Implementation of Time Synchronization for BTnodes

Distributed Systems Laboratory
Winter Term 2006/07

by
Qin Yin

Professor: Friedemann Mattern
Supervisor: Matthias Ringwald

Institute for Pervasive Computing
Distributed Systems Group
ETH Zurich
April 2007

Abstract

Time synchronization plays an important role in wireless sensor networks which aim at bridging the gap between the physical and virtual world. In this thesis, two time synchronization approaches are investigated and implemented for BTnodes: global continuous clock synchronization and local on demand time transformation. From both theoretical and practical point of view, the tree-based clock synchronization approach has some disadvantages while time transformation algorithms is more applicable in many data fusion scenarios. The accuracy of time transformation is analyzed with the aid of logic analyzer which could record time stamps of signal sent by connected BTnodes.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Wireless Sensor Network

Wireless sensor networks [4] are an increasingly attractive means to bridge the gap between the physical and virtual world. A WSN consists of large numbers of cooperating small-scale nodes, each capable of limited computation, wireless communication, and sensing. Sensor networks differ substantially from traditional distributed systems in its major resource constraints:

- Energy: only small batteries can be attached to a node
- Communication: unreliable wireless communication with short range, network topology changes dynamically
- Computation: compared with computers, the processing speed is limited
- Memory: Only a small amount of memory is available

These limitations complicate the design of protocols and applications for WSNs.

1.2 Time Synchronization in WSN

In a wide variety of application areas, WSNs are envisioned to be used to fulfill complex monitoring tasks. Sensor nodes cooperate in order to merge individual sensor readings into a high-level sensing result, such as integrating a time series of position measurements into a velocity estimate. This kind of data fusion [6] processes make particularly extensive use of synchronized time. A paradox of wireless sensor networks, then, is that they make stronger demands on a time synchronization system than traditional distributed systems, while simultaneously limiting the resources available to achieve it. In the following, various classes of synchronization are listed[7], and we have choose one to fulfill the application's requirement with the smallest possible effort in terms of computation, memory and energy.

- Scope: the geographic span of nodes that are synchronized, and completeness of coverage within that region;
- Internal or External: external synchronization is the synchronization of all clocks in the network to a time supplied from outside the network; internal synchronization is the synchronization of all clocks in the network, without a predetermined master time;
- Lifetime: either continuous synchronization that lasts as long as the network operates, or on-demand synchronization which can be achieved in two ways: event-triggered or time-triggered;
- Rate or Offset: rate synchronization means that nodes measure identical time-interval lengths; offset synchronization means that node measure identical points in time;
- Transformation: we could make all clocks display the same time at any given moment by performing rate and offset synchronization; or we could transform timescales by transforming local times of one node into local times of another node.

1.3 Motivation

Nodes of the wireless network form spontaneous connections when they are brought within communication range of each other, providing typically a symmetrical communication link where message exchange is possible in both directions. The established network topology might remain stable most of the time (not all the times because of the unreliable wireless communication) if all the nodes in the network are immobile. However, in many ad hoc networks, the nodes are mobile and the communication range is limited, so that the network topology changes dynamically and reconfigures frequently.

Depending on different characteristics of sensor network and different scenarios of application, we have to choose different time synchronization methods. In this thesis, we address two types of time synchronization.

First, in a network whose topology remains stable most of the time, we could synchronize the clocks in the network to the clock of a specified node to keep the clock disciplined at all times and always make a consistent timestamp available. This time synchronization could meet the requirements of the applications in which all nodes are required to perform an action at a specific time or the data fusion process wants to know the temporal relation of two events.

Second, dynamic ad hoc networks shows an important property: the frequent temporary exitance of network partitions, especially in sparse ad hoc networks with only a few nodes distributed over a large area in contrast to dense ad hoc networks. In this ad hoc network, it's impossible to achieve continuous time synchronization and nodes' clocks are normally unsynchronized. Actually, we do not need to synchronize the local computer clocks of the devices but instead generate time stamps using unsynchronized local clocks. When such locally generated time stamps are passed between devices, they are transformed to the local time of the receiving device. This kind of synchronization does provide exactly the service necessary for localization system and other situations in which we need to compare the relative arrival times of a signal at a set of spatially local detectors.

1.4 Chapters Overview

This thesis is organized as follows: in Chapter 2 we introduce the hardware and software needed for the development. Especially, we take a look at the system software, protocol stack, connection manager and clock mechanism of the BTnut. Then we analyze the possibility of synchronize the BT clock, based on this two synchronization approach are proposed in Chapter 3. Detailed implementation of the two approaches is described in Chapter 4. To measure the accuracy of the time transformation approach, some experimental results are illustrated in Chapter 5. At last, in Chapter 6, the whole thesis is concluded.

Chapter 2

Platform

2.1 Hardware and Software

To implement time synchronization, BTnode developer hardware and software kit is needed. BTnode terminal connection requires: BTnodes rev3, USB programming broads and USB cables. BTnode rev3 includes system core (Atmel ATmega128, 256 kB SRAM, generic IO/peripherals, switchable power supplies and Extension connectors) and dual radio (Bluetooth radio, low-power radio and on-board antennas).

The complete listing of software tools and their versions used in this thesis, please see appendix A.

2.2 BTnut Overview

2.2.1 Bluetooth Networking

Bluetooth [1] is a radio standard and communications protocol primarily designed for low power consumption, with a short range (power-class-dependent: 1 metre, 10 metres, 100 metres) based on low-cost transceiver microchips in each device.

A piconet is an ad-hoc computer network of devices using Bluetooth technology protocols to allow one master device to interconnect with up to seven active slave devices (because a three-bit MAC address is used). Up to 255 further slave devices can be inactive, or parked, which the master device can bring into active status at any time.

Multiple piconets can be linked together when a device is master in a piconet and slave in another piconet. These spanned nets are called scatternets. Figure 2.1 is an example of scatternet with piconets connected through sharing devices. The sharing device is the master of one piconet as well as a slave for the master of the other piconet.

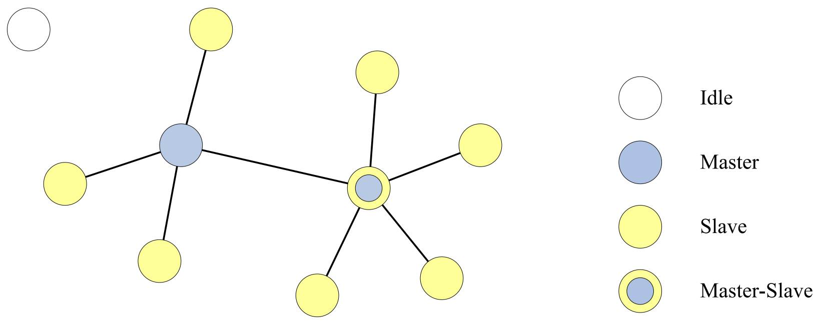


Figure 2.1: Scatternet

2.2.2 BTnut System Software

The BTnode runs with the BTnut [2] system software which is an expansion of Nut Operating System. Nut/OS [3] is an intentionally simple open source real-time operating system designed for embedded device development, and is the de-facto standard for WSN software. Its key features include:

- Cooperative multithreading
- Event queues
- Timer support
- Dynamic memory management
- Stream I/O functions

The BTnut system software not only preserves the high configurability but also has been been extended to provide BTnode specific drivers and libraries such as a Bluetooth stack and several communication protocols. It is possible to use native ANSI C for code development on the BTnode platform based on the almost complete C standard library provided.

2.2.3 BTnut Protocol Stack

The Bluetooth protocol stack itself consists of many different layers that are built on top of each other. As Figure 2.2 shows, the baseband layer, link manager layer exist on the controller chip; the controller offers the host the Host Controller Interface to control all the low level functions; the host implements HCI layer and above HCI layer are L2CAP(Logical Link Control and Adaption Protocol) layer and L2CAP connectionless layer.

