RZ 3149 (# 93195) 06/14/99
Computer Science/Mathematics 8 pages

Research Report

Service Discovery in DEAPspace

Michael Nidd

IBM Research Division
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

IBM Research Division
Almaden · Austin · Beijing · Haifa · T.J. Watson · Tokyo · Zurich

Service Discovery in DEAPspace

Michael Nidd

IBM Research Division, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

A new algorithm is presented through which a computing device can detect the presence of neighbouring devices, share configuration and service information with those devices, and also notice when devices become unavailable. This solution minimalizes the number of transmissions required from individual devices, while allowing devices with better power availability to further reduce the transmission requirement imposed on the other devices. Evidence is presented to demonstrate the effectiveness of this algorithm.

1 Introduction

Computers are getting smaller, and computer usage patterns are becoming increasingly nomadic. It is our belief that communication between portable computers is essential to improve their usefulness. Furthermore, the frequency and complexity of the reconfigurations necessary to maintain useful communication and resource sharing is more than a user should be expected to deal with manually. To this end, this paper proposes a system through which a computing device can detect the presence of neighbouring devices, share configuration and service information with those devices and also notice when devices become unavailable, and reconfigure itself accordingly.

Our solution minimalizes the number of transmissions required from individual devices while allowing devices with better power availability to further reduce the transmission requirements on other devices without explicit network reconfiguration (i.e. without introducing a dependency on any single device).

2 Differences from Wired Networks

When a protocol is designed, some things must be assumed 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 [7]. IP is insufficient for addressing devices that move between networks, so Mobile IP has been introduced [8]. Existing service discovery protocols do not perform adequately in wireless ad-hoc networks, so a new alternative is required.

In a wired network, using a centralized server is a reasonable option, and low round-trip times can be expected for transmissions. The latter assumption, while common to such current solutions as RLP, DHCP, Salutation, and Jini [1, 4, 9, 10], may not be inherent to centralized approaches, but the former is. Therefore, I will first use decentralization as an argument against using these existing protocols. After that point has been made, I will discuss the remaining options with consideration given to the large round-trip times that result from slotted access protocols and lower bit-rates than in wired environments.

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. While, in wired networks, services like DHCP can reasonably be expected to be provided by the same people who maintain the physical equipment, ad-hoc networks cannot be expected to provide such infrastructure. An ad-hoc network 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 networks 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 network. If devices can be members of more than one network, the scenario gets increasingly complicated.

Accepting that alternatives to the centralized approach are required, it is necessary to look for a way to make the distributed approach work. The two obvious categories are the push- and pull-style solutions. For a pull-style solution to work, a device must decide when to request the information. One option, to send queries only when the information is actually required, is addressed in Section 3. Alternatively, the device could maintain a view of the available services. Ideally, for this option, a request would be sent every time the set of services available to the particular client changed. This 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, Appletalk [2] is an example of one that has been successful on wired networks. If a public service, such as a printer, is connected to an Appletalk network, it periodically broadcasts its name and address to all listeners. The IEEE 802.11 standard, a protocol designed for wireless LANs [3, 6] also uses periodic broadcasts, although it broadcasts the existence of a device (address) rather than a service. The similarity between these protocols 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. One might imagine, for example, that a broadcast medium, like the airwaves, would offer cheap broadcasts. Unfortunately, air protocols for pervasive devices tend to value power consumption very highly. Frequently, 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, limiting the total bytes per second, and also means 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 the Appletalk approach unworkable.

The 802.11 approach is designed for wireless media, and deals with the problem much more appropriately. This solution allows a passive, Appletalk-like solution, except advertising broadcasts are slotted, putting 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.

In the remainder of this article I will argue that the better solution is a modified version of this passive search. The primary difference is that each member device sends the full list of known services (including its own), rather than sending only its own information in the beacon signals, Section 3 argues this idea of pro-actively shared lists versus just-in-time searches. Section 4 describes the DEAPspace algorithm [5] for sharing service lists. Finally, Section 5 presents some simulated and theoretical results in defence of those ideas, leading to discussion of those results and planned future work.

3 Pro-Active vs. Just-In-Time

A pro-active system generates an image of its environment in anticipation of future necessity. This strategy has the advantage of making individual queries fast and cheap, since they remain local to the querying device, and has the disadvantages of requiring communication overhead to maintain these images on all devices, and of relying on potentially outdated information. This final point relates the fact that if queries are to be kept local they can be based only on the most recent information received about the environment. The older this information is permitted to be, the more potential there is that the query response will reflect an out of date image.

