Supplementary File

Zhenyu Yan, Rui Tan, *Senior Member, IEEE,* Yang Li, *Member, IEEE,* and Jun Huang, *Member, IEEE*

This document includes the supplemental materials for the paper titled "Wearables Clock Synchronization Using Skin Electric Potentials".

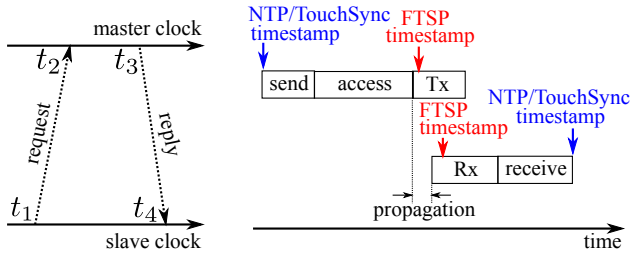
APPENDIX A
NETWORK TIME PROTOCOL AND ITS PERFORMANCE

A.1 NTP Principle and Packet Timestamping

Many clock synchronization approaches adopt the principle of NTP, which is illustrated in Fig. 1(a). A *synchronization session* consists of the transmissions of a request packet and a reply packet. The t_1 and t_4 are the slave's clock values when the request and reply packets are transmitted and received by the slave node, respectively. The t_2 and t_3 are the master's clock values when the request and reply packets are received and transmitted by the master node, respectively. Thus, the round-trip time (RTT) is $\mathrm{RTT} = (t_4 - t_1) - (t_3 - t_2)$. Based on a *symmetric link assumption* that assumes identical times for transmitting the two packets, the offset between the slave's and the master's clocks, denoted by δ_{NTP}, is estimated as $\delta_{NTP} = t_4 - \left(t_3 + \frac{\mathrm{RTT}}{2}\right)$. Then, this offset is used to adjust the slave's clock to achieve clock synchronization. Under the above principle, non-identical times for transmitting the two packets will result in an error in estimating the clock offset. The estimation error is half of the difference between the times for transmitting the two packets.

We use Fig. 1(b) and the terminology in [1] to explain how the timestamps (e.g., t_3 and t_4) are obtained in NTP and existing WSN clock synchronization approaches. The *send time* and the *receive time* are the times used by the OS to pass a packet between the synchronization program and the MAC layer at the sender and receiver, respectively. They depend on OS overhead. The *access time* is the time for the sender's MAC layer to wait for a prescribed time slot in time-division multiple access (TDMA) or a clear channel in carrier-sense multiple access with collision avoidance (CSMA/CA). It often bears the highest uncertainty and can

- *Z. Yan and R. Tan are with School of Computer Science and Engineering, Nanyang Technological University, Singapore, 639798. E-mail: {zyan006,tanrui}@ntu.edu.sg*
- *Y. Li is with Additive Manufacturing Institute, College of Mechatronics & Control Engineering, Shenzhen University, Shenzhen, China 518060. This work was completed while Yang Li was with Advanced Digital Sciences Center, Illinois at Singapore. E-mail: yli@szu.edu.cn*
- *J. Huang is with Center for Energy Efficient Computing and Applications, Peking University, Beijing, 100871. E-mail: jun.huang@pku.edu.cn*



(a) NTP principle. (b) Packet timestamping for synchronization.

Fig. 1. NTP principle and packet timestamping.

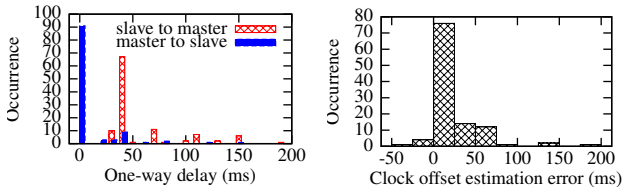


Fig. 2. Performance of NTP over a BLE connection.

be up to 500 ms [1]. The transmission (Tx) and reception (Rx) times are the physical layer processing delays at the sender and receiver, respectively. The *propagation time* equals the distance between the two nodes divided by the speed of light, which is generally below 1 μs.

As illustrated in Fig. 1(b), NTP timestamps the packet when the packet is passed to or received from the OS. Thus, the packet transmission time used by NTP is subjected to the uncertain OS overhead and MAC. Therefore, as measured in Section A.2, NTP over a Bluetooth connection can yield nearly 200 ms clock offset estimation errors. To remove these uncertainties, FTSP uses MAC-layer access to obtain the times when the beginning of the packet is transmitted/received by the radio chip. As the propagation time is generally below 1 μs, FTSP simply estimates the clock offset as the difference between the two hardware-level timestamps. Thus, the two-way scheme in Fig. 1(a) becomes non-essential for FTSP.

