MobileTimeSync - An Android App for Time Synchronization for Mobile Construction Assessment

Maik Benndorf, Marika Kaden,
Frederic Ringsleben and Thomas Haenselmann
Faculty of Applied Computer
Sciences & Biosciences
University of Applied Sciences
Germany, Mittweida 09648
{benndorf, kaden1, ringsleb, haenselm}@hsmw.de

Eric Zuchantke
Faculty of Mathematics and
Computer Science
Friedrich-Schiller-Universität Jena
Germany, Jena 07743
eric.zuchantke@uni-jena.de

Martin Gaedke
Faculty of Computer Science
Technische Universität Chemnitz
Germany, Chemnitz 09111
gaedke@informatik.tu-chemnitz.de

***Abstract*—Modern smartphones offer more use-cases than (just) pure communication. These devices, equipped with several sensors, can be used as measuring devices, for example. In order to compare and interpret the sensor data obtained from multiple devices, precisely synchronized clocks are needed. Since all clocks are subject to inaccuracies, a synchronization method is necessary. In this article, we propose MobileTimeSync - an Android app for time synchronization. We suggest how to align clocks based on ultrasound beacons to mutually take clock drift and skew into account. In addition, we will discuss the different kinds of time sources in Android smartphones. With our approach, we obtain a precision, which can be expected from traditional synchronization approaches such as Network Time Protocol but with much less effort.**

***Index Terms*—time synchronization; clock skew; clock drift; sensor networks; mobile devices; smartphones**

I. Introduction

Most computer clocks, e.g., those built into a smartphone, are subject to an inaccuracy [1]. This inaccuracy can be neglected in the general purpose of the smartphone. However, since these devices are equipped with several sensors, there are other use cases in which this inaccuracy cannot be neglected.

The following scenario illustrates our need for accurate clocks: In cases of natural disasters such as flooding, timely access to information about transport infrastructure is crucial for the first-aider as well as for the affected people. In particular, bridges could be fit for traffic, partially usable or be destroyed entirely. While in many cases a visual inspection is sufficient to assess the accessibility of roads, for the assessment of bridges further information is needed. Assessing the construction's integrity can be accomplished by the analysis of the bridge's eigenfrequencies. Therefore, a proprietary, expensive Vibration Measurement Systems (VMS) is necessary. However, in many cases, such a device is not available in a timely manner in the areas struck by a disaster. In [2], [3], we were able to show how this VMS can be replaced by low-cost, off-the-shelf smartphones that can be applied to perform an ad-hoc assessment of a bridge. Our current research is now focusing on the issue of locating the damage on the bridge. For this purpose, several smartphones have to be placed on the surface of the bridge, and their measured values have to be aligned finally. Therefore, the clocks of these devices must be synchronized.

At the Sensorcomm conference we presented an approach for acoustic time synchronization of smartphones [1]. The basic idea for this synchronization approach comes from sports. With a 100 m sprint, eight athletes stand in starting blocks. The race begins with an external signal which is the same for each athlete. This idea was adapted for the time synchronization of smartphones. For that purpose, an external acoustic signal is generated and sent out. The smartphones microphone receives this signal and the timestamp of the arriving of the signal is used for aligning different devices. If several of those signals were sent out with this procedure, it is possible to determine the clock skew and clock drift.

One issue of our previous work was the source of the sound signal. In our experiments in [1], we used a choke in the same way as it is applied in athletic sports. In our recent research, we create an Android app (see Fig. 1) for this purpose. This app serves as both a sender (sound source) and a receiver. Moreover, the app facilitates the use of time synchronization on these devices and will be presented in this paper.

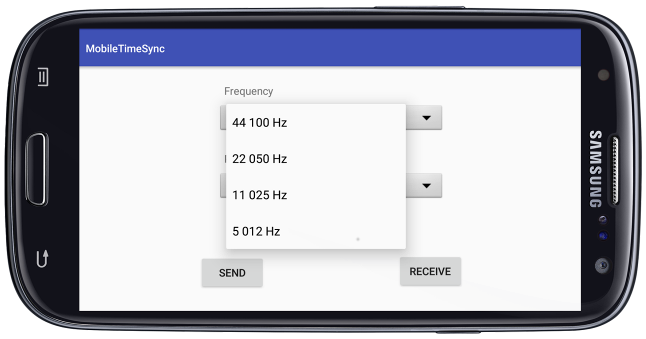


Fig. 1. Screenshot of the Android App for Time Synchronization

Therefore, this article is organized as follows: Section II is devoted to provide fundamentals and give an overview of related work in the field of synchronizing methods. Our application and methodology is described in Section *Methodology*

(Section III). Some experiments and their results are presented in Section IV. A summary and outlook are drawn in Section V.

II. Fundamentals and Related Work

The addressed problem is related to the time synchronization in sensor networks. A lot of research has been done in this area. There are several methods for synchronizing physical clocks. These methods can be classified as "internal" and "external" synchronization. For an internal clock synchronization, all nodes accept the time of a reference node in the network [4]. In external clock synchronization, the value is taken from an external clock source, such as a common time service, e.g., Network Time Protocol NTP or the Global Positioning System (GPS) [5].

A. Synchronization Methods

In this section, some of the most common synchronization methods are shortly discussed.