A just-in-time system, in contrast, requires no transmission overhead except when an actual query is being processed, and query responses are always based on the current state of the network. These advantages are at the cost of longer delays in query processing, and more susceptibility to temporary transmission problems.

Although different in many respects, these systems also share at least one important trait when used in ad-hoc networks: they will both be required to address previously unknown devices. That shared challenge means that, at some time during their normal operation, a broadcast will be necessary. The distinction is when it will be required.

As I mentioned in Section 2, broadcasts can involve significant overhead and delay. By performing all its broadcasts as part of the background task of maintaining a timely image of the environment, the pro-active algorithm is never required to wait for a broadcast to complete. In contrast, a just-in-time algorithm that does not use a central server must broadcast a request at least once, and wait for it to complete. To gain reliability, as I will demonstrate in Section 5, that broadcast will often have to be repeated, exacerbating the problem.

Thus far, I have argued that response time is the chief problem with the just-in-time solution, but have not yet addressed the problems with the pro-active solution: timeliness of results, and broadcast overhead. I will address the former in Section 5, and the latter in Section 4, describing how our resource discovery algorithm, hereafter referred to as the DEAPspace algorithm, addresses the broadcast overhead required for a pro-active algorithm.

4 The DEAPspace Algorithm

Hopefully, you are now convinced that in some situations it is a good idea for each device to try to build up a timely picture, or "world view," of what services are offered in its area. A service-offering device is most useful when all other devices in communication range know that it is available and what services it is offering. Imagine a group of four such devices, all offering some set of services. What each would like to do is make a reliable broadcast to the other three. The problem with this is, aside from the complexity of reliable broadcast under normal conditions, an individual device might not even be certain what other devices are present to receive its advertisement.

Our idea is for devices to broadcast their entire world-view. If all listening devices incorporate new elements found in each broadcast into their own views, then missing occasional broadcasts will not cause significant problems for any particular device, because the next broadcast by any device that did not miss the first one will include the same information. In this way, we get reliability through continued repetition. Also, since 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.

As the sample implementation in Figure 1 shows, the DEAPspace algorithm has a configuration parameter (the default mean advertisement period) and a random number generator. Since the receipt of an advertisement resets the transmit timer in every device, without a random jitter in the setting of the timer all devices would broadcast together. Aside from causing possible problems at the physical level, this would defeat the goal of the algorithm: reduction of the number of broadcasts from each individual device. The interesting choice is what distribution is used, and what is the mean of that distribution.

If all devices have the same mean advertisement time, then a static system will reach

```
time update(A,B) {
  time m ←EndOfTime
  foreach b ∈ B {
    if(b is my service) {
      if(b.expiry < m) m ←b.expiry
    } else {
      if(∃_{a∈A} b.id = a.id) {
        if(b.expiry < a.expiry)
          a.expiry ←b.expiry
      } else {
        insert b into A
      }
    }
  }
  return m
}
```

```
advertise(A) {
  time tout ←getTimeout(dflt)
  loop(forever) {
    B ←read(tout)
    if(timed out) {
      broadcast(A)
      tout ←dflt
    } else {
      time x ←update(A,B)
      tout ←getTimeout(min(x,dflt))
    }
  }
}
read(t) {
  blocking read with timeout t (from network)
}
getTimeout(t) {
  pick random value with mean t
}
```

update(A,B) shows how a received list B is used to update the stored list A, returning the earliest time at which a service offered by this device will expire from the list at the sending device.
advertise(A) shows the main loop of the algorithm, having the advertisement timer chosen with median value dflt, unless the return value from update(A,B) indicates that a service offered by this device will expire sooner from a list stored at a remote device.

Figure 1: A sample pseudocode implementation of the DEAPspace algorithm

a steady-state wherein advertisements 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.

The mean advertisement time also has another use. If one device is given a smaller value than others, than the majority of broadcasts will be done by that machine. The other devices would broadcast only when they saw their services expiring. This allows devices with good power supplies, such as those connected to a house current, to relieve battery-powered devices of some of their transmit responsibility. Conversely, unusually large values could be used by devices with very poor power supplies. The effects of asymmetric assignment of mean advertisement times will not be further examined in this paper.

5 Performance Evaluation

The DEAPspace algorithm, due to its changing timeout values and race conditions is very difficult to analyse theoretically, and simulation was required for its performance to be examined. For this I used a network simulator with adjustable packet-loss probabilities on a per-pair basis. In the examples presented here, simulated devices 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.

