RZ 3254 (# 93300) 07/03/00
Computer Science/Mathematics 8 pages

Research Report

Reducing Power Use in DEAPspace Service Discovery

Michael Nidd

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Reducing Power Use in DEAPspace Service Discovery

Michael Nidd
IBM Research, Zurich Research Laboratory, 8803 Rüschlikon Switzerland
mni@zurich.ibm.com

Abstract
The DEAPspace service discovery algorithm provides an efficient, decentralised mechanism for service discovery in wireless ad-hoc, single-hop networks. The basic workings of this algorithm are reviewed, and modifications are presented that can reduce the power requirements for weaker devices by allowing some devices to run modified versions of this algorithm. The modifications are analysed in terms of their helpfulness to weak devices, and of their effect on unmodified devices in proximity with these special devices.

1. Introduction

As computers become smaller and more pervasive, the benefits of interaction between them multiply. In the DEAPspace group, we are studying coordination between proximate devices into so-called "smart environments." Because we envision this collaboration happening everywhere, not just in pre-determined locations, we have developed a decentralised service discovery algorithm that allows any set of devices to locate services withing a single hop of themselves.

Previous analyses of this algorithm have been made with uniform device configuration; in this paper, I will show that, with no significant loss to performance, the power requirements for small devices can be reduced through the use of asymmetric client configurations.

1.1. Background

The DEAPspace algorithm uses proactive single-hop broadcasts to share a world view with all proximate devices. Each individual device schedules a broadcast for some time in the future, waits until that time, then broadcasts its list of known services (*service elements*, or SEs) as a service advertisement message (SAM). These broadcasts are single-hop, and will not be forwarded beyond the local transmission range of the device in question. If, while waiting for its broadcast time, a SAM is received from elsewhere, the receiving device resets its broadcast timer to some new time in the future. By choosing times with some random jitter, the devices take turns sharing their world view. When a device receives a SAM that has its local information correct, and with a reasonable time to live (expiry time), it has no need to send its own SAM, because the important information has already been sent for it. If it sees that its local information is absent or about to expire, it can schedule its next SAM transmission sooner than usual, improving its chance of winning the broadcast race, and being known to others. When a device is scheduling SAM transmissions from this lower range, it is said to be in a *panic* state.

When a new device enters the transmission range of an existing group, it needs only to receive a single advertisement to learn about the entire set of services now available. Moreover, because the transmissions are scheduled reactively, the time for mutual discovery (the existing group knows about the new device, and vica verca) is better with DEAPspace than with regular beacons [1].

To avoid collisions, short-range radio MACs often schedule broadcasts, requiring some sort of extra work on the part of the device that wants to transmit. Because of this, sending broadcasts may require the transceiver to be active longer, resulting in more power consumption. Although the DEAPspace algorithm was designed to be completely decentralised, it would be useful to be able to use asymmetric allocation of behaviour parameters to shift some of the broadcast load to devices with better power availability.

The simplest way to achieve this shift is to allow power-rich devices to choose their normal (non-panic) timeouts from a range lower than the regular devices' normal range, but still higher than their panic range. In this way, the power-rich device will more often win the race to broadcast, but the normal devices will still be able to renew their services when they are about to expire, or when new devices arrive.

A better way to save energy is to deactivate the transceiver circuits. Because the DEAPspace algorithm has a small number of large messages, with a reasonably predictable arrival pattern, some devices in the group can be configured to hibernate their transceivers at the times when receiving a message is least likely.

Because theoretical analysis in the case of non-uniformly configured devices is very difficult, the effects of asymmetric parameter assignment have been explored through simulations. In the remainder of this paper, I will first introduce and justify the simulator, then review the goals and measures of quality for service discovery, then analyse the effects of these two power-saving methods on the performance of the algorithm.

2. Simulations

To test the effectiveness of the DEAPspace algorithm, I first implemented the device code in Java using a generic network interface. By connecting this code to an IP or Ethernet back end, I was able to check for any unexpected behaviour; then by connecting it to a network emulator that could introduce delays and losses, I was able to measure the internal behaviour of a group by running actual applications in real time.

