Chapter 6
Asymmetric Blind Rendezvous Algorithms

Abstract In this chapter, we present asymmetric algorithms for the blind rendezvous problem. In the settings, we fix Alg as:

$$RS = < Alg - AS, Time, Port, ID, Non - Obli > \tag{6.1}$$

where $Time \in \{Syn, Asyn\}$, $Port \in \{Port - S, Port - AS\}$, and $ID \in \{Non - Anon, Anon\}$. Although there are eight different rendezvous settings when Alg is fixed as asymmetric and $Label$ fixed as non-oblivious, in designing asymmetric algorithms, users' ID do not matter much. This is because the users' IDs are typically used to break symmetry in distributed computing, but we already assume the users can be distinguishable and they execute different algorithms. Therefore, we present how to design efficient algorithms for the four rendezvous settings (synchronous and port-symmetric, asynchronous and port-symmetric, synchronous and port-asymmetric, asynchronous and port-asymmetric), no matter the choice of ID from $\{Non - Anon, Anon\}$. In Sect. 6.1, we present two different types of rendezvous algorithms for the synchronous and port-symmetric rendezvous setting, and these algorithms are extended for the asynchronous and port-symmetric rendezvous setting in Sect. 6.2. In Sects. 6.3 and 6.4, we introduce efficient algorithms for the synchronous, port-asymmetric and asynchronous, port-asymmetric rendezvous settings. Finally, we summarize the chapter in Sect. 6.5.

6.1 Synchronous and Port-Symmetric Rendezvous

Consider two users u_i and u_j, suppose their available port sets are $C_i, C_j \subseteq U$ respectively. In the following settings:

$$RS = < Alg - AS, Syn, Port\text{-}S, ID, Non - Obli > \tag{6.2}$$

two users have the same start time and both available port sets are the same: $C_i = C_j$.

Z. Gu et al., *Rendezvous in Distributed Systems*,
DOI 10.1007/978-981-10-3680-4_6

6.1.1 Smallest Port Accessing Algorithm

Algorithm 6.1 Smallest Port Accessing Algorithm

1: Denote the set of available ports as $C \subseteq U$;
2: Find the smallest label $s \in C$ and access port s all the time;

This setting is the simplest one and two users can adopt the Smallest Port Accessing (SPA) algorithm (as shown in Algorithm 6.1) to achieve rendezvous. In the algorithm, the user chooses the port with the smallest label all the time. It is obvious that two users with symmetric port situations will rendezvous in their first attempt.

6.1.2 Quorum-Based Channel Hopping

Quorum-based Channel Hopping (QCH) [1, 2] generates the hopping sequence based on the quorum system which is defined in [1]:

Definition 6.1 Given a finite universal set $U = \{0, 1, \ldots, n-1\}$ of n elements, a *quorum system* S under U is a collection of non-empty subsets of U, which satisfies the intersection property:

$$p \bigcap q \neq \emptyset, \forall p, q \in S \tag{6.3}$$

Each $p \in S$ (which is a subset of U) is called a quorum.

There are several ways of constructing a quorum system under set U and we will introduce a simple method called *cyclic quorum systems*, which is first introduced in [4]. To begin with, we introduce *relaxed cyclic difference set*.

Definition 6.2 A set $D = \{d_1, d_2, \ldots, d_k\} \subseteq U$ is called a relaxed cyclic (n, k)-difference set if for every $d \neq 0 \mod n$, there exists at least one ordered pair (d_i, d_j) where $d_i, d_j \in D$, such that $d_i - d_j \equiv d \pmod{n}$.

For example, if $n = 7, k = 3$, set $D = \{0, 1, 3\}$ is a relaxed cyclic $(7, 3)$-difference set under Z_7, where $Z_n = \{0, 1, \ldots, n-1\}$. Clearly, for any value $d \in \{1, 2, \ldots, 6\}$, there exist two elements in D that suit the equation. We define the cyclic quorum system as follows.

Definition 6.3 A group of sets $B_i = \{d_1 + i, d_2 + i, \ldots, d_k + i\} \mod n$, where $i \in \{0, 1, \ldots, n-1\}$ is a cyclic quorum system if and only if set $D = \{d_1, d_2, \ldots, d_k\}$ is a relaxed cyclic (n, k)-difference set.

