# Perfect Difference Sets for Neighbor Discovery: Energy Efficient and Fair

Jó Ágila Bitsch Link, Christoph Wollgarten, Stefan Schupp, Klaus Wehrle
Communication and Distributed Systems (Informatik 4), RWTH Aachen University
Aachen, Germany
{jo.bitsch,christoph.wollgarten,stefan.schupp,klaus.wehrle}@rwth-aachen.de

## ABSTRACT

We present an energy efficient neighbor discovery framework that enables Linux and TinyOS based systems to discover and connect to neighbors via IEEE 802.11 and IEEE 802.15.4, which are only available sporadically. Using quorum schemes, we schedule on and off times of the wireless transmitters, to guarantee mutual discovery with minimum power given a specific latency requirement. Neighbor discovery is fundamental to intermittently connected networks, such as disruption and delay tolerant networks and optimizing it, can lead to significant overall energy savings.

Using perfect difference sets, our results indicate that we reduce the latency by up to 10 times at a duty cycle of 2% compared to the state of the art. We further define and characterize our neighbor discovery scheme with respect to fairness for asymmetric energy scenarios. Using these results, we allow energy-harvesting applications to adjust neighbor discovery based on their current energy requirements as a well defined trade-off.

## Categories and Subject Descriptors

C.2.1 [**Computer-Communications Networks**]: Network Architecture and Design—*Wireless Communication*

## General Terms

Algorithms, Design, Measurement

## Keywords

Neighbor Discovery, Sporadic Connectivity, Wireless Networks

## 1. INTRODUCTION

Detecting the presence of communication partners in communication range is fundamental to intermittently connected networks, such as disruption and delay tolerant networks. In real world deployments, applications spend the majority of their operating time looking for neighbors. Consider the case of DTN test in Laponia [4], where over the course of 60 days, between 27 nodes, 9575 contacts were recorded, averaging to one contact every 4 h. While contacts are not distributed evenly, this already hints at the time spent idly looking for communication partners. Similarly, in sensor networks used for the observation zebras in the wild [13], sensor nodes spend the majority of the time without connectivity. Thus, reducing the energy needed during this time leads to significant energy savings.

In this paper, we show the feasibility of perfect difference sets. This scheme results in a provably optimal schedule for neighbor discovery for two communication partners that are using the same duty cycle. Then, we consider the case, that two nodes with different energy levels are using different duty cycles and generalize our approach to the asymmetric case.

To the best of our knowledge, there is no general framework for energy efficient neighbor discovery yet for wireless sensor networks and Linux devices. Therefore, we present a TinyOS framework, as well as a Linux kernel module and implementation for use in applications in scenarios facing sporadic communication.

As an example application, we present an energy efficient podcast sharing application. In this context we also discuss, how our system deals with neighbors which do not have new information available and can be ignored.

### 1.1 Contributions

The main contributions of this paper are:

1. **Energy efficient neighbor discovery based on PDS:** We utilize Perfect Difference Sets to discover the presence of a neighbor at the provably smallest energy cost per given latency.
2. **Characterization of asymmetric neighbor discovery using PDS:** As different nodes are subject to different energy restrictions, we characterize the behavior of PDS based neighbor discovery when the nodes use different duty cycles.
3. **Integration of neighbor discovery into Linux and TinyOS:** We implemented an easy to use framework for Linux as well as for TinyOS to make use of neighbor discovery schemes.
4. **Energy efficient podcast sharing:** To show the feasibility of the approach for real applications, we built an energy efficient podcast sharing application on top of our Linux framework.

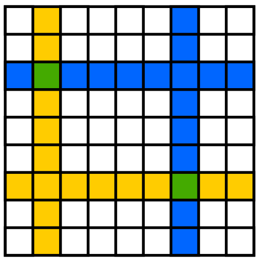

**Figure 1: The grid quorum scheme works on a rectangular array. Each node selects an arbitrary column and row in which it turns on its radio. Thus we can guarantee at least two intersections per period.**

## 2. RELATED WORK

We can classify related work into two categories: (1) Neighbor discovery schemes and (2) opportunistic file sharing approaches. While different approaches for neighbor discovery exist, they typically don't consider the asymmetric, or lead to suboptimal performance. Opportunistic file sharing applications are a good initial use case, as they provide a useful service to a user but do not depend on complicated routing schemes.

