# 國立交通大學

## 資訊科學與工程研究所

## 碩 士 論 文

以彈性 quorum 機制解決 IEEE 802.11 無線隨意網路的時間不同步問題

An Adaptive Quorum-based Mechanism for the Clock Asynchronism Problem in IEEE 802.11 MANETs

研 究 生：陳淑敏

指導教授：曾煜棋　教授

中 華 民 國　九 十 五　年 六 月

# An Adaptive Quorum-based Mechanism for the Clock Asynchronism Problem in IEEE 802.11 MANETs

Student: Shu-Min Chen
Advisor: Prof. Yu-Chee Tseng

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

July 2006

Hsinchu, Taiwan

# 以彈性quorum機制解決IEEE 802.11無線隨意網路的時間不同步問題

學生：陳淑敏　　指導教授：曾煜棋教授

國立交通大學資訊科學與工程學系（研究所）碩士班

## 摘　　要

對於IEEE 802.11無線隨意網路下的電力管理機制而言，節點間是否達到時間同步化扮演著非常關鍵的角色。目前已有許多針對IEEE 802.11無線網路的時間同步協定被提出來了，然而，這些協定卻不能保證可以成功地解決時間不同步問題。此外，即使存在著一個完美的時間同步協定，可以在多階的無線隨意網路下達到時間同步化，時間不同步問題仍然可能會因爲節點的移動而發生。因此，在這篇論文裡，我們提出了一套以quorum爲基礎的協定，這套協定將可用來加強時間同步協定在IEEE 802.11無線隨意網路下的時間不同步問題。根據模擬結果顯示，我們提出的協定將可以有效地改善IEEE 802.11無線隨意網路下的時間不同步問題。

**關鍵字**：無線隨意網路，電力管理機制，時間同步

# An Adaptive Quorum-based Mechanism for the Clock Asynchronism Problem in IEEE 802.11 MANETs

Student: Shu-Min Chen Advisor: Prof. Yu-Chee Tseng

Department of Computer Science
National Chiao-Tung University

## ABSTRACT

In wireless mobile ad hoc networks (MANETs), it is essential that all nodes are synchronized to a common clock for *power saving mechanism (PSM)*. Many protocols have been proposed to fulfill clock synchronization for IEEE 802.11 MANETs. However, all these protocols can not guarantee that they can completely solve the asynchronism problem in a highly mobile ad hoc network. Besides, even if there exists a perfect clock synchronization protocol which can guarantee that it can fulfill clock synchronization in a multi-hop network, the asynchronism problem may still arise because of mobility. Therefore, in this work, we propose a quorum-based mechanism to assist the existing clock synchronization protocols in solving the clock asynchronism problem in IEEE 802.11 MANETs. Our simulation results show that our proposed protocol can effectively improve the clock asynchronism problem for highly mobile IEEE 802.11 MANETs.

**Keywords:** Mobile Ad Hoc Network, Power Saving Mechanism, Clock Synchronization.

# Acknowledgements

My advisor, prof. Yu-Chee Tseng, is the first one I would like to express my gratitude to. With the wonderful research conditions he provided and his attentive instructions, I came to discover the pleasure of research. I am also grateful to my senior, Sheng-Po Kuo. Without his help and suggestions, I would not be able to have this thesis done. In addition, I would like to thank all HSCC members for their generous advice. Discussing with them benefited me in many aspects. Finally, thanks my parents and my boyfriend, Liang-Chieh Chen, for their support.

Hsiao-Ju at CS, NCTU.

# Contents

# List of Figures

# Chapter 1

# Introduction

In wireless mobile ad hoc networks (MANETs), it is essential that all nodes are synchronized to a common clock for *power saving mechanism (PSM)*. In IEEE 802.11 PSM, each node wakes up at the beginning of the beacon interval to exchange messages. Through message exchanging, nodes can schedule packet transmissions for the current beacon interval. If a node does not have any packet transmission scheduled, it can switch its radio transceiver off and go to the sleep mode for the rest of the beacon interval. In order to exchange messages properly, node's message exchanging period should be synchronized. Otherwise, PSM will not function well.

To fulfill the requirement of clock synchronization, IEEE 802.11 standards specify a distributed *Timing Synchronization Function (TSF)* for the ad hoc mode. In this mechanism, since there is no per-designed infrastructure in MANETs, each mobile node will compete to broadcast its timing information through *beacon* frames. The TSF mechanism is quite enough for a small and static ad hoc network. In [1], Huang and Lai first discover the scalability problem of IEEE 802.11 TSF. When the number of network nodes is increasing, beacon contention may get more serious and thereby cause the clock asynchronism problem. To alleviate the scalability problem, a simple protocol called ATSP was proposed in [1]. The basic idea of ATSP is to give the fastest node the highest priority to send beacons. However, ATSP can not handle the issues of scalability and mobility very well at the same time. Therefore, during the past few years, several protocols have been proposed in [2, 3]. TATSF, proposed in [2], classifies nodes into multiple tiers with different beacon contention frequencies. Another protocol called SATSF [3] allows the fastest node(s) to compete every beacon interval and inhibits slower nodes from beacon contention. Besides, in SATSF, a slower node will self-adjust its clock frequency to be closer to that of a faster node. These protocols can effectively solve the asynchronism caused by beacon contention, but they are designed for fully-connected networks, not for

MANETs.

For a multi-hop ad hoc network, it is much more difficult to achieve time synchronization. Several clock synchronization protocols for multi-hop MANETs have been proposed such as *Automatic Self-time-correcting Procedure (ASP)* [4], *Multi-Hop Adaptive Timing Synchronization Function (MATSF)* [5], and *Multi-Hop Timing Synchronization Function (MTSF)* [6]. The basic idea of ASP is to increase the probability of successful beacon transmissions for faster nodes, and slower nodes can *self-correct* its timer automatically after collecting enough timing information from faster nodes. MATSF adopts the similar self-correcting idea. Besides, in order to spread the faster timing information throughout the whole network quickly, MATSF groups the faster nodes into a dominating set. The nodes in the dominating set will compete for beacon transmission every beacon interval, while the rest of nodes compete only once in a while. The self-correcting mechanism of ASP and MATSF has poor backward compatibility because it changes the beacon format for an additional sequence number. As for MTSF, its basic idea is to create a spanning tree rooted at the fastest station. Each node selects the fastest neighbor as its ”parent”, and schedules its beacon transmission non-overlapping with its parent. Although MTSF does not need to change the beacon format, its major problem is that all nodes' beacon transmission schedules rely heavily on the spanning tree. Hence, it is not suitable for mobile nodes in a MANET.

The main goal of the above mentioned protocols is to minimize the maximum clock drift of the network. However, they can not guarantee that they can successfully achieve multi-hop clock synchronization all the time. Once if some nodes are *asynchronous* with their neighbors, i.e., their wake-up schedules do not overlap with each other, these protocols can not reduce the clock drift and re-synchronize nodes. Furthermore, nodes' mobility may aggravate the clock asynchronism problem in MANETs. Suppose that two groups of nodes at a distance have non-overlapping wake-up schedules. When they are approaching each other, the mentioned synchronization protocols can not do anything to discover mutually.

In this paper, we propose a quorum-based mechanism to relieve the clock asynchronism situation in IEEE 802.11 MANETs. Based on the concept of quorum, we divide the beacon intervals into two types: *quorum* and *non-quorum* intervals. In the quorum interval, each node remains awake for the entire beacon interval, and contends to send its beacon frame even if it has received one. In the non-quorum interval, it can enter the sleep mode after the ATIM window if it does not have any communication schedule as usual. According to the characteristics of a quorum, asynchronous neighbors can discover each other and then get their clocks synchronized.

# Chapter 2

# Problem Definition and Backgrounds

## 2.1 Problem Definition

We say that the *clock asynchronism problem* occurs when the time difference between any two neighboring nodes is larger than the length of one beacon window. In this case, the faster node's beacon window will not overlap with the slower node's ATIM window (refer to Fig. 2.1(a)). So the faster node can hear the beacon sent from the slower node but the slower node may not have a chance to hear the faster node's beacons and thus to get synchronized with the faster node. The clock asynchronism problem may remain unsolved until the amount of time drift between the faster and the slower nodes is a multiple of one beacon interval (refer to Fig. 2.1(b)). To see how serious the clock asynchronism problem is, suppose that node $A$ is faster than node $B$ by $20\mu s$ per beacon interval in Fig. 2.1. Assuming that the lengths of a beacon interval, an ATIM window, and a beacon window are $100000\mu s$, $16000\mu s$, and $1240\mu s$, respectively (based on the IEEE 802.11 DSSS specification), it will take $\frac{100000-1240-16000+1240}{20} = 4200$ beacon intervals ($= 420$ seconds) to move from the scenario in Fig. 2.1(a) to the scenario in Fig. 2.1(b).

