DISTRIBUTED RESOURCE COORDINATION STRATEGIES
FOR MOBILE AD HOC NETWORKS

AN ABSTRACT SUBMITTED ON THE FOURTEENTH DAY OF APRIL 2006
TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER
SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
OF THE GRADUATE SCHOOL
OF TULANE UNIVERSITY
FOR THE DEGREE
OF
DOCTOR OF PHILOSOPHY
BY

Marcó M. Carvalho

APPROVED: Bill Buckles, Ph.D.

Sergey Drakunov, Ph.D.

Jing Peng, Ph.D.
ABSTRACT

This work was motivated by the distributed information processing needs in tactical environments such as disaster relief and military operations. Tactical network environments are characterized by mobile ad hoc networks under resource and policy constraints, tasked for critical missions.

Mission success directly depends on the effective use of computation resources in the field for distributed data processing and information propagation. However, the dynamic nature of the network and the lack of centralized coordination components make it very difficult to globally allocate and maintain resources for distributed tasks in a manner that is itself distributed, efficient and adaptive to the volatile nature of the environment.

Current approaches to the problem can be broadly classified in three main categories, centralized decision making (applicable only to small scale networks), local greedy decision making and arbitrage models also known as agent-based negotiation.

In this work, we introduce a new solution to the problem which utilizes online learning strategies at the local node level, to quickly evolve the global resource allocation solution that asymptotically converges to a global optimum.

The resource allocation problem in mobile ad hoc networks is first formulated as a k-arm bandit problem at the local level. As data flows through the network, each node...
locally learns the best policies to the used under different data flows, different constraints and local network topology.

Two learning strategies ($\varepsilon$-greedy and SoftMax) are adapted to the problem domain and used for tests and comparisons. A proof-of-concept implementation of the proposed resource allocation algorithm is introduced, discussed and tested in simulated networks.

The preliminary experimental results and the theoretical guarantees provided for the algorithm indicate that the approach is applicable to the resource allocation problem in mobile ad hoc networks for tactical environments.
Acknowledgement

I would like to first and foremost thank my advisor Dr. Bill Buckles and Dr. Clark Glymour for all the help, encouragement and support during the last few years. Without their help and guidance, this work would not have been possible.

I would also like to thank Dr. Alberto Cañas, who first encouraged me to start the PhD program and supported my efforts through the whole period. I must also recognize the support I had from Dr. Ken Ford, who was always there when I needed help.

I must also thank all my colleagues at the Institute for Human and Machine Cognition who believed in me and were always ready to help.

A special thanks to all the Faculty of the EECS Department at Tulane University who found the energy to support and encourage me, even under major adversities and personal losses. I will remember your courage and professionalism as an example to be followed in my career.

Above all, I want to say thanks to my wife Silvia and my daughters Isabel and Camilla who, once again, abdicated with me of many weekends and countless nights of family time to support this endeavor. They are the reason and the true deservers of this merit.

At different times during the last few years, this work has been partially supported by the Louisiana Transportation Research Center under contract number 736-99-1219, the U.S. Army Research Laboratory under contract W911NF-04-2-0013, by the U.S.
TABLE OF CONTENTS

ACKNOWLEDGEMENT ............................................................................................................................II

LIST OF FIGURES ......................................................................................................................................VII

CHAPTER 1. INTRODUCTION ..................................................................................................................1

1.1 Tactical Networks: The FCS Environment .........................................................................................2
1.2 Resource Allocation in Tactical Networks .........................................................................................6
1.3 Research Outline ...............................................................................................................................9

CHAPTER 2. BACKGROUND AND PREVIOUS WORK ........................................................................11

2.1 Local Data Processing .....................................................................................................................11
2.2 Remote Data Processing ..................................................................................................................13
  2.2.1 Publish/Subscribe Frameworks ................................................................................................14
  2.2.2 Agent Negotiation or Arbitrage Models ....................................................................................16
2.3 Distributed (In-Stream) Data Processing .........................................................................................17

CHAPTER 3. MODEL DESCRIPTION ....................................................................................................22

3.1 The Environment and Task Description ..........................................................................................22
  3.1.1 The Network Diagram ............................................................................................................23
  3.1.2 Data-State Transition Diagram ...............................................................................................25
  3.1.3 Data Task Description ............................................................................................................28
  3.1.4 The state of a node ..................................................................................................................32
  3.1.5 Actions ....................................................................................................................................33
3.2 Processing Data Tasks .....................................................................................................................35
3.3 Model Summary ...............................................................................................................................39

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>PROBLEM DESCRIPTION</td>
<td>40</td>
</tr>
<tr>
<td>4.1</td>
<td>ACTION COST ESTIMATION</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>THE RESOURCE ALLOCATION PROBLEM</td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td>LEARNING STRATEGIES FOR THE RESOURCE ALLOCATION PROBLEM</td>
<td>48</td>
</tr>
<tr>
<td>5.1</td>
<td>A MODIFIED ACTION COST FUNCTION</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>LEARNING POLICIES</td>
<td>52</td>
</tr>
<tr>
<td>5.2.1</td>
<td>The k-Arm Bandit Problem</td>
<td>53</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Learning Resource Allocation Strategies in Tactical Networks</td>
<td>59</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Convergence Properties</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>THE RESOURCE ALLOCATION ALGORITHM</td>
<td>65</td>
</tr>
<tr>
<td>6.1</td>
<td>DATA MAINTAINED AT EACH NODE</td>
<td>66</td>
</tr>
<tr>
<td>6.2</td>
<td>MESSAGE TYPES</td>
<td>66</td>
</tr>
<tr>
<td>6.2.1</td>
<td>The Data Message</td>
<td>67</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Broadcast Cost Messages</td>
<td>68</td>
</tr>
<tr>
<td>6.3</td>
<td>BUILDING COST ESTIMATES FOR THE NEIGHBORS</td>
<td>69</td>
</tr>
<tr>
<td>6.4</td>
<td>ACTION SELECTION USING SOFTMAX AND $\epsilon$-GREEDY STRATEGIES</td>
<td>71</td>
</tr>
<tr>
<td>6.5</td>
<td>PSEUDO-ALGORITHM</td>
<td>76</td>
</tr>
<tr>
<td>7</td>
<td>EXPERIMENTAL EVALUATION</td>
<td>79</td>
</tr>
<tr>
<td>7.1</td>
<td>THE SIMULATION ENVIRONMENT</td>
<td>79</td>
</tr>
<tr>
<td>7.2</td>
<td>SIMULATING DATA TASKS</td>
<td>83</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Simulation Parameters</td>
<td>84</td>
</tr>
<tr>
<td>7.3</td>
<td>EXPERIMENTAL RESULTS</td>
<td>86</td>
</tr>
<tr>
<td>7.3.1</td>
<td>The 4-Node Fixed Topology</td>
<td>86</td>
</tr>
<tr>
<td>7.3.2</td>
<td>The 4-Node Changing Topology</td>
<td>90</td>
</tr>
<tr>
<td>7.3.3</td>
<td>The 7-Node Changing Topology</td>
<td>92</td>
</tr>
<tr>
<td>7.3.4</td>
<td>An Example of Unstable Equilibrium</td>
<td>95</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1. The Army's Future Combat Systems ................................................................. 3

Figure 2. A Tactical mobile ad hoc network (source: ARL) .............................................. 4

Figure 3. A Mobile ad hoc network example .................................................................. 7

Figure 4. Resource allocation for local data processing .................................................. 12

Figure 5. Resource allocation for remote data processing ............................................. 14

Figure 6. Resource allocation for in-stream data processing ......................................... 18

Figure 7. Data description in directed diffusion (Intanagonwiwat et al., 2000) ............... 19

Figure 8. A NET diagram showing the pfactor and tfactor values .................................. 24

Figure 9. Data State-Transition (DST) diagram for the 'document' data type ............... 26

Figure 10. DST for 'document' showing function costs and datum sizes ....................... 27

Figure 11. An example of a valid action ........................................................................... 35

Figure 12. The state transition cycle of a node handling a data task. ............................. 35

Figure 13. An example of data task processing process .................................................. 36

Figure 14. Data task processing – second step ............................................................... 37

Figure 15. Data task processing – third step .................................................................... 38

Figure 16. A sample data task for cost estimation .......................................................... 43

Figure 17. The minimum spanning tree for data processing cost estimation .................. 43

Figure 18. A global cost estimate example ...................................................................... 45

Figure 19. Local view of the environment from ni's perspective ..................................... 45
Figure 20. Hedge vs. Exp3 probability distributions (τ = 1) ........................................ 57
Figure 21. Hedge vs. Exp3 probability distributions (τ = 2) ........................................ 58
Figure 22. The description of a data message ................................................................. 67
Figure 23. Propagating data task costs with data messages .......................................... 70
Figure 24. A list of possible actions ............................................................................. 73
Figure 25. A data task cost table .................................................................................. 74
Figure 26. Simulation environment .............................................................................. 81
Figure 27. Four nodes network for test and DST .......................................................... 87
Figure 28. Four node fixed topology Boltzmann vs. ε-greedy ...................................... 88
Figure 29. Boltzmann vs. ε-greedy with variable and fixed ε ........................................ 89
Figure 30 Final data flow configuration for the 4-Node fixed problem .......................... 90
Figure 31. Four-node network with changing topology ............................................... 91
Figure 32. 7-node network with changing topology ...................................................... 92
Figure 33. A 7-node network changing topology ......................................................... 93
Figure 34. Removing node n3 from the 7-node topology ................................................ 94
Figure 35. Re-inserting node n3 from the 7-node topology ............................................. 95
Figure 36. An unstable condition with the Boltzmann strategy .................................... 96
Figure 37. Unstable probability distribution using the Boltzmann strategy..................... 98
Figure 38. The unstable condition corrected with the sliding window ........................... 99
Figure 39. The hidden node problem .......................................................................... 105
Figure 40. 802.11 networks in ad hoc and infrastructure mode .................................... 107
Figure 41. Promiscuous listening mode ...................................................................... 111
Chapter 1. Introduction

Tactical networks are generally characterized as mobile ad hoc networks under policy and resource constraints. These types of network environments are commonly found in military and disaster recovery operations.

The coordinated allocation and utilization of resources (both for data processing and communication) is an important and challenging research issue for these types of environments.

The decentralized and dynamic natures of tactical environments require resource coordination strategies that are distributed, efficient and adaptive. Furthermore, coordination mechanisms are expected to be resilient to environmental changes and failures, with minimum requirements for maintenance and configuration.

Possibly one of the most descriptive examples of a tactical network is the battlefield environment envisioned in the Army’s Future Combat Systems (FCS) program, a multi-billion dollar effort to restructure the US Army into a data-centric agile and flexible battle force that will heavily rely on information, rather than massive fire power to improve lethality and significantly reduce costs for deployment and operations.

Although the overall concept of the program is significantly broader than the tactical network infrastructure, for the purpose of this work, the focus will be the tactical network environment required for the FCS vision.
1.1 Tactical Networks: The FCS Environment

Dependable communication capabilities are amongst the most important technical requirements for tactical environments such as those found in military combat operations. Highly volatile and ad hoc in nature, these types of environments bring new challenges and requirements to the traditional practices and research in communications and data networks.

Complex military missions involving coalition forces, robotic support units, remote sensor beds, and autonomous vehicles require underlying communication infrastructures that are flexible, efficient, and robust in order successfully operate in combat.

The capacity to efficiently generate, process and share information horizontally between peer nodes in the battlefield is paramount in tactical environments. It realizes the notion of information-centric warfare that has been highly promoted by FCS.

The Army’s FCS program envisions a system of systems, connecting a number of lightweight operation units through a tactical communications infrastructure. The goal is to empower the combat forces with information and agile equipment, as oppose to massive combat tanks that are slow, and expensive to transport and relocate.

The reduction in heavy armor and in lieu of agility and flexibility will be compensated by superior intelligence and information awareness directly available to the soldier and vehicles in the field.
In this new environment (Figure 1), the communications network is the central linking point for all units, becoming one of the most important and critical elements in the system.

The communications infrastructure must be flexible enough to support high capacity data links between operational units as well as highly dynamic ad hoc environments at the edge of the environment.

The focus in this work is on the mobile ad hoc networks deployed at the edge of the operations theater (Figure 2). These critical networks are potentially highly dynamic,

---

1 Image Source: Army Research Laboratory and http://www.globalsecurity.org/military/systems/ground/fcs.htm)
with no centralized infrastructure or coordination components, and yet highly constrained in terms of mission requirements, resource availability and both security and administrative policies.

Furthermore, in these types of environments, nodes are highly heterogeneous in almost all their computational aspects, ranging from combat vehicles with large communications and processing resources to small unmanned vehicles that are battery operated, with very limited resource capabilities.

In general, the main characteristics generally accepted for the communications infrastructure in tactical environments are the following:

- Mobile Ad hoc Network: Mobile Ad hoc networks (MANET) are characterized by dynamic (or mobile) hosts connecting with each other in ad hoc fashion (with no support of any fixed infrastructure) to build local area networks. Tactical networks might leverage, but not depend, on any fixed network infrastructure to operate efficiently.
- Resource Efficient: Most tactical networks rely on several hosts that are battery operated or highly constrained in resource capabilities. Efficient and balanced resource utilization is fundamental.

- Adaptive: Tactical networks must adapt to changes in operation goals or operation tempo. For instance, while monitoring the environment, the network should minimize resource utilization to maximize its lifetime however, during combat the network should prioritize performance, in lieu of resource usage.

- Resilient: The communications infrastructure must survive to arbitrary node or link losses, degrading gracefully as resources expire. Self reorganization and self-healing are important features for tactical networks.

- Data Centric: The primary goal of tactical networks is to share information and control messages. The focus is on the data, as opposed to the network.

- Secure: Security is paramount but it will not be addressed in this work. It is assumed that an underlying security protocol is in place at the link level, preventing unauthorized (or compromised) nodes from joining the network.

- Policy Constrains: Tactical networks must support and enforce policy constraints for both resource utilization and information release between nodes and applications. This is a fundamental requirement for supporting coalition operations in both military and civilian scenarios.

The primary type of traffic in tactical networks is sensor and control data. These two types of traffic have very different characteristics and requirements. Sensor data is usually unreliable, tolerating some levels of packet loss in lieu of low levels of latency.
and jitter (variance of latency). Other specialized types of traffic, such as control commands, depend on reliable communications with packet retransmission and strict sequencing, usually requiring the use of specialized communications libraries for tactical networks (Suri et. al, 2005).

Data traffic, although more tolerant in general is usually much more demanding on the network resources than control data. First, because there’s usually a much higher volume (in these types of environments) of sensor data than any other type of data, and second because while control traffic is point to point and intermittent (referring here to the control of an unmanned vehicle, for instance), data traffic is usually continuous and concurrently distributed often from one sensor to multiple clients, with different requirements, policies and resource constraints (Carvalho, et. al. 2005).

The challenges involved in these types of environments are very similar to the challenges usually found in disaster relief operations. For the purpose of this research, the most important requirement is the efficient use of network resources. Although, most of the other requirements will also be addressed at some level in the proposed framework, the focus of the subsequent discussions will be primarily on resource allocation for distributed data processing and transmission.

1.2 Resource Allocation in Tactical Networks

Let’s consider, for example, the tactical network environment illustrated in Figure 3. In this example, node ‘UAV’ (circled) is an unmanned aerial vehicle capturing a continuous sequence of images of enemy positions nearby. A number of other friendly
nodes are shown in the figure, connected by lines representing their communications capabilities.

Let’s also assume that nodes A, B and C require periodic data updates (images) from sensor ‘UAV’. However, due to local hardware capabilities and clearance constraints, node ‘A’ must receive only low resolution, filtered images, while nodes ‘B’ and ‘C’ should receive the high resolution versions, for targeting.

In this example, most conventional data networks would task the sensor (UAV) with two requests, one for a high resolution image and one for a low resolution, filtered image. The sensor would process the data accordingly and send both updates through the network.

Assuming that data processing is done locally at the UAV, the distribution of the data itself is done through standard multicast (or unicast) routing protocols. In the recent years, several multicast protocols have been developed (or adapted) to operate in mobile ad hoc networks, and efficiently distribute data from one sensor to multiple clients.

For tactical networks, however, this solution might not be appropriate either because the ‘UAV’ has no spare processing capabilities to convert and filter the images, or because network links are unable to support both transmissions concurrently.
Taking advantage of the multi-hop path from the source node to the multiple destinations, the ideal solution would be to distribute only the high resolution unfiltered images from the sensor to nodes ‘C’ and ‘T’ for instance, and have node ‘T’ forward the images to node ‘B’ and also filter and reduce their resolution before forwarding them to node ‘A’.

