- Ubuntu®

1. INTRODUCTION

The development of human language stems from the innate need of intelligent beings to communicate with one another. In the case of early humans, communication was crucial to survival as man was generally not as strong or as fast as his animal predators and prey. However, with advanced cognitive and communication abilities, early man eventually conquered and proved his worth in the prehistoric ecosystem.

Technology has since advanced to allow electronic communication. In 1971, the first electronic distributed message was sent from one computer to another [1]. This one message paved the way for the development of email and later, the internet, the distributed information communication infrastructure of the world. Rather than consisting of single points of contact, systems now have seemingly unlimited number of resources easily disposable for solving problems.

Regardless of the objective or specific objective, whether it is to express oneself, to disseminate information, or coordinate resources and personnel, the goal for the communication is ultimately to reach a mutual agreement or understanding — a consensus. While the uses of language have evolved and diverged greatly, the mechanisms for allowing networked devices to collaborate with one another to solve problems or reach consensus autonomously still lack in maturity. Furthermore, with the advent of the modern technological age, most information and

---

This thesis is in the style of *IEEE Transactions on Automatic Control*.

data processing as well as communication occur in a distributed fashion. With electronic devices not only being deployed in localized environments, but also dispersed globally, the importance of autonomously managing them increases continuously.

1.1 Objective

The primary objective of this research is to provide a means for autonomous or semi-autonomous agents to solve a consensus problem dynamically using real-time analysis and allow dynamic reallocation of resources via collaboration and communication. This objective can be decomposed into several sub-objectives:

1. Create a distributed and decentralized middleware for communication and data exchange
2. Create a common interfacing mechanism between agents
3. Create a mechanism to utilize consensus-reaching algorithms
4. Provide a means for information dissemination to the appropriate agents

1.2 Utility

The world as a whole is becoming increasingly digitalized, where everything from government, infrastructure, economies, and war have strong technology ties to increase efficiency and ease. However, the true potential of technology assimilation is not realized unless technologies can work together to provide more powerful solutions. If technology can solve human problems with little to no interaction from the users or operators, the utility increases dramatically.

The power of distributed computing is understood well and realized, but its application is limited and unimaginative although nontrivial. Rather than using distributed computing to perform complex calculations, one could use a network of small, lightweight, and agile agents or devices to analyze problems and to solve them optimally autonomously and in real-time. Potential application areas include, but are not limited to battlefield analysis, distributed surveillance, infrastructure monitoring, smart sensor networks, weather monitoring, and smart distributed safety systems. At this pivotal point in technological evolution, methodologies must be developed that can utilize the ever increasing power as well as ever decreasing size of devices.

1.3 Outline

The next section, Literature Review, covers much of the current work in both distributed systems as well as the approaches to solving the consensus problem for this topic. The following section, Problem Definition, lists the problems to be addressed along with the assumptions that involve them. Afterwards, a theoretical solution is proposed along with the requirements and software solution in the Lightweight Middleware Solution section. In the next section, Implementation, two test cases demonstrate the effectiveness of the solution. Finally, the Conclusion summarizes the thesis and presents any open problems.

2. LITERATURE REVIEW

As the utility for distributed smart device networks gains notoriety and traction, the needs and uses continue to evolve rapidly. The uses for distributed smart device networks have been recognized; however, several aspects have received little attention from the research community to solve for dynamism. Additionally, the consensus problem itself is still relatively new. Furthermore, because this research focuses upon deployment of lightweight devices, the applicability of many of the current solutions in the literature also is critiqued.

2.1 The Consensus Problem

The ideas of the consensus problem and the methods to solve it have been an active area of research since the 1980s [2]. Typically, the consensus problem is solved by parameterizing the agents involved in the system and their vantage points as decision variables in an *a priori* optimization algorithm to find an optimal or near-optimal consensus point [3], [4].

However, when dealing with a live environment in which agents already interact, the system must be able to adapt to changes. When agents join or leave the network, the solution must be found quickly while still accounting for the dynamic changes. Predetermined consensus points may be able to cover the spectrum of possibilities for a small number of agents; however, as the number of agents scales to larger numbers, the ability to store the consensus point for every scenario becomes infeasible – especially with lightweight devices.

2.2 Information Distribution

When dealing with multi-agent systems, an important consideration is the distribution of data. More importantly, how is the correct data shared with the appropriate agents? In small networks, any standard network topology should be sufficient, the most popular of which appear in Figure 2.1.

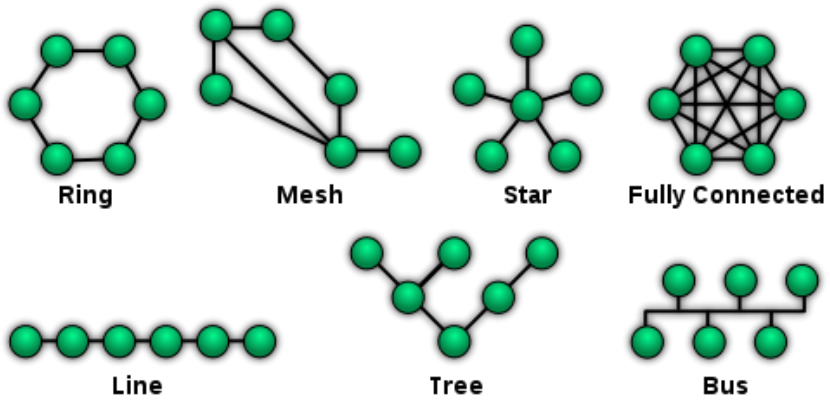

Figure 2.1. Standard network topologies [5]

However, in the simple topologies, *e.g.* ring, line, and bus, scalability suffers as all the data must pass through all the agents. In the star and tree topologies, removing the central node or connecting nodes can render the system inert. Finally, the fully connected topology maintains data robustness, but the does not scale well to a large number of lightweight devices.

In an attempt to alleviate these problems, several solutions have been proposed. One choice is the use of synchronous message passing with end-point verification. In this type of system, data incapable of propagating to every agent asynchronously, such as data received at a target, is likely to be out of date by the time it traverses the network [6]. Instead, agents must store data received and after a predetermined time, verify that the source is still active [6].

Another popular choice is the so called publisher/subscriber system. Typically with these systems, a publisher is an agent which disseminates information to its subscribers [7], [8]. As such, the subscribers have predetermined lists of publishers from which they await information. While an effective way to distribute information for networks with a small number of agents, it does not scale well and does not perform properly when failures occur. This technology was improved with event-based notification systems [9]. While this technology is scalable and fault-tolerant, it is still possible to reduce the overhead and improve the data integrity with increased fault-tolerance and system dynamism.

2.3 System and Data Robustness

Another problem identified by early works was the failure of consensus algorithms given faulty processes or agents [10]. During the early methodologies, the consensus problem was solved by using existing and new optimization algorithms [10], [11]. However, the issues facing the consensus problem were real-

ized, and one of the main concerns was ensuring data integrity and system resilience when failures occur [11].

However, the problem of how to ensure that the distributed system remains functional when unexpected circumstances remove agents has yet to be addressed. Additionally, when dealing with lightweight devices, the amount of information being distributed and stored must not exceed the capabilities of the devices. In fact Culler *et. al.* have identified this restriction as one of the main limiting factors for the advancement of sensor network research [12].

2.4 Middleware Development

While the idea of middleware itself has existed since the late 1960's as the description of the software that lies between the application layer and the service layer [13], the truly distributed systems approach to middleware did not begin gaining traction until the early 1990's [14].

A common method for solving consensus problems is grid computing. In grid computing, resources of agents are shared among one another in an effort to solve a common problem. Typically, grid computing utilizes distributed computers to share computing power, data, and other tools, *e.g.* Globus [15]. However, large computational grid software is not appropriate for deployment on small devices. Sarjoughian *et. al.* provide a logical framework for designing lightweight middleware systems which can also apply to this solution [16].

3. PROBLEM DEFINITION

Knowing that the collaboration among agents is growing in necessity and that the current state-of-the-art lacks several core competencies to address these needs, several problems must be addressed. A formalized statement of the problems follows. Then, some definitions are provided to help familiarize the reader with the terminology of this thesis; and, finally, the assumptions about the problem as a whole are presented.

