Practical Time Synchronization for Bluetooth Scatternets

Matthias Ringwald, Kay Römer
Institute for Pervasive Computing, ETH Zurich, Zurich, Switzerland
Email: {mringwal,roemer}@inf.ethz.ch

***Abstract*— By means of so-called *Scatternets*, Bluetooth provides the ability to construct robust wireless multi-hop networks. In this paper we propose a practical protocol for time synchronization of such Bluetooth multi-hop networks. Our protocol makes use of the internal clock maintained by Bluetooth, requires minimal communication overhead, and provides an accuracy of few milliseconds across multiple hops. The protocol has been implemented and evaluated on BTnodes, an embedded computing platform which uses Bluetooth for ad hoc networking.**

I. Introduction

Bluetooth [6] is a widely used standard to form wireless personal area networks among devices and computers. Originally designed as a cable replacement, Bluetooth employs techniques such as frequency hopping to provide robust and high-bandwidth communication even in noisy environments. While in most applications only single-hop networks are formed, Bluetooth also provides a so-called *Scatternet* mechanism to form larger multi-hop networks.

Time synchronization is a fundamental service in almost any computer network, including Bluetooth networks. Surprisingly, Bluetooth does not provide time synchronization as a service to applications even though time synchronization is needed internally, as medium access is based on time-division multiple access (TDMA). However, the Bluetooth API provides a few functions that allow limited access to the internal clock that is maintained to control medium access. In this paper, we propose and evaluate a practical algorithm for synchronizing multi-hop Bluetooth Scatternets which makes use of these functions. The algorithm provides a synchronization accuracy of few milliseconds across multiple hops with minimal communication overhead.

Our work was motivated by an ongoing effort where a Bluetooth Scatternet is used as part of a tool to analyze wireless sensor networks (WSN) [11]. WSN are embedded into the environment to perceive aspects of the real world using sensors attached to the network nodes. The function of such a sensor network is very sensitive to the environment, frequent node and communication failures often render a WSN inoperational. We use a second network – a Bluetooth Scatternet, which provides more reliable and high-bandwidth communication – to inspect a WSN to find the cause of failures. For this, Bluetooth nodes are equipped with a second, low-power radio to overhear message exchanges in the WSN. Copies of overheard messages are delivered to a user via Bluetooth multi-hop communication. To correlate messages gathered by different Bluetooth nodes, time synchronization is required in the Bluetooth Scatternet such that overheard messages can be time-stamped by the receiving node according to a common time scale. The accuracy of synchronization must be sufficient to reliably order overheard messages according to reception time. In the WSN we consider, the duration of a message transmission is in the order of few tens of milliseconds. Therefore we need a synchronization accuracy of few milliseconds.

In Sect. II we provide background information on Bluetooth and introduce our approach to time synchronization. In Sects. III, IV, and V, the core of our protocol is described. We discuss implementation aspects in Sect. VI and evaluate our proposal

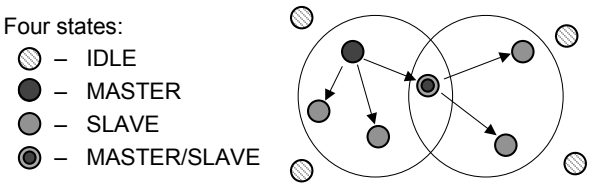

Fig. 1. Illustration of Bluetooth Scatternets.

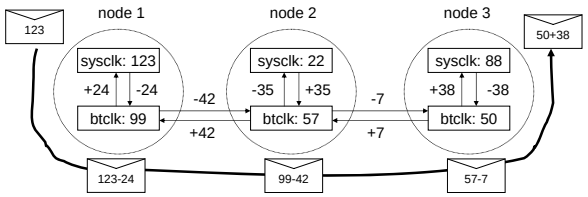

Fig. 2. Illustration of the synchronization protocol.

in Sect. VII.