1) Synchronization by NTP: NTP is a protocol for synchronizing computer clocks using a set of distributed servers around the world. This protocol is also known as Simple Network Time Protocol (SNTP). It is built on top of the User Datagram Protocol (UDP) [6]. Fig. 2 shows the process of an NTP based time synchronization. The protocol was announced with a precision in the range of nanoseconds (ns) [7] [8]. This protocol has been utilized in numerous clients for several years. Juda Levine [9] reports in 2011 about 5×10^9 requests per day. The accuracy of the protocol and the related assets have been studied in numerous works [10]–[12]. The network latency has a significant impact on the accuracy. Zhao et al. [11] evaluated the accuracy with less than 10 ms under Local Area Network (LAN) condition and less than 100 ms under Internet conditions. As a reference for their evaluations, they used the time of the GPS.

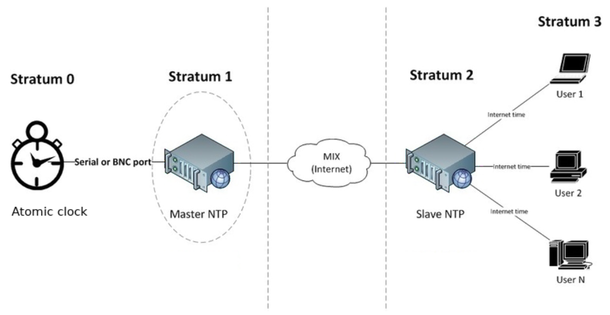


Fig. 2. Overview of time synchronization via NTP [13]

2) Synchronization by GPS: The Global Positioning system, was designed and is still under the control of the United States Department of Defense. Nevertheless, it is also freely accessible by anyone. The system consists currently of 32 operating satellites that are orbiting the earth at an altitude of approximately $20,000$ km. Every satellite contains multiple atomic clocks that support very precise timing data [14]. For determining the position, the receiver needs signals from at least three of these satellites. The position of the receiver can be calculated by the difference between the signal sent and received by the receiver. With this ability to receive very accurate data from multiple sources an accurate time can be obtained. In 2015, Mazur et al. [14] determined the accuracy of such time synchronizations within up to billionth of a second. This system can be accessed anywhere in the world and has very high accuracy. But it requires a direct line of sight to the satellite, and an initial connection takes a long time in some cases.

3) Time Synchronization on Mobile Devices: One recent work in this area is provided by Sridhar et al. In [15] they proposed Cheepsync - a synchronization service for Bluetooth Low Energy (BLE) Broadcasters. Cheepsync consists of a custom transmitter platform and an Android smartphone as the receiver. In order to overcome the non-deterministic delays, they used a common BLE chip. Since the transmission command directly addresses the hardware, they were able to generate a low-level timestamp in the range of 10 μs.

Another approach is proposed by Lazik et al. [16]. They used ultrasonic beacons to synchronize the time on mobile devices. Therefore, they built up a network with one network master. The master is connected to a GPS receiver and transmits ultrasonic chirps in a frequency that is outside of the human hearing but still detectable by the microphone of smartphones. They reported that the devices could be synchronized with an average accuracy of 720 μs. At the beginning of their experiments, they investigated devices with Android and iOS. They reported a high-level jitter on the Android device (in the order of milliseconds and higher) and chose the iOS devices for the rest of their experiments. The jittering is justified by Android's task scheduler. Within this work, they also benchmarked the NTP timing performance (on iOS). They ran their experiments with three different communication channels: Long Term Evolution Technologie (LTE), Wireless Local Area Network (WLAN), and one idle WLAN router that is directly connected to a Stratum 1 NTP server fed by a dedicated GPS clock. Using LTE they measured an average jitter of 47 ms, the average WLAN jitter is measured with 30 ms, and finally, the average jitter in the ideal case with the WLAN router connected to NTP Server is measured with 19.3 ms.

B. Available Time Sources on Android Devices

In Android devices different time sources are provided. A distinction can be made between the hardware timers which exist in every device, the timers offered by the Operating System (OS) and finally the timers provided by Java Virtual Machine. To use the correct time source for the synchronization, these sources are explained in more detail in the following.

1) The Hardware Timers: Accessing hardware timers on Android devices can be useful if the application only has to run on targeted hardware. It can be rewarding to do so because these timers are incredibly accurate, monotonic, non-decreasing clocks and cheap to read. On some ARM CPUs the overhead to read the time can be as small as only one CPU cycle (e.g., for Cortex A9 processors). But this high performance comes at a price as directly accessing the hardware requires an assembly code which is not portable, takes a lot of time to develop and is hard to read. Furthermore, if the program ought to be used on different devices most of them will have different hardware clocks. Errors can also occur when the device enters CPU saving or low-frequency states. All in all this method would only be suitable if the hardware is precisely targeted, which is not in our case [17].

2) Android Based Timers: Times from the Android operating system can be obtained by using clock sources from the Linux kernel. As in any system based on the 2.6 kernel, Android holds a linked list of all its clock sources sorted by rating and chooses the best available clock for all functions getting time from the kernel. One function to access these time sources is the `clock_gettime()`-posix function, which returns the time in a resolution of ns. However, this produces a small, acceptable overhead on ARM CPUs. If the `clock_gettime()`-function is running on a system which implements a syscall for it the overhead can also be removed. But as long as this method is only usable in native applications developed with C or with the usage of the Java Native Interface (JNI), it may not be suitable for particular use cases [17].