3.1 Problems

When trying to reach an agreement in an effort to solve a problem automatically, several issues arise. The first problem encountered involves the process for determining when a consensus is reached. Without human operators or users directly providing logic, an innovative algorithm must be created. However, because the objective is to solve this problem in real-time, the algorithm must be adaptive and respond to changes in the environment.

The second problem to address is the distribution and sharing of information among the autonomous agents. Any relevant information necessary for finding the consensus point should be distributed to the agents that are involved directly. It is wasteful of resources and time to send every piece of information to every agent in the system. As such, a proper data-passing algorithm must be employed.

The third problem to address is the robustness of the distributed network. Given that the agents are to be distributed, if a centralized data and communication repository or server were to fail or go offline, the entire infrastructure would become unusable. Therefore, data must be distributed such that the entire system will not fail if one agent fails. Given a decentralized infrastructure, appropriate assignment is crucial to avoid unnecessary waste or processing.

Finally, if this system is to function on embedded devices, the software and hardware requirements must be minimal. Thus, a lightweight approach must be used to ensure that the system can function on the smallest feasible devices.

3.2 Assumptions

Assumption 1. *Each agent has a limited view of the environment as a whole and is also limited by the sensing capabilities of the hardware of the device it controls, i.e. an agent can only detect that which it is capable of detecting.*

This distinction is necessary to separate not only the independent agents, but also the data that are observable by each agent. Figure 3.1 is a simplified graphical representation of this assumption.

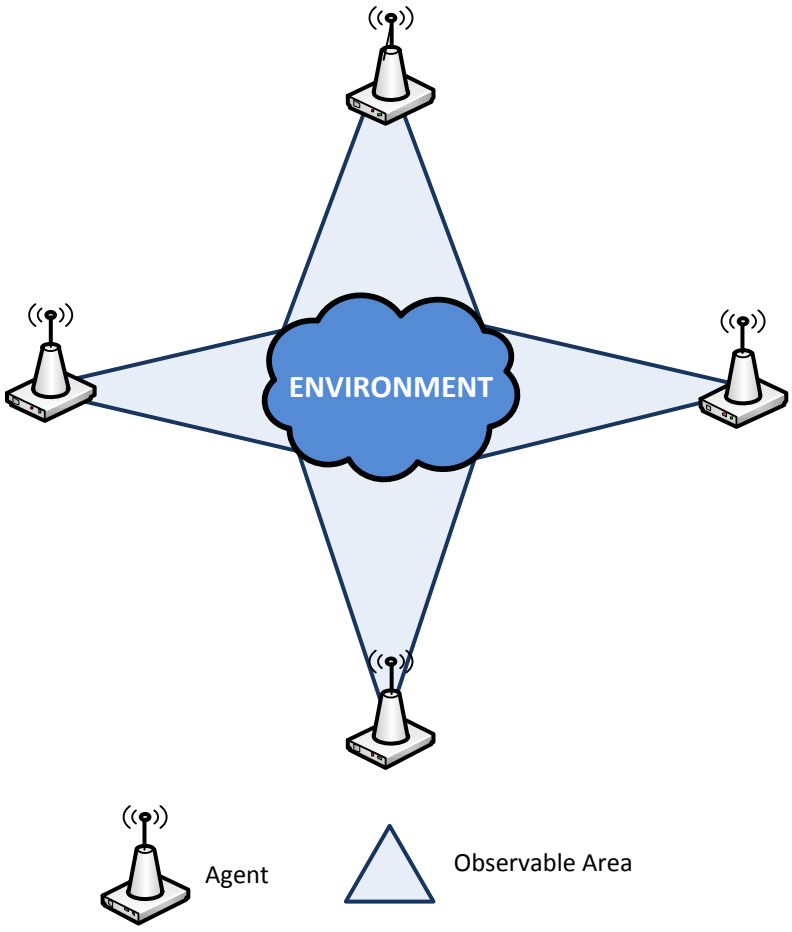

Figure 3.1. Individual views of environment by agents

Assumption 2. *The network is volatile, and agents will join and leave the system at an unpredictable rate.*

By assuming deployment scenarios in which hardware or network failures occur, the solution must retain system integrity. Similarly, if new agents are installed, the system should capitalize on the increased capacity.

Assumption 3. *The system must be able to accommodate a large number of agents.*

Despite the limited resources of the laboratory environment, the system must be designed such that it can scale from a small number of agents, e.g. 3, to a large number, e.g. ten thousand.

Assumption 4. *The devices on which the agent software resides are limited in terms of computational power and functionality.*

Assuming the system software will reside on sensors, embedded devices, or small robots, the hardware requirements must be minimal.

Assumption 5. *The devices are non-homogeneous.*

While a system may be composed of one type of device, unforeseen circumstances may arise which necessitate newer or otherwise different hardware. As such, the system must work with both a heterogeneous and homogeneous array of devices.

3.3 Definitions

Definition 1. A *middleware* is a software layer that connects software components to one another or agents to one another.

Definition 2. The *system* is an interconnected network of devices, the communication between them, and the environment in which they operate.

Definition 3. A *dynamic system* is one in which the properties of the system change over time, often unpredictably.

Definition 4. An *agent* is an autonomous entity that observes and acts upon an environment through the middleware and directs its activity towards achieving goals [17].

Definition 5. A *device* is an electronic machine capable of attaining some end.

Definition 6. A *centralized network* is a network architecture in which agents communicate through a central server.

Definition 7. A *peer-to-peer network* is a network architecture in which agents communicate with one another rather than through a central server.

Definition 8. A *distributed hash table (DHT)* is a service for passing data between nodes according to a unique (key, value) pair while limiting the propagation of data to enable large-scale applications.

Definition 9. A *data store* is a database stored on each node containing the DHT data. A common data language must be used between all nodes.

Definition 10. A *node* is a connection point in the network that is capable of sending and receiving data over a communication channel.

Definition 11. A *bootstrap node* or *rendezvous host* is a predefined initial connection point to which other nodes connect to join the peer-to-peer distributed network. After *initialization*, the new node is able to communicate normally without bootstrap node dependency.

Definition 12. An *edge node* is a node that exists within the system that can communicate with other edge nodes. Bootstrap nodes are also edge nodes.

Definition 13. *Initialization* is the process of connecting a new node to the network, usually through a bootstrap node unless the new node is itself is the first node to start the system.

4. LIGHTWEIGHT MIDDLEWARE SOLUTION

To overcome these challenges, I propose a distributed device network that features dynamic discovery with limited to no operator intervention. Before designing the middleware component, the requirements for the devices and the software must be defined clearly to ensure that the correct solution is implemented.

4.1 Requirements

Requirement 1. *New devices should integrate into the system quickly and easily.*

In a dynamic system, devices join the network and leave the network unexpectedly. Therefore, the system must be sensitive enough to detect these changes. However, if devices are difficult to integrate into the system, the rest of the system cannot utilize the increased capacity to find a better solution.

Requirement 2. *Devices should be able to talk to one another both internally and externally.*

Without basic network protocols such as Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), the devices cannot communicate with one another. By utilizing already proven technologies, the middleware should directly solve the challenges stated in Subsection 3.1.

Requirement 3. *Devices should store limited data.*

When dealing with lightweight devices, processing power and memory are severely limited. Therefore, the middleware must accommodate the hardware limitations of the devices on which it resides.

Requirement 4.*The middleware must be modular.*

Because this problem deals with a non-homogenous array of devices, the ability to substitute quickly not only devices, but also their individual capabilities, is essential. By providing the means to substitute one device for another, the can adapt better to respond to dynamism.

Requirement 5. *The system and data storage must be decentralized.*

In a centralized system, a single point of failure, typically known as a “server”, exists that threatens the integrity of the system as well as the data. If the server is compromised, the rest of the system is incapacitated. To safeguard against data loss or system downtime, the middleware must distribute the communication channels as well as the data storage.

Requirement 6. *Devices must be able to join from any internet-enabled location.*

As long as a device is connected to the internet, it should be able to join the system. Any intermediary firewalls should not interfere with the connection.

Requirement 7. *The devices must be able to adapt to changes in the environment.*

If the goal of the agents changes while executing, the system must be able to respond. Additionally, if agents join or leave while executing a task, the system must adjust to the change in capacity.