This method was effective, but slow, so I then coded a special-purpose simulator (in C) for service discovery. The results from this simulator was compared to the behaviour observed with the emulator, and to the behaviour predicted by theoretical analysis.

2.1. Expected Behaviour

The expected behaviour was derived from three directions: a model of the number of normal rounds expected to pass between occurrences of a device entering a panic state, the observed total broadcasts per minute on an emulated network, and hand-derived values for total broadcasts per minute expected under specific conditions. This last case is explained separately [1], and the first two I will now describe.

Modelling the number of rounds expected to pass between panic states is a combinatorial problem. The challenge is to enumerate possible series of events, attach probabilities to them, and add them up. To help this analysis, it helps to count only the number of broadcast rounds, rather than actual time, and to ignore the possibility of collisions or losses. The behaviour of a single device in a group of n can be described by first writing down the generating function Φ for all series of rounds where the device sends a broadcast (x) and rounds where it receives a broadcast (y) that end in a series of $b+1$ consecutive receives. For now, x and y are just place holders, their exponents denote the repetition of their associated events in the state being described, and the coefficient is the frequency of that combination of occurrences (i.e. the number of valid permutations)[1].

$$\begin{aligned}\Phi &= y^{b+1} + [(1+y+y^2+y^3+\ldots+y^b)x]y^{b+1} \\ &\quad +[(1+y+y^2+y^3+\ldots+y^b)x]^2 y^{b+1} \\ &\quad \ldots + [(1+y+y^2+y^3+\ldots+y^b)x]^\infty y^{b+1} \end{aligned} \quad (1)$$

In this case, b is the number of rounds that a device can allow to pass without renewing its services, and without entering a panic state. Also, for future use, use z to count the number of rounds, regardless of who sent or received. In closed form, this gives Equation 2.

$$\Phi(x) = \frac{(yz)^{b+1}}{1 - \frac{(xz)(1-(yz)^{b+1})}{1-(yz)}} \quad (2)$$

[1] For an introduction to this notation, see [2].

In the series described by Equation 2, every term $N(x^u y^v z^w)$ represents N ways in which exactly u sends and v receives can result in w rounds passing before the first time the device in question has received $b+1$ times in a row, thus entering a panic state ($w = u+v$). By differentiating Φ w.r.t. z, then setting z to one, the terms will look like this: $Nw(x^u y^v)$. If x is the probability of sending on a particular round, and y is the probability of not sending (receiving), then the sum of these terms is the expected number of rounds between panic states. Performing this differentiation, and substituting $y = 1-x$ gives Equation 3.

$$E(x) = \frac{\partial}{\partial z}\Phi|_{z=1,y=1-x} = \frac{1}{x(1-x)^{b+1}} - \frac{1}{x} \quad (3)$$

Unfortunately, when a device enters a panic state, 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 in a panic state are guaranteed (by our current definition) to lose the next race. If the value predicted by Equation 3 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 3 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 a particular round. Therefore, the probability of a particular device that is not in a panic state having a fair race to be the next sender is $(1-E(x)^{-1})^{n-1}$, and the probability of it winning that race is n^{-1}, giving Equation 4.

$$x = \left(\frac{1}{n}\right)\left(1 - \frac{1}{E(x)}\right)^{n-1} \quad (4)$$

Solving Equation 4 for various values of n and b, then comparing these results to simulations shows strong agreement up to the point where n approaches b, as shown in Figure 1. If n and b have similar values, it is increasingly likely that multiple devices will be simultaneously in a panic state. This makes the probability of being already in a panic state during any given round significant, compared to the probability of entering a panic state during that round, and invalidates the

