# Improving Neighbor Discovery with Slot Index Synchronization

Shuaizhao Jin*, Zixiao Wang†, Wai Kay Leong†, Ben Leong†, Yabo Dong*, Dongming Lu*
*School of Computer Science and Technology, Zhejiang University
†Department of Computer Science, National University of Singapore
*{jinszhao, dongyb, ldm}@zju.edu.cn
†{zixiao, waikay, benleong}@comp.nus.edu.sg

***Abstract*—Neighbor discovery is essential for *docking* applications, where mobile nodes communicate with static nodes situated at various rendezvous points. In existing neighbor discovery protocols, the probabilistic protocols perform well in the average-case but have aperiodic, unpredictable and unbounded discovery latency. While the deterministic protocols can provide a bounded worst-case discovery latency, they achieve this by sacrificing the average-case performance. In this paper, we propose a new synchronization technique, called Mobility-Assisted Slot index Synchronization (MASS). MASS improves the average-case performance of deterministic neighbor discovery protocols via *slot index synchronization*, without incurring additional energy consumption. We evaluate MASS through both theoretical analysis and simulations of the real traces from a tourist tracking system deployed at Mogao Grottoes, a famous cultural heritage site in China. We show that MASS can reduce the average discovery latency of state-of-the-art deterministic neighbor discovery protocols by up to 2 orders of magnitude.**

## I. Introduction

The availability of cheap wireless sensor nodes has made *docking applications* [3] feasible for widespread deployment. In docking applications, there are two sets of sensor nodes: static and mobile nodes. Static nodes are placed at fixed rendezvous points and mobile nodes are attached to moving objects of interest. Some examples of docking applications include the uploading of cargo transit history [15], the tracking of cattle movement during feeding times [28], the tracking of hikers via their encounters with trail-side waypoints [10] and our application, which tracks tourists at a cultural heritage site [18].

For the conservation work in Mogao Grottoes, a famous cultural heritage site in China, it is essential to track and study how the movement of tourists around the caves affects the micro-climate. The current system in place consists of static nodes placed in the caves and mobile nodes carried by the tour guides. The static nodes emit beacons which are picked up by the mobile nodes for each tour group that enters a cave. Not only is the detection of the tour group important, it is also essential to accurately measure the duration of stay within each cave. In addition, power is also a constraint as modern infrastructure is not allowed to be installed in the caves to supply power to the static nodes.

In the current system, the static nodes are configured at a duty cycle of 0.6% (they wake up and send beacons for 30 ms every 5 s) and the mobile nodes are always active and continuously listen for the beacons. While this achieves an average discovery latency of 2.5 s, the mobile nodes have to be charged daily, which is inconvenient and is also a major impediment to our plans for scaling up the system. Our goal is to reduce energy consumption by introducing duty cycling for the mobile nodes, without compromising discovery latency.

Neighbor discovery protocols will allow both the static and mobile nodes to duty cycle while ensuring discovery. Our key insight is that *discovery latency can be minimized by synchronizing the active and sleeping slots of the nodes*. This is because when the slot indices are synchronized, the nodes will discover each other at the next earliest active slot.

While this might sound straightforward, it turns out that synchronizing the nodes for a docking application introduces a number of challenges. First, the synchronization must be done in a distributed manner as the static nodes cannot directly communicate with each other. Instead, they can only communicate by using the mobile nodes to relay information. Second, the synchronization should converge quickly in order to achieve any gains. Finally, the intrinsic inaccuracies in the clocks of the sensors will result in *clock drift*. If the synchronization of the slots drifts by a small amount, the resulting latency could potentially become the worst case, instead of being optimal. In other words, synchronization could significantly degrade performance instead of improving performance if we are not careful.

We propose and evaluate *Mobility-Assisted Slot index Synchronization (MASS)*, which improves the average-case performance of deterministic neighbor discovery protocols via *slot index synchronization*. In MASS, the nodes first elect a reference node in a distributed manner and then synchronize their slot indices with that reference node. We exploited the natural visiting patterns of tourists to elect the reference node. For our Mogao Grottoes traces, MASS is able to synchronize all the 60 static nodes within three hours, which is less than half of the 10-hour operational period of the site. But even if it is not able to synchronize all the nodes, we can still reduce the discovery latency for the nodes that are synchronized. To mitigate the impact of the clock drift, MASS employs existing techniques to estimate the clock skew between the nodes and compensate for the clock drift with respect to the elected reference node. The experiments show that our sensor nodes can achieve the error of 0.98 ms and 2.4 ms per hour for 50% and 90% of the time, respectively, which is well within one slot interval.

We evaluate MASS via simulation using a real 31-day trace obtained from the existing Mogao Grottoes tourist tracking system and show that MASS can reduce the discovery latency of 4 existing deterministic protocols (BlindDate [27], Searchlight [1], Disco [3] and U-Connect [11]) by about 2 orders of magnitude over 75% of the time. Therefore, with MASS, we are able to achieve a discovery latency of less than 5 s for 70% of the time, even when the duty cycle of the nodes is reduced to 0.5% to save power. By reducing the duty cycle from 100% to 0.5%, our mobile nodes will be able to last 200 times longer and be charged every 6 months instead of daily.

To the best of our knowledge, we are the first to propose the use of slot index synchronization to improve neighbour discovery latency in docking applications. We addressed the resulting challenges by utilizing existing techniques such as clock skew estimation and clock drift compensation, and proposing a synchronization process and priority metric. It remains as future work to thoroughly investigate how MASS performs on practical sensor networks, especially on the tourist tracking system deployed at the Mogao Grottoes.

The rest of the paper is organized as follows: in Section II, we present an overview of existing neighbor discovery protocols as well as clock synchronization and drift compensation techniques. In Section III, we describe our docking application scenario, our key insight and explain the challenges for our system. In Section IV, we describe the MASS algorithm. We evaluate MASS in Section V and conclude in section VI.

## II. Related Work

**Clock Synchronization.** There are two common approaches for clock synchronization in multihop wireless networks [25]: (i) reference-based clock synchronization; and (ii) distributed clock synchronization. In reference-based clock synchronization, the nodes synchronize their clocks with respect to one or more reference nodes. These reference nodes are known as roots in tree-based protocols [6, 23], gateways in cluster-based protocols [4], and time servers in NTP-based protocols [31]. The main drawback of reference-based protocols is that they are not robust to the failures of reference nodes. Furthermore, they all assume that a reliable means of communication exists between the nodes. This assumption does not hold for our tourist tracking system.