3) Java Timers: Because many Android applications are written in Java, the Java time functions are also considered for use as they are implemented in the Java Virtual Machine (JVM) and therefore are easily accessible from any Java application. There are four different types of clock functions offered that may prove useful [18]:

- `SystemClock.uptimeMillis()`
- `System.currentTimeMillis()`
- `System.currentThreadTimeMillis()`
- `SystemClock.elapsedRealtime()`

`uptimeMillis()` returns the system uptime since last boot in milliseconds (ms). The clock is guaranteed to be monotonic and is unaffected by most power saving mechanism, but it stops when the device enters deep sleep [18]. Therefore, this can be a reliable time source only if the measured time does not span deep sleep. However, due to the possibility of device going into 'deep sleep mode' this clock is not the best choice.

`currentTimeMillis()` returns the time in ms since the epoch. It can be used to measure daytime and dates but can be set by the user or the phone network. So the returned time may jump around [18]. In conclusion, the time provided by this function would be great to use in the application, but is too unreliable due to time changes.

`currentThreadTimeMillis()` returns milliseconds running in the current thread [18]. However, this can only be used for time measurement in combination with timestamps received from other time sources. Even if it is used that way, this will produce more overhead for time calculation then using uptimeMillis(). So, it is not considered as a helpful time source to compare timestamps among devices.

The functions `elapsedRealtime()` and `elapsedRealtimeNanos()` return the time since the system's last boot in ms and ns. Both functions include deep sleep and are guaranteed to be monotonic not to be influenced by the CPU entering power saving modes [18]. Thus, it can be useful for time comparison, but it should be noted that this is a relative time and cannot be used for comparison across different devices.

4) Advanced Time Calculation with Combined Methods: Another method for achieving accurate timestamps is called TrueTime [19]. The proposed algorithm takes advantage of the function `elapsedRealtime()` by requesting a timestamp from an NTP Server and storing this timestamp along with the `elapsedRealtime()` upon receiving the NTP timestamp in the device memory. Now every time a timestamp is needed the stored `elapsedRealtime()` can be subtracted from the current `elapsedRealtime()` and this difference is then added to the stored NTP timestamps creating the new current time in ms. Advantages of this method are that these timestamps are impervious to device clock changes as the NTP server dictates the time and the accuracy of this procedure above multiple devices is very high as long as every device receives the same timestamp via NTP. Also, the created times will be accurate until the device reboots, and it does not require a stable data connection. Disadvantages include the need for a data connection every time after reboot before the method can be used and the precision of the timestamp received that may vary depending on the latency of the network connection.

5) Summary: In summary, there are two types of times we can get from the system. First of all, there is the daytime which is dependent on which time the device was manually set to by the user or automatically.

Also, there are methods that will return timestamps measured relative to certain events like system boot. These methods deliver very accurate timestamps, but they are not comparable across different devices. But this is not necessary for our synchronization method. To compare these methods, we conducted an experiment with one of our phones. Therefore, we compared the Functions `System.currentTimeMillis()` and `SystemClock.elapsedRealtimeNanos()`. The mean difference between both the time sources is 0.1906 ms ($\pm$0.3919). The mean difference between ground truth and `elapsedRealtimeNanos()` is 9.020 ns ($\pm$7.779). And finally, the mean difference between ground truth and `currentTimeMillis()` is 9.088 ms ($\pm$7.79). The results can also be seen in Fig. 3. The function `elapsedRealtimeNanos()`

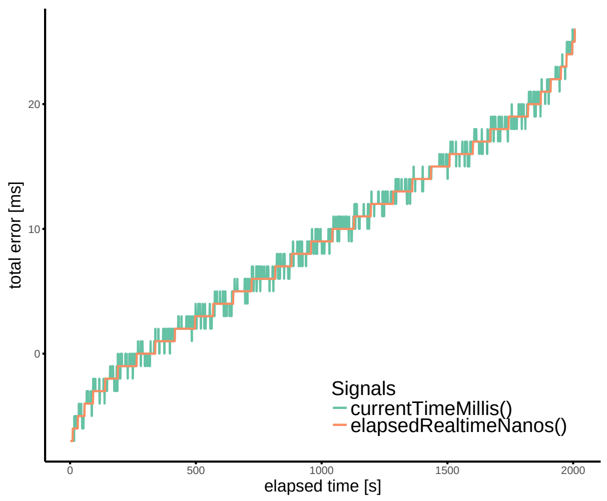


Fig. 3. Comparision of the two signals: (sorted) difference of ground truth to elapsedRealtimeNanos() (orange) and currentTimeMillis()(green)

performs only marginally better than the function `currentTimeMillis()`. Nevertheless, since only the function `elapsedRealtimeNanos()` is monotonic and not influenced by deep sleep, we decide to use this time source.

C. Problems

In combination with the time synchronization on Android devices, there are some problems which are mentioned below.

1) Clock-Skew and Drift: This is not an Android specific problem but rather a general problem of most of the computing devices. These devices are equipped with a hardware oscillator assisted computer clock. The frequency of the hardware oscillator determines the rate at which the clock runs [20]. This clock becomes inaccurate because the frequency varies.

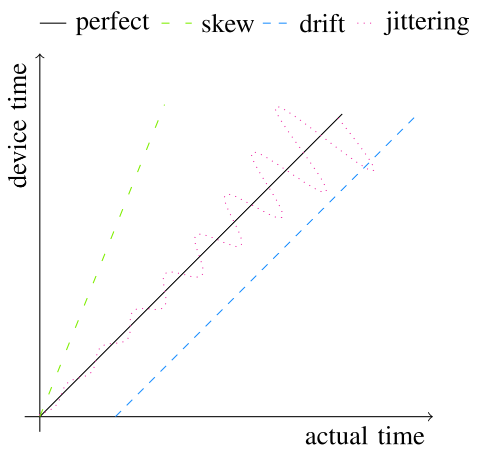


Fig. 4. The difference between a perfect clock, one with drift, one with a skew and one, as it is likely to occur in real [21].