The goal of this first set of trials was to demonstrate the reliability of this algorithm. 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 served to validate the simulation.

5.1 Environment and Tasks

The simulated 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 142 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 services in the advertisement most recently received. Once a service was discovered, data was exchanged. Synchronous communication (between "present" devices) was assumed to be reliable.

5.2 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 1:

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

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 change is ignored in this analysis.

An implementation of this revised algorithm was run on the simulator, with the results shown in Figure 2.

Having validated the simulator, I went on to use it with an implementation of the DEAPspace algorithm. The results, shown in Figure 3, illustrate 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 3 show not only that the DEAPspace algorithm gives at least as timely a picture as the basic algorithm, but 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) at packet-loss rates more than 20%.

6 Discussion and Future Work

The promising results shown in Section 5.2 remove any worries of the DEAPspace algorithm suffering from the the possible disadvantage mentioned in Section 3 of a pro-active protocol

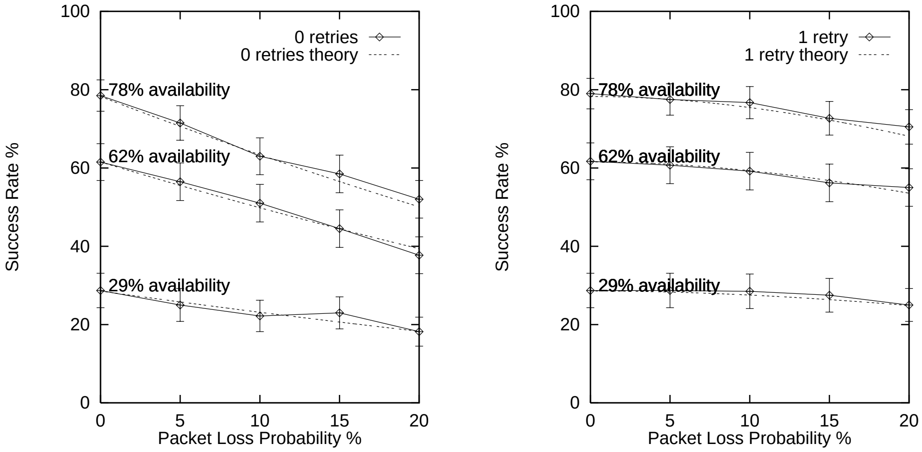

Figure 2: Simulated vs. theoretical results for the revised basic discovery algorithm. **Left:** no retries, **Right:** one retry. Error bars are given for 95% confidence interval.