II. Protocol Overview

Bluetooth operates in the unlicensed 2.4 GHz ISM band and uses frequency hopping to achieve reliable communication even in noisy environments. A group of devices which are synchronized to a common clock and frequency hopping pattern is called a *Piconet*. The device which provides the reference time for synchronization is called *master*, all other devices are referred to as *slaves*. Piconets have a star topology with the master at the center, that is, direct communication is only possible between a master and a slave, but not between slaves. A *Scatternet* consists of several inter-connected Piconets in which some nodes are part of more than one Piconet at the same time as illustrated in Fig. 1, where large circles indicate a Piconet.

Bluetooth is implemented as a *radio modem* with a well-defined command interface, the so-called "Host Controller Interface" (HCI), that is connected to the main processor via a serial interface (e.g., USB or RS232). The Bluetooth radio modem is itself a complex embedded computing device that contains, among others, a separate processor that runs the Bluetooth protocol stack and a real-time clock to control medium access. Access to the internal state of the modem (e.g., the real time clock) is only possible via commands that are sent to the modem via the serial connection. In fact, Bluetooth provides two commands related to the real time clock: `HCI_Read_Bluetooth_Clock` to read out the current value of the real-time clock, and a second command `HCI_Read_Clock_Offset` to read some (but not all) bits of the current offset of the clock to the clock of a connected node (see Sect. III).

Even though Bluetooth provides the above two commands as a foundation for time synchronization, the implementation of Bluetooth as a radio modem has far-reaching implications on the design of a synchronization protocol. Firstly, each network node has two clocks: the system clock and the Bluetooth clock. Typically, the operating system and applications use the (unsynchronized) system clock, whereas we intend to use the Bluetooth clock for synchronization among different nodes. That is, we need to synchronize the system clock with the Bluetooth clock in some way. Secondly, reading the Bluetooth clock (offset) is a costly and lengthy operation as it involves exchange of protocol messages between the main processor and the Bluetooth modem over a serial connection (in contrast, reading the system clock is cheap and fast). This implies that the Bluetooth clock (offset) should be read rarely. Moreover, execution time of a Bluetooth command is highly variable, as the reply to a command may be delayed by arriving data messages. That is, accurate synchronization of the system clock with the Bluetooth clock is non-trivial.

Fig. 2 illustrates the design of our protocol which was inspired by the above observations. Three nodes are shown which are connected in a chain topology using Bluetooth Scatternets. Each node is indicated by a circle that contains the current value of the system clock and of the Bluetooth clock. All clocks are unsynchronized – they advance freely at their respective rates without being disciplined. However, each node maintains offsets between its system clock and the Bluetooth clock (and vice versa) and its Bluetooth clock and the Bluetooth clocks of connected nodes. These offsets are illus-

trated in Fig. 2 by tagged arrows that point from clock A to another clock B. By adding the arrow tag (i.e., the clock offset) to the value of clock A, one can obtain the corresponding value on clock B. We use the above Bluetooth commands to obtain the offsets between Bluetooth clocks of connected nodes (see Sect. V) and to obtain the offset between the system clock and the Bluetooth clock (see Sect. IV). As the offsets change infrequently (the clock drift of the various clocks is small compared to the required synchronization accuracy), these commands are invoked infrequently to update the offset values.

The thick arrow in Fig. 2 illustrates the exchange of a time-stamped message between node 1 and node 3 via node 2. When generating a new message, node 1 reads its system clock (i.e., 123) and includes this value as a time stamp in the message. Next, the time stamp is converted to the Bluetooth clock of node 1 by adding the respective clock offset -24. Then, the time stamp is converted to the Bluetooth clock of node 2 by adding the respective clock offset -42 and the modified message is sent to node 2. There, clock offset -7 is added to the time stamp to convert to Bluetooth clock of node 3. The message is then sent to node 3, where the clock offset +38 is added to transform to the system clock of node 3. The resulting time stamp 88 equals the value of node 3's system clock at the time when the message was generated in node 1. With this approach, all time stamps a node receives from different nodes will be synchronized in the sense that they refer to the time scale defined by its local system clock.

The above approach is sufficient for many applications (including ours in Sect. I) and has two important advantages. Firstly, our approach does not require a designated node that acts as a time reference for other nodes and thus our protocol can easily deal with node failures and topology changes. Secondly, the difficulties of disciplining clocks are completely avoided.