Requirement 8. *Based upon logic algorithms, the devices must be able to reach consensus autonomously.*

In order to reach consensus for a specific scenario, the middleware agent on each device must include logic patterns to aid each device in making the right decisions. With each device having its own logic, they must determine the consensus point by communicating with one another.

Requirement 9. *User interaction with the devices and the system should be limited, if necessary.*

The purpose of this system of autonomous devices is to solve the consensus problem without the need for a human to micro-manage it. Additionally, if users influence the process, the algorithms might yield an incorrect consensus point or take longer than necessary.

Thus, user interaction should be limited to the initial configuration of the devices. After the device has the middleware installed and the properties configured, the

device can be deployed and join the system. At this point, the user should not need to touch the device again unless hardware maintenance is required.

4.2 System Design

Taking into account the requirements for the middleware as well as the assumptions that shape the system and the environment, the following solutions are proposed for incorporation into the lightweight middleware.

4.2.1 Decentralized Networking

As discussed in Subsection 3.1, one of the problems present in many distributed systems is reliance upon a centralized hub or server. If and when the central hub or server fails or unexpectedly leaves the network, the rest of the network follows suit. Figure 4.1 provides an illustration of a centralized network topology. The need for decentralizing the distributed system is obvious.

Luckily, decentralized solutions already exist. As shown in Figure 2.1, fully connected and mesh network topologies remove the need for a central hub or server by distributing the information. While fully connected networks distribute data, all the data are replicated across all the nodes. Therefore, using the fully connected network topology would violate Requirement 3.

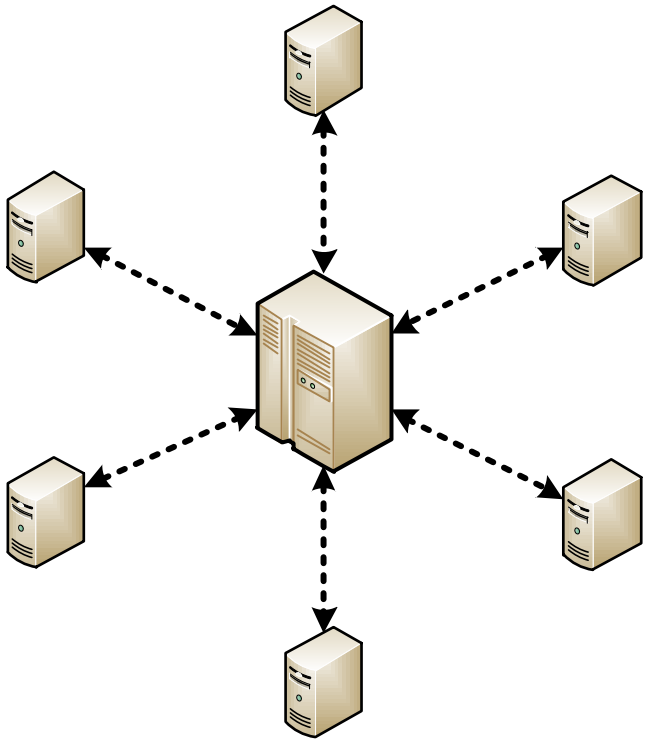

Figure 4.1. Centralized network topography

However, the mesh network does not necessarily require that every node be connected to every other node. If the mesh network can establish communication channels intelligently among nodes, both Requirement 2 and Requirement 5 would be satisfied. A peer-to-peer network, as shown in Figure 4.2, facilitates the communication between peer nodes as necessary. If a certain node requests data from another, a communication channel enables the data transfer either unidirectionally or bidirectionally.

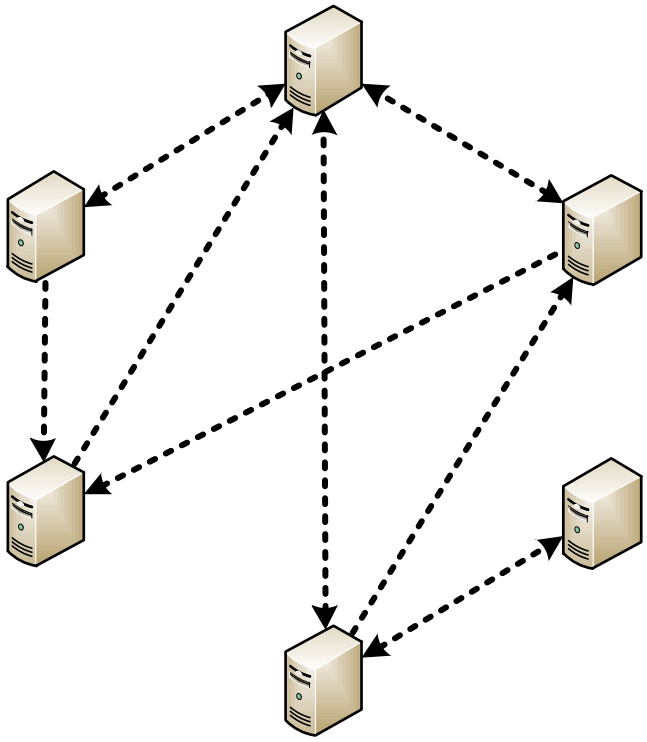

Figure 4.2. Peer-to-peer network topology

4.2.2 Data and Communication

Once a node joins the system, it must update its data repository. By using the well-developed concept of a distributed hash table (DHT), the new node can quickly learn of its surroundings by asking its nearest neighbors. Likewise, the nearest neighbors of the new node learn of the new node and store its related data - an example of which appears in Figure 4.3 and Algorithm 4.1. In the figure, the green lines represent the communication channels with the new node and its nearest neighbors. The black line represents an established channel between the two existing nodes.

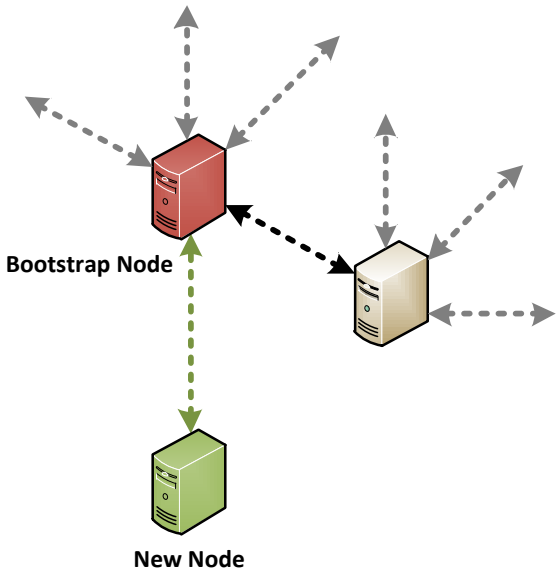

Figure 4.3. New node initial collection

```
# Load the configuration for the device
config = load_device_configuration()

# Load any predefined neighbors to seek
known_nodes = None
if config.known_nodes
    known_nodes = config.known_nodes

# Initialize the datastore on the device
datastore = create_datastore()

# Create the node object using datastore
node = create_node(datastore)

# Connect the node to the P2P network
node.join_network(known_nodes)
```

Algorithm 4.1. New node initialization

If a node fails or otherwise leaves the system, the information associated with that node also leaves. This prevents the system from becoming overburdened by extra data that is not useful in finding the consensus point.

In order to ensure proper communication among nodes, the system requires a common interface and data representation language. A fundamental component of a DHT is the hashed (key, value) pair. Upon data creation, it passes through a unique hashing function that converts readable data into an encoded key. Then, this key is shared along with its values to the receiving nodes. Figure 4.4 provides an illustration of this concept, and Algorithm 4.2 is an example using the hashing function with sample output.

