Chapter 19
Neighbor Discovery in Wireless Sensor Networks

Abstract Wireless sensor networks (WSNs) are widely used in many applications such as air pollution monitoring, natural disaster prevention, health care monitoring, etc. The sensor nodes are deployed in the monitored area and they form a wireless network through communication with nearby sensors. In this chapter, we introduce the fundamental process in constructing a wireless sensor network, which is called **neighbor discovery**, where the sensors can find nearby neighboring sensors when their distance is within a threshold. There are two main reasons for studying the neighbor discovery problem. First of all, the deployed sensors as a configuration may vary dynamically according to different reasons. For example, new sensors may be added and old ones removed; and some types of sensors have the ability to move around inside the area. Therefore, a sensor may need to find nearby neighboring sensors when they move into its communication range or some new sensors are deployed in the vicinity. The second reason is the limited power supply. In most situations, sensors are powered by battery and their energy is very limited. For example, suppose a sensor is powered by a 1200 mAh battery, and the processor consumes 2 mA under full power and the radio consumes 20 mA when it is turned on. If the sensor keeps the radio on and computing never stops, the lifetime of the sensor is a little more than two days. Therefore, sensors need some special method to save energy in order to extend their lifetime. A trivial way is to add sleeping mode, where a sensor keeps silent in a sleep state for most of the time, wakes up for work for only a small fraction of the time. Practically, in sleep mode, the processor's power consumption drops to 2 μA and the radio power drops to 1 μA, and thus lifetime can be extended significantly. If the sensor wakes up for only 1% of the time and sleeps for 99% of the time, the estimated lifetime would be half a year. If the wake-up portion is 0.1% of the time, the lifetime can be over five years. Therefore, we assume sensors are silent for most of the time, and they wake up mainly for data collection and communication. The neighbor discovery problem is to design the schedules of the sensors' sleep mode and wake mode, such that two nearby sensors can be in wake mode at the same time to find each other, which is a kind of rendezvous problem. In this chapter, we first propose a motivational example in Sect. 19.1, and formulate the problem in Sect. 19.2. We introduce the trivial brute force algorithm for the neighbor discovery problem in Sect. 19.3 and another two algorithms: relaxed difference set based algorithm and co-prime algorithm in Sects. 19.4 and 19.5 respectively. Finally, we summarize the chapter in Sect. 19.6.

Z. Gu et al., *Rendezvous in Distributed Systems*,
DOI 10.1007/978-981-10-3680-4_19

19.1 Motivational Example

In sensor networks, the sensors may turn on or off their radio periodically, randomly or according to some pre-defined rules. In Fig. 19.1, three sensor nodes have different wake-up schedules, where the top lines of each node mean the radio is on while the bottom lines mean the radio is off. The sensor nodes use different wake-up schedules and two neighboring users can find each other if they are close enough and their radios are both turned on at the same time. Suppose all three nodes are near to each other, and they can discover each other if they turn on the radio at the same time. As shown in the figure, node *a* and node *b* may find each other when they both turn on the radios, but node *a* and node *c* cannot find each other since their wake-up times do not intersect. We call the latter uncoordinated schedules which could be the result of improper initialization, crash at unpredicted times, etc.

Some MAC protocol research proposed a sampling method where a sensor node would sample radio activities during its sleeping times, to try to learn their wake-up schedules of nearby nodes. If the neighboring nodes' schedules can be predicted, the node can add some additional wake-up times to its own schedule to achieve coordination with the other nodes. For example, node *a* in can learn the other two nodes' schedules during its sleeping times, and it will revise its wake-up strategy such that it can discover its neighbors (see Fig. 19.1).

Although the learning method can help solve the neighbor discovery problem, it increases the ratio of wake-up time, which leads to shorter lifetime. Therefore, in this chapter, we study how to design efficient algorithms such that the sensor nodes can finally discover each other with a coordinated wake-up schedule, while keeping a low wake-up ratio.

In designing neighbor discovery algorithms for sensor networks, we face the following challenges:

(1) Symmetric schedule: the sensors nodes should adopt the same wake-up schedule since they are anonymous and placed in a distributed fashion in the monitoring area;
(2) asynchronous time clock: each sensor node may have a local time clock, and they will execute according to its own time clock.