In distributed clock synchronization, all nodes run the same distributed synchronization algorithm, which will cause their local clocks to converge to a common global time value. Some techniques to achieve global synchronization include having each node advance its clock to the fastest clock [21, 34], or to the average clock values of the local nodes [13, 20, 22]. Glossy is a recent network flooding architecture that achieves an average time synchronization error below one microsecond [5]. However, it is not suitable in docking applications because the mobile nodes are not always connected. To cope with frequent topology changes and long inter-contact duration, Choi et al. developed distributed asynchronous clock synchronization (DCS) for delay tolerant networks [2]. Global clock synchronization is achieved by asynchronously compensating for clock errors using relative clock information exchanged among nodes in DCS. The drawback of distributed synchronization algorithms is that convergence is slow (it typically takes hundreds of iterations) [8]. Furthermore, such protocols only work well in a closed system, where no new nodes join the network once it starts.

**Neighbor Discovery Protocols**. Neighbor discovery protocols are needed in duty-cycled networks to ensure discovery. In these protocols, time is typically divided into slots of equal size. At each slot, sensor nodes either sleep or wake up according to some pattern determined by the protocol. Such protocols can be divided into two broad categories: *probabilistic* and *deterministic*. In probabilistic protocols, the sleep-wake schedule is based on some randomized function [17]. With the right parameters, probabilistic protocols can achieve good average-case performance. Unfortunately, the worst-case performance is unbounded, i.e., there is a small chance that two nodes will never discover each other.

On the other hand, deterministic protocols provide a bounded worst-case latency as the sleep-wake schedule is designed to ensure discovery within one sleep-wake period. Prime-based protocols such as Disco [3] and U-Connect [11] guarantee a bounded discovery latency based on the Chinese Remainder Theorem [19]. Quorum-based deterministic protocols [12, 26] have slots that are grouped into a 2-dimensional $m \times m$ array for a period of $m^2$. Each node picks one row and one column of the entries as its active slots, thus ensuring that any two nodes will have at least two overlapping active slots in each period. The state-of-the-art Searchlight [1] protocol further reduces the overlapping slots to at least one by grouping the slots into an array of size $\lfloor \frac{m}{2} \rfloor \times m$. The duty cycle can be further reduced by *striped probing*, where the probe skips every two slots. Extending the duration of the active slots ensures overlap and guarantees detection. BlindDate [27] is a recently proposed protocol to improve the performance of Searchlight by having two dynamic probe slots traversing towards each other in opposite directions within each period. BlindDate also requires the extension of active slots like Searchlight. Sun et al. proposed Hello [24], a generalized framework for deterministic protocols that can represent existing protocols such as Quorum, Disco, U-Connect and Searchlight, using a set of parameters. They proved that Searchlight is the optimal symmetric protocol. Our technique, MASS, complements existing deterministic protocols and can significantly reduce discovery latency through slot index synchronization.

## III. A Case for Slot Index Synchronization

In this section, we introduce the background for our target docking application system, and show that slot index synchronization is a promising technique to improve the performance of deterministic neighbor discovery protocols. We also explain why distributed synchronization algorithms are not feasible for our application.

### A. Tourist Tracking in Historical Sites

Mogao Grottoes, also known as the Caves of the Thousand Buddhas, is a famous cultural heritage site in China. It consists

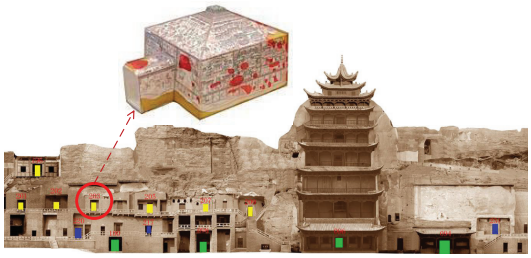


Fig. 1: Mogao Grottoes

of hundreds of caves cut into the side of a cliff, some of which have Buddhist murals painted all over the walls and ceilings. Figure 1 shows a small section of the Mogao Grottoes. Thousands of tourists visit the Mogao Grottoes every day, and the high volume of human traffic is causing deterioration to the priceless historical artwork. Therefore, it is important to monitor and limit the number of visitors to each cave to mitigate the over-exposure of the artwork due to human contact.

The challenge of tracking tourists in such a historical site is that modern infrastructure such as power supply lines cannot be installed. Neither are fixtures allowed to be drilled into the walls for fear of damaging the cultural artefacts. The current tracking system deployed uses static low-power sensor nodes placed at the corners of the caves which periodically emit beacon signals. Each tour guide holds a mobile sensor node that continuously listens for and logs these beacons as the tour group visits each cave. At the end of the tour, the durations of the stay for each tour group in the caves is extracted and recorded from the mobile nodes.

The tourist tracking system is an example of a *docking* application. As the cave walls block the radio signals, static nodes cannot communicate with each other directly. It is also not feasible to deploy more nodes to create a fully connected wireless sensor network due to conservation restrictions on the number of nodes that can be deployed.

The lack of power infrastructure also means that static nodes have to be powered by battery, which implies that energy consumption is a very important consideration. To reduce energy consumption, the static nodes currently perform duty cycle where they spend 30 ms every 5 s listening and broadcasting beacons. On the other hand, the mobile nodes have to be continuously listening to detect static nodes as well as record the entering and leaving time of tourists, thereby requiring daily battery recharging. A neighbor discovery protocol could be used to allow both the static and mobile nodes to duty cycle, at the cost of increased discovery latency.

The *discovery latency* is determined by the neighbor discovery protocol and the duty cycle rate determines the trade-off between discovery latency and energy consumption. In our case, it is equally important to also measure the duration of stay for tourists in the caves to monitor the environmental effects of human traffic.

### B. Slot Index Synchronization

Probabilistic neighbor discovery protocols like Birthday [17] have lower average-case discovery latencies, but the worst-case

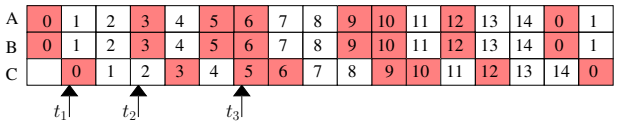



Fig. 2: Disco with a pair of primes (3, 5)

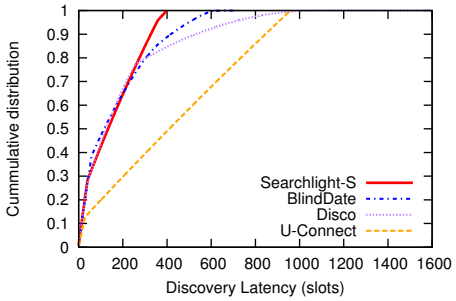



Fig. 3: Cumulative distribution of discovery latency for the various protocols at duty cycle of 5%.

latency is unbounded. Deterministic protocols like Disco [3] and Searchlight [1] have slightly higher average discovery latencies but bounded worst-case latencies. Our key observation is that *slot index synchronization has a significant impact on the discovery latency for deterministic protocols*.

