# SERVICE DISCOVERY IN TRANSIENT AD-HOC WIRELESS NETWORKS

THÈSE N° 2481 (2001)

PRÉSENTÉE AU DÉPARTEMENT DE SYSTÈMES DE COMMUNICATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUES

PAR

Michael NIDD

BMath, MMath, University of Waterloo, Canada
de nationalité canadienne

Accepté sur proposition du jury:

Prof. André Schiper, président
Prof. R. Molva, directeur de thèse
Prof. Ernst Biersack, corapporteur
Dr. Dirk Huseman, corapporteur
Prof. Jean-Yves Le Boudec, corapporteur
Dr. Harry Rudin, corapporteur

Lausanne, EPFL
2001

# Acknowledgements

Well, that was an adventure!

This thesis would not have been possible without the guidance of my supervisor, Dr. Refik Molva. His knowledge not only of what constitutes sufficient evidence of a claim, but also of navigating the labyrinth of rules for studies that spanned three universities, two countries, and a commercial lab was invaluable. He has my thanks, and my deep respect.

I would also like to thank the committee for their various comments and contributions. They allowed me to improve what I thought was the final draft to the version that you hold in your hands. Beyond his committee duties, Dr. Dirk Husemann both provided the initial project framework within which this thesis came about, and further contributed ideas, feedback, and advice through our many discussions.

The duration of my studies was funded and supported by the IBM Research Zurich laboratory. My stay with ZRL has been very enjoyable and informative. The support of my managers at the lab, and also of administrative staff both there and at Eurécom has been incredible, and has certainly earned my gratitude.

In addition to these people, the constant support of friends too numerous to mention was vital both for spotting errors and inconsistencies with the text, and also for spotting and curing many personal periods of stress. I would particularly like to thank Sonja Buchegger and Anthony Bussani for volunteering assistance above and beyond anything I could have asked.

Aside from direct contributions to the content, assistance in adapting to life in a new country was very much appreciated. The help and advice of friends like Robert Humbel, Klaus Kursawe, Cyriel Minkenberg, James Riordan, Peter Trommler, and Christian Rohner made the adjustment not only possible, but enjoyable.

Finally, but possibly most importantly, I am thankful for the solid support of my family. They may have wondered why I would move so far away, and probably questioned (to themselves) why I was going back to school, but they always stood behind me without complaint. My parents in particular have been able to offer encouragement and support all through my studies, and deserve thanks most of all.

Thank you all.

Sincerely,

Michael Nidd

# Abstract

Since the earliest days of electronic computing, the ratio of size to computational power has been going down. Retrospect shows that, around 1980, some threshold was crossed that launched a new industry based on desktop computing in the office. Today, the beginning of the new millennium is the background to a similar shift in the industry. Pocket-sized computing devices are now a fast growing market segment and, in the way that desk-top computing enabled completely new office applications, the portability has reached a level where completely new applications are now possible for computing technology. Furthermore, along with size, weight, and cost, a quality vital to the continued popularity of portable computing is the ability for portable devices to interact with other devices easily.

As the number of devices in use increases, the need for genuine network support also increases. Because the devices are so easily portable, networks must be designed to expect their topology to change frequently. Networks capable of reconfiguring themselves rapidly, and without user intervention, are generically referred to as ad-hoc networks.

In order to continue the reduction in both size and manufacturing costs of portable devices, inter-device cooperation is necessary. One example of this cooperation is to use an audio headset for both telephone audio I/O and PDA audio I/O. This means that a user who already owns and uses a portable audio I/O device can purchase a smaller, possibly cheaper PDA that does not include internal audio I/O capability. One important characteristic of this type of application is its spatial locality. The target platform does not involve packet forwarding, because the target applications use the physical locality that corresponds to logical (single-hop) locality as a cue to identify devices useful to the user.

Introduced in this thesis is a new method for discovering the services available in the immediate area of a portable device in an ad-hoc network. The new method pro-actively maintains a list of available services on each local device, resulting in faster response to queries, and better

tolerance of data transmission errors. The specific improvements offered through this algorithm separate it from other pro-active alternatives by offering faster responsiveness to the arrival of new devices.

After presenting a basic theoretical analysis of the behaviour that should be expected from the algorithm, measurements of its behaviour are shown for a simulated environment. The simulated behaviour is shown to agree not only with the results predicted by the preliminary analyses, but also with observed behaviour of an implementation tested on a real-time network emulation, and actual implementations on both 10baseT Ethernet and IEEE 802.11 wireless networks.

Having completed analysis of the basic algorithm, improvements are presented that offer power saving advantages for devices using this new algorithm. The advantages of these improvements are quantified, and their applicability is discussed. Finally, an outline of the actual service description and query language is also presented.

# Résumé

Depuis les premiers jours de l'informatique, le rapport entre la taille et la puissance des ordinateurs décroît. Des études rétrospectives montrent qu'une limite a été atteinte vers les années 1980, avec le lancement des ordinateurs personnels. Le commencement du nouveau millénaire semble être le point de départ d'un nouveau changement dans ce sens. Le marché des ordinateurs de poche croît très rapidement et, de même que les ordinateurs individuels ont rendu possible l'introduction des applications bureautiques, la portabilité a atteint un niveau suffisant pour permettre l'émergence de nouvelles applications. En plus des avantages en termes de taille, poids et coût qu'ils offrent, ces ordinateurs portables devront pouvoir interagir entre eux.

D'autre part l'augmentation du nombre de portables nécessite un support réseau approprié qui permette de fréquents changements de topologie. Le terme "réseaux ad hoc" désigne de tels réseaux qui sont capables de se reconfigurer automatiquement et rapidement.

La poursuite de la réduction de taille et de coût des portables nécessite que ceux-ci soient dotés d'une capacité de communiquer directement entre eux, afin de mieux partager les ressources réduites. Un exemple de coopération serait l'utilisation d'un casque audio en tant que source sonore d'un téléphone et d'un assistant électronique. Dans ce cas, l'utilisateur d'un tel dispositif portable d'écoute peut acheter un assistant plus petit et moins cher grâce à l'absence de la sortie audio. Une caractéristique importante de ce type d'application est la localité de la communication entre les dispositifs compatibles. Seules les communications locales à travers un lien direct étant justifiées dans ce contexte de proximité, l'architecture envisagée ne prend pas en compte l'acheminement des paquets.

Cette thèse introduit une nouvelle méthode pour la découverte de services dans l'entourage immédiat d'un dispositif portable au sein d'un réseau "ad-hoc". Cette nouvelle méthode maintient d'une façon proactive une liste des services disponibles au niveau de chaque dispositif, permettant ainsi une réponse plus rapide aux requêtes soumises et une meilleure tolérance aux

erreurs de transmission. Par rapport à d'autres algorithmes proactifs de découverte de services, l'algorithme proposé offre aussi une amélioration du temps de réponse lors de l'apparition de nouveaux dispositifs portables.

Après une présentation d'une analyse théorique du comportement que l'on peut attendre de l'algorithme, des mesures de ce comportement sont présentées pour un environnement simulé. La simulation a permis de confirmer aussi bien les résultats de l'analyse théorique que le comportement de l'algorithme observé à travers une émulation temps réel du réseau et une mise en oeuvre sur un réseau Ethernet 10baseT et un réseau sans fil IEEE 802.11 .

Des améliorations de l'algorithme qui permettent des économies d'énergie sont ensuite proposées. La présentation de ces adaptations est suivie de l'évaluation quantitative des améliorations et d'une discussion sur leur applicabilité. Les services et le langage de recherche actuellement utilisé sont décrits en dernier lieu.

# Contents

# List of Figures

# Chapter 1

# Introduction

As the number of portable computing platforms grows, so does the need for ad-hoc networking. Also called “zero-configuration” networks, these small, self-organising, short-lived networks enable a new level of collaboration between devices.

The need for consolidating interfaces has been addressed for the domestic consumer market already, with “universal remote controls.” Home entertainment systems (television, stereo, VCR, etc.) all use similar remote interfaces, and having separate controller devices for each unit is inconvenient. Similar consolidation is needed for the business market, where day timers, phone books, telephones, and so on are all duplicated between desk computers, PDAs, laptops, and cell phones. Fixed connection points, or “docks,” are inconvenient to install, awkward to move, and restrictive. The modern vision of “any time, anywhere computing” requires easy interconnection independent of the desktop.

Until recently, the advantage of wired docking sites was that each device immediately knew of the connection, and knew what protocol was appropriate because the relevant port was dedicated to a single type of device. With new personal area networking options like IEEE 802.11, HIPERLAN, and Bluetooth, this advantage has mostly disappeared; wireless connections will soon replace cable for many docking applications.

Cable replacement is the motivation for installing wireless communications equipment in many portable devices, but the added benefit is that this same equipment can enable new, previously unrealizable applications. Much like MIDI synthesizers, that are often sold without a keyboard, new consumer devices can now be sold with no built-in user interface. An early example of this is the Bluetooth wireless headset, that offers a hands-free connection to cellular

telephones independent of the telephone manufacturer. This improves on basic cable replacement applications by using a manufacturer-independent interface, but still fills the same general function of providing a long-lived connection.

Removing the cable, unfortunately, eliminates more than a piece of copper. Connecting a new cable to a device is a good indication of user intent, as when audio devices automatically disable their speakers upon the connection of a headphone jack. Furthermore, cables mean there is never a decision between multiple equivalent peripherals; their priority and function can uniquely be determined by if and where they are connected. In a wireless world, the initial substitutes for connector shapes and connection events have been proprietary protocols and activation buttons. Now, as bandwidth and compatibility requirements have driven the market towards standard protocols, service description and discovery is required to differentiate the various devices that may find they have a shared communication link, but no use for it. The activation buttons are still with us, but that too may change in time.

This thesis presents a new technique for discovering the services that are available within the radio broadcast range of a device. The proposed technique is designed for fast (timely) discovery of new services, and also offers opportunities for power saving techniques to extend the battery life of conformant devices.

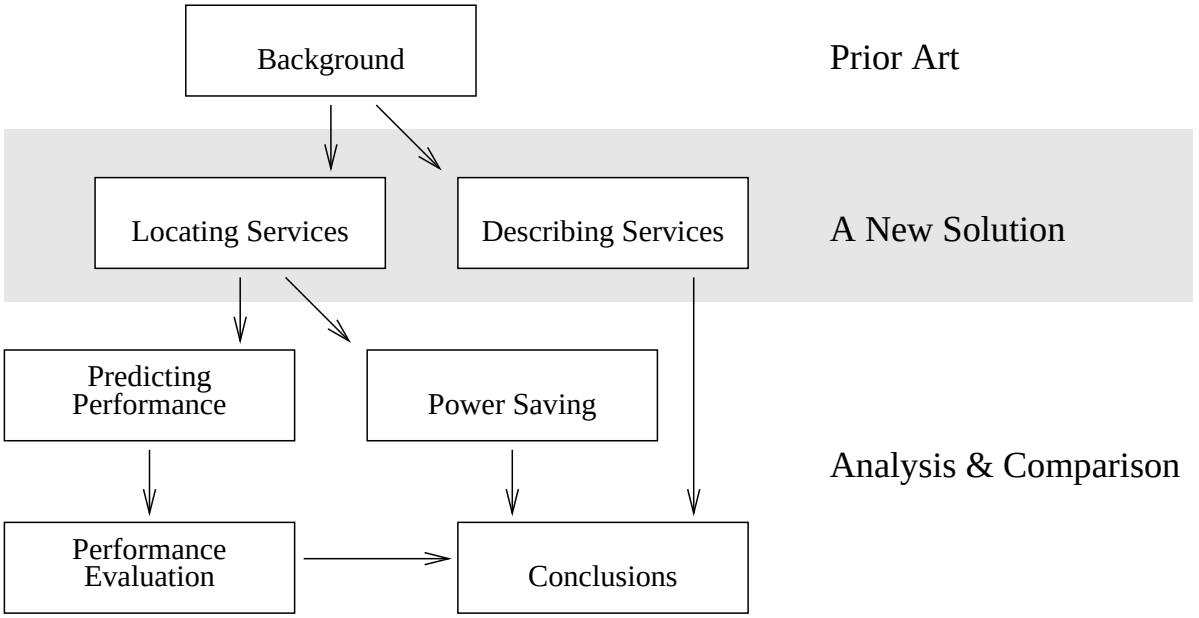
Typical target scenarios include the following:

1. A cellular telephone receives an incoming call. If the owner's headset is nearby, the call is routed through that. Otherwise, if it is near the owner's desk telephone, the call is routed through that. If neither of these alternatives are available, the cellular telephone handles the call with its own handset.

2. A palmtop computer has a PostScript document stored, and the user wishes to have a printed copy. No printer is immediately available, so the user asks to be alerted the next time a printer is nearby. With short-range devices, the normal walking speed of a user may result in passing a printer with only fifteen or twenty seconds of being within communication range.

In both of these cases, timeliness is important. In the first case, responsiveness is important. A call has arrived, and the telephone must discover what selection of devices is nearby in time to respond before the caller gets impatient. In the second case, no active trigger event occurs, but rather the passive event of communication becoming possible must be promptly detected.

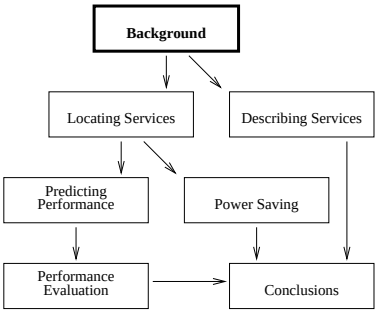
The body of this document will follow the general format shown below. After giving some background information about service discovery and other related current subjects, it will present a new service discovery algorithm. This will be followed by a theoretical analysis of the behaviour of this algorithm, then by a broadened comparison of its relative performance in comparison with other solutions to the same problem and some discussion of power use advantages. Finally, an overview is given of the description language that was implemented.



# Chapter 2

# Background

*This chapter introduces ad-hoc networks and service discovery. The similarities and differences between service discovery and route discovery are also discussed, and the unique problems of wireless networks are explained, in preparation for the new solutions presented in Chapter 3.*




Ad-hoc networks are very useful, and will become increasingly popular as network-ready devices continue to become less expensive. The most compelling use of networks is to enable service sharing between devices so, naturally, this has been a popular area of research [Pas01]. Successful, or at least popular solutions include DHCP (Dynamic Host Configuration Protocol) and SLP (Service Location Protocol). While these protocols have benefitted from the attention of the networking community, this attention has been motivated for wired infrastructure. This chapter presents background information about some existing service discovery systems, going on to explain which aspects of these systems are still useful in wireless networks, and which aspects must be changed.

## 2.1 Ad-Hoc Networks

As the size and weight of network-aware computers has gotten smaller, improving portability, the stability of network topologies has suffered. Recognition of this fact has led to research in *ad-hoc networking*, the formation of local networks involving whatever computers are active

on the local medium. These ad-hoc networks may have very short-lived topologies, in which computers enter and leave the group frequently.

Initial ad-hoc networking research considered an area called *nomadic computing* [Kle95], in which users were still using applications designed for static networks, but using them on portable platforms that might connect from diverse locations. This led to solutions like DHCP [Dro93] to assign a network address and inform a new computer of routing information for its current network. DHCP (discussed in more detail in Section 2.2.3) permits computers to configure their network parameters automatically to suit their local environment but, used by itself, gives the computer a new identity when that computer moves to a new network subnet. By behaving in this way, no changes are required to existing routing protocols, but long-lived operations are difficult if the computer changes subnets frequently.

An early improvement that solved the long-lived operations problem was Mobile IP [Per96], which allows traffic destined for a particular node to be routed normally to its home network, then tunneled to its current location, where a cooperating node unpacks the data, and delivers it to the mobile node as if that node were on its home network. This allows long-lived operations with mobile nodes, but does so by hiding the mobility from the application layer.

Today, IETF (Internet Engineering Task Force) research in mobile ad-hoc networking is focused in the MANET (Mobile Ad-hoc NETworking) working group. The working definition for mobile ad-hoc networks that the MANET group uses is this:

> A "mobile ad-hoc network" (MANET) is an autonomous system of mobile routers (and associated hosts) connected by wireless links – the union of which form an arbitrary graph. The routers are free to move randomly and organize themselves arbitrarily, thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a standalone fashion, or may be connected to the larger Internet.

Most of the research done through MANET has addressed the problem of routing in these networks, and has delivered interesting results in that area.

Platforms suitable to MANET development can enable the types of applications outlined in the introduction to this thesis. Although these applications will not use the routing capability of MANET, as they use the single-hop service availability as a strength, they will naturally encounter many of the same challenges found in the development of other MANET applications.

The relation of MANET research to this thesis is discussed further in Section 2.3, following a more thorough introduction of the service discovery problem.

## 2.2 Service Location Protocols

The role of a service location protocol is twofold: allow a new, unconfigured device to locate service providers, and describe the services offered by those providers in a useful way. In practice, most of the first problem has been pushed to the lower layers, usually using a combination of multicast and registry servers; the second problem, that of describing services and generating queries, has been the focus of rather more attention.

The various approaches to service location can broadly be categorized into push and *pull model* solutions, based on where the information sharing is initiated. In a pull model, a client initiates the exchange of information by sending a request to either a known set of servers, or as a general broadcast. A typical example of this would be ARP [Plu82], whereby a client converts an IP address to a device address by broadcasting a request on its local broadcast domain. If the specified device is available, a response is unicast to the client. The other category is *push model*, wherein devices offering services announce themselves, usually by broadcast, without waiting for a request. An example of this, also drawn from the set of IP protocols, is router information exchange [Dee91], in which routers periodically broadcast their state, allowing all neighbouring routers to adjust their routing tables accordingly.

An alternative categorization for service discovery approaches is *centralized* vs. *distributed*, based on the presence or absence, respectively, of a single authority on service availability. Because examples of these approaches are directly relevant to the subject of this thesis, we will first look at some existing approaches in detail, then consider those approaches in terms of the two categorizations just presented.

### 2.2.1 Bluetooth

Bluetooth is a popular new protocol for single-hop[1] ad-hoc networks. It was designed primarily as a cable replacement, connecting telephone handsets with wireless headsets or portable

[1]Eventually, routing is planned to allow “scatternets” with more than single-hop traffic, but that portion of the standard is not yet complete.

computers. This target application set led to the design of a protocol optimized for infrequent topology changes.

#### Protocol

Bluetooth [Blu01] is an integrated solution that combines a physical layer specification with a protocol stack that goes up to the application layer. Service discovery is spread over two levels: the baseband protocol allows devices to be discovered based on proximity, even if the requester and responder are in different logical groups (called *piconets*), and the upper-layer service discovery allows the actual services offered by devices to be queried.

#### Discussion

An important factor that affects the speed of service discovery is the existence (or absence) of a shared channel. The Bluetooth baseband communication uses CDMA, meaning that data communication between any pair of devices is impossible until they synchronize their clocks. The initial synchronization requires one device to transmit a large number of single frequency inquiries, and the other device to enter a mode during which it does nothing except listen for such inquiries. This type of baseband design allows for scalability, but makes discovery very slow. If a shared channel is present, it allows a great deal of information sharing. For example, if one device sends a query and receives a reply, another device might “listen in” on the reply, temporarily caching it. Because of the optimisation opportunity it offers, a shared channel is often assumed in ad-hoc wireless research.

### 2.2.2 CORBA, ANSA

Among the most general attempts to generate total system views are CORBA (Common Object Request Broker Architecture [Sie00]) and ANSA (Advanced Network System Architecture [Arc93, CS94]). These try to look at whole, wide-area networks as single systems, and offer ways for all service offerings and requirements to be expressed and exchanged.

#### Protocols

In both cases, service providers submit interface definitions to one or more servers, to which servers potential clients send interface requests. CORBA allows the *ORB*s (Object Request

Brokers) to manage the service connection, while ANSA uses *traders* simply to process requests and return the address of a *factory*, which will respond to service requests by assigning a thread to the new client. The general approach, however, is very similar between these two protocols. In fact, ANSA can be described in terms of the CORBA specification [Her94].

#### Discussion

These architectures were developed to facilitate fast, component-wise development of distributed systems. Their interface definitions allow component implementations to be upgraded or replaced seamlessly by offering well-defined interfaces, independent of implementations. These interfaces allow multi-vendor systems to be created with no explicit configuration. Both are heavily centralized, having all service registrations, deregistrations, and enquiries sent through a common point.

### 2.2.3 RLP, DHCP, Salutation

Reworking existing systems to conform to large groups is not an appealing option for many developers. Consequently, some projects less ambitious than CORBA and ANSA have enjoyed more success. RLP (Resource Location Protocol) was a simple, stopgap solution proposed in 1983 as a way for computers to locate arbitrary services in the Internet. Ten years later a more restricted protocol, DHCP (Dynamic Host Configuration Protocol), was proposed as a way for nomadic computers to quickly learn the appropriate configuration parameters for a new network (local IP address, gateway address, etc.) In 1995, Salutation tried to fill the need for more general service discovery, discovering fax-transmission devices, document storage servers, voice-message servers, and so on.

#### Protocols

RLP [Acc83] names resources by the assigned ID of their lowest level Internet transport protocol, combined with a variable length identifier based on some well-known property of the resource. The well-known property might be anything from an assigned IP port number to a text string. Services may broadcast a request for a service, or unicast a request to a known resource location server.

DHCP [Dro93] uses a more rigidly defined set of values. This is possible because it is designed for only one thing: communicating the information necessary to allow a newly arrived, unconfigured IP-capable host to use the local network. DHCP is centralized, and uses broadcast to discover the available servers.

Salutation [Sal99] could use a central server, but the normal usage is for each device to have its own server, and for that server to generate a list of all other available servers independently of device queries. On receiving a query from an application, the server then queries all other known servers (or a single specified server), and returns the results to the application. The services that can be discovered are from a discrete, well-defined set, and queries can specify quantified parameter values using relational operators. Once a service is selected, the data connection is routed through (and managed by) both servers.

**Discussion**

The general progression of protocol development shows a tendency towards fast, automatic configuration for devices. This progression was also contemporary to Intel, Compaq, and Microsoft introducing *Plug and Play* in 1992 for discovering new hardware added to a single computer. The motivating goal was to reduce the role of system administrators in adding new hardware, making it easier for users to buy new devices and use them without requiring special training or extra effort. Salutation moved this trend into a distributed office environment, giving firm definitions of fax, voice mail, and document storage device descriptions. It extended the query capability of RLP, allowing configuration parameters to be compared to requirements by defining operators such as “less than” for queries, and learned from the success of DHCP the value of standardized service descriptions.

### 2.2.4 SLP, Jini, UPnP

In 1997, SLP was proposed as a general, non-proprietary way to describe services on the Internet, while being optimised for local networks. It was revised in 1999 to improve scalability, and continues to get attention from the networking community. At the same time, other developers have been looking for ways to get better speed and network utilisation from less general protocols. Among the alternative solutions that claim to have benefits outweighing their costs are Jini and UPnP, their version 1.0 specifications having become available in January 1999 and

June 2000 respectively.

### Protocols

Service description in SLP [GPVD99] is string-based, with a two-layer hierarchy. These layers are called the *abstract* layer and the *concrete* layer. Beyond that, descriptions have a string-based list of attributes that should correspond to a generally known template, but are not required to do so. They are located using only the service types, and the attributes can be requested from the service agent once it has been located. An example type is the following:
service:printer:lpr://server.local.com/ports=8232
Queries can be for all service agents (SAs), for particular abstract service types, or for particular concrete service types.

For service discovery in SLP, directory agents (DAs) should be used. They are located via DHCP or multicast, and all service agents should registered with them. If a device cannot locate a DA, or chooses not to use one, it can multicast a service request to all SAs. Such a multicast request should be repeated for reliability. Repeated requests include a list of previous responders to prevent duplicate responses.

Jini [Sun99] uses a uniform deployment of the Java environment to improve the flexibility of the query/service-matching mechanism, and to allow device drivers to accompany service descriptions. Services are identified by their Java class, with requests matching instances (including derived instances) of a requested Java class. As with SLP, service definitions are further extended with an arbitrary set of name/value pairs (although Jini allows values with more complex types than just strings.) These name/value pairs (called *characteristics*) can be included in the queries.

UPnP [Mic00] is heavily influenced by SLP, but has dropped hierarchical service naming. UPnP has also added some extra companion protocols for getting an IP address without DHCP, and for using XML to define the precise interface being offered by a service.

### Discussion

SLP returns to the RLP idea of naming services partly by a well defined standard value (the abstract type), and partly by a programmer-defined type that allows for easy extension (the concrete type). SLP also makes the important step of standardizing a method for getting more

detail about a service once it is found. The basic Jini model of locating a service registry via multicast loses the generality of SLP, which only recommends the use of registry servers.

Jini allows the service characteristics to be made part of the query itself, while SLP only permits the list to be retrieved once the service has been located. Jini does not go so far as Salutation, allowing relative values for these attributes to be compared. This is because there is rarely a generic interpretation for such comparisons. Salutation, by requiring fully standardized definitions for all types, can define the interpretations a priori; Jini would have to define class-specific comparison functions, greatly reducing its search efficiency. Even so, it allows the list of services discovered to be better tailored to the requirements of the application, reducing the amount of unnecessary network bandwidth used, and simplifying the work of applications parsing the list.

### 2.2.5 HomeRF, HomePNA

Home networking protocols attempt to create a LAN that connects the various devices found in homes. Initially, these were for turning light switches and thermostats into peripheral devices for a home computer. This central computer idea is sometimes referred to as an *information furnace* approach, because the computer itself would be hidden away, while it would affect the entire building, like a central furnace. More recently, home networking has come to offer more peer-to-peer services, like allowing multiple computers to simultaneously share a single connection to the Internet, or providing file sharing services.

#### Protocols

HomeRF [Hom00] has offered a merger of IEEE 802.11 (CSMA/CA) with DECT (Digital European Cordless Telephony). By time-dividing the channel into six isochronous slots and one wide asynchronous slot, it offers a single carrier medium for all home requirements. In accordance with the relatively static nature of the digital connections, however, discovery of digital devices is the same as for any Ethernet. NDIS (Network Device Interface Standard) interfaces are available for Windows applications, and vendors can use their favourite technique for locating collaborators. Telephones can still locate their base, as with DECT, but the data transmissions are Ethernet.

HomePNA (Home Phoneline Networking Alliance) [FH00] uses a different medium for the

same solution. Rather than time dividing the air, it uses existing telephone lines on a frequency higher than normal home telephone systems require. As with HomeRF, configuration (device discovery, service location, etc.) is left to the tried and true standards of any wired network.

**Discussion**

In general, while home networking has increased the pervasiveness of network connections, it has done so by using standard communication protocols over new media. Devices and their services are discovered using the same set of techniques that can be found in existing fixed-node LANs, which is to say that it is left up to the higher level system using the network, and is not specified in the networking standards themselves.