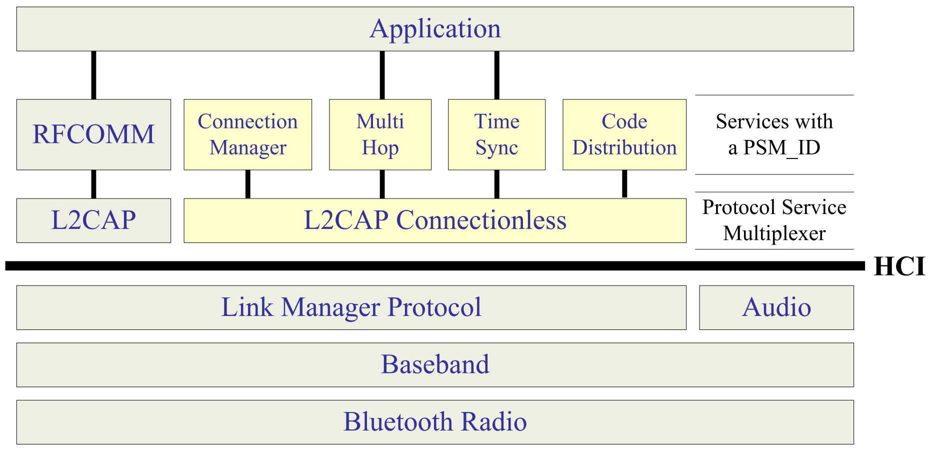


Figure 2.2: BTNut Protocol Stack

L2CAP connectionless layer provides functions to send connection-less data to directly connected neighbors. To receive l2cap connection-less data, an application has to register its service routine at the protocol/service multiplexor, together with a ”PSM” - a unique number identifying the service. Connection-less data can then be sent to a specific service by sending it to the corresponding ”PSM”. On top of L2CAP connectionless layer, host implements connection manager layer, multi-hop layer and code distribution. Bluetooth protocol stack constitute the foundation of the implementation of time synchronization.

2.2.4 BTnut Connection and Transport Manager

The Bluetooth standard contains no specification for the formation and control of multihop topologies or for the data transport across multiple hops. An additional functional layer must provide these services, i.e. for a modular structure, one layer for the topology control and one layer for the data transport.

The connection manager constructs and maintains a connected network in a distributed fashion. A robust algorithm is needed to automatically take care of nodes that join or leave the network and provide self-healing topologies in a completely distributed fashion. The basic principle is simple: every node periodically searches for other nodes in its communication range and connects to some discovered devices based on certain conditions.

The transport manager takes care of multihop packet forwarding. The transport manager takes use of the information of available connections provided by the connection manager. It provides a connectionless transport type and forwards the packets by either broadcasting or unicasting.

This thesis is based on tree connection manager, and multi-hop connectionless transport manager.

- Tree Connection Manager
  The tree connection manager [5] builds up a tree topology as showed in Figure 3.1. Thus, there are no loops possible in the network topology and the upperlaying transport manager will not receive any broadcast messages twice. The tree topology is a suitable solution for the network-wide time synchronization. The tree is constructed in this way: when two nodes connect, the node with a lower tree ID takes over the other node's tree ID and broadcasts it to its former tree. Like this, the new tree ID is distributed to all nodes in the new tree. We can assign a node as the root node by setting its EEPROM value using connection manager terminal command. This value will be read form the EEPROM at startup.

- Mhop Connectionless Transport Layer
  In connectionless transport layer, routing table is established through broadcasting packets. Each node receiving a broadcast packet stores the source node's address together with the connection handle the packet was received on in the forwarding table. Incoming packets having a target address that matches one of the entries in the forwarding table are then forwarded directly over the corresponding connection handle. Higher layer services can use the functions provided to send connection-less multi-hop packets to a specific service on the target device. On the receiving side, the service is identified by a unique "PSM" identifier which is included in the multi-hop packet header.

2.2.5 BTnode Clock Mechanism

- NutOS Time Management

  Nut/OS provides time related services, allowing application to delay itself for an integral number of system clock ticks by calling $NutSleep()$ or read the milliseconds counter value by calling $NutGetMillis()$. The counter value is incremented every system timer tick. During system start, the counter is cleared to zero and will overflow with the 64 bit tick counter (4294967296). With the default 1024 ticks/s (may vary depending on the configuration) this will happen after 7.9 years. The resolution is also given by the system ticks.

- BT Clock

  The BT Clock ticks every $0.3125ms = 3.2kHz$. Zeevo bluetooth module on BTnode Rev 3 returns the BT Clock divided by four. The returned time therefore has a precision of $1.25ms$. To somehow match the BT specification, the returned number is multiplied by four in the stack hence resulting in the two lowest bits always being zero. We can read the estimate of the value of the BT clock by calling *bt_hci_read_clock*(). Reading the BT clock takes some time: a command is send to the BT module over the serial connection, there will be 1-2 thread switches, and the result of this command will be processed. Therefore, normally we use the NutOS clock to time stamp event. Besides, a new version of this function returns a NutOS clock timestamp at the same time to make the dispersion of returned NutOS clock and BT clock as small as possible.

- BT Offset

  We can read clock offset to remote devices. The clock offset is specified as Bit 16-2 of $(CLKslave - CLKmaster) \mod 2^{17}$ which means that the clock difference is always positive and has a $1.25ms$ resolution. Therefore, the BT Clock offset is within $0 - 2^{15} * 1.25ms$, that is $0 - 20s$, and it is as accurate as its resolution implies. The mutual error between two nodes must be less than $1.25ms$, because in Bluetooth there are two slots within this period and in order to communicate both nodes have to synchronized roughly to this clock. So the clock offset might be even more accurate to up to $50\mu s$.

The BTnode Rev 3 Zeevo does not support retrieval of a remote clock but this can be accomplished by reading local BT clock and remote BT offset, getting remote BT clock, and then calculating BT clock difference accordingly. The details is explained in 3.2. The accuracy of the difference is guaranteed by the accuracy of the offset. Based on this idea, BT clock can be used for network wide synchronization and the NutOS clock can be used as a 'estimated' fast access to the BT clock.

Chapter 3

Concept

3.1 Overview

In this thesis we will implement both global continuous time synchronization and local on-demand time transformation for BTnodes. These two time synchronization approaches address separately the problems described in 1.3. The comparison of these two approaches is listed in Table 3.1 according to the synchronization classes illustrated in Section 1.2.

	Global Continuous Clock Synchronization	**Localized On-demand Time Transformation**
Sync vs. Trans	Clock synchronization	Time transformation
Scope	Connected Network	Localized
Lifetime	Continuous	On-demand
Rate vs. Offset	Offset	Offset
Internal vs. External	External	External

Table 3.1: Comparison of Two Synchronization Approaches

Obviously, the two approaches are named after their different characteristics. First, clock synchronization is achieved through continuous offset synchronization while time synchronization is done by transform local times of one node into local times of another node. Second, the scope is different, clock synchronization happens in the whole connected network while time transformation only occurs within the subnet in which event sensors could connect to the sink node. Third, continuous clock synchronization makes all the network nodes maintain synchronization all the times while time transformation is only triggered by the signal send by the sink node.

The two approaches are same in two aspects. On the one hand, both of them are offset synchronization which means at some time t, the software clocks of all node in the scope show t. It's trivial for clock synchronization since every node in the network has the same clock as the specified one. As for time transformation, because we need to combine event time stamps from different nodes, offset synchronization is needed. On the other hand, both of them are external synchronization which means the clocks are consistent within the network and with respect to the externally provided system time (either from the specified node or from the sink node).

From the comparison of the above two approaches, we can see some drawbacks of global clock synchronization:

- Dependency on Network Topology
  In sensor networks, the network topology is dynamic; nodes may be mobile and repeatedly join or leave the network. It is even possible that a tree topology is partitioned to form a forest.

- Dependency on Root
  In this approach, the root plays a vital role in the clock synchronization. If the root node fails, a new root has to be designate manually or elected automatically.

- Lifetime and Scope
  Continuous clock synchronization makes all the network nodes maintain synchronization all the times in the whole connected network which is quite energy consuming.

- Accuracy
  In global clock synchronization, every node is synchronized to the clock of the root, thus, the synchronization error is determined by the depth of the node in the tree topology. In localized time transformation, the error is determined by the number of hops from the event sensor to the sink node which will not be too far away from each other.

The drawbacks become more and more distinguished in the process of implementing this approach. From both theoretical and practical point of view, the time transformation approach has more advantages.

3.2 Synchronization of BT clock

Though BTnode Rev 3 Zeevo does not support retrieval of a remote clock, we can calculate this through local BT clock, remote BT offset and remote BT clock. The accuracy is guaranteed by the accuracy of the remote BT offset. Consider two BTnodes N_i and N_j that can exchange messages. These two nodes have to establish some relationship between their BT clocks $BTCLK^i$ and $BTCLK^j$. Node N_i sends a message containing a local timestamp $BTCLK_a^i$ to node N_j, where it is received at local time $BTCLK_b^j$. The offset of the two BT clocks is $BTOFF_{i,j}$ which is specified as Bit 16-2 of $(CLKslave - CLKmaster)$ mod 2^{17} and has a $1.25ms$ resolution.