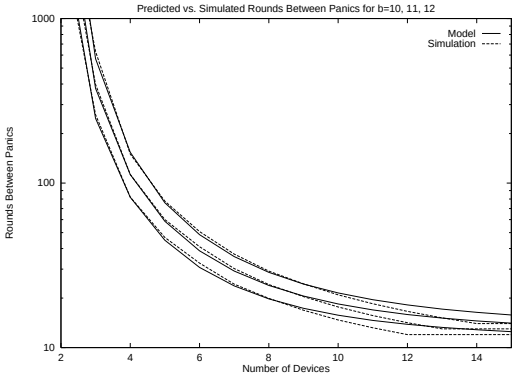


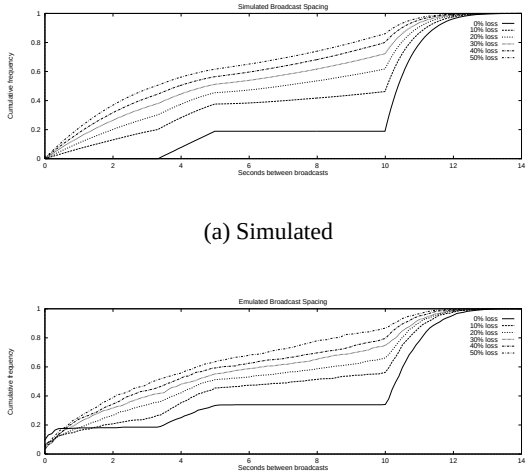
Figure 1: Comparing the results of Equation 4 to simulations for three values of b, and various numbers of devices

assumptions used to derive Equation 4, so the lack of agreement in this region is not surprising.

The fact that the behaviour of each device is so dependent on the behaviour of others puts a great many restrictions on theoretical models of this kind. A state transition model would also be possible, but the number of states necessary to model even small scenarios is so large that analysis would be almost indistinguishable from simulation anyway. So, for simplicity, and also for an easier transition to counting time instead of rounds, I have made all further analysis of the system using simulation and implementation.

For the actual implementation, each device is coded as a Java object that connects to an underlying network interface. The network interface has been created to be generic, allowing the devices to be tested over real and emulated underlying network layers. The service discovery module executes in its own thread. A pseudocode overview of this module is shown in Figure 2, using X to represent the range of time intervals from which the broadcast timer is usually reset, and X' to represent the range that is used during a panic state.

Using a network emulator, this implementation can be used to give the actual distribution of round durations. These results compare favourably with the results of simulation, as shown in Figure 3. Right now, some discrepancy can be seen in the counts of very short rounds, especially for very low packet loss rates. This is because the emulator uses propagation delay, while the simulator does not. The result of propagation delay is sometimes having two winners of the



(a) Simulated

(b) Emulated

Figure 3: Cumulative distribution for round durations with six normal devices, having $X = [10, 15]$ and $X' = [3.33, 5]$

race to broadcast, since the second to actually send can queue its SAM before receiving the SAM from the actual winner. Because our model environment gives responsibility for collision avoidance to the lower layers, this discrepancy will be resolved by modifying the emulator in time for the final version of this paper. The results presented here are completely consistent with expectations, and lead me to be confident that the modified emulation will agree very closely with simulations.

3. Modifications for Special Devices

In general, modifications have tradeoffs; in this section, I will examine two power-saving modifications for the DEAPspace algorithm, and study the tradeoffs that they offer, in terms of several important criteria:

Total Transmissions: The total number of broadcasts offered to the network by all devices. This value represents the network load incurred by the algorithm under whatever circumstances are being examined.

Fraction Expired: The fraction of time during which the SE for some device exists unrenewed in some

```
advertise(LOCAL) {
    time tout ←getTimeout(X)
    loop(forever) {
        REMOTE ←read(tout)
        if(timed out) {
            foreach l ∈ LOCAL
                if(l is my service)
                    l.expiry ←NormalExpiry
            broadcast(LOCAL)
            tout ←getTimeout(X)
        } else {
            Interval I ←update(LOCAL,REMOTE)
            tout ←getTimeout(I)
        }
    }
}

read(t) {
    blocking read from network with timeout t
}
```

```
Interval update(LOCAL,REMOTE) {
    foreach r ∈ REMOTE {
        if(r is not my service) {
            if(∃_{l∈LOCAL} r.id = l.id) {
                if(r.expiry > l.expiry)
                    l.expiry ←r.expiry
            } else {
                insert r into LOCAL
            }
        }
    }
    if(∃_{s∈my services} s ∉REMOTE) return X'
    foreach r ∈ REMOTE {
        if(r is my service and r.expiry < minExpiry)
            return X'
    } else return X
}
getTimeout(I) {
    pick random value on interval I
}
```

Figure 2: Sample pseudocode implementation of the DEAPspace algorithm

other device's list for longer than its default time to live value.