III. Bluetooth Clock

The Bluetooth clock is a 28-bit counter with 0.3125 ms resolution and a mandatory maximal drift of $\pm$ 20 ppm. This results in an overrun every $2^{28} \times$ 0.3125 ms $\approx$ 1 day.

Each Bluetooth device has a unique 6-byte baseband address (BD_ADDR) similar to the medium access control address of Ethernet devices. The hopping sequence is a pseudo-random sequence of communication frequencies calculated from the BD_ADDR of the piconet master device. Because of the frequency hopping, a special procedure called *inquiry* is required to discover other devices (i.e., their address and hopping sequence). During an inquiry, a device uses a special inquiry hopping sequence and doubles its hopping rate to rendezvous with other devices. As a result of an inquiry, the BD_ADDR and the difference between the local Bluetooth clock and the clock of the remote devices are acquired. Based on this information, a node can calculate the hopping sequence of discovered nodes and is thus able to connect to these devices.

The clock offset to discovered devices is defined as bits 2-16 of the difference between the clock of the discovered device and the local clock. Similarly, the clock offset between two connected devices is specified as bits 2-16 of the difference between the clock of the slave node (CLKslave) and the clock of the master node (CLKmaster). With the reduced range (only bits 2-16) of these clock offsets, a maximum time interval of $2^{17} \times$ 0.3125 ms = 40.96 s with a resolution of 1.25 ms can be specified.

IV. Offset between System and Bluetooth Clock

Measuring the offset between the system clock and a Bluetooth clock is non-trivial as reading the Bluetooth clock requires sending a command message to the Bluetooth modem over the serial interface between the main processor and the Bluetooth modem and receiving a reply message (a so-called event) over the serial interface that contains the requested clock value.

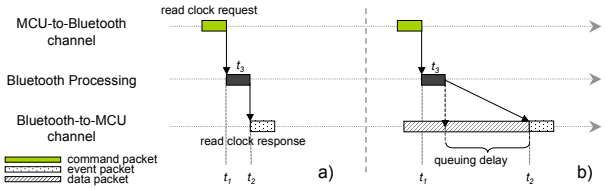

Fig. 3. Reading Bluetooth clock over Host Controller Interface. a) no parallel data traffic, b) incoming data packet.

This protocol is illustrated in Fig. 3 (a). First, the command is sent over the channel connecting the main processor (MCU) with the Bluetooth modem ("MCU-to-Bluetooth channel"). After the Bluetooth modem has received the last bit of this message over the serial line, the command will be processed. At some (unknown) point in time, the actual readout of the clock will be performed, before the reply message is generated and sent to the main processor via the "Bluetooth-to-MCU channel".

We could use a typical round-trip-time measurement to compute the clock offset between the system clock and the Bluetooth clock. Here, we would use the system clock to measure the point in time t_1 when the last bit of the command has been sent and the point in time t_2 when the first bit of the reply has been received. The clock offset could then be approximated by the returned Bluetooth clock value t_3 minus $(t_1 + t_2)/2$.

Unfortunately, the reply from the Bluetooth module may be significantly delayed if the Bluetooth module has received a data message via radio and is sending this message to the main processor as illustrated in Fig. 3 (b), such that the communication channel ("Bluetooth-to-MCU channel") is blocked. The resulting highly variable delay $t_3 - t_2$ results in significant errors when using the above approach.

Likewise, processing of the command in the Bluetooth modem may be delayed if the modem is busy receiving data. However, we performed experiments that confirmed that the offset $t_3 - t_1$ is much more stable than $t_3 - t_2$ even under heavy communication load. Therefore, we use $t_3 - t_1$ as the offset between the system clock and the Bluetooth clock in our protocol. Still, there are occasional outliers that are substantially larger than the average clock offset. To remove these outliers, we apply a simple median filter. Instead of using $t_3 - t_1$ as the clock offset, the median filter remembers the last n measured offsets and returns the median value among them. The evaluation in Sect. VII will show the effectiveness of this approach for small time windows.

