

A Decentralized Algorithm for Coordinating Independent Peers: An Initial Examination

Girish Suryanarayana and Richard Taylor

Institute of Software Research, University of California, Irvine
{sgirish, taylor}@ics.uci.edu

Abstract. Peer-to-peer (P2P) applications are composed of a distributed collection of peers that cooperate in order to perform some common task. Though P2P applications have attracted the attention of researchers, there has been little exploration of the deep issues; rather initial attention has been on widely known but technically shallow applications such as found in Napster and Gnutella. One particularly rich domain for examining the utility of P2P applications is distributed, decentralized crisis response. This paper studies the applicability of a peer-to-peer approach in such an emergency response situation. We have developed a prototype peer-to-peer infrastructure that models a group of firefighters who communicate with each other while fighting fires. Each firefighter (peer) runs a novel distributed "k-server" algorithm that makes local autonomous decisions based on the information received from other firefighters. While this study was limited in that it used a simulation to study the algorithm, the emergent behavior observed suggests that further, more detailed investigations are warranted.

1 Introduction

Communication, completeness, and accuracy of information and decision-making are the most important characteristics of applications that belong to an emergency response domain. Typically, in such situations, a centralized authority gathers information about the system from workers in the field and makes decisions regarding their placement. Accordingly, workers are distributed across the system, with each one carrying out his tasks. This central control can be a bottleneck and can cause disastrous delays. Further, in an emergency situation there is not enough time and resources to set up a centralized authority and follow a command structure.

Peer-to-peer (P2P) applications are composed of a distributed collection of entities (called peers) that cooperate in order to perform some common task. The enormous success of systems like Napster [2], Gnutella [3], Magi [5] etc. has given a major boost to the P2P paradigm. Consequently, there has been an increase in the awareness of the potential utility of the P2P approach. However, decentralized emergency response applications have been largely unexplored. As stated above, in an emergency response application, setting up a centralized solution is often not practical given short time frames or physical constraints. P2P applications, however, can operate without a

central infrastructure and are self-organizing thereby reducing set up time and resource costs.

In this paper, we describe our approach of applying P2P techniques to an emergency response application. We first present a model of the system, mapping elements from the emergency response domain to P2P concepts. A set of parameters is associated with every peer and request that determines their characteristics.

Then, we present a distributed algorithm that is run on every peer in the system. Our algorithm is a variant of distributed solutions to the “ k -server” problem [24]. In the k -server problem, servers are placed in a virtual ‘space’ and can move within the space to serve requests. Over time, requests arrive at points in the space. As soon as a request comes in, a server must be moved in order to serve the request. The act of movement incurs some cost. The goal is to decide which servers to move so that a sequence of requests can be served with a cost as small as possible [21].

To evaluate the algorithm we built a simulator that simulates a group of firefighters trying to extinguish brush fires. We refined the algorithm gradually by experimenting with various input parameters and observing the resultant system behavior. Evaluation results primarily show that though no firefighter has complete knowledge of the system and is unaware of the state of other firefighters, the universal system goal is nonetheless achieved, thus exhibiting an emergent behavior.

Our approach draws from a large number of computer-related disciplines including networking, distributed algorithms, wireless communications, and graphical real-time simulations. This paper contributes a novel distributed k -server algorithm that is executed on every peer and a P2P infrastructure that allows extensive experimentation to evaluate P2P approaches and techniques. The paper also represents a significant data point in the feasibility and utility of P2P approaches.

2 Motivation

Consider the scenario where groups of firefighters are battling a large brush fire that is spread over many square miles. It is essential that a firefighter be able to communicate to the rest of the firefighters various kinds of information, for example, the intensity of the fires, the direction in which the fires are spreading, the direction of winds etc. This communication is necessary because critical decisions need to be made based on the information that each of the firefighters provide, for instance, which fires should be tackled first and by whom. Since a wrong decision can result in a disaster, it is imperative that the information provided be complete and correct so that correct decisions can be made by the system. Moreover, a firefighter should also be able to call for assistance in fighting a fire and be able to provide information regarding the status of the fire he is fighting. Thus, it is vital that all the firefighters in the system stay connected as much as possible.