Until now, most existing clock synchronization protocols aim at minimizing the maximum clock drift among nodes in a connected and initially synchronized network. It lacks a mechanism to get nodes synchronized when their clocks seriously drift away. Generally speaking, the assumption that nodes are initially synchronized is not true either. If nodes are not synchronized first, most clock synchronization protocols may not function well, until their beacon windows meet with each other.

Nodes' mobility will even aggravate the clock asynchronism problem in a MANET. Since most clock synchronization protocols will try to keep the clock drifts among nodes within a small bound, it is very likely that two partitions of a MANET will have non-overlapping beacon windows and thus lose synchronism. For example, Fig. 2.2(a) illus-

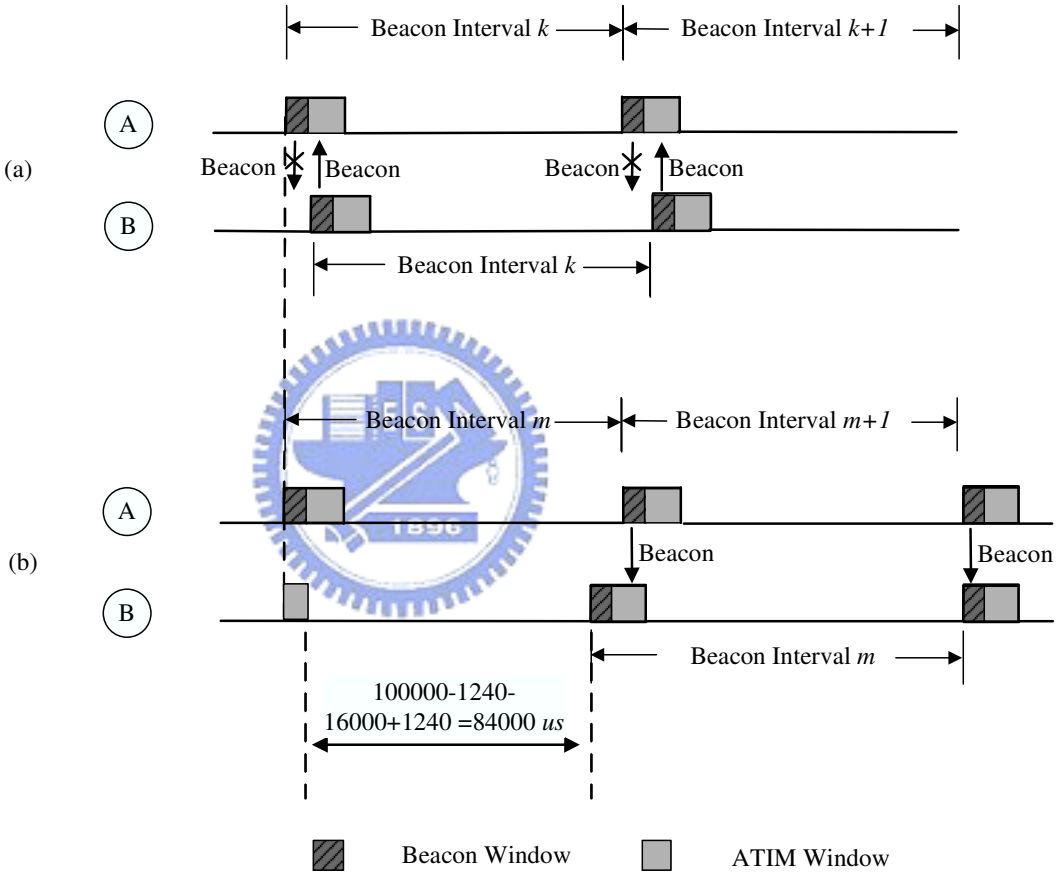



Figure 2.1: (a) An example of the clock asynchronism problem, and (b) resynchronization after $B$ catches up with $A$'s beacons.

trates a situation where two groups of mobile nodes are each perfectly clock synchronized but are disconnected initially. When these two groups merge into one, as shown in Fig. 2.2(b), nodes in these two groups may not be able to discover each other, and thus the network disconnectivity problem may remain unsolved for a while. This is clearly harmful, especially when the MANET is highly mobile.

Even if a MANET remains connected, the clock asynchronism problem may cause the network to lose some communication links. For example, in a large-scale MANET, where a good clock synchronization protocol is running behind, a small clock drift in each hop may accumulate into a large amount of drift after multiple hops. As shown in Fig. 2.3, if neighboring nodes' clock drift is bounded by $\frac{1}{5}$ of one beacon window, nodes $A$ and $H$, which are separated by $5$ hops, may still remain out-of-synchronization. Therefore, when node $F$ moves to node $A$'s communication range, the communication link between $A$ and $F$ may not be discovered for some while, making the network layer mistakenly interpret that $A$ and $F$ are quite far away.

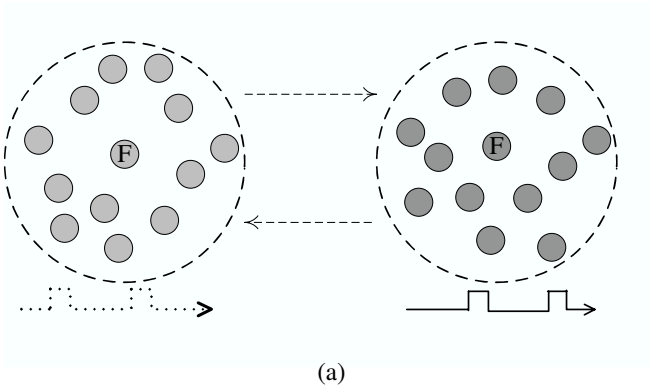
To summarize, most clock synchronization protocols aim at reducing the clock drift among hosts. Even with such protocols, the clock asynchronism problem can not be completely avoided, especially in a highly dynamic MANET. Our goal in this work is to design an exception handling mechanism as an enhancement to existing clock synchronization protocols to quickly discover asynchronous neighbors that cannot be found by typical clock synchronization protocols.

## 2.2 Reviews

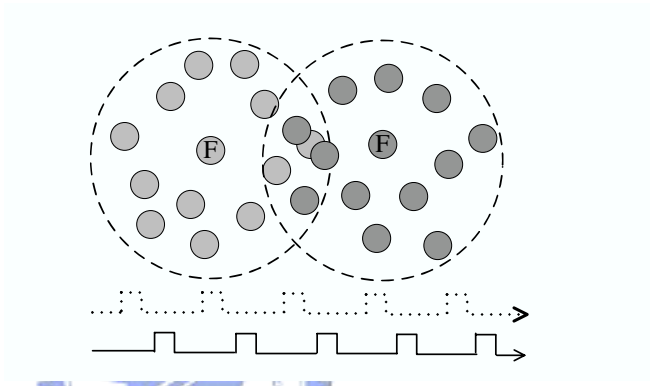
In this section, we review the power saving mechanism specified in the IEEE 802.11 standard first. Then we present the IEEE 802.11 *Timing Synchronization Function (TSF)* and the *Adaptive Timing Synchronization Procedure (ATSP)* proposed in [1]. In addition, we also present a multi-hop clock synchronization protocol called *Automatic self-time-collecting Procedure (ASP)* proposed in [4].

### 2.2.1 The Power Saving Mechanism of IEEE 802.11 Ad Hoc Networks

In the IEEE 802.11 ad hoc mode, stations cooperate to support the power saving mechanism since there is no infrastructure. Active stations will buffer packets for those stations in the sleep mode and try to notify them for data transmission. Sleeping stations will also wake up periodically to listen to the possible notification messages. The notification

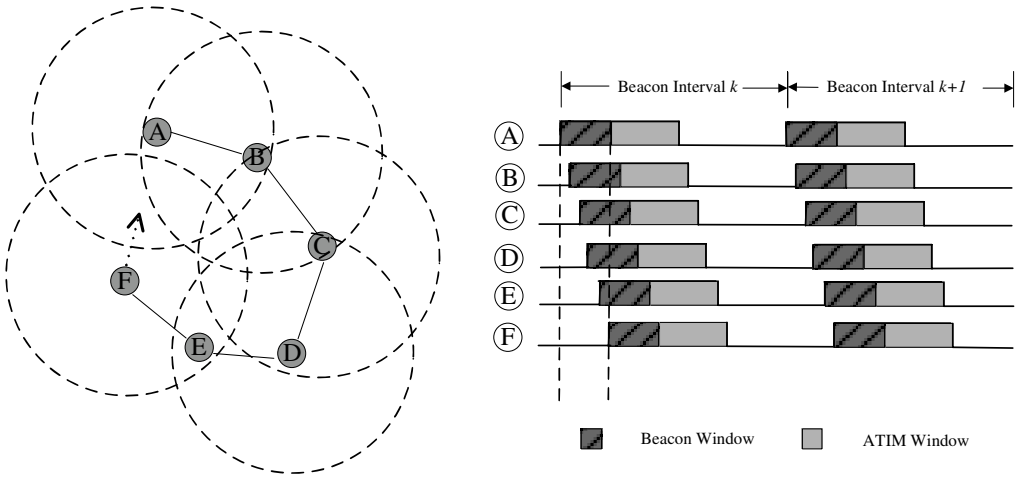



(a)




(b)

Figure 2.2: An example where clock asynchronism occurs when two disconnected components of a MANET meet each other.