The data path illustrated in Figure 3 describes a Data Distribution Tree (DDT). An optimum DDT describes the optimum path for data processing and distribution for a given global optimization criteria such as cost or latency minimization. As the network topology, or data requests change in time, the data distribution trees must adapt to comply with the optimization criteria under the new conditions. The DDT solution is dynamic, reacting to changes in the tactical environment.

It is important to note that the data distribution tree will not necessarily provide either the lowest routing path, or the lowest aggregate processing cost, but it will provide the lowest combined costs for data processing and distribution.

The challenge in finding such a solution is in the fact that both costs are interdependent. That is, depending on how the data is processed through the path, communications costs will change, which will in turn affect the criteria used for selecting the initial processing nodes in the first place.

The strict layering and separation of concerns in traditional communication networks have played a very important role in providing the necessary compatibility to enable emergence of large scale networks such as the Internet. The same characteristics, however, constitute a problem for most tactical networks, because the routing of the data
is usually completely data-agnostic and isolated from higher level components that are capable to process and transform the data.

Partly due to these types of legacy constraints, most resource allocation techniques for in-stream data processing currently available tend to treat the problems in isolation and, in the best cases, use metrics provided by the underlying routing protocol (such as path cost and estimated latency) to estimate the costs for allocating data processing resources. The interplay between these two components however makes it very hard to achieve a global minimum for the resource allocation problem in most practical settings.

Conceptually, the problem of dynamic resource allocation in tactical networks consists in finding and maintaining the best data distribution trees that together minimize the global utilization of resources for data processing and data transmission, while complying with policy constraints both at the levels of the nodes an network.

1.3 Research Outline

In this work, a novel distributed solution for the resource allocation problem for tactical networks is proposed. By simultaneously addressing both the issues of data processing and data transmission in the network, the proposed solution quickly finds a data distribution tree in the network and learns, as traffic flows about alternative better solutions, locally adjusting the DDT as necessary.

The proposed resource allocation strategy combines a specialized communication protocol (designed as part of this work), with well establish concepts in reinforcement learning to build a distributed solution for the problem. As part of this research, a
simulation environment is designed and implemented to provide a controlled setting for testing and validating the proposed strategy.

Chapter 2 provides a brief background on mobile ad hoc networks, followed by a review of the previous resource allocation work in Mobile Ad hoc Networks. Chapter 3 introduces the formal description of the problem and the notation of a data task that will be the center of the discussion through the remaining of the work.

In Chapter 4, the resource allocation problem is discussed in the context of the classical online learning k-arm bandit problem. Also in this chapter, two different strategies for action selection in the k-arm bandit problem are presented and discussed in the context of resource allocation problem.

The proposed algorithm is introduced in Chapter 5, followed by Chapter 6 where experimental results of simulated tests are presented and discussed. The work is concluded in Chapter 7, with a few suggestions for improvements and future research issues.
Chapter 2. Background and Previous Work

The resource allocation problem for data processing in Mobile Ad hoc Networks can be generally classified in three main groups: a) Local data processing, b) Remote data processing and c) Distributed (or in-stream) data processing. In each case, the goal is to allocate resources for data processing and transformation from a source node to multiple sink nodes requiring (possibly) different variations of the data.

2.1 Local Data Processing

In the first problem type, local data processing, the source of the data is responsible for providing the necessary transformations required by each client. Similar to conventional client-server models, local data processing essentially allocates all processing to the data source (i.e. the server). This case is illustrated in Figure 4, where the source node (UAV) generates a datum D0 that must be converted into D3 before being delivered to the client node 4 (soldier). Datum D0 can be, for instance, a raw image from a video camera, while D3 can be a filtered version of the image.

2 A brief review of some of the technical aspects of mobile ad hoc networks is provided in Appendix A.
The research focus on these types of problems is basically in the allocation of resources for data distribution (i.e. data routing).

More specifically, Quality of Service (QoS) protocols for mobile ad hoc networks have been the focus of attention for a number of research efforts in the past few years. The goal of QoS is to provide ‘guarantees’ on level of service and cost estimates for different types of data flows at different nodes. The information provided by QoS (and equivalent) protocols can be used by the source node to decline, accept or arbitrarily modify data requests by the client.

Curran (2003) proposed a reinforcement learning-based algorithm for routing in ad hoc networks. The SWARM protocol is data agnostic, focused only on packet routing. When receiving a data packet, each node chooses the appropriate action (next hop) based on current policies. The state transition probabilities are defined based on the delivery success and failure ratios for each link. Packet transmissions are acknowledged immediately, with a fixed low cost (negative reward) for success and high costs for failures. The work was later extended by Dowling et al. (2004) who proposed the
collaborative reinforcement learning-based routing protocol called SAMPLE, for mobile ad hoc networks.

Chang (2004) has also proposed the use of reinforcement learning techniques for data routing in mobile ad hoc networks. Although the approach did not address tactical issues such as service decomposition and distribution, it did allow for interaction between data routing and node mobility. In his work, Chang proposes an environment where node mobility can be influenced by routing decisions. In fact, the location of nodes can be directed by the algorithm in order to optimize data routing. His results once again come to show that reinforcement learning is likely to be applicable to highly dynamic environments.

Peng and Deyun (2006) also leverage from reinforcement learning algorithms to improve QoS routing strategies. In his work, Peng proposes a heuristic-based algorithm that utilizes reinforcement learning to estimate best QoS routing paths from previous experience, reducing the number of QoS flood and probing packets for path maintenance in mobile networks.

2.2 Remote Data Processing

In the second problem type, remote data processing, the goal is to determine a node in the network that is a better candidate for the data processing task. That is, to have node UAV delegate the filtering of the image (D0) to be done by another node in the network (or possibly the sink itself). This example is illustrated in Figure 5 where the data processing task (i.e. converting D0 into D3) is done by node 3.
Like in the previous types of problems, the task of allocating resources for data processing is separate from the allocation of resources for data transmission in the network.

2.2.1 Publish/Subscribe Frameworks

Publish-subscribe data frameworks are very common for data distribution on mobile ad hoc networks. In general publish subscribe models rely on building multicast data trees from a source to multiple clients. Data processing is usually not part of these algorithms and resource allocation essentially consists of finding the best spanning tree for data distribution.

In 2004, Baheni proposed a data aware variation of conventional multicast protocols that took into account the nature of the data being transmitted when building the multicast trees. The protocol, called ‘Da-Multicast’, utilized data subscription information and pre-defined relationships between topics to build dynamic groups for data distributions.
Other topic-based publish/subscribe systems such as such as TPS (Eugster et al., 2001) and JORAM (Maistre, 2003) also leverage from multicast protocols and the assumption of a clear hierarchy on data and events to build efficient multicast groups for topic-based data distribution. Multicast based protocols often provide an efficient solution to the problem but they assume that only nodes participating in the multicast group would share the data for distribution (at the level of the multicast tree). Furthermore, most multicast protocols have no notion of task decomposition and distribution.

Other examples of distributed publish/subscribe frameworks are overlay networks previously proposed by Katz and Brewer (1994), Chen and Schawn(2005), and others. Overlay networks seek to abstract lower layer routing with an overlying data aware framework, or middleware that relies on distributed (or centralized) algorithms to coordinate resource utilization. Overlay networks tend to be effective for small network domains, but depending on the level of routing granularity controlled by the overlay network, it can induce significant overhead and scalability problems.

Similar to overlay networks, specialized communications middleware also provide application level control (or influence) of routing decisions, but only for special types of traffic data. An example of a communications middleware successfully applied in small scale tactical environments is FlexFeed (Carvalho and Breedy, 2002). FlexFeed uses a centralized coordination algorithm to allocate data processing and data communication tasks in the network. The framework is effective for small networks but it fails to scale with the number of nodes, data tasks or node mobility (Carvalho, Suri and Arguedas, 2005).
Still relying on global information, but not in a central coordination point, alternative resource allocation approaches have also been provided. Sorensen et. al. (2004) proposed a reconfigurable middleware called CORTEX for context aware data distribution in mobile networks. CORTEX doesn’t support service decomposition and distribution but it does take communications cost as a factor for service allocation.

2.2.2 Agent Negotiation or Arbitrage Models

From an agent negotiation, or economical perspective, there were also several attempts to address the issue. Carvalho, Pechoucek and Suri (2005) proposed a hybrid approach where partial state information about resource availability could be used in conjunction with agent negotiation for a decentralized approach to the problems. Based on previous agent negotiation research, that work proposed the use of Acquaintance Models and Remote Presence (Pechoucek, Marík and Stepankova, 2001) for resource allocation. Arbitrage models were also popular with the grid computing community, where Kothari, Sabhash and Zhou (2003) proposed a profit maximization algorithm for resource allocation in a fixed grid network, extending a similar approach proposed by Buyya et. al. (2001).

Reinforcement learning techniques are not new in data networks. They have been previously proposed for several data routing algorithms (Littman, 1993; Boyan, 1994; Choi and Yeung, 1996; Miikkulainen and Kumar, 1999; Stone, 2000 and Tao, Baxter and Weaver, 2001), and service scheduling issues, both for fixed and ad hoc networks.
Extensions of these protocols have also been proposed for multicast routing algorithms. Garcia et al. (2003) and Galstyan et al. (2004) proposed the use of reinforcement learning techniques for resource allocation in grid applications.

Although bandwidth costs were also taken into account in Galstyan's research, the work was focused on service scheduling in grid networks, disregarding some of the important features for tactical environments such as service decomposition and distribution. Nevertheless, Galstyan work provides another good indication of the applicability of reinforcement learning techniques for resource allocation.

2.3 Distributed (In-Stream) Data Processing

A third and perhaps more interesting type of data processing in mobile ad hoc network is what we refer to as Distributed Data Processing. In these types of data problems, a task is actually fragmented into sub-tasks that can distributed through nodes in the network in order of minimize, for instance, overall data costs.

The way in which tasks are fragmented is a function of the capabilities of network nodes, topology and policies. In the special case of data streams, a sequence of equal-type data packets must be transformed in nodes that participate in the data flow. We refer to distributed data processing of data streams as 'in-stream' data processing, because nodes participating in the actual multi-hop streaming of the data will also be responsible for the partial transformation attributed to them.

An example showing a single source and a single client is presented in Figure 6, where the data conversion between datum D0 available at the UAV (node 1) and datum
D3 requested by soldier is accomplished in two steps at two different nodes (D0 to D2 at node N2) and (D2 to D3 at node N3).

![Diagram](https://via.placeholder.com/150)

**Figure 6. Resource allocation for in-stream data processing**

The research on in-stream data processing (as defined here) for mobile ad hoc networks is relatively new.

A number of data centric routing protocols such as SPIN-IT (Woodrow and Heinzelman, 2001) and Directed Diffusion (Intanagonwiwat, Govindan and Estrin, 2001) were also proposed as extensions for conventional MANET routing algorithms, primarily for data streaming.

In directed diffusion, the data generated at the sensor is named by the sensor node by a set of attribute-value pairs. When a client decides to place a data request, it essentially specifies the set of attributes that describes that data it needs, and sends this information (called interest) to all the nodes it can reach through a broadcast.

An example of a data description contained in an interest packet is shown in Figure 7. In this example the "rect" entry specifies a geographical area where the data...
requested is to be observed. Only sensors capable of monitoring that specific area will be relevant to this request.

<table>
<thead>
<tr>
<th>Type</th>
<th>four-legged animal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>1s</td>
</tr>
<tr>
<td>Rect</td>
<td>[-100, 200, 200, 400]</td>
</tr>
<tr>
<td>Timestamp</td>
<td>01:20:40</td>
</tr>
<tr>
<td>ExpiresAt</td>
<td>01:30:40</td>
</tr>
</tbody>
</table>

Figure 7. Data description in directed diffusion (Intanagonwiwat et al., 2000)

Directed Diffusion is based on local interaction between nodes, meaning that when a node (A) broadcasts an interest for some type of data, the information is relayed by A’s neighbors and propagated through the network. At each step the node receiving the data interest will perceive the request as if it were coming from its immediate neighbor, and not from the request originator (A). This characteristic of the protocol will facilitate the expansion of data distribution trees, but it creates a problem for the proposed scenario. For the FCS scenario it is fundamental to ensure that appropriate information release policies are enforced, which is a very complex task if the actual information about the final destination of the data is not available to the source.

Nevertheless, the interest packet propagates from the client (via broadcast) throughout the network. The network nodes maintain a cache of all interests received within a certain period of time. Based on timestamp and expiration information they can determine if a new interest is a duplicate or not. Using the local cache, every node can decide to re-send the interest received to a sub-set of neighbors or not. An interest that has been recently sent (which is determined by their timestamps) will optionally not be resent, to avoid unnecessary overhead.
The subset of nodes to which the interest packet will be resent is essentially an implementation choice. A broadcast model can be used where every node simply re-broadcasts the interests to all its neighbors, or just to a sub-set of them, to reduce traffic. Often geographic position information is also used to reduce the scope (quadrant) of the propagation of data interests.

The interests will propagate to all nodes in the network that either generate the data (sensor nodes) or have access to data specific to the interest (intermediate nodes on already established paths). In any of these cases, the node can act as a source for the request and start providing the feed.

Data packets received by the client contain path information so the client (originator of the interest) can then decide which path is better and start sending route reinforcement packets back through that path. Positive reinforcement will increase the data transmission rate to the desired frame rate and negative reinforcement will reduce it to a beacon-like state, not to receive data but to maintain the alternative route alive.

In most data centric protocols, requests are defined as ‘data needs’, and propagated through the network. Nodes that have (or can generate) the requested data initiate the transmission to target node, who decides, based on aggregate cost of the received data, on preferred configurations, directly notifying each redundant provider to stop transmitting.

Data centric protocols are very well fit for MANET environments. Such protocols are likely to yield a hierarchical data distribution tree from a source to multiple clients but, in general, they fail to provide in-stream data processing solutions by distributing processing costs. This is because, each node that is aware of the request attempts to fully
produce the target data, not considering the possibility of delegating processing to downstream nodes. Furthermore, nodes keep no memory of their experience so future requests of similar data, even over similar topologies, will result in re-discovering the data distribution trees, with unnecessary overhead and latency for finding optimal solutions.
Chapter 3. Model Description

For the purpose of this work, the formal description of the resource allocation problem is based on three concepts that, together, characterize the network environment and a task to be performed. These concepts are the Network, the Data-state transition (i.e. data transformation) and the Data Task.

Although the discussions throughout this chapter refer to a single task assigned to a node, they imply the notion of a sequence of similar tasks constituting a data stream. In this context, a data stream from a source node to a set of target nodes is essentially a sequence of equal data tasks assigned to the source node to be individually handled. In this chapter we formally define the network environment and the notions of node state, tasks and actions.

3.1 The Environment and Task Description

The physical environment consists of a mobile ad hoc network represented by a network diagram. The network describes the current physical constraints of the system, both in terms of nodes (hosts) and communication (links) capabilities. The tasks, referred to as ‘data tasks’ define both processing (data transformation) and transmission jobs to be accomplished by the network.

In the context of this work, a task is always specified from a source node to a set of clients. The concepts presented here, however, can be generalized to include fusion
tasks, i.e. tasks where multiple source nodes provide data (to be fused or aggregated) to a set of clients.

3.1.1 The Network Diagram

**Definition 1** A Network Diagram (NET) is defined as a graph structure \( \text{NET}(N, L) \) where \( N = \{n_1, n_2, \ldots, n_r\} \) is a set of vertices representing nodes (hosts) in the network and \( L = \{l_1, l_2, \ldots, l_s\} \) is a set of edges representing the communication links between nodes. Data transmission between nodes can only occur through a link \( l_i \) in the graph, and all links are assumed to be bi-directional and symmetric.

Associated with each node there’s a list of data processing functions that can be performed at that node. Each node also defines a hardware-specific processing cost factor denoted as \( pfactor \). Both the list of available functions and the \( pfactor \) can vary in time, based on node’s resource availability or policies.

Associated with each edge there’s a link-specific transmission cost factor denoted as \( tfactor \) that can also vary in time. Communication edges are assumed to be symmetric so the same \( tfactor \) is valid for communications in both directions.

At any given time, a node is aware of its current \( pfactor \) and the \( tfactor \) for the connected links. Both the \( pfactor \) and the \( tfactor \) are used to estimate the costs for processing and transmitting a data packet respectively.

The \( tfactor \) is a weight that can be used to describe the load on the communications channel representing, for instance, bandwidth availability, congestion or link quality. Similarly the \( pfactor \) associated with each node may be used as an indicator of processing load (or availability), policies or simply as a prioritization factor for data processing amongst nodes. For the purposes of this work, all these aspects of resource...
availability for both nodes and links are summarized as a single cost factor, namely the \( tfactor \) and \( pfactor \).