### 2.2.6 Active Badge, HIPERLAN, IEEE 802.11

Sometimes, device discovery is part of the network protocol design. Active Badge uses various sensors to report to a central server the identities of mobile devices in its area. Similarly, IEEE 802.11 allows devices to periodically announce their presence by broadcasting their physical address. Most sophisticated of the three, HIPERLAN (HIgh PErformance Radio Local Area Network) requires not only for all devices to periodically announce their presence, but also for all *forwarding nodes* (nodes willing to participate in multihop packet forwarding) to include a list of their neighbours.

Some protocols have need for discovery at the lowest levels. In the case of Active Badge, locating devices is the primary goal of the system. IEEE 802.11 discovers other devices in order to maintain consistent clocks in the local area (necessary for its wireless medium access coordination) and gets the added bonus of allowing devices to learn the network addresses of their neighbours. HIPERLAN uses the periodic beacons primarily for routing; allowing forwarding nodes to maintain a fully connected map of the extended HIPERLAN.

**Protocol**

Active badge is a prototype system for tracking the locations of objects and people in an office environment. Half of the system is comprised of an array of well-powered IR transceivers (called sensors), installed at known locations in the ceilings, and connected to form a conventional network. The other half is the portable “badge” devices, also containing IR transceivers.

Each “active badge” [HH94] independently transmits its own (globally unique) identity every 10 seconds. Sensors would continually listen for this broadcast. Upon receiving a badge ID, a sensor would package its own identity with the badge ID, and send the information to a central network server.

The IEEE 802.11 standard, a protocol designed for wireless LANs [Chh96, IEE97] uses periodic broadcasts to maintain the clock synchronization necessary for spread-spectrum communication. While the IEEE 802.11 standard is defined for both ad-hoc and static networks, and both use these *beacons* (as the periodic broadcasts are called), the interesting version here is the ad-hoc IBSS (independent basic service set) mode. Each IBSS has a defined beacon interval that is communicated to new devices upon arrival. Each member of the IBSS, upon receiving a beacon, sets a count-down timer to the beacon interval value plus a random backoff value. If another beacon is received before the timer has expired, a the timer is reset as before; if the timer expires before a beacon is received, then a beacon is sent and the timer is also reset as before. Beacons include, among other things, the device’s local timer value, address, and network ID. With this information, devices can learn not only the correct timer value for synchronization, but also the identities of all neighbouring devices.

HIPERLAN is an ETSI (European Telecommunications Standards Institute) standard [ETS96] for communication in wireless local networks. A design goal for HIPERLAN was to present a network view similar to Ethernet, but using ad-hoc radio communications. This restriction meant that, while the protocol would be defined only for the DLL (Data Link Layer) and below, rudimentary routing would be necessary to overcome the physical limitations of the medium. To accomplish this goal, special MAC data units (called *HELLO* data units) were defined, through which devices participating in a HIPERLAN must announce their presence to all neighbouring devices at least once every 40 seconds (recommended). Furthermore, if a node intends to participate in forwarding packets, it includes information about its immediate neighbours in these announcements.

#### Discussion

In active badge, there is a clear division between clients and servers. In contrast with examples like Bluetooth, in which all devices can be discovered via the same mechanisms, the sort of *complimentary protocol* used in active badge divides the devices into those that can be discovered

and those that can discover them. Complimentary models are also appropriate when, for example, the service being offered is network connectivity. In that case, as with the client location goal of active badge, neither the server devices nor the client devices need to discover each other, and the physical characteristics will probably be fairly constant across each of the two groups. In such a situation, being able to tune the discovery behaviour to save power for whichever group has more stringent battery requirements can be a huge advantage. Furthermore, if the servers are normally stationary, and only the clients mobile, then prefetching/caching can be used to advantage.

It is important to note that the power restrictions may go either way [TBJ00]. Sometimes the servers are connected to an infinite power supply (the most frequently studied case), and sometimes they are not only battery powered, but hidden in ceiling tiles or other inaccessible places where changing batteries is difficult. By choosing "client beaconing" (having the clients initiate the information transmission) the active badge system allows the clients to be dormant most of the time. The servers must be constantly active, listening for client beacons, but this is acceptable since they are connected to an infinite power source.

### 2.2.7 Summary

This section has introduced some characteristic examples of how service discovery is currently used and implemented. Of those schemes presented, only HIPERLAN, IEEE 802.11, and active badge (the three protocols that consider the solution at the Data Link Layer and below) use a push model for service discovery, with all others expecting clients to issue a request when they require services. The disadvantages of the pull model are offset in some systems (e.g., Jini and UPnP) by allowing their servers to accept notification requests from clients, wherein clients ask the server to inform them of any new service information of some specified type as soon as it arrives. Such a workaround is, of course, only possible in protocols that have central servers for processing the requests.

Similar uniformity is present in the choice to use central servers whenever possible. Some protocols (e.g., RLP, SLP, and UPnP) allow for broadcast requests when no central server can be found, but prefer to have a central server for normal operation. Only Bluetooth, active badge, HIPERLAN, and IEEE 802.11 never expect a discovery server to be present. Of these, active badge might be excluded, because the role of clients is solely to locate the nearest server, so

if they knew where to find a discovery server, they wouldn't have anything to ask it (i.e., their goal would already be achieved.) Similarly, IEEE 802.11 has a different mode that can take advantage of a central server (base station) when one is available, although that base station still pushes the beacons, rather than waiting for a request to be received.

Why have the existing protocols been designed this way? Consider first the choice of centralization. All protocols designed for fixed networks are designed for installation in a configuration that already contains a number of central servers, like the internet gateway, router, and name server. The only protocols that choose not to use a central server are those that give highest priority to ease of configuration. Both Bluetooth and HIPERLAN are designed for the use in ad-hoc groups of users, rather than in large, managed networks. (While HIPERLAN is designed for large groups, this refers to multi-hop groups; they must still be configured automatically for their local set of neighbours.) This common design point means they must function with both minimal extra hardware, and minimal manual configuration. A network with even one special server already installed, maintained, and reachable (like the name server), could easily have a new server installed on that machine.

There is almost universal agreement on use of the pull model. If a central server is being used, then the pull model is clearly superior. The use of a central server reduces the problem of service discovery to initially finding that server and then reliably exchanging services requests with it. Finding the server can be done via manual configuration, or through unreliable broadcasts or multicasts being repeated until a reply is received. In a static (or nearly static) configuration, the delay associated with this process is probably acceptable. Reliable point-to-point communication is clearly fast enough for most requirements so, once the central server is found, the remainder of the discovery problem is solved. Network reconfiguration, like arriving or departing nodes, can be found by allowing clients to request updates from the server whenever they occur. The only reconfiguration that cannot be easily handled is the server going down. This last problem is the reason for RLP, SLP, and UPnP allowing optional operation without a server.

Of the selections that do not use a central server, one (Bluetooth) uses a pull model, and three (active badge, HIPERLAN, and IEEE 802.11) use a push model. For Bluetooth, a push model would be almost impossible, because broadcast is not possible. A limited form of push, informing devices that have already joined a piconet of services known to other devices in the same

piconet is possible [Nid01], but communicating with a previously unknown device requires the initiator of the connection to cease all other communication activities for a significant amount of time. Because Bluetooth is designed for the normal case of service discovery to involve a server that has not already joined the piconet of the client, pushing information would be far too expensive an operation to be used for most normal discovery operations. In the cases where the media allows it, the distributed protocols have both chosen to use push. We will return to possible explanations for this decision in Section 2.4, but first it is worth considering the related problem of route discovery.

| Protocol | Defined for Layers | Push/Pull | Centralized | Service Definition |
|---|---|---|---|---|
| Bluetooth | Session →Physical | Pull | No | Physical discovery by standardized class value, then soft query by extensible standard description |
| CORBA | Session →Transport | Pull | Yes | Functional interface specification |
| ANSA | Session →Transport | Pull | Yes | Functional interface specification |
| RLP | Session →Network | Pull | Optional | IP protocol ID + application-defined value |
| DHCP | Session →Network | Pull | Yes | Standardized names |
| Salutation | Session →Network | Pull | Yes | Standardized names + optional parameter values |
| SLP | Session →Network | Pull | Optional | Standardized names + optional application-defined name for initial discovery, then formatted list of parameter values available by request |
| Jini | Session →Transport | Pull | Yes | Class template + parameter values |
| UPnP | Session →Network | Pull | Optional | XML-based standard templates + user extensions |
| Active Badge | Application →Physical | Push | Yes | Unique device ID |
| HIPERLAN | Network →Physical | Push | No | Unique device ID |
| IEEE 802.11 | Network →Physical | Push | No | Unique device ID |

**Figure 2.1:** Service location protocol summary

## 2.3 Route Discovery in Ad-Hoc Networks

Packet routing in fixed networks is a well-known problem. It has been solved many times but, while these solutions allow for a changing network, they tend to emphasize scalability over responsiveness to change. As the MANET research has highlighted, responding to change must be a high priority for any routing protocol used in mobile ad-hoc networks.

Routing for large ad-hoc networks is difficult, as this section will show. There is a tradeoff between *pro-actively* determining routing tables, and *reactively* searching for the destination only when it is actually required. A pro-active solution corresponds to the push model introduced above, delivering faster response at the cost of frequently outdated information and steady "background noise" to update local tables. Reactively searching for the destination corresponds to the pull model, causing large bursts of network traffic and poor response time, but delivering more current results and requiring no activity except in response to application demands. Whichever routing technique is chosen, its role in an ad-hoc network is to know about a changing environment under strict timeliness conditions. It is therefore useful to this discussion of service discovery to consider the types of routing algorithms that exist.

### 2.3.1 Gossiping

Routing is really a problem of information sharing. If the *connectivity* of a network (the set of nodes with which each node can directly communicate) is known, then finding routes is a simple process of applying a shortest path algorithm (e.g., Djikstra's). Unfortunately, there is not usually any one node that knows the connectivity of the whole network. Each node individually knows its own local connectivity, or can easily discover it, but getting this information collected at a single node is more challenging. This "information dissemination" problem is an old one in graph theory. It is usually phrased as follows: given $n$ people with telephones (a completely connected graph) and each having a unique piece of information, further assuming that any person can talk with at most one other person during a single telephone call, how many calls are necessary for all people to have learned all the pieces of information?

An overview of the various solutions that have been proposed, and of the related problems of making communication unidirectional (directed graphs) or allowing conference calls ($k$-uniform hypergraphs[2]), or that of eliminating duplicated information are surveyed in [HHL88]. Interesting variants include the following:

- If nodes can send a message to one neighbour at a time, but choose that neighbour randomly (with the optional restriction of knowing better than to send identical information to the same neighbour twice), how many messages will be required to complete the gossiping problem [Lan54]? This result is represented in [HHL88] as an approximate result

[2] A $k$-uniform hypergraph on $n$ vertices is a hypergraph on $n$ vertices in which each edge contains $k$ vertices.

to the question "If each node knows the set of nodes with which it is connected, but not the connectivity of any other node, how many messages will be required for all nodes to know the connectivity of all others?"

- New information is continually arriving at nodes, and they must still convey it to all other nodes in a timely manner. (Perpetual Gossiping [LR93])

- If node $i$ broadcasts everything it knows to all of its neighbours at time $i \mod n$, find the optimal labeling to have all nodes discover all information in minimum time [AK83].

It is readily apparent that the solutions to these problems have bearing on service discovery, but there are important differences: most significantly, the connectivity of the graph is changing, and message delivery may not be reliable.

### 2.3.2 Practical Solutions

An adjacency matrix representing the total connectivity of a network increases in size with the square of the number of nodes. This scalability restriction means that combining gossiping with a theoretically optimal (static) graph analysis solution will have problems with large networks. A natural optimisation, and one used in distance vector routing [PB94], is for nodes to compile their own list of how many hops they are to each other node and which adjacent node is in line to achieve that distance, and advertise just that list. Receiving one of these lists from each neighbour allows a node to determine the optimal next node to reach any destination, and reduces advertisement size by a factor of $n$ (for $n$ nodes). The total information that an individual node must gather is reduced by a factor of $\frac{n}{\log_i n} \simeq n$ where $i$ is the typical number of neighbours for a node. Lists can be further compressed through the use of node groupings as with the *Grid Location Service* (GLS) [LJC+00] (also called *virtual backbones* in [HGBV01]), offering even better scalability, but the general theme of these algorithms is the same. Each node lets its neighbours know how effectively it can forward a packet to each other node in the network.

This problem can be seen as one of service discovery in which each node offers $n$ services, specifically, the ability to reach each node in an advertised number of hops. In some schemes, the service description will include not only the number of hops, but some measure of the expected reliability of the connection being offered [HGBV01]. In this instance of the service

discovery problem, simply locating the required service (a route to node $x$) is easy, and differentiating between different instances of the service (route lengths and reliabilities) becomes the most important aspect of the solution.

### 2.3.3 Reactive Routing

Reactive routing, also called *on-demand* routing, waits for a route to be required before taking any action. This corresponds to the pull model of service discovery. Arguably the simplest example of this class of routing protocol are those using *flooding* to transmit the identity (address, name, or other description) of the desired destination to one or more neighbours, which then update the return path and forward the request on to their neighbours. Eventually, the request arrives at the proper destination, and is returned along the way it came. The usual challenges in these protocols are to minimize the number of actual transmissions required for the request, while still allowing the shortest route to be discovered. Recently, use of this technique to find the route that is most "battery friendly" to intermediate nodes has also been under study [SWR98].

A popular use of flooding is seen in dynamic source routing (DSR) [JMHJ01]. When this system is asked to find a route to a particular destination, it floods the network with queries. Each time a query packet is propagated, the intermediate node adds itself to a list, which list can then be used in reverse as a source route for returning the completed route to the initiator. Nodes ignore duplicate requests, implicitly assuming that the first copy to arrive must have come via the most efficient path. Naturally this cannot be done for every packet to be sent, so a small cache is kept at each node with the most recently discovered routes. When a failure is detected in a route, a *route error* packet is sent to remove it from the caches of all intermediate nodes that can still be reached, and a new flood query is started. (The route may also be "salvaged" by initiating the new query from an existing intermediate node, retaining a portion of the original route.)

In DSR, all routes are found, used, and removed in response to actual client action, hence its classification as reactive. Once routes are found, they will be reused until they break. This means that no overhead other than (optional) normal acknowledgements are required except when application requirements change, or nodes leave the group. It also means that new nodes will not be discovered or considered for improving cached routes (except through gratuitous replies, explained below) but, for many applications, this is not a problem [MBJJ99].

Actually, DSR has two components that could be considered proactive: gratuitous replies and route caching. "Gratuitous replies" allow nodes overhearing an ongoing connection to send a spontaneous routing message to the source, offering a better path to the destination. This could be called a pro-active behaviour, but is intended more for maintenance/improvement of active connections, and does not directly address discovery time for completely new nodes.

Route caching is more directly proactive, allowing intermediate nodes in a connection to use subsequences of that route for their own purposes. For example, if a connection from node $a$ to node $f$ follows the route $a, b, c, d, e, f$, all the intermediate nodes $b$ through $e$ learn a route to all other nodes in the path. In this example, some route discovery messages are avoided through this caching, but it is still primarily reactive; the extra cached routes were learned only as a result of the actual requirements of nodes $a$ and $f$.

### 2.3.4 Pro-Active Routing

Flooding requires a lot of messages to be sent and received before a route has been found. One way to reduce the message requirement is for machines to maintain lists of nodes that are reachable through them. By preparing reachability lists in advance of specific requests, nodes are being *pro-active*. This behaviour corresponds to the push model of service discovery, accepting a background level of general maintenance work, whether or not requests are being used, in exchange for less work processing each individual request. For fast route discovery, pro-active strategies are very useful.

In pro-active systems, the sending node hopes that its table gives a sufficiently complete and current picture of the network that it can determine where to send the packet based solely on that information. Temporary inconsistencies between views can result in short-lived forwarding loops, but the systems generally give much better response time than reactive ones. A comparison [DnYS98] of the various routing protocols being considered by the MANET group in 1998 showed pro-active routing to give better end-to-end delays and better fraction of packets successfully delivered, at the expense of a higher total network traffic load[3]. Examples of pro-active algorithms include Optimized Link State Routing [JMQ00] and Destination-Sequenced Distance Vector Routing (DSDV) [PB94].

[3]Some of these differences are addressed in [MBJJ99] by using caching and snooping with DSR, but the earlier paper considers the fundamental differences in the techniques.

## 2.4 Sharing Service Information

When a protocol is designed, some assumptions must be made about its target environment. When the environment changes, and some of these assumptions become invalid, the effectiveness of the protocol is damaged. The shift from a fully wired infrastructure to the rapidly changing, heterogeneously connected environment found today has already given rise to numerous examples of this. SMTP is unacceptable for intermittently connected users, so POP has been introduced [MR96]. IP is insufficient for addressing devices that move amongst networks, so Mobile IP has been introduced [Per96]. Existing service discovery protocols do not perform adequately in wireless ad-hoc networks, so a new alternative is required.

In a wired network, common assumptions are that using a centralized server is a reasonable option, and that low round-trip times can be expected for transmissions. The latter assumption, although common to such existing service discovery solutions as RLP, DHCP, Salutation, and Jini may not be inherent to centralized approaches, but having a central server clearly is. Therefore, the arguement given here will first explain the motivations against centralization, then present the remaining options with consideration given to the large round-trip times that result from slotted-access protocols.

The service discovery protocols mentioned above use a centralized information server that listens for broadcast or multicast packets on a well-known address. This server accepts registration requests for available service offerings from the network, and answers queries about those services. Although, in wired networks, services like DHCP can reasonably be expected to be provided by the same organisation that maintains the physical equipment, ad-hoc groups cannot be expected to provide such infrastructure. An ad-hoc group may be formed by any two (or more) devices that come into proximity with one another, so guaranteeing at least one server in such an arbitrary group requires almost all candidate devices to be running that server. On top of the increased overhead resulting from this practice, such routine deployment of servers means that groups would regularly be formed with more than one server as a member. In this situation, maintaining the centralized approach would require servers to exchange state, and elect which server is to represent the discovery protocol in each ad-hoc group. If devices can be members of more than one network, the scenario becomes increasingly complicated.

Accepting that alternatives to the centralized approach are required, a way must be found to make the distributed approach work. The two obvious categories are the push- and pull-style

solutions. The problems with the pull model include timeliness and power consumption. The timeliness troubles come from delays inherent in acquiring a broadcast opportunity, sending the request, and repeating it in case of loss; and the increased power consumption of servers is the result of so much time spent listening for connections. Deeper insight is provided into these difficulties in Chapters 5 and 6, also showing that they are offset somewhat by a conservation of bandwidth (no messages are required except when a client actually requires a service). This saving is not necessarily very large, however, and significant advantages can be realized by using a *push model* solution to have devices maintain a view of all the services that are available to them. The idea of maintaining such a world view is also referred to as "pro-active service discovery." Ideally, to maintain an accurate view, a device would send a request every time the set of services available to that particular client changed. This behaviour would require either user intervention, which we do not consider acceptable, or another discovery algorithm running at a lower layer, which is redundant.

In considering push-style solutions, HIPERLAN and IEEE 802.11 both allow a device to learn the identities of its neighbours via periodic broadcasts (beacons) from all nodes. A disadvantage to using these beacons for service discovery is that they both require a new member of the group to be present for at least as long as the maximum time between broadcasts for all other member devices before the new device has a complete view of the available services. That means, if devices are to have a view of their environment no more than ten seconds out of date, then all devices must broadcast their service offerings at least once every ten seconds.

Requiring frequent broadcasts from all machines is not desirable. While one might imagine that a broadcast medium, like the airwaves, would offer cheap broadcasts, air protocols for pervasive devices tend to value power consumption very highly. In many wireless media access control (MAC) protocols, broadcasts are scheduled, allowing devices without active connections to turn off the power to their receiver circuits outside of these times. This behaviour means that the full bandwidth of the medium is not available to broadcast packets, thus limiting the total bytes per second, and also that access time is increased by the requirement that a device wait for the next available broadcast slot. Combined with the generally lower and more restricted bandwidth of wireless media, this strategy makes independent periodic beacons an inadequate approach.

The IEEE 802.11 approach deals with the problem much more appropriately. This solution

allows a passive, beaconing solution, except that advertising broadcasts are slotted. This slotted beaconing puts an upper bound on the number of advertisements per second, regardless of how many devices are present. In situations where there is not enough time available to wait for a broadcast, an "active search" request can be sent, requiring all devices receiving the request to respond immediately. By offering these two alternatives, the designers are acknowledging that neither is perfect: a passive search is slow, and an active search is both expensive and unreliable. Something better is called for.

## 2.5 Chapter Summary

The astute reader will have noticed that this background chapter has presented "around" the problem of service discovery in ad-hoc networks. Related material in the history of general service discovery was presented that was mostly intended for either fast or reliable connections. Ad-hoc networks were discussed, but mainly in the context of route discovery. This format is made necessary by the lack of existing standards for general service discovery in ad-hoc networks.

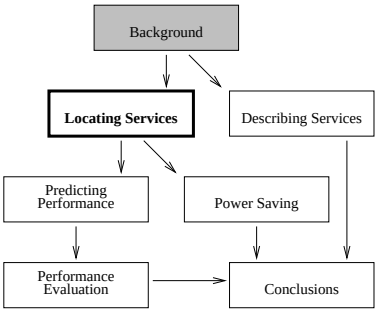
To date, wireless protocols have mainly targetted the provision of services that mimic existing wired devices. At the moment, for example, the wireless application receiving the most media attention is web browsing. In the coming decade, the potential of wireless devices and ad-hoc networks will become more clear; for now, protocols enabling the new applications enabled by wireless ad-hoc networking have very little competition.

Therefore, because the author is not aware of any other work currently addressing this specific problem, this chapter has presented a cross section of related work that offers insight into the possible directions that a new solution might take. The remainder of this thesis will present and defend such a solution.

# Chapter 3

# Locating Services

*Having learned the state of the art for service discovery and wireless networks from Chapter 2, we will now look at a new algorithm for giving timely information about services available within a single hop of wireless devices. Some further background discussion will then examine routing protocols in the context of this new algorithm.*




---

If devices are going to provide meaningful services, then they have to cooperate. This means that each device must know what services are available to it, and also ensure that other devices know what services it is offering. Reliably knowing this information would require all devices to be in constant communication with all other devices in their neighbourhood but, for bandwidth and power consumption reasons, this is not an acceptable solution. The challenge, therefore, is to describe a useful compromise between the desire for minimal data communication, and the desire for full knowledge of the immediate environment.

The most obvious solution for this problem is for all devices to wait until they require a service, then broadcast a request for candidates that can provide this service. This pull model solution would eliminate unnecessary communication, since all communications would be either requests or responses. Such a solution might be good if any provider were as good as any other, and timeliness of delivery were not important, as with a calendar application trying to remind the user of an appointment. A calendar device could advertise for anything that can go

"beep." Assuming that one beeper is as good as another, any responder would be asked to beep, the user would notice this alert, and respond to the alarm.

If a device is to make an informed selection from the services available, it must have timely and accurate knowledge of its options. Unfortunately, using the pull model to accomplish this service discovery in a wireless ad-hoc environment, even if technically possible, will be slow. Accuracy requires a reliable broadcast for requests, and acknowledgements for the responses from answering devices. The periodic burst errors that characterize the medium [ZR99] necessitate repetition of requests, and periodic use of power-saving idle modes (disabling the transceiver circuits, as allowed for in IEEE 802.11) delays responses. For the general case, a solution must be sought that offers prompt discovery but still places only low demand on the network.

A protocol is presented in this chapter that allows a device to maintain a timely internal list of other available devices, and ensure that its service offerings are known to neighbouring devices. Analyses of its performance, and comparisons to other algorithms follow in later chapters.

## 3.1 The DEAPspace Algorithm

So prefetching is worth while, but how should we do it? The initial hypothesis was that broadcasting the service descriptions to all devices, rather than responding to individual queries, would be helpful. By doing this, a single mechanism could be used both for discovering services, and for monitoring their continuing availability: seeing an advertisement allows a device to conclude that a service is available, and failing to see an advertisement for some period of time allows it to conclude that the service is not available. Of course, these answers lead to more questions; specifically, how often should devices advertise their services, and how long a gap should allow a device to consider a service absent.

Trying to develop on this idea, a second hypothesis was added: that the effectiveness of discovery could be improved by allowing devices to collaborate in service announcements. If a service advertisement includes not only the services offered by the device actually sending that announcement, but also the services offered by its neighbours, the network could be made more efficient. When a device sees its services advertised on its behalf by a collaborator, it can cancel its own next advertisement. Not only could multiple broadcasts be replaced by one, but this would provide a feedback mechanism to allow devices to discover what is known about their

offerings by other machines.

Clearly, if devices are retransmitting on behalf of each other, they should have some mechanism for ensuring that the device they are advertising for is actually still present in the network. The simplest way to do this is by associating expiry times with services. Ideally this time will be long enough to avoid expiring services that are still offered, just because they haven't been heard from for some time. If it is too long, and some device receives an outdated advertisement, it is still not a tragedy. By sorting its alternatives in reverse order of expiry time, either there is no alternative available anyway, so it was worth the try, or the services that are actually present (and hence renewing their expiry times) will be tried first. For all these reasons, devices should update the expiry times on their own services only, just repeating the value received for remote services. Notice that, because expiry times extend monotonically farther into the future, repetition of outdated records (due to lost transmissions) will not interfere with devices that did receive the lost update, since the outdated transmission time can be recognised and ignored. It is also important that new devices have an opportunity to advertise their presence as soon as possible after arriving in a new zone, as is shown in Figure 3.1.

Figure 3.2 shows a more serious worry of prefetching, demonstrating that short timeouts and frequent renewals would still be desirable. This difficulty is inherent to all pro-active approaches, and must be considered when designing the algorithm.

These are enough hypotheses and observations to get started designing an actual algorithm. The basis of this system is that devices should broadcast their entire world-view on a regular basis, and listen between transmissions for the broadcasts of other devices, updating their world-view accordingly. By allowing all devices to incorporate the new elements from others into their own views, individual lost messages have very little effect on the overall system, because the next broadcast by any device that did not miss the first one will repeat (and update) the same information. In this way, we get reliability through continued repetition. Also, as each device revises the contents of the world view before repeating it, the information undergoes a constant slow alteration as each broadcaster updates and corrects the world-view. It is awareness of time that separates this algorithm from typical gossiping approaches. Information is not only shared, but also timely.

