# Autodirective Audio Capturing Through a Synchronized Smartphone Array

Sanjib Sur†, Teng Wei† and Xinyu Zhang
University of Wisconsin-Madison
{sur2, twei7}@wisc.edu and xyzhang@ece.wisc.edu
†Co-primary authors

## ABSTRACT

High-quality, speaker-location-aware audio capturing has traditionally been realized using dedicated microphone arrays. But high cost and lack of portability prevents such systems from being widely adopted. Today's smartphones are relatively more convenient for audio recording, but the audio quality is much lower in noisy environment and speaker location cannot be readily obtained. In this paper, we design and implement Dia, which leverages smartphone cooperation to overcome the above limitations. Dia supports spontaneous setup, by allowing a group of users to rapidly assemble an array of smartphones to emulate a dedicated microphone array. It employs a novel framework to accurately synchronize the audio I/O clocks of the smartphones. The synchronized smartphone array further enables autodirective audio capturing, *i.e.*, tracking the speaker's location, and beamforming the audio capturing towards the speaker to improve audio quality. We implement Dia on a testbed consisting of 8 Android phones. Our experiments demonstrate that Dia can synchronize the microphones of different smartphones with sample-level accuracy. It achieves high localization accuracy, and similar beamforming performance compared with a microphone array with perfect synchronization.

## Categories and Subject Descriptors

C.3.3 [**Special-Purpose and Application-based Systems**]: Signal processing systems

## Keywords

Ad-hoc microphone array; smartphone synchronization; acoustic localization; acoustic beamforming

## 1. INTRODUCTION

The pervasiveness of smartphones is driving a continuous growth of mobile multimedia applications, which take advantage of the microphone and camera modules for spontaneous audio-visual capturing. Yet such applications are mostly constrained to short-range individual use, undoubtedly because of the private nature of standalone smartphone devices. Common smartphones are not well suited for demanding multimedia applications. For example, *(i)* A smartphone's audio-capturing quality can be significantly degraded in noisy environment, with noises coming from ambient electronic devices and interfering voices. *(ii)* A smartphone does not work well in lecture recording applications, where the lecturer may walk far away from the microphone and her speech may be immersed in noise. *(iii)* A similar problem occurs in meeting recording or teleconferencing scenarios, where certain speakers can be far from the microphone.

Currently, such challenging audio capturing tasks require dedicated commercial platforms. For example, Polycom CX-5000 [1] and Microsoft RingCam [2] incorporate a microphone array to enhance audio quality and locate speakers. The location information is used to guide a steerable camera to focus its view angle on a main speaker. Ricoh [3] developed an audio-visual recording system with a similar objective, but using video image processing to track conference participants. These high-end systems overcome the narrow view-angle and low audio quality of smartphones, but tend to be expensive and lack portability. Simple solutions using close-talking microphones may be used in lecture-recording cases, but they are intrusive and distracting [4].

In this paper, we propose an alternative system that leverages an ad-hoc group of smartphones (referred to as a *smartphone array*) to enhance audio capturing in the above scenarios. These smartphones may be temporarily collected from participants in a meeting or lecture. Through tight cooperation, their individual microphones can form a virtual microphone array that beamforms to a main speaker, and localizes the speaker in real-time to facilitate video recorders, *e.g.*, a steerable camera [5]. We refer to such a system as autoDirective audio (Dia) capturing.

The advantages of Dia are multifold. It inherits all the benefits of a dedicated microphone array. Yet it allows spontaneous setup at almost no extra cost. In addition, Dia has the potential to enable directive, portable audio-visual recording, by aligning multiple smartphones' cameras and triggering each opportunistically according to speaker's location.

Realizing such potential entails several grand challenges, the major one lying in synchronization of the smartphones. Speaker tracking heavily relies on Time-Difference-Of-Arrival (TDOA) statistics among distributed microphones. Therefore, the smartphones must synchronize their microphones with sufficient precision, *i.e.*, the audio sampling clocks must *be triggered simultaneously* and *tick at the same rate*. Sim-


*MobiSys'14,* June 16–19, 2014, Bretton Woods, NH, USA.

http://dx.doi.org/10.1145/2594368.2594380.

ilarly, beamforming relies on synchronization to align the audio samples. Sampling offset may cause different microphones to cancel instead of enhancing each other's signals.

Dia meets the above challenges using a distributed two-level synchronization framework. It first synchronizes the CPU clocks of different smartphones, by allowing them to timestamp the WiFi beacons overheard from a nearby access point, and calibrate the CPU time to align with each other. Then, with the help of a PC server, each smartphone's audio I/O clock gets synchronized with its own CPU clock. Consequently, the sampling clocks of distributed microphones are synchronized to a global clock, thus enabling the same functionalities as a dedicated microphone array.

Given this synchronized array of smartphones, Dia incorporates two cooperative audio signal processing modules to achieve autodirective audio capturing. First, it runs a practical beamforming algorithm that assigns weights to the synchronized audio signals captured by different smartphones, the optimal weight vector corresponding to the beam direction that maximizes the speaker's voice quality. Second, Dia estimates the TDOA of audio signals from multiple microphones to track the speaker's angle relative to the smartphone array. It employs a novel *binary mapping* approach to scale the localization scheme to more than two smartphones with unknown separation.

We have implemented Dia based on the Android platform. Our experiments demonstrate that Dia can synchronize distributed smartphones with an accuracy of around 2 samples at 16 kHz sampling rate. With an array of 8 smartphones, Dia is able to boost the voice quality by 11 dB, which is comparable to an oracle scheme with perfect synchronization. More importantly, Dia's performance scales as more smartphones are put together. In addition, our field test shows that Dia is able to track the speaker's direction with an error of around 10 degrees when the smartphones' separation is known, and track the speaker's region with more than 90% of accuracy under unknown smartphone separation.

We make the following contributions in Dia.

*(i)* We design a novel synchronization mechanism that can synchronize the audio I/O clocks of distributed smartphones at sample-level, which achieves comparable performance as a microphone array. The synchronization algorithm is not only a key component of Dia, but also a standalone contribution. It opens up a wide range of spontaneous audio sensing applications that used to be available only on expensive dedicated hardware platforms.

*(ii)* We propose a practical audio beamforming system that leverages cooperative signal processing on the synchronized smartphones to boost audio quality. Further, we develop a speaker tracking algorithm on top of Dia that provides location guides to external video capturing devices.

*(iii)* We implement Dia on a mobile platform and conduct comprehensive experiments to validate its performance, against an oracle scheme and in actual application scenarios.

The remainder of the paper is structured as follows. Sec. 2 presents an overview of Dia and its usage cases. Sec. 3 introduces the design and implementation of the two-level synchronization algorithm, followed by the audio beamforming and speaker localization/tracking algorithms in Sec. 4 and Sec. 5. Sec. 6 elaborates on the experimental evaluation of Dia on top of our Android implementation. Sec. 7 discusses limitations and open problems in Dia. Sec. 8 surveys related work and finally, Sec. 9 concludes the paper.

# 2. Dia: AN OVERVIEW

## 2.1 Application Scenarios

Dia enables spontaneous setup of a distributed microphone array using an ad-hoc group of smartphones. It precisely synchronizes the distributed microphones to (i) enhance audio coverage and quality by beamforming speaker's voice to the embedded array of microphones, and (ii) localize and track the speaker's direction relative to the smartphone array. Such functionalities are useful in a wide range of application scenarios. We provide a few examples below.

*(i) Smart conferencing.* Imagine a conferencing session, where participants intend to start an ad-hoc discussion and spontaneously capture their speech as audio minutes. The conference can occur anywhere and anytime, *e.g.*, in a meeting room, a company cafeteria, a picnic area in a park or a dinning table in a restaurant. Due to lack of portability, dedicated audio recording devices are not readily applicable. Requesting each individual to use their smartphones as close-talking microphones may ensure voice quality, but this is obtrusive and can be distracting [4]. It is also non-trivial to stitch the recordings of individual smartphones to form a coherent piece of meeting record. With Dia, the participants can simply place their smartphones together and record high-quality audio in a similar way to a commercial microphone array [6]. Dia automatically focuses on a main speaker and tracks the speaker in case of movement or role switching. It compiles the audio signals of all smartphones into a single audio track that can also be streamed to a remote site.

*(ii) Autonomous lecture recording.* In a classroom or lecture hall, Dia can act as a low-cost automatic recording system. It can run on an array of smartphones belonging to the audience. Through synchronous, cooperative audio signal processing, Dia will be able to maintain recording quality even in low signal-to-noise-ratio (SNR) conditions due to, *e.g.*, lecturer walking away from the microphones, and noise from electrical appliances and outside vehicles/machinery.

Dia works under several premises in the above setup. It needs at least two smartphones each equipped with a microphone and WiFi interface. It presumes the smartphone owners trust each other and are willing to allow tight cooperation (exchanging messages) between their phones. In addition, Dia relies on a back-end server to process the audio signals. Thus, the smartphones need to have network connection to the server (*e.g.*, through WiFi or 3G).