(a) (b)

Figure 2.3: An example where clock asynchronism causes the network to lose the communication between nodes $A$ and $F$.

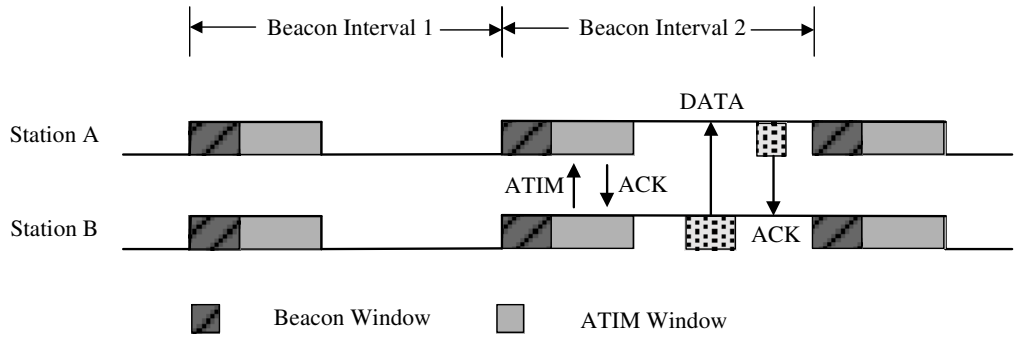



Figure 2.4: Power Management in IEEE 802.11 Ad Hoc Networks.

messages are called *Ad hoc Traffic Indication Messages (ATIM)*. Stations can exchange ATIMs only during the ATIM window, a time window with a fixed length. During ATIM windows, all stations should stay active to receive possible ATIMs. Stations which have buffered packets for other stations compete to send ATIMs. Those stations that receive the ATIMs will reply an ACK packet to the sender and keep active in the rest of the beacon interval. An example is shown in Fig. 2.4. In the first beacon interval, both stations $A$ and $B$ can go the the sleep mode since they do not receive any ATIM frame. In the second beacon interval, however, station $A$ has to stay active after the ATIM window since it received an ATIM frame from station $B$. Then $A$ and $B$ can exchange DATA and ACK frames after the ATIM window.

In the IEEE 802.11 standards, a distributed timing synchronization function (TSF) is proposed for power saving mechanism. In TSF, each node shall maintain a local TSF timer with modulus $2^{64}$ counting in increments of microseconds ($\mu s$). The value of the TSF timer is the summation of a variable *offset* and the node's clock. Clock synchronization is achieved by periodically exchanging timing information through beacon frames. A beacon frame contains a timestamp declaring when the beacon was sent. After a node receives a beacon frame and finds that its own TSF timer is slower than the timestamp specified in that beacon, it will add the timing difference to its offset.

In order to periodically exchange timing information for clock synchronization, all nodes adopt a common value $aBeaconPeriod$ which defines the length of a beacon interval. Based on the value of $aBeaconPeriod$, each node divides their time into a series of beacon intervals which are exactly $aBeaconPeriod$ time units apart. At the beginning of each beacon interval, each node participates in beacon generation process as follows.

1. Calculate a random delay uniformly distributed in the range between zero and $2 \cdot aCWmin \cdot aSlotTime$. (The values of $aCWmin$ and $aSlotTime$ are 15 and

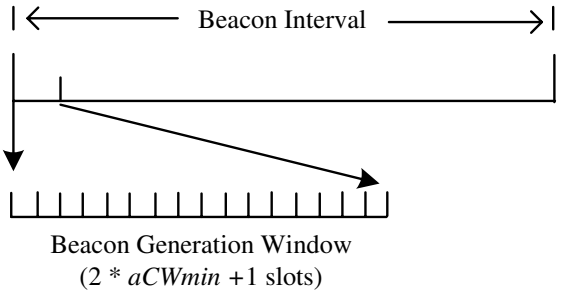



Figure 2.5: Beacon generation window.

50 $\mu s$ for *Frequency Hopping Spread Specturm (FHSS)* and are 31 and 20 $\mu s$ for *Directed Sequence Spread Spectrum (DSSS)*).

2. Wait until the random delay timer expires.

3. If a beacon is received before the random delay timer has expired, the node cancels the random delay timer. Otherwise, when the random delay timer expires and no beacon has arrived during the delay period, the node shall transmit a beacon with its TSF timing information.

4. Upon receiving a beacon, the node sets its TSF timer to the timestamp of the beacon if the timestamp is later than the node's TSF timer.

As illustrated in Fig. 2.5, at the beginning of each beacon interval, the period when nodes compete to send their beacons is defined as *beacon generation window* consisting of $2 \times aCWmin + 1$ time slots (each of length $aSlotTime$). Each node is randomly scheduled to transmit a beacon at the beginning of one of these slots.

### 2.2.2 Adaptive Timing Synchronization Procedure

In the IEEE 802.11 ad hoc networks, because nodes can only set their timers forward, the node with the fastest clock may suffer from asynchronism with a high probability if it fails to transmit beacons for too many beacon intervals. To alleviate these asynchronism problems, an *Adaptive Timing Synchronization Procedure (ATSP)* is proposed in [1]. The main idea of ATSP is to give faster stations a higher priority to send beacons. Based on this idea, each node $i$ is assigned a parameter $I(i)$ to determine how often it should participate in beacon contention. Node $i$ contends for beacon transmission every $I(i)$ beacon intervals. Let $I_{max}$ be the maximum value of $I(i)$ and $C(i)$ be the counter in node $i$ that counts the number of beacon intervals. Initially, $I(i)$ is randomly generated between 1 and $I_{max}$ and $C(i) = 1$. In each beacon interval, node $i$ participates in beacon

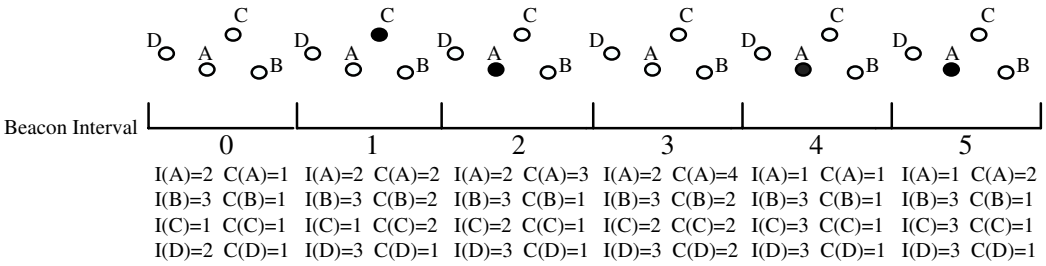



Figure 2.6: An example of the ATSP algorithm. The black node (if any) in each beacon interval is the node which wins the beacon transmission in that interval.

contention iff $C(i)\ mod\ I(i) = 0$. When node $i$ receives a beacon with a faster timestamp than its own one, its timer will be set to the received timestamp. Then, its priority will be decreased through increasing $I(i)$ by 1 if $I(i)$ is less than $I_{max}$, and its counter $C(i)$ will be reset to 0. On the other hand, if node $i$ does not receive any faster timestamp than its own for $I_{max}$ consecutive beacon intervals, it decreases $I(i)$ by 1 if $I(i)$ is larger than 1 and sets $C(i)$ to 0. At the end of each beacon interval, each node increases its $C(i)$ by 1.

Consider the example in Fig. 2.6, where $I_{max} = 3$. The order of clock speed is $A > B > C > D$. Initially, the value of timer is $A > B > C > D$, and $I(A) = 2, I(B) = 3$, $I(C) = 1$, and $I(D) = 2$. At the beacon interval 1, since $C(C)\ mod\ I(C) = 0$, only $C$ participates in beacon contention, and thereby sends its beacon. This causes $D$ to increase its $I(D)$ by 1 and set $C(D)$ to 0. At the end of this beacon interval, every node increases its own counter $C(i)$ by 1. At the beacon interval 2, both $A$ and $C$ participate in beacon contention. In this example, $A$ sends its beacon first, resulting in $I(C) + 1$, and $C(B) = C(C) = C(D) = 0$. Note that $I(B)$ and $I(D)$ can not be increased since they are equal to $I_{max}$. At the beacon interval 3, according to the rules of ATSP, no node contends to send their beacons. At the beacon interval 4, note that since $I(A)$ has remained unchanged value for $3(= I_{max})$ continuous beacon intervals, $A$ will decrease $I(A)$ by 1, and sets $C(A)$ to 0. Finally, the fastest node $A$ will participate in the beacon contention in every beacon interval.