Fig. 4 shows the difference between the clock drift - in this case, the clock is behind or ahead by a fixed offset, the clock skew - here the offset is growing with the time, and the jittering - in this case, the device clock is affected by internal (e.g., processor utilization) or external (e.g., temperature, humidity [22]) fluctuations. To show the appearance of the clock skew with an example, we present here a result of an experiment shown in [1]. In this experiment, we compared the local time on three different smartphones over 24 hours with the time values of the GPS sensor.

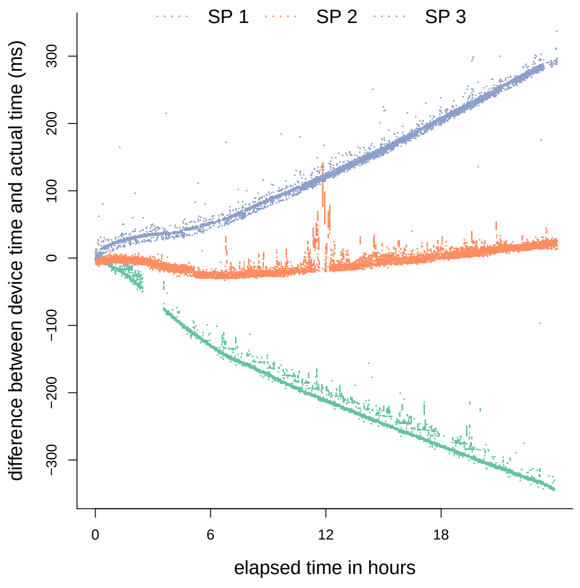


Fig. 5. The clocks skew of the devices within 24 hours compared to the timestamp given by GPS.

As it can be seen in Fig. 5 the offset of all three devices is growing over time. To keep these clocks in time, Zhenjiang Li et al. [23] use the flickering lights of fluorescent lights.

2) Jittering: Jittering is an Android specific problem. The duration of the execution of a function can vary. If the execution is heavily dependent on time measurement (like the comparison of sensor values in short amounts of time), these timing errors can lead to all kinds of possible misfunctions of the algorithm. So, it is advisable to look into the problems which can be caused by timing errors. The difference in the function execution time is called Jitter. According to an analysis from the University of Florida, the Jitter increases when polling device sensors at higher frequencies [24]. Data can be received from the device sensors via the sensor manager, which also creates timestamps for the data. However, this can unfortunately be received from nonmonotonic clock (creating problems for comparison of sensor data) as some hardware manufacturers do not fulfill the hardware specifications for sensor polling from Google. Sources for Jitter on Android devices are issues with the Android Compatibility Definition as it lacks hardware standards and requirements like minimizing Jitter are only ”should” criteria. Moreover, it can be constrained from the devices resources. These are mainly Memory (Garbage Collection / Thread Queues), Processor (Scheduler / Interrupt Handling) and Energy. To reduce resulting issues, developers can force the sensors to synchronize with the `elapsedRealtimeNanos()`-clock from the Android SDK to get real-time timestamps for the sensor data as this is not guaranteed for every device. In case of data received from multiple sensors, the study recommends to create timestamps for every sensor individually and synchronize the data afterwards [24]. Monotonic time series can only be reliably achieved through manual synchronization with a strictly monotonic clock as the requirements for Android. This also gives the possibility to synchronize sensors with non-monotonic clocks. Also, event delivery delay can be reduced by ensuring that the device has enough available resources during the runtime of the application. However, this can be difficult to achieve as many consumer devices may suffer from resource starvation even without an application trying to reserve as many resources as possible. The ideal proposed alternative for developers is to find polling frequencies for the sensor at which Jittering cannot occur or stays within a forcible range [24]. To achieve these, developers can refer to the Android Compatibility Definition (CDD) for the respective android version.

III. Methodology

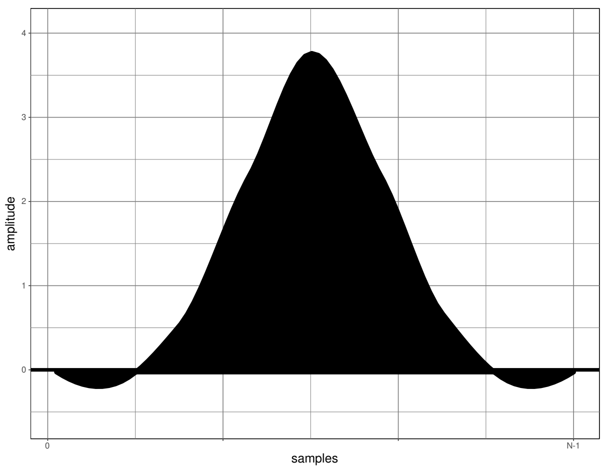


Fig. 6. Flat Window

In the previous section, we explained some synchronizing methods. Synchronization with a common time service will not be considered at the moment. This is because we can not make any assumptions about the availability, connection, and bandwidth. A prerequisite of synchronization among different devices is that they share the same network. To make the use case as simple as possible, it is not planned to create a network between those devices. Another option would be to synchronize via GPS. However, a direct view to the satellite must always be guaranteed. So, we decided to use acoustic synchronization. In [1] we created sound beacons with a choke from the sport. These acoustic beacons were recorded by the microphones of the smartphones. Afterwards, the peaks were isolated from the recorded data and the time at the peak was used to align the data of the various devices. In the continuation of this work, we developed an app. This app (see Fig. 1) serves both as generator and transmitter of the peak and the receiver.