The speaker's location information provided by Dia can facilitate a variety of external applications. An immediate one might be a cooperative video capturing system that uses the smartphone array's cameras to capture a meeting, lecture, *etc.* The smartphones may be placed along a line or circle, such that their view angles cover an entire area of interest. At any time, only one camera is triggered that covers the speaker according to Dia's location information. The resulting video frames can be stitched according to their timestamps. Alternatively, Dia can enable a commercial steerable camera system [5] to follow the speaker in real time.

## 2.2 System Architecture and Basic Operations

Figure 1 illustrates Dia's architecture and basic flow of operations. Dia acts as a software module running on the smartphones and a back-end server. It consists of three ma-

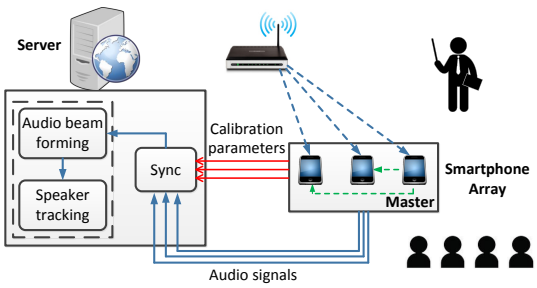


**Figure 1: Architecture of Dia.**

jor modules: two-level synchronization, audio beamforming, and speaker localization/tracking.

Dia works as follows in a typical use case. First of all, users collect multiple smartphones and place them along a regular geometry shape (*e.g.*, line or circle, to be discussed in Section 5). After such initial setup, *two-level synchronization*, a distributed I/O synchronization protocol, is triggered and operated by both the server and smartphones. Specifically, one smartphone is arbitrarily chosen as the master and it coordinates the other phones to collect relevant timestamp information from ambient WiFi beacons, and from their own audio interface, in order to estimate a set of parameters for timing calibration. The timing calibration is conducted jointly by the server and smartphones, which enables each smartphone to calibrate its CPU clock against a global clock, and its local audio clock against the CPU clock. Afterwards, the smartphones will be able to start audio capturing at the same time, and precisely align their audio samples, even though they are driven by independent CPU clocks and audio I/O clocks rolling at different rates.

The audio streams of all smartphones are delivered to the server through either WiFi or cellular network. The server then runs an *audio beamforming* module that processes the synchronized audio samples to enhance audio quality and suppress noise. The underlying rationale lies in a high level of diversity provided by multiple microphones, similar to multi-antenna spatial diversity in MIMO wireless communications [7]. In addition, the server runs a *speaker localization/tracking* framework that leverages relative timing between the microphones' audio samples to track the speaker's angle relative to the smartphones.

Both of the beamforming and speaker tracking modules benefit from higher performance (in terms of audio SNR and localization granularity) as the number of smartphones increases. Such extensibility is a unique advantage compared with dedicated microphone array hardware.

Note that audio data processed by the server can be either recorded to a file or streamed in real-time to a remote site (we focus on the former case in our implementation). Dia mainly targets the case with one dominating speaker at any time. We will discuss potential approaches to addressing competing speeches in Section 7.

# 3. AUDIO I/O CLOCK SYNCHRONIZATION AMONG SMARTPHONES

In this section, we first motivate the problem of tight audio clock synchronization by analyzing the application requirement, then we introduce the design and implementation of Dia's two-level synchronization framework in detail.

## 3.1 Why Synchronization?

Microphone-array based autodirective audio capturing requires tight synchronization between the microphone elements [8]. For audio beamforming, received signals of different microphones need to be coherently combined. Severe timing offset will cause significant phase misalignment between the signals and hence preventing beamforming. Similarly, misalignment induces ambiguities when estimating the signal's TDOA to different microphones, thus causing localization error.

Audio devices' synchronization errors come from two sources: *initial timing offset and sampling-rate difference (clock drift)*. The former is due to the devices' CPU clock difference and timing uncertainties between CPU and audio I/O — even if an application instructs the devices to start simultaneously, the actual time when they are triggered may differ significantly. Human voices commonly fall below 2 kHz, with up to 4 kHz in rare cases [9]. Thus, a constant initial offset of below 1/4000=0.00025 $s$, or 250 $\mu s$, between the different microphones does not noticeably affect the microphone-array's performance. In effect, signals with such offset can be considered as interpolations of the original signal, which can still be coherently combined. For speaker tracking applications, an initial timing offset of 250 $\mu s$ translates into a distance estimation error of 343 m/s $\times$ 0.00025 s=0.086 m, which is easily acceptable in practice.

However, achieving this 250 $\mu s$ timing alignment is a non-trivial task for COTS smartphones. Due to unpredictable jitter caused by application execution and OS scheduling, existing signaling-based network synchronization protocols can only achieve [10, 11] millisecond-level granularity. GPS can provide $\mu s$ level synchronization, but it does not work well in indoor environment.

Even after perfect initial synchronization, the second source of errors, *i.e.*, sampling clock drift, remains a serious challenge. Driven by independent oscillators, the smartphones' audio sampling clock rates are bound to differ and, even minor clock skew will accumulate the sampling time offset to quickly diverge beyond the synchronization requirement.

To make the problem more concrete, we measured the audio sampling offset of 5 Galaxy Nexus smartphones (3 I9250 models and 2 I515), by playing a very short 'linear chirp' tone from a sound source at an interval of 15 seconds. We used chirp tones in both human audible frequency range of 10 kHz – 10.5 kHz and inaudible range of 22.05 kHz – 22.3 kHz. To isolate the impact of propagation delay, all the smartphones and the sound source are placed within 6 cm distance. The sampling offsets are obtained by visually checking the onset peak position offset between 'chirp' signals received by the smartphones. Figure 2 plots the relative sampling offset (number of samples drifted) of 4 other phones taking one I9250 as a reference point. Clearly, the audio sample drift between multiple smartphones increases over time and escalates to an intolerable value of up to 22 samples (499 $\mu s$) within merely 60 seconds of recording.

**Does audio-beacon based synchronization work?** At first blush, it might seem sufficient to designate one smartphone as a master node, which periodically broadcasts audio beacon signals as synchronization pilots. All other nodes record the beacons along with desired voice signals. Afterwards, a server can process the signals offline, and synchronize the signals by searching for the optimal beacon-time alignment through cross-correlation. Our earliest attempt to

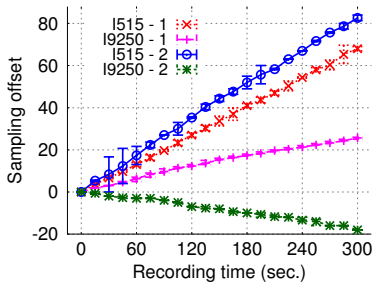


**Figure 2: Audio sampling offset between multiple distributed smart phones. Error bars represent maximum and minimum. Sampling rate equals 44.1 kHz.**

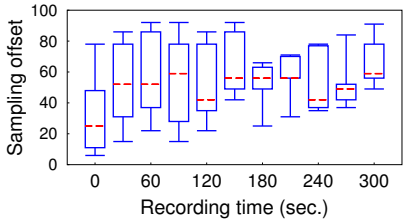


**Figure 3: Box plot showing the minimum, first quartile, median, third quartile and maximum sampling offset calculated using audio beacon based synchronization.**

build Dia relied on this audio-beacon based synchronization principle. However, two major bottlenecks corrupted the effort.

First of all, we find that audio reverberation and fading effects distort the beacon signals significantly, causing erratic uncertainties in the offline synchronization. We conducted an experiment using a very short 'chirp' tone as beacon and broadcast signal at an interval of 30 seconds. In ideal scenario, the cross-correlation process would have shown the same trend of linear offset increment as in Figure 2. However, we find that cross-correlation process is quite random in nature, as plotted in Figure 3. This is also the reason why, instead of using cross-correlation, we have to visually check the onset peak position offset between the 'chirp' signals as in Figure 2, in order to gauge the sampling offset.

The second bottleneck lies in the conflict between beacons and desired voices. Sampling rate estimation inevitably has some residual errors that accumulate over time. Thus, synchronization beacons need to be broadcast periodically for recalibration purpose. This will interfere the actual recording process, which simply defies the purpose of performing synchronization for high audio quality in first place. Although it is possible to send the audio beacons through inaudible frequency band, the first bottleneck still persists.

## 3.2 Sample-level Audio I/O Synchronization in Dia

### 3.2.1 System model and problem formulation

Any hardware clock module $HC(\cdot)$ in the smartphone can be modelled as:

$$HC(t) = \int_{t_0}^{t} r(\tau)d\tau + \theta(t_0) \quad (1)$$

where $r(\tau)$ is the clock rate at time $\tau$ and $\theta(t_0)$ the initial clock offset at time $t_0$ with respect to an oracle clock.

Ideally, $r(\tau)$ maintains a constant value of 1. In practice, the clock drifts over time. But for bounded time interval, the clock drift is reasonably bounded with a small drift $\rho$, *i.e.*, $r(\tau) = 1 + \rho$.