Note that the above approach results in a systematic error which equals the time interval between transmission of the last bit of the command message and the actual readout of the Bluetooth clock. Assuming that the error is mainly a function of the Bluetooth implementation, it will cancel out in the end-to-end synchronization protocol illustrated in Fig. 2 if both sender and receiver use identical Bluetooth hardware. The reason for this is that we transform each time stamp twice between system clock and Bluetooth clock: once at the sending node and once at the receiving node. Since these transformations are in reverse direction (i.e., system clock to Bluetooth clock on the sending node and Bluetooth clock to system clock on the receiving node), the error will cancel out.

V. Offset between Bluetooth Clocks

To obtain the offset Δ between the Bluetooth clocks of two connected nodes, the `HCI_Read_Clock_Offset` command can be used. However, the clock offset returned by this command only contains bits 2-16 of the clock difference. We therefore need to reconstruct the complete clock difference. For this, a local Bluetooth time stamp is sent over an established connection as shown in Fig. 4. The method for reconstructing the missing bits of Δ described below assumes that the transmission latency of the message is less than the clock offset range of 40.96 s, which is a reasonable assumption.

In the following, we will refer to the difference between two Bluetooth clocks as *clock difference* Δ and refer to the lower part of this clock difference Δ returned by Bluetooth commands as *clock offset*. Also, we assume that all variables hold integer multiples of a Bluetooth clock tick of 0.3125 ms.

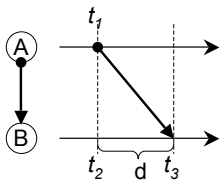

Fig. 4. Sending a Bluetooth Time Stamp from Node A to Node B.

We can therefore express the assumption that d is below 40.96 s as follows (see Sect. III):

$$0 \leqslant d < 2^{17} \tag{1}$$

In the following we will use the notation V_{b_c} to refer to the integer value of V where bits with index b-c in the binary representation are preserved and all others are set to zero. The least significant bit has index 0. For example, $15_{1\_2} = 6$.

From the `HCI_Read_Clock_Offset` command, we obtain bits 2-16 of the clock difference Δ:

$$\Delta_{2\_16} = (CLKslave - CLKmaster)_{2\_16}$$

Let us assume that node A in Fig. 4 is in the slave role and sends the current value t_1 of its clock CLKslave to node B which records the time t_3 of its local clock CLKmaster at reception.

Node B can calculate an approximate clock difference Δ' as:

$$\Delta' := t_1 - t_3 \tag{2}$$

As t_3 represents the time when the message was sent plus the (unknown) transmission delay d, (2) can be reformulated as follows:

$$\Delta' = t_1 - (t_2 + d) = (t_1 - t_2) - d = \Delta - d \tag{3}$$

As we know Δ' and Δ_{2_16}, we can use (3) to calculate Δ_{17_27} as follows. Since $0 \leq d < 2^{17}$ by assumption, (3) implies that either $\Delta'_{17\_27} = \Delta_{17\_27}$ or $\Delta'_{17\_27} = \Delta_{17\_27} - 2^{17}$. $\Delta'_{17\_27} = \Delta_{17\_27}$ can only hold iff $\Delta'_{2\_16} \leq \Delta_{2\_16}$, otherwise it would follow that $\Delta' > \Delta$ in contradiction to (3). In summary, we can compute the missing bits of Δ using the following equation:

$$\Delta_{17\_27} = \begin{cases} \Delta'_{17\_27} & \text{if } \Delta'_{2\_16} \leqslant \Delta_{2\_16}, \\ \Delta'_{17\_27} + 2^{17} & \text{otherwise.} \end{cases} \tag{4}$$

If node A would have been in master mode, the following analogous equation has to be used:

$$\Delta_{17\_27} = \begin{cases} \Delta'_{17\_27} & \text{if } \Delta'_{2\_16} \geqslant \Delta_{16\_2}, \\ \Delta'_{17\_27} - 2^{17} & \text{otherwise.} \end{cases} \tag{5}$$

VI. Implementation

We implemented our time synchronization protocol on BTnode Rev. 3 nodes [4]. They basically consist of an ATMEL ATmega128 8-bit microcontroller, 256 KB SRAM and a Zeevo ZV-4002 Bluetooth module. The operating system running on these devices is BTnut which is an extension of Ethernut Nut/OS [9]. BTnut provides parts of the standard Bluetooth stack (HCI, L2CAP, and RFCOMM) together with our own implementation of Scatternet formation and routing using L2CAP connectionless data packets. [2] provides details on the Scatternet formation and routing protocols.