### 2.1 Neighbor discovery schemes

Neighbor discovery can be modeled as a quorum problem. For this, we divide time into time slots and select slots in such a way, that we can guarantee mutual discovery in certain number of slots. Using quorums in computer science is not a new idea. Researchers typically use them for mutual exclusion problems, see [9, 14], but also for network topologies and other fields.

Quorums are a set theoretical construct. The construction of a quorum system can guarantee the intersection between subsets is not empty. An example of a very simple construction bases on a *Grid* array. Maekawa [14] originally designed this scheme for mutual exclusion. The core idea is to distribute the time slots of the desired period on a rectangle. The quorum is generated by selection of an arbitrary column and row in this rectangle. Following this construction, no matter which column and row we choose, nodes are guaranteed to have at least two intersections with any other node in the network, see Figure 1.

Distributing the slots on a *Torus* instead of a square, Lang and Mao presented a new construction [12]. Instead of choosing a row and a column, we choose a column and a diagonal branch of half the length of the perimeter of the torus. Making use of the wrap-around of the torus, we can guarantee that for two nodes at least one branch intersects with the column of the other node. Instead of two intersections, we now have only one in most, though not all cases. *E-Torus* [8] is an extension to this concept, adding additional branches to create an arbitrary number of guaranteed intersections.

In 2008, Dutta and Culler presented *Disco* [3]. Each node chooses two primes. The node then turns on its transceiver during time slots which are multiples of these primes. With the help of the Chinese remainder theorem we can guarantee a discovery.

Then in 2010, Khandalu presented *U-Connect* [11]. Each node chooses a single prime number $p$. Nodes are active at each multiple of that prime. When two nodes choose the same prime, an intersection cannot be guaranteed. Therefore each node performs a hyper-cycle at the square of the chosen prime. This means it sends out beacons for $\lceil \frac{p}{2} \rceil$ slots. Following the Chinese Remainder Theorem we can guarantee discovery.

As an example of a probabilistic approach, McGlynn introduced the *Birthday* protocol [15]. A node randomly chooses if it turns on its radio or not in each slot using weighted probabilities. Making use of the birthday paradox we can give probabilistic guarantees, that a node meets a neighbor with very high probability.

In comparison Bluetooth device discovery [1] and WiFi IBSS discovery [7] do not make use of any scheduled radio off-times. Thereby, energy-efficiency is comparatively small.

### 2.2 Applying neighbor discovery

*BlueTorrent* [10] and *PodNet* [6] are cooperative P2P file sharing applications based on Bluetooth and Ad-hoc-Networking for neighbor discovery and data exchange. Very similar to our opportunistic podcast sharing application, they make use of BitTorrent like approaches to maximize the delivery probability.

The authors of BlueTorrent try to find optimal settings for the initial bluetooth inquiry and the connection establishment, but the approach still suffers from the energy hunger of the Bluetooth device discovery. Similarly PodNet treats device discovery rather simplistic but rather focuses on the application.

## 3. SYMMETRIC NEIGHBOR DISCOVERY

In this section, we assume that each node has the same energy requirements and characteristics. Each node will therefore employ the same schedule with respect to activity of its transceiver. We will begin by formalizing the problem in a theoretic formulation. Then, we show how Perfect Difference Sets provide a provably optimal solution for symmetric neighbor discovery.

### 3.1 Theoretic Formulation

We assume time is divided into time slots. Furthermore, we assume discovery is periodical, we therefore have a set $S$ of $|S|$ slots. A quorum to be used for neighbor discovery has to fulfill the following four properties: (1) Non-empty intersection, (2) non-triviality, (3) equal size, and (4) rotation closure.