We also use set $D = \{0, 1, 3\}$ as an example. We construct the cyclic quorum system as:

$$S = \{\{0, 1, 3\}, \{1, 2, 4\}, \{2, 3, 5\}, \{3, 4, 6\}, \{4, 5, 0\}, \{5, 6, 1\}, \{6, 0, 2\}\} \quad (6.4)$$

It is easy to check that any two elements in the quorum system S intersect. The QCH algorithm constructs different sequences on the basis of different quorums. Suppose there are N ports $\{1, 2, \ldots, N\}$ and there exists a cyclic quorum system $S = \{B_0, B_1, \ldots, B_{n-1}\}$ under Z_n. The QCH algorithm constructs sequence S_i for each quorum B_i as follows.

(1) Step 1: Denote $B_i = \{d_1, d_2, \ldots, d_k\}$;
(2) Step 2: For each port $1 \leq j \leq N$, construct a frame of N time slots $\{u_0, u_1, \ldots, u_{n-1}\}$ as:

$$u_l = \begin{cases} j \ \ if\ l \in B_i \\ * \ otherwise \end{cases} \quad (6.5)$$

where $*$ can be any port in set $\{1, 2, \ldots, N\}$.
(3) Step 3: The constructed sequence S_i is composed of such N frames and each frame consists of n elements.

For example, there are three ports $\{1, 2, 3\}$ and we can construct a cyclic quorum system under Z_3 as:

$$S = \{\{0, 1\}, \{1, 2\}, \{2, 0\}\} \quad (6.6)$$

For each quorum in set S, we construct the corresponding sequences. For quorum $\{0, 1\}$, we construct the sequence as:

$$S_1 = \{11 * |22 * |33*\} \quad (6.7)$$

where the symbol | separates different frames and $*$ is any port in $\{1, 2, \ldots, N\}$. Similarly, we can construct the other two sequences as:

$$\begin{aligned} S_2 &= \{*11| * 22| * 33\} \\ S_3 &= \{1 * 1|2 * 2|3 * 3\} \end{aligned} \quad (6.8)$$

For two different users u_a and u_b, they can choose different quorums in the constructed cyclic quorum system and they would choose the ports for rendezvous attempt according to the corresponding sequence. For example, user u_a chooses the constructed sequence S_1 to access the port periodically while user u_b chooses sequence S_2 for rendezvous. According to the definition of cyclic quorum system, it is easy to see that the corresponding quorums should intersect and the corresponding choice in the sequence should be the same port, which implies rendezvous. Therefore, the QCH algorithm can guarantee rendezvous for two synchronous users.

Notice that, the QCH algorithm is designed for the special situation that all port are available. By a small modification, it can be applied to the scenario that two users have symmetric available ports.

Suppose the available ports for the symmetric users are $C = \{p_1, p_2, \ldots, p_n\} \subseteq U$, which implies there are n available ports for the users. We reconstruct set $C' = \{1, 2, \ldots, n\}$ to apply the QCH algorithm. When we need to choose port i in set C', we replace it with port p_i in set C, which can be used to guarantee rendezvous in the port symmetric situation.

6.2 Asynchronous and Port-Symmetric Rendezvous

Consider two users u_i and u_j, suppose their available port sets are $C_i, C_j \subseteq U$. In the following settings:

$$RS = < Alg - AS, Asyn, Port\text{-}S, ID, Non - Obli > \tag{6.9}$$

two users start the rendezvous process in different time slots and both sets are the same, i.e. $C_i = C_j$.

6.2.1 Asynchronous Quorum-Based Channel Hopping

Asynchronous QCH (A-QCH) [3] is modified for asynchronous users, but only applicable to two available channels. We describe the A-QCH algorithm briefly and readers may refer to [3] for more details.

The QCH algorithm in Sect. 6.1.2 cannot be applied to two asynchronous users, because two users choosing different quorums p, q in a quorum system have clock skew; we can consider the situation as one user is adopting the rotated quorum by some bias, such as $rotate(q, k)$, which means each element in quorum q rotates k numbers. Then, quorum p and $rotate(q, k)$ may not intersect. The modification in A-QCH uses two cyclic quorum systems to construct such port accessing sequence, but it only works for two available ports.