In a typical embedded platform, there exists hardware registers that can be probed by software to obtain *logical clock* values $LC(\cdot)$:

$$LC(t) = \int_{t_0}^{t} r(\tau).p(\tau)d\tau + \theta(t_0) \quad (2)$$

where $p(\tau)$ denotes the software probing interval. For the CPU, $LC(\cdot)$ is simply the system timestamp, whereas for audio I/O, it is the sample index counted from the microphone's triggering time. Henceforth, the clocks we mention are logical clocks by default.

In COTS smartphones, it is infeasible to directly program the audio I/O clock to achieve distributed synchronization. Dia circumvents this barrier via a novel two-level principle: *(i) Inter-device synchronization:* synchronizing the smartphones' CPU clock against a global clock. *(ii) Intra-device synchronization:* synchronizing each smartphone's audio I/O clock against its own CPU clock. We formulate this principle in more detail below.

Suppose the logical CPU clock of smartphone $i$ at time $t$ is $LC_i^{cpu}(t)$. For a bounded time interval, it can be assumed that there exists a linear relationship between $LC_i^{cpu}(t)$ and a global clock $GC(t)$:

$$LC_i^{cpu}(t) = a_i(\delta).GC(t) + b_i(\delta) \quad (3)$$

where $\delta = (t - t_0)$ is bounded. The timing model parameters $a_i(\delta)$ and $b_i(\delta)$ are constant within $\delta$.

Since both the audio I/O clock and CPU clock advance linearly over time, there is an implicit linear relation between them:

$$LC_i^{audio}(LC_i^{cpu}(t)) = \alpha_i(\delta).LC_i^{cpu}(t) + \beta_i(\delta) \quad (4)$$

with timing model parameters $\alpha_i(\delta)$ and $\beta_i(\delta)$ being constant within $\delta$. Here, $LC_i^{audio}(LC_i^{cpu}(t))$ is the audio sample index at CPU clock $LC_i^{cpu}(t)$.

Given the above two timing models, the problem of synchronizing the audio samples between multiple smartphones can be formulated as calculating $GC(LC_i^{audio})$, *i.e.*, the global timestamp of audio samples $LC_i^{audio}$, and compensate for the differences across multiple devices. Said differently, if we know the global clock value corresponding to an audio sample of one smartphone, we can use Eq. (3) and (4) to calculate the audio sample number of a different smartphone at that global clock value, and compensate for the differences. Dia's two-level synchronization algorithm is built atop this principle.

### 3.2.2 Two-level synchronization algorithm

**Illustrative example.** Figure 4 showcases an example of how Dia synchronizes two smartphones with 1 sample offset. For simplicity, let us assume the timing model parameters in Eq. (3) and (4), *i.e.*, $a_i(\delta)$, $b_i(\delta)$, $\alpha_i(\delta)$ and $\beta_i(\delta)$, are known for both smartphones. For a given audio sample $s_1$ in smartphone 1, it knows its local timestamp $lt_1$ following Eq. (4). It can further infer the corresponding global timestamp $gt$ following Eq. (3). This global time stamp can be passed to smartphone 2, who infers its own local time stamp $lt_2$ following Eq. (3) (with different timing model parameters). Then, it uses Eq. (4) to calculate the corresponding audio sample number $s_2$. In this way, both the smartphones know the exact audio sample index corresponding to a particular

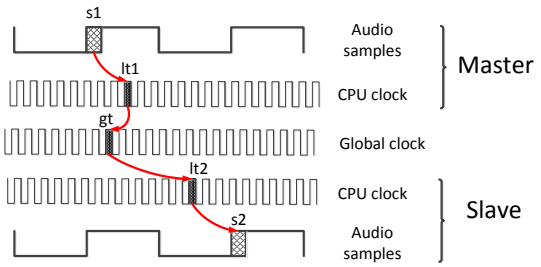


**Figure 4: An example timing diagram showing, the audio sample synchronization between two smartphones, having 1 audio sample offset.**

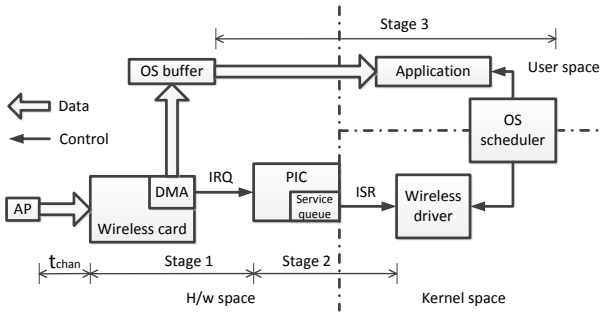


**Figure 5: Network I/O control and data flows in smartphone.**

global timestamp $gt$, which in turn synchronizes the audio samples.

The key problem in realizing Dia's principle is to estimate the CPU and audio timing models. In Dia, the smartphones leverage a WiFi AP's beacon messages to estimate parameters for the CPU timing model following Eq. (3) , and then each smartphone locally infers the audio timing model based on CPU/audio timestamps following Eq. (4). This requires each smartphone to obtain highly reliable logical timestamps for the WiFi beacons and audio samples.

**Probing logical clock timestamps.** Unavailability of hardware timestamps forces us to resort to software probing of timestamps. The software probing interval must be constant across multiple devices (Eq. (2)). This is hard to achieve if we probe the timestamps directly from application-level. We analyze this problem in Figure 5, which shows the processing diagram of the network I/O path in a typical smartphone[1], comprised of 3 stages:

*Stage 1.* The input data are digitized and partially processed in hardware. The Direct Memory Access (DMA) engine transfers the data to an OS buffer and generates an Interrupt ReQuest (IRQ). In devices with similar hardware, this takes an approximately equal amount of time.

*Stage 2.* The Programmable Interrupt Controller (PIC) registers the IRQ in an interrupt service queue, and the OS kernel schedules itself to call the Interrupt Service Routine (ISR) of the particular device driver, and then serve the interrupt. Since PIC handles IRQ from multiple hardware modules, a variable amount of delay is introduced here.

*Stage 3.* Once the interrupt is serviced, the OS delivers the data to the user space. The variable delay in this step is significantly higher than that in *Stage 2*, as it depends on many factors including but not limited to, priority of the application in a multi-processing environment, CPU utilization, *etc.*

Clearly, to fix the software probing interval to a constant value, we should probe in *Stage 1*. Unfortunately, there is no scope of CPU to register the timestamp before the IRQ is serviced by the OS. Thus, in Dia, we aim to achieve a reasonable determinism by probing the hardware clocks only at the ISR (*Stage 2*). However, there could still be jitters, the root cause emerging from the fact that the core OS in today's smartphones is not a real-time. In what follows, we will show how we overcome this jitter in Dia.

**Estimating timing parameters under timestamp jitters.** Modern 802.11 APs embed a 64-bit timestamp in each broadcast beacon (so called TSF timer), which can be captured by smartphones as global timestamp. For each Wi-Fi beacon packet $j$ containing timestamp $GC(t^j)$, smartphone $i$ keeps a tuple $\langle GC(t^j), LC_i^{cpu}(t^j)\rangle$ where, $LC_i^{cpu}(t^j)$ denotes the arrival CPU timestamp of the beacon packet $j$.

Given $n$ sets of observation tuples $\langle GC(t^j),\ LC_i^{cpu}(t^j)\rangle$ $for\ j = 1, 2, \ldots n$, we want to estimate the parameters $\widehat{a_i}(\delta)$ and $\widehat{b_i}(\delta)$ for bounded $\delta$, following Eq. (3). This is a classical problem of linear curve fitting which can be solved using a linear regression method [12]. However, the jitters in timestamps statistics, obtained by software probing, might reduce the goodness of fitting and hence the reliability of estimation.

We solve this problem by using a least trimmed square (LTS) regression model [13], which removes outlier observations with large residual errors and then computes a least mean square regression model for rest of the observations. This is particularly suitable in reducing the dependency on the OS jitter in serving the ISR.

We apply a similar method to estimating the audio timing model parameters, $\widehat{\alpha_i}(\delta)$ and $\widehat{\beta_i}(\delta)$ following Eq. (4). Corresponding to the audio sample index $k$, we have the CPU timestamps $LC_i^{cpu}(t^k)$. This gives a set of observation tuple $\langle LC_i^{cpu}(t^k), k\rangle$. Now, given $m$ sets of observation tuple $\langle LC_i^{cpu}(t^k), k\rangle$ $for\ k = 1, 2, \ldots m$, the problem of estimating timing models $\widehat{\alpha_i}(\delta)$ and $\widehat{\beta_i}(\delta)$ is solved again by using LTS regression.

**Reliable relationship between logical clocks.** A linear relationship between global $\leftrightarrow$ local logical clock and local $\leftrightarrow$ audio I/O logical clock is assumed across the above two-level synchronization algorithm design. We now present an empirical evidence to show this relationship holds in practice for bounded time interval $\delta$.