Unfortunately, the implementation of the commands `HCI_Read_Clock_Offset` and `HCI_Read_Clock` on the Zeevo Bluetooth module suffers from several bugs, which we had to work around when implementing our protocol. While the `HCI_Read_Clock_Offset` command was always present in the Bluetooth specification, the `HCI_Read_Clock` command was introduced by Bluetooth specification 1.2 in 2003. The Zeevo Bluetooth module on the BTnode was sold as a

pre-1.2 version. Although it supports the required commands for our time sync approach, some commands do not follow the specification or are not properly implemented. We provide details on the bugs and our work-arounds below.

A. `HCI_Read_Clock`

This command is supposed to return the value of the local Bluetooth clock or the value of the Bluetooth clock of a connected Piconet master (depending on the command parameters). However, our Zeevo modules always return the value of the local Bluetooth clock, the clock of the master could not be read. Further, the returned Bluetooth clock value was expressed as a multiply of 1.25 ms instead of the specified 0.3125 ms. Finally, we noticed occasional significant outliers when analyzing the computed offsets between system clock and Bluetooth clock (see Sect. IV). The reason for this is a bug in the the implementation of the read clock command, which returns the same Bluetooth clock reading twice in 1% of all cases. As the command execution takes about 10 ms in our configuration, two `HCI_Read_Clock` commands cannot have been issued and answered within the clock resolution of 1.25 ms. As it turned out, the second value was a duplicate of the first, and we resorted to reading the Bluetooth clock twice and using the second value only if it was different from the first.

B. `HCI_Read_Clock_Offset`

This command is supposed to return the current clock offset to a connected device. However, once a connection between two BTnodes has been established, the offset returned by this command never changes even though the two clocks drift apart. Only after closing and re-opening a connection the returned clock offset changed. That is, the command always returns the clock offset at the time when the connection was established.

As periodic disconnects are no option in many applications, another way to read the clock offset was needed. After some experimentation, we noticed that a connected device is also reported by an Bluetooth inquiry. This comes as a surprise as inquiries are generally used to find new devices and not to collect information about already connected neighbours. As inquiries also return the current clock offset, periodic inquiries can be used to update the clock offsets to connected neighbours. However, the clock offset returned by an inquiry is slightly different from the value returned by `HCI_Read_Clock_Offset`. The latter always returns CLKslave - CLKmaster no matter if the Bluetooth device is a master or a slave. The inquiry returns the same value only if the invoking Bluetooth device is a master. If invoked by a slave, the returned value is $2^{17}-$(CLKslave - CLKmaster).

VII. Evaluation

To evaluate our approach, we first study the accuracy of synchronization between system and Bluetooth clock in the presence of parallel data transmissions. Then we measure the synchronization error within an 8 node multi-hop Scatternet. We also show some preliminary results of this approach running on a linux laptop with a built-in Bluetooth module.

A. Reading the Bluetooth Clock under Load

We evaluate the accuracy of the approach to measure the offset between the system clock and the Bluetooth clock (see Sect. IV). We consider a Bluetooth Piconet of two node A and B. The speed of the serial connection between the main processor and the Bluetooth modem was set to 115200 baud. After B has connected to A, A will start reading its Bluetooth clock 1000 times at regular intervals. A will then signal B to start sending data messages. A will discard these messages, but continues to read out the Bluetooth clock. After 1000 readouts, A signals B to stop sending data. A continues to read out its Bluetooth clock for another 1000 times.