With ATSP, the node with the fastest TSF timer will have a very high probability of successfully sending its beacons, thereby synchronizing all other stations. Compared to the IEEE 802.11 TSF, ATSP provides a simple but effective solution to improve the clock synchronization mechanism in single-hop MANETs.

### 2.2.3 Automatic Self-time-collecting Procedure for MANETs

The main ideas of ASP are to increase the successful beacon transmission probability for faster nodes and to spread the faster timing information throughout the whole network. In ASP, each node maintains a table called $Neighbor\_Table$ to keep track of its neighbors and their TSF timers through periodical beacon transmissions. Based on $Neighbor\_Table$, node $i$ first calculates an integer variables $p_i$, which is defined as the period of how many beacon intervals for node $i$ to compete to transmit a beacon. For example, if $p_i = 5$, node $i$ will try to transmit a beacon every 5 beacon intervals. $p_i$ is calculated as follows.

$$p_i = \left\lfloor \left( \frac{max(1, NA_i)}{max(1, NL_i)} \right)^{\alpha} \right\rfloor, \alpha \in N$$

where $NA_i$ is the number of node $i$'s neighbors and $NL_i$ is the number of node $i$'s neighbors whose TSF timer is equal to or slower than that of node $i$. The parameter $\alpha$ is used to adjust the number of nodes to contend for the beacon transmission. For example, for node $i$, when $NA_i = 6$ and $\alpha = 1$, $p_i$ will be one only when $NL_i$ is more than three, while in the same case, but if $\alpha = 2$, $p_i$ will be one only when $NL_i$ is more than four. This means that a large $\alpha$ reduces the number of nodes that can transmit their beacons in every beacon interval.

Unlike 802.11, each node changes its *offset* only when it has received a beacon containing a timestamp later than its local timer. In ASP, a slower node not only follows this standard rule, but also tries to obtain the clock oscillation difference between itself and a faster node. With this information, a slower node can update its *offset* periodically in order to synchronize to the faster nodes automatically. A slower node obtains the oscillation difference by comparing the difference of its TSF timers with the successively received beacons containing a faster timestamp from the same node. We define $Pass\_Time1$ as the elapsed time that a node receives two successive beacons from the same faster node and $Pass\_Time2$ as the time difference between these two beacons' timestamps. Based on $Pass\_Time1$ and $Pass\_Time2$, a slower node $i$ calculates an integer variable $a_i$ as follows.

$$a_i = \left\lfloor \frac{Pass\_Time1}{Pass\_Time2 - Pass\_Time1} \right\rfloor$$

Then, based on the value of $a_i$, node $i$ can automatically increase its *offset* by one in every $a_i$ microseconds. Note that it is important that the beacon sender (i.e., the faster node) did not change its *offset* during the two successive beacon transmissions. Otherwise,

the beacon receiver (i.e., the slower node) will estimate its clock oscillation to the beacon sender improperly. In order to solve the problem, each node maintains an integer variable $Seq\_No$ and appends the value of $Seq\_No$ at the beacon frames (here, ASP adds a 4-bit field for $Seq\_No$ in the beacon frames. So, $Seq\_No$ will be reset to zero when its value is increased to sixteen). When a node's *offset* is updated because of a faster beacon, the node's $Seq\_No$ will be increased by one. With the design of $Seq\_No$, the correct calculation of $Pass\_Time2$ shall be taken from two beacons sent from the same node with the same $Seq\_No$. Note that due to a node may update its $Seq\_No$ more than once in a beacon interval, the received timing information stored in each node will be abandoned after $k$ beacon intervals (here, ASP sets $k$ to eight) in order to prevent the wraparound problem of $Seq\_No$.

To sum up, ASP makes nodes with faster TSF timers transmit their beacon frames frequently. In addition, a slower node can automatically synchronize to a faster node if it has received the beacon with the faster timing information twice from the same node with the same $Seq\_No$.

# Chapter 3

# The Proposed Quorum-Based Mechanism

## 3.1 Concept of Quorum

We apply the concept of quorum to help asynchronous nodes discover each other. Quorums have been widely used in distributed systems (e.g., to guarantee mutual exclusion or fault tolerance [7, 8]). A quorum is a set of entities from which one has to obtain permission to perform some critical action. Any two quorum sets must have non-empty intersections. This property helps us to design synchronization schemes such that any two asynchronous nodes have chances to receive each other's beacons.

In [9], it has been shown that a group of quorums, such as the *grid quorum* [10], the *torus quorum* [11], the *cyclic quorum* [12], and the *finite projective quorum* [10], can be applied to efficient synchronization protocols.

## 3.2 Structure of Beacon Intervals

Below, we will use the *grid quorum* to explain our *Quorum-based Clock Synchronization (QCS)* protocol. A grid quorum is formed by a 2D matrix such that each quorum contains a random column and a random row of entries. Clearly, any two quorums must have a non-empty intersection. Here, an $N \times N$ grid quorum will be used. Each node will partition its beacon intervals into groups such that $N^2$ beacon intervals constitute one group. In each group, its $N^2$ beacon intervals are arranged in a row-major manner into an $N \times N$ grid. Each node then arbitrarily picks one row and one column and designates these $2N - 1$ beacon intervals in the selected column and row as *quorum intervals*, and the remaining $N^2 - 2N + 1$ beacon intervals as *non-quorum intervals*.

Nodes' actions in quorum and non-quorum intervals are defined as follows.

- Each quorum interval includes three parts: *quorum beacon window*, *ATIM window*, and *data window*. During a quorum beacon window, a node will try to send out its beacons. A beacon should be sent even if the node has received other nodes' beacons. During an ATIM window, a node may send/receive traffic announcement to/from other nodes. After the ATIM window, a node has to stay awake throughout the whole data window.

- Each non-quorum interval also consists of three parts: *non-quorum beacon window*, *ATIM window*, and *data window*. In a non-quorum beacon window, beacon transmission is optional and will depend on an existing clock synchronization protocol. The clock synchronization protocol can be anyone mentioned in the Chapter 2.2. Here, we will use a binary function $f_n(i)$ to denote whether node $n$ will compete for the beacon transmission in the $i$-th beacon interval or not. The value of $f_n(i)$ depends on the adopted clock synchronization protocol. We will leave the detail discussion in the next section. A node's behavior in an ATIM window is the same as that in a quorum interval. However, a node's behavior in a data window will depend on whether there is any traffic announcement in the former ATIM window related to itself. If so, it has to remain awake throughout the data window; otherwise, it may go to sleep.

Fig. 3.1 illustrates an example of grid quorum and the structures of quorum and non-quorum intervals. With such a structure, it has been proven in [9] that two neighboring nodes can always hear each other's beacons at least once every $N^2$ beacon intervals, no matter how much their clock values drift away. The value of $N$ also plays an important role in the QCS protocol to balance the performance of time synchronization and power consumption. A smaller $N$ can reduce the time required to synchronize two asynchronous nodes when they become neighbors but at the cost that it has to stay awake in more data windows when quorum intervals appear. In the Chapter 4, we will investigate this issue under different values of $N$.

## 3.3 Beacon Transmission Rules

In the QCS protocol, beacon trasmission is mandatory in a quorum interval. However, in a non-quorum interval, beacon transmission is controlled by two processes: the *beacon-reception* process and the *beacon-window* process. These two processes are illustrated in Fig. 3.2.

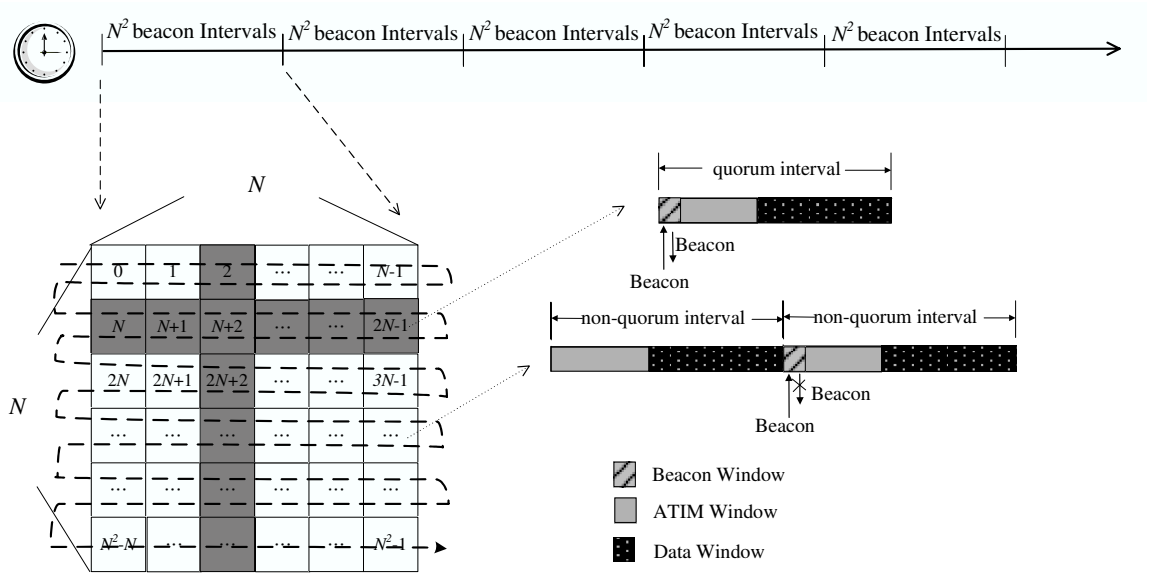



Figure 3.1: The structures of grid quorum, quorum interval, and non-quorum interval.