Easily we can see:

$$|BTCLK_a^i - BTCLK_a^j| \mod 2^{17} = BTOFF_{i,j} * 4 \tag{3.1}$$

That is:

$$BTDIFF_{i,j} = |BTCLK_a^i - BTCLK_a^j| = \alpha * 2^{17} + \beta * BTOFF_{i,j} * 4 \tag{3.2}$$

Since we can only get the values of $BTCLK_a^i$ and $BTCLK_b^j$, we use $BTCLK_b^j$ as an approximation for $BTCLK_a^j$. So:

$$|BTCLK_a^i - BTCLK_b^j| = |BTCLK_a^i - BTCLK_a^j| + \Delta \tag{3.3}$$

Since α is calculated through:

$$\alpha = (|BTCLK_a^i - BTCLK_a^j| - \beta * BTOFF_{i,j} * 4 + \Delta) \mod 2^{17} \tag{3.4}$$

Here, Δ is the message delay consisting of send time, medium access time, propagation time and receive time, so the value of Δ is much smaller than $2^{15} * 1.25ms$. Meanwhile the value of $|BTCLK_a^i - BTCLK_a^j| - \beta * BTOFF_{i,j} * 4$ is close to $\alpha * 2^{17}$. However, Δ might influence the value of α in some corner cases when the value of Δ is negative. Thus, in order to get the correct value of α, we should take Bit 16 into account. If the Bit 16 is 1, it means α' we get is influence by a small negative and should be adjusted to the correct α by adding 1, that is, $\alpha = \alpha' + 1$.

The value of β is determined by the roles of the two nodes:

$$\beta = \begin{cases} 1 & : \quad \text{Node } N_i \text{ is Slave} \\ -1 & : \quad \text{Node } N_j \text{ is Slave} \end{cases} \tag{3.5}$$

$BTDIFF_{i,j}$ can be used in both synchronization approaches to transfer one BT timestamp in one node to the BT timestamp in another node.

3.3 Global Continuous Clock Synchronization

3.3.1 Scenario

In this thesis global continuous clock synchronization is implemented in BTnode network applying the tree connection manager. In this tree topology network, we could specify one master node as the root and propagate its clock all over the tree structure periodically as Figure 3.1 shows. single-hop synchronization is applied along the edges of the tree. As the accuracy degrades with the hop distance from the root, the leaf nodes might have the biggest deviation from the root's clock.

Thee-based global continuous time synchronization faces some problems: Firstly, the accuracy of synchronization is determined by the depth of the tree; Secondly, in many cases, nodes are mobile and repeatedly join and leave the network, thus the network topology might be dynamic and even partitioned which makes synchronization scope smaller; Third, the root might fail, a new root has to be specified manually or elected.

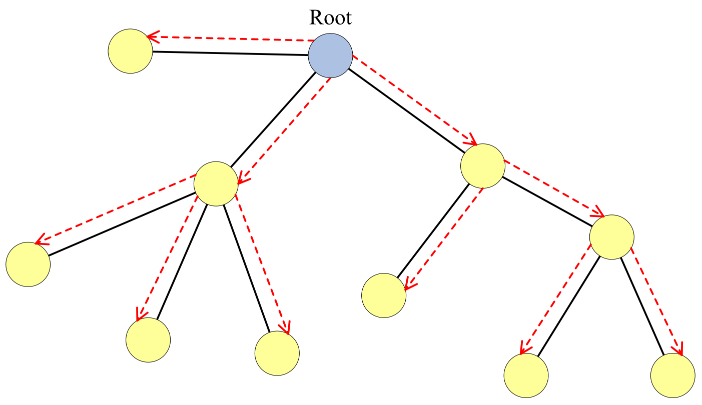


Figure 3.1: Global continuous Time Synchronization

3.3.2 Synchronization Approach

Consider again the synchronization of two BTnodes N_i and N_j. Node N_i sends a message containing a local BT timestamp $BTCLK_a^i$ and NutOS timestamp $NUTCLK_a^i$ to node N_j, where it is received at local BT clock $BTCLK_b^j$ and NutOS clock $NUTCLK_b^j$. Since there is deviation between a BTnode's BT clock and NutOS clock, suppose the deviation at time t is Δ_a^i[1]. We have:

$$NUTCLK_a^i = (BTCLK_a^i + \Delta_a^i) * 5/16 \tag{3.6}$$

$$NUTCLK_b^j = (BTCLK_b^j + \Delta_b^j) * 5/16 \tag{3.7}$$

$NUTDIFF_{i,j}$, the difference between NutOS clocks of two nodes can be calculated simply by $BTDIFF_{i,j} + \Delta_a^i - \Delta_a^j$ or $BTDIFF_{i,j} + \Delta_b^i - \Delta_b^j$. However, we can hardly get the exact values of Δ_a^i , Δ_a^j or Δ_b^i , Δ_b^j. But since the message delay is very limited, we can subtract Equation 3.7 from Equation 3.6 to get an approximate value:

$$NUTDIFF_{i,j} = (BTDIFF_{i,j} + \Delta_a^i - \Delta_b^j) * 5/16 \tag{3.8}$$

Substitute Δ_a^i and Δ_b^j in Equation 3.8 with Equation 3.6 and Equation 3.6 separately, we get:

$$NUTDIFF_{i,j} = (BTDIFF_{i,j} - BTCLK_a^i + BTCLK_b^j) * 5/16 + NUTCLK_a^i - NUTCLK_b^j \tag{3.9}$$

Till now, we can get the differences of the BT clocks and NutOS clocks of any two neighboring node. If each node in the tree topology calculates and stores its BT and NutOS clock differences periodically from the root's, the clock synchronization among the whole tree structure can be achieved.

[1]Since the two module are initialized separated and the two clocks drift differently, the relation between the $BTCLK$ and $NUTCLK$ is $BTCLK = \alpha * NUCLK + \beta$ where α is approximate to 16/5 and β is determined by concrete application. However, here we use the formula $BTCLK = NUCLK * 16/5 + \Delta$ as a substitution. The reason why this works will the explained in Chapter 5.2

3.4 Localized On-demand Time Transformation

3.4.1 Scenario

Localized on-demand time transformation can be illustrated by an example: when some event happens, each node in the range records the event time stamp with respect to its own local clock. Some time later, a 'sink' node comes to join in the network and broadcasts a data collection pulse to all nodes in the area. Nodes that receive this pulse now synchronize the timestamps of the event they recorded by transforming the locally generated time stamps to the local time of the receiving device when they are passed between devices. In this way, the timestamps are synchronized along the route way to the sink as depicted in Figure 3.2.

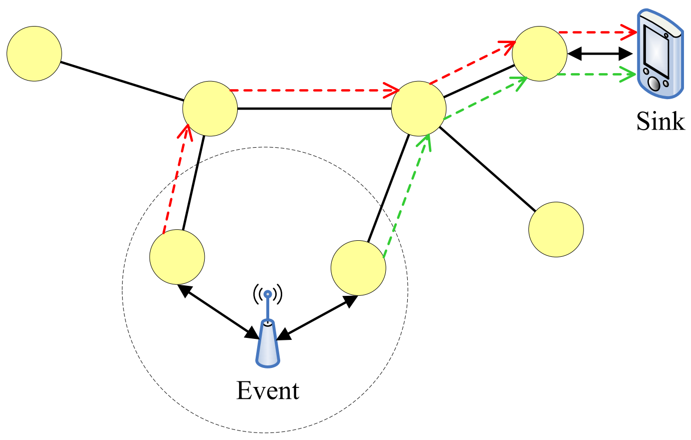


Figure 3.2: Localized On-demand Time Transformation

3.4.2 Synchronization Approach

As explained before, the BT offset can be used to calculate the difference of two BT clocks, in other words, the BT clock can be used for network wide synchronization. However, normally we use the NutOS clock to time stamp events. Especially as using the BT clock takes some time: a command is sent to the BT module over the serial connection, threads will be switched, the result of this command will be processed. In this way, NutOS clock is used as a fast access to the BT clock.

The only problem left the synchronization of the BT clock and Nut/OS clock in one BTnode. The detailed experimental results of the difference between the NutOS and BT clock is showed in Chapter 5.

Consider the scenario BTnode N_i sends a timestamped event packet to the 'sink' node N_j through a intermediate nodes N_k. N_i first transfers the NutOS event times tamp to a BT time stamp, and sends the BT time stamp to N_k. N_k transfers the received BT time stamp to its BT time stamp with the aid of $BTDIFF_{k,i}$, in the same way, N_j transfers the received time stamp by $BTDIFF_{j,k}$. Since N_j is the 'sink' node, it will estimate the real happening time of the event.

Chapter 4

Implementation

Based on the concepts discussed above, we implemented two kinds of time synchronization services in BTnodes. Clock synchronization service is used to force all the nodes in a tree network have the same clock as the root's. The root in the network periodically broadcasts the clock data to its children who will adjust their clocks accordingly. The children then repeat the same process until all the BTnodes in the network have the same clock. Time transformation service addresses this problem is a different way that every node in the network keeps different offsets for its neighbors. Whenever a timestamped event need be transmitted to the sink, the intermediate node modifies the event timestamp according to the clock offset of the next hop node. This procedure ensures that the sink can get the accurate timestamp relative to its own clock after several hops.