A.2 Performance of NTP over BLE Connection

As our objective is to devise a new clock synchronization approach that uses application-layer timestamping as NTP does, this section measures the performance of NTP to

provide a baseline. We implement the NTP principle described in Section A.1 on Flora, a wearable platform. Our setup includes a Flora node and a Raspberry Pi single-board computer that perform the NTP slave and master, respectively. They are connected via Bluetooth Low Energy (BLE). More details of the Flora setup can be found in Section xx of the paper. Fig. 2(a) shows the distributions of the one-way application-layer communication delays over 110 NTP sessions. The slave-to-master delays are mostly within $[40, 50]$ ms, with a median of 42 ms and a maximum of 376 ms (not shown in the figure). As specified by the BLE standard, the master device pulls data from a slave periodically. The period, called *connection interval*, is determined by the master. The slave needs to wait for a pull request to transmit a packet to the master. In the Raspberry Pi's BLE driver (BlueZ), the connection interval is set to 67.5 ms by default. As the arrival time of a packet from the slave's OS is uniformly distributed over the connection interval, the expected access time is $67.5/2 = 33.75$ ms. This is consistent with our measured median delay of 42 ms, which is about 8 ms longer because of other delays (e.g., send and receive times). The exceptionally long delays (e.g., 376 ms) observed in our measurements could be caused by transient wireless interference and OS delays. For the master-to-slave link, the delays are mostly within $[0, 10]$ ms, with a median of 8 ms and a maximum of 153 ms. A BLE slave can skip a number of pull requests, which is specified by the *slave latency* parameter, and sleep to save energy. Under BlueZ's default setting of zero for slave latency, the slave keeps awake and listening, yielding short master-to-slave delays.

The asymmetric slave-to-master and master-to-slave delays will cause significant errors in the NTP's clock offset estimation. At the end of each synchronization session, the RPi computes this error as $\delta_{NTP} - \delta_{GT}$, where δ_{NTP} and δ_{GT} are NTP's estimate and the ground-truth offset, respectively. Fig. 2(b) shows the distribution of the errors. We observe that 28% of the errors are larger than 25 ms. The largest error in the 110 NTP sessions is 183 ms. Such an error profile does not well meet the ms accuracy requirements of many applications [2], [3], [4]. Though it is possible to calibrate the average error to zero by using prior information (e.g., the settings of the connection interval and slave latency), the calibration is tedious, nonuniversal, and incapable of reducing noise variance.

APPENDIX B
TOUCHSYNC PSEUDOCODE

The pseudocode programs running at the TouchSync slave and master are presented in Algorithm 1 and 2, respectively.

Algorithm 1 Slave's pseudocode for a synchronization process

```
1:  Global variables: t1, t4, δ's solution space Δ = ∅, session index k = 0
2:
3:  command start_sync_session() do
4:      k = k + 1
5:      t1 = read_system_clock()
6:      send message request = { } to master
7:  end command
8:
9:  event reply1 received from master do
10:     t4 = read_system_clock()
11:     if SEP signal strength is good:
12:         wait till SEP data covering t1 and t4 are available
13:         run SEP signal processing pipeline in Section 5.2
14:     else:
15:         generate internal periodic signal[1]
16:     endif
17:     compute φ1 and φ4
18: end event
19:
20: event reply2 received from master do
21:     compute θq and θp using Eqs. (1) and (2).
22:     RTT = (t4 − t1) − (reply2.t3 − reply2.t2)
23:     solve Eq. (7), Δ' denotes the set of all possible solutions for δ
24:     if k == 1: Δ = Δ'; else: Δ = Δ ∩ Δ'; endif
25:     if Δ has only one element δ: use δ to adjust clock;
26:     else: start_sync_session() // start a new synchronization session
27: end event
```

Algorithm 2 Master's pseudocode for a synchronization process

```
1:  event request received from slave do
2:      t2 = read_system_clock()
3:      ... // execute other compute tasks
4:      t3 = read_system_clock()
5:      send message reply1 = { } to slave
6:      if SEP signal strength is good:
7:          wait till SEP data covering t1 and t4 are available
8:          run SEP signal processing pipeline in Section 5.2
9:      else:
10:         generate internal periodic signal[2]
11:     endif
12:     compute φ2 and φ3
13:     send message reply2 = {t2, t3, φ2, φ3} to slave
14: end event
```

APPENDIX C
COMPUTATION OVERHEAD

To understand the overhead of TouchSync, we deploy our TinyOS and Arduino implementations to Zolertia's Z1 motes and Floras, respectively. The Z1 mote is equipped with an MSP430 MCU (1 MHz, 8 KB RAM) and a CC2420 802.15.4 radio. Both implementations sample SEP at 333 Hz. On Z1, we configure the length of the circular buffer defined in `touchsync.h` to be 512. Thus, this circular buffer can store 1.5 seconds of SEP data. This is sufficient, because the time periods $[t_2, t_3]$ and $[t_1, t_4]$ that should be covered by the SEP signal segments to be retrieved from the circular buffer and processed by the master and slave are generally a few ms and below 100 ms, respectively. On Flora, we

1. Line 15 deals with a situation where the SEP signal is too weak. The detailed explanation can be found in Section 6 of the paper.