The demands on communication, completeness and accuracy of information and decision-making are not limited to a fire-fighting scenario. There are several other scenarios where a similar solution is required. These include other emergency situa-

tions that arise during natural disasters like earthquakes, floods, and storms. Sensor devices in an exploratory mission, robots in hazardous situations and soldiers in a hostile territory are other cases where a group of entities work autonomously to attain a common system goal.

The US Department of Labor states “In national forests and parks, forest fire inspectors and prevention specialists spot fires from watchtowers and report their findings to headquarters by telephone or radio. When fires break out, crews of firefighters are brought in to suppress the blaze using heavy equipment, hand-tools, and water hoses.” [1] In a typical scenario, if the necessary infrastructure to coordinate firefighting exists and is in place, the firefighters will first evaluate the fires and inform the central controller at a distant location about the intensity and capacity of each fire. Based on such information received from various firefighters, the central controller will then make decisions about the importance of each fire and the allocation of firefighters among the fires. Each firefighter will accept directions from the controller and move towards the fire that he has been assigned to deal with. However, this central authority is, in fact, a bottleneck: commands are sent out in a sequential manner introducing arbitrary delays and the possibility of a system failure at the command post. Moreover, one cannot know the locations of fires that will occur in the future and accordingly cannot set up a system in advance to fight the fires. When an actual fire occurs, response will be slowed because of the requirement to utilize the centralized control. Furthermore, communication is limited because of distance and terrain, so in some circumstances the centralized control structure may break down completely.

3 Background

3.1 Peer-to-Peer Applications

In a peer-to-peer architecture, each of the peers can act both as a client and a server. When a peer requests a service, it acts like a client, and when it fulfills a request from another peer, it adopts the role of a server. Requests for service are called peer-to-peer *events*.

Peer-to-peer (P2P) applications came to the fore primarily due to Napster [2], which was an immensely popular system for exchanging music files. Napster allows a user to first search for a file using a keyword and returns a list of all peers who have the file. The user then connects directly to a peer and downloads the file without involving any intermediate server in the process. However, since Napster uses a central server to keep track of all the clients that are connected to it and which files they are sharing, it is not considered as a pure P2P application by many. However Clay Shirky in [7] claims Napster is a P2P application because it allows for variable connectivity by having the addresses of Napster nodes bypass DNS and gives peers at the edges of the network significant autonomy by moving control of file transfers to them.

In contrast, an application that implements the Gnutella protocol [3] does not rely on a central authority to manage the network or to broker transactions. A peer con-

nects to a Gnutella host and uses a ping-pong message sequence to locate other peers nearby [8]. When a peer needs a file, it queries the peers it is connected to. The peers search on their system for the requested file and also pass the request onto other peers to whom they are connected. The results are returned to the querying peer who can then select a peer and download the file directly from that peer.

Though P2P rose to the forefront primarily due to music sharing, it has also been used in collaborative tools such as Groove [4]. Groove is a decentralized Internet-based platform for secure collaboration. Groove users make immediate and direct connections with other users for a wide variety of activities including: working on a project, sharing drafts, coordinating schedules and discussing issues etc. Groove also offers communication tools like instant messaging and threaded discussions as well as content-sharing tools like shared files and shared contacts. It also permits other features like co-browsing and co-editing of Microsoft Word documents. Activity in Groove occurs in a “shared space”—a secure space in which group members carry on conversations. A copy of the Groove shared space is stored locally on the computers of each of the members of the space. When one member adds something new to the space, that change is reflected on everyone’s machine – thus every member of the space gets a single consistent view of the project’s data [9].

Micro-Apache Generic Interface (Magi) is an open architecture framework that facilitates messaging and deployment across a broad range of computing platforms [6]. Magi is a general-purpose, peer-to-peer infrastructure for both embedded and enterprise applications. It offers an end-user application for installation on desktops, laptops and even PDA devices and allows secure collaborative applications to run on these machines. It also offers features like building work team communities, file sharing, chat and messaging. Each Magi peer provides a set of core services that can be extended to create modules for each peer based on the services it provides and the capabilities of the platform on which it is installed [5].

The aforesaid developments in the P2P world have brought this field to prominence. This has led to awareness of the potential of the peer-to-peer approach among the research community.