Time to Mutual Discovery: The number of seconds from the moment when communication becomes possible between a new device and an existing group until all devices know the services offered by all other devices.

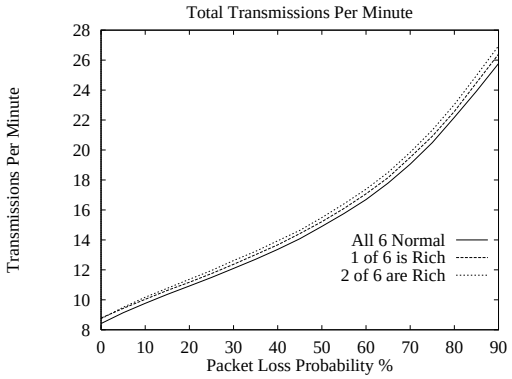
3.1. Rich Devices

If a target platform has the property that transmitting a broadcast, including all associated protocol overhead, is more expensive than reading one, then it is enough to shift the burden away from normal devices, and onto power-rich devices. This can be done by simply arranging to have rich devices usually choose broadcast times smaller than the times usually chosen by other devices. Here, I will leave the distribution unchanged, but lower the range from which a timeout is 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.

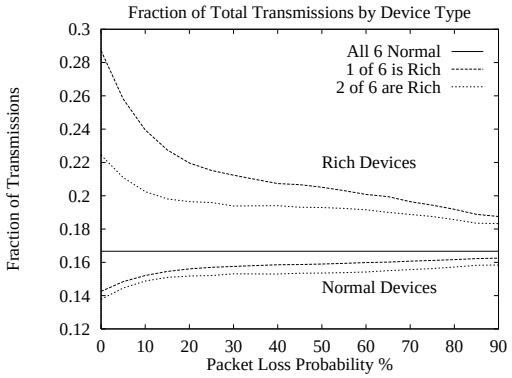
Figure 4 shows that while this scheme will slightly increase the overall network load in the vicinity of one of these rich devices, it will reduce number of broadcasts sent by normal devices in that same area. In this example, the broadcast times for normal devices are taken from the range $X = [12, 15]$ 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 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 5, 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 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