For clock synchronization service, the synchronized clock is only restricted within a single tree network. It means outside the tree network the clock of BTnodes is out of synchronization. It is also possible that there are a lot of disconnected tree networks which form different synchronized islands. In such case, clock synchronization can not guarantee a unified global time among all BTnodes. For time transformation service, there is no synchronized global clock existing. Every node has its own view of current time, however when performing transformation, the node can send the correct timestamp to its neighbor. The synchronization happens during the connection construction in order to synchronize the two connected nodes as early as possible.

4.1 Clock Synchronization Service

The clock synchronization is done hop by hop through a tree network. When the network forms a tree structure, the clock synchronization thread in the tree root is notified. This type of event triggers the operation of clock synchronization in which the root node begin to broadcast synchronization packet to all of its direct children. This precess is repeated until all the leaf nodes in the network get the clock synchronization data.

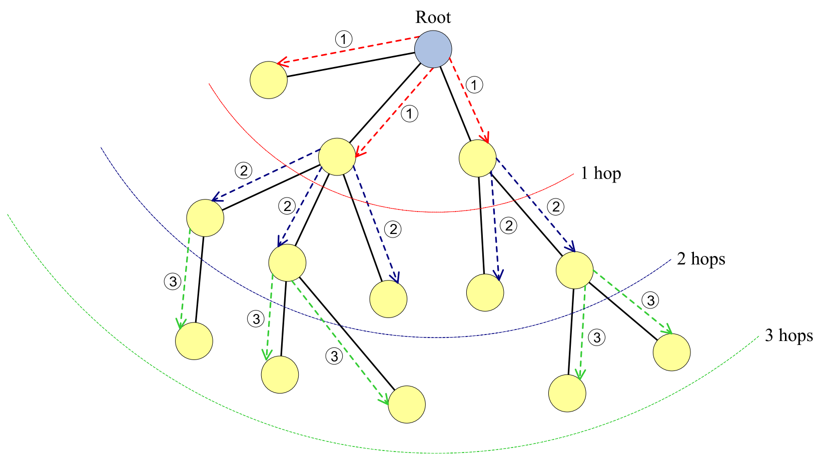


Figure 4.1: A Three-hop Example of Clock Synchronization

Figure 4.1 shows a three-hop example of clock synchronization. Root node is the initiative node which periodically broadcasts its time information. The red line delimits the range of the 1 hop from the root. When receiving the time information, the direct descendants of the root perform similar operation as the root did. After 3 hops, all the nodes in the range determined by the green line have a synchronized clock.

4.1.1 Data Encapsulation

All Clock-Sync PDUs consist of an integral number of octets, numbered starting from 1 and increasing in the order that they are put into a L2CAP packet. The bits in each octet are numbered from 0 to 7, where 0 is the low-order bit.

When consecutive octets are used to represent a binary number, the lower numbered octet contains the more significant bits of the binary number[1].

[1]It is also known as big-endian order used in network communication.

When the encoding of (an element of) a Clock-Sync PDU is represented using a diagram in this clause, the following representations are used:

- Octet 1 is shown toward the top of the page, higher numbered octets being toward the bottom.
- Where more than one octet appears on a given line, octets are shown with the lowest numbered octet to the left, higher numbered octets being to the right.
- Within an octet, bits are shown with bit 8 to the left and bit 1 to the right.

4.1.2 Clock-Sync Packet Definition

A summary of the form of a Clock-Sync packet for L2CAP is shown in Table 4.1. First, it contains the sender's bluetooth clock and sender's system clock for the receiver to calculate absolute offset. Then, the bluetooth offset follows. The reason why a packet includes bluetooth offset is to save time for the receiver since in a very short period the offset is constant. The last part is the absolute bluetooth/system clock differences between the sender and the root. The receiver can use this part to get cumulative absolute differences of its own view.

Field	**Meaning**	**Octet Number**
BT Clock	Sender's BT Clock	1-4
SYS Clock	Sender's System Clock	5-8
BT Offset	BT Clock Offset between Sender and Receiver	9-12
Root BT Diff	Root BT Clock - Sender BT Clock	13-16
Root SYS Diff	Root System Clock - Sender System Clock	17-20

Table 4.1: Clock-Data Packet Body

4.1.3 Protocol and APIs

A formal description of clock synchronization protocol is illustrated in Figure 4.2. It defines the state machine of root role and child role. Normally, root node starts with SYNC state by some initial routine. In SYNC state, root node broadcasts Clock-Data packets to all its neighbors. The payload fields of Root BT Diff and Root SYS Diff are set to 0. It then unconditionally transfers to IDLE state. When a tick event happens after a fixed time period, the root node becomes SYNC state again and repeats the same action. Child node starts with IDLE state which means it is ready to receive Clock-Data packet. When such a packet is arriving, it transfers to RECV state in which the child node performs clock synchronization calculation and sends the result to all its neighbors except the packet sender in SYNC state. After finishing synchronization phase, it goes back to IDLE state to wait for another Clock-Data packet.

Since the network may dynamically change all the time, probably a root node becomes a child node while another child node takes charge of root role during a very short period. When it happens, for the transformation of root role to child role, it is done implicitly since IDLE state can reach to RECV state. For the transformation of child role to root role, it must be triggered by some event such as root init which may be generated by some periodical thread.

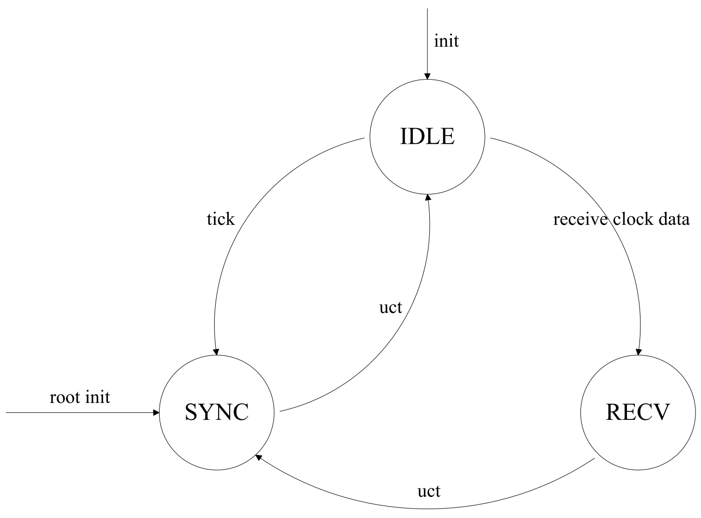


Figure 4.2: Clock Synchronization State Machine

To provide synchronized clock information to applications in upper layers, clock synchronization service implements the following APIs:

```
u_long time_sync_get_sys_clock(void);
u_long time_sync_get_bt_clock(void);
```

`time_sync_get_sys_clock` is used to get synchronized system clock while `time_sync_get_bt_clock` is used to get synchronized bluetooth clock. The synchronization behaves in the way that when some application needs read current time it uses `time_sync_get_` methods instead of functions provided by system APIs, then it gets a global synchronized time that is consistent in the whole tree structure. If the application is to capture the movement of some object, all the nodes who detect the movement event would report correct global time to the application so that it can calculate an accurate trace.

4.2 Time Transformation Service

In order to serve applications in upper layers with a consistent time, the time transformation is constituted of two phases. During the first phase, each neighbor of two nodes sends time information of itself including bluetooth clock and bluetooth offset to the other side. When receiving its neighbor's time data, the BTnode calculates an absolute difference between the two nodes. This initialization process occurs when two BTnodes establishes their first connection. After completing the first phase, every BTnode in the network has the exact knowledge of its neighbors' bluetooth clock. The second phase happens when some application sends a timestamped packet to the destination in another BTnode whose distance could be a single hop or multiple hops. In case of the single hop, the destination node changes the timestamp before delivering to the upper layer. The middle nodes in multi-hop behave equivalently except for not delivering and resending the packet.

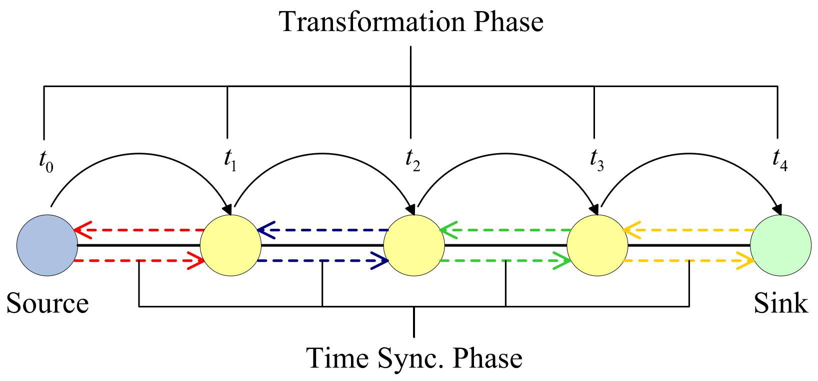


Figure 4.3: A Four-hop Example of Time Transformation

Figure 4.3 shows a four-hop example of time transformation. The initial phase ensures all nodes know the absolute differences between its own bluetooth clock and other neighbors'. The transformation phase consists of four hops in total where t_0 represents the time of event occurrence relative to the source node and t_4 represents the time of event occurrence relative to the sink node. In the middle hops, BTnodes transform the received timestamp and forward to the next hop.