In many practical applications, a large number of sensor nodes are deployed in the monitoring area and it is impossible to design a special wake-up schedule for each

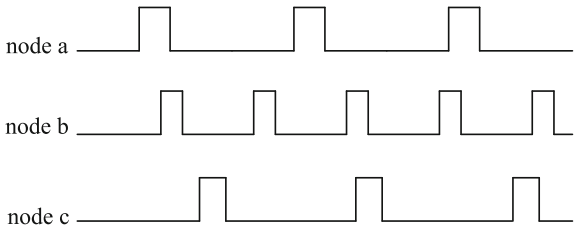

Fig. 19.1 An example

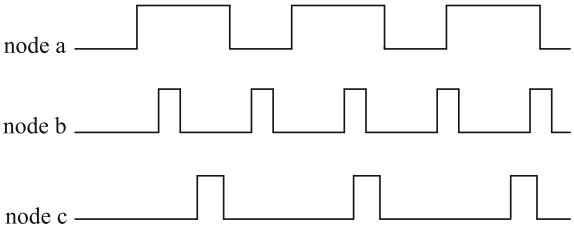

Fig. 19.2 An example of learning the neighboring nodes' wake-up schedule

sensor node. Therefore, a symmetric schedule for all sensors is preferred. Different sensor nodes may be deployed at different times, their local clocks are different and they will execute the neighbor discovery algorithm according to its own clock. The designed algorithms should take all this into account.

19.2 Problem Definition

We define the terms and variables used to formally describe the neighbor discovery problem for wireless sensor networks.

Definition 19.1 (*Time slot*) we divide time into slots of equal length, where two neighboring nodes can discover each other during one unit time slot if they both have their radios turned on.

Definition 19.2 (*Slot aligned network*) suppose there exists a global clock where the time is divided into slots of equal length. For any node in the network, the start time of its own clock must be the start time of any slot in the global clock.

For example, in Fig. 19.3, the start time of node *a*'s local clock meets the start time of the global time slot 3, while node *b*'s start slot meets the start time of the global time slot 6. If some nodes are not slot aligned, choose a nearby time slot to continue. This slot aligned idea is also used in traditional rendezvous.

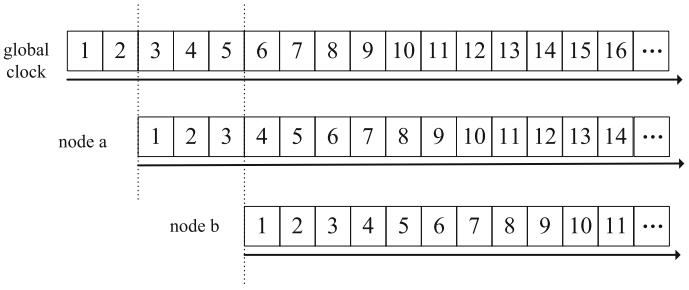

Fig. 19.3 An example of slot aligned network

Definition 19.3 (*Wake-up schedule*) each sensor node has limited energy and the wake-up schedule is defined as the time pattern of when it should wake up or sleep.

By designing the wake-up schedule, a sensor node will be only awake for a portion of the time. This extends the sensor's lifetime and it can also discover its neighbors in an energy-efficient manner.

For a sequence:

$$S = \{s(1), s(2), \ldots, s(t), \ldots, s(T)\}$$

where $s(t)$ means the state of the sensor at time t, we formally define the wake-up schedule as such:

Definition 19.4 (*The neighbor discovery (wake-up) schedule*) of the sensor node u_a is defined as a sequence $S_a = \{s_a(1), s_a(2), \ldots, s_a(t), \ldots, s_a(T)\}$ where T is the length and each element $s_a(t)$ is:

$$s_a(t) = \begin{cases} 0 \text{ if sensor } u_a \text{ sleeps in slot } t \\ 1 \text{ if sensor } u_a \text{ wakes up in slot } t \end{cases}$$

Definition 19.5 (*Duty cycle*) d_a of the neighbor discover schedule S_a is defined as:

$$d_a = \frac{|\{1 \leq t \leq T : s_a(t) = 1\}|}{T}$$

We define the asynchronous situations of time slots as:

Definition 19.6 (*Clock drift*) means every two neighboring sensor nodes may have random clock drifts. Suppose node u_a is k time slots earlier than node u_b, we can rotate the neighbor discovery schedule by k slots as:

$$rotate(S_a, k) = \{s_a^k(t) | s_a^k(t) = s_a((t + k - 1) \mod T + 1), t \in [1, T]\}$$

Suppose two sensor nodes u_a and u_b have corresponding neighbor discover schedules S_a and S_b, and user u_a is k time slots earlier than node u_b; we define the neighbor discovery problem between them as:

Definition 19.7 (*Neighbor discovery*) means there exists time slot $t \in [1, T_a * T_b]$ such that $s_a^k(t) = s_b(t) = 1$ where $T_a = |S_a|$, $T_b = |S_b|$.