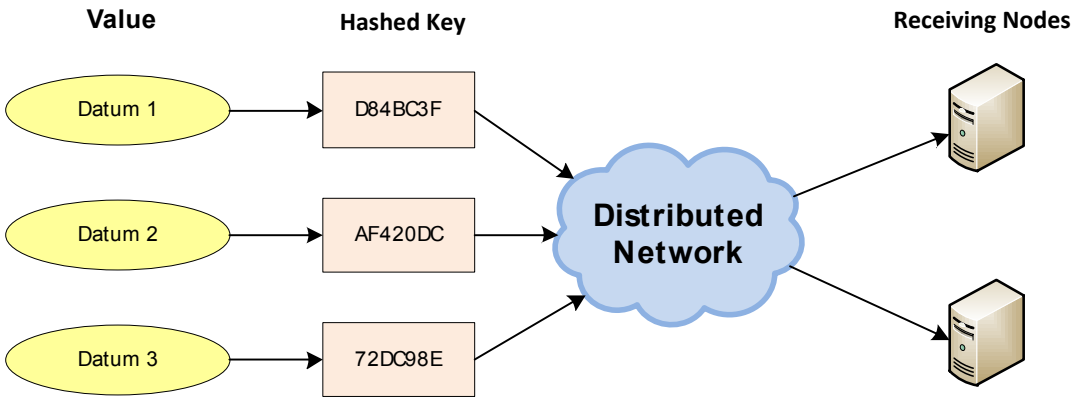

Figure 4.4. Distributed hash table

```
 1| data_value_1 = “Datum 1”
 2| hashed_key_1 = hash_function(data_value)
 3| hashes.append(hashed_key_1)
 4|
 5| data_value_2 = “Datum 2”
 6| hashed_key_2 = hash_function(data_value)
 7| hashes.append(hashed_key_2)
 8|
 9| # Print Pairs
10| for hash in hashes
11|      print hash | hash.lookup_value()

OUTPUT
D84BC3f | Datum 1
AF420DC | Datum 2
```

Algorithm 4.2. Creating a hashed (key, value) pair

Additionally, because the DHT only passes data to nodes as needed instead of broadly broadcasting to every node simultaneously, less data clogs the communication channels and less information must be stored on the individual devices.

As proposed, the data and communication architecture satisfy Requirement 1, Requirement 2, Requirement 3, and Requirement 5.

4.2.3 Bootstrap and Edge Nodes

In order to facilitate the design of the distributed system software, two types of nodes are necessary: bootstrap and edge nodes. The purpose for the bootstrap node is to allow other nodes, either fellow bootstrap or edge, to join the network by serving data to them. In essence, the bootstrap node is a predefined stable node that is already part of the distributed network that provides configuration

information to new nodes – a stable constant in a dynamic system. Only one bootstrap node is required to start the peer-to-peer network; any other bootstrap node in the system exists only for redundancy.

Bootstrap nodes are also capable of poking holes through firewalls that exist between the distributed system network and the new node requesting access.

By contrast, edge nodes join the network by contacting a bootstrap node. Despite this setup, any edge node can become a bootstrap node, and any bootstrap node still functions as an edge node by default. The only distinction between the two is that a bootstrap node is visible to other nodes external to the network as well. Hence, the communication port for the bootstrap node must be accessible from outside the network if located behind a firewall or other routing devices. In Algorithm 4.1, an edge node would have the bootstrap nodes defined as “known hosts” in the configuration file for the device.

Because the system is dynamic, it is not feasible to require the middleware to scan constantly for new devices. Instead, new devices should communicate with the bootstrap nodes to receive the necessary credentials to join the network. As shown in Figure 4.5, the steps for the initialization process are:

1) The new node must contact the bootstrap node and request access.
2) Bootstrap node verifies new node and sends back connection data. In this step, the message back opens the firewall(s) for the new node.
3) The new node may now join the network and communicate freely.

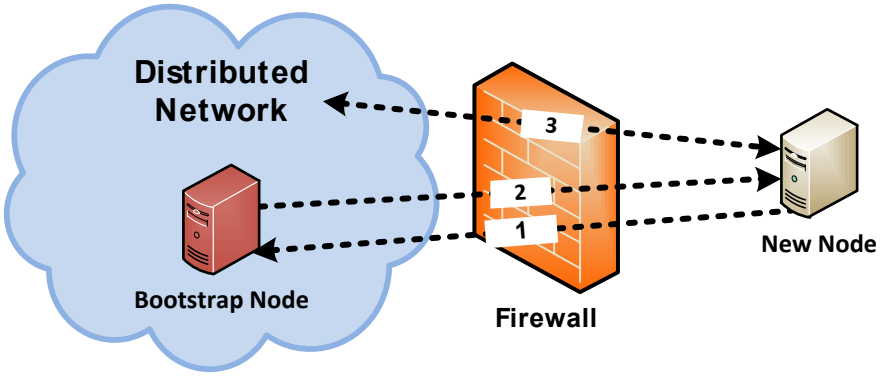

Figure 4.5. New node initialization

The node architecture as described satisfies Requirement 1, Requirement 4 Requirement 6 and partially satisfies Requirement 2.

4.2.4 Web Nodes

In addition to the edge and bootstrap nodes, any of either type can be assigned an additional role: web node. The purpose of a web server residing on a node is to provide platform-independent access to the system. Through the use of a web-based interface, a user can gain access to the system from any internet-ready device from any location capable of connecting to the network. Further-

more, a web-based approach provides an additional degree of flexibility for deployment scenarios. For example, the system status can be assessed remotely. Algorithm 4.3 demonstrates the method used by the web node to display device status to the web interface.

```
# Query datastore for online devices
devices = datastore.query(“get online devices”)

json_data = None
for device in devices
    this_data = jsonify(device.address,
                        device.uuid,
                        device.name,
                        device.description
                        device.current_task,
                        device.messages)

    json_data.append(this_data)

# Send JSON data to web interface
send_to_web(json_data)
```

Algorithm 4.3. Show system status on web interface

Furthermore, given the dynamic capabilities of the proposed middleware, users require an interface into the system from which tasks can be assigned. Algorithm 4.4 demonstrates the mechanism for assigning tasks to agents that are actually capable of solving the task based on the requirements of the task.

```
# Select a predefined task
task = tasks[choice]

# Check datastore for online devices
devices = datastore.query("get online devices")

# Load middleware agents
agents = None
for device in devices
    # Capabilities match requirements
    if compare(device.capabilities, task.requirements)
        agent = device.get_agent()
        agents.append(agent)

# Send task to agents on devices
for agent in agents
    agent.send_task(task, agents)

# Send task function
function send_task(task, agents)
    rpc_call(agent.address, agent.port, task)

    # Update Node Contacts
    node.contact(agents)
```

Algorithm 4.4. Assign task to available agents

A web node designed with this goal should enable users to see the state of the system quickly from different views. Primarily, the user should be able to see:

- A list of devices currently connected to the system,
- The information stored on each device,
- The objective, if any, of a given device, and
- The consensus point found by each group of agents.

Additionally, the more nodes assigned the additional web server role, the more redundant the system becomes. If a web node leaves the network, web traffic can reroute to another web node that is still online.

The web node functionality as described satisfies Requirement 9.

4.2.5 Task Completion

When the user assigns a task to the system, the middleware agents residing on each device should receive the instruction and automatically coordinate with one another to assign the task appropriately. Once the task has been assigned, the system can adapt dynamically to changes to successfully find a solution agreed upon by the agents and execute to completion.

When each agent has properly assessed the environment in relation to its assigned task, it sends a message to the other agents assigned to the same task via the DHT. After a predefined waiting period, the agents confer with one another according to any logic algorithms they have been assigned to determine how the task would be completed best. At this point, the agents send the instructions to the devices to execute the task to completion.

However, if during this process a device leaves the network caused by a hardware failure or user interruption, the middleware must adapt in real-time. If a task originally has four devices, and one leaves, the other three must reposition themselves to observe the environment from a wider viewpoint. Conversely, if a

new device joins the system and is automatically assigned to a task already in progress, it may affect the consensus algorithm and take over responsibility, if it is best suited for the task. Algorithm 4.5 demonstrates this logic.

```
last_contacts = None
last_assigned = None

while running()
    contacts = task.get_agents() # excluding self

    # Check if the number of contacts has changed
    if contacts is not equal to last_contacts
        consensus = consensus.find(contacts)

    assigned = consensus.assigned_agent

    # Change the execution state to stop the running agent
    if assigned is not equal to last_assigned
        executing = False

    # Check if the agent is assigned
    if assigned is self
        status = EXECUTING

        # Execute task
        executing = True
        task.execute()
    else
        status = WAITING

    # Used for iterative comparison
    last_contacts = contacts
    last_assigned = consensus.assigned
```

Algorithm 4.5. Task execution for each agent

Throughout the entire task assignment and executing process, up-to-date information is pushed to the web node so that any users observing the system know what is happening in real-time.