Another important aspect for time transformation service is that the delta of bluetooth clock and system clock should be adjusted periodically since there is a small drift between two clocks. It is implemented by some backend thread who samples several pairs of bluetooth clock and system clock to induce fresh delta.

4.2.1 Data Encapsulation

Data encapsulation is exactly the same as 4.1.1.

4.2.2 Time-Trans Packet Definition

A summary of the form of a Time-Trans packet for L2CAP is shown in Table 4.2, starting with the packet type. The fields shown in the table are defined in the following paragraphs.

Field	**Octet Number**
Packet Type	1
Packet Body	2-N

Table 4.2: Time-Trans Packet Format

Packet Type

This field is one octet in length, taken to represent an unsigned binary number. Its value determines the type of packet being transmitted. The following types are defined:

- Time-Data A value of 0000 0000 indicates that the packet carries time data between two neighbors.
- Singlehop-Timestamp-Event A value of 0000 0001 indicates that the packet carries a timestamp associated with an event data in single hop transmission mode.
- Multihop-Timestamp-Event A value of 0000 0010 indicates that the packet carries a timestamp associated with an event data in multiple hop transmission mode.
- Multihop-Request A value of 0000 0011 indicates that the packet carries a route request in multiple hop transmission mode when the source node can not find the next hop for the destination node.

- Multihop-Response A value of 0000 0100 indicates that the packet carries a route response in multiple hop transmission mode to establish multi-hop route.

Packet Body

The Packet Body field is present if the packet length is the type is Time-Data, Singlehop-Timestamp-Event, Multihop-Timestamp-Event or Multihop-Request. The body contents of a Time-Data is described in Table 4.3. BT Clock indicates the bluetooth clock of the sender at the point of sending message. The value of BT Offset is the same for both the sender and the receiver because it is got from mod operation.

Field	**Meaning**	**Octet Number**
BT Clock	Sender's BT Clock	1-4
BT Offset	BT Clock Offset between Sender and Receiver	5-8

Table 4.3: Time-Data Packet Body

The body contents of a Singlehop-Timestamp-Event is described in Table 4.4. PSM specifies a unique number of the service which resides on the destination node. When the packet reaches the destination node, it will be delivered to that service. The pair of timestamp and event describe the fact that the event occurred at the point of the timestamp.

Field	**Meaning**	**Octet Number**
PSM	A Unique Service Number	1-2
Timestamp	The Time Associated with Event	3-6
Event	An Event Occurred at Timestamp	7-N

Table 4.4: Singlehop-Timestamp-Event Packet Body

The body contents of a Multihop-Timestamp-Event is described in Table 4.5. Comparing with Singlehop-Timestamp-Event packet, the multi-hop packet adds a destination address field which is used to find the forwarding entry at each middle hop.

The body contents of a Multihop-Request is described in Table 4.6. The only field is source address for establishing route.

There is no body for Multihop-Response packet since it is only used for establishing route on demand.

Field	**Meaning**	**Octet Number**
PSM	A Unique Service Number	1-2
Timestamp	The Time Associated with Event	3-6
Dest Address	The Destination BTnode MAC Address	7-12
Event	An Event Occurred at Timestamp	7-N

Table 4.5: Multihop-Timestamp-Event Packet Body

Field	**Meaning**	**Octet Number**
Source Address	A The Destination BTnode MAC Address	1-6

Table 4.6: Multihop-Request Packet Body

4.2.3 Protocol and APIs

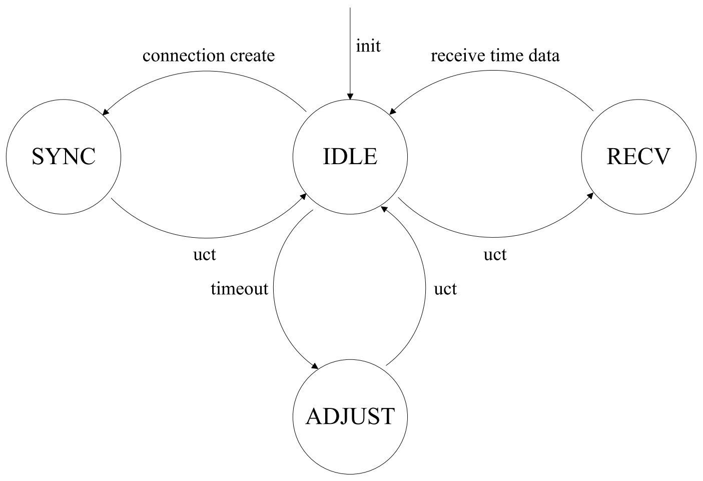


Figure 4.4: Time Transformation State Machine

A formal description of time transformation protocol is illustrated in Figure 4.4. It defines the state machine of time transformation service role. Every node begins with IDLE state in which the service entity waits for some events to trigger state transfer. If the lower level facility detects the event of connection creating, the service entity enters into SYNC state where it actively sends a Time-Data packet to one of its neighbor who recently connects to it. It then goes back to IDLE state to wait for other kinds of events. To measure the relation between system clock and bluetooth clock as accurate as possible, the service entity should periodically adjusts the delta of two clocks. It is done within ADJUST state.

If the service entity becomes RECV state after receiving some Time-Data packet from its neighbor, it will calculate an absolute offset and keep it for future transformation use.

To provide system clock/bluetooth clock transformation, time transformation service implements the following APIs:

```
long bt_clock_to_sys_clock ( long bt_clock);
long sys_clock_to_bt_clock ( long sys_clock);
```

The purpose of the transformation between the two clocks is due to the fact that the cost of reading bluetooth clock is much more than that of reading system clock. Giving the transformation can improve the efficiency by reading NutOS clock instead of reading BT clock. In addition, keeping an updated Δ can efficiently improves the accuracy of the transformation.

Time transformation service also implements two types of communication APIs to support sending timestamped event packets and decoding received timestamped event packet:

```
long send_tstamped_pkt ( long event_bt_tstamp,
                         u_char * data,
                         u_short data_len,
                         bt_hci_con_handle_t hdl,
                         u_short psm);
```

`send_tstamped_pkt` has the exactly the same parameters as function `l2cap_cl_send` except for one additional parameter `event_bt_tstamp` which is the BT time stamp of the event. `send_tstamped_pkt` will add a time stamp field to the original packet and sent it then.

```
long mhop_send_tstamped_pkt ( long event_bt_tstamp,
                              u_char * data,
                              u_short data_len,
                              bt_addr_t dest,
                              u_short psm,
                              u_char bc_flag,
                              u_char ttl);
```

Parameters of `bc_flag` and `ttl` are the same as the ones in `mhop_cl_send_pkt` except for one additional parameter `event_bt_tstamp` which is the BT time stamp of the event. `send_tstamped_pkt` will add a time stamp field to the original packet and send it hop by hop using the use of single hop transmission mechanism. Here, the routing information is obtained by looking up the routing table provided by the mhop facilities.

```
long handle_tstamped_pkt ( u_char * pkt,
                           u_short pkt_len,
                           u_char ** data,
                           u_short * data_len);
```

Since the packet we send has an additional time stamp field, so `handle_tstamped_pkt` is provided to decode the received timestamped packets. To separate timestamp and event data at the destination side, applications should invoke function `handle_tstamped_pkt`. Here, `pkt` is the received timestamped event data packet; `pkt_len` is the length of the packet; `data` is a pointer to the pointer of the original event data; `data_len` is the amount of event data received; the return value is transferred local time stamp.