The beacon-reception process is triggered when a node $i$ receives a beacon from a node $j$. Let $T_i$ be the current time of $i$, $T_j$ be the timestamp in $j$'s beacon, $BW$ be the length of a beacon window, and $N_i$ be the current number of node $i$'s neighbors. Two types of events are considered to be monitored:

- Event A: If node $i$'s clock is faster than node $j$'s clock by the length of a beacon window (i.e., $T_i > T_j + BW$), event $A$ will be triggered.
- Event B: If node $i$'s clock is slower than node $j$'s clock by the length of a beacon window (i.e., $T_i < T_j - BW$) and node $i$ has at least one neighboring node (i.e., $N_i > 0$), event $B$ will be triggered.

These events are apparent to show that the node who receives the beacon has some out-of-synchronization neighboring nodes and it is capable of synchronizing their timer. Hence, after being aware that one of these events has taken place, the node should compete to send its beacons in the following beacon intervals for a short period of time so as to synchronize these neighbors. Fig. 3.2(a) shows the flow of the beacon-reception process. The parameter $N_A$ is defined as the number of beacon intervals where a node should continuously compete to send its beacons when event A is triggered, and $N_B$ is for event B. A counter $Cnt$ is used to denote the remaining beacon intervals during which a node should compete to send its beacons. In order to meet the quorum intervals of those out-of-synchronization neighbors, the system parameters $N_A$ and $N_B$ should be larger than

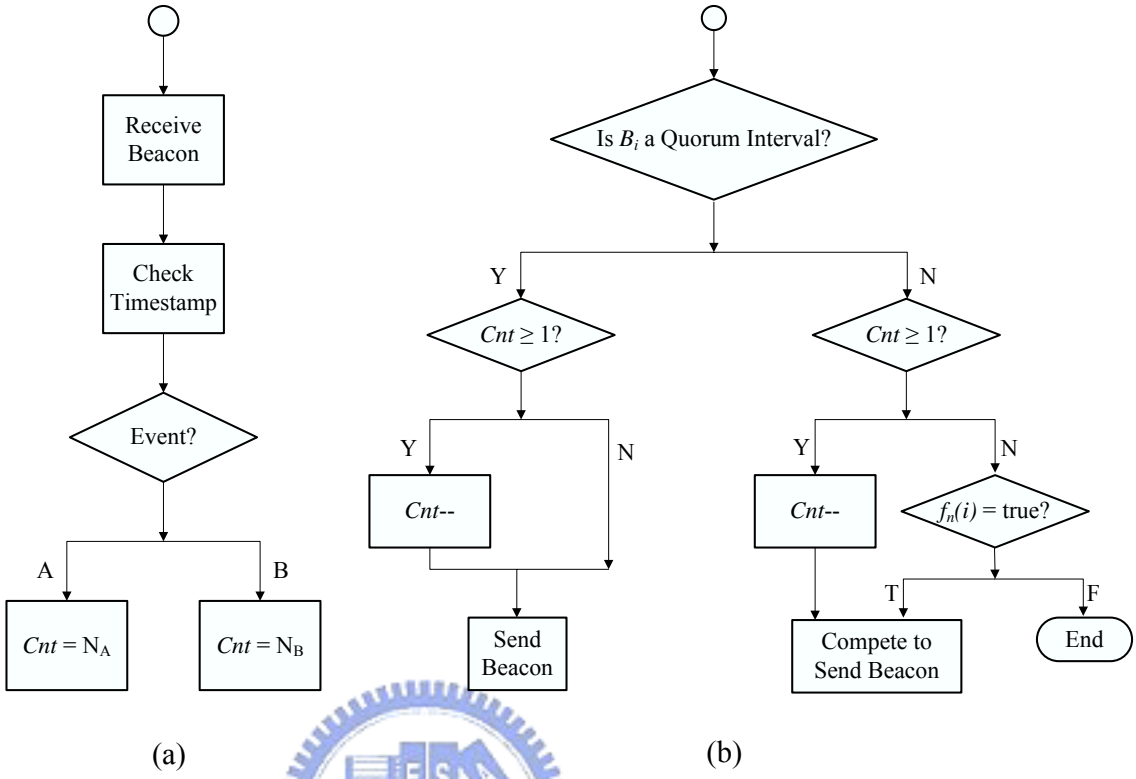



Figure 3.2: The flowchart of (a) the beacon-reception process and (b) beacon-window process.

the quorum size $N$ because a quorum interval at least appears once for every $N$ beacon intervals.

The beacon-window process is triggered by the current type of beacon interval (refer to Fig. 3.2(b)). For each coming beacon interval $B_i$, we need to identify its type first. If $B_i$ is a quorum interval, the node has to send a beacon during its beacon window even if it has already received others' beacons. This is because QCS protocol relies on the beacon exchange during the overlapping quorum intervals of two asynchronous nodes to adjust the slower's timer. If we only follow the beacon transmission rule of the cooperated synchronization protocol, we may loss the benefit of the quorum system. If $B_i$ is a non-quorum interval, the node should check its counter $Cnt$ first. If $Cnt$ is set because the event A or B has been triggered, the node has to compete to send its beacon and decrease $Cnt$ by one until $Cnt$ counts down to zero. When $B_i$ is a non-quorum interval and $Cnt = 0$, we follow the rules of the cooperated synchronization protocol. In this case, the function $f_n(i)$ dominates the beacon contention procedure to determine if the node $n$ should compete to send a beacon during this beacon interval.

Fig. 3.3 illustrates the advantage of reducing a node's beacon competition frequency for a period of time when the event A is triggered. In this example, node $P$ with a faster timer can be aware of the existence of an out-of-synchronization neighboring node $Q$ by receiving from $Q$'s beacon during $P$'s ATIM window. Hence, node $Q$ will have a chance to be synchronized to $P$'s timer by the continuous beacon transmissions of node $P$. Fig. 3.4 shows the situation of event B. At first, node $B$ has a synchronized neighboring node $C$ and receives a beacon with a faster timestamp from node $A$. Then, node $B$ gets synchronized to node $A$ and thereby loses the connection with node $C$ because of the asynchronism problem. In this case, node $B$ should try to compete to send its beacons in the following beacon intervals in order to synchronize node $C$.

As shown in Fig. 3.5, it demonstrates how our QCS protocol cooperates with ATSP. In this figure, node $A$'s timer is faster than node $B$'s by at least a beacon window. The variables $C$ and $I$ are the basic parameters of ATSP (refer to Chapter 2). So $f_n(i)$ of node $n$ in QCS protocol is set to $true$ when $C(n)\ mod\ I(n) = 0$ during the beacon interval $i$. Assume $I_{max} = 10$. Initially, $C(A) = 4$, $I(A) = 2$, $C(B) = 1$, $I(B) = 5$, and the counters $Cnt$ of A and B are both zero in the beacon interval $k$. Although node $A$ competes to send beacons in every two beacon intervals, node $B$ misses these beacon transmissions because they are encountering the clock asynchronism problem. During the beacon interval $k + 3$, node $A$ meets a quorum interval resulting that $C(A)$ is reset to zero because node $A$ has sent its beacon during that beacon interval. During the beacon interval $k + 4$, since node $A$ receives node $B$'s beacon, event A is triggered. So node $A$ sets $Cnt = N_A$ and competes to send its beacon every beacon interval until its $Cnt$ is decreased to zero. During the beacon interval $k + 5$, node $B$ meets a quorum interval and thereby receives node $A$'s beacon through its data window. As a result, node $B$ gets synchronized to node $A$, increases $I(B)$ by one, and sets $C(B)$ to zero.