3.2 Mobile Ad-hoc Networks

A mobile ad-hoc network (MANET) is a collection of mobile wireless nodes that form a temporary network without the aid of a centralized administration or standard support services that are regularly available on conventional networks [10]. Such a collection of mobile hosts that enables communication between users without an infrastructure is called an “ad-hoc” network. It can be deployed rapidly because it needs no infrastructure. In a mobile ad-hoc network different hosts communicate over wireless links and messages may traverse multiple wireless links before reaching their destination. Since mobile nodes tend to “wander around” changing their location, ad-hoc networks must deal with frequent changes in topology as well as one-way connectivity and transient connectivity. Consequently, research on mobile ad-hoc networks has concentrated primarily on routing protocols that are necessary to facilitate the ex-

change of packets between two hosts that may not be able to communicate directly [12][14].

In the algorithm that we have developed, firefighters communicate using message broadcast. Research results in mobile ad-hoc communication have shown that message broadcast communication tends to consume excessive network bandwidth [16]. However, the MANET community has developed routing algorithms that enable efficient message distribution that could be leveraged in the implementation of our algorithm [11] [13].

3.3 Control Algorithms

A considerable amount of effort has gone into research on the online " k -server" problem in algorithm literature [20] [22]. In the k -server problem, first defined in [21], initially each server is positioned at some point of the metric space. A metric space can be considered as an edge-weighted graph over n vertices with edge weights corresponding to distances between the endpoints. Requests arrive for service at points of the space. When a request comes in at some point of the space, a server must be moved to that point (if none is there) in order to serve the request. Besides the server that moves to the point of request, the other servers are also free to move so that they can position themselves favorably for handling subsequent requests. When a server moves, a cost equal to the distance it covers is incurred. The cost of a request is equal to the sum of the costs incurred by all servers during the service of that request. The cost of a sequence of requests is equal to the sum of the costs of all requests in the sequence.

The goal of researchers has been to design algorithms that will decide which servers to move when a request arrives so that any sequence of requests can be served with a cost as small as possible. Whereas in an off-line problem the complete input is known and the goal is to search for a time-efficient algorithm, in the case of an online problem the input is not known beforehand but is revealed only during the execution of the algorithm. We deal only with the on-line k -server problem because, for us, the sequence of requests for service is not known in advance, but revealed request by request [19].

The algorithm research community has mostly concentrated on centralized solutions to the " k -server" problem [22] [23]. Such a typical solution consists of a global-control algorithm that gets the requests and governs the motion of the servers with no cost incurred for acquiring the knowledge about all the requests and for transmitting motion instructions to the servers.

A distributed k -server algorithm [24] has to deal not only with the problems of lack of knowledge of future requests but also with incomplete information about previous requests. While a centralized solution to the " k -server" problem incurs cost only when a server moves from one location to another, in the case of a distributed algorithm additional cost is incurred because of the transmission of messages among the servers.

We map the “ k -server” problem to our firefighters application. Fires are the points of space that require service. Each firefighter acts like a server that moves to a fire to service it and the cost incurred is the energy that is expended by a firefighter to do certain tasks.

Though the distributed k -server problem is the basis of our problem, our domain introduces significant differences from the traditional k -server problem. Each point of request (fire) may require more than one server (firefighter) to service it. A firefighter may provide service only when his energy level is above a certain threshold. When a firefighter moves from one location to another, he spends energy that depends on the distance he moves. While a server does not incur costs while servicing a request in an ideal distributed “ k -server” setting, a firefighter exhausts most of his energy in fighting fires. The problem is thus to find a satisfactory solution that will decide which firefighters to move towards which fire so that all the fires are extinguished in as little time as possible with minimum energy expenditure.

4 Approach

To address this issue we first modeled the system, mapping elements from the fire-fighting domain such as fires and firefighters to P2P concepts such as work requests and peers, respectively. After examining the problem, we developed an algorithm that is a variant of the distributed “ k -server” algorithm specifically to address the firefighter problem.