To this end, we used the same 8 Galaxy Nexus smartphones mentioned in Section 3.1 and register beacon packet arrival times from a single WiFi access point. The timestamps are registered inside the ISR of the Broadcom WiFi driver [14]. We also register the original timestamps that were embedded into the beacon packets which act as the global logical clock of that particular access point.

Table 1 shows the regression results of inter-device synchronization for different smartphones. $R^2$ is the *Coefficient of Determination*, a widely used measure of how well the result of linear regression method is [12]. We can see that the $R^2$ values are well above 0.999 for all the smartphones, which indicates very good fitting results. We also performed similar experiment to find the regression results on the relationship between audio sample number to the local CPU

[1]The processing diagram of the audio I/O path is similar, except that the data comes from the microphones.

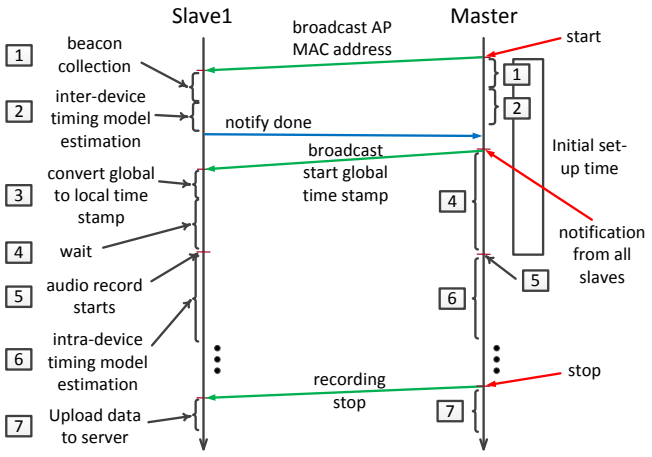


**Figure 6: Space-time diagram of the signaling protocol between smartphones.**

clock and achieve similar fitting result. We omit this in the interest of space.

| Smart Phone | Slope $a$ | Intercept $b$ | $R^2$ |
|---|---|---|---|
| 1 | 0.99998 | -34111970 | 0.99968 |
| 2 | 1.00020 | -12020470 | 0.99962 |
| 3 | 0.99967 | 18574170 | 0.99982 |
| 4 | 1.00005 | -23536980 | 0.99977 |
| 5 | 1.00006 | -33825370 | 0.99977 |
| 6 | 1.00008 | -46648270 | 0.99979 |
| 7 | 1.00010 | 56972960 | 0.99970 |
| 8 | 1.00010 | -55440930 | 0.99976 |

**Table 1: LTS regression results between the global logical clock and CPU logical clock of 8 smartphones, in the form of $LC_i^{cpu}(t) = a_i(\delta).GC(t) + b_i(\delta)$, where $i$ is the smartphone index. $R^2$ is the metric indicating the goodness of fitting.**

## 3.3 Implementing Two-Level Synchronization

Dia's implementation of two-level synchronization involves a PC server and multiple smartphones. It extends the prior example of two-smartphone synchronization to multiple smartphones, by designating one of them as master, to whom others (slaves) need to synchronize their audio samples. The master communicate with the slaves through WiFi network.

**Signaling protocol.** Figure 6 shows the signaling protocol between the master and each of the slave smartphones. When Dia is launched, the master broadcasts a wireless AP MAC address, whose beacons will be monitored by all smartphones. During the beacon capturing process, the smartphones store the timestamps tuple $\langle GC(t^j), LC_i^{cpu}(t^j)\rangle$. After receiving a sufficient amount of beacons, each smartphone runs the inter-device CPU timing model estimation as in Eq. (3). Upon completing this procedure, each slave smartphone sends a notification message to the master.

Once the master has received notifications from all the slaves, it broadcasts a future global triggering timestamp at which all the smartphones should start audio capturing. Note that, by this time a slave already knows the inter-device timing model parameters and thus can convert the global triggering timestamp to its local timestamp following Eq. (3). Each smartphone, including the master and slaves, waits till its clock hits the local triggering timestamp before it turns on the microphone and starts audio capturing.

During audio capturing, the smartphones keep track of the tuple $\langle LC_i^{cpu}(t^k), k\rangle$ to prepare for the intra-device audio-CPU synchronization. On receiving sufficient number of CPU timestamps corresponding to audio samples, each smartphone runs the intra-device timing model estimation (Eq. (4)). The audio recording runs in parallel to this process. Note that, the intra-device timing model estimation could also be done in the server side. However, this induces tedious message exchanges between the server and the smartphones, who need to send all the CPU timestamps corresponding to each audio samples.

When the user terminates Dia, the master smartphone broadcasts a stop command to all slaves. Each smartphone $i$ then uploads the recorded audio samples $A_i$ along with the inter- and intra-device timing model parameters to the PC server. The server runs Algorithm 1 to synchronize the audio samples of all the slaves with respect to the master smartphone. Specifically, the server first finds out all the global timestamps corresponding to the audio samples of the master smartphone using the timing model parameters. Then, it uses each slave's own timing model parameters to convert the previous global timestamps to the audio sample index. In this way, the audio samples from each slave smartphone get aligned with the master.

**Algorithm 1** Audio I/O synchronization: Server

**foreach** smartphone $i$
  Receive $\langle A_i, \widehat{a}_i(\delta), \widehat{b}_i(\delta), \widehat{\alpha}_i(\delta), \widehat{\beta}_i(\delta)\rangle$
**end foreach**
**foreach** audio sample $j$ in $A_{master}$
  Calculate $LC_{master}^{cpu}(j)$ such that, $j = \widehat{\alpha}_{master}(\delta).LC_{master}^{cpu}(j) + \widehat{\beta}_{master}(\delta)$
  Calculate $GC(j)$ such that, $LC_{master}^{cpu}(j) = \widehat{a}_{master}(\delta).GC(j) + \widehat{b}_{master}(\delta)$
**end foreach**
**foreach** smartphone $i \setminus master$
  Calculate all $LC_i^{cpu}(\cdot)$ such that, $LC_i^{cpu}(\cdot) = \widehat{a}_i(\delta).GC(\cdot) + \widehat{b}_i(\delta)$
  Calculate all sample indices $S_i(\cdot)$ such that, $S_i(\cdot) = \widehat{\alpha}_i(\delta).LC_i^{cpu}(\cdot) + \widehat{\beta}_i(\delta)$
  Extract samples from $A_i$ using indices from $S_i(\cdot)$
**end foreach**

**Practical issues.** There are a number of additional issues we have addressed in implementing the two-level synchronization algorithm.

*(i) Obtaining WiFi beacon timestamps.* We register the beacon arrival times from a single Wi-Fi AP on smartphones, by adding flags inside the smartphone's Wi-Fi driver. In particular, for Galaxy Nexus, we register the timestamps inside ISR of the Broadcom Wi-Fi driver [14] to reduce the extraneous jitters introduce in further OS processing. However, registering CPU timestamps inside ISR triggers a severe software bug inside the driver, which was eventually fixed by applying software patches and driver modifications. We also register the original timestamps that was embedded into the beacon packet which acts as the global logical clock of the AP.

*(ii) Obtaining fine-grained audio sample timestamps.* Audio hardware does not give interrupt until a sufficient amount of audio samples are received. For example, we found out that in Galaxy Nexus, which uses audio hardware of OMAP-4460 [15], the audio driver gets interrupt after every 1056 audio samples, irrespective of the sampling frequency used. Hence, we only receive local CPU timestamps per 1056 audio samples. However, the samples between two CPU timestamps are equally spaced and therefore, to obtain more fine-grained audio sample timestamps, we perform a linear interpolation of the timestamps within the 1056 samples.

*(iii) Scanning a fixed wireless channel for beacons.* The WiFi driver of the smartphones are pre-programmed to scan all available channels (14 in 2.4 GHz range) for AP discovery. We found that this causes excessive latency for Dia to collect beacons from the target AP. We further modified the wireless driver inside the smartphone's Linux kernel, to scan only on a fixed channel over which the AP transmits beacons.

*(iv) Unavailability of beacon timestamps.* Some wireless drivers in smartphones hide the WiFi beacon timestamps from user level programs. Few wireless devices for smartphones are implemented as FullMAC [2], and therefore does not even forward the beacon timestamps received from the wireless access point to the kernel driver. In this extreme situations, the master smartphone itself can act as a source of global timestamps and broadcast periodic packets containing global timestamps.

# 4. ENHANCING AUDIO QUALITY WITH BEAMFORMING

In this section, we introduce how Dia leverages the synchronized smartphone array to develop a beamforming framework for audio enhancement.

## 4.1 MVDR Beamforming Algorithm

We adopt the state-of-the-art Minimum Variance Distortionless Response (MVDR) [16] algorithm to realize audio beamforming. In theory, MVDR can achieve a high spatial resolution and large SNR improvement with a few number of microphones.