A. Specifics

Due to the operating system, there are some specifics that need to be mentioned.

- The sound is encoded by 16 bit. Hence, values in the range from $-32,768$ to $32,767$ with a maximum signal to noise ratio of 96.33 dB can be achieved.
- The audio samples are provided by the operating system as chunks. Therefore, only the time when a package is received can be measured. The time for the samples is calculated from the sample rate by interpolation. The size of the chunks depends on the buffersize that is proposed by Android.

In order to reduce external influences, we disable most of the applications on the device.

B. Sound Synthesis

To generate the synthetic peak, we use the Flat top window function. Fig. 6 shows one peak generated with this function. Depending on the selected sample rate and the selected periodicity, this peak is embedded in an audio file one or more times. The standard length of this audio file is about 30 minutes.

The synchronization process is started by pressing the send button. This will play the audio file and can be recorded by all nearby devices.

C. Sound Recording

All devices involved in synchronization must be close to the transmitter (to minimize the influence of sound). On these devices, the synchronization process is started by pressing the Receive button. In our example, we have chosen a sample rate of $44,100$ Hz and a frequency of 1 Hz. This means that the peak was embedded in the audio file once per second. Fig. 7 shows (a) a single peak of the generated signal and (b) the sequence of several such peaks.

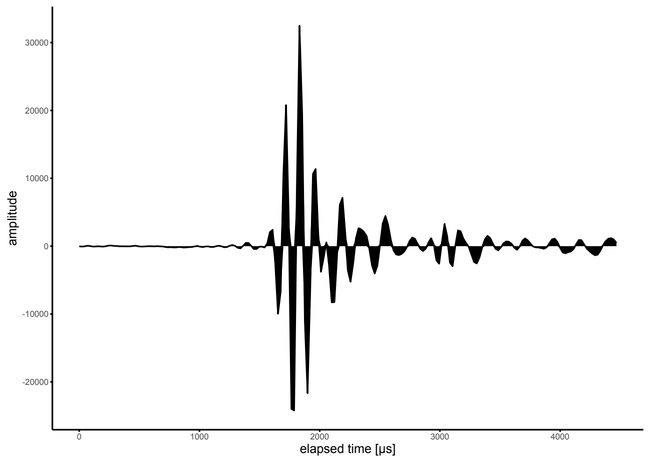
D. Processing

The following operations are performed in the background of the app.

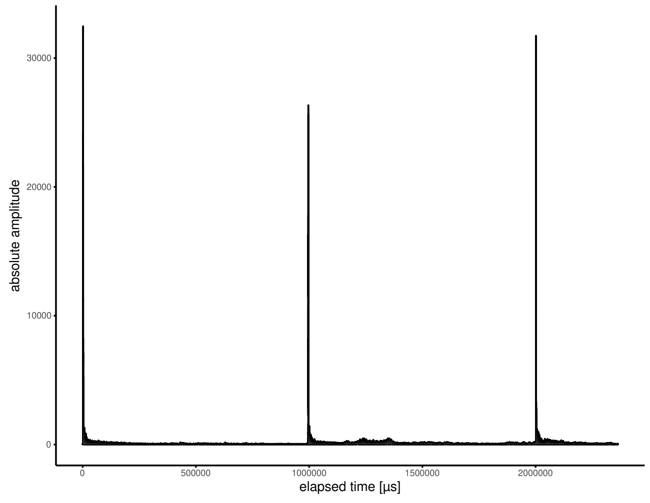
1) Interpolating the time for each sample: As mentioned above, the data is only provided in chunks. This means that for each sample in this chunk, the times must be interpolated. One step for interpolation is calculated by the following equation:

$$I_s \quad = \quad \frac{(t_n - t_{n-1})}{S_r} \tag{1}$$

where I_s is the stepwidth for the interpolation, t_n is the timestamp the current chunk is received, $t_n - 1$ the timestamp the last chunk was received and S_r is the selected sample rate in our example $44,100$ Hz.



(a) Single peak



(b) Multiple peaks

Fig. 7. App recorded (a) single peak or (b) multiple peaks.

2) Preprocessing: In the given signals of each device missing values can occur, i.e., the peak of the audio signal cannot be detected. In these cases the signal is replenished, i.e., if the difference between two records is greater than $periodicity \times 10\ \%$ the missing value is added in between. Thereby, the distance is fixed by $periodicity \times sample\_rate$.

E. Filtering and storing the values

Afterwards, the values are filtered by amplitude. For this purpose, a threshold value can be configured in the App. The default value is 80 % of the maximum amplitude. All filtered values are stored in a database and can be used to align the devices.

IV. Results and Experiments

During our experiments, we used different model devices from different manufacturers and will briefly denote this with Sp1, Sp2 and so on. These devices are equipped with up to three MEMS-Microphones (Micro-Electro-Mechanical-Systems) and are all operating with the Android operating system in versions from 4.2 up to 8.1.