4.1 “Firefighter” Model

In the firefighter model each firefighter takes the role of a peer that communicates with other peers to exchange information. Every firefighter is assumed to have a device (like a Personal Digital Assistant) that not only enables communication but also permits information storage and computing. Each firefighter has four salient properties: his position (expressed as an (x, y) coordinate in a two-dimensional plane), his maximum energy capacity (expressed as an integer), his energy level (expressed as an integer) and his current task (resting, looking for fires, moving towards a fire or fighting fires). Energy is consumed when a firefighter moves towards a fire and when he fights the fire. A firefighter gains energy when he is resting or looking for new fires. Associated with each firefighter is an attribute reflecting how much energy he expends running and fighting fires. The energy level of a firefighter dictates his behavior. There are three energy thresholds: critical energy threshold, minimum energy threshold and the maximum energy threshold. If a firefighter is currently *en route* to a fire and his energy level drops to the minimum energy threshold he must stop and rest. However, if he is fighting a fire when this happens, he continues fighting until either the fire is extinguished or his energy level reaches the critical energy threshold. When his energy level goes below the critical energy threshold, he must stop and rest irrespective of his current task. A firefighter who has completed work on a fire will rest

until his energy capacity has reached the maximum energy threshold. We have defined a firefighter's maximum energy threshold to be 80%, the minimum energy threshold to be 20% and the critical energy threshold to be 1% of his maximum energy capacity. The algorithm, explained later, determines a firefighter's status and current task.

In this model we associate "hot spots" of the fire with P2P events. These events are passive objects that are manipulated by firefighters. Each fire is created requiring a certain amount of effort to be extinguished. In this model, the amount of effort required to extinguish a fire does not increase over time and only decreases by the actions of working firefighters.

Similar to other pure peer-to-peer applications, the communication among firefighters occurs without the intervention of any central routing authority. A firefighter peer does not need to know about the existence of other firefighters in the system. We limit communications and information exchange exclusively to radio and visual proximity to an event. This implies that any two firefighters that are not in radio range cannot communicate directly; however, if an intermediate is available he can act as a proxy and can forward the messages he received to other firefighters within his range. Because of the scale of the environment the impact of other forms of communications such as verbal and visual gestures can be ignored without loss of expressive power. We also assume that the digital communication system follows a probabilistic model of delivery; that is, message delivery is not guaranteed. In reality, digital communications normally have a sharp drop-off point at a fixed distance; however, various obstructions can cause significant signal attenuation resulting in decay. The radio model is simplified to facilitate implementation. In our model, as in a typical radio system, the strength of the signal is inversely proportional to the square of the distance from the transmitter (inverse square law). The likelihood of the message being received is directly proportional to the signal strength. The messaging model used is broadcasting, with a static hop-count. Each receiver will automatically rebroadcast a received message (with a decremented hop-count) if its hop-count is greater than zero.

4.2 System Architecture

Fig. 1 is a simple illustration of the architecture of the "Firefighters" model. Firefighters communicate with each other through the universal radio band that acts as a connector bus. Any message that is sent to the radio band is forwarded to all other firefighters *provided they are within radio range of the sender*. Thus the radio band also acts like a "distance filter" since it filters messages based on the distance from the sender.

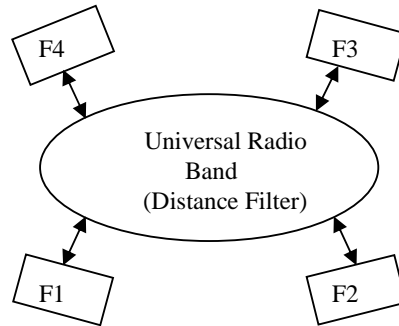


Fig. 1. Conceptual Model of the system architecture. *F* refers to firemen

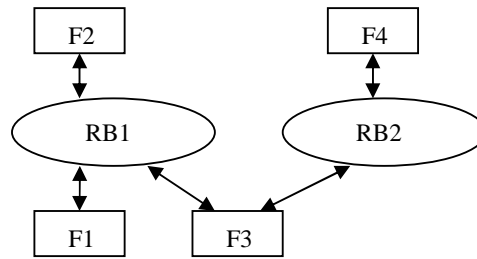


Fig. 2. Distance-limited Model of the system architecture. *RB* refers to radio band

An alternative view of the architecture can be seen in Fig 2. In this diagram, the radio band has been divided to illustrate the distance-limited communication among firefighters. RB1 and RB2 are two parts of the universal radio band. F1, F2 and F3 are all within communication range of each other and so they are connected to RB1. However, F4 is within range of only F3 and so F3 and F4 are connected to RB 2. However, this view of the architecture is not constant since the position of firefighters is not fixed. The architecture is dynamic and may change as the firefighters move to new locations.