In a group of $n$ devices, this approach allows each individual device to transmit less frequently by a factor of $n$ by making the size of each transmission larger by the same factor.

| time | event | world view of X (service, expiry) | default service choice at X |
|---|---|---|---|
| 1 | A advertises | (A, 11) | A |
| 3 | A leaves | (A, 11) | A (Unavailable) |
| 5 | B advertises | (A, 11), (B, 15) | B |
| 9 | B advertises | (A, 11), (B, 19) | B |
| 11 | A expires at X | (B, 19) | B |

**Figure 3.1:** In this example, services A and B are equally acceptable to X; services renew their own expiry times to ten time units in the future when they advertise. Service A arrives first, is discovered by X, and then leaves. For a brief time, if X wants a service, it will try to connect with a non-existent device (and fail), but there was no valid service available anyway. Once an alternative has been discovered, it is automatically made the default choice between the two, because it has the later expiry time.

| time | event | world view of X (service, expiry) | default service choice at X |
|---|---|---|---|
| 1 | A advertises | (A, 11) | A |
| 5 | B advertises | (A, 11), (B, 15) | B |
| 7 | B leaves | (A, 11), (B, 15) | B (Unavailable) |
| 9 | A advertises | (A, 19), (B, 15) | A |
| 13 | A advertises | (A, 23), (B, 15) | A |
| 15 | B expires at X | (A, 23) | A |

**Figure 3.2:** As in Figure 3.1, services A and B are equally acceptable to X, and renew their own expiry times to ten time units in the future. In this example, Service A arrives first, and is discovered by X, then B arrives and is also discovered by X. This time, however, B leaves. If X wants a service between times 7 and 9, it will try to connect to B (and fail) even though A would have been a better choice. This will result probably in poor connection establishment time to the user as the system waits for the first connection attempt to time out.

This approach could be implemented with each device scheduling its own broadcast periodically, like the regular beaconing algorithm with a longer period, but that would miss a good chance for adaptability. By scheduling with the short period, but cancelling and rescheduling when a broadcast is seen from another device, the network becomes automatically responsive to loss. Some jitter is required to avoid causing a pathological case for the underlying network with all the devices sending their next transmission at the same time, so when a broadcast is scheduled the time is taken from a range. Whichever device happens to choose the lowest value will be the one to transmit an advertisement, whereupon it and all other devices that receive the advertisement will schedule again.

### 3.1.1 Assumptions

The algorithm is positioned above the MAC layer, and makes the following assumptions about the underlying network:

- Broadcast messages are possible.
- Messages are delivered either correctly or not at all. No partially corrupt packets will be encountered.
- The MAC is able to keep packet loss rate mostly independent of the behaviour of higher-layer protocols, excepting increased loss due to congestion. That is to say, similar packet loss rates represent similar environments, even with different behaviour at higher layers.

No routing is assumed or expected, because the goal is to locate services within direct transmission range. These assumptions are not unreasonable, and are true for such common protocols as IEEE 802.11 and Ethernet.

### 3.1.2 Algorithm

To be more concise, individual nodes (devices) are constructed with the following properties:

- Each node maintains a list of service descriptions.
- Nodes participate in the advertising of services through a broadcast mechanism.

- A service is described by a service description that includes at least a time-to-live and the address of the node offering the service.

Individual service descriptions shall be referred to as service elements (SEs), and a list of SEs shall be referred to as a service advertisement message (SAM). If a time-to-live value is set to $x$ seconds, at time $t$, then the SE will be said to *expire* (no longer be valid) at time $t + x$ unless the time-to-live is reset or updated before that time.

The behaviour of each node is governed by the following rules, in which emphasised letters are used to connect a rule to its use in the sample implementation that follows.

- The broadcast mechanism works as follows:
  - Broadcasts are scheduled to occur within bounded time windows, with at least one node broadcasting in each window, and the absolute range for each window being determined by an adaptive back-off mechanism. $\Leftarrow \mathbb{A}$
  - Nodes that receive a SAM in which one of their SEs is absent or about to expire increase their chance to broadcast next by choosing a shorter back-off time. $\Leftarrow \mathbb{B}$
  - Before broadcasting, a node re-initialises the time-to-live for its local services. $\Leftarrow \mathbb{C}$
- Each node processes a newly received SAM (called REMOTE in the pseudocode examples) by merging it into its internal world view (called LOCAL).
  - When merging, the time-to-live values in LOCAL are updated to equal the values in REMOTE if and only if that SE refers to a service offered by a device other than the one performing the merge, *and* the time-to-live value in REMOTE is later (farther in the future) than the value already in LOCAL. $\Leftarrow \mathbb{D}$

In order to give a sample implementation, some configuration information must be decided. These values are used in the pseudocode that follows:

- Time-to-live values are reset to a maximum value NormalExpiry.
- Normal broadcast timeouts are taken from range $X$.
- Short broadcast timeouts, used when a device sees one of its own SEs missing or about to expire in a received SAM, are taken from range $X'$ strictly less than $X$. In this case, "about to expire" is defined as having a time-to-live value less than MinExpiry.

- The set of services in a received SAM are called REMOTE, the set of services in the (locally stored) current world view is called LOCAL, and the subset of LOCAL representing services actually offered by the local device itself is called MINE.

Also, define the functions **read(`t`)** and **getTimeout(`I`)**:

**read(`t`)** blocks on the network interface (i.e., suspends execution of its thread, waiting for network interface events), returning a SAM if one is received in time less than `t`, or timing out otherwise. In receiving a SAM from the network, the **read** function converts time-to-live values to the local reference clock (via a reference clock value in the SAM or some similar mechanism).

**getTimeout(`I`)** returns a value chosen randomly on the interval `I`

```
1  advertise(LOCAL) {
2     time tout ←getTimeout(X)
3     loop(forever) {
4        REMOTE ←read(tout)
5        if(timed out) {
6           foreach s ∈ LOCAL                ⇐ ℂ
7              if(s ∈ MINE)
8                 s.expiry ←NormalExpiry
9           broadcast(LOCAL)
10          tout ←getTimeout(X)
11       } else {
12          Interval I ←update(LOCAL,REMOTE)
13          tout ←getTimeout(I)              ⇐ 𝔸
14       }
15    }
16 }
```

**Figure 3.3:** An implementation of the advertise function

The **advertise** function defined in Figure 3.3 is the main line of the algorithm. Lines 2 and 3 just schedule the first advertisement and begin the advertising loop. Line 4 reads a SAM from

the network, and line 5 checks to see whether one was received in time. If it was, then the local list is updated with the received one, and the next timeout is chosen based on the result of the update function. If the read timed out, then the local list has the expiry times on local services renewed, and is then broadcast to the group. In practice, the loop on lines 6 through 8 should probably also include a check to remove expired non-local services from the list, but this could equally well be included in the broadcast function or as an internal property of SE objects in the implementation language. In the interest of a succinct example, explicit SE expiry code has not been included in this sample implementation.

```
1   Interval update(LOCAL,REMOTE) {
2       foreach r ∈ REMOTE {
3           if(r ∉ MINE) {
4               if(∃_{s∈LOCAL} r.id = s.id) {
5                   if(r.expiry > s.expiry)
6                       s.expiry ←r.expiry          ⇐ 𝔻
7               } else {
8                   insert r into LOCAL
9               }
10          }
11      }
12      if(∃_{s∈MINE} s ∉REMOTE) return X′   ⇐ 𝔹
13      else if(∃_{r∈REMOTE} {r ∈ MINE AND
14                  r.expiry < MinExpiry}) return X′
15      else return X
16  }
```

**Figure 3.4:** An implementation of the update function

The **update** function defined in Figure 3.4 is a bit more complicated, but still comes directly from the algorithm as it is described above. Lines 3 through 10 iterate for each entry in the received list, updating the expiry times of any services that have been renewed since they were last heard about, and adding any previously unknown ones. Lines 12 through 15 return the time range from which the **advertise** function should schedule its next advertisement, returning a

short range ($X'$) if any local services were missing or about to expire, and a later range ($X$) otherwise.

These functions do not specify how the lists should be stored, or how the broadcasts should be carried out, as these are internal and lower layer issues respectively. While the effects of using different versions of the **update** function on different devices will be explored in the Chapter 6 discussion on asymmetric behaviour, for the present discussion all devices will be assumed to agree on this implementation, and on the values for $X$ and $X'$.

If all devices have the same mean advertisement time, then a static system will reach a steady-state wherein SAMs are sent at slightly shorter intervals than that mean value. The chosen mean, therefore, should reflect the desired timeliness of the world views held by the various devices.

## 3.2 Related Work on Protocols

The DEAPspace[1] approach is reminiscent of pro-active route discovery protocols for ad-hoc networks, like the Intrazone Routing Protocol (IARP) included in the Zone Routing Protocol (ZRP) description[PH99], Optimized Link State Routing Protocol [JMQ00], or the route information base establishment system used in HIPERLAN [ETS96, §§6.5]; all of these examples transmit information proactively at regular intervals, allowing nodes to maintain continually updated information about their neighbours. These similarities are the result of the similar goals: route discovery is a special case of service discovery in which all SEs are adjacency tables.

### 3.2.1 Applications to Routing

Would DEAPspace be an appropriate platform for a new route discovery protocol? Probably not. One of the strengths of the DEAPspace algorithm is that it allows devices to discover services that might be partially hidden, thereby having unreliable broadcast connections, but feasible acknowledged connections. In routing, unreliable links are usually best avoided, so the fault-tolerance of DEAPspace becomes a weakness.

Furthermore, the internal representation does not scale well for routing tables. In service

[1]This protocol was developed as part of the DEAPspace (Distributed Embedded Application Platform device spaces) project at IBM Research, Zürich Laboratory.

discovery, each service is unique, and should be stored by all clients; in pro-active routing protocols, the whole table is unique. With a normal (e.g., distance vector) routing algorithm, as each table is received, it is used to update the local routing table, but is not explicitly stored. In order for one node to represent all of its neighbours, it would have to know the exact table to send thus, although the total network traffic would remain constant, the storage required would increase by a factor of the number of neighbouring nodes.

### 3.2.2 What can be learned from routing

Although these differences exist, the two areas of work still share the goal of allowing each node to inform its neighbouring nodes about the capabilities that it offers, whether those capabilities are reachable nodes or general services. It is not surprising then that design points similar to those used when designing DEAPspace led to some of the same architectural decisions, including the use of pro-active solutions. Pro-active routing systems fall roughly into two categories, referred to here as *regular* broadcast protocols (similar to HIPERLAN, in which all stations send periodic beacons [ETS96]), and *slotted* broadcast protocols (similar to IEEE 802.11, in which stations take turns sending beacons [IEE97]). For the purposes of service discovery, the significant content of these beacons is the address of the sending device.

The difference between the slotted and regular approaches lies in their scalability. As the number of devices gets large, a slotted protocol limits the amount of bandwidth being sacrificed to discovery, while a regular protocol limits the effect on timeliness. Slotted protocols also offer lower power consumption for servers, because their inherent predictability allows devices to make intelligent use of idle mode.

A sensible (and common) way to implement slotted access uses random contention for each broadcast slot. As a result, even if the network is not heavily loaded, statistics dictate that a device will sometimes miss several attempts in a row to broadcast, resulting in a longer interval between successive broadcasts than planned, and therefore a longer expected time to be discovered. In a regular system, every device broadcasts at the same interval, regardless of the other devices in the group. Because of this, the discovery time for devices in a regular system can be expected to be better than in a slotted system. It is therefore fair, when comparing timeliness of a new discovery algorithm with the timeliness of a beaconing algorithm, to use a regular beaconing algorithm as the basis for comparison.

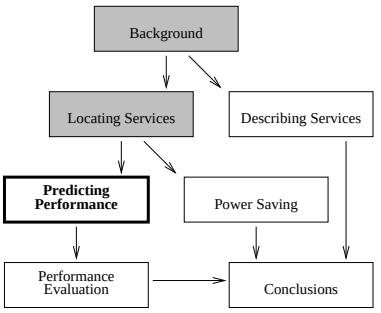
## 3.3 Chapter Summary

Chapter 3 has presented a new idea that is central to this thesis: an algorithm for service discovery in ad-hoc networks. It is a decentralized algorithm, and has been designed to perform well in quickly changing topologies. The following chapters will analyse the success of this design, first studying its expected performance, then comparing it with the alternatives.

# Chapter 4

# Predicting Performance

*Some theory is presented for predicting what behaviour can be expected from an implementation of the new algorithm. These calculations help to give an intuition into the choice of configuration variables to be used in tuning a deployed solution.*




Thus far, a problem has been presented, along with some existing work that can help towards a solution, and the details of a new solution were then given. It is common practice to compare network protocols via the use of simulations, and in fact this is what has been done, and will be presented in later chapters. Rather than going straight to simulation results, however, it is useful to spend some time explaining what results should be expected from those simulations, and why.

This chapter presents a theoretical analysis of the expected long-term behaviour of devices running the DEAPspace algorithm through discussion of its behaviour in an ideal environment, then by explaining how that behaviour changes in the presence of packet loss. These calculations are meant to help both with choosing configuration parameters and with validating simulations used in following chapters.

## 4.1 The Duration Of Steady-State

Consider first an ideal steady state for DEAPspace devices, in which some device broadcasts at time $t$. Ideally, all devices will receive that message, and schedule their next broadcast for some time from the range $X$ (as defined in Chapter 3). One of these choices will prove to be the earliest, and the device that made that choice will win the race, sending the next broadcast. Obviously, the expected duration for this round will depend on $X$. If all devices receive this broadcast also, then they will all reschedule again, and the scenario repeats.

Three things can disrupt this steady state: loss, device arrival (or departure), and broadcasts scheduled from the alternate range $X'$. The effects of packet loss will be addressed later in this chapter, and changing the group membership is the topic of Chapter 5. For now, consider reasons for a device to receive a service advertisement message (SAM) in which its own service elements (SEs) are expired or about to expire. When such a SAM is received, it triggers the device offering those SEs to choose its next broadcast time from the smaller timeout range $X'$. Call this act *worrying*.

To analyse the ocurrance of worry, we will start by assuming a nearly ideal environment, then look at the results of dropping various assumptions. For the first analysis, assume the following:

1. Devices neither arrive nor leave the group
2. Broadcast messages are instant and atomic
3. Broadcast messages are never lost or corrupted
4. If $n$ devices simultaneously schedule a broadcast (called an $n$*-way race*) at a time in the future chosen from range $X$, the race will have a unique winner at time $T$.

The strength of assumption 4 is that it lets us forget about time (for now) and deal exclusively with the number of rounds that pass. This assumption allows us to convert the time it takes a device to worry about its services expiring to a number of rounds. Let $b$ represent the number of rounds in which no other device is worrying that a device can let pass without renewing its services, and without worrying itself. This means that $b$ is a configuration parameter of the system. We can now restate the definition of worrying as what happens if, in steady state, any particular device loses $b + 1$ broadcast races in a row.

## 4.2 Solving for an ideal system

One of the simplest questions that we can ask about this ideal system is how long steady state can be expected to last. If we can determine this in terms of the configuration parameters, then users of the algorithm will be in a good position to choose useful values for their target environment.

To find the duration of steady state, under the assumptions listed above, start by writing down a regular expression for all possible sequences of events beginning by a particular device winning a broadcast race, and ending with $b + 1$ losses. The language defined by this regular grammar includes precisely all situations that can lead to worry. In this Equation 4.1, consider that 0 identifies a lost race, 1 identifies a won race, and exponents denote repetition (for an introduction to this notation, see Appendix A):

$$\mathcal{L} = [(\varepsilon + 0 + 0^2 + 0^3 + \cdots + 0^b)1]^* 0^{b+1} \tag{4.1}$$

For clarity, the term *fair race* refers to the situation where all devices simultaneously choose a time from the same range, and the device that chooses the earliest time "wins" the opportunity to broadcast next. In a group of $n$ devices, a given device can expect to win a fair race about 1 time out of $n$ in steady-state, since steady-state is defined as the situation in which all races are fair.

Equation 4.1 includes some sequence of wins and losses in which any run of losses is no longer than $b$, except for the final one, which is exactly $b + 1$ long. It is important to note that the language defined in Equation 4.1 is *non-ambiguous*, meaning that each possible sequence can be generated in exactly one way. Implicit in the definition of $\mathcal{L}$ is an initial win. Without that, the length of the first run of losses could not be known.

The following generating function[Rio58, Mac60] enumerates the strings of the regular language $\mathcal{L}$:

$$\begin{aligned} \Phi_n = & y^{b+1} + [(1 + y + y^2 + y^3 + \cdots + y^b)x]y^{b+1} \\ & + [(1 + y + y^2 + y^3 + \cdots + y^b)x]^2 y^{b+1} \\ & \cdots + [(1 + y + y^2 + y^3 + \cdots + y^b)x]^\infty y^{b+1} \end{aligned} \tag{4.2}$$

Equation 4.2 is a direct expansion of Equation 4.1, in which the power of $y$ counts the number of losses, and the power of $x$ counts the number of wins. For brevity, use the geometric

series compression $1 + y + y^2 + \cdots + y^b = \frac{(1-y^{b+1})}{1-y}$:

$$\Phi_n = \frac{y^{b+1}}{1 - \frac{x(1-y^{b+1})}{1-y}} \tag{4.3}$$

Equation 4.3 does not use a probability for the initial win, since that is a necessary precondition, and therefore has a probability of one.

Because we are interested in total length, introduce a new variable, $z$, to count the total number of trials (i.e., wins plus losses):

$$\Phi_n = \frac{(yz)^{b+1}}{1 - \frac{(xz)(1-(yz)^{b+1})}{1-(yz)}} \tag{4.4}$$

Now, in Equation 4.4, the power of $z$ is the total length of the string, the power of $y$ is the total number of races lost, and the power of $x$ is the total number of races won. For example, expanding $\Phi$ via Taylor series about zero in both $x$ and $y$ shows that if $b = 4$, the coefficient of $x^8y^{14}$ is $8800z^{22}$. The power of $z$ is just the total number of trials (8 wins + 14 losses = 22 trials), and the coefficient is the number of ways that 8 wins and 14 losses can be arranged such that they end with a win and five losses (in that order), and contain no other runs of losses with length greater than four. As an example that you can verify in your head, also notice for $b = 5$ the coefficient of $x^8y^7$ is $8z^{15}$, since six of the $y$'s are at the end, and the final $y$ can be located before any of the eight $x$'s.

We have now enumerated all the strings. The next step is to start working towards the expected length. Each term already has $x$ to the power of the number of wins, and $y$ to the power of the number of losses, so the probability of the situation described by that term is right there, and the coefficient automatically handles the multiplicity of each probability, but what good is a sum of probabilities? Substitute any $x + y = 1$, $z = 1$ into the function, and you find that the sum of all probabilities is $1.0000000$ meaning, not surprisingly, that eventually a device will lose $b + 1$ times in a row for any valid $b$.

To get the expected duration, the probability of each possible string is multiplied by its length. (That's just the definition of "expected value," as found in any introductory probability book.) So, we differentiate with respect to $z$. Remember that the exponent on $z$ is the length of the string, and that differentiation will multiply every term by the exponent of $z$. After differentiating, we set $z$ to one, to cancel it out of the whole expression, since we won't be needing it any more.

$$\frac{\partial}{\partial z}\Phi_n|_{z=1} = \frac{y^{b+1}(b+1)}{1-\frac{x(1-y^{b+1})}{1-y}} - \frac{y^{b+1}\left(-\frac{x(1-y^{b+1})}{1-y} + \frac{xy^{b+1}(b+1)}{1-y} - \frac{x(1-y^{b+1})y}{(1-y)^2}\right)}{\left(1-\frac{x(1-y^{b+1})}{1-y}\right)^2} \tag{4.5}$$

Now that every term is multiplied by the length of the string that it represents, all we need to do is substitute the probability of winning for $x$, and the probability of losing for $y$. Obviously, the probability of losing is one minus the probability of winning, so substitute $y = 1 - x$ into Equation 4.5:

$$E(x) = \frac{\partial}{\partial z}\Phi_n|_{z=1,y=1-x} = \frac{1}{x(1-x)^{b+1}} - \frac{1}{x} \tag{4.6}$$

By solving Equation 4.6 for $x = \frac{1}{n}$, we can now see that, in a steady-state environment with $n$ devices, an individual device would expect to worry about once every $\frac{n^{b+2}}{(n-1)^{b+1}} - n$ rounds.

## 4.3 Considering the total system

Unfortunately, when a device worries, it causes the next round not to be a fair race. By picking a timeout from the lower range, all devices that are not already worried are guaranteed (by our current assumptions) to lose the next race. If the value predicted by Equation 4.6 is large, then this effect will just be noise, and should not really cause any trouble. If worrying is a relatively frequent occurrence, however, the unfair races must be accounted for in calculating the expected duration of steady-state.

To deal with this feedback problem, assume the probability $x$ of a particular device winning a given race is known. Equation 4.6 can be used to turn this into an expected number of rounds, $E(x)$. By inverting $E(x)$, we get the probability of worrying on any given round. Therefore, the probability of $n$ devices all not worrying on a particular round is $(1 - E(x)^{-1})^{n-1}$. The exponent is $n-1$, rather than $n$, because we are analysing the duration steady state, so we know that at least one device (the one for which the steady state is being analysed) must not be in a state of worry (from the definition of steady state).

The probability of a particular device winning a fair race is $n^{-1}$. The probability of a particular round being fair, given that at least one device (the one we are considering) is not currently worrying, is $(1 - E(x)^{-1})^{n-1}$. By combining these two probabilities, we find the probability of

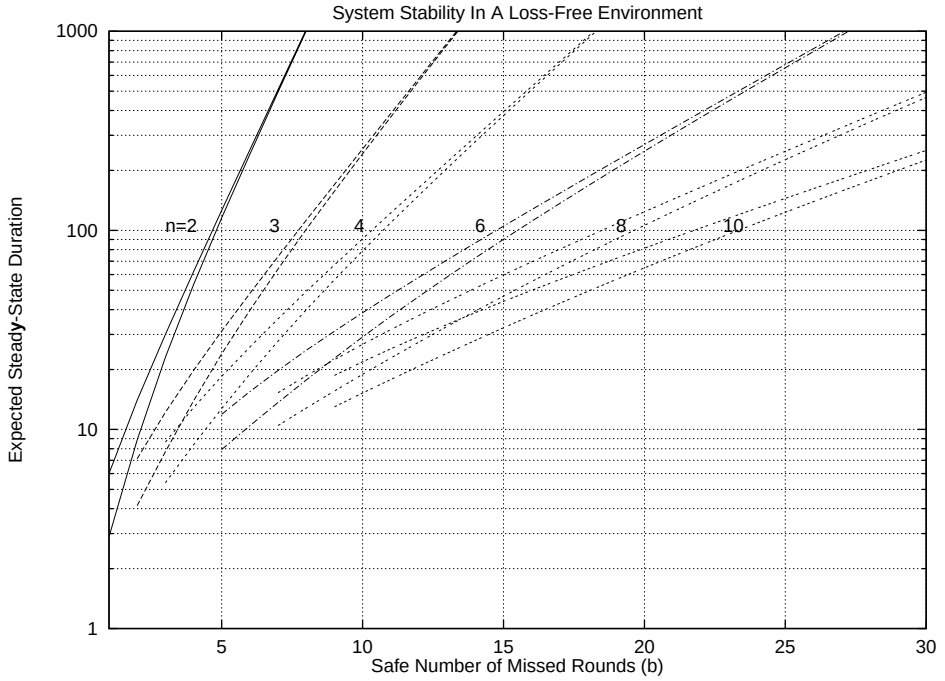



**Figure 4.1:** Corrected and uncorrected expected steady-state duration (on the logarithmic y-axis) against the number of consecutive rounds for which an individual station can fail to send a renewal advertisement without affecting its choice of broadcast timeout ($b$, on the x-axis) for two through ten devices ($n$). In each pair, the lower curve has used Equation 4.7 to get a more accurate result.

a particular device winning a given round is given by Equation 4.7.

$$\begin{aligned} x &= \left(\frac{1}{n}\right)\left(1 - \frac{1}{E(x)}\right)^{n-1} \\ &= \left(\frac{1}{n}\right)\left(1 - \left(\frac{1}{\frac{1}{x(1-x)^{b+1}} - \frac{1}{x}}\right)\right)^{n-1} \end{aligned} \tag{4.7}$$

General solutions to Equation 4.7, giving $x$ in terms of $b$ and $n$, are difficult to write down, let alone work with, so a little function (Appendix B) was written to approximate its solution using Newton's method. Let $\tilde{x}$ represent the value of $x$ that satisfies Equation 4.7.

Substituting the solutions to Equation 4.7 back into Equation 4.6 gives a realistic expected duration for steady-state, given values for $b$ and $n$. Naturally, as $b$ gets large, Equation 4.6 gets large, and Equation 4.7 approaches $\tilde{x} = \frac{1}{n}$. That is to say that as devices worry less frequently, they cause fewer unfair races, so assuming all races to be fair is increasingly valid for larger values of $b$. This convergence is shown in Figure 4.1, which plots $E(\frac{1}{n})$ and the corrected $E(\tilde{x})$.

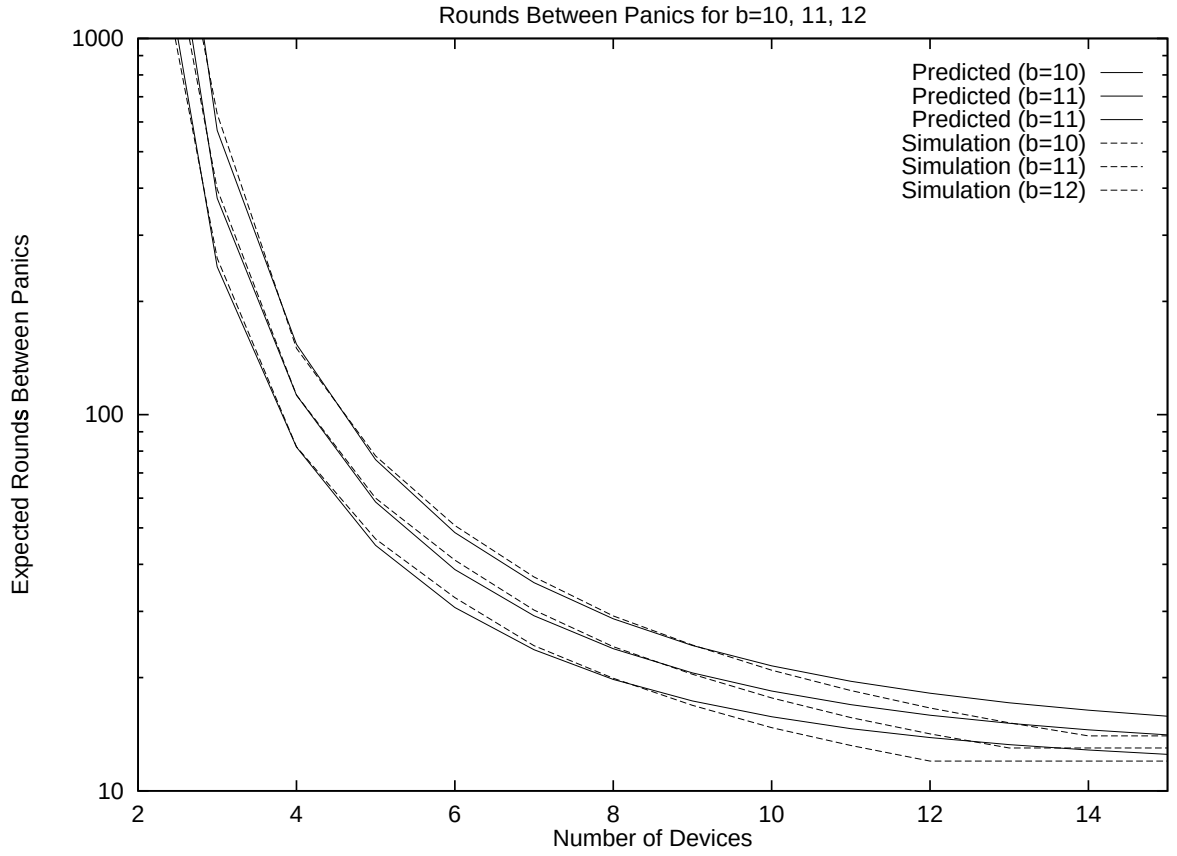



**Figure 4.2:** Comparing the results predicted in Equation 4.7 with the observed behaviour of a discrete event simulation.