A. Single Error Analysis for One Device

Fig. 8 shows, as an example for one of the devices, the simple error of the measured values during a half-hour synchronization process. The difference between ground truth (audio file) and the measured time of the received signal is colored gray and called single error. This deviation can be explained by the jittering of the operating system. The red line represents the mean value of the deviation (0.0105 ms ($\pm$4.600)). Since the mean derivation is slightly positive, the deviations are accumulated, and thus, the total error increases over time.

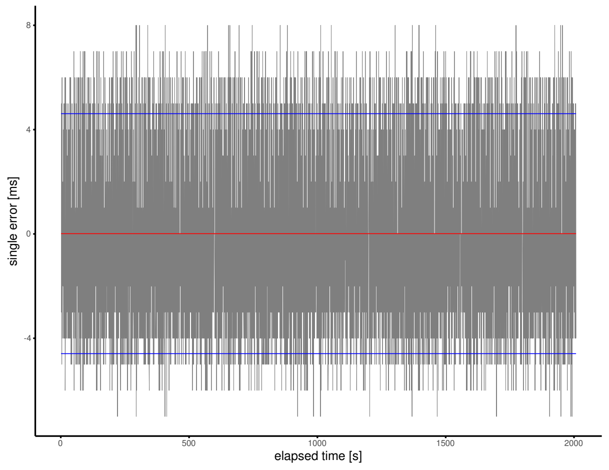


Fig. 8. Single derivation of the received signals to the ground truth of 1000 ms with the mean single error (red) and standard deviation(blue)

B. Single Error Comparison

All devices have similar behavior in the total error development (see Fig. 9). The deviation to the ground truth of Sp1 is less compared to especially Sp3. This can also be seen in the statistical values listed in Table I. Moreover, the single mean errors of each signal are small (about 0.01 ms). However, the standard deviations are high for all three devices (greater than 4 ms). This reflects the maximum of the single error, which is around 10 ms. These values correspond to those of [16].

C. Synchronization of the Time Signals

Now, the acoustic signal can be used to synchronize the devices. After a fixed time interval the acoustic signal is recorded, the devices are synchronized and thus the total error of time deviation can be reset to zero. We used smartphone SP1 and select different synchronization intervals.

TABLE I. STATISTIC VALUES ABOUT THE ERRORS OF THE SINGLE SMARTPHONES (ELAPSED TIME WINDOW 15 MIN)

device	Sp1	Sp2	Sp3
mean total error (mte) [ms]	2.0111	7.1256	9.2700
standard deviation (std) [ms]	(±4.1750)	(±4.705)	(±4.3541)
maximum total error [ms]	13	18	20
mean single error [ms]	0.0100	0.0111	0.0189
standard deviation [ms]	(±4.6101)	(±4.5342)	(±4.7198)
maximum single error (mse) [ms]	8	7	12

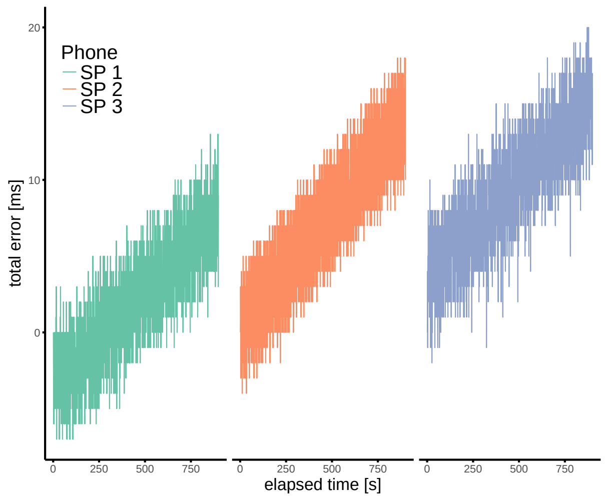


Fig. 9. Total error development of the three different smartphones

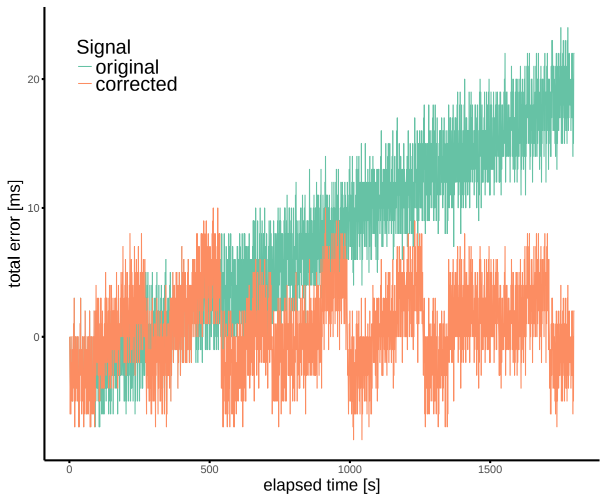


Fig. 10. Total error of the original signal and the corrected signal (adjustment interval: 90 ms)

In Table II the statistical values for the different adjustment intervals are given. It can be seen that the maximum total error, as well as the standard deviation, is nearly the same for all intervals. The mean total error increases with the longer interval lengths. Fig. 10 shows the two total errors for two different correction intervals, one from the original signal and one from the corrected signal. It can be seen that we can reduce the accumulation of the total error. To get a better understanding of the error distribution, in Fig. 11 the histograms of the total errors are visualized. For the original signal, the total error is widespread and has an overhang to larger positive values compared to the corrected signal. The corrected signal has smaller total errors and a smaller distribution around the zero point. The differences in the error distribution according to the size of the intervals are not decisive, as it can be seen in Table II.