When both elements $s_a^k(t)$ and $s_b(t)$ are equal to 1, two sensors wake up at time slot t (in node u_b's clock), and then they can discover each other when they are close enough. Slot t is also called discovery slot between the two sensors. When node u_a is k slots later than user u_b, we can let $k < 0$ and the definition also works.

For example, the neighbor discover schedule for node u_a is:

$$S_a = \{0, 0, 0, 1, 0\}$$

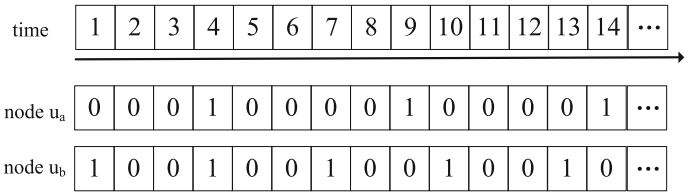

Fig. 19.4 An example of neighbor discovery between two synchronous nodes

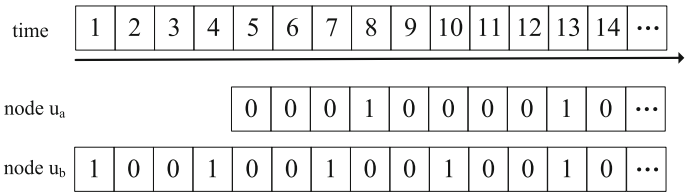

Fig. 19.5 An example of neighbor discovery between two asynchronous nodes

while the neighbor discover schedule for node u_b is:

$$S_a = \{1, 0, 0, 1, 0, 0\}$$

As shown in Fig. 19.4, if there is no clock drift between two nodes, they can find each other in time slot 4, and Fig. 19.5 shows the example when node u_a is 4 time slots earlier than node u_b. Clearly, two nodes can also discover each other in time slot 13 (and it is time slot 9 in node u_a's clock). In these two examples, two sensor nodes adopt different wake-up schedules which is not that practical. Symmetric wake-up schedule is preferred by two and definitely by more nodes.

19.3 Brute Force Algorithm

Since the sensor nodes may start the neighbor discovery algorithm at any time, if the sensor nodes keep awake all the time, two neighbors can clearly find each other. However, the duty cycle then of the sensors is 100%, which is inefficient. An easy improvement is to make the sensor nodes wake up for k time slots, where $k \geq \lceil \frac{N+1}{2} \rceil$. We describe the method as follows:

(1) Time is divided into rounds and each round contains N time slots;
(2) in each round, the sensor node keeps awake in the first $k \geq \lceil \frac{N+1}{2} \rceil$ time slots and asleep for the other time slots.

Notice that, when $k = \lceil \frac{N+1}{2} \rceil$, two different sensors should have at least one common slot when they are all awake, and this is called the "51%" solution, since the node keeps awake for more than half of the time in each round.

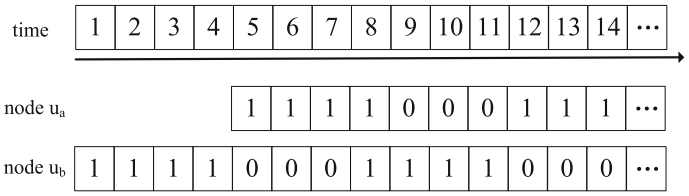

Fig. 19.6 An example of brute force algorithm

For example, if $N = 7$ and $k = 4$, two nodes u_a and u_b can find each other no matter when they start the process in Fig. 19.6. Clearly, it is easy to compute the time complexity to discover each other, which is bounded within $T \leq \lceil \frac{N+1}{2} \rceil$.