These cost multipliers are not necessarily fixed. Changes in these factors may be arbitrary and follow no specific pattern. Although some authors (Chang and Kaelbling, 2004; Basagni et al., 1998, Pei et al., 1999) have previously proposed routing algorithms that leverage from mobility models and utilization patterns to predict future network states, the approach adopted in this research avoids modeling changes in the network, relying solely on the current state of the network to estimate the minimum cost for tasks. The assumption is that changes in the state of the environment are significantly slower than the allocation of resources. Note that this does not imply that any solution for a static environment would suffice. The dynamic nature of the tactical environment requires solutions that are locally adaptive to changes, although not necessarily based on predictions of future network states.

![Figure 8. A NET diagram showing the pfactor and tfactor values.](Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.)
Figure 8 illustrates an example of a network graph showing all functions available at each node as well as all processing (\( pfactor \)) and transmission (\( tfactor \)) values on nodes and edges respectively. The figure shows a snapshot of the costs and topology at an instant (\( t_i \)). Because changes on cost factors and topology can be arbitrary, the NET diagram (Figure 8) is assumed to be 'constant' only for an interval \( \Delta t \) where changes fall within predefined thresholds for tolerance.

3.1.2 Data-State Transition Diagram

Before defining a data task, the notion of a data transition diagram must be introduced. A data transition diagram essentially specifies (through a graph structure) all possible state transitions supported by a data type.

An instance of a data type (i.e. a datum) can be duplicated, processed or transferred between nodes. Examples of data types are video images (snapshots), documents and sensor data-captures in general.

When processed, a datum retains its type but it might change states. For instance, a 640x480 24bit JPEG image can be converted to an equivalent 320x240 24bit PNG image, but it would maintain its type as an ‘image’. Each possible configuration represents a new state for this specific image data type.

Each data type in the framework has a finite, well defined number of possible states and state-transition paths. A data type is fully described by a data state transition graph (\( DST \)), defined as follows:

**Definition 2** A Data State-Transition Diagram (\( DST \)) is a directed graph \( DST(D,F) \), where \( D = \{d_1,d_2,...,d_n\} \) is a set of vertices representing all possible states that a datum
can assume and \( F = \{ f_1, f_2, \ldots, f_n \} \) is a set of edges representing all functions \( f_i \) that can be applied on the data.

The structure of the \( DST(D, F) \) constrains the functions that can be applied to each state of the datum, and specifies the data transitions resulting from the application of each function \( f_i \in F \).

Consider, for instance, a data type called ('d'), for 'document'. Let's also establish that an instance of the 'document' data type can be in one of four states:

a) MS Word document (d1),

b) PDF document (d2),

c) Open-Office document (d3), or

d) Postscript document (d4).

The states for datum 'd' and its transition functions are illustrated in the DST \( DST(D, F) \) diagram shown in Figure 9. In this example, \( D = \{ d_1, d_2, d_3, d_4 \} \) is the set of possible states for datum 'd' and \( F = \{ \alpha, \beta, \gamma, \varphi \} \) represents the set of functions for data-state conversion.

Other data types (e.g. images, or GPS data) would have their own DST diagrams describing all their possible states and state-transition functions.

Figure 9. Data State-Transition (DST) diagram for the ‘document’ data type

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
As illustrated in Figure 9, an MS Word document \( (d_1) \) can be converted into a PDF document \( (d_2) \) through the function \( \alpha \), (i.e. \( d_2 = \alpha(d_1) \)). An Open-Office document \( (d_3) \) can be converted into a PDF document \( (d_2) \) through the function \( \gamma \), but it cannot be converted into an MS Word document \( (d_1) \), as there are no functions defined for that transition in the diagram.

Associated with each possible data state there is a size value (represented as an integer). The size of a datum can be directly mapped, for instance, to the actual number of bytes occupied by the datum, and it is intended to weight the costs for transmitting the datum through a network link. Larger data sizes will result in higher transmission costs.

Also in the DST, associated with each function (edge), there’s a cost factor that will weight the actual costs of converting a datum from one state to another. A DST showing both the datum sizes and the function costs is shown in Figure 10.

![Figure 10. DST for 'document' showing function costs and datum sizes.](image)

The data information provided by the DST is static. It depends only on the data-type and it doesn’t change in time, or location in the network. Given an instance of a datum, every host in the network knows (or can obtain) the DST for the datum’s data.
type. The proposed framework simultaneously supports multiple data types and data transactions competing for shared network resources.

There's a mapping between the functions described in the DST diagram for a data type (Figure 10) and the functions available at an node as defined in the NET diagram (Figure 8). Going back to the previous 'document' example, based on the information shown in Figure 8, node \( n_4 \) is only capable of converting an MS Word document into PDF (\( \alpha \)), while node \( n_2 \), for instance, is capable of converting an MS Word document into PDF (\( \alpha \)), and also into an Open Office document (\( \varphi \)).

### 3.1.3 Data Task Description

A data task is defined as a combined data processing and data transmission job assigned to a node in the network. It is constrained (in the context of this work) to a single source and potentially multiple destinations. A network node is capable of handling one data task at a time, but multiple data tasks can be simultaneously processed at multiple nodes in the network.

A data stream is a sequence of data packets that are individually processed. In this discussion a data task is defined as a task for handling a single data packet. A data stream implies a sequence of data tasks. To simplify the formal definition of a data task, the concepts of a source pair and a target pair will be first introduced.

**Definition 3:** For a given NET\((N,L)\) and DST\((D,F)\), let's define a **Source Pair** \(sp(\text{NET, DST})\) and a **Target Pair** \(tp(\text{NET, DST})\) as tuples \((n, d)\) where \(n \in N\) and \(d \in D\). A **Target Pair Set**, denoted by \(TP = \{tp_1, tp_2, ..., tp_h\}\), is an unordered set of target pairs.
To simplify notation, for discussions where a single data type is considered over a fixed (or temporarily fixed) network, the Source Pair \( \text{sp}(\text{NET}, \text{DST}) = (n_i, d_j) \) and the Target Pair \( \text{tp}(\text{NET}, \text{DST}) = (n_k, d_p) \) will be simply denoted as \( \text{sp} \) and \( \text{tp} \) respectively, under the implied assumption that \( n_i, n_k \in N \) and \( d_j, d_p \in D \).

Leveraging from the previous definitions, a data task can be defined as follows:

**Definition 4**: For a given NET\((N,L)\) and DST\((D,F)\), a Data Task is defined as a tuple \( \text{dT}(\text{NET}, \text{DST}) = (\text{sp}, \text{TP}, \text{phist}) \) where \( \text{sp} \) is a source pair \( \text{sp}(\text{NET}, \text{DST}) = (n_i, d_j) \), \( \text{TP} \) a target pair-set in \( \text{NET}, \text{DST} \), and \( \text{phist} \) an unordered set of nodes \( \{n_i\}, n_i \in N \).

In a data task, the source pair identifies which node has which datum, and the set of target pairs defines the target nodes and the datum they should receive. A data task essentially specifies a source pair (node/datum) and a set of destination pairs (node/datum) as part of a task to be accomplished. To simplify notation, a data task \( \text{dT}(\text{NET}, \text{DST}) = (\text{sp}, \text{TP}, \text{phist}) \) is simply denoted as \( \text{dT} \) in contexts where a single data type \( \text{DST} \) and \( \text{NET} \) are considered. Furthermore, in order to facilitate the notions introduced in subsequent discussion, let’s also define two auxiliary functions \( f_n \) and \( f_d \) as follows:

**Definition 5** Let’s define a Node Function, denoted by \( f_n(\gamma) \) for \( (\gamma = \text{sp}) \) or \( (\gamma = \text{tp}) \), as a function \( f_n\left(\langle n_i, d_j \rangle\right) = n_i \) that returns the node \( n_i \) of the argument \( \gamma \).

**Definition 6** Let’s define a Datum Function, denoted by \( f_d(\gamma) \) for \( (\gamma = \text{sp}) \) or \( (\gamma = \text{tp}) \), as a function \( f_d\left(\langle n_i, d_j \rangle\right) = d_j \) that returns the datum \( d_j \) of the argument \( \gamma \).
Both the node and the datum functions can be applied to a data task 
\( (dT = (sp, TP, phist)) \), which is equivalent to applying the functions in the source pair 
\((sp)\) of the data task. That is \( f_n(dT = (sp, TP, phist)) = f_n(sp) \) and similarly \( f_d(dT = (sp, TP, phist)) = f_d(sp) \).

A data task fully specifies the source node \( sp = \{n_i, d_j\} \) and the task to be performed. It is, by definition, assigned to the node \( n_i = f_n(sp) \) specified in the source tuple \( sp = \{n_i, d_j\} \) and can have an arbitrary finite size for the target pair set and the path history.

The path history \((phist)\) rather than specifying part of the task, defines constraints on how the task can be handled or delegated. The \(phist\) essentially defines a blacklist for the tasks, that is, a list of nodes in the network that must not be relied upon for assistance with this specific task. A valid data task must satisfy two conditions:

a) All data \(d_j\) specified in the source pair \((sp = \{n_i, d_j\})\) or target pairs \(tp = \{n_k, d_p\} \subseteq TP\) must be of the same data type. That is, all datum instances must be part of the same DST diagram. Furthermore, there must be at least one data transition path (in the DST graph) from the source datum \(d_i = f_d(sp)\) and each of the data \(d_j = f_d(tp) | tp \in TP\).

b) None of the nodes specified in the target pairs must be part of the \(phist\) set, unless the task is defined as a terminal task. A terminal task is defined as follows:
Definition 7: A terminal task is a data task \( dT = (sp, TP, phist) \) where the target pair set \( TP \) is a singleton \( \{tp\} \mid |\{tp\}| = 1 \), and the following conditions hold:
\[
\begin{align*}
  f_n(sp) &= f_n(tp) \\
  f_d(sp) &= f_d(tp).
\end{align*}
\]

An example of a terminal task is \( dT_{final} = n_1 : d_i \# n_1 : d_i \# phist \). Terminal tasks are void tasks that require no further processing or transmission of data. By definition, a non-terminal data task \( dT = (sp, TP, phist) \) assigned to node \( sp = (n_i, d_j) \) must be handled by that specific node, however, the concept of handling a task (as it will be described later) includes the notion of converting the task to another ‘equivalent’ task (or a set of ‘equivalent’ tasks) to be delegated to other nodes in the network (see Processing Data Tasks).

For notational purposes, a data task can be represented as a single string, separated by ‘\#' symbols to indicate each of the three elements of the tuple. For instance, the data task \( dT = (sp, TP, phist) = ((n_1, d_i),\{(n_2, d_j), (n_4, d_k)\},\{n_0\}) \), can be represented by the string \( dT = n_1 : d_i \# n_2 : d_j \# n_4 : d_k \# n_0 \). In this example, node \( n_1 \) is given datum \( d_i \) and the task of ultimately delivering datum \( d_j \) to node \( n_2 \) and datum \( d_k \) to node \( d_k \) (without relying on node \( n_0 \) (in \( phist \)).

In the general case, the string notation for the data task is defined as follows:
\[
\begin{align*}
  dT &= (sp, TP, phist) \\
  dT &= f_n(sp) : f_d(sp) \# \{f_n(tp_1) : f_d(tp_1)\} \# \{n_1\} \mid tp_1 \in TP \land n_1 \in phist
\end{align*}
\]

This notation will be extensively used for the description of the problems, simulations and results. The column symbols between nodes and datum types are
optional in the notion and might be dropped when there is no risk of ambiguity, allowing
for the task $n_2: d_1 \# n_3: d_2, n_4: d_2 \# n_1$, to be also represented as $n_2 d_1 \# n_3 d_2, n_4 d_2 \# n_1$.

3.1.4 The state of a node

A node $n_i \in N$ can be in one of two types of states: idle, or active. The default
state of a node is in an idle. When a node receives a data task from one of its neighbors it
transitions from an idle to an active state.

If the received task is a terminal-task (i.e. if it is addressed to the node itself), the
node simply consumes the datum and immediately transitions to back to an idle state. In
this case, the node is not required to choose an action for handling the task.

If, on the other hand, the task received is non-terminal the node must choose an
action that will locally handle the task. The local handling of task might involve data
transformations or task delegation to one or more of its neighbors. After applying the
selected action, the node state will transition to idle, and the node becomes ready to
receive new data requests from its neighbors.

An active state of node $n_i$, denoted as $s_i^{\text{active}}$, is always equal to a single data task
being handled by node $n_i$ at the time, and is defined as:

$$s_i^{\text{active}} = dT = (sp, TP, phist) \left\{ \begin{array}{ll}
    f_n(sp) = n_i, & \text{and} \\
    dT_i \text{ is not a terminal state}
\end{array} \right.$$

An idle state of node $n_i$, denoted as $s_i^{\text{idle}}$, is always equal to a set of data tasks
and defined as follows:
\[ s^{idle}_i = DT \mid \forall dT_i \in DT \begin{cases} dT_i \text{ is a terminal state, or} \\ f_n(dT_i) \neq n_i \end{cases} \] (2)

Where \( DT = \{dT_0, dT_1, ..., dT_k\} \) is a finite (possibly empty) set of data tasks. Recall that \( f_n(dT_i) \) is equal to \( f_n(sp_i) \), where \( sp_i \) is the source pair of data task \( dT_i \).

### 3.1.5 Actions

An action is defined as a transition from any active state (\( s^{active}_i \)) to an idle state (\( s^{idle}_i \)), as follows:

**Definition 8**: For a given NET\((N,L)\) and DST\((D,F)\), an action \( a(s^{active}_i) \) or \( a(dT) \), represents a state transition \( s^{active}_i \rightarrow s^{idle}_i \), where \( s^{idle}_i = DT' = \{dT'_0, dT'_1, ..., dT'_k\} \) and \( s^{active}_i = dT \), is defined as follows:

\[
\begin{align*}
(a & = dT = sp, TP, phist) = DT' \\
\exists DST_{path}(dT, dT') & \in DST(D,F) \\
f_n(sp) & \text{ is a neighbor of } f_n(dT') \text{ in } NET, \forall dT' \in DT'
\end{align*}
\]

In Definition 8, \( DST_{path}(d_i, d_j) \) denotes a path in \( DST(D,F) \) connecting states \( d_i \) to \( d_j \) and indicates that a data conversion between the two states is theoretically possible.

Note that not all \( s^{active}_i \rightarrow s^{idle}_i \) transitions constitute valid actions. For instance, the action \( a_i(n_1 : d_1 \# n_3 : d_2 \#) = \{dT' = n_2 : d_1 \# n_3 : d_2 \# n_1\} \), corresponds to a state transition \( dT \Rightarrow \{dT'\} \), that is, \( (dT = n_1 : d_1 \# n_3 : d_2 \#) \Rightarrow \{(dT' = n_2 : d_1 \# n_3 : d_2 \# n_1)\} \). The results
of this action is the state transition \( s_1^{\text{active}} \rightarrow s_1^{\text{idle}} \) (and \( s_2^{\text{idle}} \rightarrow s_2^{\text{active}} \)) and the semantics is that node \( n_1 \) with datum \( d_1 \) and the task to deliver \( d_2 \) to \( n_3 \) (i.e. \( dT = n_1 : d_1 \neq n_3 : d_2 \neq \) ) delegates the job to node \( n_2 \), providing it with datum \( d_1 \) i.e. \( \{dT' = n_2 : d_1 \neq n_3 : d_2 \neq n_1 \} \). This is a valid action and it will result in having node \( n_2 \) responsible for the next move.

However, the action \( a_j(n_1 : d_1 \neq n_3 : d_2 \neq ) \Rightarrow \{dT' = n_2 : d_1 \neq n_3 : d_2 \neq n_1 \} \) would be invalid because the ‘job’ delegated to node \( n_2 \) (i.e. to deliver \( d_2 \) to node \( n_3 \)) is different than the original job for which \( n_1 \) was responsible (i.e. \( n_3 : d_2 \)). Note that in the first case, the target pair-set is maintained through the conversion, which constituted a valid delegation of the task to a neighbor node thus a valid action. In the second example, the target pair-set was modified, which will result in a different outcome as node \( n_2 \) continues to handle the task.

Intuitively, in the first transition, the same data task (or an equivalent of it) is ‘delegated’ to node \( n_2 \), which is now responsible for delivering datum \( d_2 \) to node \( n_3 \). In the second example, the target pair-set no longer includes the original task of delivering datum \( d_2 \) to node \( n_3 \). This constraint on the formation of the target pair set for a valid action is formally stated in the action definition (third condition in the list).