Equation 4.7 now allows an "ideal timeout" (one that will allow steady-state to be maintained during almost all of any given execution time) to be calculated. A system-designer can choose the largest number of devices expected to be joined into one group, define what a long time is, and work backwards to find a suitable choice for $b$. For example, if groups are not expected usually to involve more than five members, and would like steady-state to last at least a hundred rounds between incidences of worrying, Figure 4.1 shows that $b = 14$ would be sufficient. In fact if, in the case stated above, the expiry time is set to allow fourteen rounds of non-renewal to pass before devices start to worry, devices can expect to go some 122 rounds between instances of worry.

Figure 4.2 shows predicted values against the values observed in a discrete event simulation. The predicted curve is a good approximation of the observed behaviour until the number of devices gets close to $b$. This is to be expected, since this situation will mean that some device is always worrying, meaning that there is a chance of a device choosing a short timeout, but still

not winning the next race. Simulated behaviour will be discussed in more detail in Chapter 5.

This result is both good and bad. On the good side, it allows us to study the behaviour of the system under the assumption that a steady-state can exist, and occurrences like lost packets and the arrival of a new device can be treated as perturbances of that steady-state. On the not-so-good side, the above example showed that for five devices to maintain steady-state more or less indefinitely under ideal conditions will require timeouts on the order of twenty rounds. That means that the time to notice the absence of a device is about twenty times longer than the time to notice its presence. For such a small example-system, better performance should be expected.

Happily, this is not the end of the story. Causing a device to worry and send a message earlier than usual is not so bad. Yes, if this is a frequent occurrence, the average offered network load will be slightly higher, but is that bad? Section 4.4 addresses that question.

## 4.4 When Devices Worry

Section 4.1 defined "worrying" to describe the situation when a device sees an advertisement in which one or more of its services is absent or scheduled to expire in less than a given time, and Equation 4.7 was presented as a way to predict the time between instances of devices worrying. Now we address the question of what happens when a worrying situation arises.

When a device worries, it chooses its next scheduled broadcast-time from an earlier range than usual. It makes analysis easier if there is no overlap between the usual range $X$, and the shorter range $X'$, so let's go with that assumption for now. Assume also, for now, that no packets are lost.

Under these assumptions, even if a device worries only one round before its services expire, no unusual behaviour will be exhibited, except for one shorter-than-usual broadcast round. Consider the case where $X = [10, 15]$, $X' = [8, 10)$, and $n = 5$ devices. Assume that, from the way an entry from $X$ is chosen, the usual lowest choice $\tilde{X}$ is twelve. The system continues with devices broadcasting every twelve seconds or so, until device $A$ receives a broadcast showing its services will expire in fifteen seconds. $A$ worries, and chooses its next broadcast time from $X'$; this time it chooses nine seconds. At the same time that $A$ chose that timeout, the other four devices also choose their next broadcast times (since they are synchronised by

the receipt of the same broadcast), but those times are all chosen from the interval $[10, 15]$. Naturally, nine seconds later, $A$ wins the broadcast race; all five devices then synchronise on the new broadcast, and schedule their next broadcast for ten to fifteen seconds in the future. An observer would have to be paying very close attention to see anything unusual in the observable behaviour caused by this whole process.

If, as in the above situation, devices begin to worry only one round before their services expire, and two devices begin worrying as a result of the same broadcast packet, only one will renew its services in time. But how often will this happen? If steady state has been continuous for more than $b$ rounds, then each device has had at least one broadcast separated $\tilde{X}$ from any other broadcast. Remember, we are still considering only a loss-free network, and uniform assignment of configuration parameters among the devices so, for one round of worrying to be insufficient, two devices would both have to lose the same $b$ rounds in a row, after broadcasting consecutively, separated by less time than the most recent winning timeout value (causing them to begin worrying as a result of receiving the same broadcast packet.)

Although we are assuming that all rounds are the same duration, there is of course some variation in the exact time between broadcasts, with some rounds being slightly longer than others. This is relevant because, for two devices to worry as a result of receiving the same packet, their expiry times must have both crossed their worry threshold in the time since the previous packet was sent. Therefore, the gap between their renewals (some $b$ rounds earlier) must have been less than the gap between the current broadcast and the previous. This condition may come about due to a previous case of worry, or just due to the nature of random number selection, but there is enough randomness in the whole process that it is reasonable to say that it's about an even chance whether any particular gap between transmissions is longer or shorter than any particular preceding gap. This gives us the factor of $\frac{1}{2}$ in Equation 4.8.

The only other thing that must occur, given the correct timing, is for the winners of the race $b$ rounds and $b + 1$ rounds in the past to be different, and for these particular two devices both to have lost the previous $b$ rounds. Because we are starting from steady state, and devices are uniformly configured, this is $\frac{n-1}{n}$ (the chance that they are different) times the number of ways to choose $b$ winners from $n - 2$ devices divided by the number of ways to choose $b$ winners from $n$ devices.

Putting this all together gives us the following:

$$\left(\frac{1}{2}\right)\left(\frac{n-1}{n}\right)\left(\frac{n-2}{n}\right)^{b} \tag{4.8}$$

Consider again the example from the end of Section 4.1, in which five devices were present, and $b$ was set at 14, meaning that more than a hundred rounds of steady-state would be expected to pass before any device worried. The probability of this worrying actually causing any significant effect is, by Expression 4.8, $(\frac{4}{10})(\frac{3}{5})^{14}$ or 0.03%. This result means that, even given the necessary conditions for trouble, there is less than one chance in 3190 that anything bad will result.

In the current example, the smallest advertisement interval ever chosen is eight seconds, and the largest is fifteen. Since this means that three advertisements can never have happened in an interval shorter than any single advertisement interval, a maximum of two devices could ever be caused to worry as the result of a single broadcast. (Again ignoring the possibility of two preexisting groups merging very quickly. This and other unusual events will be discussed later.)

To incorporate the earlier timing estimates into the probability of having two devices begin to worry on the same round, we have now seen enough to begin with the claim that it is a rare event. Because it is rare, we need not consider the feedback problems encountered with Equation 4.6. Furthermore, if one device is going to worry on a given round, then we know it did not broadcast on any of the $b$ rounds leading up to that round. Therefore, the probability of a second device having lost the same $b$ fair races is actually lower than the probability of a device losing an arbitrary $b$ fair races. This observation, considered implicitly in the formation of Expression 4.8, would lead to an expected probability of two devices worrying on any particular round to be given by the following:

$$P_d = \left(\frac{1}{2}\right)\left(\frac{n-1}{n}\right)\left(\frac{1}{\Phi_n(\tilde{x})\Phi_{n-1}(\tilde{x})}\right) \tag{4.9}$$

Calculating a value for Equation 4.9 with our ongoing example of five devices gives about a $0.01\%$ chance (per round) of any particular device remaining worried for more than one round if $b$ is ten. Furthermore, recall that a round of worrying is shorter than a normal round, so configuring worry to start, in this case, sixteen seconds before expiry would give the device a second chance to broadcast before expiring.

### 4.4.1 Approximating the round duration

At the least, it is reasonable to consider instances of double worry to be rare. Therefore, in the following calculations, consider the duration of a round of advertising to be entirely determined by the meta-stable behaviour comprising normal timeouts and occasional worrying. This implies that a worrying device will always win its race, and each device has $\Phi_n(\tilde{x})$ rounds of non-worry for each round of worry, during which rounds of non-worry, the other $n-1$ devices will worry about once each. Therefore, the average round duration $T$ can be approximated by taking the weighted average of $E_n(\tilde{x})+1-n$ rounds with duration $\tilde{X}$, and $n$ rounds with duration $\tilde{X}'$:

$$T \approx \frac{\tilde{X}(E_n(\tilde{x})+1-n)+\tilde{X}'n}{E_n(\tilde{x})+1} \tag{4.10}$$

In our ongoing example of five devices, using the recently chosen parameter $b=10$, this gives an expected round duration of $\frac{12\times(45+1-5)+9(5)}{45+1}=11.7$ seconds.

These calculations have shown the algorithm to be much more robust than the preliminary results showed in Section 4.1. We now have a group of $n$ devices set up so that each device expects to go for tens of thousands of advertising rounds (in the case of eleven-second rounds, that means days) without ever expiring. It is therefore reasonable for further study of system behaviour to assume that steady-state will be maintained unless packets are lost, or the system configuration is changed.

We will now consider the robustness of the system in the presence of packet-loss.

## 4.5 When Packets Go Missing

This algorithm is very tolerant to packet loss and, in most cases, the loss of a broadcast packet will cause no unusual behaviour at all. If a broadcast is lost before the sending device is worried about timing out, even if the sending device knows it is lost, that device's behaviour will not be affected.

If the lost packet is sent by a device that is starting to worry, it is slightly more serious, but not much. Section 4.1 shows that parameters can be chosen such that devices will very rarely have to choose a value other than from their usual range, implying that when one does so, it will almost always succeed in winning the next race, since it will usually be the only device choosing

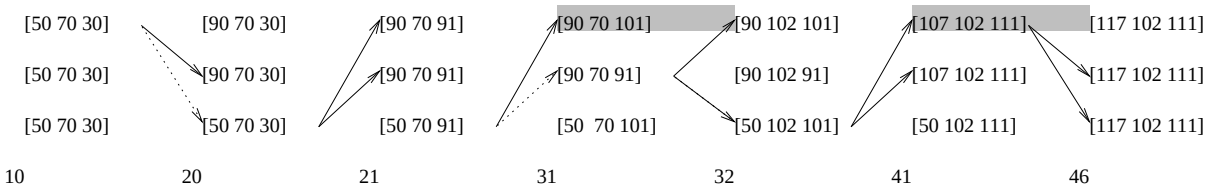



**Figure 4.3:** Vector elements $[a\ b\ c]$ represent the expiry times known by each device for the top, middle, and bottom devices respectively. Shaded areas indicate that the indicated device is in a state of worry. In this case, expiry is set 70 seconds in the future, and devices worry if they see their own expiry-time advertised less than 20 seconds in the future. To ease explanation, the winner always chose 10 seconds, and the other devices all chose 11 seconds. A device that worried always chose 5 seconds. Note that these values are not part of the algorithm, but were used here only to avoid the confusion that using random values might cause. The top device almost expires from the bottom device's list, but only after the bottom device failed to receive three advertisements in a row. The top device worried at time 31, when it received an advertisement showing its own imminent expiry at time 50, but stopped worrying when the middle device transmitted (at time 32) with a later expiry shown for the top device. On receiving the broadcast at time 32, the top device reset its advertisement timer to 42 (from 36).

a timeout from the lower range. Even if that advertisement is lost, it will almost certainly not be lost to all devices. If the next device to broadcast received the packet successfully, then its broadcast will update the device that missed it (unless that device has dropped out completely, which case is covered next). If the next device to broadcast did miss it, but there are at least three rounds between worrying and actually expiring, then the situation will result in another short timeout when the old list again causes the same device(s) to worry.

A key design-point of the DEAPspace algorithm is its stability. A small number of received broadcasts can make up for a large number of lost broadcasts. Because of this property, the more devices join a group, the more reliable the service discovery is. This property is demonstrated in the example of Figure 4.3.

## 4.6 Chapter Summary

This chapter has presented an analysis of the generally expected behaviour of the DEAPspace algorithm. This analysis was originally developed as a feasibility test, assisting with the decision

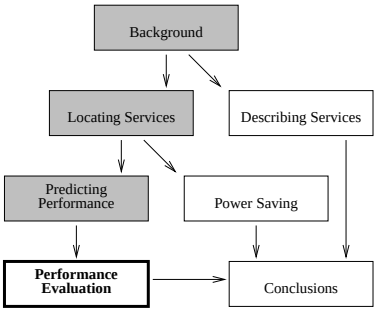
of whether or not the algorithm was worth an implemented trial. In this respect it was successful, predicting reasonable discovery times and scalability, and therefore warranting further study. The goals of this further work would clearly include finding answers to the sorts of complicated situations left unresolved in Section 4.5; asking questions about events that, while comparatively infrequent, may affect the usability of the algorithm in practical contexts.

To answer these types of questions, the usual solution lies with simulation; and this thesis is no exception. The next chapter will introduce a simulation of the DEAPspace algorithm, and use it to examine some more complex environments. It will first demonstrate simulations of algorithm behaviour under ideal (loss-free, uniform parameter distribution) conditions. Having introduced the output format and demonstrated the accuracy of the simulator, it will go on to present some interesting environments and the behaviour of the algorithm in those environments.

# Chapter 5

# Performance Evaluation

*This chapter introduces two tools that were used for analysing the behaviour of the DEAPspace algorithm under real conditions. The tools are used for mutual validation, then validated against theory. The performance of the new algorithm is then compared with that of competitive alternatives under both normal and extreme conditions.*




---

Chapter 3 described a new service discovery algorithm, and Chapter 4 explained that, due to its changing timeout values and race conditions, it is very difficult to analyse this algorithm thoroughly, so simulation is required for proper examination. For this examination, two tools have been created. One is a network emulator that presents an interface interchangeable with an actual network interface coded for the Java VM. The emulator has adjustable packet-loss probabilities and propagation delays, allowing the applications to be developed and tested in a controlled environment before "going live." The emulator also provides a central point from which information about network access can reliably be collected. Because interchangeable interfaces are allowed for, the implementation can be tested over real networks to verify that no unexpected behaviour emerges that was somehow not seen by emulation.

The second tool is a hand-coded discrete event simulation designed uniquely for service discovery, included in Appendix C. Because the emulator mimics a real network, tests of thousands of rounds take hours to complete, while a discrete event simulation can be completed for

many more rounds in minutes. This allowed far more trials, and therefore more reliable and reproducible results, than would have been possible from emulation alone.

## 5.1 Simulations

The network emulator was written for testing various service discovery ideas, and has proven to be a very useful tool. It offers real-time testing of behaviour with real clients over varying network conditions, and then allows those exact same clients to be connected to a real network to verify their behaviour. Its strength, however, is also its weakness. Being a real-time emulator, doing a large number of iterations is difficult. Also, it introduces some unpredictable behaviour that is consistent with the specific types of networks under study, but is not generally expected for an arbitrary client and network.

For reasons of generality, most of this chapter uses a discrete event simulator that was coded specifically for service discovery. By ignoring latency caused by throughput rate, queuing delays at the various clients, and network access time, it produces a more reproducible, and therefore more reasonably compared result.

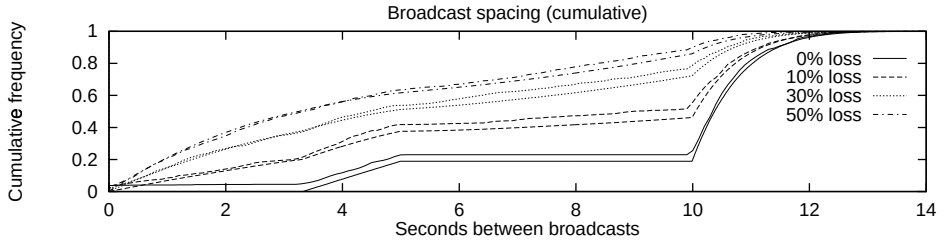
Before going directly into using the results of these simulations, it is useful to present first how they differ from emulated performance.

As demonstrated in Figure 5.1, the emulated behaviour is very similar, but not exactly the same as the simulated performance. This difference comes primarily from two sources:

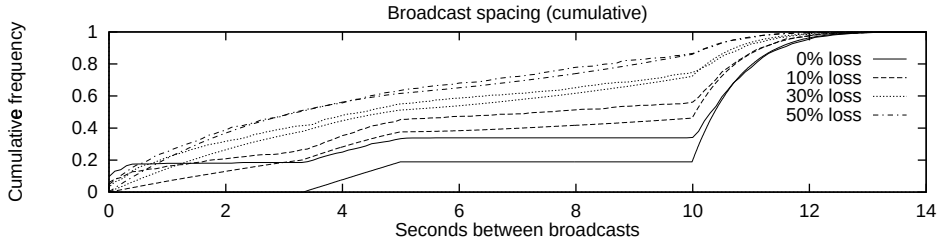
- Thread scheduling
- Nearly simultaneous transmissions

An example of thread scheduling is that, even with zero packet loss, some rounds are observed with a duration between $X'_{max}$ and $X_{min}$ in the emulated performance. This should be impossible but, because sometimes the discovery thread doesn't happen to get any processor cycles for an unusually long time, it may interpret a broadcast as being sent seconds later than it actually was. This leads to theoretically impossible round durations being observed. Such scheduling difficulties are unavoidable in any real-time emulation where the behaviour of many parallel processors is being emulated by a single serial processor.

A second way in which emulated performance differs from theory is that one thread may send an advertisement to the network while another advertisement is being received. The simulator assumes that as soon as one device decides to send an advertisement, all devices that are




**(a)** Emulated network data transmission speed instantaneous



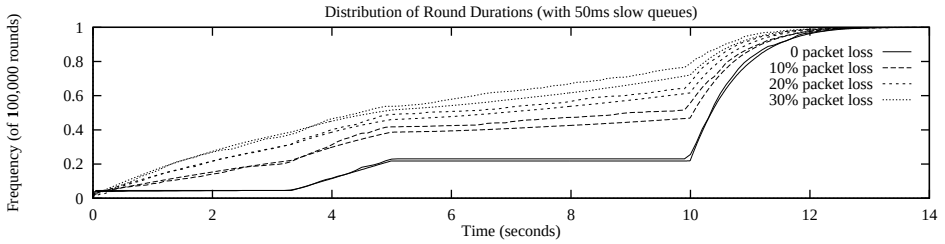

**(b)** Emulated network data transmission speed 1 Mbps

**Figure 5.1:** Advertisement round duration spread, simulated and emulated, with 6 devices, $X = [10, 15], X' = [3.33, 5]$. In both examples, the smoother (lower) line of each pair represents a simulation of 100,000 rounds, and the other is 1,000 emulated rounds.

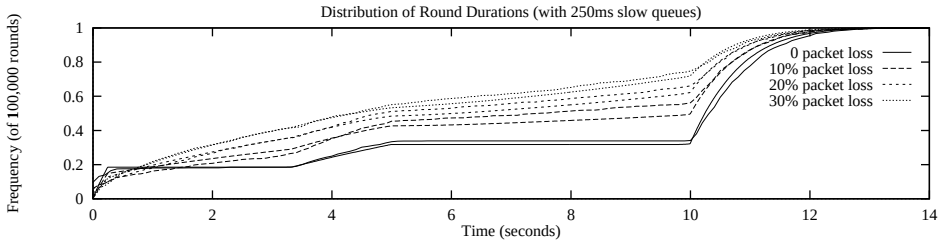
going to receive it do so immediately (i.e., all races have a unique winner). In a real implementation, if two devices decide to advertise at almost exactly the same time, the network layer and MAC protocols may avoid collision, but both packets will still be sent. This leads to some very short rounds, which is why the y-intercept for the simulated curves is zero, but not for the emulated curves. Modifying the simulation to account for this effect, as shown in Figure 5.2, demonstrates further the convergence of results between the emulator and the simulator.

These differences are the reason for using an emulator, since it can bring attention to issues that might be missed by a purely theoretical argument. Such practical difficulties, however, might be solved by considering them in the design of the network layer. Thread scheduling difficulties can be bypassed by timestamping packets before passing them to the transport layer, and simultaneous transmissions can be bypassed by allowing a cancel condition to be described to the network layer when transmitting. Such solutions might not be considered elegant, but they would be possible.

By using the simulator, not only can many more trials be considered, but the qualities shown




**(a)** Emulated network data transmission speed instant, simulated overlap time 50 ms.




**(b)** Emulated network data transmission speed 1 Mbps, simulated overlap time 250 ms.

**Figure 5.2:** Advertisement round duration spread, as in Figure 5.1, except the simulator allows packets to be already in the send queue when another is received. The amount of overlap time allowed is much more in the case of 1 Mbps traffic because the duration allows for more drift in the processor time allocated to each emulated device.

are fundamental to the algorithm described, and are therefore of more interest than artifacts of interactions between communication layers and the hardware platform.

## 5.2 Fidelity

In this first set of examples, the emulator was used to study devices that were always in one of two states: present or absent. The packet loss probability for communication between an "absent" device and any other device is 100%. Between two "present" devices, packet loss probability was held constant through each individual trial. Packet loss was considered independently, meaning that if the loss probability was 10%, then a broadcast would reach (on average) about 90% of available devices, rather than having a 10% chance of being lost completely.

The goal of these trials was to demonstrate the reliability of the algorithm as compared with an on-demand solution. It was compared with the basic discovery algorithm of broadcasting a

request, then waiting for the desired service to reply. This latter algorithm, being much easier to examine theoretically, also helped to validate the emulator results.

### 5.2.1 Environment and Tasks

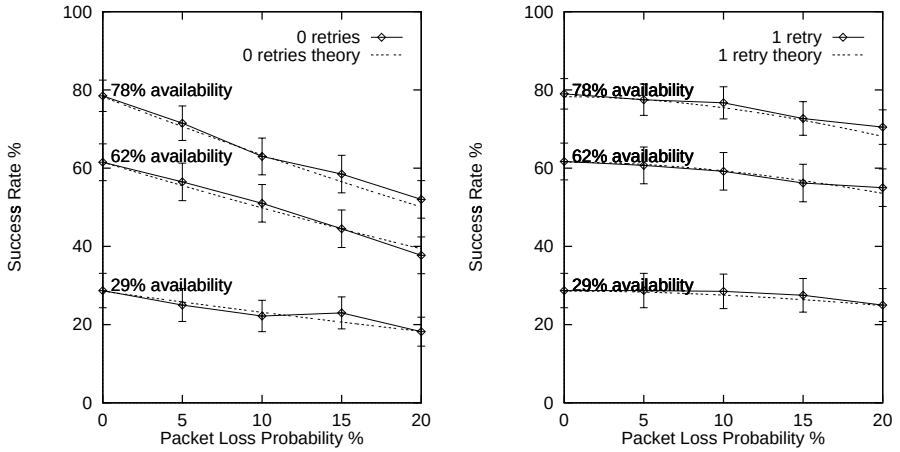
The environment contained six servers and one client. Each server offered exactly one service, and each service was unique. Each server was "absent" for one five minute period once per cycle, with a cycle time unique to each server (7, 11, 13, 17, 19, or 23 minutes). The client requested one service every 27 seconds, thereby requesting each particular service once every 162 seconds. When required (for the DEAPspace algorithm) expiry times for services were one minute, so no caching of previous successful discoveries was allowed. When using the DEAPspace algorithm, the advertisement timer was chosen (with a flat distribution) from the range $[\max(0, t-5), t]$ where $t = \min(15, t')$, $t$ measured in seconds, and $t'$ being the earliest expiry time of the device's own service in the advertisement most recently received. Once a service was discovered, data was exchanged. Synchronous communication (between "present" devices) was reliable.

### 5.2.2 Results

Consider first the basic discovery algorithm, which succeeds in discovering a service if the broadcast is received by a device providing that service, and the subsequent reply is also received by the initial requester. For packet loss probability $p$, and probability $q$ that the service is present at all, this means the success probability is $q(1-p)^2$. One improvement on the basic algorithm is to allow retries. If a request is repeated up to $r$ times, this leads to a new probability of success, as shown in Equation 5.1:

$$\begin{aligned} q(1-p)^2 \sum_{i=0}^{r} (1-(1-p)^2)^i &= q(1-p)^2 \left[ \frac{1-(1-(1-p)^2)^{r+1}}{1-(1-(1-p)^2)} \right] \\ &= q(1-(1-(1-p)^2)^{r+1}) \end{aligned} \tag{5.1}$$

The limiting case of this more detailed analysis, $r = 0$, is the same as the case first presented: success probability $q(1-p)^2$. Note that retries actually improve the success probability slightly more than this, since they stretch the transaction over a longer period of time, so the service might become available between the first and second attempt. This property of $q$ is ignored in this analysis.




**(a)** no retries **(b)** one retry

**Figure 5.3:** Emulated vs. theoretical results for the revised basic discovery algorithm. Error bars are given for 95% confidence interval.

An implementation of this on-demand algorithm was run on the emulator, with the results shown in Figure 5.3.

Having demonstrated the emulator behaviour to be consistent with the network model, an implementation the DEAPspace algorithm was installed. The observed behaviour, shown in Figure 5.4, illustrates the performance of this algorithm against both the basic algorithm and against the ideal result, that being a success rate equal to the actual probability that the service is present.

The charts in Figure 5.4 show not only that the DEAPspace algorithm gives at least as timely a picture as the basic algorithm, but also that it is less affected by network unreliability. Moreover, the result from the DEAPspace algorithm is not significantly different from the ideal (success rate equals actual availability) even for packet-loss rates more than 40%.