Each subset of $S$ in the quorum system needs to have a non-empty intersection with each other subset. If there was an empty intersection, we could not guarantee discovery.

$$\forall S_i, S_j \in S, i \neq j : S_i \cap S_j \neq \emptyset$$

No subset of $S$ can be a superset of the other. The only quorum system with one subset being a superset of another would be the trivial quorum system, with the subset being identical to $S$.

$$\forall S_i, S_j \in S, i \neq j : S_i \not\subset S_j$$

As each node is assumed to have the same energy requirement, we require each used subset to be of the same size.

$$\forall S_i, S_j \in S, i \neq j : |S_i| = |S_j|$$

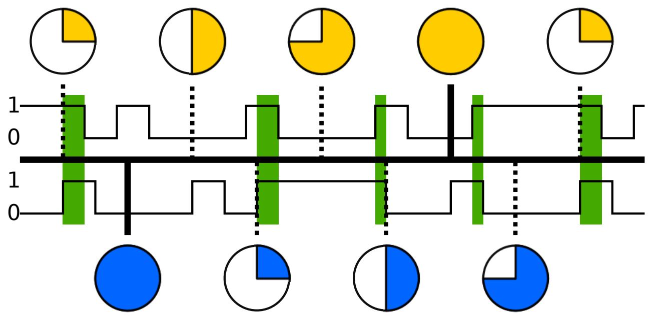

**Figure 2: Rotation closure ensures that no matter by how far time shifts between two nodes, an intersection in the active slots still can be guaranteed. Here two nodes operate with a time shift slightly larger than a quarter of the period length and still obtain the desired intersection. Both use the *Grid* algorithm.**

Finally, we require rotation closure, see Figure 2. For any $S_i, S_j, i \neq j$ with $\phi_{i,j}$ being the phase offset between $i$ and $j$ and $\Psi(i,t)$ being the neighbor discovery schedule of node $i$ at time $t$ with $\Psi(i,t) = 1$ if $i$ is active at time $t$:

$$\forall t \exists \phi_{i,j} : \Psi(i,t) = \Psi(j, t + \phi_{i,j}) = 1$$

This last requirement has two consequences: (1) Each node has the same schedule. There is no node specific initialization. (2) Nodes are not required to be synchronized at all. An arbitrary offset leads to a discovery.

## 3.2 Perfect Difference Sets

Maekawa [14] proved the lower bound for the quorum size for mutual exclusion and showed that finite projective planes fulfill this property. Taking the requirement of equal size, he derives that a quorum for each element of $S$ exists. He then defines $D$, the number of quorums in which an element of $S$ is represented in. As every element is represented in its own quorum, it must be contained in $D-1$ quorums.

Following this, for $K$ being the desired size of the quorum system, the number of distinct members in $S$ must equal:

$$N = (D-1) \cdot K + 1$$

As the number of elements in $S$ must be equal to the union of the elements of all quorums, it follows that $K = D$.

$$N = \frac{K \cdot N}{D} \Rightarrow K = D$$

Therefore, we obtain:

$$N = (K-1) \cdot K + 1$$

With $K = p^n + 1$, with $p^n$ being a power of a prime, this corresponds to the number of vertices and edges in a finite projective plane, for which non-empty intersection also holds. Following [19], we can therefore reduce the problem of finding the required quorum with finding the appropriate finite projective plane. Thus, the order of the size of the quorums with respect to the size of $S$, $N = |S|$ is:

**Table 1: We show the power latency product for different algorithms and how long a period takes for a given duty cycle. Keep in mind that $p, p_i, p_j \sim \sqrt{N}$.**

| Algorithm | power–latency–product | 2% | 5% | 10% | 20% |
|---|---|---|---|---|---|
| Disco | $\frac{N}{p_i} + \frac{N}{p_j}$ | 27719 | 3473 | 1199 | 119 |
| E-Torus | $\sqrt{N} + \lceil \frac{\sqrt{N}}{2} \rceil \cdot B$ | 15625 | 2500 | 900 | 100 |
| Grid | $2 \cdot \sqrt{N} - 1$ | 9801 | 1521 | 361 | 81 |
| PDS | $\sqrt{N}$ | 2451 | 381 | 91 | 31 |
| Torus | $\sqrt{N} + \lceil \frac{\sqrt{N}}{2} \rceil$ | 5625 | 900 | 225 | 49 |
| U-Connect | $p + \lceil \frac{p}{2} \rceil$ | 5366 | 856 | 176 | 51 |

$$\begin{aligned} & & N &= p^{2n} + p^n + 1 \\ & & &= p^n(p^n + 1) + 1 \\ \Rightarrow & & (p^n+1)^2 - p^n &< (p^n+1)^2 \\ \Rightarrow & & &\text{size of quorums } \mathcal{O}(\sqrt{N}) \end{aligned}$$

Thus the size of the resulting quorums meets the lower bound $\mathcal{O}(\sqrt{N})$.