To illustrate this, Figure 2 shows the discovery process between three nodes that run Disco with the parameters (3, 5). This gives a duty cycle of about 50% and a period of 15 slots. The shaded and white boxes represent active and sleeping slots respectively. In this example, node A and B are strictly synchronized, and node C is out-of-sync by one slot. Suppose all three nodes come into contact at time $t_1$, node A and B will discover each other at time $t_2$ with a latency of two slots, while node C will discover the rest at time $t_3$ with a latency of five slots. It is clear from this illustration that when the nodes are synchronized, the worst-case latency is the largest gap between the active slots.

To investigate the improvement that can be achieved with synchronized nodes, we enumerated all possible slot offsets between a pair of nodes running the same symmetric protocol, and recorded the discovery latency as the number of slots it takes from first contact to the first intersecting active slot for the two nodes. We treated adjacent active slots as a successful detection as well, because perfect alignment rarely exists in real life and partially overlapping slots are sufficient for detection. This is especially important for BlindDate and Searchlight-S (Searchlight with striped probing) because perfect alignment may cause failure of detection. To overcome this, these protocols extend the active slot so as to ensure overlap of adjacent active slots.

We computed the distribution of the discovery latency of the different protocols with duty cycles of 5% and 1% [1, 24, 27], and plot the result for 5% duty cycle in Fig. 3. The result for 1% duty cycle is similar. Our analysis shows that, with the same duty cycle, Searchlight-S is the best performing protocol with the lowest average-case and worst-case latency.

We also compared the average-case and worst-case latency of each protocol against the case where both nodes have their slot indices synchronized in Table I. The key observation is

TABLE I: Average and worst-case discovery latency. Values in brackets indicate the latency in seconds.

| Protocol | *Overall Performance* | | *Synchronized Index* | | Average Latency Improvement | Worst-case Latency Improvement | Parameter(s) |
|---|---|---|---|---|---|---|---|
| | Average Latency (slots)[s] | Worst-case Latency (slots)[s] | Average Latency (slots)[s] | Worst-case Latency (slots)[s] | | | |
| | **5% Duty-cycle, 25 ms per slot** | | | | | | |
| Searchlight-S [1] | 151 [3.78] | 399 [9.98] | 12.3 [0.31] | 37 [0.93] | 12.28 | 10.78 | 40 |
| BlindDate [27] | 168 [4.20] | 685 [17.13] | 13.8 [0.35] | 46 [1.15] | 12.17 | 14.89 | 12 |
| Disco [3] | 194 [4.85] | 1,071 [26.78] | 12.7 [0.32] | 36 [0.90] | 15.28 | 29.75 | (37, 43) |
| U-Connect [11] | 423 [10.6] | 960 [24.00] | 14.6 [0.37] | 30 [0.75] | 28.97 | 32.00 | 31 |
| | **1% Duty-cycle, 5 ms per slot** | | | | | | |
| Searchlight-S | 4,711 [23.56] | 9,999 [50.00] | 65.7 [0.33] | 197 [0.99] | 71.70 | 50.76 | 200 |
| BlindDate | 6,387 [31.94] | 17,821 [89.11] | 71.4 [0.36] | 238 [1.19] | 89.45 | 74.88 | 60 |
| Disco | 10,125 [50.63] | 35,655 [178.28] | 64.1 [0.32] | 180 [0.90] | 157.96 | 198.08 | (181, 211) |
| U-Connect | 11,123 [55.62] | 22,800 [114.00] | 74.6 [0.37] | 150 [0.75] | 149.10 | 152.00 | 151 |

that the average and worst-case latency are significantly lower when the nodes are synchronized on their slot indices. This result is intuitive since two synchronized nodes follow the same sleep-wake pattern and thus wake up at the same time. What is surprising is the amount of improvement. Even for the best performing protocol, Searchlight-S at 1% duty cycle, the average latency is reduced by 72 times. With the slot size of 5 ms, we can reduce the average latency from 24 s to 0.33 s, and the worst-case latency from 50 s down to about 1 s.

While slot index synchronization improves performance greatly, we cannot always guarantee the synchronization between two nodes due to the inherent errors in the sensor hardware. Hence, the effectiveness of slot synchronization is dependent on the accuracy of synchronization.

### C. Distributed Synchronization Not Feasible

In a docking application, static nodes are not able to communicate with each other directly and have to rely on mobile nodes to relay information. One approach to achieve synchronization would be to use a distributed clock synchronization algorithm such as DCS [2]. The duty cycle of the nodes can then be derived from the resulting reference clock, and synchronizing their clocks will naturally synchronize the slot indices.

However, this approach does not work for two reasons. First, we have found that DCS either fails or takes an extremely long time to synchronize the clocks of nodes to within an accuracy of one slot duration. This means that either node indices will never be synchronized, or the improvement would be small due to the long convergence time. Second, DCS is designed to work in a closed system, where no nodes leave or join the system. Otherwise, it can no longer guarantee convergence. In our docking application, the mobile nodes can leave and new mobile nodes may enter the system at any time.

Because of the shortcomings of the distributed algorithms, we adopt a reference-based technique for synchronization, where a node is elected to be the reference node that all other nodes synchronize with. Our key observation is that although the mobile nodes do not follow a pre-determined path, the movement is not completely random. Thus, we exploited the movement pattern of tourists and designed a simple algorithm to elect the reference node.

## IV. Mobility-Assisted Slot Synchronization

We have shown that slot index synchronization can significantly improve latency. In this section, we describe the design of Mobility-Assisted Slot index Synchronization (MASS) for our tourist tracking application at Mogao Grottoes.