The example shown in Figure 11 satisfies all the conditions specified in the Definition 8. The example specifically highlights the fourth condition in the definition, that is, the required existence of a \( DST_{\text{path}}(f_d(dT), f_d(dT')) \) in \( DST \) for all data transitions.
Note that the validation of the action relies solely on information available to node $n_2$. Aside from the list of neighbors (usually available to any node through the lower layer data link protocols) there are no assumptions or dependencies made on the capabilities of neighbor nodes.

### 3.2 Processing Data Tasks

As previously defined, a node in the network can be in one of two states: an idle state or an active state. When a node is given a data task, it is essentially moved out of its idle state into an active state. The node is then obliged to take one single action that will process the task and bring it back to its original (idle) state. This process is illustrated in Figure 12.
From a local perspective, the processing of a task consists of finding an action that transforms the task and allows the node to fall back into idle state. From a global perspective, the processing of a data task consists of a sequence of transformations 
\{dT_i\}, \{dT_j\}, \ldots, \{dT_k\}\ such that all data tasks in the last set are terminal tasks. 

In order to illustrate the process, let's consider the network graph shown in Figure 13. In this example, node \(n_1\) receives task \(dT\) which moves it out of the idle state and into an active state that corresponds to task \(dT\).

![Figure 13. An example of data task processing process](image)

Given datum \(d_i\), node \(n_1\) is tasked to deliver datum \(d_2\) to nodes \(n_3\) and \(n_4\). Let's consider, for this example that datum \(d_i\) can be converted to datum \(d_2\) through a function \(\alpha\), available at all nodes. Let's also disregard costs at this time, as we're primarily concerned with the flow of task and state transitions.

Given the connectivity constraints shown in the network graph illustrated in the Figure 13, node \(n_1\) 's logical choice is to delegate the task to its only neighbor, node \(n_2\). Let's assume that \(n_1\) has chosen to delegate the job, as is, to node \(n_2\). For that, node \(n_1\) takes the following action:
\[ \alpha(dT) = n_1 : d_1 \# n_2 : d_2, n_4 : d_2 \# \Rightarrow dT' = n_2 : d_1 \# n_3 : d_2, n_4 : d_2 \# n_1. \]

This transition is illustrated in Figure 14, where the changes in the local state of nodes \( n_1 \) and \( n_2 \) are also shown. In practice, the transition created by \( n_1 \) consists of a message being sent from node \( n_1 \) to \( n_2 \) carrying the datum \( d_1 \) and the description of the task \( dT' \).

\[ dT' = n_2 : d_1 \# n_3 : d_2, n_4 : d_2 \# n_1 \]

Figure 14. Data task processing – second step.

Note that now in \( dT' \), \( phist \) includes node \( n_1 \), which essentially prevents \( n_2 \) from using \( n_1 \) as a candidate for task delegation (avoiding loops). Node \( n_2 \) is responsible for choosing a task transformation (i.e. an action) that will allow it to return to the idle state. Given the constraints imposed by the network topology and the history path of the data task, node \( n_2 \) is faced with a number of possible transitions for data task \( dT' \).

Let’s assume that the node \( n_2 \) has decided on the following transition from \( dT' \) into \( \{dT_0',dT_1'\} \):
This transition consists of a local data transformation at $n_2$ (from datum $d_1$ to $d_2$) and two messages being sent. One containing task $dT_0^*$, which is addressed to node $n_3$, and another containing task $dT_1^*$, addressed to node $n_4$.

The state of the network after this task is shown in Figure 15, where each of the end nodes $n_3$ and $n_4$ received a different data task as specified in the transformation.

Note that task $dT_1^* = n_4: d_2 \# n_4: d_2 \# n_1, n_2$ is a terminal task, as per Definition 7, so node $n_4$ simply consumes the datum and returns immediately to an idle state. Node $n_3$, on the other hand, remains in an active state represented by data task $dT_0^*$. Following the same procedure, node $n_3$ will choose a task transition such as:

$$dT_0^* = n_3: d_2 \# n_3: d_2 \# n_1, n_2 \Rightarrow dT_0^* = n_3: d_2 \# n_3: d_2 \# n_1, n_2, n_3$$
The resulting terminal task will be consumed by node $n_3$, which will then return to
the idle state, ending the process of handling the original data task. Note that in the
resulting task, node $n_3$ is in the target-pair set and is also listed in $phist$. This is only
acceptable for terminal tasks.

A data task is considered to have been successfully handled if all its target pairs
received, at some point, the requested data. Partial fulfillment of the task (i.e. some, but
not all of the terminal nodes receiving the datum) is possible in practice but it constitutes
a failure of the overall task. The choice for data task transformation at each node
constitutes an ‘action’. At each step, the process followed by each node for choosing the
appropriate action for handling the current data task is based on the data task itself and
the current state of the network (local topology).

3.3 Model Summary

The model introduced in this chapter forms the basis for the problem description
discussed in chapter 4. The model, however, is not limited to that discussion.

The model describes the notion of the physical network, data transitions and data
tasks, also providing mechanism to support the interaction between these components.

Outside the context of resource allocation, the same formulation can be applied,
for instance, to effectively describe and enforce policy constraints for information release
and task delegation. Other applications that can benefit from this formulation include data
consistency checks between nodes and domains, and applications in cross-domain
information exchange, both topics of very high relevance in tactical environments and
coalition scenarios in operations other than war (OOTW).
Chapter 4. Problem Description

In the previous chapter, we have introduced a model for the environment (NET), the data transitions (DST), and for a data task. The resource allocation problem addressed in this work is essentially defined as finding the best distribution of processing tasks in the network that minimizes both the overall costs for data processing and the costs for data transmission.

In this chapter, we will introduce the cost model for data processing and data transmission at a local node, followed by the formal description of the resource allocation problem.

4.1 Action Cost Estimation

In general, an action combines a data processing step with a data transmission. In its simplest form, an action simply consists on delegating the task to a neighbor node (data transmission) but it could also include local processing of the datum into one or more states.

The weights provided in both the DST and NET diagram allow for the cost estimation of both data transfer and data processing costs. Equation 3, for instance, shows an example for cost estimation for a simple task \( dT = n_i d_i \# n_j d_p \# \text{hist} \) between two neighbor nodes \( n_i \) and \( n_j \) in a given environment.
Equation 3 shows the cost estimate for having node $n_i$ (with datum $d_k$) deliver datum $d_p$ to its neighbor node $n_j$ (assuming that datum $d_k$ can be converted to $d_p$ through a function $f_{st}(d_k, d_p)$ available at $n_i$).

The equation considers that the data conversion from states ($d_k \rightarrow d_p$) happens at node $n_i$, before the transmission to node $n_j$. In this case, the data cost for the data conversion is given by $f_{st}(d_k, d_p) \cdot n_{i \text{factor}}$, while the cost for the transmission is given by $d_p \cdot l_{i,j}$.

Conversely, if the datum $d_k$ was first transferred to node $n_j$ to be then converted to $d_p$, the associated costs would be given by equation (4), where the transmission portion is now weighted by $d_k \text{size}$, and the processing cost by $n_{j \text{factor}}$ (under the assumption that the transformation function $f_{st}(d_k, d_p)$ was also available at $n_j$).

$$\text{Cost}(\alpha(dT)) = f_{st}(d_k, d_p) \cdot n_{i \text{factor}} + d_p \cdot l_{i,j} \quad \text{(3)}$$

$$\text{Cost}(\alpha(dT)) = d_k \text{size} \cdot l_{i,j} + f_{st}(d_k, d_p) \cdot n_{j \text{factor}} \quad \text{(4)}$$

Equations 3 and 4 produce exactly the same outcome at node $n_j$ (i.e., datum $d_p$), however at different costs. An interesting decision to be made by node $n_i$ is to determine which of the solutions (3) or (4) is the best (lowest cost) for the given task, namely provide node $n_j$ with datum $d_p$, without necessarily knowing the capabilities or data transformation costs of node $n_j$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
For a generalized action converting \( dT \) to a set \( DT = \{dT_0, dT_1, ..., dT_k\} \) (equation 5), where \(|DT| = k\), the overall costs consist in the sum of all transmission and data transformation tasks.

\[
a(dT = (sp, TP, phist)) \Rightarrow \{(sp_0, TP_0, phist'), (sp_1, TP_1, phist'), ..., (sp_k, TP_k, phist')\}
\]

(5)

The data transmission sub-tasks are assumed to be independent from each other and their costs can be summed directly. The costs for the data conversion tasks, however, are estimated through a combined function \( f_{d^s} \left( ds, \{ds_0, ds_1, ..., ds_k\} \right) \), where

\[d_{sp_j} = f_{d^s} \left( sp_j \right),\]

that leverages from the commonality in the data. The data cost, in this case, can be estimated by equation (6).

\[
\text{Cost}(a(dT)) = f_{d^s} \left( d_s, \{d_{sp_0}, d_{sp_1}, ..., d_{sp_k}\} \right) \cdot n_{\text{factor}}^p + \sum_{j=1}^{k} d_{sp_j} \cdot f_{d^s} \left( sp_j \right)
\]

(6)

In equation 6, the first term describes the combined costs for data processing while the second term aggregates all data transmission costs. The data processing costs are defined by the cost transformation of the minimum spanning tree from \( d_s \) (source) to \( \{d_{t_1}, d_{t_2}, ..., d_{t_p}\} \) (destinations), over the DST graph.

For example, consider the task shown in Figure 16. The graph shows only the local information available to the node, including its local neighbors and the DST for the data type described in the task.
Figure 16. A sample data task for cost estimation

If, given the state described in Figure 16, node \( n_2 \) chooses the action:

\[
\alpha(dT = n_2 : d_1 \# n_8 : d_2, n_5 : d_4, n_9 : d_5 \# n_2) \Rightarrow \begin{cases} 
&T'_2 = n_1 : d_2 \# n_8 : d_2 \# n_2 \\
&T'_1 = n_6 : d_4 \# n_5 : d_4 \# n_2 \\
&T'_3 = n_7 : d_5 \# n_9 : d_5 \# n_2
\end{cases}
\]  

(7)

The outcome of the action is shown in Figure 17. At the left side of the figure, the network graph shows the actual data transmissions from node \( n_2 \) to nodes \( n_1, n_6 \) and \( n_7 \), as defined in equation 7.

Figure 17. The minimum spanning tree for data processing cost estimation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
At the right side of the Figure 17, the minimum spanning tree for the data conversion is shown over the DST diagram. The cost of the action is shown in equation 8, where the $f_{st}(d_1, \{d_4, d_9, d_5\})$ is defined as the cost of the minimum spanning tree from $d_1$ to $\{d_4, d_9, d_5\}$.

$$\text{Cost}(a(dT)) = f_{st}(d_1, \{d_4, d_9, d_5\}) \cdot n_2^{\text{factor}} + \sum_{j=1}^{\left|\mathcal{D}_T\right|} \left( f_{a}(dT_j) \right)^{\text{size}} \cdot f_{st}(d_1, f_a(dT_j))$$

$$= f_{st}(d_1, \{d_4, d_9, d_5\}) \cdot n_2^{\text{factor}} + \left( d_4^{\text{size}} \cdot f_{st}(d_1, f_a(dT_1)) + d_9^{\text{size}} \cdot f_{st}(d_1, f_a(dT_2)) + d_5^{\text{size}} \cdot f_{st}(d_1, f_a(dT_3)) \right) n_7$$

In the general case, the cost of an action $a(dT)$ is given by equation (9)

$$\text{Cost}(a(dT)) = DT' = f_{st} \left( f_{a}(dT), \{f_{a}(dT_j)\} \right) \cdot f_n(dT')^{\text{factor}} + \sum_{j=1}^{\left|\mathcal{D}_T\right|} \left( f_{a}(dT_j) \right)^{\text{size}} \cdot f_{st}(f_a(dT_j), f_a(dT_j))$$

, for $\forall dT_j \in DT'$

### 4.2 The Resource Allocation Problem

As defined in the previous item, the cost of an action reflects the local loss incurred by the node on handling the data task.

From an external observer, however, the cost for delivering the data from the source node to the set of receivers is given by the sum of the task processing costs at each step. For instance, consider the example shown in Figure 18, where node $n_1$ is given task $dT$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Upon receiving the data task, node $n_i$ moves to an $s^\text{active}_i$ state and will choose the best (cheapest) action that will bring it back to its idle state. Recall, that from $n_i$'s perspective, the information available about the request and the network is not the complete NET graph, but instead a local view of the topology, illustrated in Figure 19.

Figure 18. A global cost estimate example

Figure 19. Local view of the environment from $n_i$'s perspective

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Based on the information illustrated in Figure 19, node $n_x$ has to choose the best action to handle task $dT$. Based on the cost estimation provided in equation (8), the best option available to node $n_x$ is to send the data request (without any local data processing) to node $n_2$, that is, to take the action $a(dT = n_x : d_1 \# n_4 : d_2 \#) \Rightarrow n_x : d_1 \# n_4 : d_2 \# n_x$.

The cost of such action for this example is $(d_{size} \cdot \mu^{factor} = 200)$, which is the lowest cost for node $n_x$. If taken, however, the action will put node $n_2$ in state $n_2 : d_1 \# n_4 : d_2 \# n_1$, from which the best action would to handle the task would have a minimum cost of 2000. (note that node $n_2$ does not have function $a$ available so the only possible action is to forward the task to node $n_4$). The overall (accumulated) cost of the task amounts 2200, which is not the best overall cost.

The overall cost (from a systems perspective) to realize task $dT$ is the sum of the costs incurred by all nodes in the distribution tree, that is:

$$TotalCost = \sum_i^{N} \text{Cost}(a_i(dT)) + MissCost$$

(10)

Where $A = (a_1, a_2, \ldots, a_k)$ is the ordered set of actions taken by different nodes through the course of processing the task and $MissCost$ is a fixed (maximum) cost value associated with missing the delivery of a packet or reaching a task state from which there are no available actions and the packet has to be dropped.

Let's define a set $NTask = \{n_1, n_2, \ldots, n_k\}$ of all nodes involved in processing a given data task in a network, that is, all nodes involved in the data distribution tree for a given data type.
The resource allocation problem consists of finding the best allocation of resources in the network that would realize the task successfully with the lowest possible cost. \((\arg_{A} \min(TotalCost))\).

Each node, has to make a local decision that is not necessarily the best local choice, in order to allow for a global solution that will evolve to a minimum aggregate cost (in the optimum case).

Finding a solution for the resource allocation problem essentially consists on minimizing, at run-time, the global cost for all concurrent data requests in the network, that is, to solve the cost estimation problem in a dynamic environment. In practice, the problem consists of incrementally improving the cost estimate at each node, recognizing that the cost functions themselves are dynamic.
Chapter 5. Learning Strategies for the Resource Allocation Problem

For all practical purposes, the combinatorial nature of the resource allocation problem makes it very difficult to use strategies based on global information. Sub-optimal solutions are usually well accepted as long as they can be quickly identified and initialized to minimize latency.

In fact, a general approach used for these types of problems is to identify and assign a “reasonable” resource allocation solution, in order to gain some time for examining the options for a more refined alternative while the data is flowing. From that perspective exhaustive searches are usually not practical and heuristic searchers both at the local (and sometimes global levels, for small scale problems) are usually adopted.

In this work we propose the application of online learning strategies for the resource allocation problem in tactical networks. The goal is to formulate the problem in a way that supports the application of a resource coordination protocol that will leverage from previously established localized learning techniques.

In order to achieve such formulation we will introduce a few modifications in original description of the problem.
5.1 A Modified Action Cost Function

As illustrated in the example presented in chapter 4, the search for a local best cost at the node level might not necessarily imply a global minimum for the complete task.

Allowing nodes to perform a search (or even a heuristic) search for a solution requires some level of state information (from remote hosts) to be maintained at each node, or to be discovered on demand. In both cases, however, there is usually a significant overhead involved.

Ideally, each node should be able to obtain all the information necessary for making its local decisions from its immediate (one hop) neighbors. Similar to conventional data agnostic routing algorithms, a node should be able to identify an equivalent of a next hop for the data task.

However, the ‘next hop’ in this case might be a ‘set’ of nodes, each with a pre-defined set of tasks. Furthermore, the discovery and maintenance of such information should be done with minimum overhead.

From that perspective, the action cost function can be augmented to weight the local cost estimations with a factor for each target neighbor that reflects their expected costs involved in handling the assigned tasks.