For the actual construction of a discovery schedule, we use the result of Singer [17, 2]. He constructs *perfect difference sets* (*PDS*) from finite projective planes. These are numberings of the nodes in finite projective planes, that allow us to create all elements 0 to $N-1$ as the difference between two elements of this set.

While perfect difference sets originally have a different interpretation, they perfectly fit the quorum properties needed for neighbor discovery. For a $p^n$, the power of a prime, we can create quorums of size $p^n + 1$ to cover $p^{2n} + p^n + 1$ slots.

As we can create all differences between 0 to $N-1$ for $N = p^{2n} + p^n + 1$, it follows directly that we have rotation closure. All quorums have the same size, as they are the same schedule rotated. The quorums are not trivial and there is an intersection between all quorums. Therefore, the slot schedule is exactly the elements from the perfect difference sets.

To give an example, consider $p^n = 2^1 = 2$. This results in the perfect difference set $G = \{0; 1; 3\}$ for covering schedules of length $p^{2n} + p^n + 1 = 7$, which is the period where the schedule repeats. Therefore, the complete schedule is $\Psi_i = \{1; 1; 0; 1; 0; 0; 0\}$. This schedule has a duty cycle of $\sim 43\,\%$. In Table 1, we show the power latency product for the different neighbor discovery schemes and associated period lengths for a given duty cycle.

## 4. ASYMMETRIC NEIGHBOR DISCOVERY

In the context of perpetually running networks, network nodes have widely differing energy budgets. Consider TurtleNet [18], where researchers equipped turtles with sensor nodes. Those nodes also include solar panels to harvest energy. As a subset of turtles are exposed to the sun more, there energy budget is higher. Other nodes will have to use a more restrictive duty cycle. Duty cycles are asymmetric.

In this section we discuss fairness in this asymmetric case. We then characterize how *PDS* deals with this situation and how *Grid* as a very simple approach handles asymmetry.

## 4.1 Fairness in Neighbor Discovery

Consider the following scenario: Node $A$ and $C$ harvest 10 times more energy than node $B$. The latency of a neighbor discovery protocol will depend on the node $C$, the least active one. Since nodes $A$ and $C$ can spend more energy on neighbor discovery, they should not be penalized with a high latency but rather benefit by having a smaller latency. We capture this intuitive notion in a more formal way by defining the *Energy Gap*.

Given two nodes $A$ and $B$, that can harvest energy $e_A$ and $e_B$ respectively. Let $e_A, e_B > 0$ and $e_A \geq e_B$ . The *Energy Gap* between nodes $A$ and $B$ is defined to be the ratio:

$$\alpha_{A,B} = \frac{e_B}{e_A}$$

Given the above definition, we can now formalize our *Fairness* criteria. If the energy gap $\alpha_{A,B}$ between node $A$ and $B$ is closer to 1, the average latency $\bar{l}_{A,B}$ must be smaller. So, an algorithm is fair, if the following holds:

$$\forall B, C | \alpha_{A,B} \geq \alpha_{A,C} : \bar{l}_{A,B} \leq \bar{l}_{A,C}$$

## 4.2 Grid

For simplicity, we consider the *Grid* neighbor discovery scheme on a $n \times n$ grid. The algorithm chooses one row and one column to be active in. Therefore, there are $2n - 1$ active slots out of a total of $n^2$ slots. Thus the duty cycle is $\frac{2n-1}{n^2} \sim \frac{2}{n}$. Accordingly, a more energy constrained node chooses a greater $n$.