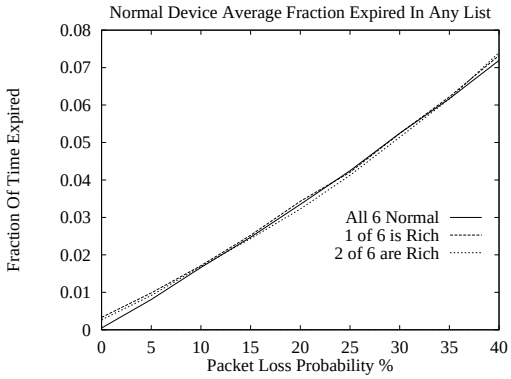


(a) Total Transmissions

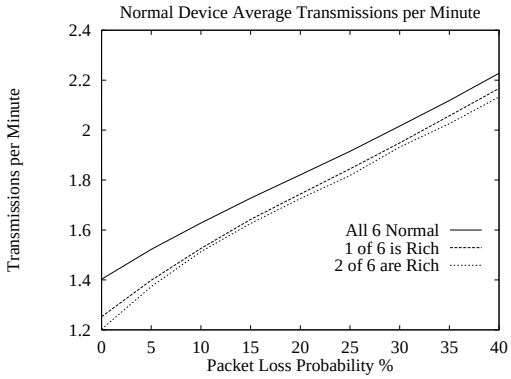


(b) Fraction by Device Class

Figure 4: 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



(a) Expired Frequency



(b) Transmission Frequency

Figure 5: 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

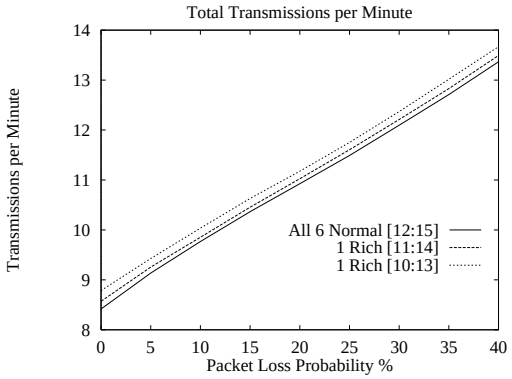
the number of broadcasts made by the normal devices is increased.

What we learn from this is that by asymmetrically assigning timing parameters, we can shift a disproportionate fraction of the broadcast responsibility to devices of our choosing. This is an interesting property, and is helpful in situations where broadcast transmissions are significantly more expensive than receptions. Having more general application, however, is actually to power-down the transceiver circuitry. Identifying good times for weak devices to stop listening is useful on all platforms, including those low power radios for which the signal processing circuitry uses more power than the actual transmission amplifier.

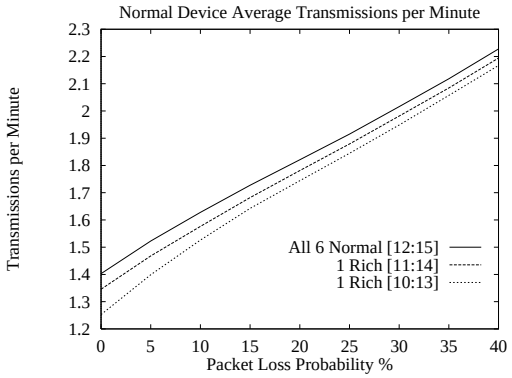
3.2. Weak Devices

Because sharing world views has the effect of greatly reducing the total broadcast frequency, compared with having all the devices advertise their own services (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 with the normal behaviour of other devices.

The specific technique that I have used involves pe-



(a) Total Network



(b) Average Normal Device

Figure 6: 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

riodic idles with duration equal to the minimum broadcast delay (in this case, 4 seconds). This will be done every time a broadcast is received in which the weak device's own services are all not near expiry, or when a broadcast is transmitted, excepting when it is transmitted as a result of seeing a service about to expire. 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.

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

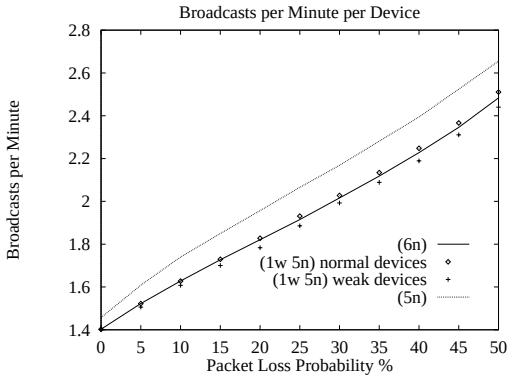
```
advertise(LOCAL) {
    time tout ←getTimeout(X)
    loop(forever) {
        REMOTE ←read(tout)
        if(timed out) {
            foreach l ∈ LOCAL
                if(l is my service)
                    l.expiry ←NormalExpiry
            broadcast(LOCAL)
            tout ←getTimeout(X)
        } else {
            Interval I ←update(LOCAL,REMOTE)
            tout ←getTimeout(I)
        }
        if(tout > X'_max) {
            tout −= X'_min
            hibernate(X'_min)
        }
    }
}
```