Chapter 5

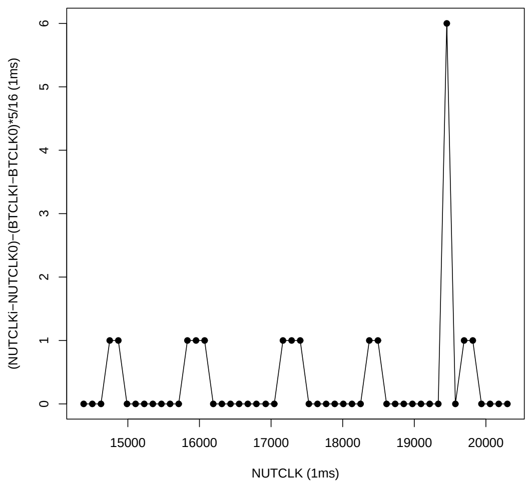
Measurement

Accuracy measurement and evaluation is one of the most important aspects of time synchronization for BTnodes. Since reading BT clock includes thread switches, its value will be influenced by the working load of the BT module. In other words, if the BT module is busy with communication, then it will take more time to return the BT clock value. We will take a deep look at the relationship between the BT clock and NutOS clock, the influence of the working load of the BT module, and discuss the reason why we use NutOS clock to timestamp events in Section 5.1. Because of different start time and different frequency of BT clock and NutOS clock, the deviation of the two clocks along is changed along with time. We will analyze the deviation in Section 5.2. In Section, the accuracy of the two time synchronization algorithms will be measured.

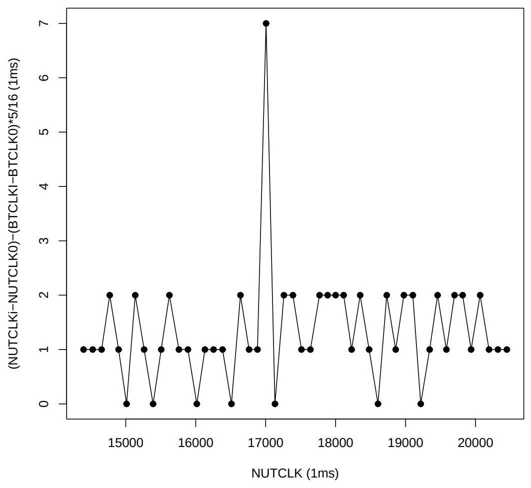
5.1 Difference between NutOS and BT Clock

Through `bt_hci_read_clock`, we can read the estimate of the value of the BT Clock and get a NutOS system clock timestamp at the same time. Normally, the BT clock should be close to the NutOS clock and satisfies the equation of $BTCLK = NUTCLK * 16/5 + \Delta$ (The reason why we can use fixed coefficient will be explained in Section 5.2). But if the process of reading the BT clock involves more thread switches, the BT clock will be read earlier than the NutOS clock. Thus, the closer the two clock are returned, the more accurate the two clocks are. In other words, if the $BTCLK$ is fixed, the smaller the value $NUTCLK$ is, the closer the two clocks are returned. So among a couple of sample pairs, we should choose the pair with the biggest Δ.

To depict the relationship between the BT clock and the NutOS clock, we read 50 pairs of clocks as samples. Using the first pair of $(NUTCLK_0, BTCLK_0)$ as a base measurement, all the other pairs are normalized to $((NUTCLK_i - NUTCLK_0) * 16/5 - (BTCLK_i - BTCLK_0)) = \Delta_0 - \Delta_i$. In this way, we get Figure 5.1(a). Since the bigger Δ is, the better the pair of clocks is, we can conclude that in this figure the lower the point is, the better it is. As now the BT module is almost free, all the nodes should near the lowest point which is depicted in Figure 5.1(a) in which the majority of the nodes are around $y = -1$. The exceptionally node is probably caused by the thread scheduling.



(a) Free



(b) Busy

Figure 5.1: Difference between NutOS and BT Clock

To show the influence of the working load of the BT module, we make another BTnode constantly send ping message to the target BTnode every $10ms$, in the way, the BT module in the target node has to response to the callback function and switch the thread frequently. Still, we read 50 pairs of clocks as samples. In Figure 5.1(b), the returned samples are normalized to $((NUTCLK_i - NUTCLK_0) * 16/5 - (BTCLK_i - BTCLK_0)) = \Delta_0 - \Delta_i$ to be showed in the coordinator. Easily, we can see that the differences become bigger, which means, the NutOS clocks are returned later.

5.2 Drift of Clock Differences

Since the two module are initialized separated and the two clocks drift differently, the relation between the $BTCLK$ and $NUTCLK$ is $BTCLK = \alpha * NUCLK + \beta$ where α is approximate to 16/5 and β is determined by concrete application. As we explained earlier, we would like to use NutOS clock as a fast access to the BT clock which require some way to transform between BT clock and NutOS clock. We can use the formula $BTCLK = \alpha * NUCLK + \beta$, collect a couple of samples and calculate α and β. However, this method will inevitably introduce floating point computation which is energy consuming.

Therefore, we use the formula $BTCLK = NUCLK * 16/5 + \Delta$ as a substitution. The reason why this works is showed in Figure 5.2. Suppose the red line is the real line with a slope of 3.19997 which is calculated from 100 sample clock pairs in one BTnode, the green ones are $BTCLK = NUCLK * 16/5 + \Delta$. As we can see, if we update Δ frequently enough, we can use the green lines to substitute the red one, more importantly, it involves no float point computation.

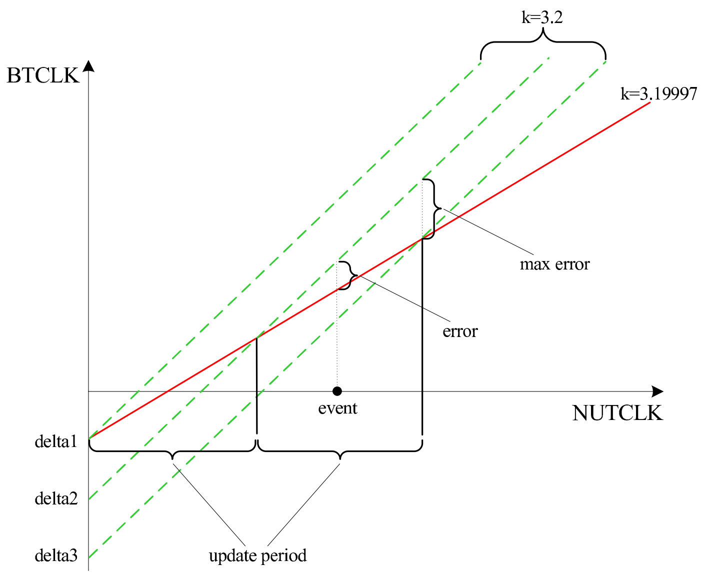
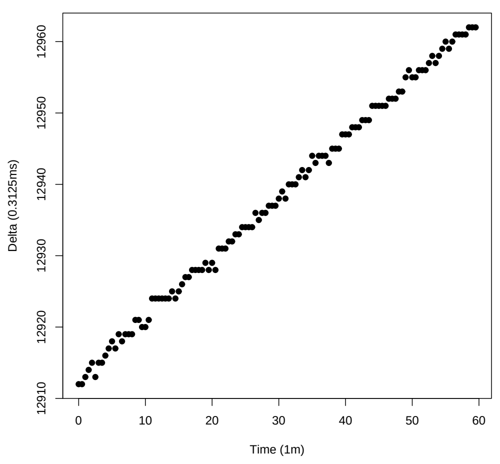
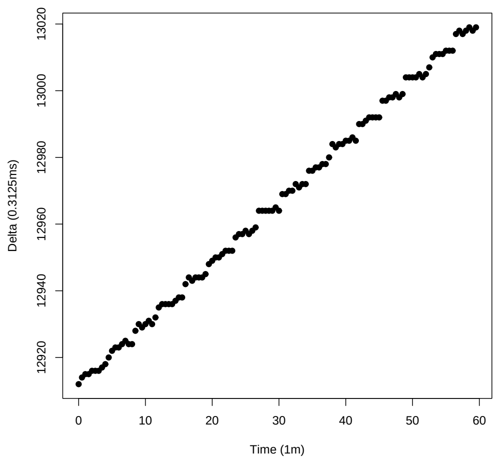


Figure 5.2: Two Time Transformation Methods

Since BT module and NutOS module are initialized separated, Δ has different initial values. Meanwhile, it drifts at different rate in different BTnodes along with the time. We will show this through an experiment. The setting of the experiment is as follows: Δ is calculated every 30 seconds, and it is the optimal value get from 10 samples which are read every 100ms. The experiment is executed for an hour and 120 sample values are showed in Figure 5.3. The drifts of the two BTnodes are separately 50*0.3125ms and 107*0.3125ms (the highest drift among 7 BTnodes) for an hour. In order to make clock drift not exceed 1ms, the value of Δ could be adjusted every 30 seconds (actually every 1 or 2 minute is fine as well). Compared to the energy for the BT module, this frequency is acceptable.