## 5.3 Comparing Push Models

What we have seen so far has demonstrated that the DEAPspace algorithm is better than on-demand service discovery in at least some ways. Arguments for considering the relative power costs of on-demand and proactive discovery techniques will be covered in Chapter 6, but first let's compare different proactive solutions. The goal of these comparisons is to demonstrate

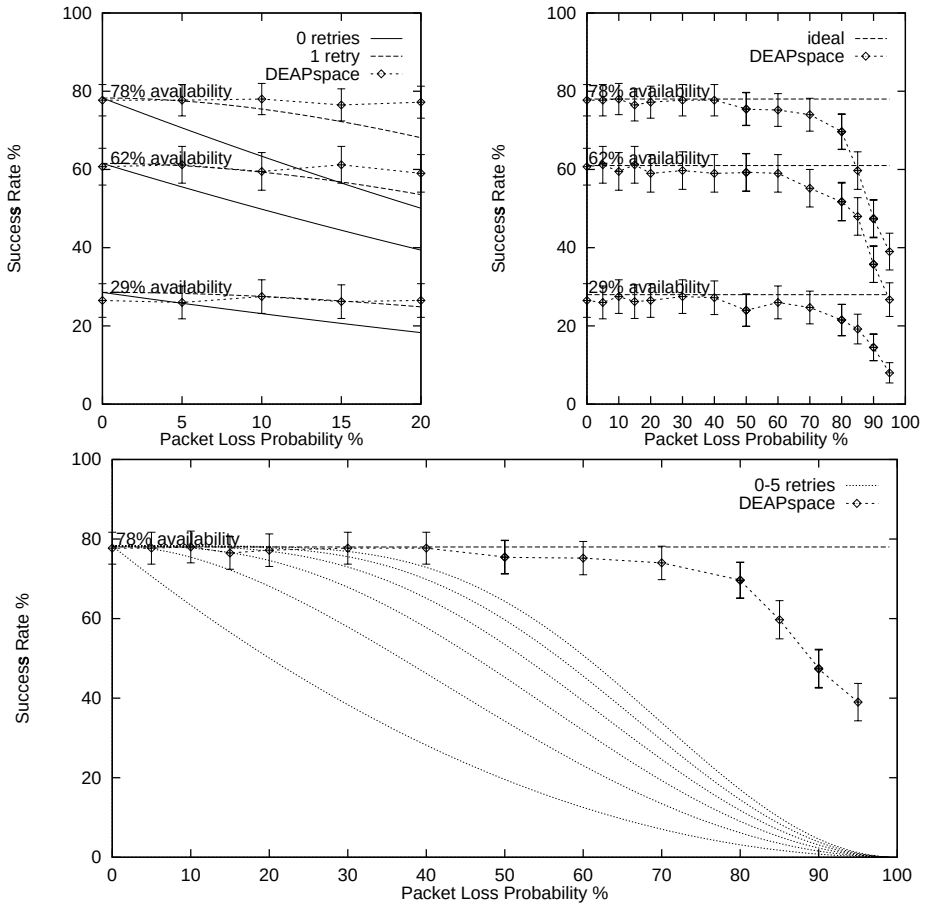



**Figure 5.4:** Comparing the DEAPspace algorithm with, on the **Upper Left**, theoretical results for the basic algorithm and, on the **Upper Right**, the ideal result. The **Lower** graph compares ideal, the basic discovery algorithm for zero through five retries, and the DEAPspace results over a full range of packet loss conditions. Error bars are given for 95% confidence interval.

why DEAPspace is better than similar push-model solutions already in use.

The argument in favour of the DEAPspace algorithm begins with the claim that the bandwidth required for the DEAPspace algorithm to send one broadcast with $n$ SEs is about the same as that required for the regular algorithm to send $n$ broadcasts with one SE each. This equivalence is justified by the fact that there is at least some overhead associated with sending a broadcast packet, even if that overhead is only the packet-header containing a broadcast address, and this overhead is at least as expensive as the timestamp required for each SE in DEAPspace. The performance comparisons presented in this section are for configurations that offer comparable loads (in terms of total bytes per minute) to the underlying network. Because the load caused by regular beaconing is directly configurable, this was accomplished by choosing parameters for the DEAPspace algorithm, observing the average (simulated) time between broadcasts, then using that as the period for the regular beaconing algorithm to which the DEAPspace algorithm is compared.

Having established an equal footing for the alternatives, we can now address the claim that the time for acquisition (discovery) of available services is better with DEAPspace than with either slotted or regular schemes. This improved performance is achieved because more information is being conveyed when one device broadcasts its world view than when each device broadcasts its own information. The additional information conveyed to the member devices is a snapshot of what is known about them by at least one other member of the network.

Suppose we tune both algorithms' configuration parameters to have equal network load with 10% (uncorrelated) packet loss probability, and groups of up to six devices. One measure for the effectiveness of these discovery algorithms is what happens when the sixth device encounters an existing group of five others. For the sake of simplicity, consider only the regular and DEAPspace schemes; in general, regular schemes offer faster discovery than slotted ones, so this is a reasonable comparison.

For the regular broadcast scheme, as for the slotted, a period $T$ can easily be established for the expected time between advertisement repetitions. In the case at hand, some quick math shows that the probability of all five getting the first broadcast is $(1 - 0.1)^5 = 59\%$, by the second broadcast, it is $(1 - (0.1)^2)^5 = 95\%$, and after three broadcasts, it is 99.5%. In the other direction, the new device will learn about all five existing devices with the same probability distribution, except it will usually take slightly longer for all devices to have had a turn at

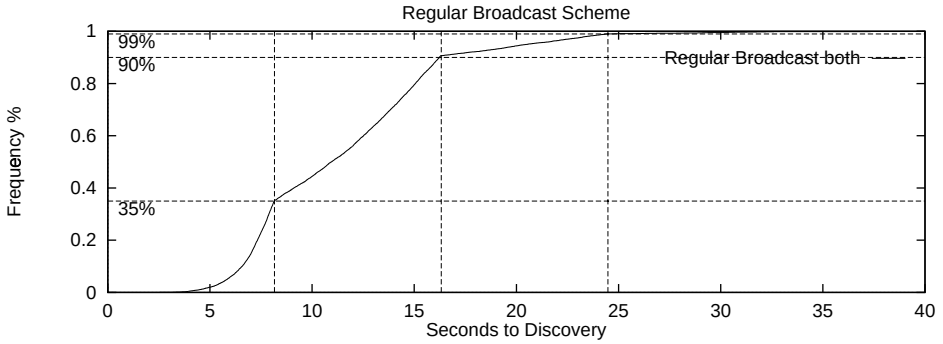



**Figure 5.5:** Cumulative frequency for mutual discovery times using the regular broadcast algorithm over a simulated 100,000 trials for one new device joining an existing group of five, compared with theoretical predictions ($T$=8.16 seconds).

broadcasting. This makes the time for mutual discovery about $0.59^2 = 35\%$ less than $T$, $0.95^2 = 90\%$ less than $2T$, and $0.995^2 = 99\%$ less than $3T$. Comparing these with the simulation results for regular broadcasting in Figure 5.5 shows agreement so far.

Now consider DEAPspace under the same conditions, with the following notation and parameter restrictions (to ease examination):

- Let $A$ represent the newly arriving device, $G$ represent the group of five devices already in steady state, and $G_i$ represent some particular member of $G$.
- The range of $X$ ($X_{\max} - X_{\min}$) is less than $X'_{\min}$. (This means that a device that misses a broadcast, and therefore does not reset its timer, will send its out-of-sync broadcast before the next new round starts.)
- The average values chosen from $X$ and $X'$ by a single device are $\overline{X}$ and $\overline{X'}$ respectively.
- The average lowest value chosen from $X$ out of 5 choices is $\tilde{X}$.

Initially, both $A$ and $G$ are broadcasting periodically, with independent periods. Once they have come within range of each other, either $A$ or some $G_i$ will transmit first. Under ideal conditions without packet loss, one will transmit, the other will receive the transmission, choose a timeout from $X'$, transmit next, and that transmission will also be received, resulting in all devices having an accurate world view. With 10% packet loss, this will succeed in about $0.9 \times 0.9^5 = 53\%$ of cases and can be expected to take about $\frac{1}{3}\overline{X} + \overline{X'}$ time[1].

[1]The expected minimum for two random variables uniformly distributed on [0,1] is $\frac{1}{3}$, so the expected time

This behaviour is confirmed by the simulation results shown in Figure 5.6, which compares the probabilities of event $E = \{$*all devices have an accurate world view*$\}$ as a function of time for the regular and the DEAPspace algorithm. The results have been obtained with 100,000 simulation runs assuming 10% packet loss (independent loss probability for each receiver of each message). The parameters for the DEAPspace algorithm were chosen as $X = [12, 15]$, $X' = [4, 5]$, NormalExpiry = 115 seconds, and MinExpiry = 15 seconds, resulting in an observed average time between advertisements of 8.16 seconds. The advertisement period of the regular algorithm was chosen equal to this average of 8.16 seconds to make the comparison fair in terms of network load. In this simulation, five devices were first allowed to exist together for some time, then a new device was introduced at a random time. Time was measured from when the new device ($A$) was first able to communicate with the existing group ($G$) until all devices had discovered all others (event $E$). The DEAPspace algorithm achieves $E$ at about the same rate as the regular algorithm. Notice the good agreement of the simulated DEAPspace curve and the analytically derived intersection of ordinate $\frac{1}{3}\overline{X} + \overline{X'} = 9$ and abscissa $P(E) = 0.53$, as well as the regular curve at $T$, $2T$, and $3T$.

To analyse the remaining 47% of DEAPspace cases, note first some properties of the environment under consideration:

- At least one member of $G$ will virtually always ($1 - 0.1^5 = 99.999\%$) hear a broadcast from $A$ (assuming that loss is independent), but all members of $G$ will hear any particular broadcast from $A$ only about $0.9^5 = 59\%$ of the time.

- In general, if one or more devices does not receive a transmission, one of those devices will be the next to transmit, because they will not have reset their timers. (Infrequent exceptions to this rule exist, but require several devices to be worrying at the same time, so they will not be significant.)

- During time $X_{\max}$ following a transmission by any device, all devices will have either sent or received another transmission.

For so long as $A$ fails to receive the broadcasts from $G$, it will continue to broadcast its own local list with a period of $\overline{X}$. Each of these will trigger at least one member of $G$ to choose a

until either $A$ or $G$ broadcasts is about $\frac{1}{3}\overline{X}$. This is only an approximation, as the expected period for $G$ is $\tilde{X} < \overline{X}$, suggesting it should be slightly less, but in 34% of cases where $G$ transmits first ($1 - 0.9^4 = 0.34$), there will be a second broadcast from some other $G_i$ that missed the first, bringing the expected value up again.

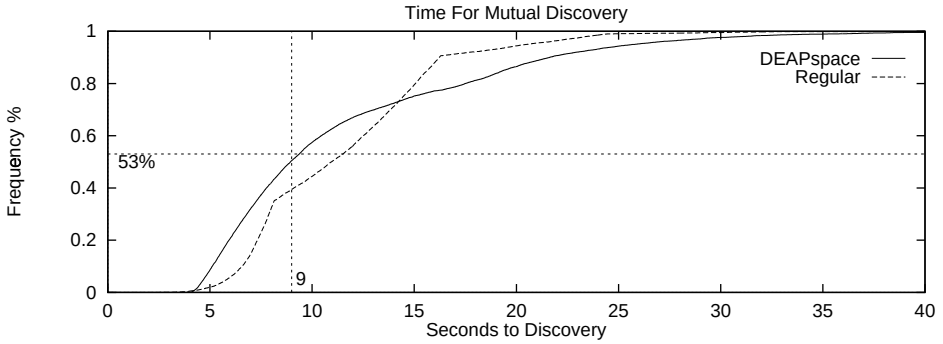
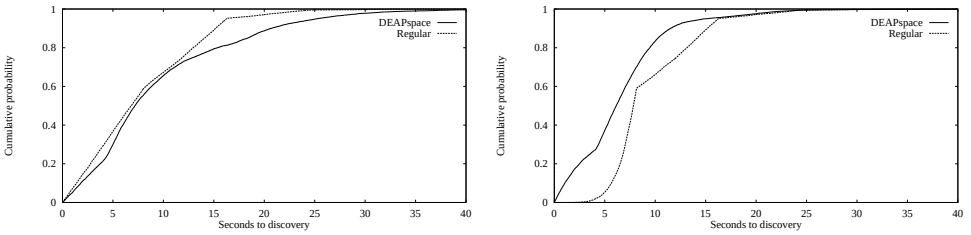


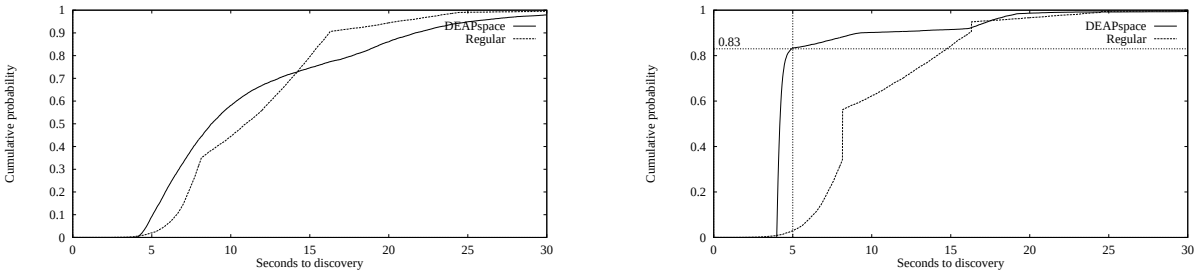

**Figure 5.6:** $P(E)$ (probability all devices have an accurate world view) as a function of time for the DEAPspace and the regular algorithm.




**(a)** $P(E_1)$ **(b)** $P(E_2)$

**Figure 5.7:** $P(E_1)$ (probability G has discovered A) and $P(E_2)$ (probability A has discovered G) as a function of time for the DEAPspace and the regular algorithm.

timeout from $X'$, and send its own list. Some $G_i$ that did not receive the broadcast from $A$ might broadcast earlier, but some member of $G$ will certainly broadcast its list within time $X'_{\max}$ of the broadcast from $A$. This implies that the DEAPspace algorithm allows $G$ to discover $A$ at about the same rate as the regular algorithm does (about one try per round) but allows $A$ to discover all of $G$ faster than the regular algorithm does, because the rounds automatically shrink while the environment is changing. This behaviour is illustrated in Figure 5.7, which compares the probabilities of the events $E_1 = \{G\ has\ discovered\ A\}$ and $E_2 = \{A\ has\ discovered\ G\}$ as a function of time for the regular and the DEAPspace algorithm. The simulation parameters are identical to those given for Figure 5.6. The results are as expected, showing the DEAPspace algorithm to have slightly slower discovery of $A$ by $G$, but faster discovery of $G$ by $A$.




**(a)** without connection detection

**(b)** with connection detection

**Figure 5.8:** Probability $P\{E\} = P\{all\ devices\ have\ an\ accurate\ world\ view\}$ as a function of time for DEAPspace algorithm and regular algorithm with and without connection detection from physical layer.

### 5.3.1 Performance With Connection Detection

The discovery behaviour can be considerably improved if the underlying physical layer can provide a notification indicating that a device has "joined" a network (e.g., when it has synchronised with the spreading sequence of a spread spectrum system and the MAC layer has been enabled). Figure 5.8 shows the behaviour of the algorithms with and without notification. The regular scheme benefits from the shorter time for $G$ to discover $A$, but has exactly the same time for $A$ to discover $G$, because $G$ does not respond to the new arrival. In contrast, with the DEAPspace algorithm, G will recognise the arrival of A, and respond by switching to shorter timeouts.

Taking advantage of connection detection helps DEAPspace because at least one of the group will receive the initial advertisement, and respond in four to five seconds. In less than five seconds, each member of the group $G$ has usually had two opportunities to learn about the new device $A$, and $A$ has had at least one opportunity to learn about all of $G$. Specifically, the value at time 5.00 seconds can be predicted:

- The number of devices expected to receive first message = $5 \times 0.9 = 4.5$
- All of those devices will choose a new timeout from range [4,5].
- One will choose the earliest timeout, and send its advertisement, that leaves 3.5 that do not win the race, but $1 - (0.9^{3.5}) = 31\%$ of the time, one of those devices will miss the first response transmission, and send a second one, but each of the devices that missed the

initial broadcast from $A$ has about a $\frac{4}{13}$ chance of first interrupting with its regular transmission. In fact, the probability of interrupting increases if more than one device missed the first transmission, but we can approximate at about $\frac{4}{13}$ times 41% (the probability that one or more devices missed the first broadcast from $A$.) That gives about a 13% chance that the response transmission was interrupted, but there is a further chance that the interruption was not received by a sender (assuming four received the first broadcast, there is a 34% chance that at least one will miss the interruption – dropping the probability of *effective* interruption to 8.5%.)

- Therefore, after five seconds, we have an 8.5% chance that one of the devices in $G$ has missed the first broadcast, and preempted a repeat of $A$ information in the reply. Of the remaining 91.5% of cases, 31% will have the information about $A$ sent three times, and 69% twice. In the cases without interruption, therefore, we can expect a chance around $(31\% \times 0.99^5) + (69\% \times 0.999^5) = 98\%$ that all of $G$ will have learned about $A$. That gives a total probability around 89.5% that all of $G$ will have learned about $A$ in the first five seconds.

- The other direction – $A$ learning about $G$ – is less complicated, since any member of $G$ can do it. Some member will certainly send in the first five seconds, and there is about a 34% chance that one of the other four will miss that, and a 90% chance that a device to miss it also received the first broadcast, meaning that there is about a 31% chance of $A$ getting two chances to hear about $G$, and a 69% chance that it got only one. That makes a $(69\% \times 0.9) + (31\% \times 0.99) = 93\%$ chance that $A$ will have learned about $G$ in the first five seconds.

- These probabilities are not entirely independent, but the behaviour of $A$ is actually not very dependent on the reasons for $G$ transmitting, so multiplying the probabilities can still give a meaningful result. Specifically, it predicts an 83% probability of all devices having learned about all others in the first five seconds. This corresponds with the first sharp corner in the DEAPspace curve of Figure 5.8 (at 5 seconds, 8,337 of 100,000 trials)

Clearly, DEAPspace is able to take good advantage of knowing when a new network has been joined. The regular algorithm gains something, but is helped only in getting messages about $A$ to $G$ faster. Because it is not adaptive, there is no change to how quickly $A$ learns of $G$.

The improvement from 0s to 8.15s is only from one message being certainly at 0s, instead of randomly positioned somewhere in the interval. The jump at 8.16s is the second transmission from $A$. This takes the probability from $0.9^5 \times 0.9^5 = 35\%$ to $0.9^5 \times 0.99^5 = 56\%$. Another jump happens 8.16s further along, at the third send from $A$.

Using connection detection even makes DEAPspace competitive with regular broadcasts for the case of two devices meeting, as can be seen in Figure 5.9. Because this example uses the same parameters used in Figure 5.7, set to create comparable steady-state load for six devices with 10% packet loss, the DEAP space configuration in Figure 5.9 has a lower network load under normal operation than the regular scheme.

In the case of two devices, if the first two messages after they come into range are not lost, the DEAPspace algorithm will have accomplished mutual discovery by time $X'_{max}$, and regular beaconing will have done so by time $T$. This is the most clear example of how the ability to recognise and respond to change helps the DEAPspace algorithm to achieve fast mutual discovery.

What all these comparisons have shown is that the DEAPspace algorithm, when combined with connection detection, performs as well as or better than comparably configured regular broadcast.

## 5.4 Simultaneous Start

While the DEAPspace algorithm was designed with specific attention given to the scenario of a single transient device discovering an existing group, this does not describe every useful scenario. For example, consider a meeting room in which several users, who do not usually leave their radio modules active, decide to share meeting notes. If all $n$ devices are switched on at about the same time, DEAPspace will initially have all devices sending lists with one element. This will make the optimal DEAPspace discovery time about $n$ times the average timeout chosen from the shorter range, while the optimal regular discovery time will remain about the same as the normal broadcast period. Figure 5.10 shows the time for all devices to have discovered all others, using the same timeout values as in Figure 5.7 over a range of packet loss probabilities.

As can be seen from Figure 5.10, the regular algorithm is always better for low packet loss

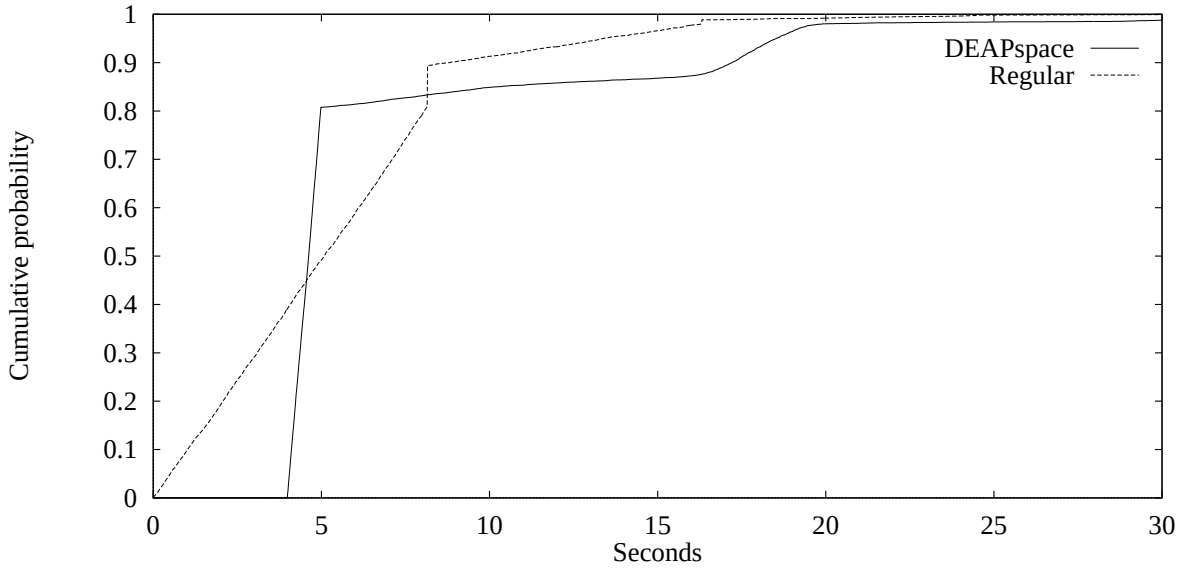
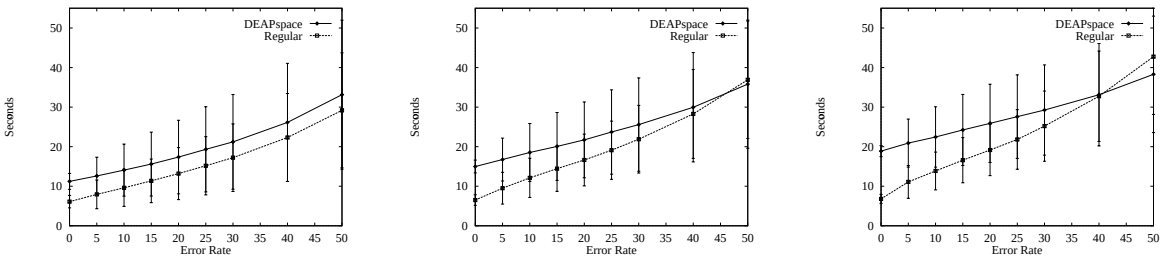
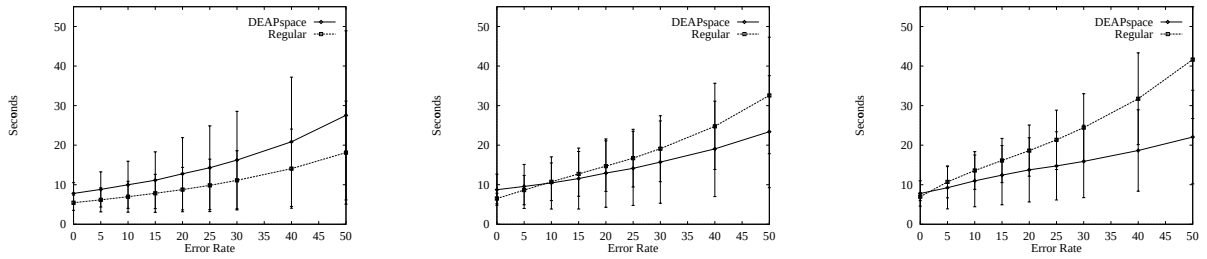



**Figure 5.9:** $A$ joins $G$ with connection detection, where $G$ is a single device. Timer values and loss rate are the same as in Figure 5.7. (Notice the obvious point on the DEAPspace curve ats $X'_{max}$=5 seconds and 81%, the probability of two particular messages being received with 10% loss probability.)




**(a)** 3 devices **(b)** 4 devices **(c)** 5 devices

**Figure 5.10:** Time required for mutual discovery when all devices are started simultaneously for 3, 4, and 5 devices. Bars show one standard deviation.




**(a)** groups of 1 **(b)** groups of 2 **(c)** groups of 3

**Figure 5.11:** Time for two groups to discover each other, for groups of 1, 2, and 3 devices. Bars show one standard deviation.

rates and for small numbers of devices. Some benefit comes from the DEAPspace adaptive nature for larger quantities of devices in lossy environments, but the constant timeout resetting in DEAPspace is a liability for the simultaneous start scenario.

## 5.5 Two Groups Meet

Another potentially interesting situation is having two previously-existing groups of devices merge. This merging scenario lends itself better to the DEAPspace ability to discover lists, rather than individual devices. For small groups (e.g., two groups of one), the shorter periods of the regular scheme, and the fact that it does not reset timeouts based on receipt of other transmissions makes it faster than DEAPspace. However, as the results presented in Figure 5.11 show, the advantages of DEAPspace overmatch those of the regular scheme for even small packet loss with groups as small as two devices. The discovery time for DEAPspace is almost independent of group size (actually improving slightly for larger groups), while the regular scheme suffers as group size increases.

## 5.6 Using DEAPspace for Route Discovery

As has already been mentioned, route discovery can be seen as a special case of service discovery. Each node offers a large number of services (reachability of each other node in the network). Also, most services will be offered by more than one neighbour of each node, differentiated by the anticipated quality of the route through each of those neighbours.

A strength of DEAPspace is to allow nodes to discover services offered by nearby nodes, even when communication to that node is temporarily interrupted. If a colour printer is available nearby, and a colour document must be printed, then it will frequently be the case that no alternative service offering will suffice. In this case, it is useful for the potential client to know about the service, even if communication is temporarily interrupted. In routing, if the first hop is not connected, then a different route will almost always be more suitable.

In short, DEAPspace offers fast discovery of new servers at the cost of slower discovery of expired ones, but routing is a subset of service discovery challenges for which this tradeoff is not good. Discarding bad routes quickly is just as important as discovering new good ones. Because of this, DEAPspace is not the most appropriate choice for typical route discovery.

### 5.6.1 Geographic Routing

A possible exception to DEAPspace usefulness for routing lies in geographic routing schemes [vHH01, LJC+00, BCSW98]. Geographic routing uses the fact that, with wireless ad-hoc networks, geographical proximity often implies topological proximity, and uses knowledge of the physical layout of the network to find likely routes. One very significant challenge in these problems is allowing the nodes to learn the physical layout of the network as it evolves over time. In the three examples cited here, beacons are used to allow nodes to build a picture of the total network.