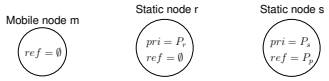
Since static nodes cannot directly communicate with each other, mobile nodes are used to relay information between the static nodes. The challenge is that mobile nodes do not move along pre-determined paths. Thus, we cannot determine prima facie which static node will be visited next. However, we observed from the traces of our real-world application that the paths of the mobile nodes tend to follow a certain set of patterns. We can exploit the patterns to elect relatively stable reference nodes in a distributed manner, and have the non-reference nodes synchronize their slot index with these reference nodes. In addition, we use the clock of the reference node as the reference clock for clock drift compensation.

All static and mobile nodes will adopt the same neighbor-discovery protocol with a common set of parameters. Thus, if all (or most) of the static nodes have their slot indices synchronized, mobile nodes can likewise have their slot indices synchronized with every static node. This results in a very small discovery latency when they come into contact with any of the synchronized static nodes.

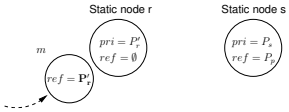
### A. Distributed Reference Election & Synchronization

To elect the reference nodes, we introduce the notion of a priority metric $P$. Each static node $s$ computes a priority $P_s$ based on information carried by the mobile nodes in a distributed way. In addition, each static node $s$ also stores the recorded priority of its reference, $P_{s.ref}$ obtained from the mobile nodes. Similarly, each mobile node, $m$, also stores the priority ($P_m$) of the last static node that it is synchronized with. When a mobile node $m$ encounters a new static node $s$, the static node first updates its priority $P_s$. Then, the priorities $P_s$, $P_{s.ref}$, and $P_m$ are compared and exchanged.

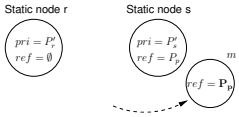
If $P_m$ is larger than $P_s$ and $P_{s.ref}$, then the static node will synchronize its clock and slot index according to that of the mobile node, and set its reference $P_{s.ref}$ to $P_m$. Indirectly, the static node will have now synchronized with the static node that the mobile node has last synchronized with. On the other



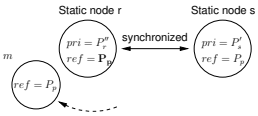

Mobile node $m$ has no initial reference. Static node $r$ also has no reference while static node $s$ has previously synced with node $p$.




Node $m$ encounters node $r$, synchronizes with $r$ and updates its reference to $P_r$.




Now $m$ leaves and encounters node $s$. Suppose $P'_r < P'_s$ and $P'_s < P_p$, node $m$ syncs with node $s$ and updates its reference to $P_p$.




Suppose now $m$ returns to node $r$, and suppose even after updating $P''_r$, $P''_r < P_p$, then node $r$ will sync with $m$ and record $P_p$ as its reference. Now nodes $r$ and $s$ will be in sync.

Fig. 4: Reference node election.

TABLE II: Summary of update rules for reference node election

| Condition | Static node | Mobile node |
|---|---|---|
| $P_m > P_s$ and $P_m > P_{s.ref}$ | $P_{s.ref} \leftarrow P_m$ | No Change |
| $P_m < P_s$ or $P_m < P_{s.ref}$ | No Change | $P_m \leftarrow \max(P_s, P_{s.ref})$ |

hand, if $P_m$ is smaller than $P_s$ or $P_{s.ref}$, the mobile node will synchronize its slot index with the static node. It will also set its reference $P_m$ to the larger of $P_s$ and $P_{s.ref}$. Figure 4 illustrates the steps of this process and Table II summarizes the above update rules. With this simple algorithm, the node with the largest priority value will be elected as the reference node, from which other nodes will take reference. It remains for us to describe how a static node $s$ determines and computes its priority $P_s$.

**Node Priority**. For our reference node election algorithm to work well, not only does the priority $P_s$ need to be a metric that can be easily computed in a distributed way, the resulting priority for different static nodes should also preferably be distinct. In our docking application, we observed that some caves were more popular than others due to either its intrinsic attractiveness to the tourists, or its relative location, i.e., it might be near route entrances. Such caves will have more visitors than others. Thus, we decided to utilize the average inter-arrival time between mobile nodes at each static node $s$ to obtain its priority $P_s$.

The static nodes compute $P$ by measuring the time elapsed since the last visit to the cave. The priority is accumulated using an exponentially-weighted moving average (EWMA) with a smoothing factor $\alpha = \frac{1}{8}$. In other words, the equation to update $P_i$ to $P_{i+1}$ is:

$$P_{i+1} = \alpha t + (1 - \alpha) P_i \tag{1}$$

where $t$ is the elapsed time since the last visit to this cave.

In Figure 5, we plot the actual average inter-arrival time

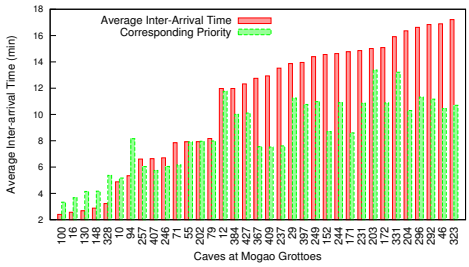



Fig. 5: Average inter-arrival time and the priority of different caves at Mogao Grottoes for day one.

of each cave over one day of a real trace and the computed priority $P_i$. Although the computed priority is often not close to the actual daily average values, the trends are similar, i.e., the more frequently visited nodes will have a higher priority.

It is entirely plausible that for some docking applications, the static nodes have similar visiting frequencies and the computed priorities might be very close to one another. In such cases, the inter-arrival time might not be the best metric and another metric could be used. However, our reference election algorithm is still applicable if a good metric can be found.

### *B. Clock Drift Compensation & Slot Synchronization*

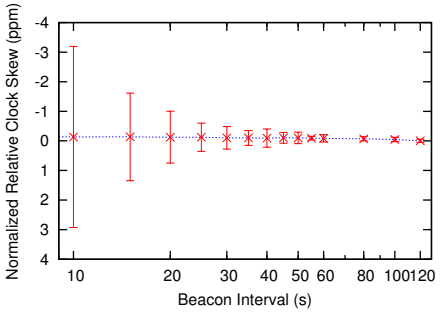
Once a reference node is elected, each node will synchronize their clocks to that reference node. However, clock drift will introduce errors, even if a node is perfectly synchronized at the start. Hence, both static and mobile nodes in our system must continuously estimate and compensate for clock drift.