To understand how MVDR works, consider an array of $M$ smartphones. Without loss of generality, we assume each smartphone has one microphone, denoted by $P_1, P_2, \cdots P_M$. Due to spatial separation, the signals from the desired sound source arrive at the microphones with different delays. We can construct a delay vector $\mathbf{T} = [t_{1,2}, \ldots, t_{1,i}, \ldots, t_{1,M}]^T$ representing the relative TDOA between $P_1$ and all other microphones, *e.g.* $P_i$. Then the *spatial signature*, $\mathbf{C} = [1, e^{-j\omega t_{1,2}}, \ldots, e^{-j\omega t_{1,M}}]^T$ can represent the phase difference between one smartphone and others at frequency $\omega$. Such signature can be straightforwardly translated from TDOA.

The MVDR algorithm applies complex weight $w_i^*$ to smartphone $P_i$'s audio signals in each frequency bin. The weight will change the phase and amplitude of the signals. If a *weight vector* $\mathbf{W} = [w_1, w_2, \ldots, w_M]^T$ for the $M$ smartphones is properly selected, the desired sound signals can be maintained in the output of summed signals from different smartphones, whereas the noise can be mutually canceled.

[2]A wireless device is called FullMAC, when the MAC layer management functionalities, *e.g.*, beaconing, association *etc.* are implemented in h/w to reduce processing latency.

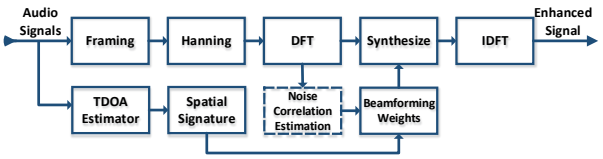


**Figure 7: Audio beamforming module in Dia.**

To attain such a goal, the weight vector $\mathbf{W}$ has to satisfy two conditions:

*(i)* It must steer the audio capturing "beam" to the desired direction, *i.e.*, keeping the gain of the signal strength from that direction to be a constant value 1. This gain constraint can be expressed mathematically as $\mathbf{W^H C} = 1$. In this way, the desired signals can be mutually constructive without loss. This is where the term "distortionless" comes from – to preserve the quality of the desired signals (in both amplitude and phase).

*(ii)* Meanwhile, it should minimize the output energy of noise signals. Intuitively, the weights in $\mathbf{W}$ can be tuned such that the majority of noise can cancel each other. To this end, $\mathbf{W}$ must capture some statistical information — delay spread and energy distribution in frequency domain — from the noise. The statistical information can be characterized by noise correlation matrix $\mathbf{R_n}$, which describes the phase and amplitude relation among noise signals from different smartphones. Detailed steps to obtain the noise correlation matrix will be introduced in Section 4.2.

The above two conditions in MVDR can be jointly formulated as an optimization problem. The optimal weight vector $\mathbf{W}$ has been proven in closed-form [16] as:

$$\mathbf{W} = {\mathbf{R}_n}^{-1}\mathbf{C}(\mathbf{C}^H{\mathbf{R}_n}^{-1}\mathbf{C})^{-1} \quad (5)$$

Below we describe how we incorporate the MVDR theory into Dia's practical beamforming module.

## 4.2 Audio Beamforming System Design

After synchronizing the audio samples from all smartphones, Dia's server coherently combines the samples to realize MVDR, in order to suppress noise and interference. This is realized in Dia's beamforming module, as shown in figure 7, which comprises the following steps.

*(i) Framing the audio signals.* Dia processes audio signals on a per-frame basis. A larger frame improves the precision of statistical parameters (*e.g.*, noise correlation) in MVDR, but will incur greater computational complexity and higher latency. In Dia, we choose an empirical frame length of 40 ms, which is found to strike a good balance. To smooth their transition, the frames are partially overlapped by 20 ms and regulated by a Hanning Window. Then, Discrete Fourier Transform (DFT) is applied to the framed signals to split them into frequency bins.

*(ii) Estimating noise correlation matrix.* The estimation of noise correlation matrix $\mathbf{R_n}$ requires a period that only contains the ambient noise and interference sound. This can often be done prior to the actual audio recording. Let vector $\mathbf{X}_{ik}$ denote the noise signal of frame $k$ from smartphone $P_i$ for one frequency bin (index omitted). The noise correlation matrix can be calculated as:

$$\mathbf{R_n} = \begin{pmatrix} \mathbf{R}_{11} & \cdots & \mathbf{R}_{1M} \\ \vdots & \ddots & \vdots \\ \mathbf{R}_{M1} & \cdots & \mathbf{R}_{MM} \end{pmatrix}$$

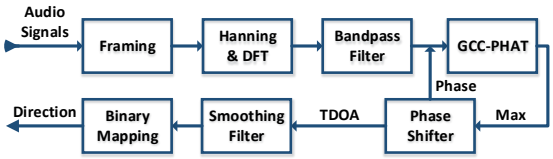


**Figure 8: Speaker tracking module in Dia.**

where $\mathbf{R}_{ij} = \frac{1}{L}\sum_{k=1}^{L} \mathbf{X}_{ik}\mathbf{X}_{jk}^{H}$. $L$ is the number of frames for noise estimation, default to 60 (or 2.4 s) in Dia.

In our current implementation, Dia only tackles relatively stationary noises, *e.g.*, those from electrical appliances, outdoor machinery and vehicles. Suppressing bursty interference requires adaptive estimation of $\mathbf{R_n}$, which we leave for future work.

*(iii) Calculating beamforming weights.* Given the estimated TDOA vector $\mathbf{T}$ (to be discussed in Section 5), Dia's server calculates the spatial signature $\mathbf{C}$. Then, using Eq. (5), it computes the beamforming weights. These weights $\mathbf{W}$ are applied into each of the original frames. The weighted frames are converted back to time domain, thus obtaining signals with enhanced quality.

# 5. SPEAKER TRACKING AND LOCALIZATION

Dia's localization module tracks the speaker's angle relative to the smartphones, based on the TDOA between all the microphones. It can deal with both continuous locations (*e.g.*, a walking lecturer) and discontinuous locations (*e.g.*, role switching in a round-table discussion). Below we detail the components in this module, as shown in Figure 8.

**Framing audio signals.** Dia's voice tracking scheme frames and windows the audio signals in a similar way to beamforming. The key difference is that TDOA estimation is extremely sensitive to multipath reflections. A larger frame size can amortize the impact but it incurs longer latency for voice tracking. In Dia, we use an empirical frame size of 200 ms — 5 times that of the beamforming frame size — to balance this tradeoff.

**Estimating the TDOA.** To estimate the TDOA of each slave relative to the master, we adopt the generalized cross correlation with phase transform (GCC-PHAT) algorithm [17], which has been shown to be effective in practical environment. GCC-PHAT computes the cross-correlation between the audio frames of two microphones in frequency domain, and shifts the relative phase between the two frames. The phase that maximizes the cross-correlation corresponds to the time offset between the frames and hence the TDOA. Before running GCC-PHAT, we apply a bandpass filter to attenuate the signals outside the voice frequency range.

**Enhancing robustness of TDOA estimation.** In implementing GCC-PHAT, we found the discontinuity of human speech induces significant error. Figure 9 shows the GCC-PHAT based TDOA estimation over 20 seconds. Two speakers speak consecutively, each for 10 seconds. Short pauses are intentionally added between speech. The first speaker sits in between two microphones, and thus the TDOA should always equal 0. However, the actual GCC-PHAT estimation varies significantly during the speech gap. We found that speaker movement, ambient noise and the destructive effects of multipath reflections also causes small variations.

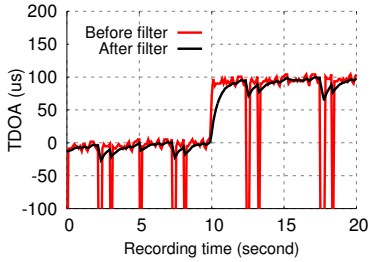


**Figure 9: TDOA over time with speaker switching.**

To resolve these challenges, we introduce a time-domain linear filter, inspired by the Kalman filter, which removes TDOA outliers via smoothing:

$$T_{i+1} = \alpha D_{i+1} + (1-\alpha)T_i \tag{6}$$

where $D_i$ is the GCC-PHAT based TDOA estimation for frame $i$ and $T_i$ the filtered TDOA estimation.

A smaller smoothing factor $\alpha$ can better eliminate outliers, but will cause larger latency during speaker role switching. To balance stability and latency, we adapt $\alpha$ according to the type of TDOA outliers. For minor variation, we choose a larger $\alpha_H$ to reduce latency, because such variations are mostly generated by speaker's movement. Those large variations are incorrect estimations caused by speech gaps. Thus, a smaller value $\alpha_L$ is used for better stability. Formally,

$$\alpha = \begin{cases} \alpha_H & \frac{D_{i+1}}{T_i} < Tres_v \\ \alpha_L & \frac{D_{i+1}}{T_i} >= Tres_v \end{cases}$$

where $Tres_v$ is the threshold determining the type of variation. In Dia, we choose $\alpha_H$, and $\alpha_L$ to be 0.2 and 0.02 respectively. For the lecture room scenario, $Tres_v$ is chosen to be 0.3 as there is only one target speaker. For conference room case, $Tres_v$ is set to be 0.8 because the audio-beam has to switch among multiple speakers.