If two nodes $N$ and $M$ choose values $n$ and $m$, with $n \leq m$, the energy gap is $\alpha_{N,M} \sim \frac{2}{m} / \frac{2}{n} = \frac{n}{m}$. Thus, we must show, that for two values $m_1$ and $m_2$, with $m_1 \leq m_2$, the average latency $\bar{l}_{N,M_1}$ is smaller than the average latency $\bar{l}_{N,M_2}$. In the following, we will see that this is possible.

Looking at the actual schedule of the *Grid* scheme, we see that there are two types of active periods. A long continuous period of length $n$ for the chosen row and $n - 1$ single slots each separated by $n - 1$ inactive slots for the chosen column, see Figure 1. For two nodes $N$ and $M$, with two different parameters $n$ and $m$ with $n \leq m$ we can guarantee, that one of the single slots of node $N$ will always intersect with the long slot of node $M$, because the long slot length is larger than the inter-slot spacing of the short slots by construction.

If we assume an equal distribution of offsets, we are on average $\frac{m \cdot m}{2}$ away from said intersection, the worst case being $m^2$. This average case (as well as the worst case) only depend on the more energy constrained node $M$. From this follows that *Grid Neighbor Discovery* is indeed fair for asymmetric neighbor discovery, as defined above.

## 4.3 Perfect Difference Sets

Unfortunately, the internal structure of a perfect difference set does not exhibit any usable regularity, we can use to guarantee average discovery latency. Indeed, for specific rare offset and schedule combinations, where the schedule lengths are not coprime, there exist wrap arounds, where no intersection can be found. Examples of this are schedules for the combination of schedule lengths $\{91; 273\}$ and $\{651; 4557\}$. We did an exhaustive search, and those where only combinations where nodes have a duty cycle $\geq 1\,\%$. In practice, however, Perfect difference sets behave fair, as we will discuss in Section 6.3.

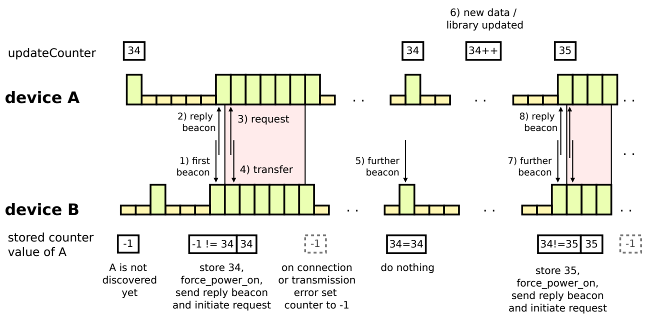
6) new data / library updated
updateCounter
34
34
34++
35
device A
2) reply beacon
3) request
8) reply beacon
1) first beacon
4) transfer
5) further beacon
7) further beacon
device B
stored counter value of A
-1
-1 != 34
34
-1
34=34
34!=35
35
-1
A is not discovered yet
store 34, force_power_on, send reply beacon and initiate request
on connection or transmission error set counter to -1
do nothing
store 35, force_power_on, send reply beacon and initiate request

**Figure 3: When a node is discovered, an initial exchange takes place. From this point on, the node stores an update counter. The update counter changes as the system state of a node changes, e.g. when a new podcast becomes available. Only when this update counter changes, a neighboring node will connect to the node.**

## 5. IMPLEMENTATION

We implemented the different neighbor discovery schemes for three different platforms. Initially, we created (1) a Python simulation. Driven by sensor network applications, (2) we implemented TinyOS applications, which we used also for evaluation in the Motelab [5] testbed. Finally, we also implemented the different neighbor discovery schemes for (3) the current Linux kernel and build a energy efficient podcast sharing application. In the following, we discuss our practical considerations and abstractions.

## 5.1 Simulations

Although, our initial use case was in sensor networks, we decided to first implement a very abstract Python simulation. This allowed us to better investigate different ideas with respect to perfect difference sets, as well as fairness. Not having to consider environmental factors, such as link loss and radio mode switching times, we are able to perform exhaustive searches, as in Section 6.3, and gather more statistics.