(a) BTnode: 00:04:3F:00:01:F0



(b) BTNode: 00:04:3F:00:01:C8

Figure 5.3: Drift of BT clock and NutOS clock

5.3 Global Clock Synchronization

At the first stage of measuring global clock synchronization, we got some results contrary to our intuition: the errors increase linearly instead of fluctuating periodically. Through looking into the algorithm introduced in Section 3.1, we find that the accuracy of the algorithm is mainly caused by the BT offset between two neighboring nodes. Based on this discovery, we get an important finding from the data of the failure experiment: the following function doesn't work properly.

```
long bt_hci_read_clock_offset ( struct btstack * stack,
                                struct bt_hci_cmd_response * response,
                                bt_hci_con_handle_t app_con_handle );
```

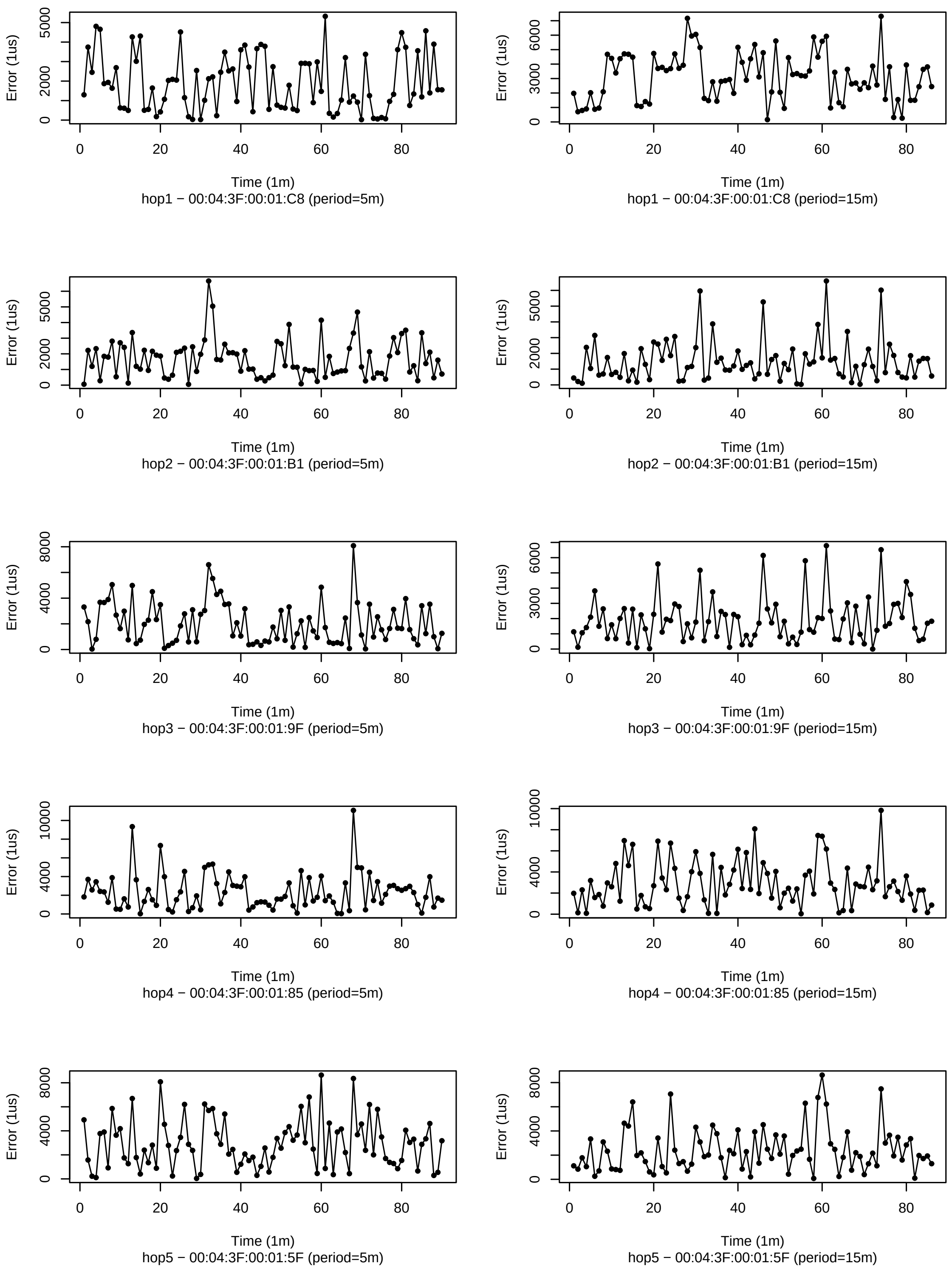


Figure 5.4: Synchronization Errors in 5-hop Tree

It returns only the value when the connection is established and doesn't change since then. So in our implementation, we have to resort to a much more low level function.

```
long bt_hci_inquiry ( struct btstack * stack,
                      struct bt_hci_cmd_response * response,
                      u_char time,
                      u_char number,
                      struct bt_hci_inquiry_result * res );
```

The experiment setting of the tree-based clock synchronization is as follows: all the nodes in the tree record its BT and NutOS clock differences from the root node. Since every node has the global clock $GLOCLK$, the nodes will signal the logic analyzer whenever $GLOCLK \mod 60000 \leq 10$, that is, every 1 minute. The logic analyzer records all the time points when BTnodes signal it and calculate the time point differences from root node to other nodes.

In this experiment, six BTnodes F0, C8, B1, 9F, 85, 5F are connected one by one to form a communication link with 5 hops. BTnode F0 is the root node and propagates its clocks to its children every 30 minutes. In addition, the BT offset between two neighboring nodes is updated every 5 minutes. The period is much longer because of the cost of invoking `bt_hci_inquiry`. The errors of 5-hop global clock synchronization is depicted in Figure 5.4, the BT offset update frequency is 5 minutes in the left column and 15 minutes in the right column. We list the data in this way in order to show the influence to the synchronization error of the frequency of updating BT offset. The average error in the right column are bigger than that in the left column, meanwhile, we can see the errors are fluctuating every 15 minutes. This effect is more obvious in the next section when the periods are 5 minutes and 20 minutes separately.

5.4 Local Time Transformation

In local time transformation, we should focus on two aspects: the transformation of the BT clock and NutOS clock in one node and the synchronization of the BT offset between two neighboring nodes which is also very important in global clock synchronization. As we explained in Section 5.2, the transformation of the BT clock and NutOS clock is achieved by updating the Δ every 30 seconds. In Section 5.3, we explained the synchronization of the BT offset and influence of the frequency to update BT offset.

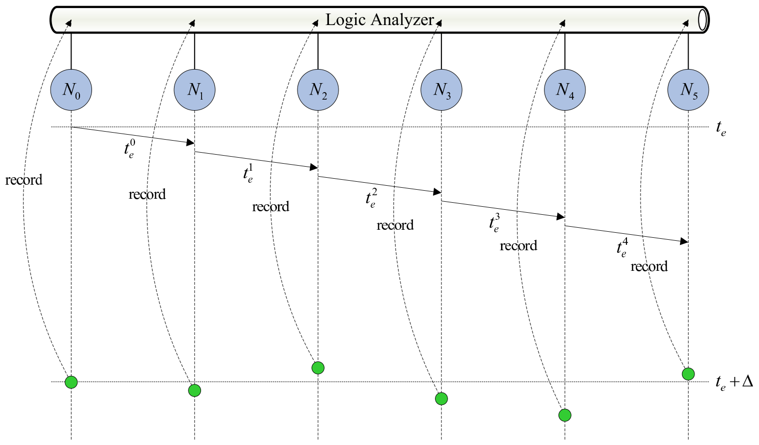


Figure 5.5: Multiple Hop Configuration

The experiment setting of multiple hop time transformation is illustrated in Figure 5.5. The nodes in the intermediate hops all should transform the event time stamp before delivering it to the next hop. All the BTnodes will signal the logic analyzer after a time period after its transformed event time stamp. The logic analyzer records all the time points when BTnodes signal it and calculate the difference of the time transformation.

For clarity, let's take single hop communication case which is showed in Figure 5.6 for example. The time stamp of an event is recorded by BTnode N_i as t_e. Later, at some time point t_a^i, N_i send a timestamped event packet to another BTnode N_j who receives the packet at t_b^j. N_j will transfer the time stamp of the event to its local clock t_e'. To show how accurate the transformation is, a logic analyzer is used. The logic analyzer could record accurately the time stamp when the BTnodes which are connected to it send a signal to one of its PIN ports. In order to see the difference of t_e and t_e', we could ask the two BTnodes to wait a fixed time period Δ and set its corresponding PIN port at $t_e + \Delta$ and $t_e' + \Delta$ separately. The logic analyzer will record the two time stamps and calculate the time differences.