It is likely that a good solution could be based on the list sharing ideas of DEAPspace to assist geographical routing solutions. The change is that, unlike in other ad-hoc routing approaches, outdated information about a node is better than none, because an approximate location still assists with routing. This application is outside the scope of the problems being addressed in DEAPspace, but holds promise for future work.

## 5.7 Chapter Summary

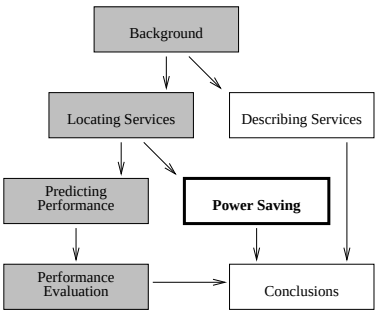
This chapter has shown that the time required from when a device enters a new environment until it has discovered the services available therein is better with the DEAPspace algorithm than with the non-adaptive alternative of regular beaconing. Furthermore, the new algorithm is particularly well suited to work with underlying protocols that use distinct, one-sided join

events, like the initial synchronization with a CDMA signal in a peer-to-peer network. This was shown both by comparing theoretical and simulated values for the different algorithms, and also by demonstrating the behaviour of an actual implementation.

# Chapter 6

# Power Saving

*Improved timeliness would be less impressive if it came at significant cost to network bandwidth or power. Chapter 5 demonstrated the DEAPspace algorithm to be viable in terms of network load. This chapter will evaluate its viability in terms of actual power consumption.*




Solutions based on the Push-model certainly use more network bandwidth than pull, except in very high load environments. Quantification of this statement is a goal of this chapter. As we are targeting very short range environments, one hundred distinct services in a single LAN is a generous upper bound for the near future, and the hierarchical object identifiers in our current implementation allow almost any service to be described in less than a hundred bytes [HHM+00]. Actual data rates are strongly dependent on the protocol being used, but 50 KBps is a reasonably conservative estimate for the real data transmission rate in a short range network so, if we want to advertise all available services about once every ten seconds, these numbers show a worst-case usage of about 2% of the available bandwidth. Even allowing for framing difficulties (packet header sizes, media contention, etc.), this is well within tolerable background levels for most applications. Beaconing this much data will generate a large number of broadcast packets which, in some protocols, would be a problem. However, since DEAPspace would replace the hundred broadcasts every ten seconds with a single large broadcast every ten seconds, broadcast frequency restrictions will not present any problem.

There is no doubt that however little traffic is caused by a push solution, the pull solution will require less, but it is not really the saving in network bandwidth that concerns most people; it is the perceived saving in power. This expected saving comes from the knowledge that, in traditional radio communication, transmission is expensive. That is no longer true for short range networks, where being active is expensive; what a device actually does while active, whether it transmits or just listens for connection requests, is less relevant than the fact that it is active at all. That is to say that if devices must be taken out of their idle mode to listen for service requests, the marginal cost for transmitting instead of just listening is proportionally very small. In practice, a pull model server can use idle time by listening for requests only periodically, as with the Bluetooth inquiry scan, but this fix requires client requests to be repeated many times, and badly increases response time. As will be shown in Section 6.2, DEAPspace is able to put servers in idle mode with very little effect to the discovery performance. In short, compared with a pull-model solution, DEAPspace offers a responsive environment at the sole cost of a small amount of background traffic.

## 6.1 State of the Art

To support these claims, Bluetooth development offers insight into the state-of-the-art, as it is still coming onto the market now. For bandwidth, DM5 packets contain 224 data bytes and take 3125 $\mu$sec to transmit, with 625 $\mu$sec between consecutive packets, giving about 60 KBps. DM1 packets, the lowest data rate, give 17 data bytes in 625 $\mu$sec, with 625 $\mu$sec gaps between transmissions, leading to a real data rate of 14 KBps. These numbers, however, are only the values seen by upper layers; the contribution to network congestion should not include the time spent waiting for other transmitters. If only on-air time is considered, these become 72 KBps and 27 KBps respectively. If forward error correction (FEC) is not used (i.e., DH packets are used instead of DM) these rates become 108 KBps and 43 KBps, but service discovery would normally use FEC. Bluetooth was designed to work on very low power, inexpensive chips, so these rates represent the lower end of the state of the art data rates. Given these values 50 KBps is not an unreasonable rate for our example hardware.

For power consumption, consider again Bluetooth. It was designed for the types of devices and networking ranges that DEAPspace targets, so it should give an indication of reasonable values. One chip on the market is the Silicon Wave SiW1502 IC, designed to enable Bluetooth

products [Sil00]. Preliminary measurements of this chip show current draw of 57 mA when transmitting, 60 mA when receiving, and 20 $\mu$A when in standby (idle) mode. This is a clear example of a case in which a server constantly listening for requests will use more power than one alternatively beaconing, listening for a response, and sleeping. More specifically, it tells us that idle time is the single most relevant issue to power consumption in the transceiver chip.

## 6.2 Using Idle Mode

Because sharing world views has the effect of greatly reducing the total broadcast frequency, compared with having all the devices advertise their own services, by replacing $n$ broadcasts of one SE each with one broadcast of $n$ SEs, the proposed scheme results in much longer pauses between broadcasts. During these pauses, some power-sensitive devices may wish to use whatever idle mode is available from their hardware platform. It is important that whatever solution is implemented should not interfere significantly with the normal behaviour of other devices.

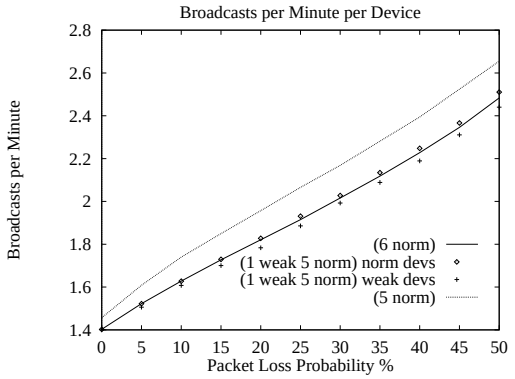
The specific technique proposed here, as detailed in Figure 6.1, involves periodic idles with duration equal to the minimum broadcast delay ($X'_{min}$: in this case 4 seconds). These idle times will be initiated every time a broadcast is received in which the modified device's own services are all not near expiry, or when a broadcast is transmitted. Using longer idle times could cause devices to miss the last-minute renewals sent to prevent imminent expiries, and using shorter idle times would only allow broadcasts to be received from devices that missed the previous transmission. For this discussion, devices that implement the modification presented in Figure 6.1 will be referred to as "weak," in reference to their power availability.

Implementing this simple modification on one device in a group of six allows that device to be hibernating more than a quarter of the time when packet loss is less than one in two, while the total network load is not significantly different[1] from the load caused by six normal devices, as can be seen in Figure 6.2. When packet loss gets worse, causing devices to expire more frequently, individual devices will more often be choosing their transmission times from $X'$, meaning that hibernation becomes less frequent, so whatever packets do arrive correctly are more likely to arrive while the receiver is active. With our test parameters as before, all

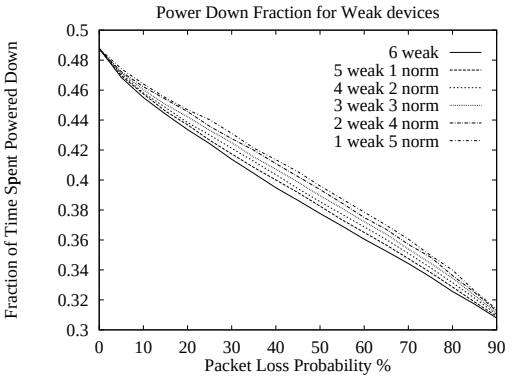
[1] With the same sample configuration parameters used in Chapter 5, $X = [12, 15]$, $X' = [4, 5]$.

```
1  advertise(LOCAL) {
2     time tout ←getTimeout(X)
3     loop(forever) {
4        REMOTE ←read(tout)
5        if(timed out) {
6           foreach s ∈ LOCAL
7              if(s ∈ MINE)
8                 s.expiry ←NormalExpiry
9           broadcast(LOCAL)
10          tout ←getTimeout(X)
11       } else {
12          Interval I ←update(LOCAL,REMOTE)
13          tout ←getTimeout(I)
14       }
15       if(tout > X'max) {
16          tout −= X'min
17          hibernate(X'min)
18       }
19    }
20 }
```

**Figure 6.1:** Allowing weak devices to hibernate during service discovery




**(a)** Transmission Frequency




**(b)** Hibernation Time

**Figure 6.2:** Behaviour of a group of six devices, in which one through five of them use the weak device modification described in Figure 6.1, having $X = [12, 15]$ and $X' = [4, 5]$

devices are expiring in the lists kept by the normal devices slightly less often than they would if all six were normal, although the list kept by the weak device tends to be slightly worse. In general, and especially at low packet loss rates, this modification does not affect the total system performance.

A possible drawback to this technique might have been degraded timeliness of discovery. Fortunately, with reasonably sized groups like the six-member groups used above, there is very little effect on timeliness. For a normal device entering a group that contains a weak device, some extra delay for mutual discovery results from the weak device sometimes being idle when the new device arrives, but the normal devices discover each other as fast as ever, because the first message either way causes some device to enter a panic state, the same as usual. If the new device is the weak one, then the situation is slightly worse, but the total effect is still very small (Figure 6.3).

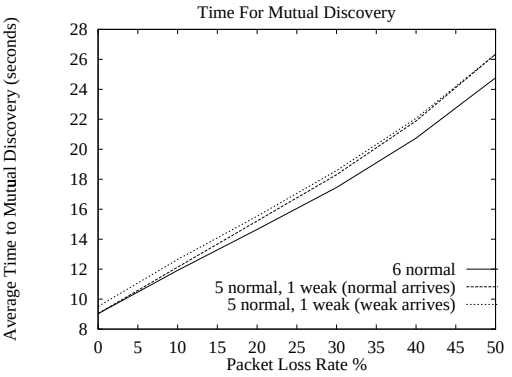



**Figure 6.3:** Time to mutual discovery when a new device enters an existing group of five devices

This source of this difference is the cases in which the existing group is the first to transmit, and the weak device is idle at the time. If a device initiates an idle period every broadcast cycle, then the fraction of time spent idle by that device will be about $idle\ time \div \overline{X} = 4 \div 13.5$.

The group will transmit first about half the time, so one would expect this scenario to actually occur about $\frac{1}{2}\frac{4}{13.5} = 15\%$ of the time, and cause discovery to be delayed until the new device transmits, about another 11.5 seconds (assuming it was about half way through a four second delay at the beginning of a 13.5 second cycle.) The various panic states of the existing group make the difference slightly less (for low packet loss rates) than the $0.15 \times 11.5 = 1.6$ seconds that this would suggest but, in general, mutual discovery of the new device is contained by this limit. Furthermore, if a signal is sent to the detection algorithm when the underlying network establishes a connection, then the incoming device will always be first to transmit [Nid00], meaning that this problem will never occur at all.

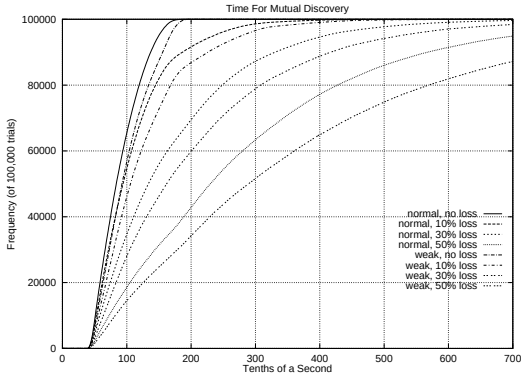



**Figure 6.4:** Comparing the time to mutual discovery for two weak devices versus two normal devices.

The greatest danger with this technique is the case of two devices meeting, where both are weak. As with the larger group, if connection establishment is being signalled to upper layers, there will be no drawback; but consider the case where it is not being signalled. Because both devices were previously alone, they will both be sending SAMs about once every 13.5 seconds, and hibernating for 4 seconds following each broadcast. In the worst case, one device (call it $a$) has just started hibernating when communication becomes possible, and the other ($b$) sends during this period. One of the devices, probably $a$, will be the next to send. Because $b$ was in hibernation, the earliest $b$ could transmit is $X_{min} - hibernation\ time$ later so, recalling that $hibernation\ time$ is $X'_{min}$, if $X$ and $X'$ have been chosen such that $X_{min} - X'_{min} < X'_{min}$, (i.e., $X_{min} < 2X'_{min}$) then this broadcast is certain to be during an awake period for $b$. If this message is lost, then $a$ will be hibernating during the next broadcast from $b$. In other words, slightly less than a third of the time, the chance of losing the first two packets is the same as the chance of losing the first one. For example, when the packet loss is 20%, the chance of one of the first two messages getting through is $1 - (0.2^2) = 96\%$ for two normal devices, but $1 - (\frac{1}{3}0.2 + \frac{2}{3}0.2^2) = 91\%$ for two weak devices. As can be seen in Figure 6.4, this can mean a difference of two or three seconds when the packet loss is high, but also means the lone devices were achieving this rate while hibernating a third

of the time. In many applications, this tradeoff is worth while.

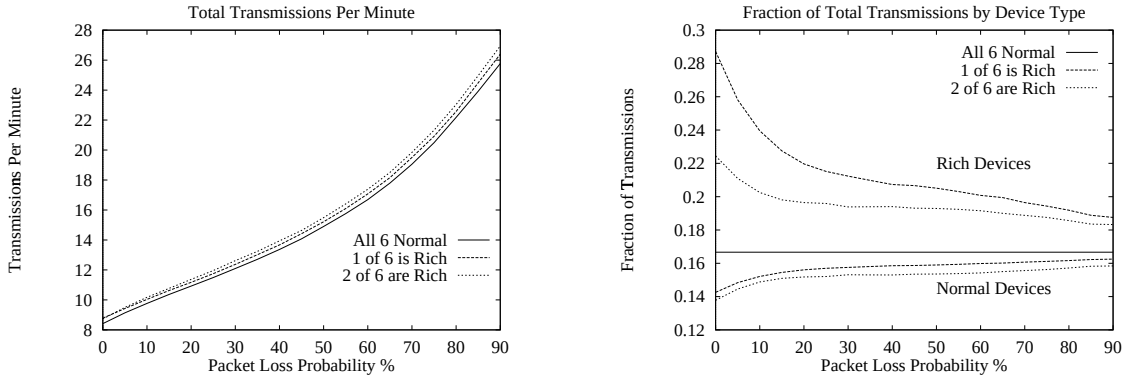
## 6.3 Reducing Broadcast Frequency

While the most common implementations of the ideal DEAPspace target environment do share the property of needing idle time as the overwhelmingly most important component of their service discovery protocol, other possibilities still exist. Because the exact platform was not, and is not, a certainty, other possible modifications should also be considered. In particular, what properties would emerge if the adjustable parameters in the discovery algorithm were set asymmetrically?

Obviously, configuring the devices differently means that the observable behaviour can be expected to fall into groups, with similarly configured devices grouping together. As broadcast frequency is the most obvious observable property, perhaps it can be predictably manipulated to shift the broadcast load in a given direction. It is possible that an application would require fewer devices, and longer range. Such a system might keep the abstract model for which the DEAPspace algorithm is designed, but use transceivers with significantly more current draw when transmitting than when receiving. (For purposes of this discussion, devices with good power availability will ve referred to as “rich,” in reference to their power-rich status.) If that is the case, then asymmetric parameter assignment can be used to reduce the power requirements for some small devices.

The simplest way to achieve this shift is by arranging to have rich devices usually choose broadcast times smaller than the times usually chosen by other devices. In this example, the shift is achieved by leaving the distribution unchanged (uniform), but lowering the range from which timeouts are selected. An alternative would be to use the same range for all devices, but skew the probability density function used by rich devices towards the lower end of that range.

Figure 6.5 shows that while this scheme will slightly increase the overall network load in the vicinity of one of these rich devices, it will reduce number of broadcasts sent by normal devices in that same area. In this example, the broadcast times for normal devices are taken from the range $X = [12, 15]$ seconds, and for rich devices from $\mathcal{X} = [10, 13]$ seconds. The expiry times are 60 seconds, and a device will choose from $X' = [4, 5]$ seconds if it sees a SAM showing its own services within 20 seconds of expiry. By allowing $X$ and $\mathcal{X}$ to overlap, the normal devices




**(a)** Total Transmissions **(b)** Fraction by Device Class

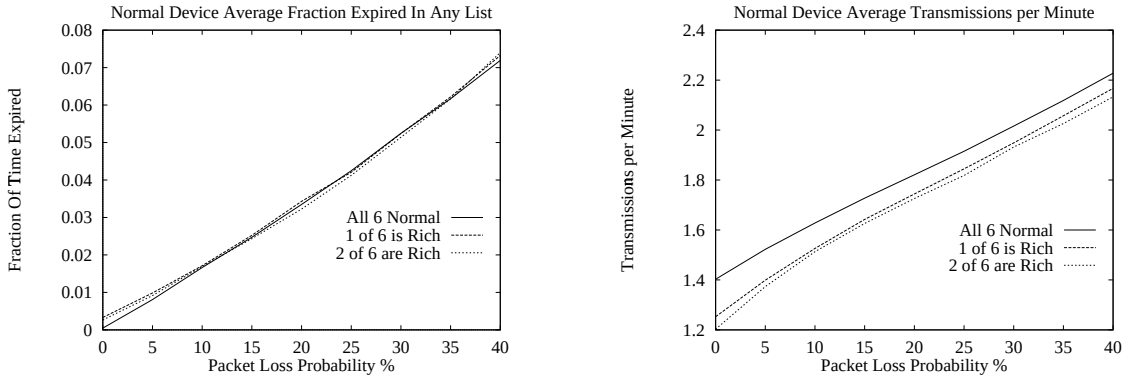
**Figure 6.5:** In the presence of zero, one, or two rich devices, the total number of broadcasts per minute for the local network, and the average per device for rich and normal devices

will still make some normal (non-panic) broadcasts. In this way, the number of short rounds caused by panic will be kept down, but the rich devices will still assume more of the broadcast load.

As shown in Figure 6.6, this scheme slightly increases the total number of broadcasts per minute, but it reduces the number of broadcasts sent from the normal devices. Moreover, it does not affect the frequency with which devices are expired in the lists of others.

Unsurprisingly, the smaller the difference between the rich and normal devices, the smaller the effect that the rich device will have on the system. Figure 6.7 shows what happens when the overlap is increased by changing the normal range for rich devices from [10,13] to [11,14], the total number of broadcasts is reduced, and the number of broadcasts made by the normal devices is increased.

What we learn from this is that a disproportionate fraction of the broadcast responsibility can be shifted to devices of our choosing by assigning timing parameters asymmetrically. While not as generally applicable as the use of idle times described earlier, the ability for well-powered devices to independently assume responsibility for a disproportionate fraction of the broadcast load is an interesting property, and is helpful in situations where broadcast transmissions are significantly more expensive than receptions.




**(a)** Expired Frequency

**(b)** Transmission Frequency

**Figure 6.6:** In the presence of zero, one, or two rich devices, the fraction of the time spent expired by an average normal device, and the average number of broadcasts per minute for a normal device

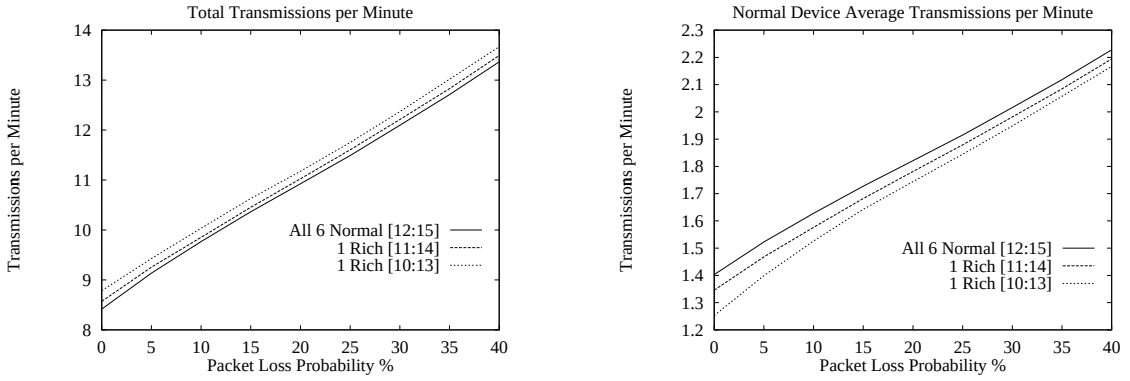
## 6.4 Further Modifications

Any number of special modifications can be envisioned for particular environments, and the more focused the optimization, the more meaningless it is to demark the advantages in a general way. For example, consider the case where an environment is expected to contain many interchangeable services that are provided by very mobile devices.

In a highly dynamic environment, containing many similar services, unexpired SEs that describe services offered by absent devices may be a problem if they are chosen in preference to a valid alternative that is still present. The fix for this is for devices in competition with each other (i.e., those that offer similar services), to advertise more aggressively. By doing this, each competing device makes it more likely that its SEs will have later expiry times, and will therefore be chosen in preference to their competitors.

Suppose a third timeout range is used by devices that see services similar to their own offerings advertised with a significantly later expiry time than their own. If they picked from a middle range (e.g., [4,5] panic, [7,8] eager, [10,14] normal) then missing devices would be preempted faster.

Using this technique means that if a device leaves, but a reasonable substitute remains, that alternate device will quickly renew itself to have a later expiry time. Normal clients will therefore choose the SEs of the remaining device as the more desirable offering, resulting in




**(a)** Total Network

**(b)** Average Normal Device

**Figure 6.7:** Number of broadcasts per minute for the whole set of six devices, and the average number of broadcasts per minute for a normal device for various ranges of timeouts for rich devices

fewer failed connection attempts to absent devices when a valid alternative was available. This fixes the problem scenario presented in Figure 3.2.

Of course, this will also increase the local network load whenever two similar devices are in the same zone while being of potentially no help if both devices are static, as with a printer room containing several printers. However, if it is combined with idle times after every normal or eager broadcast, similar to the technique from Section 6.2, the shorter typical rounds may even have a helpful effect on actual power consumption.

## 6.5 Scaling

So far, we have seen how the new algorithm behaves for small groups of devices without any concrete examples of scaling up to larger numbers of devices. This section will briefly look at what can remain the same, with reference to the earlier examples, and what should be changed when large groups are expected.

$X$ and $X'$ reflect the timeliness of discovery, and are therefore largely independent of the number of devices expected, so let's leave them at $X = [12, 15]$ and $X' = [4, 5]$, as used in the earlier examples. That being the case, the expiry time of 115 seconds becomes unrealistic since, even if all rounds were as a result of some device worrying, rounds have a minimum duration of 4 seconds, and devices must have at least one opportunity to broadcast before they expire. This returns us to the earlier dichotomy of choosing an expiry time based on the number

of transmission rounds, when the choice of time will affect the duration of the rounds. Since Equation 4.7 predicts that even if devices worry only after 150 missed rounds, and assuming no packet losses, a given device will still worry about once every 267 rounds, meaning we can expect about one in three rounds to be the result of some device worrying. That being the case, 150 rounds will be about 1500 seconds (25 minutes). Clearly, this means that absent devices will not be noticed for a long time, although new devices will still be promptly discovered. As explained in Section 6.4, the impact of this tradeoff on application performance is reasonable when responsiveness is required.

In such large groups, it may be worthwhile to send a request to the top two or three choices, rather than polling the possible servers one at a time. This approach may begin to remind you of client beaconing, but has several important differences:

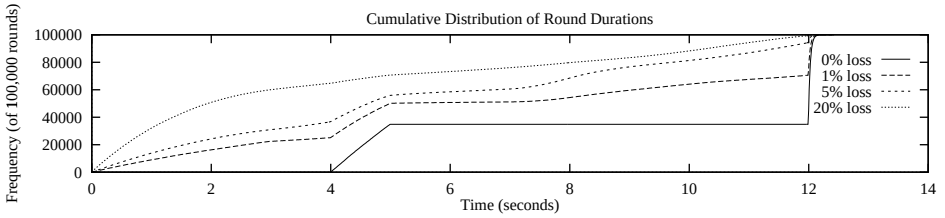
1. Connections are point to point, and do not use potentially valuable or scarce broadcast packets.

2. If SEs contain timing information, the request can be certain to correspond to an awake period for a server that is usually idle, making it possible for servers to save power without forcing clients to repeat connection attempts for an extended period.

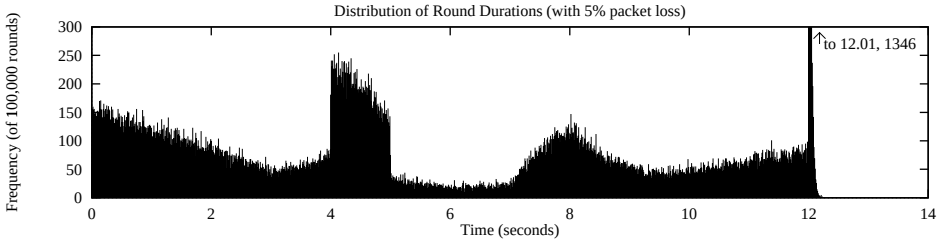
3. If no suitable server is available, the client knows immediately that this is the case. If it wishes to wait for one to arrive, it will discover its arrival promptly without the need for constantly repeating a request.

Back to the main point: configuration parameters. We are using $X = [12, 15]$, $X' = [4, 5]$, $b = 1500$ (seconds). We expect devices to be worrying fairly regularly, and $b$ is fairly long anyway, so let's give an extra 25 seconds to expiry, making the service expiry time 1525 seconds. This allows a device to worry for at least two rounds before its services actually expire, making successful renewal more likely.

With a large number of devices, almost every transmission will not be received by some device. For example, with 100 devices, and 1% loss, most transmissions resulting from a normal round will be followed almost immediately by a transmission from a device that also chose a normal timeout, but didn't receive the first try. Similarly, many 4-5 second rounds will be followed by a 7-11 second round that started off as a 12-15 second round, and lost the interruption 4-5 seconds in. (i.e., from the point of view of the transmitting device, the round lasted




**(a)** Cumulative frequency for various loss rates




**(b)** Observed frequency for 10 ms intervals of the 5% loss case

**Figure 6.8:** Time between broadcasts observed with 100 simulated devices over 100,000 rounds.

12-15 seconds, but the receiving device also received a packet after 4-5 seconds, resulting in its observation of a second round having length between 7 (12-5) and 11 (15-4) seconds.) These behaviours can readily be seen in Figure 6.8, particularly in part 6.8(b). Note the ranges:

**0-3**: Second or third transmissions from devices that picked from the same range as the winning device, but did not receive its broadcast, and therefore did not cancel their scheduled advertisement.