At each readout, A records the time of the system clock after the last bit of the command has been sent

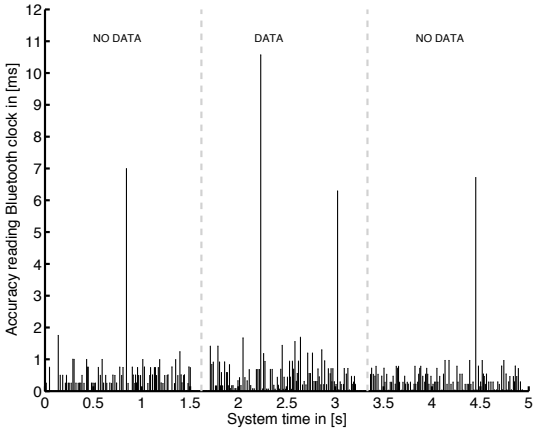

Fig. 5. Accuracy of Bluetooth clock read out with and without data traffic.

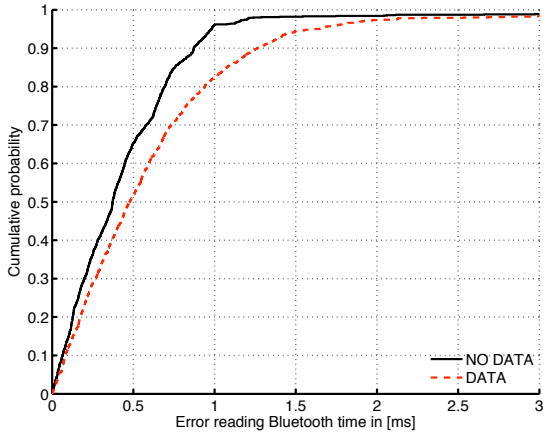

Fig. 6. ECDF of accuracy with and without data traffic.

to Bluetooth and the returned value of the Bluetooth clock. That is, A records a data point (system clock, Bluetooth clock) for each readout. We then fit a line to these 3000 data points using linear regression. This line approximates the ground-truth offset between the two clocks. We consider the distance of a data point from the regression line as a measure of the accuracy of that data point.

Fig. 5 shows the accuracy for the first 100 readings of each block (no data, data, no data). The maximum error on accuracy was 15 ms. To analyze the impact of data transmission, we we plotted the empirical cumulative distribution function (ECDF) for ”data“ and ”no data“ separately as shown in Fig. 6. The distribution for ”no data“ shows smaller errors than for the ”data“ section, but this results mainly from having less outliers compared to ”the ’data“ section.

To further reduce the reading error, we employed a median filter as described in Sect. IV, which outputs the median of the last n samples. Fig. 7 shows the average and the maximal error for different values of n over all samples. For all 3000 samples with and without parallel data traffic, the maximal error for n=5 is below 2 ms.

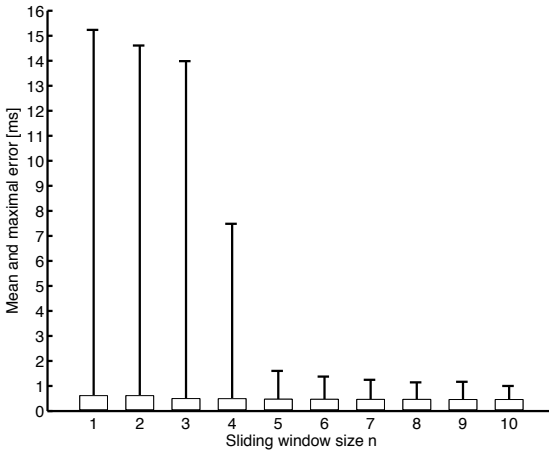

Fig. 7. Mean and maximal clock reading error for median filter with window size n.