**Clock Skew Estimation.** There are many existing algorithms for clock skew detection and estimation [7, 14, 30]. Zhong et al. showed that it is possible to continuously detect and estimate the clock skew between two nodes [33]. We believe that any of the state-of-the-art techniques could be used for MASS. For validation, we performed a simple experiment: a sensor node periodically sends beacons at different intervals to three sensor nodes that keep listening to compute relative clock skew, using the following formula [9, 29]:

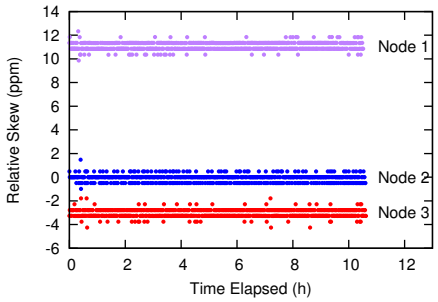
$$\delta_{AB} = \frac{(t^A_{i+1} - t^A_i) - (t^B_{i+1} - t^B_i)}{t^A_{i+1} - t^A_i}$$

where $t^A_i$ and $t^B_i$ are the local time at the sender's and receiver's clock when the $i$th beacon was sent and received. The MAC-layer time-stamping technique was used to mitigate the variance of the delivery time [16].

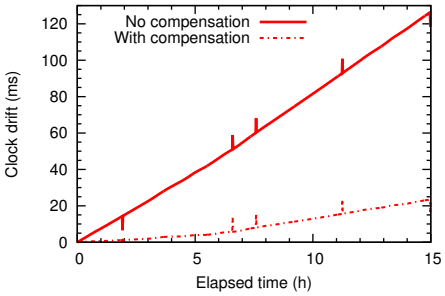
In Figure 6(a), we plot the average estimated relative clock skew and the standard deviation in parts per million (ppm) for beacon intervals from 1 s to 120 s. We found that a larger beacon interval leads to a more accurate and stable estimation. In Figure 6(b), we plot the relative clock skew between the sending node and the three receiving nodes with the beacon interval of 60 s over a few hours. We observed that the three receiving nodes can reliably detect and measure their relative clock skew against the sending node with an error within 1.5 ppm. In Figure 6(c), we plot the time difference between the clocks of two nodes with and without clock drift compensation, which suggests that it is possible to estimate and compensate for the clock drift in our system.




(a) Mean and variance of clock skew measured at one node using different time intervals.

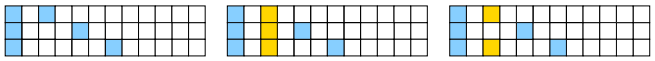



(b) Clock skew measured at 1-min intervals between the sending node receiving nodes.




(c) Amount of clock drift between two nodes with and without clock drift compensation.

Fig. 6: Clock skew of the three receiving nodes.



(a) Searchlight-S (b) Searchlight-S+1 (c) Searchlight-S+1/2

Fig. 7: Additional active slots to the Searchlight-S protocol with $p = 12$.

**Clock Drift Compensation.** Once we have an estimate of the clock skew, sensor nodes can compensate for its clock drift against the reference node by adjusting its clock periodically. Due to the granularity of the clocks, the period of adjustment cannot be too small. We found that having the nodes adjust their clocks every 100 s worked well for our sensor hardware.

From our Mogao Grottoes traces, tourists (and therefore the mobile nodes) will typically stay in each cave for more than 120 s (see Figure10(c)), which as Figure 6(a) suggests, is sufficiently long to get a good estimate of the clock skew. Since the average inter-arrival time of the mobile nodes at each cave is less than 18 minutes (see Figure 5), the expected clock drift during this interval is within one slot (5 ms) with clock drift compensation.

**Slot Alignment** While it is possible for the nodes to align their slot timing to the reference node, we found that the resulting improvements from doing so are marginal. As such, we adopt a simple index synchronization process — the nodes simply update the current slot index to the reference node's slot index. Because every node is running the same neighbor discovery protocol, every active slot of such synchronized pairs of nodes will overlap.

### C. Mitigating the Pitfalls of Small Synchronization Errors

Even with clock drift compensation, we cannot guarantee perfect synchronization all the time. It turns out to be a big problem that can potentially lead to the worst-case scenario for discovery latency. In particular, an offset of two slots will result in the worst-case scenario for Searchlight-S (the optimal symmetric protocol) [24].

To overcome this potential pitfall, we introduce a small modification to the duty-cycle pattern for the neighbor discovery protocol by adding an extra number of active slots. We illustrate this with Searchlight-S. In Searchlight-S, nodes wake up at every $p$ slots and a probe slot traverses from the first position to $p/2$ across $p/2$ sub-cycles (see Figure 7(a)), resulting in a period of $p(p/2)$. To prevent the worst-case latency from occurring when the cycles of two nodes have an offset of 2, we set the node to wake up at every $kp + 2$ slots (see Figure 7(b)). We call this scheme *Searchlight+1*. With the additional active slots, the duty cycle of Searchlight+1 will be 1.5 times that of Searchlight with the same $p$. To keep the duty-cycle constant, $p$ has to be increased by 1.5 times, thereby incurring a slightly larger latency. One possible way to reduce this additional delay is to halve the number of extra active slots by only introducing them every 2 sub-cycles (See Figure 7(c)). We call this modified scheme *Searchlight+1/2*.

We compared the average-case and worst-case latencies for our modifications when the slots indices are offset by 0, 1, 2 and 3, to Searchlight-S in Table III. We see that the latencies when the slots are offset by 2 can be significantly reduced, at the cost of slightly longer cycles, which increases the overall latency by a small amount. Figure 8 shows that our modified schemes slightly increase the overall latency.

TABLE III: Combined latencies of the modified Searchlight protocols for cycles offset by 0 and 1 at 5% duty-cycle.

| | Searchlight-S | | Searchlight-S+1 | | Searchlight-S+1/2 | |
|---|---|---|---|---|---|---|
| | Latency (slots) | | | | | |
| | Avg | Worst | Avg | Worst | Avg | Worst |
| Offset 0, 1 | 12.3 | 37 | 18.5 | 57 | 15.3 | 47 |
| Offset 2 | 199.5 | 399 | 29.4 | 59 | 47.6 | 99 |
| Offset 3 | 163.5 | 359 | 29.2 | 59 | 43.4 | 99 |
| Average | 125.1 | 265 | 25.7 | 58.3 | 35.4 | 81.7 |

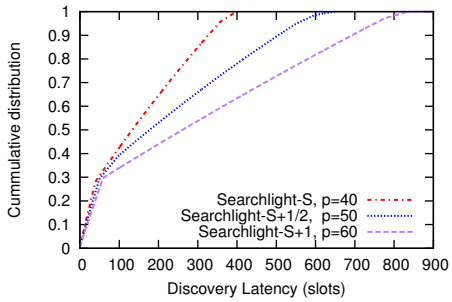



Fig. 8: Distribution of latency for the modified Searchlight protocols at 5% duty cycle.