## 5.2 TinyOS

One of the most common sensor network operating systems is TinyOS [16]. It is organized in a very modular way, and has a variety of backends. Using the common `ActiveMessageC` abstraction, we built a Neighbor Discovery Framework that instantly portable to all sensor network platforms, TinyOS supports. We organized the structure of our modules in such a way, that the neighbor discovery algorithm is exchangeable very easily. For our evaluation, we then also implemented the algorithms presented in the related work. [1]

## 5.3 WiFi

We make use of generic callbacks in the Linux Wireless mac80211 kernel module to the wireless hardware device's driver. These callbacks, named IEEE80211 operations, allow us to stay independent of the different specific hardware implementations. We implement one additional method into the `mac80211` module to control the *start* and *stop* callback

[1]We plan to make the source code available after publication.

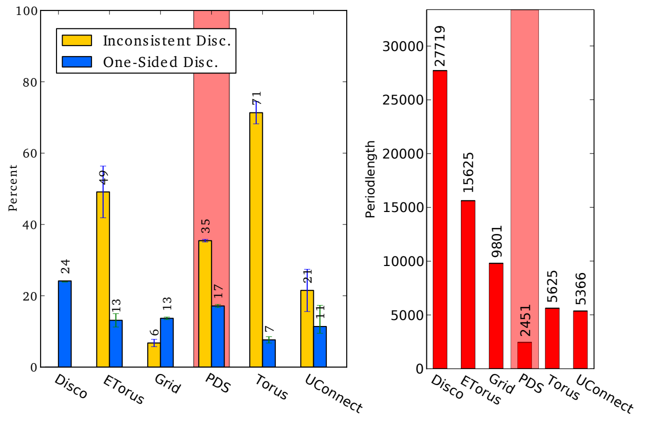

**Figure 4: Results of tests on Motelab with an applied duty cycle of 2% to every algorithm. Inconsistent and one-sided discovery errors (left). Period lengths in number of slots (right).**

functions and export it to kernel space. In addition, we build a new kernel module, `LiND_pwr`, which uses the exported method and provides a character device to user space. Thus, applications can control the power of the WiFi device's transmitter by writing to `/dev/LiND_pwr`. Our design allows an easy exchange of the neighbor discovery application, which implements the specific neighbor discovery algorithm. Beacons are injected and captured by setting the operation state of an additional virtual network interface to monitor-mode. Our beacons include an integer value, called "updateCounter". This value serves as indicator in order to determine, if it makes sense to contact a previously discovered node again. A request is initiated by sending a reply beacon, which would inform the intended receiver to stay continuously active during the exchange of additional data over TCP/IP, see Figure 3. Data is forwarded BitTorrent-like. This supports the continuation of previously interrupted transmission via different nodes.

## 6. EXPERIMENTAL RESULTS

Aside from initial experiments and the exhaustive search for asymmetric discovery latency for PDS, all experiments were performed on real nodes. The sensor network experiments were conducted using Motelab [5]. We used a fixed slot length of 100 ms for the Motelab and for the WiFi tests.

### 6.1 Motelab

The tests were run for half an hour for each algorithm, as this is the longest continuous, for which we could continuously access the Motelab testbed. This is a restriction of Motelab on which we had no influence. The analyzed duty cycles were 2%, 5%, 10% and 20%.

We used two metrics to measure the quality of an algorithm: (1) *Inconsistent Discoveries* and (2) *One-Sided Discoveries*. The former is, when for a fixed nodes, another node which is known to be in radio range is not detected in a discovery period, so, how often a discovery fails. The latter is, when one node discovers the presence of the other, but not vice-versa, therefore, the asymmetry in the discovery.