4.3 Algorithm

For our firefighters application, we execute a decentralized “*k*-server” algorithm on each of the firefighter peers. When a firefighter discovers a fire he sends out a message with information about the fire. If he has no other destination, he moves toward

the fire. When a firefighter without a current destination receives a message informing him about the occurrence of a fire, he replies with information about his current energy level and location. He then begins moving toward the fire. The first firefighter to arrive at a fire is elected that fire's leader. He is responsible for determining how much effort the fire requires and calculates the number of firefighters required to fight the fire based upon its size. The leader's PDA device extracts energy level and location information from all the messages that it received from the firefighters in response to the initial message. It evaluates the energies of all the firefighters headed towards the fire and selects the most suitable amongst them. It then notifies those firefighters approaching the event if they are needed or not.

We conservatively tell only those firefighters who are not needed to stop coming; in the event of lost messages, extra firefighters arrive rather than insufficient firefighters. While this may not be the optimal allocation of effort, it is "safe" in the sense that at least the required number of firefighters will arrive, providing they are available. Firefighters continue to fight the fire until either it is extinguished or their energy level drops below the critical energy level.

If a firefighter is free and has no knowledge of any new fire or existing fires requiring more firefighters, he assists other firefighters in fighting fires even though he is not required. This helps to extinguish all the fires in the system faster. This also avoids the extreme fatigue of certain firefighters who fight fires that require few firefighters but more effort. If a fire event has only a fraction of the necessary firefighters it can still be extinguished although it will take a longer amount of time. Following is the pseudo-code for our distributed algorithm.

```
Algorithm {
  updatePossibleDestinations();
  if destination is not null {
    if firefighter is at destination
      fightFire();
    else
      moveToDestination();
  }
  if destination is null {
    if firefighter has sufficient energy
    {
      setNewDestination();
      if destination is still null
        lookForNewFires();
      else if he has a destination
```

```

        moveToDestination();
    }
    else if firefighter is exhausted relax();
}
updatePossibleDestinations() {
    Get a list of all visible fires and add to it a list of
    known fires that are not directly visible.
    Remove from this list fires that do not need more fire-
    fighters and fires that have been extinguished;
    Send "AssistRequestMessage" for all visible fires.
}
fightFire() {
    currentStatus is "Fighting";
    if firefighter is assigned to the fire    {
        destination.workEffort();
        Decrease the energy level of the firefighter by a fixed
        amount;
        If he is the leader selectFiremen();
        If the energy level is below the critical energy level,
        remove self from fire and rest;
    }
    else destination of firefighter is set to null;
}
selectFiremen() {
    Sort through a list of messages received, on the basis
    of energy levels of firefighters approaching fire;
    Tell firefighters that are not needed to choose other
    destinations by sending "AbortMoveMessage"
}
moveToDestination() {
    currentStatus = "Moving to fire";
    Increment position of the firefighter;
    Decrease energy level by fixed amount and if it goes
    below the minimum threshold set the destination to
    null;
}
setNewDestination() {

```

```

Select the nearest fire from the list of possible destinations;
}
lookForNewFires() {
    Increment position of the firefighter;
    currentStatus is "Looking for Fires";
Increase energy level by a fixed amount only as long as
it is below the maximum energy capacity;
}
relax() {
    currentStatus is "Resting"
Increment energy level by fixed amount only as long as
it is below the maximum energy capacity;
}

```

There are four types of messages that are used by firefighters to communicate in our algorithm, namely, *AssistRequest*, *ResourceStatus*, *AbortMove* and *FireOut*. Each message consists of a unique identifier and a hop-count and contains information about the originator and the sender of the message. A firefighter rebroadcasts a message only if the hop-count is greater than zero. Every time the message is re-broadcast, the hop-count is decremented by one.

When a firefighter discovers a fire he broadcasts an *AssistRequest* message to other firefighters. When another firefighter receives an *AssistRequest* message, he puts the location of the event in his possible list of destinations. If the firefighter has no destination (i.e. is wandering and searching for fires), he sets his destination to that of the new fire and broadcasts a *ResourceStatus* message. This message contains information regarding the current energy level of the firefighter and his current location. On receiving these *ResourceStatus* messages, the leader firefighter's PDA evaluates the energy levels of each of the firefighters moving towards the fire. He then generates an *AbortMove* message, which is addressed only to those firefighters who are not needed. On receiving an *AbortMove* message addressed to him, the firefighter either starts moving towards another fire or continues searching for new fires. When a fire is extinguished, a *FireOut* message is sent by the leader firefighter.