Denote two available port as $P = \{p_0, p_1\}$, and suppose there are n time slots in each constructed frame. The algorithm works as follows:

(1) Denote the set Z_n as $\{0, 1, \ldots, n-1\}$;
(2) Find a minimal (n, k) cyclic difference set $D = \{d_1, d_2, \ldots, d_k\}$ under Z_n such that $k < \frac{n}{2}$;
(3) Construct the minimal cyclic quorum system $S = \{B_i | B_i = \{d_1 + i, d_2 + i, \ldots, d_k + i\} \mod n$ where $i \in [0, n-1]$;
(4) Find a relaxed (n, k') cyclic different set $D' = \{d'_1, d'_2, \ldots d'_{k'}\}$ under Z_n where $k' = \lceil \frac{n+1}{2} \rceil$) and $D' \bigcap D = \emptyset$;

Fig. 6.1 An example of the A-QCH algorithm

p_0	p_0	p_0	p_1	p_0	p_1	p_1	p_1	p_1
p_1	p_0	p_0	p_0	p_1	p_0	p_1	p_1	p_1
p_1	p_1	p_0	p_0	p_0	p_1	p_0	p_1	p_1
p_1	p_1	p_1	p_0	p_0	p_0	p_1	p_0	p_1
p_1	p_1	p_1	p_1	p_0	p_0	p_0	p_1	p_0
p_0	p_1	p_1	p_1	p_1	p_0	p_0	p_0	p_1
p_1	p_0	p_1	p_1	p_1	p_1	p_0	p_0	p_0
p_0	p_1	p_0	p_1	p_1	p_1	p_1	p_0	p_0
p_0	p_0	p_1	p_0	p_1	p_1	p_1	p_1	p_0

(5) Construct the cyclic quorum system $S' = \{B'_i | B'_i = \{d'_1 + i, d'_2 + i, \ldots, d'_{k'} + i\}$ mod n where $i \in [0, n-1]$;
(6) Construct the sequence with n frames and each frame contains n elements;
(7) For the jth frame, the ith element, we assign the port as:

$$s_{ji} = \begin{cases} p_0 & if\ i \in B_j \\ p_1 & if\ i \in B'_j \\ * & otherwise \end{cases} \tag{6.10}$$

where $*$ can be any port.
(8) The user accesses the port according to the constructed sequence periodically.

The method of constructing minimal (n, k) cyclic difference set and relaxed (n, k') cyclic different set can be found in [12] and we do not introduce the details. For example, if $n = 9$, and we construct set $D = \{0, 1, 2, 4\}$ and set $D' = \{3, 5, 6, 7, 8\}$. It is easy to check that both sets are relaxed cyclic difference set and $D \bigcap D' = \emptyset$. Then, we can construct the sequence as in Fig. 6.1, where there are 9 frames and each frame contains 9 elements.

Two different users compute different relaxed difference sets and the constructed sequences are different. However, by involving two quorum systems, two different users can always achieve rendezvous on the port p_0 or p_1 (notice that two symmetric users should have at least two available ports p_0, p_1 to execute the algorithm).

6.2.2 Sequential Accessing Algorithm

We propose the Sequential Accessing Algorithm in Algorithm 6.2. In the algorithm, we first count the number of elements in the available port set as n. In each time slot t, we compute the xth element in set C where x is t's modulus under n. This is similar to accessing the available ports sequentially from the 1th label to the nth label. When t is larger than n, we repeat the sequential accessing.

Algorithm 6.2 Sequential Accessing Algorithm

```
1: Denote time t := 1, the user's port set as C ⊆ U;
2: Denote the cardinality as n := |C|;
3: while Not rendezvous do
4:    Let x := (t − 1)%n + 1;
5:    Let p_id be the xth number in set C;
6:    Access port p_id in time t;
7:    t := t + 1;
8: end while
```

If two users are port-symmetric, but asynchronous, suppose one user u_i runs Algorithm 6.2 while user u_j runs a simple algorithm modified from Algorithm 6.1: user u_j chooses a label in set C_j and access the port all the time. It is easy to show that two users can rendezvous within $O(n)$ time slots.

Theorem 6.1 *Two port-symmetric, asynchronous users running Algorithm 6.2 and modified Algorithm 6.1 can rendezvous in $O(|C|)$ time slots, where C is the set of available ports.*

Proof Suppose user u_i starts Algorithm 6.2 later than user u_j. Suppose user u_j chooses the kth label in its available port set C_j, where $1 \leq k \leq |C_j|$. Obviously, when user u_i starts the algorithm, it can achieve rendezvous in k time slots, from user u_i's clock.

Supposing user u_i starts earlier than user u_j, when user u_j starts accessing the kth port, it may not achieve rendezvous with user u_i quickly. However, since user u_i repeats accessing the ports sequentially, it will definitely access the kth port within $|C_i|$ time slots.

Combining these two aspects, the theorem holds.