As seen in Figure 4, for a duty cycle of 2% only PDS has a sufficiently short period length of 245.1 s $\sim$ 4 min, also

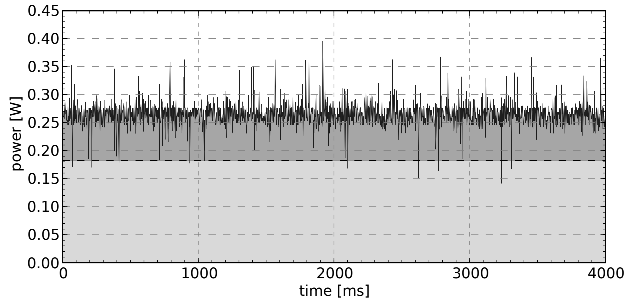

**Figure 5: This diagram shows the overall power consumption of the wireless device running in ad hoc mode.**

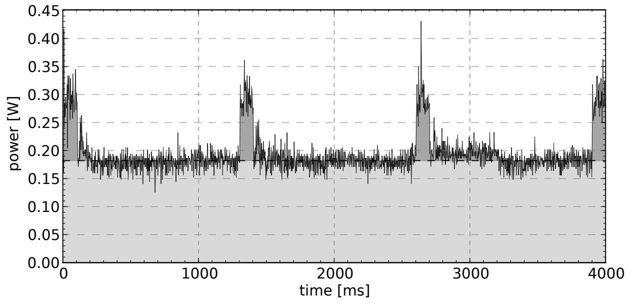

**Figure 6: This diagram shows the power consumption of the ad hoc mode in combination with `LiND_pwr` using the *U-Connect* neighbor discovery algorithm.**

compare Table 1. In general, for algorithms that produce long periods of consecutive active slots, such as *Grid*, the above-mentioned quality metrics are better. This coincides with the requirement of less radio mode switching, which might lead to better results for higher duty cycles. Keep in mind, there is no coordination necessary between the nodes in any way, see Section 3.1.

### 6.2 WiFi Tests

Switching a WiFi device's radio transmitter *off* or *on* takes a specific period of time and consumes additional power, which varies between different hardware models and vendors. To toggle our test device, a TL-WN422G USB adapter, from *off* to *on* or vice-versa, constantly takes 12 ms. During one complete switch, from *off* to *on* and back to *off* again, the device is additionally powered 14 ms of the overall period of 24 ms. However, using a neighbor discovery scheme like *U-Connect* with a maximum discovery latency set to 10 s, we achieved discovery within an average latency time of 3.2 s, while the WiFi transmitter is powered only 17% of the time. The average energy consumption of our wireless device during an *on* state is 90 mW higher than in an *off* state. Visualizations of our power measurements, see Figure 5 and 6, recorded with an oscilloscope (Tektronix TDS 2024B), illustrate the significant energy savings of the application of our `LiND_pwr` kernel module compared to continuous *Ad-hoc mode.*

### 6.3 Asymmetric Discovery

Figure 7 shows the comparison of a perfect different set of length 553 to larger perfect difference sets. In general, we

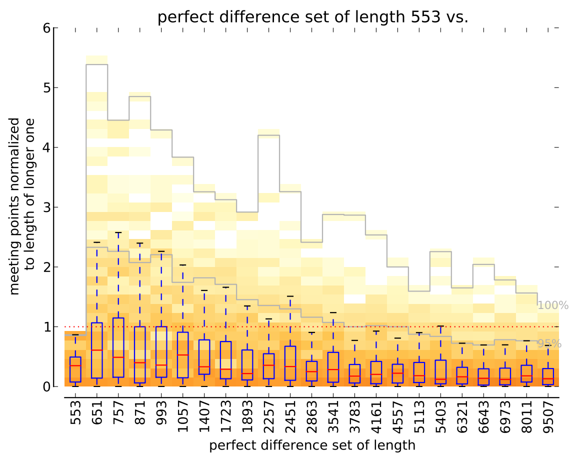

**Figure 7: We compare the PDS of length 553 to larger PDSs. While we normalize the y-axis to the length of the longer PDS, we notice in the boxplots, that even though the worst case performance, see the 100 percentile line, is not favorable, the big majority of discoveries happens within the length of the longer PDS, also consider the 95 percentile line. They actually happen faster than with *Grid*, especially as we consider longer PDSs.**

observe, that while we cannot prove the fairness of perfect difference sets theoretically, they behave fair and useable for duty cycles typically found in applications. We therefore showed, that perfect difference sets are useable and beneficial approach, not only in theory but also in praxis for symmetrical and asymmetrical asynchronous neighbor discovery.