Since most messages hold *some* interest for the receivers, the broadcast model is deemed necessary for the system to function properly. Only one message sent has an explicit receiver (*AbortMove*), and that is used only to request additional actions on that unit's part; our assumption that messages may be dropped and our use of rebroadcasting proxies further illuminates the need for broadcasting even in situations where messages have a specific destination. For best performance, all the firefighters need to receive all the messages. Firefighters who receive only a subset of messages will not perform as well as those who do. Hence the broadcasting communication model best suits the "firefighters" application.

5 Evaluation

This section describes techniques and tools that were used to evaluate the P2P infrastructure and the algorithms that we developed.

5.1 Simulator

We built a multi-threaded asynchronous application in Java to simulate the Firefighter problem. To preserve autonomy among the firefighters, we model them as individual threads. This prevents any form of implicit synchronization between their interactions. No explicit synchronization is performed except when transmitting and receiving messages, since a radio frequency can only accommodate one message at a time. Fires are generated at random locations. An independent thread known as an *EventManager* creates random events (fires) at random times.

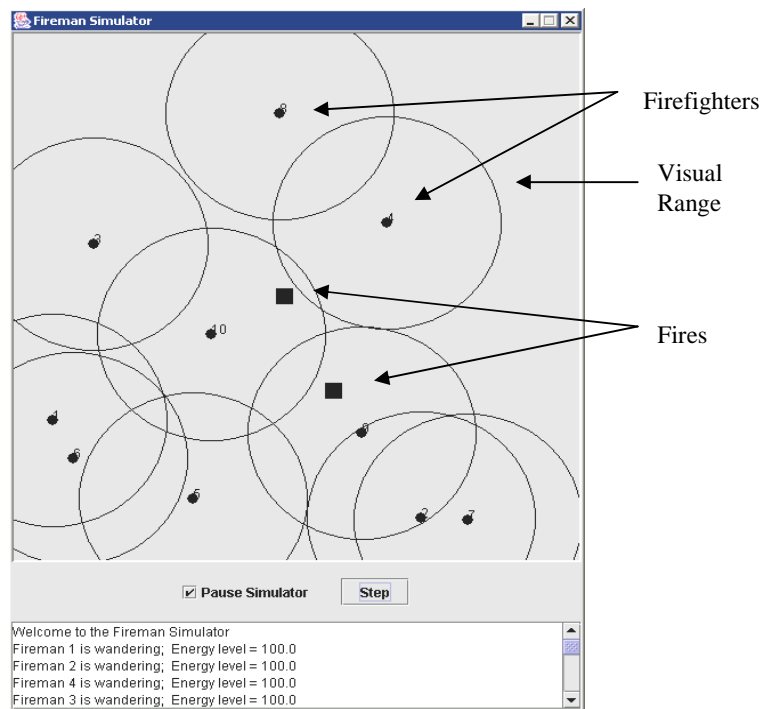


Fig. 3. Simulator Screenshot with two fires and ten firemen

An annotated screenshot of our simulator is shown in Figure 3. Each numbered black dot represents a firefighter and the black squares represent fires. The big circles around each dot represent the range of sight of the firefighters.

5.2 Algorithm Evaluation

Using the simulator, we evaluated the algorithm on the basis of the time required to extinguish all the fires and the exhaustion level of the firefighters. Input parameters were identified and the behavior of the system was studied as we varied the parameters. The feedback obtained was then used to refine the algorithm.

We first experimented with varying hop-counts for re-broadcasting and discovered that if the message hop count was too high, it resulted in packet flooding. With 5 firefighters and maximum of 10 concurrent fires, we found that a static hop count of 5 was most favorable for our application. The next parameter we studied was the number of firefighters in the system. We varied the number of firefighters, keeping the maximum number of concurrent fires constant. Though the time required to extinguish all the fires depends partially on the random work effort required per fire, we found over a number of simulations that the time required for extinguishing all the fires decreased noticeably when the number of firefighters was increased, indicating that our basic assumptions about the overall system behavior were correct.

Changing the threshold energy levels of the firefighters had a remarkable effect on system behavior. More specifically, when the minimum energy threshold of the firefighters was lowered, the overall resultant exhaustion of the set of firefighters as a whole decreased, as did the time required to extinguish all fires.