As illustrated in Fig. 6.2, the available port sets for two users are $\{1, 2, 7\}$; user u_i runs Algorithm 6.2 while user u_j runs the modified Algorithm 6.1 by accessing port 7. As shown in the figure, when user u_i starts earlier (as Fig. 6.2a) or later (as Fig. 6.2b) than user u_j, they can all achieve rendezvous in 3 time slots.

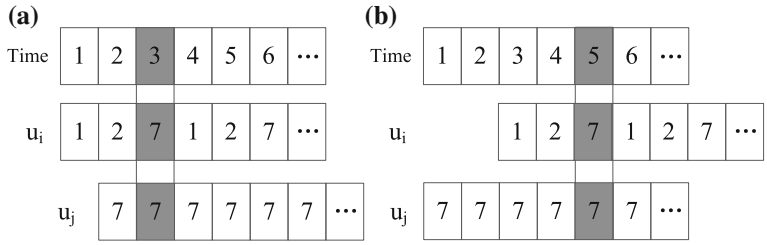

Fig. 6.2 Rendezvous examples when user u_i runs Algorithm 6.2 while user u_j runs the modified Algorithm 6.1

Actually, if both users are aware of the port-symmetric situation, they can also run a symmetric algorithm for rendezvous. Suppose both users adopt Algorithm 6.1 designed for synchronous and port-symmetric rendezvous. Without loss of generality, suppose user u_i starts $\Delta > 0$ time slots earlier than user u_j. When user u_j starts the rendezvous process at time $\Delta + 1$ (from user u_i's clock), it will access port s (the smallest port) in the first time slot (from user u_j's clock). As user u_i will always access port s, they could rendezvous in their first rendezvous attempt. Thus, the time to rendezvous is 1, where TTR is defined as the cost time to rendezvous for the user who starts latter in Problem 5.1.

Although port-symmetry is a easy situation to handle, the users are not aware of the situation and whether they are symmetric or not. Therefore, Algorithm 6.1 cannot work if two users have asymmetric ports. Therefore, we hope to design efficient algorithms that work for the asymmetric port situation, while it also has good performance when the ports are symmetric. We will introduce such algorithms in the following sections.

6.3 Synchronous and Port-Asymmetric Rendezvous

Consider two users u_i and u_j, and suppose their available port sets are $C_i, C_j \subseteq U$. In the following settings:

$$RS = < Alg - AS, Syn, Port\text{-}AS, ID, Non - Obli > \tag{6.11}$$

where two users start the rendezvous process at the same time but the sets of available ports may be different, i.e. $C_i \neq C_j$.

Algorithm 6.3 Modified Sequential Accessing Algorithm

```
1: Denote time t := 1, the user's port set as C ⊆ U;
2: while Not rendezvous do
3:   Let p_id := (t − 1)%N + 1;
4:   if p_id ∈ C then
5:     Access port p_id in time t;
6:   else
7:     Choose p_id randomly from C;
8:     Access port p_id in time t;
9:   end if
10:  t := t + 1;
11: end while
```

6.3.1 Modified Sequential Accessing Algorithm

We present the Modified Sequential Accessing (MSA) Algorithm as described as Algorithm 6.3. First of all, the user computes the port with id p_{id} corresponding to the current time slot t as $p_{id} = (t-1)\%N + 1$. Clearly, it is similar to accessing the ports by repeating the sequence $\{1, 2, \ldots, N\}$. However, due to occupancy by other services, some ports are not available for the user. Thus, it needs to select another available port randomly from set C. We show that, users u_i and u_j running Algorithm 6.2 can always achieve rendezvous within N time slots.

Theorem 6.2 *The synchronous users u_i and u_j can achieve rendezvous within N time slots, by running Algorithm 6.2 at the same time.*

Proof For the two neighboring users u_i and u_j, their sets of available ports must intersect to ensure at least one common available port exists. Therefore $C_i \bigcap C_j \neq \emptyset$. Denote the smallest number in set $C_i \bigcap C_j$ as s, clearly, $1 \leq s \leq N$.

When two users run the algorithm at the same time, when $t = s$, port s is available for user u_i since $s \in C_i$, and thus user u_i should access port s. Similarly, user u_j will access port s since it is available. Therefore, two users can access the connected ports and they rendezvous in time slot s. So the theorem holds.

6.4 Asynchronous and Port-Asymmetric Rendezvous

Consider two users u_i and u_j, and suppose their available port sets are $C_i, C_j \subseteq U$. In the following settings:

$$RS = < Alg - AS, ASyn, Port\text{-}AS, ID, Non - Obli > \tag{6.12}$$

where two users start the rendezvous process in different time slots and the sets of available ports may be different, i.e. $C_i \neq C_j$. This situation is the most difficult one in this chapter and we present some elegant results.