**Binary mapping: Mapping TDOA to speaker direction.** Given TDOA between microphones, a simple and widely used approach for speaker angle estimation is based on the following relation: $TDOA = \frac{d\sin\theta}{c}$, where $d$ is the distance between two microphones and $\theta$ the speaker's direction relative to the line segment between them. Dia leverages this approach as a basic speaker tracking mechanism.

However, this approach cannot readily take advantage of the availability of more than two smartphones to improve tracking performance. Also, it assumes known distance between the microphones, which is not always feasible to obtain, considering the fact that Dia's smartphones are placed manually and spontaneously. We have experimented with existing audio-based ranging methods like BeepBeep [18] to estimate the distance, but found the blockage between smartphone bodies caused large estimation errors.

In Dia, we design a novel method called *binary mapping* that leverages multiple smartphones to find the speaker direction without knowing the distance between smartphones. Dia's speaker location information is intended for steerable cameras, which has a view angle of tens of degrees [4]. Thus, it is sufficient to divide the area of interest into fan-shaped regions, and estimate the speaker location on a region basis.

Consider a round-table conference scenario, as shown in Figure 20, where $M$ smartphones are placed in the middle to form a circular array, facing $M$ fan-shaped regions. For each region $R_i$, there exists a unique TDOA signature vector

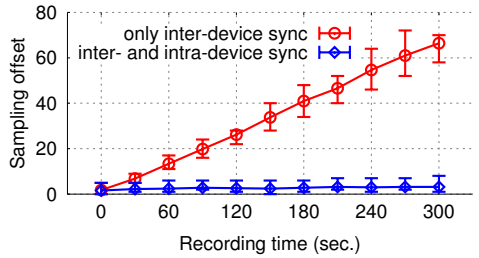


**Figure 10: Audio sampling offset between two smartphones. Error bars denote maximum and minimum.**

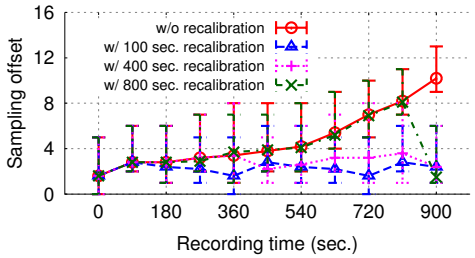


**Figure 11: Audio sampling offset with and without periodic recalibration.**

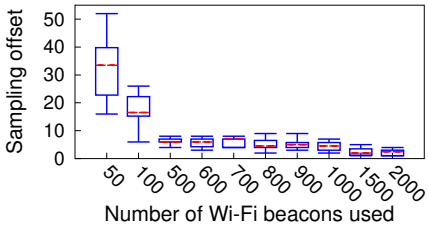


**Figure 12: Synchronization accuracy between two distributed smartphones *w.r.t.* number of initial Wi-Fi beacons collected.**

$\mathbf{S_i} = [s^i_{1,2}, s^i_{1,3}, \ldots, s^i_{1,M}]$, where $s^i_{1,j} = sign(t^i_{1,j})$, and $t^i_{1,j}$ is the TDOA between microphone $P_1$ and $P_j$ when they receive signals from region $R_i$. This TDOA signature is unique regardless of the distance between smartphones or between speaker and smartphones. It can be directly computed without any offline training. The TDOA signature for all regions can be presented in a matrix $\mathbf{S} = [\mathbf{S}_1; \mathbf{S}_2; \ldots; \mathbf{S}_M]$.

At run-time, Dia computes the speaker's signature, and matches it to that of different regions. Specifically, taking microphone $P_1$ as reference, we can obtain a TDOA vector $\mathbf{T}_1 = [t_{1,2}, \ldots, t_{1,M}]$. Then, we map the signed vector $\mathbf{T}_1^{sign} = sign(\mathbf{T}_1)$ to a region by finding the minumum Euclidean distance between $\mathbf{T}_1^{sign}$ and $\mathbf{S}_i$.

$$R_i = \arg\min_i (\mathbf{T}_1^{sign} - \mathbf{S}_i)^2 \quad (7)$$

The Hamming distance of the TDOA signatures is 2 when $M = 8$. Thus, it can tolerate up to two bits of errors in $\mathbf{T}_1^{sign}$. Occasionally, imperfect TDOA estimation and synchronization may induce more than 2 bits of errors. We adopt two approaches to improve the robustness of our tracking algorithm under such cases.

*(i)* In order to mitigate the impact of flipped bit from synchronization offset, we normalize the TDOA vector by its maximum absolute value $\mathbf{T}_i = \frac{\mathbf{T}_i}{\max(abs(\mathbf{T}_i))}$, and set $t_{i,i+1}$ to zero if it is smaller than an empirical value of 0.15. This neutralizes the impact of those small TDOA values.

*(ii)* Instead of only taking one smartphone as reference, we choose multiple smartphones as references, and estimate the region under each reference. Then we run a majority vote to finalize the estimation result.

The above binary mapping algorithm can be applied similarly when the smartphone array is placed in a linear topology. We omit the details due to space constraint.

# 6. SYSTEM EVALUATION

In this section, we first evaluate the individual modules of Dia, including synchronization, beamforming and localization. Then, we conduct a field trial of Dia in conference round-table and lecture room scenarios.

## 6.1 Synchronization Performance

To evaluate the accuracy of our two-level synchronization algorithm, we use a smartphone speaker to repetitively play a short 'chirp' tone. Two other smartphones (Galaxy Nexus I9250 & I515) run the synchronization algorithm and record the chirp signals. To isolate the impact of propagation delay, all the phones are placed within 6 cm distance. Unless noted otherwise, both the testing smartphones run at 16 kHz sampling rate and record the chirp tone for 300 seconds.

The separation of onset peaks of their received chirps represents the residual sampling offset after synchronization (same as our measurement in Section 3.1).

**Synchronization accuracy.** Figure 10 plots the sampling offset between two smartphones over time while performing inter-device and two-level synchronization, respectively. With inter-device (CPU clock) synchronization alone, the synchronization offset is around 2 samples in the beginning, but quickly grows to 20 samples after only 60 seconds of recording. With two-level synchronization, the offset is bounded to 2∼3 samples even after 300 seconds.

However, our synchronization algorithm is not perfect. Figure 11 shows that it may slowly accumulate the residual sampling offset over time. Fortunately, smartphones running Dia can always collect fresh CPU/audio timestamps and periodically request the server to recalibrate the sampling offset without disrupting ongoing recording. With a recalibration period of 100 seconds (default value in Dia), the sync error can be kept at 2 to 3 samples. This translates to a timing error of 187.5 $\mu s$, easily satisfying the 250 $\mu s$ requirement mentioned in Section 3.1.

**Impact of initialization time on synchronization.** Recall that Dia needs to collect WiFi beacons for inter-device synchronization. An immediate question is: how many beacon packets are needed to achieve the desired synchronization accuracy? Figure 12 evaluates the relationship between number of initial Wi-Fi beacons against the sampling offset after running the two-level synchronization algorithm. A larger number of initial beacons allow us to estimate the timing model parameters with better accuracy, thereby increasing the synchronization accuracy. However, beyond 500 beacons, the improvement is marginal. With the default beacon period of 100 ms, this translates to a short initial setup time of 50 s. After the initialization, the two-level synchronization and recalibration runs in background, along with the audio recording, thus eliminating waiting time.

**Impact of audio sampling frequency.** Ideally, the sampling offset should decrease when decreasing the sam-

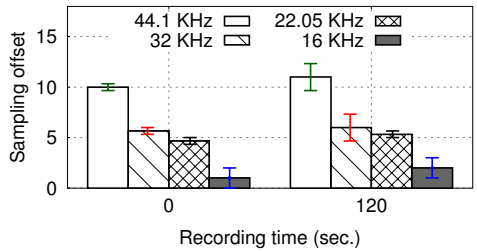


**Figure 13: Sampling offset over time between two distributed smartphones *w.r.t.* audio sampling frequency.**

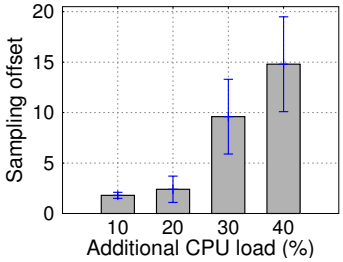


**Figure 14: Sampling offset when one smartphone is subject to additional CPU loads. Error bars represent *std.***

pling frequency, to keep the time delay between the samples constant. Figure 13 verifies this property by showing the sampling offset variation across different sampling frequencies. We see that as sampling frequency decreases, the sampling offset decreases proportionally, and the absolute synchronization time error is almost a constant.