Redefined to accommodate cost estimates from peer nodes, the action-cost function for a task \( a \left( dT = (sp, TP, phist) \right) \), is defined as follows:
The first part of the expression (in brackets) constitutes the costs associated with locally processing the action and transmitting the data to each of the selected target nodes. Like before, $f_{st}(d_T, \{d_1, d_2, d_3\})\cdot n_{\text{factor}} + \sum_{j=1}^{[\text{DT}']} \left( f_{d}(dT'_j) \cdot t_{\text{factor}}^{\text{2, DT'}} \right)$ is the combined processing cost for converting the datum $f_d(d_T)$ into each of the target task’s source data $f_d(dT'_j)$ for $\forall dT'_j \in \text{DT}'$, and $t_{\text{factor}}$ is the transmission cost factor between nodes $i$ and $j$.

The second part of the cost expression (second and third members) is the part that is being added to allow for a global cost reduction. In equation (11), the term $\prod_{j=1}^{[\text{DT}']} P_{\text{suc}}(dT'_j)$ constructs an overall probability of successful transmission for the action by multiplying the individual probabilities of success transmission for each of the target data tasks.

The transmission success probability $P_{\text{suc}}$ provides a notion of the reliability of the link and refers only to the probability of successfully transmitting the task to its immediate neighbors.

Every time a data task is transmitted the sender will receive an immediate confirmation if the message was delivered or not. Such information is normally provided.
by standard MAC protocols. For every transmission, the node will update the success probability associated with that operation and use the estimates for future cost estimations.

By multiplying the estimated probabilities of success for each of the target tasks (assuming they are independent events here), an estimate for the probability of success of the whole action can be obtained, again, only in terms of local transmission. There’s not enough information at the local node to estimate the probability of an action succeeding all the way to the target nodes.

Note that two separate sub-tasks addressed to the same node will develop separate statistics on the reliability of the link, even though they might refer to the same neighbor. In most cases, the reliability of the link would be the same for any task (or packet) sent through that link, but in the special case of tactical networks, traffic is often bound by content-based policies which might differentiate the level of bandwidth available to specific data types, resulting in different network behaviors for different tasks.

As defined, the action cost function supports the effects of such behavior by keeping the estimates separate for each action (as opposed to aggregate them in a link basis) and relying on link-based reliability only in the case when action estimates are not available.

\[ C_{f_n(dT')} \] is an estimate that each node maintains about the best cost reported by its neighbor \( f_n(dT') \) for handling task \( dT' \). This cost estimate is a running average of the costs announced by that node while handling the task and is mostly obtained through promiscuous listening or direct announcements due to task completion or failure.
The sum of cost estimates from all neighbors involved in the candidate action is then multiplied by the probability of success of the action (the product of all independent probabilities of success).

A penalty for task failure is added for the complement of the probability (probability of communications failure), where MissCost is a constant defined as a parameter to the algorithm and represents the penalty for missing a task. In general, this value is equal to the maximum value of a task failure (see 7.2.1).

5.2 Learning Policies

With the action-cost function modified to account for expected processing costs of neighbor nodes, each node is then responsible for learning which action should be taken when a data task is received, given a local list of neighbors.

Following Sutton's (1998) terminology, a policy $\pi$ is a mapping from state to action, where the notion of state in this case is the active state of the node

$(\delta^{\text{active}} = dT)$ and the current set of neighbors $H = \{h_1, h_2, ..., h_k\}$ to the node. The tuple $(dT, H)$ provides the necessary information to construct the set of possible actions for selection.

One way to find the best action is to try each and, through an exhaustive search identify which task would yield the lowest cost. There are, however, more efficient ways to choose the next action based on the incomplete experience acquired in early trials. Under certain constraints and assumptions, the problem can be reduced to a well known online learning problem in game theory known as the k-arm bandit problem.
5.2.1 The k-Arm Bandit Problem

The k-arm bandit problem proposed by Robbins (1952) defines a scenario where a gambler must choose one of ‘k’ slot machines to place a bet. At each bet (which are assumed to have fixed value) the player pulls the arm of the slot machine and collects the reward in cash, if any. The objective of the game is to maximize the gains of the gambler over a fixed number of plays.

In order to achieve his/her goals in maximizing the gains, the gambler must be able to identify, as early as possible which of the machines gives the best odds for winning. Assuming, for now, that each arm returns its rewards based on a fixed pre-defined distribution that is unknown to the gambler, the challenge in the problem is to decide at each play, when to exploit a ‘known’ distribution versus exploring a new one.

After playing a few times in one of the machines, the player can estimate what are the odds of winning on that ‘arm’ and decide on his/her next action. If the gambler assumes that to be the best ‘arm’ and continues playing, he might fail to discover an arm with better odds.

Conversely, if the gambler spends too much time trying every machine to gather statistics he would be missing on the plays that could have been done on the ‘best’ arm. There are many variations of the k-arm bandit problem, for instance, in the way the rewards functions are assigned or in the information release to the gambler at each play, however, the common objective in the game is usually the same, that is to learn a strategy for balancing the exploitation of the currently best (or greedy) solution with the exploration of new (and possible better) solutions, in order to maximize accumulated future rewards.
The dichotomy ‘exploration’ and ‘exploitation’ is a characteristic of most online learning problems and have been addressed in detail in the context of Reinforcement Learning by Sutton (1998), who enumerate a number of strategies for balancing exploitation and exploration for maximize gains in the long run.

Amongst the strategies presented by Sutton, two groups are of particular interest for this work, the $\varepsilon - greedy$ and the softmax exploration strategies.

The $\varepsilon - greedy$ strategies essentially define a $\varepsilon$ value that determines the ratio between exploratory and exploitative plays. It establishes that $\varepsilon$ percent of the time, the gambler will not use it’s best known ‘arm’ and will choose, instead, one of the other arms with equal probability.

The $\varepsilon$ value does not need to be fixed through the life of the game but, in $\varepsilon - greedy$ strategies they normally are, and the assumption is that non-greedy options are chosen uniformly and that eventually, every ‘arm’ will be tried.

Another set of strategies are the softmax action selection strategies. In these types of strategies, the choice of the next action (or ‘arm’ to be played) is obtained from a probability distribution of the expected reward of all available arms.

A very common distribution used for these types of strategies is the Gibbs, or Boltzmann distribution. After a few plays, the probability of choosing arm ‘a’ for the next play is proportional to the exponential of the accumulated observed for that action (equation 12):

$$P_{a}(t) = \frac{e^{Q(a) / \tau}}{\sum b=1^n e^{Q(b) / \tau}}$$

(12)
Where $P_a(t)$ is the probability of choosing action (a) at play (t), $Q_a(t)$ is the accumulated rewards for action (a) up to time (t), and $\tau$ is a parameter known as the ‘temperature’ for the distribution. The higher the temperature the more uniform-like the distribution will seem and the lower the temperature the greater will be the differences in the probabilities arising from differences in accumulated values.

Equation 12 assumes that there’s a mechanism for estimating the accumulated value of every action up to time $t$, in order to estimate the probability distribution for task selection.

In the version of the game known as ‘full information game’ the complete set of rewards is available to the player at each step. In ‘full information games’ when the gambler chooses one of the arms to play it receives the rewards from that arm but he is also informed of the outcome of all other ‘arms’ on that play, as if they had been selected.

For these types of games, at each step, the gambler can choose an arm and collect rewards information from all arms to build the estimates necessary for equation 12, quickly finding a strategy for playing the next iteration.

Auer et al. (1998) proposed an algorithm called Hedge based on the previous work of Freund and Schapire (1997) that allowed for action selection from a Boltzmann distribution built from a full information game.

A more interesting variation of the game is the ‘partial information game’, in which only the reward for the chosen arm is provided to the gambler. For these types of games, the construction of the probability distribution shown in equation (12) is a less evident, as the accumulated reward for non-played ‘arms’ is unknown.
One solution for the partial information game was also proposed by Auer et al. (1995) who introduced the Exp3 algorithm as an extension of the Hedge algorithm.

The main idea in the Exp3 algorithm is essentially to create a simulated reward vector (i.e. a reward for each 'arm') at each iteration and use the full information required by the Hedge algorithm to estimate the probability distribution.

Furthermore, it adjusts the probability distribution returned by Hedge by mixing it with a uniform distribution such that unknown actions retain at least a small probability of being tried. The choice of selected simulated rewards is such that the expected values of the actual rewards are maintained.

In Exp3, the Boltzmann probability is adjusted by the expression

\[ p'_i(t) = (1 - \alpha) \cdot p_i(t) + \frac{\alpha}{K}, \]

where \( p_i(t) \) is the Boltzmann probability of taking action (i) at step (t), \( \alpha \) is a factor that defines the mixture with the uniform distribution and \( K \) is the number of possible actions (arms).

The pseudo-code for the Exp3 algorithm is shown in Auer et al. (1998), with the definition and proof for the lower bounds on the gain of the algorithm, as well as the bounds for the expected regret (i.e. difference between the best possible gains and the gains obtained by the algorithm).
Figure 20 shows both the original Hedge probability distribution (based on a Boltzmann distribution assuming full information) and the probability distribution estimated by Exp3, adjusted by the simulated returns the mixed with a uniform distribution (in this example using $\alpha = 0.2$). The temperature coefficient in this example is $\tau = 1$ and the probabilities are shown after 10 tries on a 10-arm bandit problem.

In this example, each arm follows a normal distribution with average and variance shown in the upper left corner of the image. Note that, for the Boltzmann distribution, the low probability arms are significantly lower than the arms with higher accumulated values, even when the difference in accumulated values is small.
Figure 21 shows the same example now using a higher temperature value of $\tau = 2$, implying a higher tendency for exploration. Similarly, the Exp3 adjust of the curve maintains a minimum probability level for the unknown rewards.

The partial information problem addressed in the Exp3 algorithm is similar, in many aspects, to the resource allocation problem discussed here. Furthermore, the algorithm offers well-defined bounds for asymptotic gains.

The main difference is in the fact that Exp3 assumes absolutely no information about the arms that were not played while in the resource allocation problem (for some of the unknown arms — or actions, in that case), there's partial cost information available.
From that perspective, the resource allocation problem is not a full information game but it provides the gambler (i.e. the local node choosing the resource allocation action) more information than what would be normally available in a partial information game.

5.2.2 Learning Resource Allocation Strategies in Tactical Networks

As formulated in this work, the resource allocation problem in tactical environments can be treated as a k-arm bandit problem where each ‘arm’ in this case represents an action chosen by a node for handling a data task.

The main difference between this type of allocation and the traditional k-arm bandit formulation is that the rewards (or negative costs, in the case of the resource allocation problem) for a given set of actions will only stabilize after downstream nodes start finding their own optimal policies.

Because learning is local to a node, based only on its own information or information provided by immediate neighbors, the actual cost for the complete processing of the task might need to stabilize at the target (destination) nodes before it propagates its final (stable) estimate to the upstream nodes.

Under the assumption that, for a given topology there is one single lowest cost sequence of steps for minimizing the task, it is intuitive to expect that a stable solution will be found as every node in the tree finds its best policy (optimal substructure property).

To discover a data distribution tree, each node will locally make a decision on whether to process or delegate the data task to a neighbor. In fact, the decisions might
also include the fragmentation of the tasks into sub-tasks that can be either processed locally or distributed to a neighbor's node in any way.

From an online learning perspective, the node is expected to learn, from experience, how to handle a given task based on its current set of neighbors in order to minimize overall costs.

As the environment changes, the node must be able to re-learn and adapt to the new configuration. Learning is expected to take place significantly faster than most changes in the environment.

The resource allocation formulation proposed in this work was intended to closely fit the description of an online learning problem in order to allow us to leverage from the research in that field to build a solution for the problem. The proposed algorithm utilizes both the softmax strategy (using a Boltzmann distribution) for action selection and slightly modified $\epsilon$-greedy strategy.

5.2.3 Convergence Properties

The resource allocation problem formulated as a k-arm bandit problem inherits the same convergence properties and guarantees previously defined in the literature. In the proposed algorithm, each node converges to a local policy that defines an optimal action distribution that minimizes costs for a given data task and set of neighbor nodes.

The local cost equations at each node, however, also include cost estimates from its immediate neighbors, which in turn, will minimize their own cost functions. Asymptotically, as each node finds its optimal set of policies and costs for a given task the complete network evolves to a global optimum.
Following the Bellman principle, a sequence of locally optimum sub-paths will necessarily be an optimum path itself (optimal substructure property), thus the algorithm is guaranteed to asymptotically globally converge for a given data stream, if each of the nodes in the stream converge to an optimum policy.

Each node maintains an estimate of costs for each of its neighbors, which allows for the cost estimation of its actions. With the convergence of its neighbors, the estimate will approach the actual cost value and the game can be mapped to a full information game. Auer et al. in 1998 showed the convergence bounds for Hedge algorithm, which are directly applicable to the formulation introduced in this work.

Let's define \( G_t(t) = \sum_{i=1}^{t} x_i(t) \) as the accumulated gain of action (or arm) \( i \) at time \( t \). Essentially, \( G_t(t) \) constitutes the total accumulated gain of always choosing action \( i \) from the beginning of the game \( t' = 1 \) through the current time \( t \). The Hedge algorithm (similarly to the SoftMax strategy previously described) provides, at any time \( t \) a probability for choosing each possible action \( i \) in the next play. The probability of choosing an action \( i \) as a next action is given by:

\[
P_i(t) = \frac{e^{\eta G_i(t-1)}}{\sum_{j=1}^{n} e^{\eta G_j(t-1)}}
\]  

(13)

Note that the differences between equation 13 and equation 12 are essentially notational, however in Hedge, the notion of a temperature \( \tau \) is replaced by its inverse parameter \( \eta \) (i.e. \( \eta = \frac{1}{\tau} \)), with equivalent meaning.
Following Auer's notation, it can be shown that, for any $\eta > 0$ and for any sequence of reward vectors $x(i) \in [0,1]$, the expected gain provided by the Hedge algorithm is bounded as follows:

$$E[G_{\text{hedge}}] \geq \frac{\eta E[G_{\text{max}}] - \ln K}{e^\eta - 1}$$  \hspace{1cm} (14)

Where $E[G_{\text{max}}]$ is the maximum return that could be obtained if the best action would have been chosen at each step, $K$ is the number of possible actions and $\eta$ equal to the inverse of the temperature.

The difference between the maximum possible gains $E[G_{\text{max}}]$ and the expected gains yielded by the algorithm are defined by Auer as a measure of regret. It can also be shown that, with an appropriate choice of temperature at each step, the regret obtained by Hedge is upper bounded by $\sqrt{2 \cdot T \cdot \ln K}$, that is, $E[G_{\text{hedge}}] = E[G_{\text{max}}] - \sqrt{2 \cdot T \cdot \ln K}$, where $T$ is the number of plays and $K$ the number of possible actions.

The bounds obtained for Hedge are directly applicable to the SoftMax algorithm utilized in our approach. The full information game assumption that allows the calculation of the accumulated gains at each step are obtained, in our formulation via estimates of neighbor costs and local costs for data processing and transmission.

As each node converges with the bounds specified above, the overall convergence of the complete data distribution tree is, in the worst case, equal to the product of the bounds multiplied by the maximum depth of the tree.
5.2.3.1 Adaptations of the \( \epsilon \)-greedy strategy for the proposed algorithm

The \( \epsilon \)-greedy strategy used in the proposed algorithm does not use a fixed epsilon for exploration. The probabilities for exploring new actions (at each iteration) are proportional to how ‘new’ and ‘unknown’ the environment is to the node.

The idea is to create an incentive for exploration when there are significant changes in the environment.

In tactical network, changes in topology will create (or remove) a relatively large number of actions available to the node almost instantaneously. Fixed exploration strategies are not appropriate for this type of environment because they can’t adapt to changes in exploration requirements.

5.2.3.2 Adaptations of the SoftMax strategy for the proposed algorithm

The softmax strategy used as part of the proposed algorithm relies on a Boltzmann distribution of the expected reward of each action.

The rewards in this case are calculated from averages of the costs accumulated for each action (or estimated based on local processing/transmission figures and partial cost from other actions).

The average cost values are subtracted from a constant maximum value and the remaining is regarded as the reward for the action. Low cost actions, as expected will yield a higher reward. The cost estimates also take into account a link reliability measure estimated by counting the ratio of success/fail transmissions.

Besides using averages instead of accumulated rewards, another modification is the addition of a sliding window to limit the maximum variance of costs in the candidate actions.
High differences in costs do not result in a significant problem for the long term aggregate gains expected by algorithms such as Hedge and Exp3, but they do represent a problem in communication networks as they might evolve to unstable solutions.

This effect will be later illustrated in one of the simulation experiments for the algorithm. The sliding window proposed in this work restricts the number of actions considered based on their relative costs, allowing the algorithm to stabilize and converge to the action with the lowest cost.
Chapter 6. The Resource Allocation Algorithm

The resource allocation algorithm introduced in this work is essentially reactive, that is, the discovery and allocation of resources are done on demand, when a data task is provided. The underlying assumption is that data packets are part of a data stream from a source node to potentially multiple destination nodes. Under that assumption, the cost for caching and retransmitting a data packet is much higher than the cost of missing the delivery of a data packet. This assumption can be easily relaxed with minor changes to the protocol.