Figure 7: Allowing weak devices to hibernate during service discovery

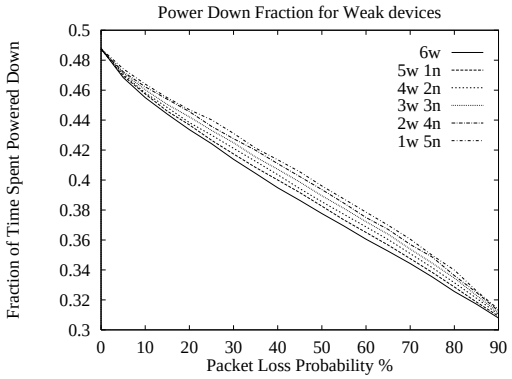
is less than one in two, while the total network load is not significantly different[2] from the load caused by six normal devices, as can be seen in Figure 8. As the packet loss gets worse, the device will automatically hibernate less frequently, allowing it to take advantage of whatever packets do arrive correctly. With our test parameters as before, all 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 is its impact on the timeliness of discovery. With reasonably sized groups, like the six-member groups used above, there is very little effect. 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 de-

[2]With the same sample configuration parameters used in Section 3.1, $X = [12, 15], X' = [4, 5]$.



(a) Transmission Frequency



(b) Hibernation Time

Figure 8: Behaviour of a group of six devices, in which one through five of them use the weak device modification described in Figure 7, having $X = [12, 15]$ and $X' = [4, 5]$

vice 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 9). This difference is from the number of cases when the existing group is the first to transmit, and the weak device is idle at the time. The fraction of time spent idle by a lone device will be about $4/13.5$, and the group will transmit first about half the time, so one would expect this to happen about 15% of the time, and cause discovery to be delayed until the new device transmits, about another 11.5 seconds. The various panic timeouts chosen by 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

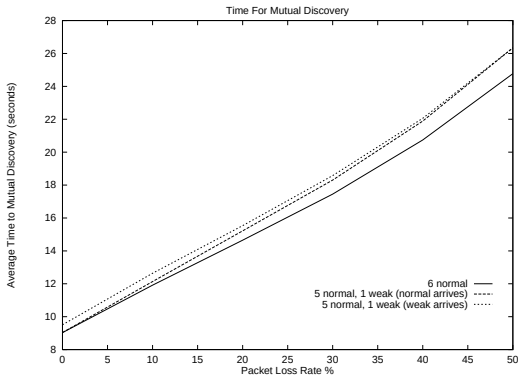


Figure 9: Time to mutual discovery when a new device enters an existing group of five devices

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 [1], meaning that this problem will never occur at all.

The most 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. This broadcast will be during an awake time 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 you can see from Figure 10, this can mean a difference of two or three seconds when the packet loss is bad, but means that the lone devices were achieving this rate while hibernating a third of the time. In many applications, this tradeoff is worth while.

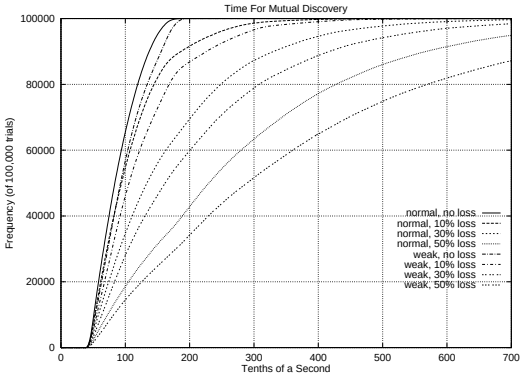


Figure 10: Comparing the time to mutual discovery for two weak devices versus two normal devices.

4. Conclusions

In this paper, I have presented some analysis techniques for a new service discovery algorithm, and used those techniques to evaluate ways to save power-consumption through asymmetric assignment of configuration parameters to the various devices. One technique was able to shift the broadcast responsibility to devices that choose to identify themselves as power-rich, and another was able to intelligently offer hibernation windows to devices that identify themselves as power-weak. Both of these techniques can be used to conserve power for weak devices, without causing significant interference with the unmodified devices.

References

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

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