The methodology described in this section satisfies Requirement 7 and Requirement 8.

4.3 Architecture

The design of the middleware and supporting software is based upon several considerations. First, free and open source software is desirable to avoid licensing issues and to minimize costs. Second, the end result must be fast and lightweight (having minimal hardware requirements). Third, the software must be usable across a wide variety of devices.

4.3.1 Networking

While the study of peer-to-peer networking is not new, it is still an active field of research. Despite the available improvements possible, the development of a new networking architecture is beyond the scope of this work. Therefore, several peer-to-peer networking implementations were compared an effort to select the most appropriate for this application. The reader should examine the following protocols: *Chord* [18], *Pastry* [19], and *Kademlia* [20].

Ultimately, *Entangled* was selected for the following reasons:

1) The distributed hash table (DHT) is based on *Kademlia*
2) It provides a mechanism for deleting (key, value) pairs from the DHT
3) It incorporates keyword-aware operations like publish, search, remove
4) It incorporates a distributed tuple space[1]
5) It is written in the Python programming language
6) It users the *Twisted* framework [21] and *SQLite* database engine
7) It is open-source under the LGPLv3+ license

By utilizing the advanced capabilities of *Entangled*, better management of the DHT is achievable. Given a key, values can be retrieved easily because they are distributed across the nodes. Any changes to the state of the nodes (failure, new arrival, departure, or maintenance) cause minimal disruption. This characteristic also allows the network to scale to a large number of nodes without affecting the responsiveness of the system as a whole.

Kademlia itself is a peer-to-peer information system based upon the XOR metric [20]. Through node lookups, *Kademlia* specifies the structure of the network and the exchange of information. Additionally, *Kademlia* uses the UDP protocol to communicate between nodes. This provides an advantage over systems that use TCP because of the broadcast functionality and firewall and router traversing capabilities. Unlike TCP, UDP is a stateless protocol, meaning data can be

[1] Concurrently accessible ordered list of objects

sent between nodes without initializing a connection. Each node has a unique identifier that enables efficient (key, value) lookup functionality.

Also central to *Entangled* is the implementation of *Twisted*, an asynchronous Python programming framework. The use of *Twisted* allows the middleware run in a non-blocking fashion that is particularly advantageous in a decentralized network where the nodes join and leave unpredictably. Unlike synchronous approaches where programs wait for responses, the asynchronous approach responds temporarily with a deferred object that can be passed as a substitute for a true response. The program runs continuously while the deferred system handles the responses until an answer is returned. If a time-out occurs, the answer is passed to the error handling system. Another benefit of *Entangled* is its implementation of remote procedure calls (RPC). This feature allows code to be instantiated on one node and executed on another. Such calls enable the system to store, delete, and search for (key, value) pairs.

Furthermore, *Entangled* makes use of *SQLite* for the DHT data store. Given the necessity for interoperability among technologies at all levels of the middleware, this database engine is particularly advantageous because of its wide-spread adoption and support. Recent versions of Python have SQLite embedded in the language because of its ubiquity.

4.3.2 Web Interface

The next major component for the middleware to address is the user interaction with the system. While the final aim is an autonomous system, users still play a key role in decision making and management in external systems. Two major goals for the web interface are: it be lightweight and platform independent. The web interface must be lightweight to ensure that it can fit on small, embedded devices while still being able to scale considerably. Platform independence is vital because of the fragmentation of internet-capable devices.

To display any web site, web server software must serve content. Popular web servers include *Apache*, *Internet Information Services* (*IIS*), *Lighttpd*, and *Nginx*, among others. *Lighttpd* was selected as the web server because it:

1) Is lightweight (less than 1 MB[2] when installed)
2) It is scalable[3]
3) It is free and open source
4) It supports server-side languages, *e.g. Python* and *PHP*

To serve the information stored on the DHT requires an interface language. Without bias, the *PHP Hypertext Preprocessor* (*PHP*) was selected because of prior experience, ease of use, and performance when combined with *Lighttpd*. The use of *PHP* enables the querying of the *SQLite* data store and the dynamic display of content.

[2] http://www.lighttpd.net/download/
[3] http://www.lighttpd.net/story

As with all web sites, content is displayed using the HyperText Markup Language, or *HTML*. Every web browser incorporates its own *HTML* renderer that offloads this processing from the web node. Additionally, most web browsers, including smart phones, support *JavaScript* natively. To simplify the development process, the *jQuery JavaScript* Library was used. *jQuery* is fast, lightweight (31 KB), and well documented [22].

With these technologies working together, asynchronous requests to the DHT data store can be made and the results published in real-time to the user via the web interface. Additionally, when a user assigns a task or updates a device or setting, the requests are sent and then immediately reflected.

4.3.3 Operating System

While mainstream operating systems such as *Windows* and *OS X* are capable of running such middleware, they cannot run on small devices. Therefore, an operating system must be selected that maintains minimal hardware requirements. Based upon the selection of technologies in Subsection 4.3.1, the *Linux* distribution *Ubuntu* is desirable. *Ubuntu* is currently one of the most popular *Linux* distributions. Additionally, support for *Python* is included by default and the other applications are easily installed and maintained via the *Synaptec* package management software. The minimum hardware requirements for Ubuntu are shown in Table 4.1.

Table 4.1. Ubuntu hardware requirements [23]

Environment	RAM	Hard Drive
Without Desktop*	64 MB	1 GB
With Desktop	64 MB	5 GB

4.3.4 Consensus Algorithms

The lightweight middleware is modular. Thus, it contains no built-in algorithms because the application of the middleware is specific to a certain purpose. Instead, the mechanism for deploying algorithm tailored to a specific purpose or task has been created. The next section contains an example of the middleware implementation and the consensus-finding methodology for the problem presented.

4.3.5 Summary

The underlying architecture used for the middleware is based upon *Ubuntu*, *Entangled*, *Lighttpd*, *PHP*, and *jQuery* - shown in Figure 4.6. Given these components, the *Entangled* network creates a decentralized distributed network. The underlying DHT ensures that nodes can be added and removed from the system without disrupting operations. Furthermore, the use of the DHT allows the network to scale in size while simultaneously handling a large number of requests.

The capabilities of each node are distributed upon initialization, and the subsequent changes are updated in the *SQLite* data store for each node according to the DHT algorithm. Finally, the web node asynchronously queries the data store to ascertain the status of the connected nodes and the system and to assign tasks dynamically via remote procedure calls. Because the system changes dynamically, the web content is updated according to the *PHP* and *jQuery* scripts.