At any time, each node is always aware of its local neighborhood and the DST for different data types. The local neighborhood information is essentially a partial view of the NET diagram obtained and maintained by the MAC layer. The partial view of the network essentially constitutes of all neighbor nodes and their link states. The DST is assumed to be available to every node for any type of data. In practice, that information can be embedded with the data packet itself with minimum overhead but because the DST is static we disregard its distribution in the protocol.

In addition to the list of neighbors and the DST diagram, each node maintains two tables, which constitute the information learned by each node. The state-action table and the transmission success rate table.
In tactical networks, changes in network topology might be common and keeping the state for all possible actions will quickly become unmanageable. The approach adopted in the proposed algorithm is to calculate the possible set of states and their cost estimates at each iteration based on the current set of neighbors. Historical costs for each task can be maintained as a single number (the average cost).

6.1 Data Maintained at each node

Each node in the network maintains the following information:

a) *A local view of the Network Graph (NET):* Consists essentially of the current list of neighbors for the node.

b) *Data Transition Graph (DST):* This information is available to any node for any data type that it can handle. The data transition graph can be provided as part of the data task or obtained by the on-demand when a new data type is obtained.

c) *A Data Task Cost (DTC) Table:* This is a table that maintains, for each data task \(dT\), a tuple \((Cost, Psuccess)\), where both \(Cost\) and \(Psuccess\) are real numbers with the first representing the best known cost for processing the specific data task and the success rate for such task (a measure of reliability).

6.2 Message Types

The proposed resource allocation algorithm relies on the flow of the data messages themselves to exchange the necessary information for maintaining the state of nodes. Only when a data task fails to be handled and can no longer be propagated, a node will broadcast (locally only, not a flood) a message reporting the failure.
6.2.1 The Data Message

Data messages are extended with additional information about the task to be performed, the accumulated cost of the task, the sender’s original data task and the best cost known to the sender for processing that original task. The data message contains the fields illustrated in Figure 22.

Figure 22. The description of a data message

The first field of the message is an identifier for the task. This field is essentially used to track down the propagation of a task through the network and does not affect the way in which messages are handled or exchanged by each node.

The Sender’s \(dT\) field contains a complete description of the data task that was being handled by the sender node for creating the current message.

The data message can be flagged as ‘exploratory’ or not. An exploratory message actually has empty payload and is treated and exchanged in the same way that any...
standard data message. This capability was added to the proposed algorithm to support the off-line or parallel exploration of actions in very large domains. Experiments utilizing this feature were not included in our current results, but are recommended for future research studies.

All data messages contain a data task to be processed by the receiving node. By definition, the receiver of the message is the node named in the source pair of the data task (i.e. \( f_n(dT) \)).

The accumulated processing cost is not used by the algorithm for any decisions through the delivery process but it provides to the end node a total estimate of the actual aggregate cost involved in processing the message. This is necessary to test the effectiveness of the learning algorithm.

The goal of the algorithm as a whole is to minimize the sum of these costs at all end nodes, which is done through the indirect minimization of the task costs reported at each node.

### 6.2.2 Broadcast Cost Messages

Broadcast cost messages are rather simple and contain only the Sender’s \( dT \) and a cost for the task. These types of messages are used by a node when it can no longer handle a data task and is forced to drop it. Because there’s no continuation of the data task, the only way by which a node can relay the information that the task has failed is through a separate message, in this case a broadcast cost message. The cost of a failed task is pre-defined as the highest possible cost in the network. The effect of the broadcast is to create the highest possible disincentive for sending this task again to the same node.
The message is returned to the sender via broadcast, rather than unicast, so the failure of the task can be registered by all neighbor nodes.

6.3 Building Cost Estimates for the Neighbors

The approach adopted here is to construct the cost estimate for actions based on partial estimates of action fragments that were previously used for each node. This is significantly different from the way costs are traditionally collected for online learning strategies but it reduces the number of trials necessary to estimate the cost of an action, under the assumption that partial tasks are independent.

Estimates based on partial information are only used to prioritize the exploration of unknown tasks. When the actual cost of a task is received, it overrides the previous partial estimates. If the assumption of independence between partial tasks holds, the actual cost of the task is lower bounded by the sum of the partial costs. If the assumption of partial dependency is violated, the only side effect is a poor prioritization of the exploration procedure for unknown tasks.

Information about the cost of a neighbor for processing a task can be obtained by observing the messages exchanged by each neighbor with its peers, that is, by promiscuously listening on task cost announcements sent as part of data messages. Consider, for instance, the network illustrated in Figure 23. In the example, node $n_i$ has task $dT = n_i d_1 \# n_o d_2, n_r d_2$ and based on previous experience it has an estimate of its best cost for handling $dT$ (let’s disregard the DST constraints for now). The cost estimate is based on equation 11, using a neighbor’s cost estimates from previous interactions.
When the action $a(dT = n_1d_1 \# n_6d_2, n_7d_2) = \{(n_3d_1 \# n_7d_2)(n_4d_1 \# n_6d_2)\}$ is chosen by $n_1$, two data messages are created with the delegated tasks respectively to nodes $n_3$ and $n_4$. Appended to the data message is the sender’s data task $dT$ and its best cost for handling the tasks.

Recalling the promiscuous listening property of wireless ad hoc networks (in particular 802.11 networks), the message sent by node $n_1$, even though it is addressed to node $n_3$, is overheard by all nodes connected to $n_1$. Only node $n_3$ will handle the data task but all other nodes overhearing the message will recall that task $dT$ (which implies node $n_1$) can be handled with a best reported cost so far of $Cost(a(dT))$. That
information will be maintained and averaged accordingly for future costs estimations by other nodes.

When node $n_3$ handles task $dT'_0 = (n_3, d_1, \#n_3, d_2)$ it will, again send a message (possible to node $n_7$), to which it will attach the sender’s data task (now $dT'_0 = (n_3, d_1, \#n_3, d_2)$) and the best cost for that task. The message will be overheard by $n_1$ (connected to $n_7$), who will adjust its expected cost for task $(n_3, d_1, \#n_3, d_2)$ and use the updated estimate for the next time it needs to handle a task that can leverage from that information.

The cost learnt for each data task is maintained in the nodes ‘data task cost table’. Information obtained through promiscuous listening provides no details about the task’s probability of success, which is assumed to be 1 (100% reliability) in that case.

The probability of success for a task is adjusted when the node actually attempts to take the action and send a data message to that neighbor. When a node sends a data message to a neighbor it receives an immediate acknowledge at the level of the MAC layer that confirms whether the message was received or not. Messages dropped due to network failure do not affect the costs reported by neighbors on how they can handle a task, but it does affect their reliability for that task, from the sender’s perspective.

### 6.4 Action Selection using SoftMax and $\varepsilon$-greedy Strategies

The selection of an action starts with a node first checking if the message is addressed to itself. If the node receiving the data task is listed in the target pair list it will first create a sub-task that is terminal.
For example, if node $n_3$ receives task $dT = n_3 d_2 \# n_3 d_4, n_2 d_3 \# \text{phist}$, it must verify that the task is to be (at least partially) consumed by node $n_3$ itself, as it is listed in the target pair set. After that, the node already know that any action that will be chosen to handle the task will include the terminal sub-task $(n_3 d_2 \# n_3 d_4)$ which involves only the cost of converting datum $d_2$ into $d_4$ at node $n_3$.

If there were no remaining nodes in the target pair set, the task would be completed and would no longer be forwarded. Node $n_3$ in that case would simply send a broadcast message announcing the success of the task, with the attached cost for handling the final conversion.

In the general case however, based on the received data task $dT$ and the current set of neighbors, the selection of an action first requires the complete set of possible actions to be built for cost evaluation.

The list of neighbor nodes is first pruned to remove any neighbors that are also listed in the $\text{phist}$ of the data task. The remaining neighbors will be considered as candidates for receiving the task.

Consider, for instance the example shown in Figure 24 where node $n_4$ receives message $n_4 d_i \# n_6 d_2, n_7 d_2 \# n_i$ from $n_i$. Let’s also assume that the transition between datum $d_i$ and $d_2$ is done through function $\alpha$.
Figure 24. A list of possible actions

Figure 24 shows the list of all actions available to node \( n_4 \). The set of actions are created based on the received task \( n_4 d_1 # n_6 d_2, n_7 d_2 # n_1 \) and the list of neighbors \( \{n_1, n_2, n_6\} \) from which node \( n_1 \) was eliminated because it is listed in \( dT \)'s phist. The selection of an action will follow either an \( \varepsilon \)-greedy or a SoftMax strategy, using the Boltzmann distribution. Even before receiving any feedback from neighbors, a prior can be estimated for the actions based on their local cost estimations. If there's absolutely no information about any prior probability for the actions, all tasks are considered equiprobable and the selection is based on a uniform random draw. The most likely scenario, however, is that node \( n_4 \) might have collected information reported by both nodes \( n_3 \) and \( n_6 \) either due to direct interactions with them in previous requests, or through promiscuously listening to messages from these nodes.
In its data task cost table (recall DTC in section 6.1), node $n_4$ maintains a running list (fixed size circular list) of all costs received per task, with an associate probability of success (if available). Just for illustration purposes, let's consider that the data task table illustrated in Figure 25.

\[
\begin{align*}
(n_3d_1 \# n_6d_2 \# n_1, n_4) & \quad 220 \quad .9 \\
(n_3d_1 \# n_6d_2 \# n_1, n_4) & \quad 210 \quad 1 \\
(n_3d_2 \# n_6d_2 \# n_1, n_4) & \quad 198 \quad 1 \\
(n_6d_1 \# n_6d_2 \# n_1, n_4) & \quad 205 \quad 1 \\
(n_6d_2 \# n_6d_2 \# n_1, n_4) & \quad 205 \quad 1 \\
\end{align*}
\]

Figure 25. A data task cost table

As previously defined in section 4.1, the cost of an action has two components, one part that is due to the costs involved in local processing and data transmission, and another part based on estimates of how target neighbors would be able to handle their tasks. The information contained in the data task cost table allows for the estimation of the second part of the cost. For instance in this example, action $a_6$ has a cost of 493 (205 + 198). Adding to this estimate the costs for transmitting the datum to each of the nodes $n_3$ and $n_4$ (there’s no data conversion costs associate with action $a_6$) one can build an estimate for action $a_6$. These estimates will be used for action selection, based on a Boltzmann distribution.

Alternatively, the proposed algorithm can also use an $\varepsilon$-greedy-like approach for selecting action. In this case, an epsilon threshold is established and used as a probability for exploration. In conventional $\varepsilon$-greedy strategies, there is one action that is 'known' (at least so far) to be the best, or greedy, action which will be used with probability (1-
all other actions have an uniform probability of being explored equal to $\varepsilon/K$ (where $K$ is the number of remaining actions).

In the algorithm proposed here, there are two basic differences: a) first, the value of $\varepsilon$ is not fixed a priori, but it is calculated, at each iteration, based on the number (and mix) of non-greedy actions. In the proposed formulation, all actions (greedy or not) can be classified in one of three groups:

a) Unknown Actions: These are actions that are known to be possible, at least in theory, but lack complete cost information. In general, local processing costs associated with these types of tasks are directly available (or can be calculated locally).

b) Known Actions: These are actions that have been tried recently and a reasonably good estimate of their costs is available.

c) Invalid Actions: Similar to known actions, these are actions that have been tried in the recent past but encountered failures, resulting in very high penalties. Invalid actions are placed in a separate category so they are not tested again for some time (to avoid penalties). Their invalid status, however, will expire after a pre-defined timeout and they will again be placed into an ‘Unknown’ status for testing.

When using an $\varepsilon$-greedy strategy, the value of $\varepsilon$ is estimated as the ratio of the Unknown and Known actions. Intuitively, this allows for higher chances for exploration when a large number of unknown actions is available in comparison with the number of known actions. It creates an incentive for the node to ‘explore’ its
surroundings based on the assumption that it has ‘changed’ recently (which is inferred by the increased ratio in unknown messages).

### 6.5 Pseudo-Algorithm

The pseudo algorithm shown here describes the resource allocation procedure at each node using the softmax strategy for action selection. The $\varepsilon$-greedy strategy follows the same algorithm, with the only difference being in the “Build_prob_Dist” function.

The parameters for the algorithm at each node are:

- $dTCost\_list\_size$: the size of the circular queue maintained for cost updates. This value specifies how many update costs for each $dT$ will be kept for estimating the average cost for the task.
- $\lambda$: Cost factor for window size estimation when using the Boltzmann strategy for task selection. $\lambda$ multiplied by the minimum cost amongst all tasks will provide the maximum acceptable cost for tasks to be included for building the probability distribution. The default value of $\lambda$ is 10.
- $\tau$: Temperature value for the Boltzmann distribution.
- $MissCost$: Maximum cost associated with failing to complete a task.

**Node\_Main\_Code**

*Initialize empty DTC table*

WHILE running

*Wait for Messages (in promiscuous mode)*

FOR each Message ‘Msg’

IF Msg was ‘overheard’ promiscuously THEN

Update DTC table $\leftarrow (Msg\.senderDT, msg\.senderDTCost)$

ELSE

Update DTC table $\leftarrow (Msg\.senderDT, msg\.senderDTCost)$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
IF Msg is of Type Data-Message
    CALL Handle_Data_Message (Msg)
ENDIF
ENDIF
ENDFOR
END WHILE
Terminate

The DTC table maintains a running average of costs and reliability per task. The update of the table with at tuple \( (dT, Cost(dT)) \) adjusts the average reported cost for task \( dT \) in the DTC table. The table maintains the average of the last \( dTCost\_list\_size \) entries.

**Handle_Data_Message (Message)**

IF Message.\( dT \) is terminal
    Consume Message
    Report (via broadcast) Cost for Consuming the Message
Else
    Build list (H) of current neighbors
    Remove from H the nodes list in Message.\( dT \).phist (H' = H \setminus Message.\( dT \).phist)
    \( SA \leftarrow \text{Build List of Possible Actions} \ (\text{Message}.dT, H) \)
    Defined a vector of costs \( \overrightarrow{\text{Cost}} \) (initially empty) for all actions
    FOR each Action 'a' in \( SA \)
        \( \text{Cost}(a) \leftarrow \text{Estimate cost from Info retrieved from DTC (if available)} \) (eq.9)
    ENDFOR
    \( \overrightarrow{p} \leftarrow \text{CALL Build\_prob\_dist} \ (SA, \overrightarrow{\text{Cost}}) \).
    Choose action (\( a_g \)) based on \( \overrightarrow{p} \)
    FOR each sub-task \( dT' \) defined in (a)
        Create a new Data Message \( Msg2 \)
        \( Msg2.\text{senderdT} = Msg.dT \)
        \( Msg2.\text{senderdTCost} = \text{Cost}(a_g) \)
        \( Msg2dT = dT' \)
        \( Msg2.phist = Msg.phist + \text{nodeName} \)
        Send \( Msg2 \) to \( f_n(dT') \)
        IF transmission succeeds
            Update DTC Table to increase reliability of \( dT' \)
        ELSE
            Update DTC Table to decrease reliability of \( dT' \)
            Announce \( Msg.\text{senderdT} \) has failed with 'MissCost'

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Build_Prob_Dist – Using Boltzmann (SA, Cost)

Remove from SA, actions with cost outside tolerance (window) of the minimum cost.

FOR each action ‘a_i’ in Cost,

Estimate the probability \( p(a_i) \) as: \( p(a_i) = \frac{e^{Cost(a_i)/\tau}}{\sum_{a_j} e^{Cost(a_j)/\tau}} \)

Adjust the \( p(a_i)' = (1 - \gamma) p(a_i) + \gamma / |SA| \)

Return the prob. dist vector \( p \)

Reducing the size of SA, the list of possible actions, based on a window around the minimum action cost is a heuristic solution for the problem of a non-converging probability distribution. This condition is recognized by the presence of large costs differences between candidate actions.

This behavior is very repeatable in networks where there’s a sudden failure of a critical node in the data path (as it will be later illustrated during the experiments), or when nodes present intermittent connectivity behavior.
Chapter 7. Experimental Evaluation

The proposed algorithm was implemented and tested on a simulated network for several network topologies and data flows. In this chapter we will describe the simulation environment and the parameters used by the algorithm and the simulations.

The purpose of these tests was to verify the behavior of the algorithm, rather than benchmark its performance given that the current implementation is a not optimized and intended as a proof of concept version of the algorithm.