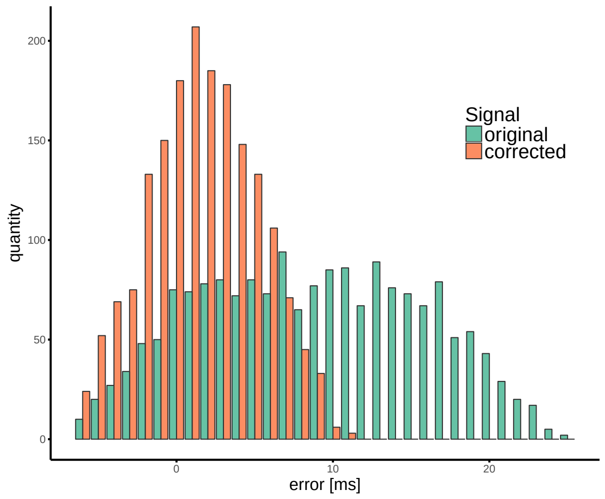


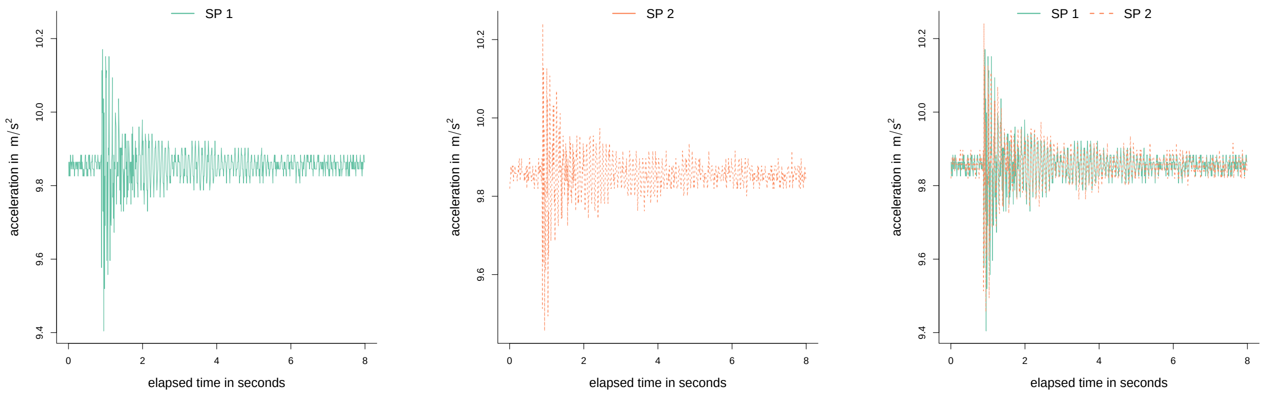
Fig. 11. Histogram of the total error distribution of the original signal and corrected signal (adjustment interval: 90 ms)

D. Aligning Sensor Data based on a Time Synchronization

Assuming a situation in which a bridge has to be assessed. The assessment is based on the measured values of acceleration sensors built-in the smartphone. We use several smartphones in this situation since we can get a more accurate result. In order to align the measured values, these devices must be synchronized - the use case for our proposed approach of acoustic sound synchronization. The object of this experiment

TABLE II. STATISTICAL VALUES ABOUT THE TOTAL ERROR FOR DIFFERENT ADJUSTMENT INTERVALS (MEAN TOTAL ERROR (MTE), STANDARD DEVIATION (STD), MAXIMUM OF ABSOLUTE ERROR (ME)

correction interval [s]	15	30	45	60	95	120	180	360
mte [ms]	0.095	−.0133	−0.3200	−0.113	0.620	−0.080	1.120	1.720
std [ms]	(±3.60)	(±3.60)	(±3.67)	(±3.37)	(±3.50)	(±3.18)	(±3.65)	(±3.52)
me [ms]	10	10	10	9	9	10	11	11



(a) vibrations of Sp1 (b) vibrations of Sp2 (c) vibrations of both devices combined

Fig. 12. Vibrations of a bridge triggered by a walking pedestrian and recorded by the smartphones. Fig. (a) shows the vibrations recorded by Sp1, (b) recorded by Sp2 and (c) the data of both devices aligned after time synchronization.

is a single span girder bridge (T-beam) having a structure of concrete-steel-composite (see Fig. 13). This bridge has a length of 30 meters, a width of 4 meters and weighs about 70 tons. During the experiment, the bridge was stimulated by a walking pedestrian. We placed the smartphones at about 1/6 of the bridge's length. Fig. 12a and Fig. 12b show the vibrations caused by the pedestrian and recorded by smartphone 1 and smartphone 2. As it can be seen, there are differences between the measured values of the two devices. Since the two devices were synchronized by our method, these measured values can be aligned. This leads to a more precise measurement result, as it can be seen in Fig. 12c and thus to a more reliable assessment of the accessibility of this bridge.

V. Conclusion

In this article, we introduced MobileTimeSync - an Android App for time synchronization on smartphones. This app generates an audio signal that contains several peaks at the same distance from each other. Afterwards, this signal is sent out by the built-in loudspeaker of the device. This signal is received by all nearby devices and can be recognized by observing the amplitude of the built-in microphones and the corresponding timestamp. Finally, we achieved an accuracy of 10 ms. Thus, we confirm the results of Lazik et al. [16]. One reason for this value is the strong jittering on these devices. In the next steps, we have to investigate whether this accuracy is sufficient for the above-mentioned issue of damage localization. Furthermore, the jittering must be investigated. One option to avoid jittering could be the Native Development Kit (NDK), which allows native system calls.