6.4.1 Sequential Access and Temporary Wait for Rendezvous

We present the Temporary Wait algorithm as in Algorithm 6.4. This algorithm works in this fashion: for each time slot t, compute the corresponding value x within range $[1, 2N^2]$ as $x := (t-1)\%2N^2+1$. We can think of this operation as repeating the time every $2N^2$ time slots. Following that, we divide the $2N^2$ time slots into N frames and each frame contains $2N$ time slots. This is why we compute $p_{id} := \lceil (x-1)/(2N) \rceil + 1$ (p_{id} corresponds to the frame that time slot t belongs to). Similar to the Modified Sequential Accessing Algorithm, if port p_{id} is not available, we will choose a random available port as a replacement. This process continues until rendezvous.

Algorithm 6.4 Temporary Wait Algorithm

```
1: Denote time t := 1, the user's port set as C ⊆ U;
2: while Not rendezvous do
3:    Let x := (t − 1)%2N^2 + 1;
4:    Let p_id := ⌈(x − 1)/(2N)⌉ + 1;
5:    if p_id does not belong to set C then
6:       Choose p_id as a random value from C;
7:    end if
8:    Access port p_id for rendezvous attempt;
9:    t := t + 1;
10: end while
```

We present a clear illustration in Fig. 6.3. The algorithm will access a fixed port for $2N$ time slots (if we do not consider the situation that it is not available and should be replaced). Then, after every $2N$ time slots, the algorithm will choose the next port for waiting (also over $2N$ time slots). And this is the reason we call it the Temporary Wait Algorithm. As shown in the figure, the user accesses a fixed port for $2N$ time slots, and the replacement happens if some port is not available. For example, port k replaces port 2 in the figure, if port 2 is not in the user's available port set.

For two users u_i and u_j, suppose one user (without loss of generality, u_i) adopts the Modified Sequential Accessing Algorithm (Algorithm 6.3) while the other user u_j runs the Temporary Wait Algorithm (Algorithm 6.4). We show that they can achieve rendezvous within $2N^2$ time slots.

Theorem 6.3 *Two users, adopting Algorithms 6.3 and 6.4 respectively, can achieve rendezvous within* $MTTR = 2N^2 = O(N^2)$ *time slots.*

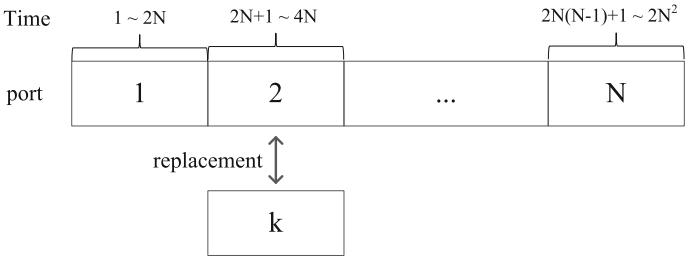

Fig. 6.3 The illustration of the Temporary Wait Algorithm

Proof For two neighboring users u_i and u_j, their sets of available ports must intersect to ensure at least one common available port exists. Therefore $C_i \bigcap C_j \neq \emptyset$. Denote the smallest number in set $C_i \bigcap C_j$ as s, clearly, $1 \leq s \leq N$.

First, we show that, supposing user u_j waits on port s for $2N$ time slots from $t+1$ to $t+2N$, if user u_i has begun Algorithm 6.3 no later than $t+N$, they can always achieve rendezvous. Actually, if user u_i starts the algorithm at time $t_i \leq t+N$, during the time slots from t_i to $t_i + N - 1$, user u_j will wait on port s, while user u_i will access port $\{1, 2, \ldots, s, \ldots, N\}$ sequentially (notice that user u_i does not access the unavailable ports, but it does affect the analysis since s is available). Therefore, they must rendezvous within these N time slots.

Then, we analyze the impact of asynchronous start. If user u_i starts the algorithm earlier than user u_j, it is clear that they can achieve rendezvous when user u_j waits on choosing port s, thus $TTR \leq s * 2N \leq 2N^2$. If user u_i starts later, the worst situation would happen when user u_j is finishing waiting on port s but u_i just starts. Considering any $2N$ time slots that user u_j waits on port s, denote them as $t+1$ to $t+2N$. If user u_i starts at time $t+2N-s+1$, user u_i will choose port s at time $t+2N+1$ but user u_j has just moved to the next port for waiting. However, after $2N*(N-1)$ time slots, i.e. from $t+N^2+1$ to $t+N^2+2N$, user u_j will also access port s and they will rendezvous in the first N time slots. Then, we can conclude that time to rendezvous is also bounded by $2N^2$ time slots if user u_i starts later.