We dropped some messages during the simulation to model signal decay and studied the resulting behavior of the firefighters. We observed some expected inefficiencies in the system. For example, firefighters who did not receive a *FireOut* message would continue moving towards the location of the fire even though it no longer existed. To address this problem, we refined our algorithm so a firefighter will check for the existence of the fire visually once it is in range of his sight. In spite of not using a guaranteed message delivery or message acknowledgement scheme in our communication model, we noticed that our algorithm ensured that the system goal was eventually achieved even with some dropped messages.

Though no firefighter has complete knowledge of the system and is unaware of the behavior, location, and number of other firefighters, an emergent behavior is observed when each firefighter executes the algorithm and makes local autonomous decisions. We found that, as long as the rate of new fire creation is less than the rate at which firefighters put out fires, all the fires are extinguished. One exception is that fires must be discovered to be fought, but this issue is inherent in the problem and would be the case in a centralized solution as well.

6 Future Work

The firefighter application demonstrates the feasibility of using the peer-to-peer approach in an emergency response situation. The following sub-sections discuss how we plan to add new features to the existing infrastructure in the future.

6.1 Limitations

Broadcast of messages leads to inefficient utilization of network bandwidth. The number of messages that are broadcast increases exponentially with the number of firefighters in the system. Our future work will include the evaluation of routing protocols based on those developed by the MANET community [15] and the subsequent selection of routing algorithms for our P2P infrastructure. In the current application, when a firefighter broadcasts a message he does not know whether it was received by any one. A message acknowledgement scheme can be used to notify the sender of received messages. This can be used to increase the reliability of message delivery since a sender can resend his message if he does not receive any acknowledgements.

Our algorithm uses a simple leader election mechanism to choose the leader for a fire. In the future, we will consider the application of leader election algorithms developed by the MANET community for our algorithm [17] [18].

In our simulator, we assume that unattended fires do not increase in size and intensity with time. Further, when a large fire is reduced to a small fire some firefighters should be able to leave that fire and move towards other demanding fires so that they do not grow beyond control. We plan to implement these changes to our system model and refine the algorithm accordingly. In the future, we also intend to compare the performance of our algorithm with other trivial solutions that are possible for such emergency response systems.

We also plan to experiment with different “wandering” algorithms to find one that enables firefighters to discover fires more effectively. We believe that enhanced communication among firefighters about their wandering behavior will allow all firefighters (as a whole) to discover fires more quickly than with random wandering.

Currently, the application uses a simple messaging framework consisting of just four kinds of messages. Though this set of messages addresses most of the information required by firefighters, it is not complete with respect to real world situations. In the case where a firefighter needs personal help because he is in danger, he needs to communicate the nature of help required to other firefighters in the system. Having a message type to convey this is a viable option, however, this may not solve the problem completely. Therefore, it may also be necessary to prioritize messages so that the most important messages are received and addressed first. We will also concentrate on extending the message framework to support more message types and developing a priority scheme for the messages.

6.2 Infrastructure Evolution

In the future, we plan to execute our distributed algorithm on wireless PDA devices (Compaq's IPAQ) to study the behavior of the system in the "field." All devices are also equipped with a GPS (Global Positioning System) unit so that each peer can exchange positional information with other peers. Our control algorithms will be implemented on top of Magi, which will be used as the P2P infrastructure running on the devices.

7 Conclusions

This exercise supports the contention that P2P architectures have significant, novel applications, arguing for further exploration of the topic. This exercise focused primarily on the decentralized algorithm used on each peer within a fire-fighting scenario. We demonstrate that a peer-to-peer solution can be applied to an emergency response situation effectively. We have developed a novel variant of the distributed "*k*-server" algorithm which, when executed, leads to an emergent behavior. Provided the existence of each of the fires is known to at least one firefighter, all the fires in the system are eventually extinguished if there are an adequate number of firefighters in the system. Though none of the firefighters possess complete knowledge of the system, they make local autonomous decisions that lead to a feasible system solution.

The P2P framework that we have constructed (in the form of our simulator) is scalable, multi-threaded and asynchronous, and provides an excellent framework for further experimentation. The flexibility of our framework allows us to inject other management models so we can compare them as well.