**Impact of background CPU load.** As mentioned in Section 3.3, the jitters in timestamps collected by smartphones depends on the CPU load. We quantify this effect by adding artificial CPU load on top of running Dia, using an open source program, called *cpuloadgen* [19]. This program generates a precise background load on the ARM processor inside the smartphones based on the well-known *dhrystone* [20] benchmark program. Figure 14 shows the resulting sampling offset when one smartphone's CPU load varies. Within 20% of additional CPU load, our synchronization algorithm is virtually unaffected. Beyond that point the sampling offset increases by 4× for only 10% increase in CPU load. The variance also increases significantly. However, we found that background CPU load when running Dia remains at approximately 2∼3%, which is far below the critical load of 20%. In addition, when Dia is running and recording (*e.g.*, during a lecture or meeting), it is highly unlikely that the user will be doing any significant background tasks on the smartphones.

## 6.2 Audio Beamforming Performance

We evaluate Dia's audio beamforming module when running on a synchronized smartphone array. The experiments are conducted in an office environment with ambient noise coming from desktop PCs, servers, and air conditioner. A smartphone playing human voice audio track is used to act as the desired sound source. An additional smartphone generates noise sound with variable power. By default, the smartphones record at 44.1 kHz sampling frequency. We use the conventional peak signal to noise ratio (PSNR) [21] as performance metric. Noise power is measured when the

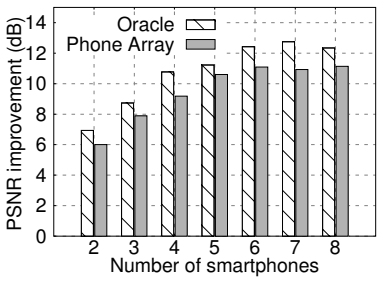


**Figure 15: Beamforming gain from Dia's smartphone array, compared with a perfectly synchronized microphone array. Average PSNR of each individual smartphone is 15.73 dB.**

noise source presents alone. Peak signal power is computed by the maximum total power (measured over consecutive frames containing desired signal and noise) minus noise. PSNR is known to eliminates artifacts caused by speech gap and power variation [21].

**Beamforming gain from a smartphone array.** We compare Dia's beamforming performance with an oracle microphone array with perfect synchronization. The oracle case is created via a trace-driven approach, which represents the best beamforming performance that can be achieved by smartphones. Specifically, we use one smartphone to emulate multiple smartphones by putting it in different but close-by locations and collect the audio data it captures from the same sound source. We play a short chirp at the beginning of each trace, and manually synchronize the trace data offline. Besides, as we use the same smartphone, there is no sampling rate offset in the audio signals.

Figure 15 shows the resulting beamforming gain, *i.e.*, PSNR improvement *w.r.t.* a single microphone case. We observe that the beamforming gain of both Dia and oracle scales with the growing number of microphones. Dia's beamforming gain is 6.01 dB with 2 smartphones, and escalates to 11.14 dB with 8. More importantly, Dia's performance closely approximates the oracle, with a maximum difference of 1.83 dB and below 1 dB in most cases. This is mainly attributed to the small residual synchronization error, which remains almost constant within each calibration period, and thus does not affect Dia's performance in a noticeable way.

**Impact of separation between speaker and microphones.** We examine the impact of distance between speaker and microphone on the beamforming gain. We consider a simple topology of two smartphones separated by 16 cm. The desired sound source is located in the middle of two phones moving 20 cm to 260 cm away from the smartphones, representing decreasing PSNR. Figure 16 shows the resulting beamforming gain. We can see that as speaker walks away, even though both original PSNR and beamforming PSNR decrease, the beamforming gain can still sustain at a high level – around $6dB$. In fact, it increases slightly, from $6.31dB$ at 20 cm to $7.79dB$ at 260 cm. Note that, even when the original PSNR is high, Dia can still boost the audio quality by 6+ dB.

**Impact of sampling offset on beamforming gain.** To verify the necessity of Dia's audio I/O synchronization algorithm, we study the impact of sampling rate offset on beamforming gain. We choose a pair of unsynchronized smartphones which has a sampling rate offset of 0.317 sample/second. We manually align the starting time of their

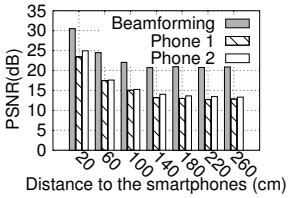


**Figure 16: PSNR variation as speaker walks away.**

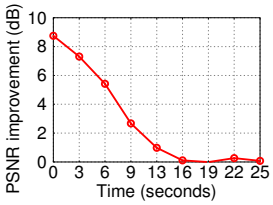


**Figure 17: Impact of sampling rate offset on beamforming gain.**

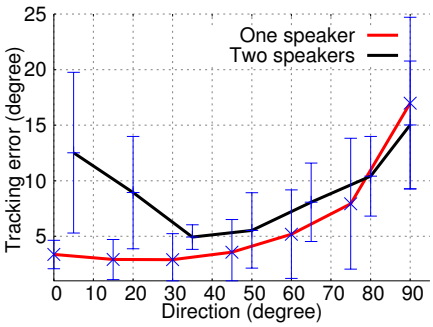


**Figure 18: Voice tracking accuracy when the speaker is located in different directions.**

recorded audio signals to isolate the impact of CPU time offset. From Figure 17, we can see that sampling rate offset degrades beamforming gain significantly over time. After 16 seconds of recording, the beamforming gain drops below 0.12 dB. This experiment stresses the importance of the intra-device synchronization scheme in Dia. It also implies the infeasibility of the audio beacon alignment approach due to the existence of residual sampling rate errors (Section 3.1).

## 6.3 Accuracy of Voice Tracking

In this section, we present voice tracking accuracy when running Dia with two smartphones. We use the basic approach introduced in Section 5 that translates TDOA into the angle of the speaker. The two smartphones are placed 16 cm apart and their middle line is the 0 angle reference. We test the system under two scenarios: *(i)* One speaker who moves from 0 to 90 degrees. *(ii)* Two speakers — one as interferer located at 0 angle, the desired one moving from 5 to 90 degrees.

Figure 18 shows the voice tracking accuracy when the desired speaker is located at different angles. For single-speaker case, the tracking error increases as the speaker's angle increases from 0 to 90 degrees. This is mainly due to multipath reflections: At 0 degree, the speaker is facing both microphones, whereas for 90-degree case, one smartphone is partially blocked by the other, which weakens the line-of-sight path, causing the increased uncertainty of TDOA estimation. Besides, when translating TDOA to speaker angle, the angle estimation $\theta$ becomes more sensitive to $T_i$ when $\theta$ is near 90 degrees (Section 5).

For the case with an interfering speaker, voice tracking can achieve the best accuracy when the two speakers are separated by an intermediate value of around 35 degrees. This is because when two speakers are close to each other, *e.g.*, 5-degree separation, voice tracking cannot reliably differentiate the TDOAs for two speakers. On the other hand, when the speaker is close to 90 degrees, similar tracking error occurs as in the one-speaker case.

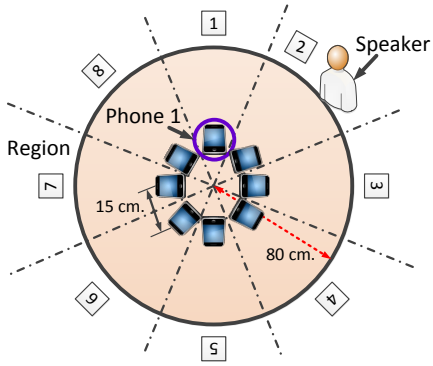


**Figure 19: Experimental setup in a typical round-table conference scenario.**

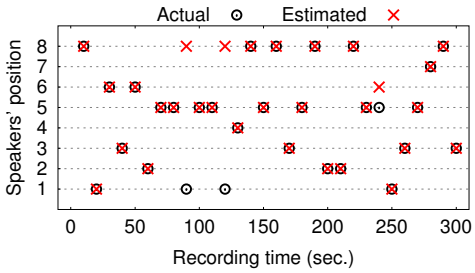


**Figure 20: Speakers' position estimation within 8 regions in a typical round-table conference scenario.**

In all the cases shown in Figure 18, the basic voice tracking module in Dia can differentiate two simultaneous speakers with granularity of smaller than 10 degrees in most cases. However, this module assumes the smartphones' separation is known. In what follows, we evaluate Dia's tracking performance in two field trials, where the smartphone separation may be unknown.

## 6.4 System-Level Testing of Dia

In this section we evaluate the performance of Dia in two application scenarios as proposed in Section 2.

### 6.4.1 *Round-table conference meeting*

We first evaluate Dia in a typical round-table conference scenario. Here, the whole conference room is divided into 8 regions as shown in Figure 19. We place 8 smartphones in the middle of the table in a circular fashion. The distance between the smartphones' microphones is approximately 15 cm (note that Dia is unaware of this distance value). The speaker is 80 cm away from the smartphone array. A region is randomly selected from those 8 regions and the smartphone plays a speech for 10 seconds therein. Since Dia's beamforming performance has been investigated comprehensively, we only focus on its speaker tracking module, particularly the binary mapping algorithm.

**Speaker tracking.** Figure 20 shows the true and estimated positions. We test our binary mapping algorithm for 50 times and observe an average accuracy of 90%. Also, we find that the regions for which the mis-detection occurs is always the adjacent region of the true region (*e.g.*, region 1 *w.r.t.* 8). When four smartphones are used to cover four regions (each 90 degrees), we observed zero region estimation error.