Combining these, two users running asymmetric algorithms can achieve rendezvous within $2N^2 = O(N^2)$ time slots.

6.5 Chapter Summary

In this chapter, we present different types of rendezvous algorithms when the users can run asymmetric algorithms. In practical applications, the users in the distributed system may have different roles in the communications. For example, if one node in the system tries to broadcast a message to all neighboring nodes, it may play the role of "sender", while the other nodes who do not send messages are regarded as "receiver". For example, wireless sensor network (WSN) is a typical distributed

Table 6.1 Rendezvous algorithms for different rendezvous settings

Algorithms	Synchronous	Asynchronous
Port-symmetric	SPA, QCH	A-QCH, SAA
Port-asymmetric	MSA	TWA

system where the sensors can have different roles in constructing communication links. Normally, each sensor node can send or receive signals through radio (bidirectional or unidirectional radios). Suppose all sensor nodes are deployed in a monitoring area where they can sense environmental data. Further suppose there exists a mobile sink (it can be a moving vehicle which carries sensors or communication units) that travels through the monitoring area; when it is close to some deployed sensor nodes, it can send signals to activate these sensors and then collect data from them. In this case, the mobile sink can be regarded as the "sender", while the deployed sensors are "receivers". Therefore, they can execute different algorithms to establish communication links.

In this chapter, we mainly introduce algorithms for four different rendezvous settings: *synchronous and port-symmetric*, *asynchronous and port-symmetric*, *synchronous and port-asymmetric*, and *asynchronous and port-asymmetric*. Since the users' IDs are used to break symmetry among the users, we do not consider the impact of users' IDs in the chapter.

For the synchronous and port-symmetric setting, we present two rendezvous algorithms that can perform well. The first one (Smallest Port Accessing algorithm, SPA) simply accesses the port with smallest label, while the second one (quorum-based channel hopping, QCH) adopts a quorum system to design efficient port accessing strategy. The SPA algorithm can be used in very limited situations, but the QCH algorithm can be applied in many rendezvous settings.

For the asynchronous and port-asymmetric setting, we present the method of extending the QCH algorithm to two asynchronous users. The asynchronous QCH (A-QCH) algorithm designs special rendezvous sequences on the basis of two disjoint quorum systems and this method can also be applied in designing symmetric algorithms for the users. Another algorithm is called the Sequential Accessing Algorithm (SAA), which accesses the ports in a sequential manner. But this algorithm has limited extensions.

For the synchronous and port-asymmetric setting, we propose the Modified Sequential Accessing (MSA) algorithm which operates on the basis of the SAA algorithm. When one user adopts the MSA algorithm while the other one runs the SPA algorithm, they can rendezvous in a short time.

For the asynchronous and port-asymmetric setting, one user adopts the MSA algorithm while the other user adopts the Temporary Wait Algorithm (TWA), and they can achieve rendezvous quickly. The intuitive idea is one user keeps accessing the ports dynamically, while the other one moves slowly enough such that the first user can peruse all the ports during the "slow" moves of the other user.

From these rendezvous algorithms, the main idea in designing asymmetric algorithms is to make one user wait on a fixed port for a sufficient amount of time, while the other user keep accessing the ports dynamically. The described algorithms are listed in Table 6.1 and readers who are interested in asymmetric algorithms can design some other algorithms for rendezvous.

References

1. Bian, K., Park, J.-M., & Chen, R. (2009). A quorum-based framework for establishing control channels in dynamic spectrum access networks. In *Mobicom*.
2. Bian, K., Park, J.-M., & Chen, R. (2011). Control channel establishment in cognitive radio networks using channel hopping. *IEEE Journal on Selected Areas in Communications*, *29*(4), 689–703.
3. Bian, K., & Park, J.-M. (2013). Maximizing Rendezvous diversity in rendezvous protocols for decentralized cognitive radio networks. *IEEE Transactions on Mobile Computing*, *12*(7), 1294–1307.
4. Luk, W. S., & Wong, T. T. (1997). Two new quorum based algorithms for distributed mutual exclusion. In *ICDCS*.