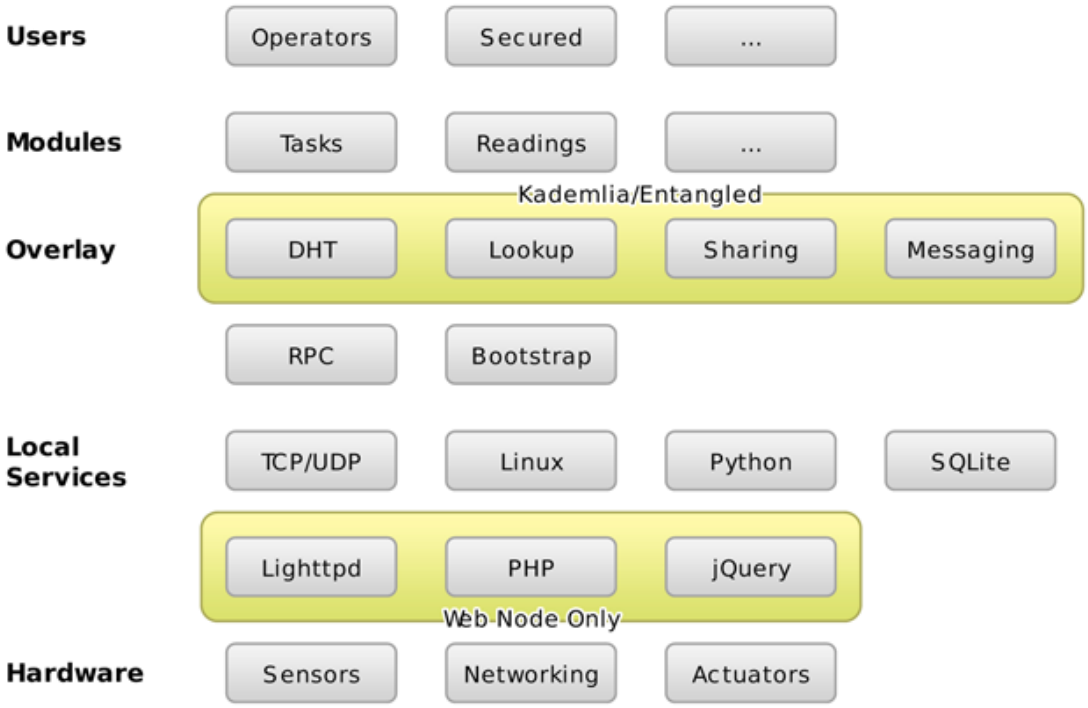

Figure 4.6. Software architecture diagram

5. IMPLEMENTATION

In order to demonstrate the capabilities of this approach for the consensus problem with distributed devices, a scenario was created that utilizes three network-capable sensing devices that can reach a consensus point heuristically in real-time given an assigned task. While many robots or sensing devices may be used with this system, the *Sony AIBO ERS-7* robotic dogs [24] were chosen because of their:

1) Wireless networking capability
2) Programmable logic for control
3) Video camera at 30 frames per second
4) Three infrared proximity sensors
5) Three-dimensional accelerometers
6) Sensors that update every 32 ms with 4 samples per update

However, because the operating system on the *Sony AIBO* robots is not accessible, each was paired with a laptop that ran the middleware. Essentially, the laptop served as the node for the peer-to-peer network and as the agent for the middleware. The agent on each laptop was programmed to control its own *Sony AIBO*, specifically on the hardware layer. Table 5.1 displays this setup.

Table 5.1. Agent for each laptop-robot pair

Device	Laptop	Agent Name
Sony AIBO 1 (dog1)	dog1_laptop	dog1_agent
Sony AIBO 2 (dog2)	dog2_laptop	dog2_agent
Sony AIBO 3 (dog3)	dog3_laptop	dog3_agent

On each laptop, the modules necessary to control the robot were loaded into the middleware software at runtime. In order to interface the hardware layer with a common language, the *Pyro Python* hardware abstraction software was utilized. Essentially, *Pyro* provides an application programing interface (API) in *Python* for controlling the *Sony AIBO* robots. A list of common commands and usage is available on the *Pyro* documentation page [24].

In addition to the robots, a web node was added to provide real-time information regarding the system. An Intel Atom-based *Sony VAIO P-series* laptop held the middleware, the web node functionality, and the bootstrap functionality. This netbook was selected for the bootstrap and web node because it showcases that the most demanding configuration of the middleware can run on a device with low hardware requirements. The final configuration is in Figure 5.1.

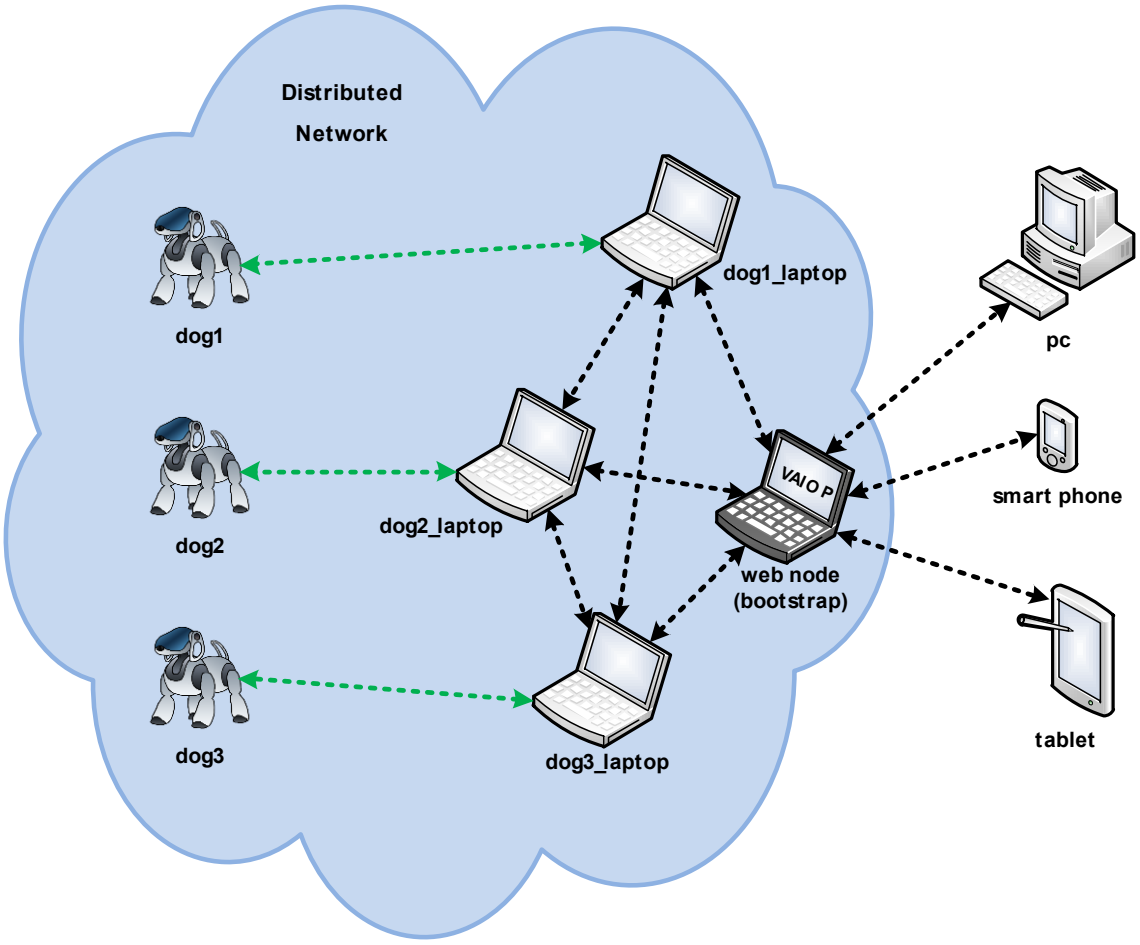

Figure 5.1. Device configuration

Also shown in Figure 5.1 are the external devices that can connect to the web node and load the web interface. Included are a tablet device, specifically an *iPad*, a smart phone, specifically an *iPhone*, and a desktop computer, loaded with *Windows XP*.

Each robot is linked statically to its prescribed laptop, *i.e.* “dog1” is hard coded to connect to “dog1_laptop”, *etc*. Each of the laptops upon which the middleware resides automatically obtains a network address via the Dynamic Host Configuration Protocol (DHCP). DHCP was selected for IP (Internet Protocol) assignment because it reflects not only the decentralized concept better, but also the dynamic connection and disconnection of nodes to the network without the need for reconfiguration.

As the robots become available, the information concerning each asynchronously appears on the web interface in the order in which they join the network. Figure 5.2 shows a sample of this interface. Upon double-clicking a device on the web interface, the user is presented with a list of the capabilities of the device, the network information, and any device-specific messages.

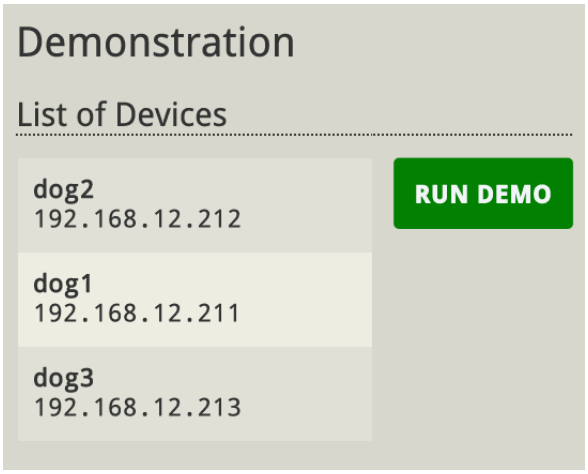

Figure 5.2. Web interface with list of devices

To demonstrate the functionality of this approach, the robots were placed in an enclosed environment and were assigned the task to find a randomly placed red target. Upon finding the target, the robots had to communicate with one another to determine which one was to approach the red target based upon distances from the targets. The robot with the shortest distance was to approach the target.