2. Line 10 deals with a situation where the SEP signal is too weak. The detailed explanation can be found in Section 6 of the paper.

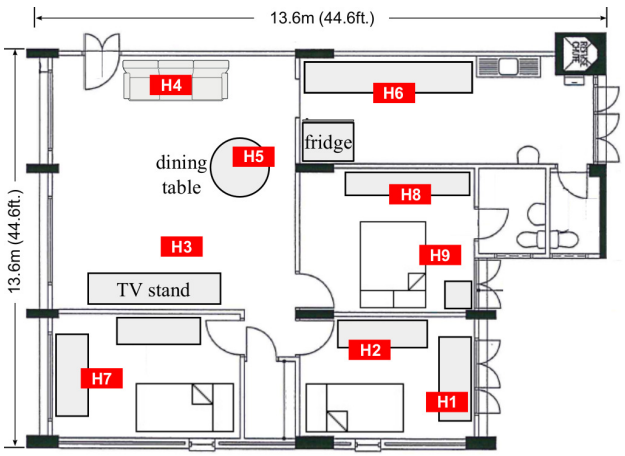


Fig. 4. Home floor plan with test points marked.

configure the circular buffer length to be 400 and redefine its data type such that TouchSync can fit into Flora's limited RAM space of 2.5 KB. Table 1 tabulates the memory usage of TouchSync and the computation time of different processing tasks. On Z1, a total of 421 ms processing time is needed for a synchronization session. The BPF uses a major portion of the processing time. Flora cannot adopt BPF because of RAM shortage. It uses MRF instead, which consumes much less RAM and processing time. Z1 and Flora are two representative resource-constrained platforms. The successful implementations of TouchSync on them suggest that TouchSync can also be readily implemented on other more resourceful platforms.

TABLE 1
Storage and compute overhead of TouchSync.

Platform	**Memory use** (KB)		**Processing time** (ms)			
	ROM	RAM	BPF/MRF	ZCD	PLL	IAS
Z1	10	5	364	9	48	1
Flora	17	1.9*	1.3	3	15	0.8

* Estimated based on buffer lengths.

APPENDIX D
FLOOR PLANS OF EVALUATION

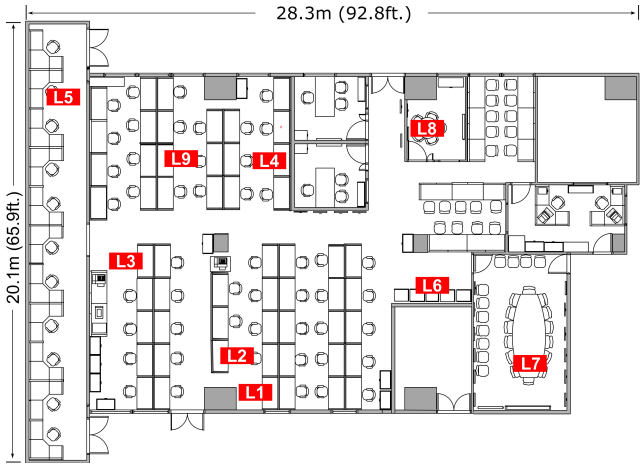


Fig. 3. Laboratory floor plan with test points marked.

Fig. 3 and Fig. 4 show the floor plans of laboratory and home, respectively. We arbitrarily select nine test points for each room, marked by "Lx" and "Hx" in figures.

REFERENCES

[1] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *SenSys*, 2004.
[2] M. Dinescu, J. Mazza, A. Kujanski, B. Gaza, and M. Sagan, "Synchronizing wireless earphones," Apr. 7 2015, US Patent 9,002,044. [Online]. Available: https://www.google.com/patents/US9002044
[3] K. Lorincz, B.-r. Chen, G. Challen, A. Chowdhury, S. Patel, P. Bonato, and M. Welsh, "Mercury: a wearable sensor network platform for high-fidelity motion analysis," in *SenSys*, 2009.
[4] F. Mokaya, R. Lucas, H. Noh, and P. Zhang, "Burnout: a wearable system for unobtrusive skeletal muscle fatigue estimation," in *IPSN*, 2016.