19.4 Relax Difference Set Based Algorithm

Another method is to design a neighbor discovery schedule on the basis of relaxed difference set (RDS). Recall the definition of RDS:

Definition 19.8 A set $D = \{a_1, a_2, \cdots, a_k\} \subseteq Z_n$ (the set of all nonnegative integers less than n) is called a Relaxed Difference Set (RDS) if for every $d \neq 0 \pmod n$, there exists at least one ordered pair (a_i, a_j) such that $a_i - a_j \equiv d \pmod n$, where $a_i, a_j \in D$.

For any n time slots, we can design a fixed relaxed difference set under Z_n and the sensors nodes wake up according to the difference set can find each other efficiently. We describe the algorithm in Algorithm 19.1.

Algorithm 19.1 Relaxed Difference Set Based Algorithm

```
Choose a positive integer n > 0;
Construct a relaxed difference set D = {a_1, a_2, ..., a_k} ⊆ Z_n;
Denote time slot t := 0;
Denote the constructed neighbor discovery schedule as S := {s(0), s(1), ..., s(t), ...};
while Not discover the other node do
  t' := t mod T;
  index := t' mod n;
  if index ∈ D then
    s(t) := 1;
  else
    s(t) := 0;
  end if
end while
```

As shown in Algorithm 19.1, we divide time into rounds of n time slots and construct a relaxed difference set under Z_n. Then, we transform the relaxed difference set D into a schedule of length n, as follows:

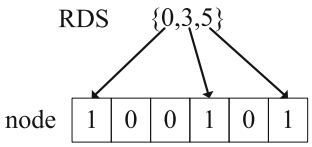

Fig. 19.7 An example of transforming an RDS to a neighbor discover schedule

$$s(t) = \begin{cases} 1 \text{ if } t \in D \\ 0 \text{ otherwise} \end{cases}$$

where $t \in [0, n)$. The sensor nodes repeat the sequence until they find the others. We illustrate the transformation in Fig. 19.7.

According to the definition of RDS, when one node is δ time slots later or earlier than another neighboring node, they can both find each other. We analyze the time complexity for discovering each other and the duty cycle of the designed schedule.

Obviously, no matter when the sensor nodes start the algorithm, they can find each other within n time slots, since the property of RDS guarantees that at least one common time slot exists such that they are awake during n consecutive time slots.

As described in Chap. 8, the size of the relaxed difference set can be as small as $\Omega(\sqrt{n})$ and the duty cycle of the constructed schedule is:

$$d = \frac{|RDS|}{n} = \frac{\Omega(\sqrt{n})}{n}$$

Actually, Chap. 8 shows the method of constructing an RDS with size $\sqrt{3n}$ and thus the duty cycle can be small.

Therefore, there is a tradeoff between the time complexity to discover each other and the duty cycle of the schedule. If we want to reduce the time of discovery, n should be small. If we want to reduce the duty cycle, we should increase n. Considering both, we can choose an appropriate n value in practice to implement the wake-up schedule.

19.5 Co-Prime Algorithm

Co-prime algorithm is based on the Chinese Remainder Theorem as discussed in Chap. 9. If two nodes use two different prime numbers to design their neighbor discovery schedules, they can find each other with very short duty cycle, if the chosen primes are different. This type of algorithms is as follows:

- For sensor node u_a, choose a random prime number p_a;
- time is divided into rounds where each round contains p_a time slots;

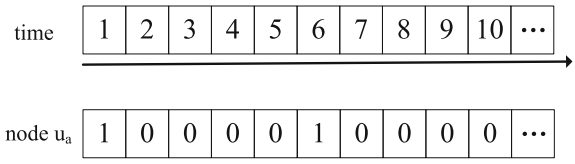

Fig. 19.8 An example of co-prime algorithm

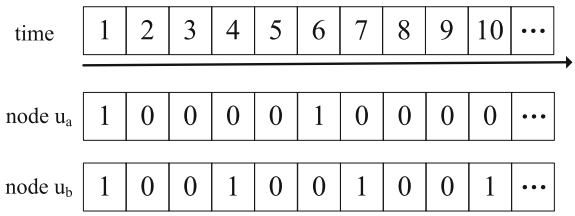

Fig. 19.9 An example of neighbor discovery when two nodes u_a and u_b start at the same time

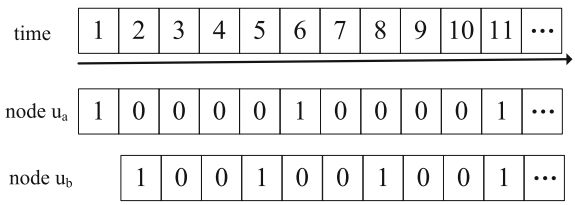

Fig. 19.10 An example of neighbor discovery when two nodes u_a and u_b start in different time slots

- the sensor node u_a wakes up in the first time slot of each round, and keeps asleep for the other $p_a - 1$ time slots.