To test the response to the dynamic changes in the system, two tests were performed: one in which the all devices survived throughout the experiment, and one in which the robot assigned to approach failed mid-experiment. For each experiment, the target was placed after devices had already started the search algorithm. To begin each test, the robots were placed in the environment without the red target present, as in Figure 5.3. In the following figures, each dog can be identified by the number of yellow stripes on the head, *e.g.* dog2 has two yellow stripes.

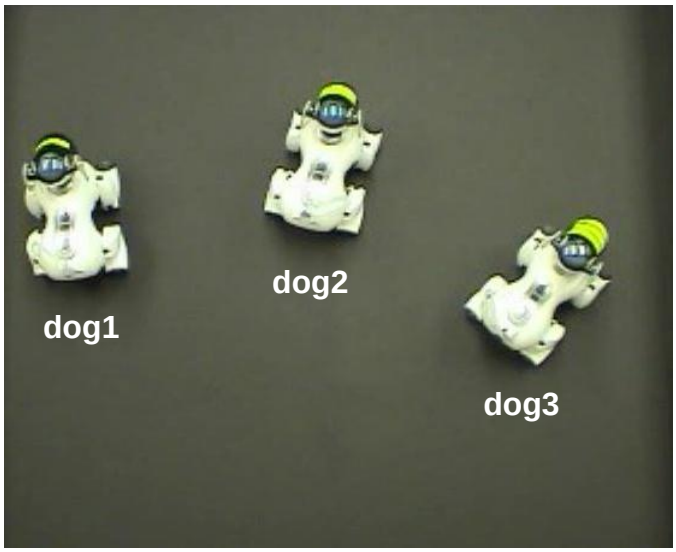

Figure 5.3. Robots in environment without red target

To initialize the experiment, the task must be assigned. On the web interface, this task was hard-coded into the “Run Demo” button that appears in Figure 5.2. Upon clicking it, a remote procedure call is executed that sends the task to the devices capable of executing it – in this case, the three robotic dogs.

As the robots performed the search sequence, the target was placed at such a distance that each device would report a different value from the proximity sensor as shown in Figure 5.4. As the robots completed the search algorithm, they would report to one another, via the DHT, the value obtained from the proximity sensor. Then, they would decide amongst themselves which robot should approach the target. This decision-making period is shown in Figure 5.5.

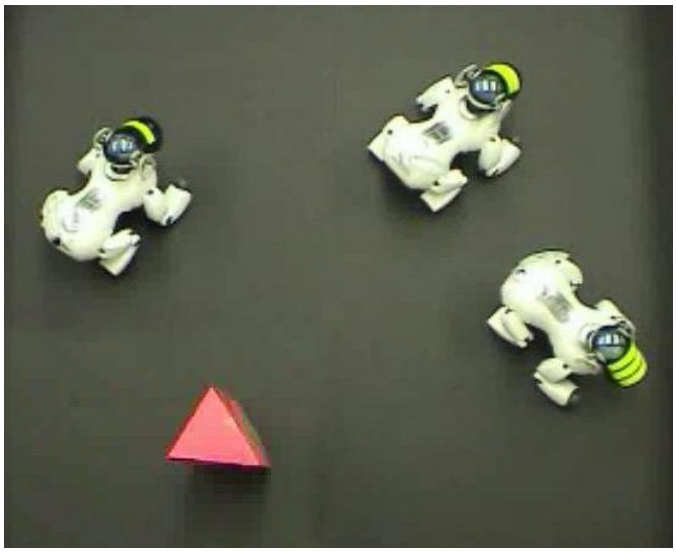

Figure 5.4. Robots in environment with red target

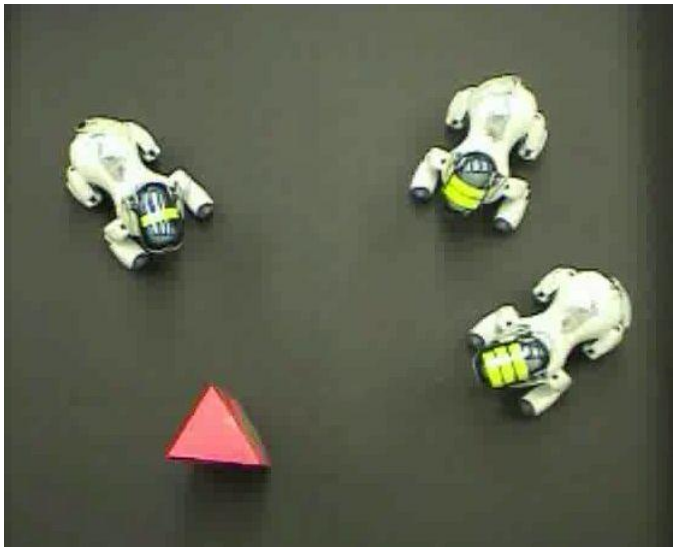

Figure 5.5. All robots have found the red target

After deciding which robot should approach the target, the assigned robot completes the task by approaching the red target, as shown in Figure 5.6.

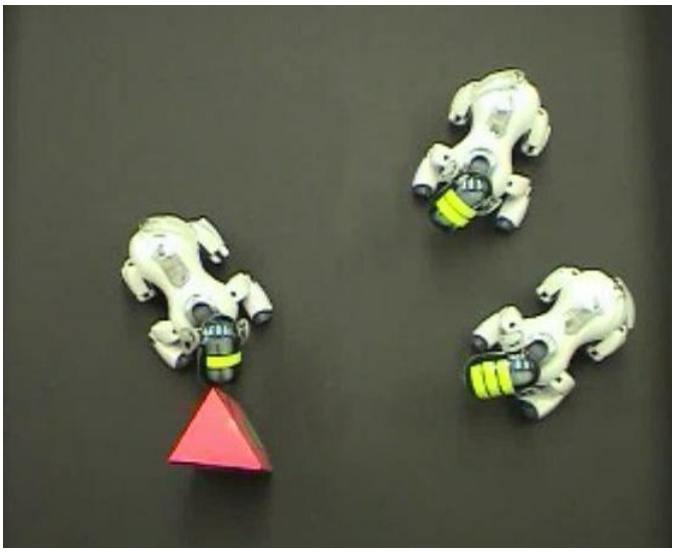

Figure 5.6. Assigned robot approaches red target

For the second experiment, the goal is to show the flexibility of the system when the assigned device fails while attempting to complete the task. If, in the previous scenario, “dog1” (the robot closest to the target) were to become disconnected from the network as it approached the target (simulated failure), the agents on the other robots would realize that the assigned robot was no longer available. Then, the agents belonging to the remaining robots would compare against each other to determine which should now approach the target and

complete the task. As is evident from Figure 5.7, “dog3” completed the task that dog1 left unfinished.

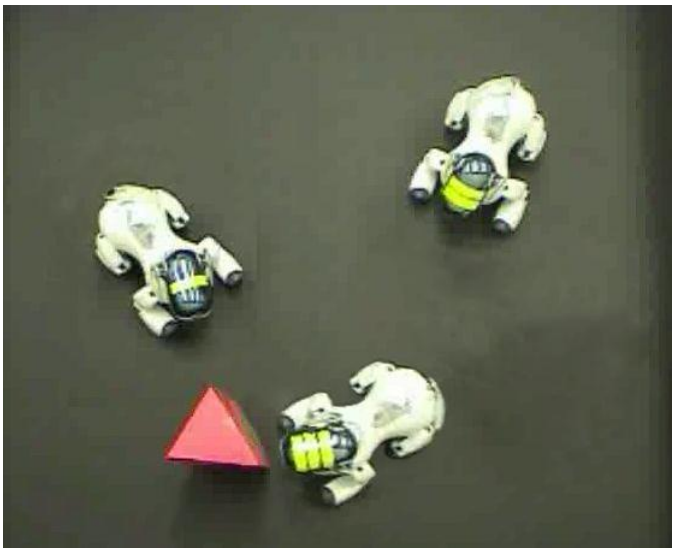

Figure 5.7. Reassigned robot approaches red target

Throughout the experiments, the status of each device is updated on the web interface asynchronously. Additionally, a live feed from the camera of each device is visible to monitor the progress. Therefore, any user observing the system has real-time information similar to that shown in Figure 5.8. All the while, the operator can modify the system and the environment as well as send tasks and messages.