**4-5**: Normal transmissions chosen from $X'$. Note the slight ramp up to 4; consider this scenario:

1. X broadcasts
2. Y receives the broadcast and, seeing that it is nearing expiry, schedules its next broadcast from the range [4,5].
3. Z is one of the devices that did not receive the X broadcast, and had scheduled a broadcast from the same range as X, so goes ahead and sends its own.
4. Y does not receive the broadcast from Z, and goes ahead and sends its broadcast 4 to 5 seconds from receiving the one from X, which turns out to be 3 to 4 seconds from the broadcast from Z, resulting in a 3 to 4 second round.

**7-11**: Transmissions scheduled from $X$, interrupted by a broadcast timed from $X'$, but the interrupt was lost. The upper end of this range is obscured by transmissions scheduled from $X$, interrupted in the first second or two, but the interruption was not received.

**12**: With so many devices competing, if no interruption comes for so long that a normal transmission chosen from $X$ can be sent properly, the winner must have chosen from very near the lower end of the range, hence the dense concentration in the first few milliseconds following the 12 second mark.

The less likely combinations of events (e.g., the 3 to 4 second range described above) become more common as the loss rate goes up, leading to the smooth distribution of times seen in the 20% loss curve of Figure 6.8.

A point that has been made earlier, but highlighted here, is the necessity that devices know *approximately* how many other devices they expect to have as neighbours. It is presumably reasons like this that led to the design of HIPERLAN that requires new devices to receive configuration parameters, including beacon frequency, when they join a new group. It is possible that including parameter value suggestions (especially for adjusting expiry time) in each SAM would assist devices in being more useful, but the exact form of such a modification would depend a great deal on the actual embodiment for which it was destined.

## 6.6 Chapter Summary

Prior to this chapter, the new service discovery algorithm was studied over a generic network interface. With the exception of Section 5.3.1, in which feedback about joining a new network group was used to improve discovery time, analysis has heretofore been limited to varying packet loss rates. These rates were assumed to be part of the environment, and beyond the control (or knowledge) of the discovery algorithm. In this chapter, we looked at some improvements that are enabled by providing the service discovery implementation with information about the underlying network behaviour.

Initially, and most effectively, a power saving improvement was presented for using network device idle modes to give well over 40% power saving, with almost no cost to performance. Some less hardware-dependent solutions used knowledge about the general power usage of the underlying hardware as a cue to asymmetric assignment of configuration parameters, offering

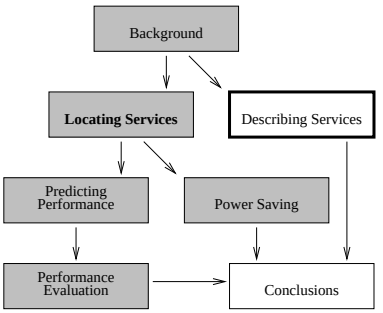
power saving to those devices with the greatest need. Finally, knowledge of increased group size was used to consider general parameter values.

This is the final chapter addressing discovery performance. Chapter 7, the final one remaining before the conclusions are drawn, will give an overview of the actual service description language used to form the SE encodings, and to accept queries.

# Chapter 7

# Service Description

*Discovery is not meaningful except when coupled with a description language for the services being discovered. Thus, for completeness, this chapter will explain the system by which DEAPspace creates and interprets service descriptions. The majority of this chapter expands on the service description section of [HHM+01].*




---

Because the DEAPspace design involves very frequent retransmission of service descriptions, compactness is very desirable, and to allow for flexibility and growth, generality is also important. To achieve compactness, we could not use a completely general description language like XML [BPSMM00], but used instead a combination of standardized definitions and user-defined types inspired by MIME (Multipurpose Internet Mail Extensions) [FB96].

The description format is not the main point of this thesis, so it will not be analysed in great detail, but is presented here with some explanation and brief examples. First, the theory behind the data structures used internally is reviewed, then the actual encoding is given.

## 7.1 Data Structure

Two important lessons from MIME contributed to the development of the data structure used in DEAPspace. The first lies in the fact that data format is the best defined description of the function of an application. The second lesson, also to be seen in SLP and other internet

| | | |
|---|---|---|
| `ServiceDescription` | ::= | Name |
| | | ServiceAttributes |
| | | InputFormat |
| | | OutputFormat |
| | | NameValuePairs |
| Name | ::= | <string> |
| ServiceAttributes | ::= | DeviceAddress |
| | | SecurityAttributes |
| | | Owner |
| | | Fee |
| InputFormat | ::= | FormatType |
| OutputFormat | ::= | FormatType |
| NameValuePairs | ::= | (<string> <string>)* |

**Figure 7.1:** Contents of a service description

protocols, is the compromise between well-defined data types and user extensions.

For a service description to be useful, it must also include more than just what the service is capable of doing. For example, the address of the device offering the service is necessary. It is also useful to allow for owner identification, and other miscellaneous details that do not directly relate to input and output formats. Keeping this in mind, the final design is as shown in Figure 7.1.

In these components, `Name` is simply a user-understandable name that can be used to refer to the service. This name is not necessarily unique to a particular device, and should describe it concisely.

`ServiceAttributes` uniquely defines a service, allowing any device to discover whether it is permitted to access the described service, and how to do so. `DeviceAddress` is simply the information necessary to establish a connection with the service being described. `SecurityAttributes` includes information about which other devices are permitted to use the service, and how to authenticate them. The `Owner` field identifies the current owner of the device, and would most commonly be used to prevent such situations as connecting the mobile telephone conversation of one user to a headset being used by a different user. `Fee` identifies

the cost, if any, associated with using the service.

`NameValuePairs` simply allow user-configurable extensions to the description language. They are not currently used, but allow for future additions, such as GPS (Global Positioning System) location.

`InputFormat` and `OutputFormat` are both of type `FormatType`. This is a flexible, qualitative description of the role of a service, and allows other devices to determine automatically whether it is *possible* for a given service to fill their requirements. The appropriateness of a given choice may depend on other factors, but if the input and/or output formats match the specified requirements, then the service can be used. A null value for either of these fields indicates a data endpoint: null `InputFormat` indicates a data source (e.g., a microphone or video camera), and null `OutputFormat` indicates a sink (e.g., a printer or bulk storage unit). The actual construction of these data structures is the subject of the next section.

### 7.1.1 Format Types

By describing all services as a combination of input formats and output formats, compatibility with new devices can easily be discovered. For example, the GSM telephone that receives an SMS message might look for an external display device as an alternate output. Such a service could be offered to save a user from retrieving the handset from an otherwise inconvenient location, perhaps using a pop-up window on that user's workstation display. A later developer, independent of the original handset vendor, might develop a text-to-speach device for the car that is backwardly compatible with the original handset, but also accept special language markups when available. In this scenario, a new handset should know the difference between the two devices, but the old handset should still be able to make use of the basic interface. For this reason, we have introduced a hierarchy to the service descriptions. For this example, we might use the following taxonomy:

- Workstation Pop-Up Input Format: `text/ascii`
- Text-to-Speech Input Format: `text/ascii/pronunciation-markups`

Because the text-to-speech device extends the basic `text/ascii` format, it is indicating the ability to accept any data that would have been appropriate for the workstation pop-up, while also accepting the extended format that includes markups.

```
    FormatType  ::=  FormatComponent*
FormatComponent ::=  OIDcomponent
                     Parameter*
                     FormatComponent*
   OIDcomponent ::=  <byte 1-254>
                  |  <255> <string>
```

**Figure 7.2:** Contents of `FormatType`

In addition to format classes, like those listed above, parameters can also be used at any level in the hierarchy. A display that accepts image data, for example, may indicate a maximum resolution and number of colours. These parameters would not affect the set of formats that can be read by that display, and would apply equally to all. Other examples of parameters would include version and revision numbers for standard formats like PostScript.

Each particular format type is identified by a unique object identifier (OID), composed of unique values for each level in the defining hierarchy. For well-known extensions, this is a one-byte value; for non-standard extensions, it is a string. The total overview of format types is as shown in Figure 7.2.

Each component of the format type has optional configuration parameters, and zero or more derived components. By using this tree structure, a service that can accept multiple extensions of a format type need not duplicate the common parameters. For example, a display that can show both JPEG and GIF encoded images includes the display parameters only once, then the parameters for the JPEG and GIF components separately.

`Parameter` is any user-defined value. Because the format for these values may not be known to all devices receiving the description, it must be possible to skip over the encoded parameters if the component type is unknown. The actual solutions to these problems are the subject of Section 7.2.

## 7.2 Encoding

Each `FormatComponent` is encoded with its own parameters, and a zero-terminated list of its derived components. Because each component defines its own parameter encoding, the parameters are prefixed with a length, to allow them to be skipped over in the case of a receiver

that does not know the format being described. In most cases, the parameters are expected to be small, so the length is expressed in one unsigned byte, with zero length in the case that no parameters are associated with a particular component. The reserved length value 255 indicates the parameters are continued after 255 bytes of data, which data will be followed by another one-byte length value.

For example, assume that the OID for JPEG images is "10.02," and the OID for GIF images is "10.gif." The image encoding format, in this small example, includes only the resolution of the display device, expressed as two 4-byte unsigned integers (encoded with least significant byte first) representing width and height respectively, and neither the JPEG nor GIF formats have any associated parameters. The format type for a 1024x768 resolution display, capable of displaying JPEG and GIF images would therefore be encoded as the following 25-byte string:
04 10 02 00 00 07 255 03 67 69 66 00 00 09 00 00 04 00 00 00 03 00 00 01 00

Expanding this string with more detail:

04 bytes before end of first local parameters section

- 10 OID for first top-level component "image"
- 02 OID of first second-level component "JPEG"
- 00 JPEG has no derived components
- 00 JPEG has no parameters

07 bytes before end of next local parameters section

- 255 OID of second second-level component is an extended type
- 03 length of string
- 67 69 66 ISO 8859-1 (ASCII) encoding for "gif"
- 00 GIF has no derived components
- 00 GIF has no parameters

09 bytes before end of next local parameters section

- 00 image has no further derived components
- 00 04 00 00 width: 1024
- 00 03 00 00 height: 768

01 bytes before end of next local parameters section

- 00 no further top-level components exist

Notice that the parameters of a format type are encoded after its derived types have been completely encoded. This facilitates the hierarchical interpreter code. The various lengths included in the encoding allow it to be broken into blocks when it is first read. The only part of a format type description that can cause confusion to an interpreter is the local parameters for an unknown format type. To deal with this, the lengths always point to the end of the next section that might cause confusion. If the decoder ever finds an OID with which it is unfamiliar, it can skip to the next block, and continue. This encoding offers a flexible, and compact representation for format types.

## 7.3 Code Structure

The Java code that interprets these format types is, like the type definitions themselves, hierarchical. It is included here for completeness, and for general interest. Readers not familiar with Java may wish to skip this section.

All format types are decendents of `FormatType`. The general flow can be seen by looking at the member function `FormatType.decode()` shown in Figure 7.3.

The piece of code shown in Figure 7.3 uses several non-standard types:

`Encoder`: This is just an object that holds a byte array, and remembers how much has been read. It offers member functions like `readString()`, `readByteArray()`, and `readInt()` that understand the data formats used to encode data in this system. It also has the inverse (write) operations that create an encoded array given the component data types.

`UnknownFormat`: This is a class derived directly from FormatType. When the local device does not know how to interpret the parameter data of the current format (and, therefore, will also not know how to interpret the parameter formats of its derived types) it creates a class of type `UnknownFormat`. This is a place holder class that does nothing except remember the encoded description of the current format type, advancing the state of the `Encoder` object passed in so the next byte read will be the first following the unknown part. This allows the full service description to be correctly recreated for later transmission, even though it was not completely understood by the device performing the encoding.

```
protected void decode(Encoder x) {
    Vector sctmp = new Vector(5,5);
    while(true) {
        DSoidComp c = DSoidComp.create(x);
        if(c==null) break;
        FormatType ft = this.createSubclass(c, x);
        if(ft == null) ft = new UnknownFormat(x);
        ft.oidComp = c;
        sctmp.addElement(ft);
    }
    int sz = sctmp.size();
    if(sz == 0) {
        this.subClass = null;
    } else {
        this.subClass = new FormatType[sz];
        for(int i=0; i<sz; i++) {
            this.subClass[i] = (FormatType)(sctmp.elementAt(i));
        }
    }
}
```

**Figure 7.3:** FormatType.decode() member function

`DSoidComp`: This is simply the `OIDcomponent` described in the encoding scheme overview. A factory member function (DSoidComp.create(x)) is used in place of a normal constructor function, because extended OIDs (strings) are returned as a `DSextendedOIDcomp`, and standardized OIDs (one-byte values) are returned as the base type `DSoidComp`.

The only other non-standard element of the presented code is the call to `createSubclass()`. This function is overloaded by all classes derived from `FormatType`, and uses the next available data in the `Encoder` parameter to create the subclass identified by the supplied OID component. If that OID component is unknown to the version of the current format type stored on the device performing the interpretation, then null is returned. This event triggers the decoding function to create a placeholder `UnknownFormat` type with whatever encoded bytes describe the unrecognized format.

The extensions of `FormatType` all have constructors that accept an `Encoder` object as their parameter. These functions call the decode function presented here first, then decode their local parameters. This code structure is the reason the parameters of a base class are encoded after those of its derived classes.

The encoding functions are even more simple, as shown in Figure 7.4.

For this function, each derived type that uses parameters simply overrides the member function `FormatType.encodeLocal(Encoder)` to be the inverse of its own decoding of local parameters. Notice that the FormatType base class keeps its own list of sub-classes in an array, and that each object (sub-classes and OIDs) know how to add their own encoded data to an `Encoder` object. Also note that the call to `writeEndMarker()` actually signals the encoder of the end of a block. These blocks are marked with lengths at the beginning, that will be added when the contents of the encoder are actually converted to a byte array for transmission.

## 7.4 Query Matching

In the current implementation, a query in Java is made by creating a template `ServiceDescription` that contains values in whichever fields are relevant to the search, and null values for all other fields. Most usually, the only non-null values will be the input and/or output formats, but attributes like `Owner` might also be important to some queries.

The values for input and output formats (when not null) contain the highest level class that

```
protected void encodeLocal(Encoder x) {}

public void encode(Encoder x) {
    if(subClass != null) {
        for(int i=0; i<subClass.length; i++) {
            subClass[i].oidComp.encode(x);
            subClass[i].encode(x);
            subClass[i].encodeLocal(x);
            x.writeEndMarker();
        }
    }
    DSstdOIDcomp nc = new DSstdOIDcomp(DSoidComp.ENDMARKER_OID);
    nc.encode(x);
}
```

**Figure 7.4:** FormatType.encode() member function

fits the application requirements. In practice, the query matching function simply uses this template to generate the OID for the desired class. If a `FormatType` tree contains at least one match for all specified components in the requested OID, including descendants of that class (having more components) then that format is considered to match the request.

An array of all service elements matching the requested template is returned to the requesting application. If particular parameter values in the `FormatType` entries are important to the application, then it must do its own filtering of the list that is returned. This avoids a requirement for including class-specific comparison functions in all extensions of `FormatType`, which functions would be necessary for meaningful comparison of parameters. The data format would permit exact parameter matches to be made without such member functions, as was done in Jini, but this was not seen as sufficiently useful to be implemented in the current version.

## 7.5 Chapter Summary

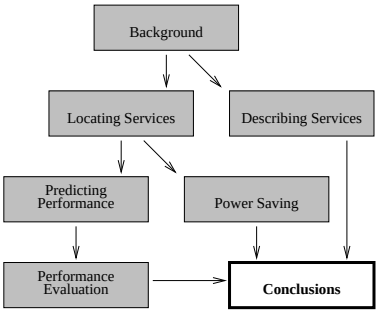
This chapter has summarized the design of the service description and encoding language used in the existing embodiments of the DEAPspace service discovery algorithm. This encoding scheme can generally be characterized by the following features:

- All data types are byte-aligned; encoded data (with the exception of booleans) will take up only as much space as necessary.
- Service descriptions are hierarchically structured data types that share information at the highest possible level.
- `FormatType` data is encoded in a post-order fashion: first those parts that differ, then those that are shared; for each level specific end markers are used.
- The decoding algorithm skips unknown data format types, and resynchronizes with the next known upper-level format type, using the end marker corresponding to the item to be skipped.

# Chapter 8

# Conclusions

*This thesis has explained the need for a service discovery solution that addresses the particular qualities of transient, ad-hoc wireless networks. It also offered a solution in the form of a new algorithm. This chapter will summarize and review the work performed, and extrapolate to some new directions that offer promise for future development of the ideas presented.*




---

This thesis opened with the problem of service discovery in ad-hoc networks. This is an important area of ongoing research, but existing solutions can be frustrated both by frequent changes in group membership (transience) and by unreliable communications (as experienced with wireless). It was argued that to deal with the transient nature of the networks, a decentralized approach would be necessary. It was also shown that addressing the limited broadcast bandwidth and poor link reliability would require collaboration between servers. An appropriate new algorithm, designed with these observations in mind, was then presented.

The DEAPspace class of algorithms provides a distributed service discovery mechanism that is very tolerant of packet loss, and uses a minimal amount of broadcast traffic. Newly arriving devices discover all the services available locally more promptly than with comparable distributed algorithms, such as server beaconing. Furthermore, this algorithm allows for generous power conservation potential through the use of device idle modes.

## 8.1 Contributions of This Thesis

This thesis has identified areas of weakness in current discovery techniques, as they apply to transient ad-hoc networks. It has proposed a new style of service discovery that counterbalances these problems, and quantified the areas of strength and weakness in this new system.

The following is a short review of how this was accomplished:

**Chapter 2** (Background):

- Reviewed existing discovery techniques, identifying trends in design strategies, and explained why and how these strategies are useful
- Reviewed contributions of route discovery to general information dissemination in ad-hoc networks, and drew parallels between the goals of route discovery and those of service discovery

**Chapter 3** (Locating Services):

- Introduced a new class of algorithm that uses server collaboration to efficiently share service information amongst a group of devices
- Made direct comparisons between this new solution and existing routing solutions

**Chapter 4** (Predicting Performance):

- Defined a base model for studying the expected usual behaviour of a group of devices embodying the DEAPspace algorithm
- Examined the recovery mode (worrying) through which devices can tolerate the random occurrence of an unusual number of consecutive lost races
- Explained the expected effects of packet loss on the performance of the DEAPspace algorithm

**Chapter 5** (Performance Evaluation):

- Introduced and validated a discrete event simulator for studying the effects of unusual network conditions on discovery performance
- Introduced and validated a real-time network emulator for studying discovery performance with actual client application code
- Explored the behaviour of the DEAPspace algorithm under artificially constructed pathological cases of simultaneous start, merging of two groups, and meeting of two lone devices
- Discussed the use of DEAPspace as a routing method, noting that it may be useful for disseminating location information for use with geographical routing strategies

**Chapter 6** (Power Saving):

- Explained a technique for using the idle mode of some hardware to reduce the energy consumption of devices embodying the new algorithm by more than 40%
- Presented methods for using asymmetric parameter assignment in algorithm embodiments, and explained the resulting behaviour
- Verified the scalability of the algorithm to groups of a hundred devices, and discussed the configuration considerations for devices that expect to join such large groups

**Chapter 7** (Service Description):

- Introduced a technique for usefully describing services in such a way as to allow backward-compatible queries to make use of future extensions
- Presented a compact encoding scheme for these service descriptions

Of these, the primary contribution is the development of a new style of service discovery algorithm. This new solution has various useful qualities:

- Makes reasonable load demands on the network.
- Offers fast discovery of newly arrived devices.
- Creates a good platform for power-conserving operation.

As a necessary co-development to the service discovery algorithm, a description and query format was also developed. This format also offers qualities useful to efficient service discovery:

- Allows both user-defined and standardized type extensions.
- Permits legacy queries to be satisfied by upgraded service offerings, while still allowing queries from newer devices to take advantage of extended description detail.
- Offers compact encoding.

## 8.2 Further Results Arising From This Thesis

While the DEAPspace style of algorithm is a fast and effective method of service discovery, it is not appropriate for every type of network. In particular, it requires peer-to-peer communication. The start of this research in 1998 was also an early stage in the development of the Bluetooth communication standard. Because Bluetooth was also targetting about a ten metre range, the author followed the developing standard with great interest. When the Bluetooth SIG chose to use a centralized communication model, he stayed in contact with the committee out of general interest.

In the end, service discovery was separated from device discovery, except for a minimal “class of device” value that can be used in a device enquiry. The timeliness of this service discovery can be greatly improved by using a push model to prefetch discovery requests to device. As a direct result of work performed on this thesis, a method for applying a push model to improve responsiveness to change in Bluetooth was proposed [Nid01].

Although the resource discovery algorithm was designed to keep reasonable limits on the amount of data actually sent to the network, reducing this amount is still useful. By using service version numbers, greatly compressed lists can allow the data volume to be slashed (during normal operation) to only two integers and an address per service element. New devices can still discover the group promptly, but the network traffic is reduced. The details of this improvement have yet to be studied, but it is another interesting contribution [HHMN01].

## 8.3 Future Work

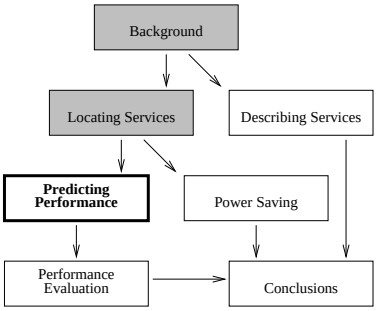
As noted in Section 6.5, the choice of configuration parameters is fostered by knowing approximately how many devices can be expected to be in a single group. In theory, it should be enough for devices to define $X$, $X'$, expiry time, and time to start worrying as functions of the length of the list of known (unexpired) devices. In practice, something more sophisticated may be necessary to coordinate devices that do not agree on these functions; if this becomes a problem, it could best be dealt with through an independent standardization effort.

What functions would be best for these purposes, and how serious would the artifacts be when differently configured devices encountered each other? The former question is answered most simply by looking at the procedure used in Section 6.5, and extrapolating. The latter is an interesting area for future work. Intuitively, it seems that there would be a brief period of confusion, quickly resolved by the speed in which mutual discovery is completed.

# Appendix A

# Generating Functions

*Chapter 4 makes use of combinatorial tools called generating functions. This section provides an introduction specifically targeted at making that chapter clear. For a more complete introduction to the subject, a textbook on combinatorics will be useful [Rio58, Mac60].*




---

The problem of combining elements from various sets is very similar to the well studied problem of combining variables from the various terms of an algebraic expression. For example:
**Question 1** *With exactly three coins, how many permutations of heads and tails are possible?*

Each of the three must have one of two values, call these $h$ or $t$. The possible sets are these:

$$hhh, hht, hth, htt, thh, tht, tth, ttt \tag{A.1}$$

By disregarding the order in which the coins are arranged, this can be rewritten as follows:

$$h^3, h^2t, h^2t, ht^2, h^2t, ht^2, ht^2, t^3 \tag{A.2}$$

Simply writing this as a sum (a series) will not hide any properties, since summing algebraic terms will collect only the terms with the same exponents, and therefore the same number of heads and tails (i.e., permutations of the same combination).

$$h^3 + 3 \times h^2t + 3 \times ht^2 + t^3 \tag{A.3}$$

This new representation still shows the eight possible permutations for Question 1, and also shows that this is composed of four groups, two having multiplicity three, and two having multiplicity one.

Further examination of Expression A.3 offers some new insight:

$$h^3 + 3h^2t + 3ht^2, t^3 = (h + t)(h + t)(h + t) \tag{A.4}$$

This highlights the similarity between choosing combinations of choices and combining terms in algebraic multiplication. In the case of multiplying $(h + t)^3$, the exponents on $h$ and $t$ show the number of each that were chosen, and the coefficient shows the number of ways that those counts can be picked. In this case, the actual value of $h$ and $t$ *has no meaning*. The variables are simply placeholders for their coefficients and exponents.

**Question 2** *Choosing one bar from each of three boxes, each box contains iron bars weighing 1, 5, and 8 kilograms, how many total weights can be formed.*

This time, we have only one thing to count, so only one variable is needed:

$$\begin{aligned}(x^1 + x^5 + x^8)^3 = \quad & x^3 + 3x^7 + 3x^{10} + 3x^{11} + 6x^{14} + \\ & 3x^{17} + x^{15} + 3x^{18} + 3x^{21} + x^{24}\end{aligned} \tag{A.5}$$

We can now see from the coefficient of $x^{14}$ that there are six ways to form fourteen kilograms (one of each type has 3! ways to be chosen.)

Suppose we change the weights to 1, 2, or 3 kilograms, and also permit some boxes to be skipped all together. The new generating function can be written like this:

$$g(x) = (x^0 + x^1 + x^2 + x^3)^3 = (1 + x + x^2 + x^3)^3 \tag{A.6}$$

The fully expanded version of $g(x)$ is long enough to be omitted from this, but whatever it actually looks like, it is equivalent to Expression A.6. This is where the distinction between algebra and combinatorics gets blurred. This notation was meant as a short way of writing long series, but we also know this:

$$(1 + x + x^2 + x^3) = \frac{1 - x^4}{1 - x} \tag{A.7}$$

Thus we have the closed form for g(x):

$$g(x) = \left(\frac{1 - x^4}{1 - x}\right)^3 \tag{A.8}$$

**Question 3** *How many ways can Swiss coins be used to make change for 1 Swiss franc?*

Begin with a generating function considering zero or more coins of each value 1, 5, 10, 20, 50, and 100 (1 centime coins still exist, and we can ignore coins of value more than 1 franc.) In this case, each coin denomination replaces the boxes of weights in the previous question.

$$\begin{aligned} f(x) &= \lim_{n \to \infty} (1 + x + x^2 + \cdots + x^n)(1 + x^5 + x^{10} + \cdots + x^{5n})(1 + x^{10} + x^{20} + \cdots + x^{10n}) \\ &\quad (1 + x^{50} + x^{100} + \cdots + x^{50n})(1 + x^{100} + x^{200} + \cdots + x^{100n}) \\ &= \frac{1}{1-x}\frac{1}{1-x^5}\frac{1}{1-x^{10}}\frac{1}{1-x^{20}}\frac{1}{1-x^{50}}\frac{1}{1-x^{100}} \end{aligned} \tag{A.9}$$

In this question, it is clear that expanding all the combinations would take too long, so the closed form has helped us. Whatever the coefficient of $x^{100}$ is in the expanded form of $f(x)$, that is the number of ways to make change for a franc.