In this experiment, six BTnodes F0, C8, B1, 9F, 85, 5F are connected one by one to form a communication link with 5 hops. BTnode F0 generates events every 1 minute and timestamps the event with its own NutOS clock. After that, BTnode:F0 sends event packet timestamped by its transformed BT clock to its neighbor C8. For the experiment, all the

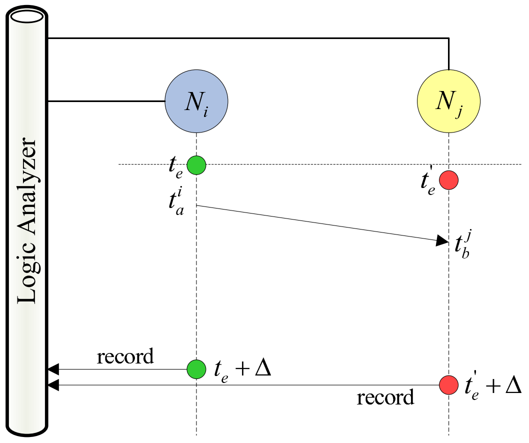


Figure 5.6: Single Hop Configuration

intermediate nodes not only transform the event time stamp to its own BT time stamp, but also calculate the NutOS time stamp of the event as the 'sink' node. In this way, all the nodes could be able to get the time point (500ms after the local event NutOS time stamp) when to trigger the logic analyzer. According to the time stamps recorded by the logic analyzer for two hours which includes 120 data tuples, we draw the figures of the differences between the timestamp of BTnode FO - event generator - and other nodes.

As we can see, time differences don't increase with respect to the time, but increase with the number of hops. The reason why time difference don't increase along with the time is: Δ in $BTCLK = NUCLK * 16/5 + \Delta$ is updated periodically and BT offset between two neighboring BTnodes is updated periodically as well. The reason for the increase along with the number of hops is: the errors are accumulated along the way. But since in most application scenarios the number of hops won't exceed 6, the errors below 10ms is acceptable.

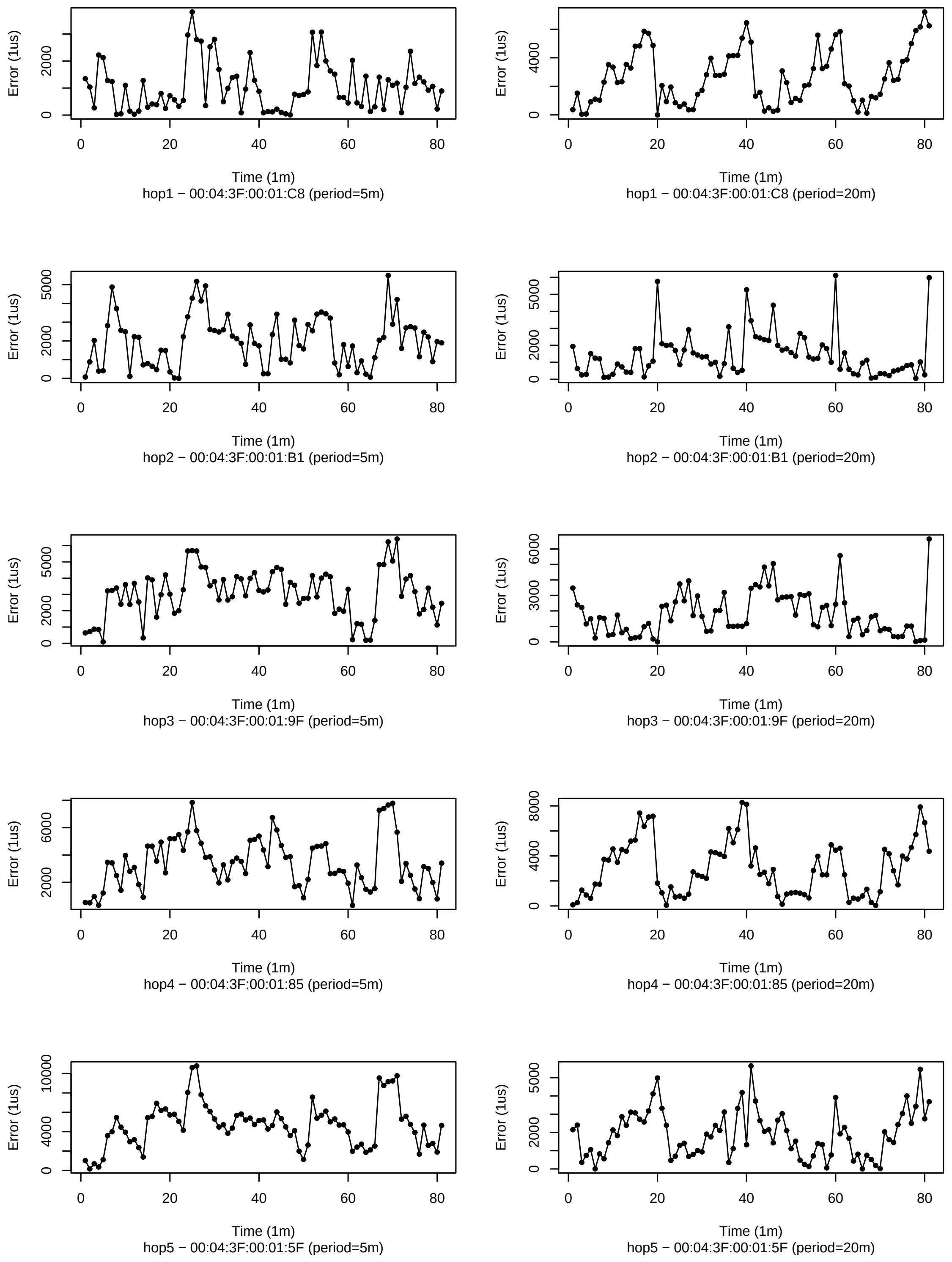


Figure 5.7: Multiple Hop Difference (Update period: 5m vs. 20m)

Chapter 6

Conclusion

Time synchronization plays an important role in wireless sensor networks which aim at bridging the gap between the physical and virtual world. Based on BTnodes, two types of services are introduced to address the issue of time synchronization.

- Clock synchronization service forces all the nodes in a tree network have the same clock as the root's. The synchronization approach is achieved by root node to periodically broadcast clock data to its children according to the tree network structure. The BTnodes who receive clock data repeat this process till reach the leaves.
- Time transformation approach keeps clock differences for different neighbors. In stead of maintaining a unified global time among all BTnodes, the BTnodes synchronize the time stamps of the event by transforming the locally generated time stamps to the local time of the receiving device when the timestamped event packets are passed between devices. In this way, the timestamps are synchronized along the route way to the sink.

Because of the considerations of clock synchronization service in its dependency on network topology and root node, its lifetime and scope, and its accuracy, time transformation approach has more advantages.

Using BT clock and BT offset, time transformation approach could achieve a high degree of accuracy. As using the BT clock takes more time, NutOS clock is used as a fast access to the BT clock. Thus, the accuracy of the transformation between BT clock and NutOS clock plays an important role in this synchronization approach. In this implementation, we use the formula $BTCLK = NUCLK * 16/5 + \Delta$ to represent this relationship.

However, since BT module and NutOS module are initialized separated, Δ has different initial values; Meanwhile, it drifts at different rate in different BTnodes along with the time. So Δ should be updated periodically. During each update, we collect a couple of BT and NutOS clock pairs and choose the smallest Δ which reflect the real values best.

With the aid of logic analyzer, we measured the accuracy of the two clock synchronization algorithms. The synchronization error doesn't increase with respect to the time, but increase with the number of hops. In a 5-hop network, the maximum error is 10ms. Each time the BT offset is updated, the synchronization error will decrease and later increase again. So in order to decrease the synchronization error, we can increase the BT offset update frequency (e.g. every 5 minutes).

Bibliography

[1] Bluetooth website: http://www.bluetooth.org. Technical report.

[2] Btnode website: http://www.btnode.ethz.ch. Technical report.

[3] Ethernut software manual: http://www.ethernut.de/en/documents/index.html. Technical report.

[4] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445–487, 2005.

[5] J. Beutel. Robust topology formation using btnodes. *Computer Communications*, 28(13):1523–1530, 2005.

[6] J. Elson and K. Römer. Wireless sensor networks: a new regime for time synchronization. *Computer Communication Review*, 33(1):149–154, 2003.

[7] K. Romer, P. Blum, and L. Meier. Time synchronization and calibration in wireless sensor networks.

Appendix A

Software Versions Used

The implementation of time synchronization for BTnodes used the following software versions:

- Windows XP
- Java 2 SDK 1.5.10
- eclipse-SDK-3.2-win32
- org.eclipse.cdt.sdk-3.1.0-win32.x86
- Silabs CP2101 USB to UART Bridge
- WinAVR 20060125
- BTnut: btnode.cvs.sourceforge.net /cvsroot/btnode 2007-01-23
- NutOS: ethernut.cvs.sourceforge.net /cvsroot/ethernut