For example, as illustrated in Fig. 19.8, if the node u_a chooses prime number $p_a = 5$, the constructed schedule is shown in the figure.

Considering any two neighboring nodes u_a and u_b, denote their chosen prime numbers as p_a and p_b respectively. If $p_a \neq p_b$, they can definitely find each other within $p_a * p_b$ time slots.

For example, node u_a chooses $p_a = 5$ and node u_b chooses $p_b = 3$; they can find each other in the first time slot if they start at the same time, as shown in Fig. 19.9, or they can discover the other in time slot 11 if node u_b is one time slot later than node u_a as depicted in Fig. 19.10 (the time complexity is actually 10 since node u_b is later).

We analyze the time complexity of neighbor discovery and the length of the duty cycle. Suppose node u_a is δ time slots earlier than node u_b. From node u_b's clock, node u_b will wake up in time slot $1 + p_b * l_b$ where p_b is the chosen prime number and l_b can be any positive integer. (Notice that, we assume time starts with slot 1, and sometimes we can also assume it starts with slot 0.) We rewrite the equation as:

$$t \equiv 1 \mod p_b$$

where node u_b wakes up in time slot t.

We then analyze when node u_a would wake up in node u_b's clock. When:

$$(t + k) \mod p_a \equiv 1$$

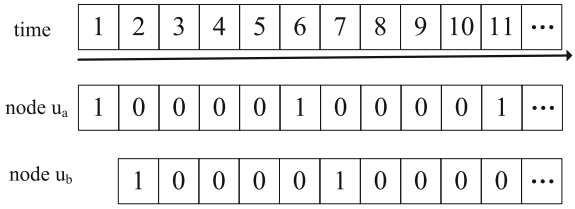

Fig. 19.11 An example of neighbor discovery does not happen if two sensor nodes choose the same prime number

where p_a is the chosen prime number, node u_a will be awake. We rewrite it as:

$$t \equiv 1 - k \mod p_a$$

where node u_a wakes up in time slot t.

Therefore, we need to compute such value t by combining both equations, and there exists $t \in [1, p_a * p_b]$ satisfying both equations from the Chinese Remainder Theorem. Therefore, the time complexity is bounded within $p_a * p_b$.

Obviously, the duty cycle of node u_a is $d = \frac{1}{p_a}$ which can be relatively short. But larger p_a value may lead to higher time complexity of neighbor discovery. Therefore, choosing the appropriate p_a value is important.

Notice that, if two sensor nodes choose the same prime number, they may not find each other if they do not start at the same time. As shown in Fig. 19.11, both nodes choose prime 5 and node u_b is 1 time slot later than node u_a, and they can never find each other. Therefore, the chosen prime numbers should be different, but it is hard to guarantee that the chosen primes are always different in practical wireless sensor networks.

19.6 Chapter Summary

In this chapter, we study neighbor discovery problem in wireless sensor networks. Since the sensors have only limited battery power, if they keep awake all the time, the sensors will go down soon. Therefore, we need to design wake-up schedules for the sensors to save energy and extend their lifetime. However, if different sensors wake up according to different schedules, two neighboring sensors may not discover each other if one of them happens to be asleep. The goal of neighbor discovery problem is to design efficient wake-up schedules for the sensors such that they can find each other in a short time, while having a low duty cycle of being awake.

We introduce three types of rendezvous algorithms. The first one is to be awake for most of the time (larger than 50%) and the sensors can find each other for sure. This type of algorithm is called the brute force algorithm. The second one is relaxed difference set based algorithm, where we use relaxed difference set to design wake-up schedules and the duty cycle can be reduced to $O(\frac{1}{N})$. The third one is co-prime algorithm where the sensors choose prime numbers to generate their wake-up schedules and it guarantees quick discovery if two sensors' chosen prime numbers are different.