In the tests shown here, changes in topology were limited to variations in processing cost factors as well as removal and addition of nodes involved in a data distribution task. Although these are preliminary tests, the experiments give an indication that the algorithm follows the expected behavior for the types of conditions to which it is submitted. Other simulations using mobility models and multiple concurrent data flows are proposed as future work.

7.1 The Simulation Environment

The simulation environment was developed to provide a controlled communications infrastructure between (simulated) network nodes. The simulation was completely written in Java™ and all tests were conducted on a dual processor 3GHz desktop running Windows™, using the standard Sun™ Java VM and standard memory configurations.
The simulation was designed to support multiple nodes as separate processes in one or more computers. The proposed algorithm disregards message fragmentation issues, that is, the assumption is that every message can be fully transmitted through a single network packet, or equivalently, that fragments of a single message always follow the strict same path for data distribution and the effects of partial message retransmission can be disregarded. This is a fair assumption especially for sensor network types of environments where sensor updates are usually smaller than the network’s Maximum Transmission Unit (MTU) configurations.

Rather than using an event driven simulation, each node was modeled as an independent system thread interacting with other threads (or nodes) only through messaging. This design allows for the occurrence of message synchronization issues and partial network information (due to simulated connectivity failure). These are common issues in wireless networks in general and can be controlled in the test environment through initialization arguments (or, in most cases, run-time settings) that define reliability of links, and overall network failure rate.

Links in the simulated environment are assumed to be bi-directional. Although wireless networks are rarely symmetric in general, the assumption of symmetry can be enforced by standard MAC-layer protocols where a node A will only accept packets at the MAC level, from nodes that are known to receive packets from A.

A naïve way to implement such protocols is, for instance, to have each node periodically broadcast hello messages with the list of nodes whose hello messages have been received. A node that finds itself on a peer’s hello message can assume the link to its peer is bi-directional and allow communications through it. If a message is received
where the node is not included in the list of peers, it is silently disregarded. The actual protocol used for building a symmetric link network is not relevant to this work. The assumption is that as bi-directional network topology can be efficiently built by underlying link protocols.

In the simulated environment, messages between nodes are exchanged through a common data link layer that also supports the duplication and random collision or message loss. By controlling the message distribution in this way, the variations of each metric can be studied in isolation and the experimental conditions can be carefully repeated without having to deal with often intractable and non-reproducible RF problems in live networks.

Figure 26, shows the main components designed for the simulation of the algorithm. The Data-Link emulator essentially provides the same services normally provided by the data link layer in conventional networks, that is, the coordination of medium access control and the distribution of messages. The network topology is described as a graph structure, accessible only to the DataLink Emulator (and initialization components).

![Figure 26. Simulation environment](image-url)
The number of hosts in the figure is essentially a function of the scale of the simulation to be performed. Network topologies and mobility models can be dynamically generated through a set of attributes specified in each simulation scenario, such as scale, density, link reliability distribution, etc. The scenarios can also specify a fixed topology and schedule (Event Scheduler) events that will occur at given times during the simulation. These events can include, for instance, the removal/addition of a node or links, as well as changes in processing and transmission costs.

The two main data structures are the NET (network topology) and the DST (data transition graph). The NET is maintained by the DLink Emulator, while the DST is statically available to any component (including the simulated nodes) in the network. That allows a node to obtain, at any time, a complete description of the data transition graph for any given data type.

Messages transferred through the DLink Emulator will be constrained by the topology of the graph structure (NET). Any changes in the shared graph structure (NET) will be immediately enforced by the DLink Emulator, resulting in changes in communications constraints.

Packet drops are defined as arguments to the emulator (and can also be changed at run-time) to specify the simulated network conditions. Based on the provided probability of dropping packets, the emulator will uniformly choose packets to be discarded.

The probability of dropping packets is specified either for the network as a whole or in a link basis, allowing the simulation to support different ‘qualities’ (in terms of reliability) of communication channels in the network.
7.2 Simulating Data Tasks

As part of the simulation, a stream of data tasks is created with pre-specified frequency and duration. The stream assigns to a node multiple copies of a data request, one at a time, at the specified frequency for the specified duration. The nodes will process data tasks one at a time and will queue pending tasks for later processing.

Multiple streams for the same (or different) data types can be created and executed in parallel over the network. The ‘data streams’ are created by the simulation environment at startup and assigned to the ‘Runner’ thread, which will synchronize the flow of messages with the simulated time of the environment.

A Data State Transition graph (DST) must have been initialized before streams become available to the simulation environment. The default configuration used in most of the experiments so far used a frequency of 15 messages per second for duration of 120 seconds, and different data requests.

Each data task will go through the network and terminate at some node (or maybe be dropped due to a network failure).

If the data task terminates successfully, that is, it reaches the intended target node as a terminal task, the node will log the event reporting the timestamp when the message was received, its accumulated cost and other additional information used for analysis that are not required by the protocol such as processing path, sequence of operations, etc.

There’s always a single data task associated with any message and each task retains its ID if duplicated or decomposed into sub-tasks. A task with ID number 10, for instance, intended to target nodes n1 and n2 will be reported by both nodes upon arrival.
as completed. Each node will report the costs received by its message and both events are correlated by the task ID, for later analysis.

If a node is unable to handle a data task, the task will be dropped and the node will announce the failure of the task (through a broadcast message). Task completion notifications are only done by target nodes, that is, nodes listed in the messages data task target-pair list. Task failures can be reported by any node in the network.

The only exception to this rule is when a message is dropped due to network failure. The sender will know about the failure immediately after sending the message and will update its perceived reliability of that specific communications link for handling that specific task. An announcement of the failure will be issued and the node will proceed to handling another action.

7.2.1 Simulation Parameters

There are several simulation parameters used to control different aspects of the behavior of the network and the algorithm. The main configuration parameters and their default values are defined below.

*Network Error Rate*: The overall network error rate defines the percentage of packets that will be dropped by the Data Link Emulator in any connection. This packet drop rate is in addition to the packet drops caused by the reliability of the link. The default value for the error rate is 0.005.

*Average window size*: This is the size of the running windows that will store cost information for policies. The average cost is calculated as a simple average from these values but minor variations have also been tested using a linearly discounted average.
The default value is 300. Lower values will increase the reactiveness of the algorithm and higher values will favor its stability.

*MissCost (Maximum Task Cost):* This is a parameter that defines the maximum cost allowed to a task. The maximum task cost is used to report a failure on handling a data task and provides a cap for total accumulated costs for a task. The default value is 1000 000.0. This is an arbitrary number and must be sufficiently higher than all expected costs in the system.

*Greedy-Explore-Known/Unknown Ratio:* When using an $\epsilon$-greedy strategy for action selection, the choice of a non-greedy choice is not from a uniform distribution of all tasks. Because we can differentiate between non-greedy actions and separate those that have been explored recently from the actions that have not been tried, the greedy search will prefer to explore one set versus the other based on this factor. The default setting is .99, meaning that 99% of the time when opting for a non-greedy action, the algorithm will pick an unknown action.

*Action-Cost-Search-Window:* In this work, we proposed a heuristic approach to control the convergence problem that can occur for certain cost distributions. This parameter specifies the multiplication factor that will be used for the lowest cost action to select candidate actions to build the Boltzmann distribution. For instance, if this factor is set to 5, only actions with a cost lower or equal to five times the minimum available cost will be considered for the distribution. The default value for this parameter is 10.

*Minimum Link Reliability:* Each node builds statistics of the success rate (i.e. communications success rate) per action. This rate provides an estimate of reliability for the node’s communication link (for that specific action) and will be used on weighting.
the costs. If a the communications success rate for a specific action goes below this parameter, the action is marked as invalid and will be disregarded for a period of time.

*Action-Success-Rate Timeout:* This parameter specifies when to timeout actions that have been previously considered invalid due to high communication failure rates. After the timeout, these actions are eligible candidates again.

*Action-Cost-Info-Timeout:* Action cost information is updated every time a node listens (promiscuously) to a data or control message from a neighbor. The cost for each action is maintained and used for cost estimation when a task decision has to be made. This parameter specifies a timeout for stale information about costs for specific action that might no longer be valid.

### 7.3 Experimental Results

In order to demonstrated and discuss the algorithm, a few scenarios were defined and simulated. In this item, we present some of our simulation results for different topologies and topology-change conditions.

#### 7.3.1 The 4-Node Fixed Topology

The first simulation test is a fixed network with four nodes and a three-state data transition graph. Both the network topology (NET) and the DST diagram are shown in Figure 27. The figure also shows all costs associated with data processing (\(pfactors, function costs\)) and transmission (\(tfactors, datum size\)).
One single data stream was created for this test. As defined in 7.2, the stream will send 10 tasks \((dT = n_1 d_1 \# n_3 d_2, n_4 d_2 \#)\) per second to the system (i.e. to node \(n_1\)) for a period of 120 seconds. The target nodes (target pair-set) for the task are nodes \(n_3\) (datum \(d_2\)) and \(n_4\) (datum \(d_2\)), with no processing capabilities (no functions available) as illustrated in Figure 27.

Nodes \(n_3\) and \(n_4\) will report (log) all successful tasks and their costs. A successful task, from a node’s perspective, consists of having the specified datum delivered at the target node (i.e. receiving a terminal task). If both nodes report that the same task was completed (which can be matched by a task ID), then the overall task \((dT = n_1 d_1 \# n_3 d_2, n_4 d_2 \#)\) is said to have succeeded and the cost of completion is the sum of the costs from both \(n_3\) and \(n_4\).
Figure 28 shows the average aggregate cost results from four independent runs for each of the policy selection strategies. Packet drops for each strategy are also shown in the figure.

In this example, the epsilon coefficient for the $\epsilon$-greedy strategy is calculated at each iteration as the ratio of actions flagged as 'unknown' and the total number of strategies available. Initially, with several unknown actions available the algorithm using an $\epsilon$-greedy strategy will be more proactive on searching for new possibilities.

Out of 300 data tasks, the $\epsilon$-greedy strategy lost an average of 28 tasks while Boltzmann lost nine tasks on average. Note that most of the loss in the $\epsilon$-greedy occurs at the beginning of the process, while unknown actions are being explored.

As actions are checked and cost estimates built $\epsilon$-greedy settles for a fixed (although not necessarily best) action. In this configuration the algorithm will only try the
actions again when their cost information (obtained the last time they were checked) becomes stale and the actions are moved again to an unknown state (that is not visible in Figure 28 as it would have happened later in ‘time’).

To illustrate the differences between the heuristic variable epsilon chosen at run time versus the conventional fixed epsilon strategies normally used in the literature, Figure 29 shows the results obtained from the same test case using fixed $\varepsilon$-greedy strategies (both at 1% and 10%) superimposed with the previous results.

Note that, in terms of $\varepsilon$-greedy strategies, these results are biased towards a fixed topology. If the topology changes very frequently, the variable epsilon would have to rely on a very short timeout for expiring costs (approximating itself to the fixed epsilon

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The average task loss for the 1% strategy was 39, while the average for the 10% strategy was 73 tasks out of the 300.

In most of the tests, the final (or stable) stream solution was the one illustrated in Figure 30.

![Diagram showing the final data flow configuration for the 4-Node fixed problem.]

\[ dT = n_1 d_1 \# n_3 d_2, n_4 d_2 \# \]

Figure 30 Final data flow configuration for the 4-Node fixed problem.

7.3.2 The 4-Node Changing Topology

A more interesting scenario is when a change in topology occurs during the flow of a data stream. This example is shown in Figure 31, where the four node topology is used for the same scenario as the previous example.

In this case, however, node \( n_2 \) is initially removed from the topology and added at a later time. Figure 31 presents the average of four tests showing how the algorithms reacted to the insertion of the node in the topology. The time of the insertion was approximately the same in each run (dotted zone in the figure). In this example, both algorithms quickly detect the new neighbor and extend their set of available actions and find a better solution for the DDT.
The $\varepsilon$-greedy implementation, as expected fluctuates a little longer until it explores the new set of known solutions. Because the probability of exploration drops as new solutions become 'known' (that is, recently tested), the curve stabilizes with the Boltzmann implementation after a while.

The costs, in both cases is the minimum cost available to in the network, the fact that the Boltzmann line at some point goes below that value (right after the addition of the node) indicates packets being dropped in some of the runs, resulting in a lower average cost.

![Figure 31. Four-node network with changing topology](image-url)
7.3.3 The 7-Node Changing Topology

Another test involving the addition and removal of nodes is shown in Figure 32 for a 7-node topology and associated data task. The data stream in this example is given by $dT = n_1d_1 \# n_4d_2, n_3d_2 \#$, injected in $n_i$ at a rate of 20 requests per second for 120 seconds.

In this simulation, the minimum cost path is through node $n_3$, which performs a transformation and then splits the stream to $n_4$ and $n_3$.

The resulting aggregate costs for the algorithm using both the Boltzmann and the $\varepsilon$-greedy strategies are shown in Figure 33.

![Figure 32. 7-node network with changing topology](image)
Figure 33. A 7-node network changing topology

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
At 20 requests per second, the horizontal axis in the figure provides an indirect measurement of 'time'. In this example, 30 seconds into the simulation, node $n_3$ is removed from the network (shown at the bottom of Figure 33) and the processing of the task is switched to flow through nodes $n_2$ and $n_6$. Coincidentally in this case, this is also the shortest communication path for the flow, but in other simulations (not shown in this specific figure) if the processing cost of node $n_6$ increases, the transformation tasks starts happening at node $n_2$, as opposed to $n_6$. The effects of the removal of node $n_3$ are shown in detail in Figure 34.

![Total Cost Boltzman vs e-greedy](image)

**Figure 34.** Removing node $n_3$ from the 7-node topology

Node 3 is added back again into the network at time 60 (around 1200 in the x-scale) with a different set of edge costs. Note that both strategies move the processing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
flow back again to node $n_3$. Figure 35 shows in detail the recovery to a lower cost configuration after node $n_3$ is re-added to the network.

![Figure 35. Re-inserting node n3 from the 7-node topology](image)

### 7.3.4 An Example of Unstable Equilibrium

The Boltzmann distribution is proven, in the long run, to yield the maximum ‘reward’ for the sequence of plays. That is, it minimizes the regret between the rewards obtained by the sequence of plays and the maximum rewards available if the game were played using the best option. There are no guarantees (in fact no expectations) of a stable solution (i.e. finding the one ‘best’ lever in the k-arm bandit problem and playing it forever).

During the simulation in dynamic environments this behavior emerged as a potential problem for the stabilization of the system. The issue occurs when the set of
The best costs available at a node are relatively close to each other but other, supposedly less efficient, options are significantly higher (in terms of cost). The problem is that, even on an exponential distribution, the differences between the options that are relatively close to each other will become too small (compared with the complete set of options) to affect the probability distribution for task selection.

The test case was the following: Consider the 4-node network illustrated in Figure 27, where node $n_i$ is given a stream of data tasks such as $(dT = n_i d_1 # n_i d_2, n_i d_2 #)$. The only difference in this case is that nodes $n_3$ and $n_4$ have processing capabilities for data transformation.

![Total Cost Boltzman - Unstable](image)

**Figure 36. An unstable condition with the Boltzmann strategy**

At time 30, node $n_2$ is removed from the network (this is the same simulation shown in the first example so far) and the nodes are forced to find a new solution. In
Figure 36 the Boltzmann strategy for the problem is shown around the time when the node is removed.

Note that the recovery into the new state moves the solution into an unstable equilibrium. The best choice oscillates between four very close options. In this example, the four actions are:

\[
\begin{align*}
    a\left[\Delta T = n_1d_1 \# n_3d_2, n_4d_2 \#\right] &= (n_3d_1 \# n_3d_2 \# n_1)(n_4d_1 \# n_4d_2 \# n_1) \quad \text{Cost: 100} \\
    a\left[\Delta T = n_1d_1 \# n_3d_2, n_4d_2 \#\right] &= (n_3d_2 \# n_3d_2 \# n_1)(n_4d_1 \# n_4d_2 \# n_1) \quad \text{Cost: 185} \\
    a\left[\Delta T = n_1d_1 \# n_3d_2, n_4d_2 \#\right] &= (n_3d_2 \# n_3d_2 \# n_1)(n_4d_1 \# n_4d_2 \# n_1) \quad \text{Cost: 205} \\
    a\left[\Delta T = n_1d_1 \# n_3d_2, n_4d_2 \#\right] &= (n_3d_2 \# n_3d_2 \# n_1)(n_4d_2 \# n_4d_2 \# n_1) \quad \text{Cost: 170}
\end{align*}
\]

Although there is a best cost (in fact a significantly lower one) amongst these four options, the other two remaining actions have a very, but still valid high. The actions would be to send the request to node four through node three and vice versa. The costs associated with these actions would be in the range of 50000 and 66300.

\[
\begin{align*}
    a\left[\Delta T = n_1d_1 \# n_3d_2, n_4d_2 \#\right] &= (n_3d_2 \# n_3d_2 \# n_1)(n_4d_2 \# n_4d_2 \# n_1) \quad \text{Cost: 50055} \\
    a\left[\Delta T = n_1d_1 \# n_3d_2, n_4d_2 \#\right] &= (n_4d_1 \# n_4d_2 \# n_1) \quad \text{Cost: 50655}
\end{align*}
\]

The large differences in cost simply prevent the distribution from creating any noticeable difference between the previous four options. The probability distribution of the expected return for each action is shown in Figure 37.
This condition can be seen if the node is going up/down or if the non-broken remaining actions are relatively close to each other – this is a problem that can be easily reproduced in the k-arm bandit simulation by having some of the arms (the best ones) returning very high rewards in comparison with the lowest ones. In that case the probability distribution cannot easily differentiate the best option.