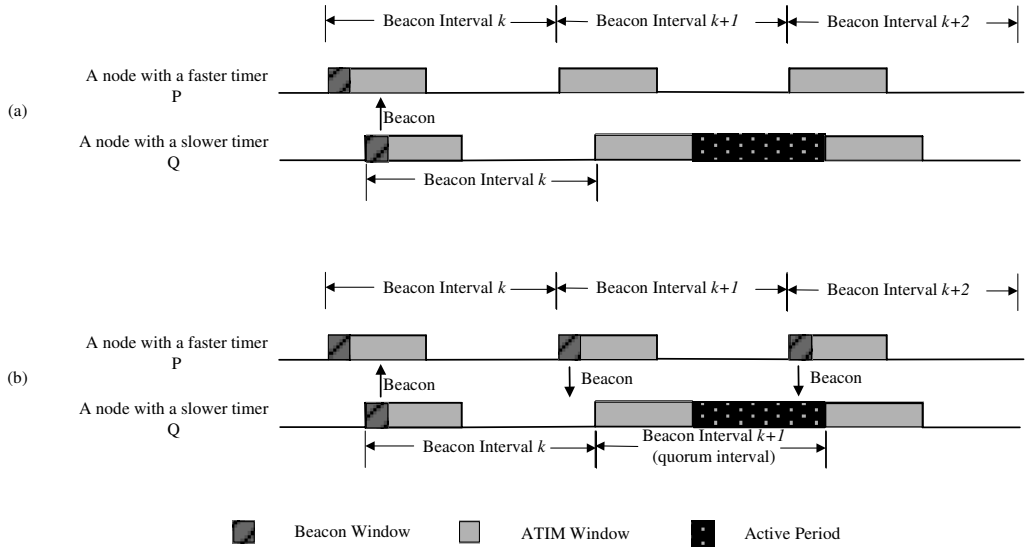



Figure 3.3: An example shows event A that a node should temporarily reduce its beacon competition frequency to one.

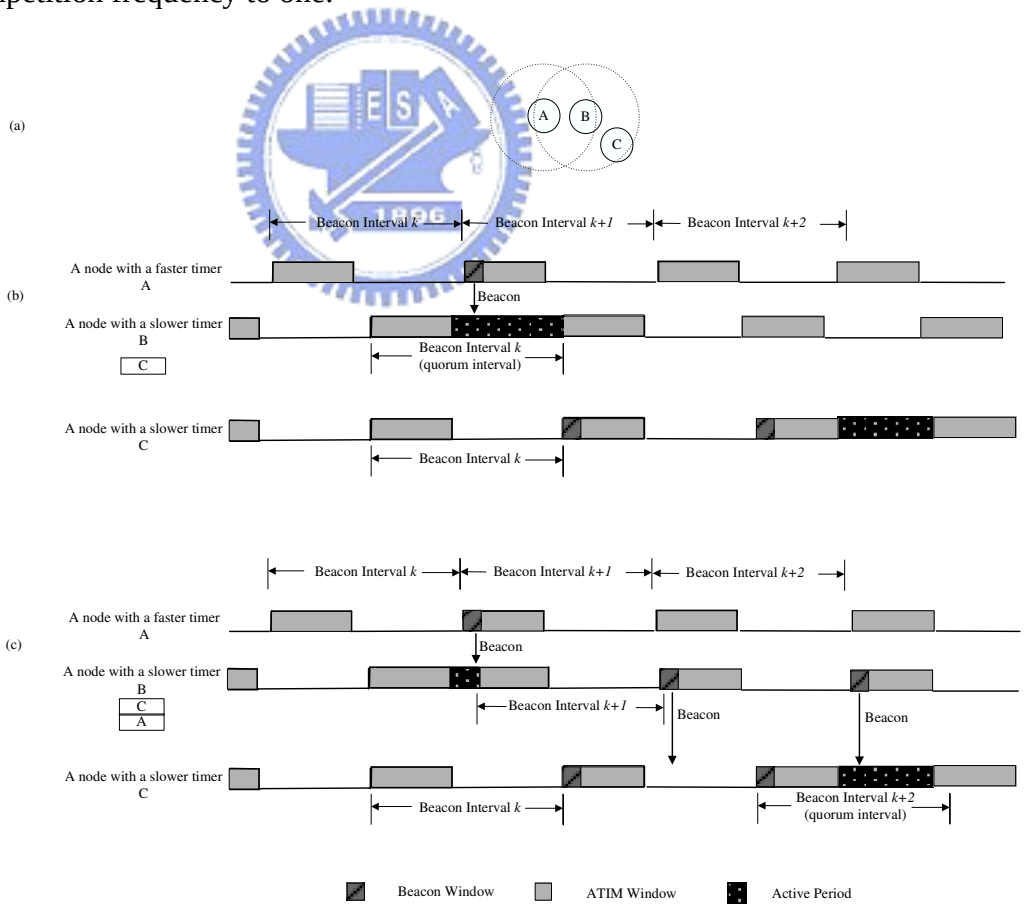
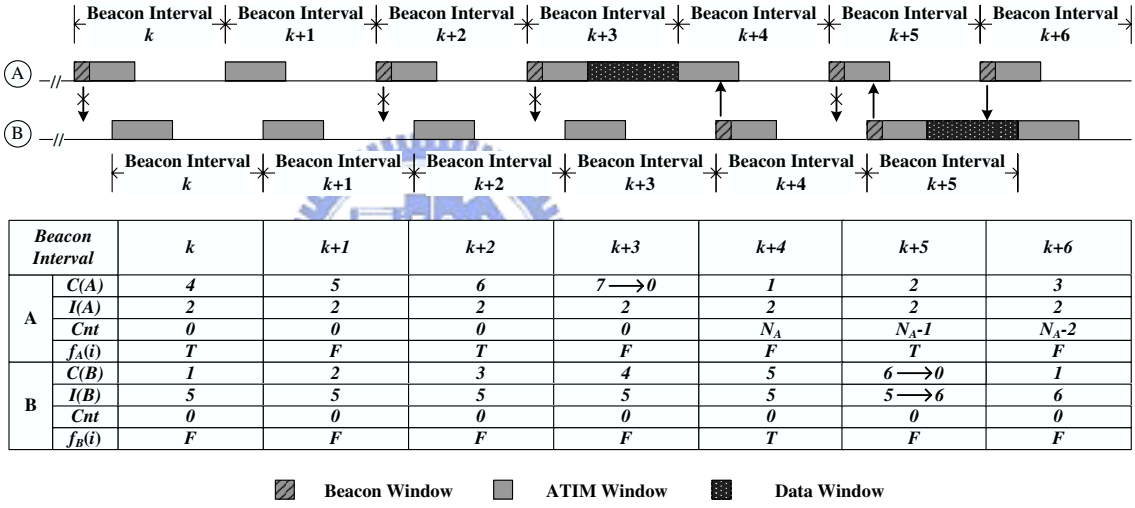



Figure 3.4: An example illustrates event B that a node should temporarily reduce its beacon competition frequency to one.




| *Beacon Interval* | | *k* | *k+1* | *k+2* | *k+3* | *k+4* | *k+5* | *k+6* |
|---|---|---|---|---|---|---|---|---|
| **A** | *C(A)* | *4* | *5* | *6* | *7 ⟶ 0* | *1* | *2* | *3* |
| | *I(A)* | *2* | *2* | *2* | *2* | *2* | *2* | *2* |
| | *Cnt* | *0* | *0* | *0* | *0* | $N_A$ | $N_A$*-1* | $N_A$*-2* |
| | $f_A(i)$ | *T* | *F* | *T* | *F* | *F* | *T* | *F* |
| **B** | *C(B)* | *1* | *2* | *3* | *4* | *5* | *6 ⟶ 0* | *1* |
| | *I(B)* | *5* | *5* | *5* | *5* | *5* | *5 ⟶ 6* | *6* |
| | *Cnt* | *0* | *0* | *0* | *0* | *0* | *0* | *0* |
| | $f_B(i)$ | *F* | *F* | *F* | *F* | *T* | *F* | *F* |

Figure 3.5: An example where ATSP is adopted in our QCS protocol.

# Chapter 4

# Simulations

To evaluate the performance of our QCS protocol, we developed the programs in JAVA. We simulated QCS adopted by IEEE 802.11 TSF and ASP. In our simulations, we use two metrics to evaluate the proposed protocol.

- Average remaining time of asynchronous pairs of nodes
  The metric measures the average remaining time of asynchronous pairs of nodes throughout the entire simulation period. We use this metric as a measurement of the gravity of the clock asynchronism problem - the greater the value of the metric, the graver the clock asynchronism problem.

- Beacon sending times
  Since nodes have to compete to send their beacon frames in the quorum intervals and beacon transmissions will consume nodes' power resources, this metric could be used to measure if too much additional beacon transmissions occur when the existing clock synchronization protocols adopt the QCS protocol as an enhancement.