## V. Evaluation

In this section, we evaluated the effectiveness of MASS in enhancing existing deterministic neighbor discovery protocols

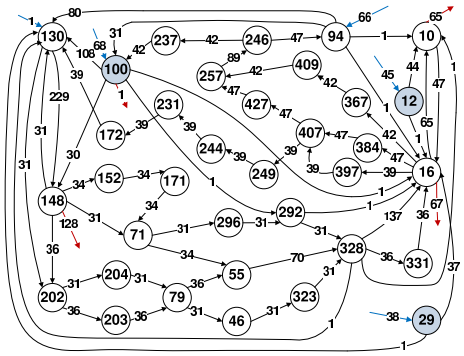



Fig. 9: Visiting route of the tour groups at Mogao Grottoes on 1 Aug 2013. The number on the arrows indicates the number of mobile nodes that took that path. Shaded caves indicate the reference nodes elected by the end of the day.

using our custom trace-based simulator. Our simulator simulates the interactions between the mobile and static nodes using the real traces of tourist movements collected at Mogao Grottoes. The operating hours of the Mogao Grottoes are from 8:00 am to 6:00 pm daily, so we have approximately 10 hours of data for each day. The data set used for our simulations consists of 31 days of traces over the one-month period in August 2013, containing 8,658 movement routes for the mobile nodes and 69,271 cave visits. As an example, the routes for a particular day are shown in Figure 9, where each cave is assigned a unique ID.

### A. Power Conservation and Latency Reduction

In neighbor discovery protocols, the power saving is determined by the duty cycle, which is the proportion of time that a node stays active. The typical duty cycles used in these protocols are 1% and 5%, and the slot size is typically 25 ms [1, 24, 27].

In Figure 10(a), we plot the cumulative distribution of the discovery latency of the five most common neighbor discovery protocols using a 5% duty cycle and a slot size of 25 ms. We see that as expected, Searchlight-S is the best deterministic protocol, which is consistent with our earlier analysis.

As energy consumption is an important factor in docking applications, we tried to reduce energy consumption by decreasing the duty cycle to 1% (while maintaining the original slot size of 25 ms) and plot the results in Figure 10(b). The discovery latency for all protocols increased by about a factor of 10. Furthermore, the worst-case latency can now be longer than the time that the tourists spend in the caves, which can potentially cause detection failures. We plot the distribution of the time that the mobile nodes spend in the caves in Figure 10(c), which confirms our intuition and shows that the worst-case neighbor discovery latency needs to be below 200 s for the reliable tracking of tourists.

We can reduce the discovery latency by using a smaller slot size. However, how small we can go depends on the hardware and operating system of the nodes. Zhang et al. suggested that the smallest feasible size for a slot is 5 ms, because any smaller size will cause jitter and make the system unstable [32]. Also, 5 ms of air time might be insufficient to exchange usable data. However, we argue that once discovery happens, the nodes can switch to a different communication protocol for the exchange of information. Thus, we set the slot size to 5 ms with a duty cycle of 1% and plot the new cumulative distribution of the discovery latency in Figure 11(a). We see that as expected, the discovery latency improves, but the latency increases to almost 5 times that of the original as in Figure 10(a).

**Adding Synchronization**. Next, we investigated how synchronization would improve discovery latency. We repeated the simulation with a duty cycle of 1% and a slot size of 5 ms after introducing synchronization with DCS and MASS. We plot the results in Figures 11(b) and 11(c) respectively.

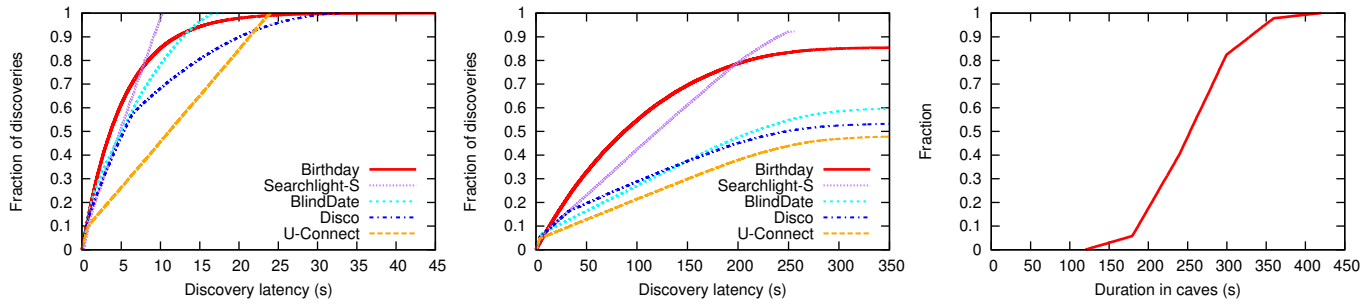
As expected, synchronization has no impact on the performance of the Birthday protocol. Although in general, synchronization improves the latency for deterministic protocols, the improvements with DCS are marginal. This is because DCS is not able to converge sufficiently fast in our scenario. For MASS, the discovery latency for the same protocols was reduced to less than one second about 75% of the time, which suggests that most of the nodes were successfully synchronized. The worst-case latency, however, remained the same, likely because either some nodes could not be synchronized such as cave 12 and 29, or they had to be re-synchronized at the start of each day.

The reduction in discovery latency is important for our tourist tracking application because our goal is to accurately track the duration of stay for tourists in each cave. The median time that visitors spend in each cave is 250 s. Detecting the visitors with a 25 s delay (at 1% duty cycle) results in an error of 10%. By reducing the discovery latency to less than 1 s, we can improve the estimation accuracy by more than 20 times.