One way we found to address this problem was to create a sliding window over the smallest costs in order to eliminate the discrepancies. For instance, consider a multiplication factor of six (which is also a parameter to algorithm) and consider for the distribution only actions with estimated cost below 6 times the minimum cost available.

Such an approach reduces the number of policies being considered at each run and avoids high discrepancies in the values, allowing for more stable solutions. If the cost
of the lowest action increases, other previously disregarded choices will be reconsidered.

As cost values expire with time, the very expensive solutions will eventually fall back into the reconsideration range and tested again.

This procedure was added to our policy selection algorithm and the results are shown in Figure 38.

Figure 38. The unstable condition corrected with the sliding window
Chapter 8. Conclusions

The main contribution of this work was the design and proof-of-concept implementation of a novel approach for the resource allocation problem in tactical networks. In our approach, we leverage form pervious research on reinforcement learning and game theory to design a distributed algorithm for resource allocation that simultaneously addresses the requirements for resource constrained in-stream data processing in mobile ad hoc networks, an important capability for tactical environments.

The algorithm introduced in this work proposes the simultaneous allocation of resources for both data processing and data transmission in the network. This combined approach to the resource allocation problem differs from techniques commonly found in the literature that are: a) centered at the application (or middleware levels) and rely on models and estimates of the underlying communications framework for resource allocation or, b) data-agnostic techniques centered at the network level that attempt to infer data processing requirements to better allocate routes.

In this work, the resource allocation problem was first defined and formally described in the context of an online learning in order to be mapped, under certain assumptions, to the well known k-arm bandit problem at the node level. From that formulation, two standard strategies for policy selection (ε-greedy and SoftMax) were slightly modified and integrated with the algorithm.
The resource allocation algorithm proposed in work is reactive, efficient, flexible and adaptive. The algorithm is reactive in the sense it does not proactively maintain any state about the network and resource availability. The discovery and allocation of resources is done on demand, as tasks become available.

The proposed algorithm is efficient in the sense that is uses virtually no control or coordination messages to exchange states between nodes. Cost information is exchanged as part of the data messages (with the addition of a small payload for cost notification). Nodes obtain resource availability information mostly through promiscuous passive listening. The only two types of messages required by the algorithm are the notification of failure and the notification of task completion. The notification of task completion can very easily be integrated as part of the MAC response message, with minimum overhead.

The flexibility of the algorithm is the generalized description of a data task and data fragmentation procedures. That allows for virtually arbitrary types of data to be handled by the framework, as long as their state transitions can be described by a DST graph.

The proposed algorithm is adaptive self-healing, in the sense that local losses in the DDT are locally addressed by neighbor nodes that will immediately start searching for alternative resource assignments.

The propagation of local failures will move upstream in the DDT as a function of the size of the average queues used to store cost information. That is a parameter to the framework and an indicator of its reactiveness to changes in the environment. In any case, the process of link failure propagation (as an added cost for the task) occurs in parallel (but at a slower rate) to the search for alternative configurations. That gives an
opportunity for the framework to locally fixed failures before triggering significant change in the complete DDT. This is a very important feature for tactical environments where localized failures in large distribution trees are expected to occur.

A sliding window algorithm was also proposed, implemented and tested as a mechanism for stabilizing the solution of action selection strategies with high cost variances. Stability is an important feature in wireless communications networks because they constrain the usage of the RF area into a smaller spatial region, reducing unnecessary interference.

The experimental results, although preliminary at this stage, have provided some evidence that the algorithm is capable of adapting to changes in both environment and task requirements. Further tests are necessary to validate the algorithm in large scale settings and benchmarking.
Appendix A. Mobile Ad hoc Networks

1. Wireless Networks

Wireless networks use electromagnetic waves for transmitting data between nodes. They enable low cost network connectivity between machines while still supporting node mobility and arbitrary topologies.

In general wireless networks are classified in terms of their application scenarios in three main groups: a) Wireless Personal Area Networks (WPAN), b) Wireless Local Area Networks (WLAN) and c) Wireless Wide Area Networks (WWAN).

WPAN environments are mostly focused the use of networked devices for personal use such as PDA’s, wireless headsets and their interactions with personal computers and cell phones. WLAN primarily describes what is sometimes referred to as Wireless Ethernet, or local shared medium networks that enable node connectivity while still maintaining mobility, and WWAN are mostly associated with larger scale wireless networks that usually rely on some type of hybrid infrastructure for data relay. A classical example is the cellular phone networks.

The focus of this work will be on the special class of applications defined by Wireless Local Area Networks. As in wired LANs, WLANs define a common medium for data communications, shared by all connected nodes. Data exchanged by any two nodes in a LAN is injected to the medium by the sender and will be picked up by the
receiver node. All other nodes connected to the medium will have direct access to the
data but will silently ignore the transmission as they are not the intended receiver.

The obvious drawback of local area networks is essentially the fact that because
nodes are sharing a common medium for data exchange, there’s always a potential data
collision, which results in the ‘destruction’ of both colliding packets and communications
failure.

Early wireless protocols inherited some of the techniques often used in wired
environments to coordinate medium access. The ALOHA protocol, for instance,
proposed by Abramson (1973) relied on explicit acknowledgement packets to detect
collision. Nodes were allowed to transmit freely and wait for acknowledgement to arrive
within an appropriate time out. If acknowledgements were not received the nodes would
assume that a collision occurred and would choose a random interval (random
retransmission delay) to retransmit the packet. A variation of the ALOHA protocol called
slotted-ALOHA (Kleinrock and Lam, 1973) introduced the concept of slotting the time
into segments and enforcing that every node would only start transmitting at the
beginning of each time slot. The idea was to reduce the number of collisions by
eliminating the collision from partial transmission overlapping. That is, if to nodes collide
when transmitting they would collide at the beginning of their time slot mitigating the
effects for the subsequent time slot transmission.

Seeking to reduce the high number of retransmission in the ALOHA protocols, a
proposal was introduced in 1975 (Kleinrock, L. and Tobagi, F., 1975) for a carrier sense
multiple access (CSMA) protocol. In CSMA, each node would essentially observe
(listen) to the data medium for any traffic before starting the transmission. The idea was
to attempt to avoid collision rather than simply detecting it. In its so called nonpersistent implementation, nodes would transmit data immediately if no traffic was detected in the channel and, if traffic was detected, the node would essentially schedule the packet transmission for a later time. A slotted version of the protocol was also proposed, following the same notion of data transmission only at fixed time slots.

The common assumption however, of sensing channel activity to avoid collision, although valid for wired mediums, was not directly mapped to the wireless environment. One well known problem, for instance, is the issue known as the hidden node problem. The hidden node problem occurs when an active node is close enough to the target node (destination of the packet) to cause interference, but far enough of the source node (sender) to be detected by sensing the channel. This phenomenon (Figure 39), very common on wireless networks, essentially invalidates the carrier sensing techniques used by some protocols.

![Figure 39. The hidden node problem](image)

In (Figure 39), the transmission between nodes ‘C’ and ‘D’ will cause interference with node ‘B’ but will be undetected by node ‘A’, who will perceive the channel as available (or clear) for transmission. The hidden node problem doesn’t occur on wired...
networks because there’s no partial connectivity in wired LANs, any transmission made by any node can be detected by every other node connected to the LAN.

In 1997 the IEEE released the first standard for wireless LANs known as the IEEE 802.11 (Crow et. al., 1997). After its original release the standard was revised and extended a few times, evolving to be currently the most widely used protocol suite for wireless LANs. Furthermore, 802.11 is likely to continue to be the de facto standard for wireless LANs as novel competing technologies (such as the European HiperLan2 and the HomeRF technology) usually have their best features quickly incorporated in revised versions of the 802.11 protocols, and have difficulty competing.

The 1999 edition of the IEEE 802.11 standard defined the concept of a Basic Service Set (BSS) as the basic unit in 802.11 networks. The minimum BSS consists of two stations, that is, two addressable wireless nodes. A BSS might have one of the nodes designated as a base station or not. A base station is essentially a special node in the BSS that controls medium access to all other nodes, that is, a node that coordinates which nodes are allowed to transmit at any given time. If a BSS has a base node, the BSS is said to be in infrastructure mode.

Alternatively, a BSS that has no base node is called an Independent Basic Service Set and it said to be in ad hoc mode. Both types of networks are shown in Figure 40.
Figure 40 also shows how multiple infrastructure BSS can be combined through wired distribution systems to build Extended Service Sets (ESS) of multiple BSS cells. In such configurations, the base station of each BSS is referred to as an access point (AP).

The 802.11 standard provides two mechanisms for medium access control (MAC), the distributed coordination function (DCF), often used in ad hoc or infrastructure modes, and the point coordination function (PCF), often used in infrastructure mode. Both coexist at the MAC layer and operate concurrently.

The Point Coordination Function (PCF) method relies on a point coordinator node (PC) that is responsible for allocating and coordinating media access for all nodes associated with the PC. Through a pooling mechanism, the PC will essentially notify each node when to transmit and for how long, minimizing the chances for collision. The PCF mechanism is primarily utilized in infrastructure mode where the base station plays the role of the point of coordination.

---

3 Image source: Crow et. al., 1997.
The basic medium access protocol is a DCF method known as the *carrier sense multiple-access with collision avoidance* (CSMA/CA). The distributed CSMA/CA directly addresses the hidden node problem by using a *virtual carrier-sense mechanism* to detect channel availability and it can be used both in infrastructure or ad hoc (or IBSS) modes.

In this method, when a node (A) is ready to transmit a packet to a target node (B), it will sense the channel for an opportunity to send a RTS (Ready-to-Send) control packet to the destination. The sender will wait for a CTS (Clear-to-Send) reply from the destination node before transmitting. Before replying with the CTS packet, the target node will also sense the channel for activity and wait for an opportunity to reply. The RTS/CTS mechanism allows both the sender and receiver nodes to detect interference, mitigating the hidden node problem and reducing packet collision.

CSMA/CA also utilizes a random backoff time following a busy medium condition. That is, every time a node senses the medium for activity, if a busy state is found, a random backoff time is waited before a new retransmission attempt from that node. Furthermore, the protocol also specifies that all directed traffic (i.e. non-broadcast traffic) should receive an immediate acknowledgement (ACK frame) from the destination node.

The PCF method is a content-free method for and DCF can be used concurrently in infrastructure based mode, where nodes essentially take turns on each of the medium access protocols.
2. Mobile Ad hoc Networks

Mobile ad hoc networks constitute a special case of conventional ad hoc network. In mobile ad hoc networks nodes are physically mobile and free to arbitrarily join or leave the network. It is important to clarify that by definition ad hoc networks imply no fixed infrastructure and flexible network topology. In general, any portable mobile node can arbitrarily join or leave an ad hoc network however, in conventional ad hoc networks nodes are often fixed while in use and idle while moving. For instance, in a wireless office environment operating in ad hoc mode, meetings and presentations are quickly assembled, on demand, only for the duration of the event. It unlikely though, that during the presentation, the machine sending slides to the projects will be moving through the building continuously changing its connectivity with other nodes and the projectors.

On the other hand, in Mobile Ad hoc Environments, the expectation is that mobile is concurrent to data transmission. A fleet of monitoring unmanned vehicles building a joint map of a terrain will be continuously exchanging video and LIDAR data while at the same time moving in relation to one another. MANETs are particularly challenging because they require fast adaptations to change in topology and resources, with no (or minimum) disruption of data service.

Tactical environments are primarily Mobile Ad hoc Environments, with further constraints on joint resource utilization and policies. The challenges imposed by MANET environments include all the challenges introduced for ad hoc network, plus a few more that will be introduced and discussed in later chapters.

There are two important characteristics of these types of networks that are worth highlight and explain further.
2.1. Packet Acknowledgement

All directed traffic is acknowledged, that is, both the FDC and DRC methods account for individual packet acknowledgement at the MAC level.

After sending a packet, the sender will wait for a pre-defined interval for the ACK packet. If the packet is received the node assumes the transmission was successful and it will assume transmission failure if there’s no acknowledgement from the peer.

An important point here is that acknowledgement occurs at the MAC level, so it is valid to assume that this information is available (or can be made available) to upper layers in the communications stack.

Broadcast and Multicast messages are not acknowledged, but all unicast messages are acknowledged ‘immediately’ (after a relatively short timeout, in the worst case) to the sender. That capability provides direct feedback in terms of success or failure to the sender and will be utilized by the proposed protocol.

2.2. Promiscuous Listening

Within communications range, every packet transmitted by a node is, at the MAC level, broadcasted to all nodes. This is essentially how any local area network operates. Nodes are expected to filter packets at that level and accept only packets that are directly addressed to the node, broadcast packets or multicast packets.

However, nodes are allowed to promiscuously listen to the network traffic to gain access to all data packets circulating the medium. This is illustrated in (Figure 41) where node (A) sends a unicast packet to node (B), which is also ‘seen’ in promiscuous mode by node (C).
Note that node (D) is within communications range of node (B) but out of node’s (A) range, so it is unable to see the data packet sent from (A) to (B). Promiscuous listening is possible for nodes within the sender’s communications range.

3. Resource Allocation for Data Streaming in MANET

There are data streaming protocols however for our purposes these will be treated as sequence of individual messages. The main point to be made is that we can afford to loose a few frames as long as we maintain the minimum levels of service. Let’s define a few metrics that will be used as parameters for evaluation.

Bandwidth utilization: In the strict sense of the word bandwidth essentially refers to the width of the frequency band used by the carrier waves. In a more practical light, however, the bandwidth can be directly translated to the number of bytes (or bits) / second.
List of References


Curran, E., “Swarm: Cooperative reinforcement learning for routing in ad hoc networks,”
Duran-Limon, et.al “Context-Aware Middleware for Pervasive Ad Hoc Environments”,
Dowling J., Curran, E., Cunningham R., and Cahill, V., Using Feedback in Collaborative
Reinforcement Learning to Adaptively Optimize MANET Routing. IEEE
Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans,
of the 16th ACM Conference on Object-Oriented Programming Systems,
Freund, Y. and Schapire, R. A decision-theoretic generalization of on-line learning and
an application to boosting. Journal of Computer and System Sciences,
Galstyan, A., Czajkowski, K. and Lerman, K. “Resource Allocation in Grid Using
Reinforcement Learning”, International Conference on Autonomous Agents and
Multiagent Systems, 2004
Garcia, P., Zsigri, A., Guitton, A., A Multicast Reinforcement Learning Algorithm for
WDM Optical Networks. 7th International Conference on Telecommunications –
and robust communication paradigm for sensor networks. In Proceedings of the


Peng, F. and Deyun, Z. - Hybrid Optimize Strategy based QoS Route Algorithm for Mobile Ad hoc Networks - Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China.


Biography

Marco Carvalho was born in Rio de Janeiro, Brazil. He graduated as a Mechanical Engineer from the Federal University of Brasilia (UnB), in Brazil. In 1996, he received a Masters degree in Mechanical Engineering with specialization in dynamic systems from the same University.

While in Brazil, his research was primarily focused in experimental and computational fluid dynamics, with a special interest in fluid-structure interactions and dynamic calibration of high frequency pressure sensors. This research was primarily sponsored by the National Council for Scientific and Technological Development (CNPq), under its RHAE (Human Resources for Strategic Areas) program.

After moving to the United States, Marco Carvalho has worked extensively in the areas of large scale data network design and maintenance. He completed his Masters degree in Computer Science at the University of West Florida and is a Research Associate at the Institute for Human and Machine Cognition (IHMC) since 2000.

His research at IHMC is primarily focused in network security and several aspects of mobile ad hoc networks for tactical environments, primarily for military applications. He has published several conference papers and two book chapters in the area.