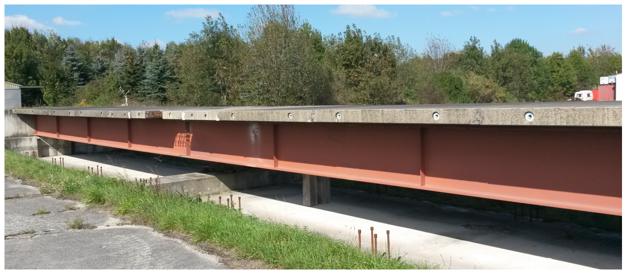


Fig. 13. The bridge for our experiment.

Nevertheless, one advantage of this approach is that network, data or GPS connections are not necessary. The only limitation is the distance between the sound source and the devices. This means that this approach is highly suitable for areas where the above-mentioned synchronization services are not available. Rolling ahead, we want to bring this App to the Android Play store.

Acknowledgement

This research has received funding from the German Federal Ministry of Education and Research within the project "Vulnerability of Transportation Structures – Warning and Evacuation in Case of Major Inland Flooding" "FloodEvac" (No. 13N13194/13N13195). Special thanks to Bharat Yadav for proofreading.

REFERENCES

[1] M. Benndorf and T. Haenselman, "Time synchronization on android devices for mobile construction assessment," in *Tenth International Conference on Sensor Technologies and Applications SENSORCOMM 2016*. IARIA, 2016, pp. 83–87.

[2] M. Benndorf, M. Garsch, T. Haenselmann, N. Gebekken, and I. Videkhina, "Mobile bridge integrity assessment," in *Proceedings: IEEE Sensors 2016*, 2016.

[3] M. Benndorf *et al.*, "Robotic bridge statics assessment within strategic flood evacuation planning using low-cost sensors," in *Safety, Security and Rescue Robotics (SSRR), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 13–18.

[4] K. Sun, P. Ning, and C. Wang, "Tinysersync: secure and resilient time synchronization in wireless sensor networks," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 264–277.

[5] B. Kusy *et al.*, "Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 1, no. 4, pp. 239–251, 2006.

[6] J. Postel, "Rfc 768 - user datagram protocol," *Isi*, 1980.

[7] D. L. Mills, "Network time protocol (ntp)," *Network*, 1985.

[8] ——, "Internet time synchronization: the network time protocol," *Communications, IEEE Transactions on*, vol. 39, no. 10, pp. 1482–1493, 1991.

[9] J. Levine, "Timing in telecommunications networks," *Metrologia*, vol. 48, no. 4, p. S203, 2011.

[10] A. N. Novick and M. A. Lombardi, "Practical limitations of ntp time transfer," in *Frequency Control Symposium & the European Frequency and Time Forum (FCS), 2015 Joint Conference of the IEEE International*. IEEE, 2015, pp. 570–574.

[11] K. Zhao, A. Zhang, and K. Liang, "Research on the uncertainty evaluation of network time service system," in *European Frequency and Time Forum (EFTF), 2012*. IEEE, 2012, pp. 122–125.

[12] S. Ping, "Delay measurement time synchronization for wireless sensor networks," *Intel Research Berkeley Lab*, 2003.

[13] W. Commons, "Ntp— file: -.jpg," 2016, [Online; accessed 01-March-2018]. [Online]. Available: http://bit.ly/2F04ILs

[14] D. C. Mazur, R. A. Entzminger, J. A. Kay, and P. A. Morell, "Time synchronization mechanisms for the industrial marketplace," in *Industrial & Commercial Power Systems Technical Conference (I&CPS), 2015 IEEE/IAS 51st*. IEEE, 2015, pp. 1–7.

[15] S. Sridhar, P. Misra, G. S. Gill, and J. Warrior, "Cheepsync: a time synchronization service for resource constrained bluetooth le advertisers," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 136–143, 2016.

[16] P. Lazik, N. Rajagopal, B. Sinopoli, and A. Rowe, "Ultrasonic time synchronization and ranging on smartphones," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*. IEEE, 2015, pp. 108–118.

[17] S. Giucastro, "Getting high precision timing on android," 2012, (retrieved: 13-Mar-2017). [Online]. Available: http://ubm.io/2HWdiZz

[18] Google, "Android developers systemclock," (retrieved: 15-Mar-2017). [Online]. Available: https://developer.android.com/reference/android/os/SystemClock.html

[19] K. Gopal, "Truetime for android," 2016, (retrieved: 29-Mar-2017). [Online]. Available: https://github.com/instacart/truetime-android

[20] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *Network, IEEE*, vol. 18, no. 4, pp. 45–50, 2004.

[21] T. Haenselmann, *Wireless Sensor Networks: Design Principles for Scattered Systems*. Oldenbourg Verlag, 2011.

[22] J. Aron, "Flickering lights could help smartphones keep time," *New Scientist*, vol. 215, no. 2874, p. 22, 2012.

[23] Z. Li *et al.*, "Flight: Clock calibration using fluorescent lighting," in *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 2012, pp. 329–340.

[24] E. Peguero, M. Labrador, and B. Cook, "Assessing jitter in sensor time series from android mobile devices," in *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–8.