## 4.1 Simulation Setup

Our simulator closely follows the protocol details of beacon generation and contention specified in IEEE 802.11 standards. The value of $aBeaconPeriod$ (the length of a beacon interval) is set to $0.1s$ and the value of $aATIMWindow$ ( the length of an ATIM window) is set to $16ms$. Each node's clock accuracy is uniformly distributed in the range of [-0.01%,+0.01%] as recommended in IEEE 802.11 standards. So starting from the synchronized clocks, the maximum clock drift between two nodes after 1 second is 200$\mu s$.

In these simulations, we use the definition of the clock asynchronism problem specified in Chapter 2. That is, the asynchronism problem happens when a node's TSF timer

is slower than another neighboring node's TSF timer over 1240$\mu s$, i.e., the length of the beacon window when DSSS is adopted. Each simulation is performed for a duration of 500 seconds in a MANET where the number of nodes is 500. Each node is randomly located in an area of 3000×3000 square meters with a transmission range of 250 meters. All nodes' initial TSF timers are synchronized to zero and they move according to the random way-point model as in [4] with maximum speed $5m/s$ and pause time 20 seconds. We run ASP with $\alpha = 3$ as it is the preferred value by [4].

## 4.2 Simulation Results

Now we present and discuss our simulation results. As mentioned before, we evaluate our QCS protocol when it is adopted by IEEE 802.11 TSF and ASP as an enhancement.

First of all, we plot the maximum clock drift between any two neighboring nodes during the whole simulation time when IEEE 802.11 TSF is adopted to fulfill clock synchronization in Fig. 4.1. Besides, IEEE 802.11 TSF adopting QCS, where $N$ is set to 32, denoted by QCS(32), is also shown in this figure. As the figure shows, the maximum clock drift suddenly increases over 1240$\mu s$ around 80 seconds after the simulation begins. In this case, the first asynchronous pair appears because a node with a slower TSF timer suddenly moves inside another node's transmission range. Since the original IEEE 802.11 TSF does not address the issue of clock asynchronism caused by mobility, the maximum clock drift continuously increases. At last the maximum clock drift increases over 70000$\mu s$.

After IEEE 802.11 TSF adopts QCS, the maximum clock drift decreases under 20000$\mu s$. Note that $N = 32$ in QCS implies that nodes only have $63$ quorum intervals in every $1024$ beacon intervals. In other words, the average of quorum intervals among all nodes is only about $30.76s$ throughout the entire simulation period ($500s$). This implies that the clock asynchronism problem can be easily relieved even though $N$ of QCS is large.

Fig. 4.2 shows the beacon sending times throughout the entire simulation period. From this figure, we can see that when the IEEE 802.11 TSF adopts QCS with a smaller $N$, the maximum or average beacon sending times increases only by a small amount. This means that QCS does not cause too much additional beacon transmissions. Note that $N = 0$ in this figure means that QCS is not adopted by IEEE 802.11 TSF.

Now we discuss how the value of $N$ in QCS can affect the average remaining time of asynchronous pairs of nodes, as shown in Fig. 4.3. Note that when IEEE 802.11 TSF does not adopts QCS as an enhancement, the average remaining time of asynchronous pairs is quite large since IEEE 802.11 TSF does not know how to handle the clock asynchronism

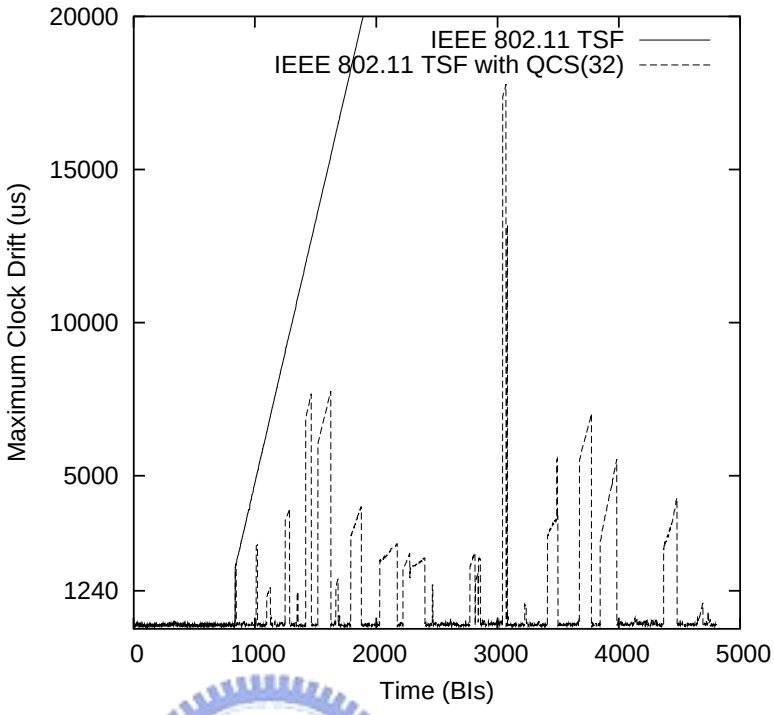



Figure 4.1: Maximum clock drift between any two neighboring nodes for IEEE 802.11 TSF and IEEE 802.11 TSF with QCS where $N$ is set to 32.

problem. In addition, it is not hard to see that when the value of $N$ in QCS is larger, the average remaining time of asynchronous pairs becomes longer because a quorum interval appears less frequently. However, it consumes more power resources when $N$ is smaller. For example, $N = 2$ implies that there are 3 quorum intervals in every 4 beacon intervals, i.e., nodes stay awake in about $75\%$ of the entire simulation period.

Although ASP has automatic self-time-correcting functions, it is not guaranteed that ASP can completely prevent the clock asynchronism problem. Fig. 4.4 shows the maximum clock drift between any two neighboring nodes during the whole simulation time when ASP is adopted for clock synchronization. As shown in Fig. 4.4, the maximum clock drift suddenly increases over $1240\mu s$ around $150s$. The reason of the occurrence of the first asynchronous pair is similar to the one in IEEE 802.11 TSF - mobility. At last, the maximum clock drift increases to $34000\mu s$ around. Similarly, after ASP adopts QCS with $N = 32$, the maximum clock drift decreases under $15000\mu s$.

Fig. 4.5 shows the beacon sending times throughout the entire simulation period. Compared with IEEE 802.11 TSF in Fig. 4.2, the maximum/average beacon sending times in ASP is fewer than that in TSF. Besides, from this figure, we can see that QCS causes some but not too many additional beacon transmissions. At last, Fig. 4.6 illustrates how

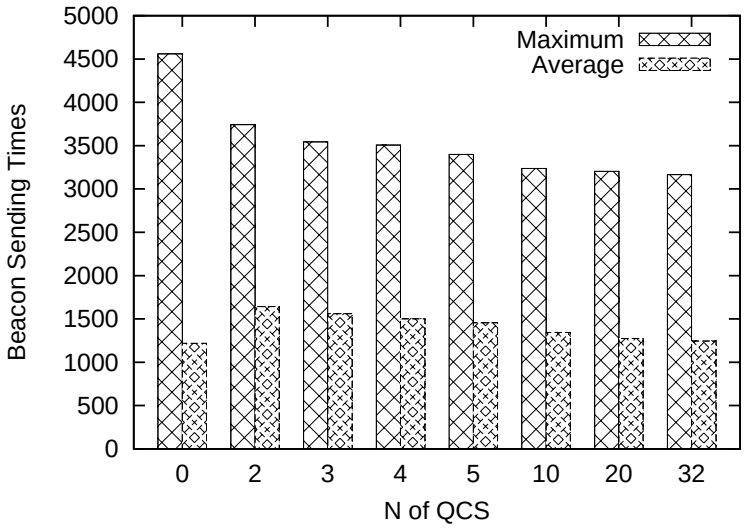



Figure 4.2: Beacon sending times for IEEE 802.11 TSF.

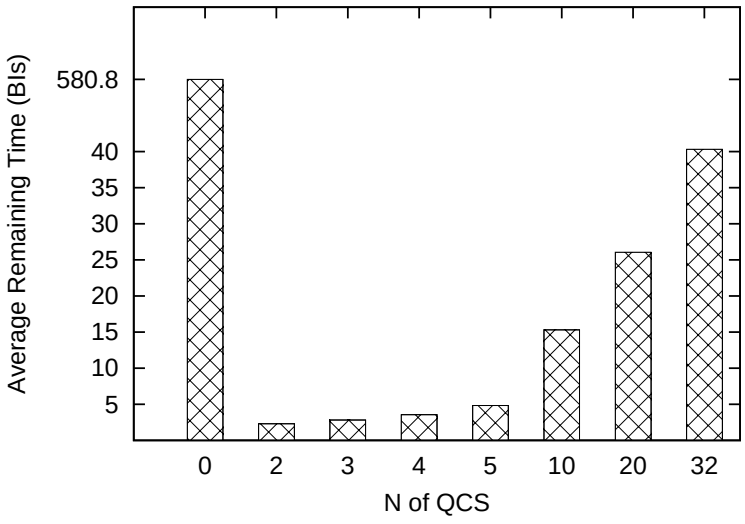



Figure 4.3: Average remaining time of asynchronous pairs for IEEE 802.11 TSF.