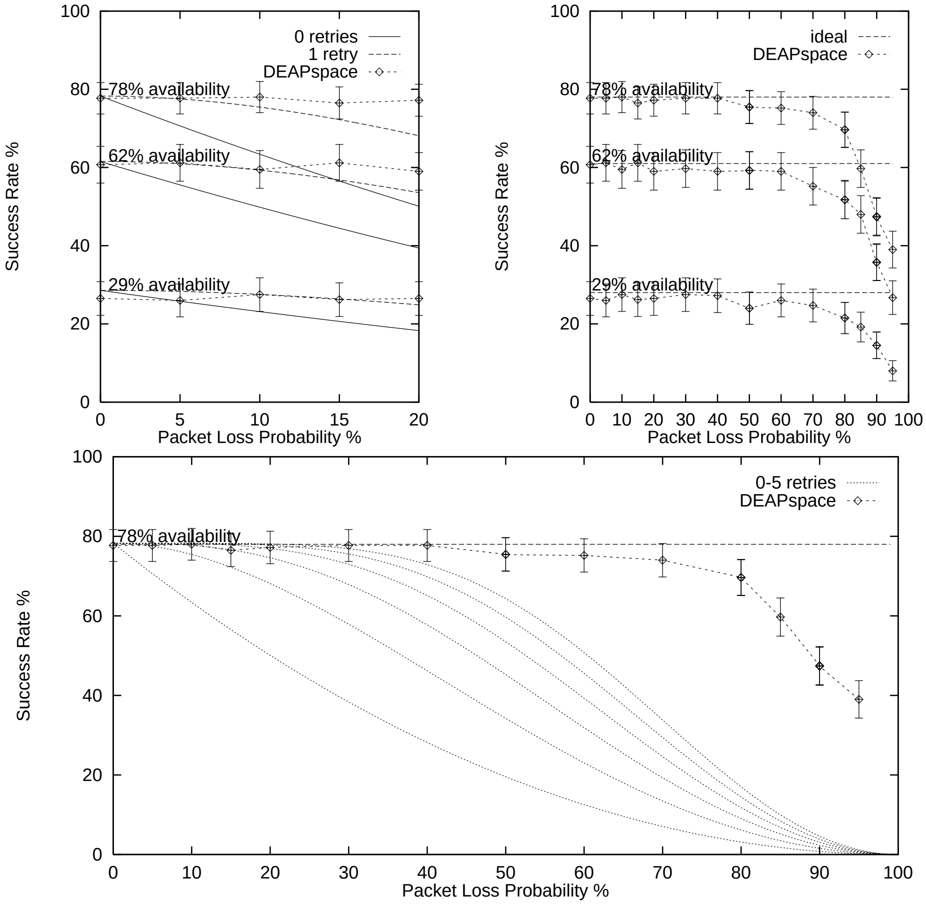

Figure 3: Comparing the DEAPspace algorithm with, on the **Upper Right**, theoretical results for the basic algorithm and, on the **Upper Left**, the ideal result. The **Lower** graph compares ideal (for zero through five retries) with DEAPspace results over a full range of packet loss conditions. Error bars are given for 95% confidence interval.

causing decisions to be based on potentially outdated information. Having addressed this problem, concentration can now be focused on the issue of communication overhead. The high success probability shown in Figure 3 for very high loss conditions comes at the cost of increased communication. As Figure 4 shows, the background communication required increases almost linearly with the packet loss probability. The consequences of this pattern depend largely on the size of individual service descriptions, so complete solutions must consider both tuning of the service discovery algorithm, and efficient representation of the data being distributed. Both of these challenges will be the subject of future papers from this lab.

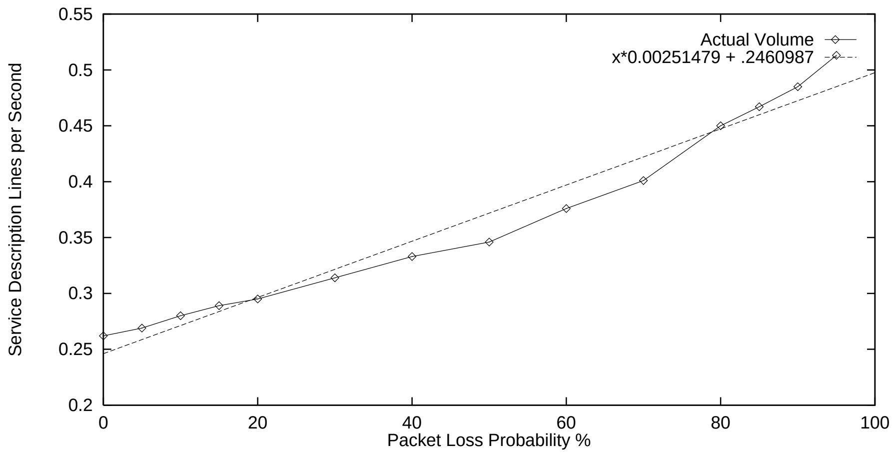

Figure 4: Average background traffic caused by the DEAPspace algorithm, measured in service descriptions per second (one description describes one service).

Further study of the discovery algorithm will also examine the effects of asymmetric assignment of default advertisement times, and of service timeouts. Using information from underlying network layers (feedback about the existence of CDMA synchronization, 802.11 advertisements, etc.) may also provide future optimizations, and will be considered.

References

[1] Accetta, M. Resource location protocol. Tech. Rep. RFC 887, Carnegie-Mellon University, December 1983.

[2] Apple Computer Inc. *Inside Macintosh: Networking*, 1994.

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

[4] Droms, R. Dynamic host configuration protocol. Tech. Rep. RFC 1541, Bucknell University, October 1993.

[5] Hild, S., Husemann, D., and Nidd, M. Service advertisements in wireless local networks. Tech. rep., International Business Machines Corporation, January 1999. patent 99101282.4-2209 pending, European patent office.

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

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

[8] PERKINS, C. IP mobility support. Tech. Rep. RFC 1939, IBM, October 1996.

[9] THE SALUTATION CONSORTIUM INC. *Salutation Architecture Specification*, 2.0b ed., October 1997.

[10] SUN MICROSYSTEMS INC. *Jini Architectural Overview*, 1999.