## A.1 Taylor Series

The problem with having a closed form solution, as in Equation A.9, is that it is not so easy to get the final answer. To do this, we must expand $f(x)$ back into a series. One way to expand an expression into an infinite series is to use Taylor series:

$$\begin{aligned} F(x) = \; & F(y) + F'(y)(x-y) + F''(y)\tfrac{(x-y)^2}{2} + F^{(3)}(y)\tfrac{(x-y)^3}{3!} \\ & + \cdots + F^{(n)}(y)\tfrac{(x-y)^n}{n!} + \ldots \end{aligned} \tag{A.10}$$

It is common to choose $y = 0$ for Equation A.10, as it makes the evaluation easier, so long as all the derivatives of $F(x)$ are defined at zero. Calculating the hundredth derivative of Equation A.9 would be difficult, but using a computer makes is feasible, yielding the answer to Question 3: There are 344 ways to make 1 franc from Swiss coins.

## A.2 Counter Variables

Sometimes the weight is not the only important quantity to measure. In the coin example from above, $f(x)$ allows the value of the coins to be counted, but ignores the number of coins used. A modified generating function that allows both things to be counted would be the following:

$$h(x,z) = \frac{1}{1-zx}\frac{1}{1-zx^5}\frac{1}{1-zx^{10}}\frac{1}{1-zx^{20}}\frac{1}{1-zx^{50}}\frac{1}{1-zx^{100}} \tag{A.11}$$

In the new equation, the power of $z$ is increased by one every time a coin is used, while the power of $x$ is simultaneously increased by the value of the coin. Expanding the Taylor series of $h$ about $x$ gives the coefficient of $x^6$ as $(z^6 + z^2)$, which translates to saying there are two ways to make six centimes in change, one with six coins, and one with two coins.

Alternately, expanding about $z$ provides the set of values that can be formed with a particular number of coins. Naturally, the number of values that can be formed by a given number of coins is large. For example, the coefficient of $z^2$ in the expansion of $h(x, z)$ is the following:

$$\begin{aligned} &x^2 + x^6 + x^{10} + x^{11} + x^{15} + x^{20} + x^{21} + x^{25} + x^{30} + x^{40} + x^{51} + x^{55} \\ &+x^{60} + x^{70} + x^{100} + x^{101} + x^{105} + x^{110} + x^{120} + x^{150} + x^{200} \end{aligned} \tag{A.12}$$

The 21 terms of this coefficient tell us that 21 different values can be formed with exactly two coins from our set. Each particular value can be formed in exactly one way, as indicated by all the coefficients being one; and the actual values in question are the various exponents of $x$.

## A.3 Differentiation

As these terms get long, as in Expression A.12 above, some common operations become helpful. For example, suppose we want the average value of two coins, chosen from the set of 1, 5, 10, 20, 50, and 100 centimes, where order does not matter. We know there are $\sum_{i=1}^{6} i = 21$ possible combinations of two coins from a set of six types, so the average value is just the sum of the values of all these combinations divided by 21. Notice, however, that this is just the sum of all the coefficients multiplied by their corresponding exponent of $x$ (i.e., each achievable value is multiplied by the number of ways to form that value, and the resulting products are summed.) This operation is precisely the derivative w.r.t. $x$, evaluated at $x = 1$! In the case of Expression A.12, the first derivative w.r.t. $x$ is the following:

$$\begin{aligned} &2x^1 + 6x^5 + 10x^9 + 11x^{10} + 15x^{14} + 20x^{19} + 21x^{20} + 25x^{24} + 30x^{29} \\ &+40x^{39} + 51x^{50} + 55x^{54} + 60x^{59} + 70x^{69} + 100x^{99} + 101x^{100} \\ &+105x^{104} + 110x^{109} + 120x^{119} + 150x^{149} + 200x^{199} \end{aligned} \tag{A.13}$$

The exponents no longer have any significant meaning, but by setting $x = 1$, they disappear anyway, leaving just the sum: 1302. The average value of two coins, chosen from the specified set, independent of order, is 1302 / 21 = 62 centimes.

For a second example, suppose order does matter, and adapt the generating function accordingly:

$$g(x) = (x + x^5 + x^{10} + x^{20} + x^{50} + x^{100})^2 \tag{A.14}$$

Equation A.14 drops the $z$ used above, since it is no longer necessary. In this equation, there will always be exactly two choices from the set of possibilities, so the only power of $z$ would have been two anyway.

As in the first example, we now have a generating function for our answer. Differentiate w.r.t. $x$, then set $x = 1$ to get the sum of the values, as shown in Equation A.15,

$$\begin{aligned} g'(x) &= 2(x + x^5 + x^{10} + x^{20} + x^{50} + x^{100}) \\ &\quad (1 + 5x^4 + 10x^9 + 20x^{19} + 50x^{49} + 100x^{99}) \\ g'(1) &= 2(1+1+1+1+1+1)(1+5+10+20+50+100) = 2232 \end{aligned} \tag{A.15}$$

then divide by the number of possible choices ($6^2 = 36$) to find that the average value of 62 centimes is the same regardless of whether you count the order of selection of not. Note also that the number of choices is just the sum of the coefficients before they are multiplied by the exponents, which is $g(1)$.

## A.4 Regular Expressions

If a language can be defined by a regular expression (i.e., if it is context free), then it can readily be expressed as an equivalent generating function. For example, this regular language:

$$((AA)^*B)^+ = B + BB + AAB + BBB + BBBB + AABB + BAAB + \ldots$$

includes all strings of $A$ and $B$ that end in at least one $B$, and have all groups of $A$ with even size. The basic procedure is to replace a sequence definition from the language definition with a series having the sum of counter variable terms with exponents equal to the multiplicity of their corresponding language symbol. For example, the following guidelines can be generally applied:

$$X \to x$$

$$X^* \to \sum_{i=0}^{\infty} x^i \to \frac{1}{1-x}$$

$$X^+ \rightarrow \sum_{i=1}^{\infty} x^i \rightarrow \frac{x}{1-x}$$

Recursive application of these rules to our language leads to the following generating function:

$$f(a,b) = \frac{\frac{b}{1-a^2}}{1-\frac{b}{1-a^2}} \tag{A.16}$$

There are other ways of writing this language; for example, the following is true:

$$((AA)^*B)^+ \equiv (B+AA)^*B$$

This new representation leads to the new generating function:

$$f_2(a,b) = \frac{b}{1-(b+a^2)} \tag{A.17}$$

Not surprisingly, a quick inspection reveals that these two generating functions for equivalent languages are also equivalent (multiply Equation A.16 by $\frac{1-a^2}{1-a^2}$). Notice that $f(a,b)$ counts both the number of $A$s and the number of $B$s. If we want to study only the lengths of valid strings, there is no particular reason to count them separately; simply substitute $x$ for both variables, creating a single counter with an exponent equal to the total length of the string, and expand the resulting equation via a Taylor series:

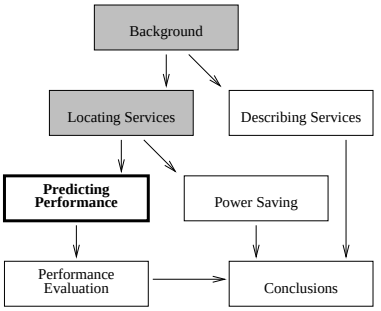
$$\frac{x}{1-x-x^2} = x + x^2 + 2x^3 + 3x^4 + 5x^5 + 8x^6 + 13x^7 + O(x^8) \tag{A.18}$$

Equation A.18 shows us, among other things, that precisely 13 strings of length 7 are members of the language $(B+AA)^*B$. Application of these techniques is often much faster than explicit analysis of a large sequence, as is the case in Chapter 4.

# Appendix B

# Newton's Approximation Method

*This very briefly reviews Newton's method for approximating the zero intercept of arbitrary well-behaved functions.*




---

Section 4.2 makes use of Newton's method for approximating the solution to Equation 4.7: $x = \left(\frac{1}{n}\right)\left(1 - \left(\frac{1}{\frac{1}{x(1-x)^{b+1}} - \frac{1}{x}}\right)\right)$ This brief appendix illustrates some C code for finding this solution.

Recall first that Newton's method refers to a technique (demonstrated by Isaac Newton[1]) of successively refined approximations to an equation of the form $f(x) = 0$. Stated simply: starting with a "reasonable" guess $x_0$ (i.e. any guess such that no inflection points of $f(x)$ exist on the range between the exact solution and $x_0$, and the slope of $f(x)$ at $x_0$ is bounded less than infinity), finding the intercept $x_1$ of the tangent of $f(x)$ at $x = x_0$ will be a better approximation of the correct solution than $x_0$. Since calculating the intercept of the tangent of $f(x)$ at each successive approximation $x_i$ involves evaluating $f(x_i)$ anyway, repeating until $\|f(x_i)\| < tolerance$ is easily incorporated into an implementation of this algorithm. The exact

[1] Actually, "Netwon's method" was first published by Joseph Raphson, a colleague of Newton, in his book *Analysis aequationum universalis* in 1690, but was demonstrated by Newton for a particular application in *Method of Fluxions*, written 1671, but published 1736 [OR96].

solution can thereby be approximated to within an arbitrary $tolerance$.

After subtracting $x$ from both sides of Equation 4.7, the following code intersects the resulting function with zero, using its first derivative w.r.t. $x$ (Equation B.1).

$$\left(1 - \frac{x}{-1 + (1-x)^{-1-b}}\right)^{n-1} \left(-\frac{1}{-1 + (1-x)^{-1-b}} - \frac{(-1-b)(1-x)^{-2-b}x}{(-1 + (1-x)^{-1-b})^2}\right) \quad \text{(B.1)}$$

The parameters $n$ and $b$ correspond to the variables of the same names as defined in Chapter 5. The function uses $\frac{1}{n}$ as an initial guess, and refines the approximation to within a specified tolerance of the actual value.

```
double approx(double n, double b) {
    double x,Ex,u,fprime;

#   define one ((double)1.0)
#   define two ((double)2.0)
#   define tolerance ((double)0.000001)

    x = one / n; /* initial guess. */
    Ex = (-one + pow((one-x),(-b-one)))/x;
    u = (one/n)*pow((one-(one/Ex)),n);

    while(fabs(u - x) > tolerance) {
        fprime = pow(one-(x/(-one+pow(one-x,-one-b))),n-one)
                 * ((-one/(-one+pow(one-x,-one-b)))
                    -(((-one-b)*pow(one-x,-two-b)*x)
                      /(pow(-one+pow(one-x,-one-b),two))))
                 - one;
        x = x - ((u-x)/fprime);
        Ex= (-one + pow((one-x),(-b-one)))/x;
        u = (one/n)*pow((one-(one/Ex)),n);
    }
    return Ex;
}
```

# Appendix C

# Simulation Code

```
#define DEVICES     6
#define ROUNDS      100000
long    LOSS_PROB = 10;

                    /* Save bcast gaps from 0-20 seconds in 10 milli blocks */
#define GAP_COUNT   2000
#define GAP_WIDTH     10
#define GAPS_PER_SEC 100

#define EXPIRY          115000

#define NORM_TOUT_MAX   15000
#define NORM_TOUT_MIN   10000
#define NORM_WIDTH      (NORM_TOUT_MAX - NORM_TOUT_MIN)
#define SHORT_TOUT_MAX   5000
#define SHORT_TOUT_MIN   3333
#define SHORT_WIDTH     (SHORT_TOUT_MAX - SHORT_TOUT_MIN)

#define BIG_RAND (0xfffff)
#define NormBcast ((((double)(random() & BIG_RAND) * NORM_WIDTH) \
                    / (double)BIG_RAND) + NORM_TOUT_MIN)
#define ShortBcast ((((double)(random() & BIG_RAND) * SHORT_WIDTH) \
                    / (double)BIG_RAND) + SHORT_TOUT_MIN)
```

```
#define LOSS_LINE (LOSS_PROB * BIG_RAND / 100)
#define IS_LOST ((random() & BIG_RAND) < LOSS_LINE)

int main(int argc, char **argv) {
    long nextBcast[DEVICES];
    long panics[DEVICES];
    int  panic_flag[DEVICES];
    long panic_runs[DEVICES];
    long timeout[DEVICES][DEVICES];
    long actualGaps[GAP_COUNT];
    long round, time;
    long tmpTime, gap, lv;
    int s, r, i, winningDev;

    if(argc != 2) {
        fprintf(stderr, "Format: %s <loss prob>\n", argv[0]);
        return;
    }
    sscanf(argv[1], "%ld", &LOSS_PROB);

    for(s=0; s<DEVICES; s++) {
        nextBcast[s] = NormBcast;
        panics[s] = 0;
        panic_flag[s] = 0;
        for(r=0; r<DEVICES; r++) {
            timeout[s][r] = EXPIRY;
        }
    }
    for(i=0; i<GAP_COUNT; i++) actualGaps[i]=0;
    srandom(50);

    time = 0;
    for(round=0; round<ROUNDS; round++) {
```

```
        /** Find next to broadcast: yes, ties go to the lower-numbered de-
vice,
         **                          but these should be very rare.  */
        tmpTime = nextBcast[0];
        winningDev = 0;
        for(i=1; i<DEVICES; i++) {
            if(nextBcast[i] < tmpTime) {
                winningDev=i;
                tmpTime = nextBcast[winningDev];
            }
        }

        /** Record Gap **/
        gap = (tmpTime - time) / GAP_WIDTH;
        if(gap < 0) gap=0;
        else if(gap >= GAP_COUNT) gap = GAP_COUNT-1;
        actualGaps[gap]++;

        /** Update the time **/
        time = tmpTime;

        /** Reset devices that receive the broadcast **/
        for(i=0; i<DEVICES; i++) {
            if(i==winningDev) nextBcast[i] = time + NormBcast;
            else if(!IS_LOST) {
                for(s=0; s<DEVICES; s++) {
                    if(s==winningDev) timeout[i][s] = time + EXPIRY;
                    else if(s==i) {
                        if(timeout[winningDev][i] < time + NORM_TOUT_MAX) {
                            nextBcast[i] = time + ShortBcast;
                            panics[i]++;
                            if(!panic_flag[i]) panic_runs[i]++;
                            panic_flag[i] = 1;
```

```
                        } else {
                            nextBcast[i] = time + NormBcast;
                            panic_flag[i] = 0;
                        }
                    } else if(timeout[i][s] < timeout[winningDev][s]) {
                        timeout[i][s] = timeout[winningDev][s];
                    }
                }
            }
        }
    }
/** Outputting of results omitted **/
}
```

## C.1 Slotted Protocol Simulation

```
#define DEVICES     6
#define ROUNDS      100000
long    LOSS_PROB = 10;
#define BIG_RAND (0xfffff)
#define LOSS_LINE (LOSS_PROB * BIG_RAND / 100)
#define IS_LOST ((random() & BIG_RAND) < LOSS_LINE)
#define SLOT_WIDTH 10000

int main(int argc, char **argv) {
    long s, missed, xmits, time, round;
    srandom(50);
    for(LOSS_PROB=0; LOSS_PROB<=100; LOSS_PROB++) {
        xmits = 0;
        time = 0;
        for(round=0; round<ROUNDS; round++) {
            missed=DEVICES;
            while(missed) {
```

```
                missed--;                   /** at least one device won **/
                for(s=missed; s; s--) {
                    if(!IS_LOST) missed--;  /** those that heard, are done **/
                }
                xmits++;
            }
            time += SLOT_WIDTH;
        }
    }
/** Outputting of results omitted **/
}
```

# Index

# Bibliography


[Acc83] M. Accetta. Resource location protocol. Technical Report RFC 887, Carnegie-Mellon University, December 1983.

[AK83] Susan F. Assmann and Daniel J. Kleitman. The number of rounds needed to exchange information within a graph. *Discrete Applied Mathematics*, 6:117–125, 1983.

[Arc93] Architecture Projects Management Ltd. *ANSAware 4.1: Application Programming in ANSAware*, February 1993.

[BCSW98] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (DREAM). In *Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98)*, October 1998.

[Blu01] Bluetooth SIG. *Bluetooth Specification Version 1.1*, 2001.

[BPSMM00] Tim Brah, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible markup language (xml) 1.0 (second edition). Technical Report http://www.w3.org/TR/2000/REC-xml-20001006, W3C Recommendation, October 2000.

[Chh96] Harshal S. Chhaya. Performance evaluation of the IEEE 802.11 MAC protocol for wireless LANs. Master's thesis, Illinois Institute of Technology, Chicago, Illinois, USA, 1996.

[CS94] Thomas Casavant and Mukesh Singhal. *Readings in Distributed Computing Systems*. IEEE Computer Society Press, 1994.

[Dee91] S. Deering. ICMP router discovery messages. Technical Report RFC 1256, Xerox PARC, September 1991.

[DnYS98] Samir R. Das, Robert Casta neda, Jiangtao Yan, and Rimli Sengupta. Comparative performance evaluation of routing protocols for mobile, ad hoc networks. In *International Conference on Computer Communications and Networks*, pages 153–161, October 1998.

[Dro93] R. Droms. Dynamic host configuration protocol. Technical Report RFC 1541, Bucknell University, October 1993.

[ETS96] ETSI. *High Performance Radio Local Area Network (HIPERLAN); Type 1; Functional Specification*, October 1996.

[FB96] N. Freed and N. Borenstein. Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies. Technical Report RFC 2045, IETF Network Working Group, November 1996.

[FH00] Edward H. Frank and Jack Holloway. Connecting the home with a phone line network chip set. *IEEE Micro*, 20(2):27–37,39, March–April 2000. http://www.HomePNA.org/docs/paper500.pdf.

[GPVD99] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. Technical Report RFC 2608, IETF Network Working Group, June 1999.

[Her94] Andrew Herbert. An ANSA overview. *IEEE Network*, pages 18–23, January/February 1994.

[HGBV01] J.-P. Hubaux, Th. Gross, J.-Y. Le Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: the Terminodes project. *IEEE Communications Magazine*, January 2001.

[HH94] Andy Harter and Andy Hopper. A distributed location system for the active office. *IEEE Network*, 8(1), January 1994.

[HHL88] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[HHM+00] Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, and Andreas Schade. DEAPspace – transient ad-hoc networking of pervasive devices. Technical Report RZ 3233, International Business Machines Corporation, August 2000.

[HHM+01] Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, and Andreas Schade. DEAPspace: Transient ad-hoc networking of pervasive devices. *Computer Networks*, 35(4):411–428, March 2001.

[HHMN01] Reto Hermann, Dirk Husemann, Michael Moser, and Michael Nidd. A mechanism for prompt and efficient service discovery in wireless networks. Technical Report European Patent Application 01112542.4, International Business Machines Corporation, May 2001.

[Hom00] HomeRF Working Group, Inc. *Introduction to the HomeRF Technical Specification*, December 2000.

[IEE97] IEEE Computer Society, LAN MAN Standards Committee. *Wireless LAN Medium Access*, June 1997.

[JMHJ01] David B. Johnson, David A. Maltz, Yih-Chun Hu, and Jorjeta G. Jetcheva. The dynamic source routing protocol for mobile ad hoc networks. Technical Report draft-ietf-manet-dsr-05.txt, Internet-Draft: IETF MANET Working Group, March 2001. work-in-progress.

[JMQ00] Philippe Jacquet, Paul Muhlethaler, and Amir Quayyum. Optimized link state routing protocol. Technical Report draft-ietf-manet-olsr-01.txt, Internet-Draft: IETF MANET Working Group, February 2000.

[Kle95] Leonard Kleinrock. Nomadic computing – an opportunity. *ACM SIGCOMM Computer Communication Review*, 25(1):36–40, January 1995.

[Lan54] H. G. Landau. The distribution of completion times for random communication in a task-oriented group. *Mathematical Biophysics*, 16:187–201, 1954.

[LJC+00] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'00)*, August 2000.

[LR93] Arthur L. Liestman and Dana Richards. Perpetual gossiping. *Parallel Processing Letters*, 3(4):347–355, 1993.

[Mac60] Maj. Percy A. MacMahon. *Combinatory Analysis.* Chelsea Publishing Co., New York, N.Y., 1960. Original published in two volumes (1915, 1916) by Cambridge University Press.

[MBJJ99] David A. Maltz, Josh Broch, Jorjeta Jetcheva, and David B. Johnson. The effects of on-demand behavior in routing protocols for multihop wireless ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1439–1453, 1999.

[Mic00] Microsoft Corporation. *Universal Plug and Play Device Architecture*, June 2000.

[MR96] J. Myers and M. Rose. Post office protocol - version 3. Technical Report RFC 1939, Carnegie-Mellon University and Dover Beach Consulting, Inc., May 1996.

[Nid00] Michael Nidd. Timeliness of service discovery in DEAPspace. In *The 29th International Conference on Parallel Processing (ICPP 2000) Workshop on Pervasive Computing*, pages 73–80, August 2000.

[Nid01] Michael Nidd. Improved service discovery for bluetooth. Technical Report European Patent Application 01810156.8, International Business Machines Corporation, February 2001.

[OR96] J. J. O'Connor and E. F. Robertson. Joseph raphson. Technical Report http://www-history.mcs.st-andrews.ac.uk/history/Mathematicians/Raphson.html, University of St. Andrews, Scotland, 1996.

[Pas01] Robert Pascoe. Building networks on the fly. *IEEE Spectrum*, March 2001.

[PB94] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *ACM SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[Per96] C. Perkins. IP mobility support. Technical Report RFC 2002, IBM, October 1996.

[PH99] Marc R. Pearlman and Zygmunt J. Haas. Determining the optimal configuration for the zone routing protocol. *IEEE Journal on Selected Areas in Communication*, 17(8):1395–1414, 1999.

[Plu82] David C. Plummer. An ethernet address resolution protocol. Technical Report RFC 826, IETF Network Working Group, November 1982.

[Rio58] John Riordan. *An Introduction to Combinatorial Analysis.* John Wiley & Sons, Inc., 1958.

[Sal99] The Salutation Consortium Inc. *Salutation Architecture Specification*, 2.0c edition, June 1999.

[Sie00] Jon Siegel. *CORBA 3 Fundamentals and Programming.* John Wiley & Sons, Inc., May 2000.

[Sil00] Silicon Wave, Inc. *SiW1502 Radio Modem IC Data Sheet*, November 2000.

[Sun99] Sun Microsystems Inc. *Jini Architectural Overview*, 1999.

[SWR98] Suresh Singh, Mike Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98)*, October 1998.

[TBJ00] Terry Todd, Frazer Bennett, and Alan Jones. Low power rendezvous in embedded wireless networks. In *MobiHOC: First annual workshop on mobile and ad hoc networking and computing*, pages 107–118, August 2000.

[vHH01] Srdan Čapkun, Maher Hamdi, and Jean-Pierre Hubaux. Gps-free positioning in mobile ad-hoc networks. In *34th Hawii International Conference on System Sciences (HICSS'01)*, January 2001.

[ZR99] Michele Zorzi and Ramesh R. Rao. Perspectives on the impact of error statistics on protocols for wireless networks. *IEEE Personal Communications*, pages 32–40, October 1999.

# About the author

Michael E. Nidd
Rütistrasse 66
CH-8134 Adliswil, Switzerland
mni@zurich.ibm.com
01 / 709 10 74

**Education**

Sept. 1993 – Dec. 1995
Master of Mathematics (Computer Science), Shoshin Distributed Systems Research Group, University of Waterloo, Waterloo, Ontario. Specific research in wireless networks, and the efficient use of computers in cellular network planning.

Sept. 1988 – Apr. 1993
Bachelor of Mathematics (Co-op Honours Computer Science, minor in Physics) University of Waterloo, Waterloo, Ontario. Graduated with distinction.

**Work Experience**

Mar. 1998 – present
Researcher, IBM Zürich Research Laboratory

May 1995 – Feb. 1998 (full-time from Oct. 1995)
Research Associate, Shoshin Mobile Project, University of Waterloo

Sept. – Dec. 1991 and May – Aug. 1992
Software systems developer, Organization Metrics Inc.

May – Aug. 1990 and Jan. – Apr. 1991
Database programmer, Cherniak, Gottleib, and Co.

Sept. – Dec. 1989
Software developer, Advanced Technology Centre, IBM Canada.

Jan. – Apr. 1989
Researcher, Information Technologies, IBM Canada.

July – Sept. 1987 and June – Sept. 1988
Programmer, Organization Metrics, Inc.

**Publications**

- Michael Nidd. Service discovery in DEAPspace. *IEEE Personal Communications*, 8(4), August 2001.
- Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, and Andreas Schade. DEAPspace: Transient ad-hoc networking of pervasive devices. *Computer Networks*, 35(4):411–428, March 2001.
- Michael Nidd Method, Network Device, and Computer Program Product for performing Service Discovery in a Pervasive Network. Technical report, International Business Machines Corporation, February, 2001. European Patent Application 01810156.8
- Michael Nidd. Timeliness of service discovery in DEAPspace. In *The 29th International Conference on Parallel Processing (ICPP 2000) Workshop on Pervasive Computing*, pages 73–80, August 2000. Also IBM Technical Report RZ 3228.

- Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, and Andreas Schade. DEAPspace – transient ad-hoc networking of pervasive devices. In *MobiHOC: First annual workshop on mobile and ad hoc networking and computing*, pages 133–134, August 2000.
- Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, and Andreas Schade. Adjacency-bound service discovery. Technical report, International Business Machines Corporation, August 2000. European Patent EP1024628A1.
- Michael Nidd. Reducing power use in DEAPspace service discovery. Technical Report RZ 3252, International Business Machines Corporation, July 2000.
- Stefan Hild, Dirk Husemann, and Michael Nidd. Service advertisements in wireless local networks. Technical report, International Business Machines Corporation, July 2000. European Patent EP1022876A1.
- Michael Nidd. Service discovery in DEAPspace. Technical Report RZ 3149, International Business Machines Corporation, December 1999.
- D. Kidston, J.P. Black, T. Kunz, M. Nidd, M. Lioy, B. Elphick, and M. Ostrowski. Comma, a communication manager for mobile applications. In *Wireless 98*, pages 103–116, Calgary, Alberta, Canada, July 1998. TRLabs, TRIO, IEEE Canada.
- Michael Nidd, Stephen Mann, and James P. Black. Using ray tracing for site-specific indoor radio signal strength analysis. In *47th annual IEEE Vehicular Technology Conference*, pages 795–799, May 1997.
- Michael Nidd, Thomas Kunz, and Ertugrul Arik. Prefetching DNS lookup for efficient wireless WWW browsing. In *Wireless 97*, Calgary, Alberta, Canada, July 1997. TRLabs, TRIO, IEEE Canada.
- Michael Nidd, Thomas Kunz, and James P. Black. Wireless applications and API design. In *International Workshop on Quality of Service*, pages 83–85. IFIP, GMD FOKUS, France Telecom, Telecom Paris, March 1996.
- Michael Nidd. Using ray tracing for site-specific indoor radio signal strength analysis. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 1995.