### B. Performance gains using MASS

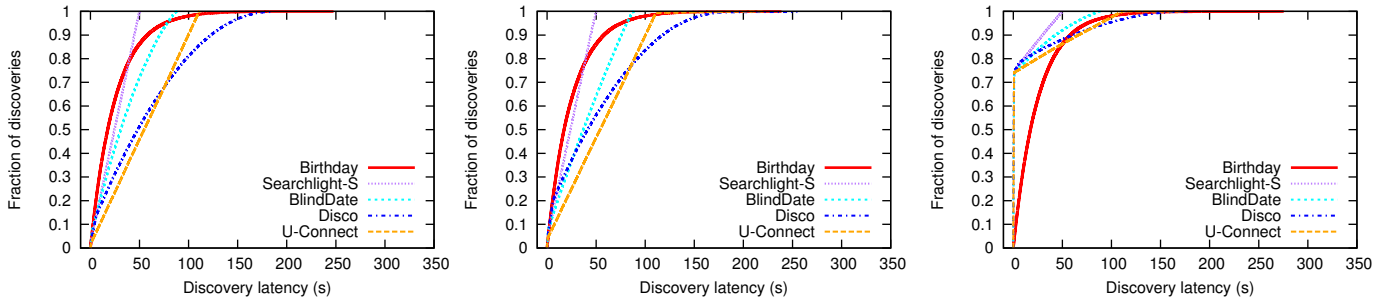
As the duty cycle affects the discovery latency of neighbor discovery protocols, we plot the percentage improvement of the average discovery latency with MASS over the original protocols in Figure 12. An improvement of 90% means that the resulting average discovery latency with MASS is 10% of the original latency. The results show that as the duty cycle increases, the improvement decreases for Searchlight-S and BlindDate. This is because at larger duty cycles, the original protocols already achieved relative low discovery latencies and there is little room for improvement. On the other hand, Disco and U-Connect show the same improvement across different duty cycles.

Reducing duty cycles is beneficial to save power, but at a cost of the increased discovery latency. We examined how MASS affects this trade off by further reducing the duty cycle while keeping the slot size at 5 ms. Figure 13 shows the distribution of the discovery latency for Searchlight-S with MASS using smaller duty cycles. Because Searchlight-S has the lowest worst-case latency, we could reduce the duty cycle to 0.5% while maintaining 100% discovery success rate in our application. As expected, MASS still manages to keep the discovery latency extremely low 75% of the time. That




(a) 5% duty cycle and 25 ms slot size. (b) 1% duty cycle and 25 ms slot size. (c) Duration of mobile nodes stay in caves

Fig. 10: Impact of reducing duty cycle from 5% to 1% for Birthday, BlindDate, Disco, Searchlight and U-Connect.




(a) No synchronization. (b) With DCS. (c) With MASS.

Fig. 11: Discovery latency of each protocols and MASS and DCS assisted versions at 1% duty cycle and 5 ms slot size.

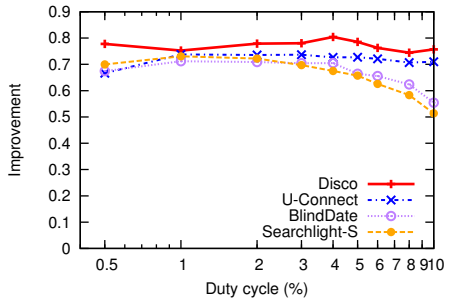



Fig. 12: Improvement for various protocols with MASS under different duty cycles with 5 ms slot size.

is, decreasing the duty cycle does not affect the improvements of MASS. Only the worst-case latency is affected.

### C. Effectiveness of Clock Skew Compensation

Typical clock skew ranges from $\pm 5$ ppm to $\pm 100$ ppm [33]. We showed experimentally that it is possible for clocks to drift apart by 5 ms in just one hour (See Figure 6(c)). If the relative clock skew is 20 ppm, and since the average inter-arrival time of the mobile nodes in our application is 15 minutes (See Figure 5), their clocks would have drifted apart by 18 ms, which is much larger than our slot size of 5 ms. This would suggest that clock drift compensation is essential to ensure the synchronization among the nodes.

To validate this intuition, we repeated our experiments without clock drift compensation and compared MASS and DCS using the Searchlight-S protocol in Figure 14. As expected, DCS has no impact on Searchlight-S. Without clock drift compensation, the improvement of MASS is very limited. The same trends are observed for the other neighbor discovery protocols. This demonstrates that clock drift compensation is essential to maintain the synchronicity among the nodes.

### D. Random Traces

Our MASS algorithm exploits the natural visiting patterns of the mobile nodes in a docking application for synchronization. In this section, we examine how MASS performs if the visiting patterns are completely random. To ensure a fair comparison, we generated random traces that had the same node distribution as our Mogao Grottoes traces.

Figure 15 shows the distribution of the discovery latencies for Disco, BlindDate and Searchlight-S with and without MASS. The results show that while MASS could still improve the latency 30% of the time, it is not as effective as before (75% of the time). This is because we used the inter-arrival time as the metric for MASS, which is random in the generated traces. Thus, our reference election algorithm could not maintain a fixed reference node for a prolonged period of time. This highlights that the selection of the metric is a key factor that affects the overall performance for MASS in docking applications.

## VI. Conclusion

In this paper, we show that slot index synchronization is a practical technique that can significantly improve the average-case performance of deterministic neighbor discovery protocols. By exploiting the mobility pattern of the mobile nodes in *docking* applications, we developed MASS, a reference-based slot index synchronization technique, that can improve the average discovery latency, while keeping the energy consumption constant. We showed with simulations based on real traces,

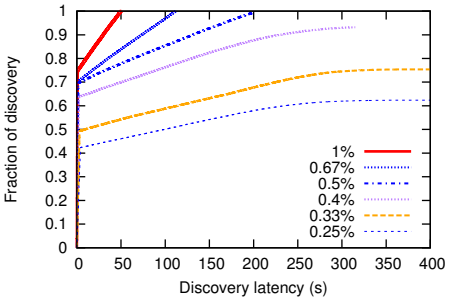



Fig. 13: Distribution of discovery latency of Searchlight-S with MASS at very low duty cycles.

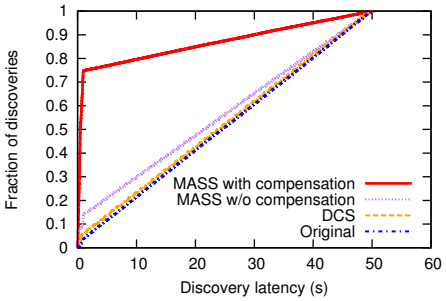



Fig. 14: Distribution of discovery latency for Searchlight-S without clock drift compensation under 1% duty cycle and 5 ms slot size.