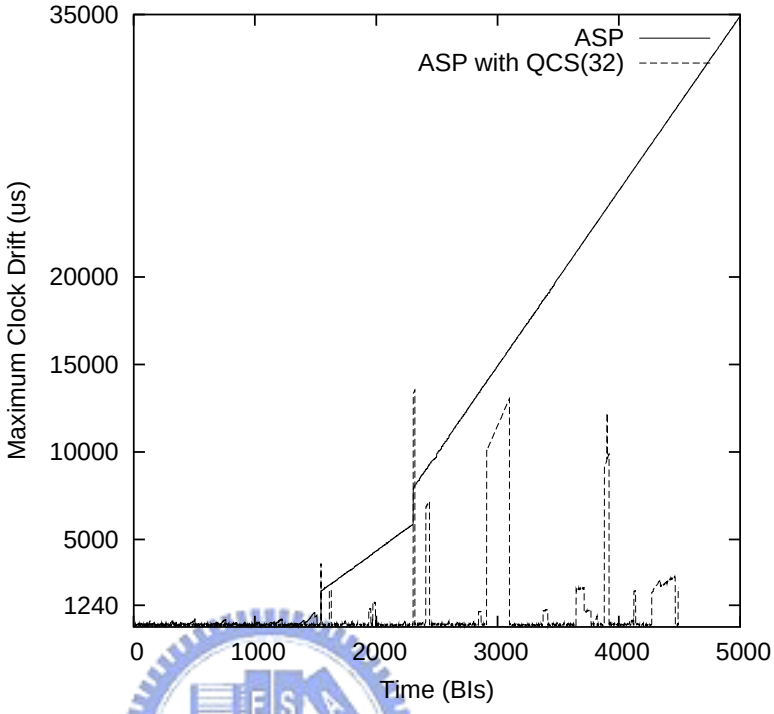



Figure 4.4: Maximum clock drift between any two neighboring nodes for ASP and ASP with QCS where $N$ is set to 32.

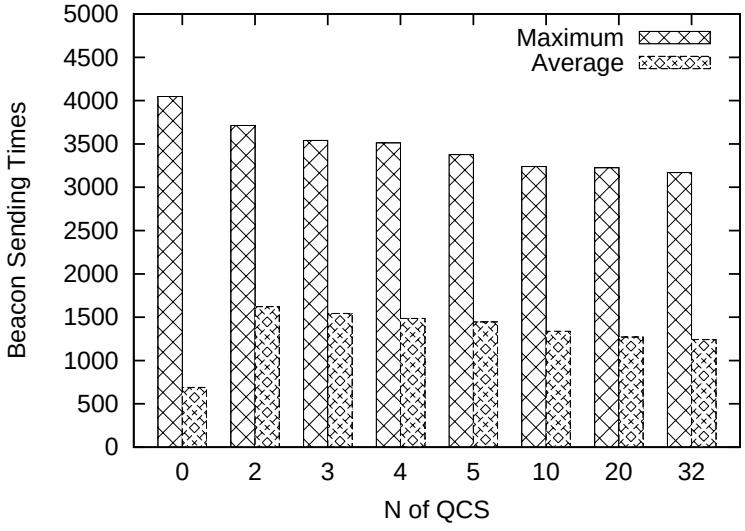



Figure 4.5: Beacon sending times for ASP.

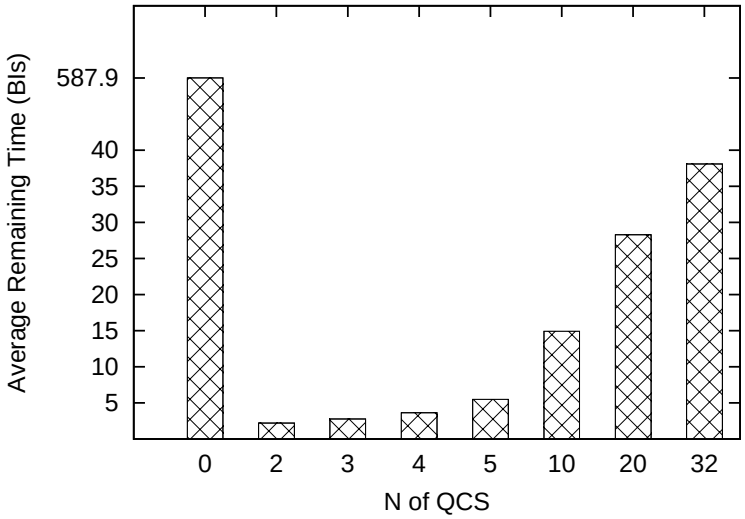



Figure 4.6: Average remaining time of asynchronous pairs for ASP.

the value of $N$ in QCS can affect the average remaining time of asynchronous pairs of nodes. As only ASP is adopted, the average remaining time of asynchronous pairs is up to $59s$. However, the situation becomes totally different when ASP adopts QCS. From this figure, we can see that the average remaining time of asynchronous pairs is smaller than $4s$ when $N \leq 32$ in QCS.

# Chapter 5

# Conclusions

In this paper, we point out that the current clock synchronization protocols lack an exceptional handling mechanism to solve the clock asynchronism problem in IEEE 802.11 MANETs. Therefore, we propose a compatible protocol called QCS to address the clock asynchronism problem in IEEE 802.11 multi-hop MANETs. Through our simulations, we show that our proposed scheme can assist the existing clock synchronization protocols in solving the clock asynchronism problem successfully.

# Bibliography


[1] Lifei Huang and Ten-Hwang Lai. On the Scalability of IEEE 802.11 Ad Hoc Networks. In *ACM Int'l Symp. on Mobile Ad Hoc Networking and Computing (MobiHOC)*, pages 173–182, 2002.

[2] Dong Zhou and Ten-Hwang Lai. Analysis and Implementation of Salable Clock Synchronization Protocols in IEEE 802.11 Ad Hoc Networks. In *2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'05)*, pages 255–263, 2004.

[3] Dong Zhou and Ten-Hwang Lai. A Compatible and Scalable Clock Synchronization Protocol in IEEE 802.11 AD Hoc Networks. In *Proceedings of the 2005 International Conference on Parallel Processing (ICPP'05)*, pages 295–302, 2005.

[4] Jang-Ping Sheu, Chih-Min Chao, and Ching-Wen Sun. A Clock Synchronization Algorithm for Multi-hop Wireless Ad Hoc Networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 574–581, 2004.

[5] Dong Zhou and Ten-Hwang Lai. A Scalable and Adaptive Clock Synchronization Protocol for IEEE 802.11-Based Multihop Ad Hoc Networks. In *2005 IEEE mobile ad hoc and sensor system conference*, 2005.

[6] Jungmin So and Nitin Vaidya. MTSF: A Timing Synchronization Protocol to Support Synchronous Operations in Multihop Wireless Networks. In *Technical Report*, 2004.

[7] Divyakant Agrawal and Amr El Abbadi. An Efficient and Fault-Tolerant Solution for Distributed Mutual Edxclusion. In *ACM Transactions on Computer Systems*, pages 1–20, 1991.

[8] Hector Garcia-Molina and Daneil Barbara. How to Assign Votes in a Distributed System. In *Journal of ACM*, pages 1029–1040, 1985.

[9] Jehn-Ruey Jiang, Yu-Chee Tseng, Chih-Shun Hsu, and Ten-Hwang Lai. Quorum-Based Asynchronous Power-Saving Protocols for IEEE 802.11 Ad Hoc Networks. In *ACM Mobile Networking and Applications (MONET)*, pages 169–181, 2005.

[10] Mamoru Maekawa. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. In *ACM Trans. Comput. Syst.*, pages 145–159, 1985.

[11] S. D. Lang and L. J. Mao. A Torus Quorum Protocol for Distributed Mutual Exclusion. In *Proc. of the 10th Conf. on Parallel and Distributed Computing and Systems*, pages 635–638, 1998.

[12] Wai-Shing Luk and Tien-Tsin Wong. Two new quorum based algorithms for distributed mutual exclusion. In *Proc. of International Conference on Distributed Computing Systems*, pages 100–106, 1997.

# Curriculum Vita

Shu-Min Chen (smchen@csie.nctu.edu.tw) received her B.S. degree in Computer Science from National Chiao-Tung University, Taiwan, in 2004. Her research interests include location system and wireless networks.