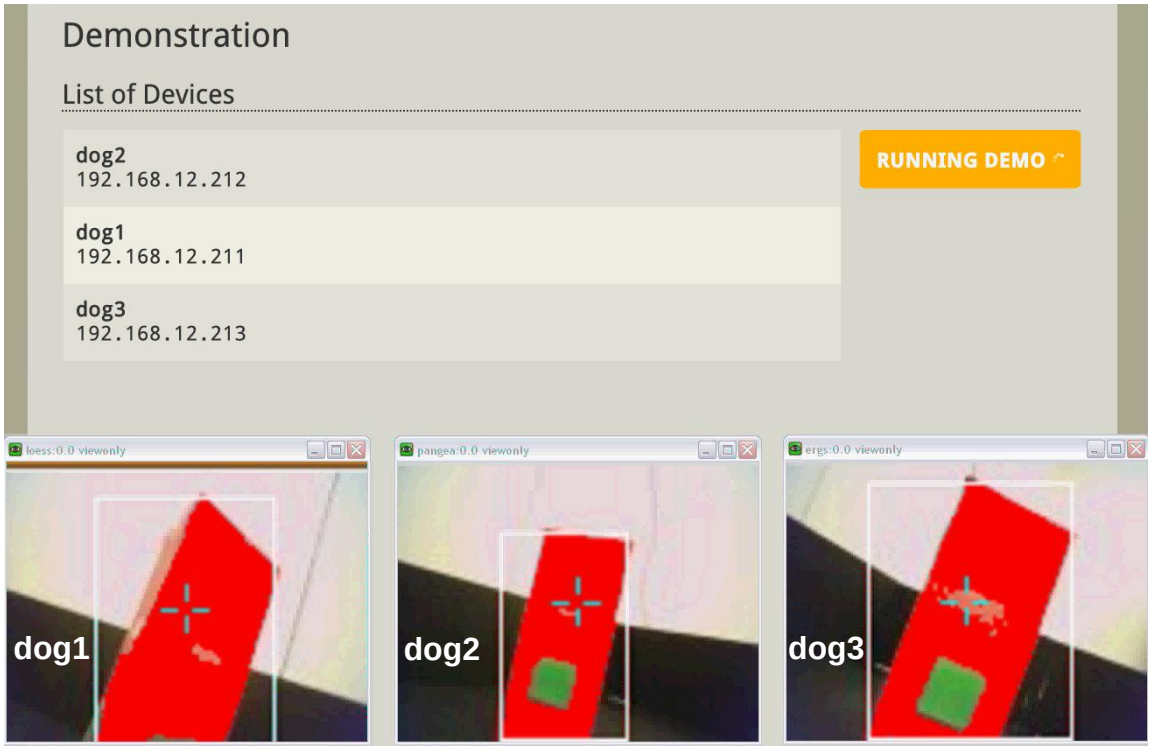

Figure 5.8. Web interface during experiment

6. CONCLUSION

Lightweight devices can run advanced middleware given the correct design approach. Using the principles of peer-to-peer networking, a sustainable distributed and decentralized network is attainable while still keeping the footprint of the middleware small. Therefore, the distributed system can scale to a large number of devices. When solving the consensus problem directly on the devices, the larger the number capable of observing and acting upon the environment, the better.

Contained in this work is the methodology not just for creating a lightweight middleware, but also for using the middleware to solve consensus problems as determined by users. In addition to this functional approach, the system is designed such that the labor involved in establishing, modifying, and interacting with devices is minimal.

A web interface also improves the user-system interaction. By presenting the usable information of the system along with the status of each device, a user acting remotely has useful, up-to-date information. Additionally, the web interface provides a means for users to assign tasks that enable the devices to solve the consensus problem using the prescribed algorithms.

The concept of dynamic distributed systems is an emerging field with promise for further potential research. While this thesis primarily focused upon the design

of a middleware that enables the communication and collaboration of networked devices, future work might focus upon multi-task assignment or improved optimization strategies.

REFERENCES

[1] R. Tomlinson. (2010, April) The first network email. [Online]. http://bbn.com/resources/pdf/FirstemailRTN.pdf

[2] M.J. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," *Foundations of Computation Theory*, vol. 158, pp. 127-140, 1983.

[3] B. Johansson, A. Speranzon, M. Johansson, and K.H. Johansson, "On decentralized negotiation of optimal consensus," *Automatica*, vol. 44, no. 4, pp. 1175-1179, April 2008.

[4] A. Tahbaz-Salehi and A. Jadbabaie, "A necessary and sufficient condition for consensus over random networks," *IEEE Transactions on Automatic Control*, vol. 53, no. 3, pp. 791-795, April 2008.

[5] Wikipedia. (2011, June) Network topology - wikipedia, the free encylopedia. [Online]. http://en.wikipedia.org/wiki/Network_topology

[6] R. Baldoni, S. Bonomi, A.-M. Kermarrec, and M. Raynal, "Implementing a register in a dynamic distributed system," in *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems*, Montreal, QC, 2009, pp. 639-647.

[7] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332-383, August 2001.

[8] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, et al., "An efficient multicast protocol for content-based publish-subscribe systems," in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, Austin, TX, 1999, pp. 262 -272.

[9] P.R. Pietzuch and J.M. Bacon, "Hermes: a distributed event-based middleware architecture," in *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*, Vienna, Austria, 2002, p. 661.

[10] M.J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374-382, April 1985.

[11] M. Barborak, A. Dahbura, and M. Malek, "The consensus problem in fault-tolerant computing," *ACM Computing Surveys*, vol. 25, no. 2, pp. 171-220, June 1993.

[12] D. Culler, P. Dutta, C.T. Ee, R. Fonseca, J. Hui, et al., "Towards a sensor network architecture: lowering the waistline," in *Proceedings of the 10th Conference on Hot Topics in Operating Systems,* Volume 10, Santa Fe, NM, 2005, p. 24.

[13] N. Gall. (2005, July) Update on the origin of the term "middleware". [Online]. http://ironick.typepad.com/ironick/2005/07/update_on_the_o.html

[14] Oxford University Press. (2011, March) Middleware, n.: oxford english dictionary. [Online]. http://www.oed.com/view/Entry/250908

[15] The Globus Alliance. (2011, June) Home. [Online]. http://www.globus.org/

[16] H.S. Sarjoughian, B.P. Zeigler, and S. Park, "Collaborative distributed network system: a lightweight middleware supporting collaborative devs modeling," *Future Generation Computer Systems*, vol. 17, no. 2, pp. 89-105, October 2000.

[17] S.J. Russell and P. Norvig, *Artificial intelligence: a modern approach (2nd ed.)*. Englewood Cliffs, NJ: Prentice Hall, 2002.

[18] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," *SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149-160, October 2001.

[19] A. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg,* London, UK, 2001, pp. 329-350.

[20] P. Maymounkov and D. Mazières, "Kademlia: a peer-to-peer information system based on the XOR metric," in *Revised Papers from the First International Workshop on Peer-to-peer Systems,* London, UK, 2002, pp. 53-65.

[21] Twisted Matrix Labs. (2011, June) Home. [Online]. http://twistedmatrix.com

[22] J. Resig. (2011, May) Jquery: the write less, do more, javascript library. [Online]. http://jquery.com

[23] Ubuntu. (2007, April) Meeting minimum hardware requirements. [Online]. https://help.ubuntu.com/7.04/installation-guide/i386/minimum-hardware-reqts.html

[24] Pyro, Python Robotics. (2010, February) using the sony aibo robot. [Online]. http://pyrorobotics.org/?page=Using_20the_20Sony_20AIBO_20Robot

VITA

Name:	Keith Anton Hall
Address:	241 Zachry Engineering Building 3131 TAMU College Station, TX 77843-3131
Email Address:	khall@tamu.edu
Education:	B.S., Industrial Engineering, Texas A&M University, 2008 M.S., Industrial Engineering, Texas A&M University, 2011