B. *Scatternet Synchronization Error*

As the Bluetooth clocks run unsynchronized and the clock offset is only available with a 1.25 ms resolution, we expect the error between two connected nodes to be less than this value. We set up 8 BTnodes in a chain topology in which each node but the end nodes act as a master/slave bridge, effectively forming a Scatternet of 7 interconnected Piconets. All nodes are connected to an 8-channel logic analyzer with 1 us time resolution. After the Scatternet has been established, the first node in the chain periodically sends a time-stamped message along the chain of nodes, applying the synchronization algorithm described in Sect. II to

synchronize the time stamp. When receiving the message, a node will set a timer to expire at system time $t+C$, where t is the (synchronized) time stamp contained in the message and C is a constant offset. When the timer expires, the node toggles the I/O pin which is connected to the logic-analyzer. Ideally, all nodes should toggle their pins at exactly the same point in time. However, synchronization errors will cause nodes to toggle their pins at slightly different points in time. Using the logic analyzer, we measure the time between the first node in the chain toggling its pin and every other node in chain toggling its pin. This amount of time is the synchronization error.

In the experiment, the BTnodes update their clock offsets every 5 minutes using a Bluetooth discovery. The experiment runs for 2 hours, resulting in 712 accuracy measurements for all 8 nodes. Figure 8 shows the average and maximal synchronization error for each node. The mean error for the last node in the chain is 5.47 ms $\pm$2.25 ms and the maximum error is 11.35 ms.

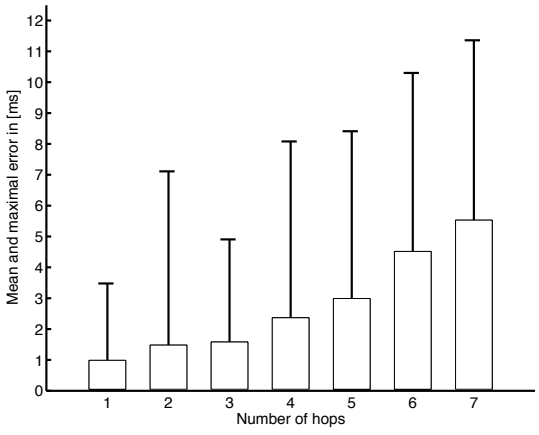

Fig. 8. Time synchronisation error for a 7-hop Scatternet.

C. Other Platforms

The BTnodes used in the evaluation are only a single example for devices which support our time synchronization protocol. Often, a more powerful device such as a laptop is used to store collected data and to provide a time reference for a whole Scatternet. For our protocol to work, a device needs to support the `HCI_Read_Clock` and `HCI_Read_Clock_Offset` commands which are available, e.g., in the BlueZ [3] Linux Bluetooth Stack. We tested an Apple PowerBook 12" with an embedded Cambridge Silicon Radio (CSR) chipset connected over USB, running the Ubuntu 6.10 Linux distribution with the default 2.6.17 kernel. Both commands are available on the CSR Bluetooth module. However, the `HCI_Read_Clock_Offset` only works correctly, if the device is in a slave role. In master role, the clock offset was not updated similar to the bug with the Zeevo module (see. Sect. VI-B). We repeated the read Bluetooth clock test and calculated the accuracy as in Sect. VII-A. In this configuration without median filter, the maximal error on accuracy was 4 ms. The mean error was 0.15 ms with a standard deviation of $\pm$ 0.24 ms.

VIII. Related Work

There exists a large body of work on time synchronization in wireless networks and in particular wireless sensor networks [12]. However, all of these approaches send explicit synchronization messages, whereas our approach uses the synchronization primitives provided by Bluetooth to build a global synchronization protocol, thus minimizing additional communication overhead. Protocols based on the exchange of explicit synchronization messages would suffer from highly variable messages latencies of typical Bluetooth implementations. In [10], for example, round trip times in a simple two-node network varied between 30 and 230 ms with an 76 ms average. In [8], round-trip times between 20 and 120 ms have been observed a similar setup.