## 7. CONCLUSIONS

We presented an energy efficient neighbor discovery scheme based on perfect difference sets with the provably optimal power-latency trade-off, which does not require any coordination between nodes. We defined fairness for asymmetric neighbor discovery, proved the *Grid* scheme to be fair, and showed that our scheme behaves fair for duty-cycles used in practice. We implemented a neighbor discovery framework into Linux and TinyOS for general use. Finally, we used our framework to build a podcast sharing application. In this context we show, how to deal with nodes which are present, but do not have new available data and can be ignored.

## Acknowledgements

This research was funded in part by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC), German Research Foundation grant DFG EXC 89. We would like to thank Ragesh Jaiswal and Amitabha Bagchi of IIT, Delhi for early feedback.

## 8. REFERENCES

[1] Bluetooth SIG. Bluetooth technical core specification version 4.0, 2010. `http://www.bluetooth.org/Technical/Specifications/adopted.htm`.

[2] Colbourn, C., and Dinitz, J. *Handbook of combinatorial designs.* CRC press, 2006.

[3] Dutta, P., and Culler, D. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (2008), SenSys '08, pp. 71–84.

[4] Farrell, S., McMahon, A., Meehan, E., Weber, S., Lynch, A., Hartnett, K., and Brodie, S. Report on an arctic summer dtn 2010 trial. Tech. rep., Trinity College, Dublin, Ireland, 2011. DRAFT 2011-05-18 Work-in-progress. `http://dtn.dsg.cs.tcd.ie/n4c-summer10/summer10.pdf`.

[5] Harvard University. Motelab. `http://motelab.eecs.harvard.edu/`.

[6] Helgason, O. R., Yavuz, E. A., Kouyoumdjieva, S. T., Pajevic, L., and Karlsson, G. A mobile peer-to-peer system for opportunistic content-centric networking. In *MobiHeld* (Sept. 2010).

[7] IEEE. IEEE Standard for Information technology, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, 2007.

[8] Jiang, J.-R., Tseng, Y.-C., Hsu, C.-S., and Lai, T.-H. Quorum-based asynchronous power-saving protocols for ieee 802.11 ad hoc networks. *Mob. Netw. Appl. 10* (February 2005), 169–181.

[9] Joung, Y.-J. Quorum-based algorithms for group mutual exclusion. *IEEE Transactions on Parallel and Distributed Systems 14* (2003), 463–476.

[10] Jung, S., Lee, U., Chang, A., Cho, D.-K., and Gerla, M. Bluetorrent: Cooperative content sharing for bluetooth users. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 47–56.

[11] Kandhalu, A., Lakshmanan, K., and Rajkumar, R. R. U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks* (2010), IPSN '10, ACM, pp. 350–361.

[12] Lang, S., and Mao, L. A torus quorum protocol for distributed mutual exclusion. In *Proc. of the 10th Int'l Conf. on Parallel and Distributed Computing and Systems* (1998), Citeseer, pp. 635–638.

[13] Liu, T., Sadler, C., Zhang, P., and Martonosi, M. Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services* (2004), ACM, pp. 256–269.

[14] Maekawa, M. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst. 3* (May 1985), 145–159. `http://doi.acm.org/10.1145/214438.214445`.

[15] McGlynn, M. J., and Borbash, S. A. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing* (2001), MobiHoc '01, ACM, pp. 137–145.

[16] Pillip Levis et al. TinyOS, 2010. `http://www.tinyos.net/`, accessed 24.11.10.

[17] Singer, J. A theorem in finite projective geometry and some applications to number theory. *Trans. Amer. Math. Soc. 43* (1938), 377–385.

[18] Sorber, J. Into the wild: taming uncertainty in perpetual mobile networks. In *Proceedings of the 2009 MobiHoc S 3 workshop on MobiHoc S 3* (2009), ACM, pp. 41–44.

[19] Tien-Tsin, W.-S. L., shing Luk, W., and tsin Wong, T. Two new quorum based algorithms for distributed mutual exclusion. In *17th International Conference on Distributed Computing Systems* (1997), IEEE, pp. 100–106.