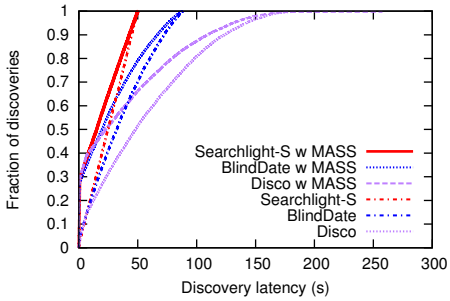



Fig. 15: Distribution of discovery latency with random routes.

that MASS can improve the average discovery latency by up to 2 orders of magnitude. Our work represents a preliminary investigation into improving existing deterministic neighbor discovery protocols by introducing slot index synchronization. It remains as future work to thoroughly investigate how MASS will perform on real sensor networks.

## Acknowledgements

This research was carried out at the SeSaMe Center. It is supported by the Singapore NRF under its IRC@SG Funding Initiative (administered by the IDMPO), the National High Technology Research and Development Program of China (No.2012AA101701), and the National Key Technology Support Program of China (No.2013BAK01B00, No.2014BAK16B00).

## References

[1] M. Bakht, M. Trower, and R. H. Kravets. Searchlight: Won't you be my neighbor? In *Proceedings of MobiCom '12*, Aug. 2012.

[2] B. J. Choi, H. Liang, X. Shen, and W. Zhuang. DCS: Distributed asynchronous clock synchronization in delay tolerant networks. *IEEE Trans. Parallel Distrib. Syst.*, 23(3):491–504, 2012.

[3] P. Dutta and D. Culler. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proceedings of SenSys '08*, Nov. 2008.

[4] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS OSR*, pages 147–163, 2002.

[5] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of IPSN '11*, pages 73–84, April 2011.

[6] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of SenSys '03*, Nov. 2003.

[7] B. R. Hamilton, X. Ma, Q. Zhao, and J. Xu. ACES: Adaptive clock estimation and synchronization using kalman filtering. In *Proceedings of MobiCom '08*, Sep. 2008.

[8] J. He, P. Cheng, L. Shi, and J. Chen. Clock synchronization for random mobile sensor networks. In *Proceedings of CDC '12*, Dec 2012.

[9] H. Huang, J. Yun, Z. Zhong, S. Kim, and T. He. PSR: Practical synchronous rendezvous in low-duty-cycle wireless networks. In *Proceedings of INFOCOM '13*, Apr. 2013.

[10] J.-H. Huang, S. Amjad, and S. Mishra. Cenwits: A sensor-based loosely coupled search and rescue system using witnesses. In *Proceedings of SenSys '05*, Nov. 2005.

[11] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar. U-connect: A low-latency energy-efficient asynchronous neighbor discovery protocol. In *Proceedings of IPSN '10*, Apr. 2010.

[12] S. Lai, B. Ravindran, and H. Cho. Heterogenous quorum-based wake-up scheduling in wireless sensor networks. *IEEE Trans. Comput.*, 59(11):1562–1575, 2010.

[13] Q. Li and D. Rus. Global clock synchronization in sensor networks. *IEEE Trans. Comput.*, 55(2):214–226, 2006.

[14] C. Liao and P. Barooah. Distributed clock skew and offset estimation from relative measurements in mobile networks with markovian switching topology. *Journal Automatica*, 49(10):3015–3022, 2013.

[15] M. Malinowski, M. Moskwa, M. Feldmeier, M. Laibowitz, and J. A. Paradiso. Cargonet: A low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events. In *Proceedings of SenSys '07*, Nov. 2007.

[16] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proceedings of SenSys '02*, Nov. 2004.

[17] M. J. McGlynn and S. A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proceedings of MobiHoc '01*, Oct. 2001.

[18] X. Ming, D. Yabo, L. Dongming, X. Ping, and L. Gang. A wireless sensor system for long-term microclimate monitoring in wildland cultural heritage sites. In *Proceedings of IPSA '08*, Apr. 2008.

[19] I. Niven, H. S. Zuckerman, and H. L. Montgomery. *An introduction to the theory of numbers*. John Wiley & Sons, 2008.

[20] M. Sasabe and T. Takine. A simple scheme for relative time synchronization in delay tolerant MANETs. In *Proceedings of INCoS '09*, Nov. 2009.

[21] J.-P. Sheu, C.-M. Chao, and C.-W. Sun. A clock synchronization algorithm for multi-hop wireless ad hoc networks. In *Proceedings of ICDCS '04*, Mar. 2004.

[22] P. Sommer and R. Wattenhofer. Gradient clock synchronization in wireless sensor networks. In *Proceedings of IPSN '09*, Apr. 2009.

[23] W. Su and I. F. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 13(2):384–397, 2005.

[24] W. Sun, Z. Yang, K. Wang, and Y. Liu. Hello: A generic flexible protocol for neighbor discovery. In *Proceedings of INFOCOM '14*, April 2014.

[25] W. Sun, Z. Yang, X. Zhang, and Y. Liu. Energy-efficient neighbor discovery in mobile ad hoc and wireless sensor networks: A survey. *IEEE CST*, pages 1448–1459, March 2014.

[26] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. *Computer Networks*, 43(3):317–337, 2003.

[27] K. Wang, X. Mao, and Y. Liu. BlindDate: A neighbor discovery protocol. In *Proceedings of ICPP '13*, Oct 2013.

[28] T. Wark, W. Hu, P. Sikka, L. Klingbeil, P. Corke, C. Crossman, and G. Bishop-Hurley. A model-based routing protocol for a mobile, delay tolerant network. In *Proceedings of SenSys '07*, Nov. 2007.

[29] M. Xu and W. Xu. Taco: Temperature-aware compensation for time synchronization in wireless sensor networks. In *Proceedings of MASS '13*, Aug. 2013.

[30] Z. Yang, J. Pan, and L. Cai. Adaptive clock skew estimation with interactive multi-model Kalman filters for sensor networks. In *Proceedings ICC '10*, May 2010.

[31] Q. Ye and L. Cheng. DTP: Double-pairwise time protocol for disruption tolerant networks. In *Proceedings of ICDCS '08*, June 2008.

[32] D. Zhang, T. He, Y. Liu, Y. Gu, F. Ye, R. K. Ganti, and H. Lei. ACC: Generic on-demand accelerations for neighbor discovery in mobile applications. In *Proceedings of SenSys '12*, Nov. 2012.

[33] Z. Zhong, P. Chen, and T. He. On-demand time synchronization with predictable accuracy. In *Proceedings of INFOCOM '11*, Apr. 2011.

[34] D. Zhou and T.-H. Lai. An accurate and scalable clock synchronization protocol for IEEE 802.11-based multihop ad hoc networks. *IEEE Trans. Parallel Distrib. Syst.*, 18(12).