Specific protocols for time synchronization with Bluetooth are rare. [5] describes an experiment in which a Piconet master sends broadcast messages to synchronize slaves among each other. As the broadcast message arrives almost concurrently at all slaves, the reception event can be used to accurately synchronize the clocks of slaves. They report very good results for the accuracy among slaves in the order of 10 us. However, this approach can only synchronize slaves among each other, but not the

master with the slaves as only the master can send broadcast messages. Hence, a Bluetooth Scatternet cannot be synchronized with this approach. A similar technique is used in [1] to synchronize the slaves of a Piconet. However, in addition they modify the firmware of the Bluetooth module to precisely measure the point in time when the master sent the broadcast message. Using this information, they can also synchronize the master with the slaves. Overall, they obtain a precision of few micro seconds in a single Piconet. While their approach could be extended to Scatternets, they do not consider this option. However, modifying the Bluetooth firmware is often impossible and/or impractical.

802.11 also uses an internal clock to control medium access and specifies a Timing Synchronization Function (TSF) to synchronize the clocks of different nodes. In both infrastructure and ad hoc modes, 802.11 only supports single-hop networks. Therefore, TSF is a rather simple protocol. In infrastructure mode, the base station regularly broadcasts time-stamped beacon messages and receivers adjust their clocks to the received time stamp. In ad hoc mode, every node broadcasts such beacon messages and receivers adjust their clocks to the sender with the latest time stamp. Huang et. al. [7] propose a simple extension to this procedure to improve the scalability of TSF.

IX. Conclusion

We presented a practical time synchronization protocol for multi-hop Bluetooth networks, so-called Scatternets. The protocol builds upon two Bluetooth commands to read the current value of the internal Bluetooth clock and the offsets of this clock to the Bluetooth clocks of network neighbors. The implementation of this protocol on the BTnode platform required a number of workarounds due to bugs in the Bluetooth firmware. We evaluated the protocol on a 7-hop network of BTnodes and found a synchronization accuracy of 12 ms. The proposed protocol requires only minimal additional communication overhead, avoids the complexity of clock disciplining, and is robust to network topology changes because it is fully distributed.

X. Acknowledgements

We would like to thank Qin Yin for implementing a first version of our Protocol and pointing out the Zeevo bug with reading the clock offsets described in Sec. VI-B.

The work presented in this paper was partially supported by the by the Swiss National Science Foundation under grant number 5005-67322 (NCCR-MICS).

References

[1] Lucia Lo Bello and Orazio Mirabella. Clock synchronization issues in bluetooth-based industrial measurements. In *Workshop on Factory Communication Systems*, June 2006.

[2] J. Beutel, M. Dyer, L. Meier, and L. Thiele. Scalable topology control for deployment-sensor networks. In *Fourth Internation Conference on Information Processing in Sensor Networks (IPSN '05)*, April 2005.

[3] BlueZ. Official linux bluetooth stack. www.bluez.org.

[4] BTnodes. A distributed environment for prototyping ad hoc networks. www.btnode.ethz.ch.

[5] Robert Casas, Héctor J. Garcia, Álvaro Marco, and Jorge L. Falco. Synchronization in wireless sensor networks using bluetooth. In *3rd International Workshop on Intelligent Solutions in Embedded Systems*, May 2005.

[6] Bluetooth Special Interest Group. Specification of the bluetooth system, core package version 1.2. www.bluetooth.org, 12 2003.

[7] L. Huang and T.-H. Lai. On the scalability of ieee 802.11 ad hoc networks. In *ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002)*, June 2002.

[8] Felix Michel and Philippe Wüger. Angewandte Uhrensynchronisation auf BTnodes. Term Project, D-ITET, ETH Zurich, 2005.

[9] Nut/OS. Embedded ethernet. www.ethernut.de.

[10] Olof Rensfelt, Richard Gold, and Lars-Ake Larzon. Lunar over bluetooth. In *4th Scandinavian Workshop on Wireless Ad-hoc Networks (ADHOC '04)*, 2004.

[11] Matthias Ringwald, Kay Römer, and Andrea Vitaletti. Passive inspection of wireless sensor networks. In *Third International Conference on Distributed Computing in Sensor Systems (DCOSS 2007)*, 2007.

[12] Kay Römer, Philipp Blum, and Lennart Meier. Time synchronization and calibration in wireless sensor networks. In Ivan Stojmenovic, editor, *Handbook of Sensor Networks: Algorithms and Architectures*. John Wiley & Sons, September 2005.