Applying the P2P approach to the emergency response domain has also brought to the fore a set of rules and principles for handling emergency situations. Clearly, a fuller exploration of this scenario will require experimentation in which actual distributed devices are used, such that the interaction between algorithm, P2P infrastructure, ad-hoc networking, power management, and so forth can be explored.

8 Acknowledgements

We would like to thank Michael Goodrich and Sandra Irani for their invaluable feedback and contributions.

References

1. Firefighting Occupations - Nature of the Work <http://www.bls.gov/oco/ocos158.htm>
2. Napster <http://www.napster.com>
3. Gnutella.com <http://www.gnutella.com>

4. Groove Networks, Inc., Platform for inter-enterprise communication and collaboration <http://www.groove.net>
5. Bolcer, G., Gorlick, M., Hitomi, A., Kammer, P., Morrow, B., Oreizy, P., Taylor, R. Peer-to-Peer Architectures and the Magi Open-Source Infrastructure. Endeavors Technology, Inc. (Dec 2000) <http://www.endtech.com/papers.html>
6. Bolcer, G. (Endeavors Technology, Inc.) Magi: An Architecture for Mobile and Disconnected Workflow. IEEE Internet Computing special edition on Internet-Based Workflow, (May/June 2000) 46-54.
7. Shirky, C. Listening to Napster in Andy Oram, ed. Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. O'Reilly & Associates, March 2001, 21-37.
8. Kan, G. Gnutella in Andy Oram, ed. Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. Andy Oram O'Reilly & Associates, March 2001, 94-122.
9. Udell, J., Asthagiri, N., Tuvell, W. Security in *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*. Andy Oram O'Reilly & Associates, March 2001, 354-380.
10. Johnson, D. Routing in Ad Hoc Networks of Mobile Hosts. Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (Dec 1994)
11. Perkins, C., Bhagwat, P. Highly dynamic Destination-Sequenced Distance Vector Routing (DSDV) for Mobile Computers. Proceedings of the SIGCOMM '94 Conference on Communications, Architectures, Protocols and Applications (Aug 1994), 234-244.
12. Johnson, D. Dynamic Source Routing in Ad Hoc Wireless Networks, Mobile Computing, Kluwer Academic Publishers, 1996.
13. Park, V., Corson, S. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. Proceedings of IEEE INFOCOM (April 1997).
14. Garcia-Luna-Aceves, J.J., Madruga, E.L. The Core Assisted Mesh Protocol. IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks, Vol. 17, No. 8, (Aug 1999) 1380-1394.
15. Royer, E., Toh, C-K. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks. IEEE Personal Communications Magazine (Apr 1999) 46-55.
16. Basagni, S., Chlamtac, I. Broadcast in Peer-to-Peer Networks. Proceedings of the Second IASTED International Conference European Parallel and Distributed Systems (Vienna, Austria July 1998) 117-122.
17. Malpani, N., Welch, J., Vaidya, N. Leader Election Algorithms for Mobile Ad Hoc Networks. Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL M for Mobility) (Aug 2000) 96-103.
18. Hatzis, K., Pentaris, G., Spirakis, P., Tampakas, V., Tan, R. Fundamental Control Algorithms in Mobile Networks. Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures (Saint-Malo, France June 1999) 251-260.
19. Borodin, A., Linial, N., Saks, M. An optimal online algorithm for metrical task system. Journal of the Association for Computing Machinery, vol. 39, (no.4) (Oct 1992), 745-763.
20. Koutsoupias, E., Papadimitriou, C. On the k-server conjecture. Journal of the Association for Computing Machinery, vol.42, (no.5) ACM (Sep 1995), 971-983.
21. Manasse, M., McGeoch, L., Sleator, D. Competitive algorithms for on-line problems Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (Chicago IL, May 1988) 322-333
22. Fiat, A., Rabani, Y., Ravid, Y. Competitive k-Server Algorithms. Proceedings of the 31st Ann. IEEE Symposium on Foundations of Computer Science, (October 1990), 454-463.
23. Grove, E. The Harmonic k-Server Algorithm is Competitive. Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, (May 1991), 260-266.

24. Bartal, Y., and Rosen, A. The Distributed k-Server Problem – A Competitive Distributed Translator for k-Server Algorithms. *Journal of Algorithms*, vol.23, (no.2), Academic Press (May 1997), 241-264.