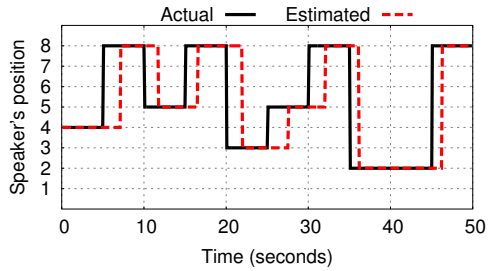


**Figure 21: Switching latency in tracking one speaker to another.**

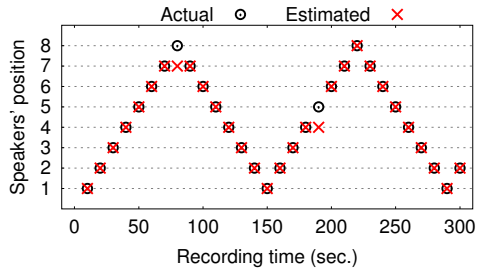


**Figure 22: Speaker's position estimation within 8 linear regions in a typical lecture room scenario.**

**Switching latency.** We now consider a scenario where multiple people around the table speaks consecutively. There voices are created by placing 8 smartphone as speakers in the different regions and randomly triggering one at different times. Under this challenging scenario, Figure 21 shows that Dia's binary mapping algorithm tends to lag behind sudden speaker switching by 1 to 2.5 seconds, with mean of 1.7 seconds. This latency may be acceptable if the external video capturing system (*e.g.*, a steerable camera) does not have stringent real-time requirement, but still leaves room for improvement.

### 6.4.2 *Lecture room*

Similar to 6.4.1, we measured the performance of Dia in typical lecture room environment. We place 8 smartphones in a linear array, with adjacent separation of 7 cm. The speaker is 70 cm away from the array. We divide the whole lecture stage into 8 partially overlapped regions. The speaker moves from one region to another linearly. Each speech lasts for about 10 s and we measured accuracy of the estimated region from the smartphone array. Figure 22 shows the true and estimated speaker position in one experimental run, lasting 300 seconds. We repeat the experiment 8 times and obtained an average accuracy of 93% in speaker region estimation. In addition, we observed similar tracking latency as in the round-table case. However, note that tracking latency occurs only when the speaker crosses the boundary of two regions, which happens less frequently than the previous case.

## 7. DISCUSSION

As an early prototype, Dia leaves a number of problems that deserve further investigation.

**I/O synchronization in heterogeneous platforms.** We have evaluated our I/O synchronization scheme in Galaxy Nexus phones. Even with similar hardware, individual phones exhibit non-trivial CPU timing and audio sampling clock offset. Since Dia's synchronization framework does not rely on any hardware dependent model, we speculate Dia can be easily migrated to other smartphone platforms. A detailed verification will be left as our future work.

**Latency when switching between speakers.** As mentioned in Section 6.4.1, Dia's speaker tracking algorithm tends to lag behind sudden speaker switching in a conference room scenario by 1 to 2.5 seconds. As future work, we will improve Dia's tracking algorithm to strike a better balance between accuracy and responsiveness.

**Dealing with competing speeches.** As shown in Section 5, Dia is able to accurately identify speaker's relative position even when there are two simultaneous speakers. Dealing with more than two speakers is a more challenging scenario. In fact, determining who is the main speaker alone requires human intervention. One possible approach is to degrade Dia to omni-directional recording in this case. This is a matter of our future investigation.

**Real-time audio streaming.** Currently, the synchronized audio samples recorded by Dia's smartphone array are processed by a server offline. An immediate extension is to enable real-time processing, which will be amenable for teleconferencing scenarios.

**Impact of imperfect alignment.** As mentioned in Section 2.1, the application scenarios require the smartphones to be grouped together in regular geometry shapes (*i.e.*, line or circle) to run the beamforming and localization algorithms. However, a small misalignment will not reduce the performance of beamforming, because the MVDR can calculate the TDOA and hence beamforming weights irrespective of the smartphone alignment. Also, Dia localizes the sound source in a coarse-grained, region-basis. The accuracy of the region estimation algorithm is unaffected by small changes in alignment or orientation.

**Multiple microphones on the smartphone.** Certain smartphones like Galaxy Nexus have two microphones, one of which is usually used for noise cancellation. Dia does not leverage the dual microphone capability in its audio capturing. However, we can potentially collect the audio signals from both the microphones (which are synchronized at hardware level) for MVDR beamforming. Dual microphones would not help in Dia's localization algorithm, because they are usually fixed at bottom/back of the smartphones, whereas Dia requires flexible placement of microphones to cover a given region.

**Energy overhead.** The main step of the synchronization algorithm is to collect the beacon timestamps from an AP at periodic intervals and report it to the application. The beacon packets are by default received by the smartphones for AP discovery and thus require no overhead. Only the inter-device synchronization runs locally in the smartphones and the run time is extremely small compared to the actual audio recording process and thus will consume negligible amount of energy. On the other hand, the beamforming and localization algorithms run only on server side and thus will not consume any extra energy in the smartphones. The main energy cost of Dia thus comes from the continuous operation of microphones and transmission of audio samples to the server through WiFi. Such cost is the same as in typical mobile VoIP applications.

## 8. RELATED WORK

**Autodirective audio-visual capturing.** Autodirective audio-visual recording systems have been extensively ex-

plored for group meeting, collaborative teleconferencing, and lecture room scenarios [22]. Smart Room technology [23] adopt vision-based algorithms for participant tracking and identification. High-end video-based telepresence systems [4] are able to sense activities through cameras. In many product-oriented systems, panoramic or steerable cameras are combined with microphones for spatial-aware audio-video capturing [3, 24]. Such systems are becoming increasingly important in this era when spontaneous multimedia capturing and distribution becomes prevalent. However, high-cost and lack of portability limits their usage cases. Our Dia system is designed to cover similar application objectives by simply assembling multiple smartphones.

**Audio beamforming using microphone arrays.** Microphone arrays are often used to emulate the effects of close-up microphones or highly directional microphones to achieve high-quality voice recording. Their unique advantages include hands-free operation and scalable performance with array size. In fact, many contemporary smartphones (*e.g.*, Galaxy Nexus) are equipped with a dual-microphone array for noise cancellation in adverse acoustical environment, *e.g.*, in a cafeteria, on a busy street or in a running vehicle. More powerful microphone array with a large number of elements have been developed. In [25], a 16-microphone array is placed in an office environment for speech enhancement. The LOUD project [21] developed a 1020-microphone array to achieve substantial noise/interference suppression. Adaptive noise suppression and audio beamforming algorithms are studied in [25, 26] and [27] provides an informative survey. Such audio-enhancement algorithms have been applied to in acoustic sensor networks [28] that form a microphone array. Compared with existing work along this line, the main contribution in Dia's beamforming module is the design and implementation of an ad-hoc smartphone array that can serve as a substrate for any acoustic beamforming applications.

**Sound source localization.** Sound source localization has diverse applications including, *e.g.*, animal population measurement, vehicle speaker tracking, and commercial systems in support of smart spaces. These applications heavily rely on microphone arrays for estimating TDOA. Various microphone array platforms have been developed for such acoustic sensing applications [29,30]. Chen *et al.* [31] used a distributed microphone array [30] to localize speakers. The algorithm works well when each speaker is close to one microphone element. Since the microphones are separated by a large distance, localization error caused by synchronization offset becomes negligible. However, sampling-rate offset can still become overwhelming and destroy the accuracy. In [32], multiple microphone arrays are used to estimate the direction of animal sounds. Similar architectural concept has been adopted in [33], where an FPGA based 4-channel microphone array is used to localize shooters. We believe Dia can significantly ease the development and deployment of such sound source localization applications.

**Network synchronization algorithms.** Synchronization is a classical problem for distributed systems and networks. Existing solutions mostly focused on synchronizing the nodes' CPU clocks. In particular, extensive research has been devoted to synchronization in sensor networks (see [34] for a comprehensive survey). The intuition lies in calibrating the CPU clocks through lightweight wireless packet exchanges, in order to achieve network-wide time consistency. Standard synchronization solutions with application-level message exchange (*e.g.*, NTP) can only achieve a granularity of several milliseconds [10]. In Dia, we need to synchronize not only CPU, but also the I/O clocks of different nodes. Distributed synchronization algorithms for microphones on desktop PCs have been proposed in [35,36], yet a practical, quantitative evaluation of the algorithms are still lacking. To our knowledge, Dia represents the first realization of sample-level I/O synchronization between modern smartphones.

**Cooperation between mobile devices.** Current smartphones are designed as standalone personal devices. Emerging device-to-device communications technologies, such as WiFi-direct, create new opportunities for cooperative mobile wireless systems. For example, MicroCast [37] allows closely located smartphones to aggregate the cellular bandwidth, and share downloaded data through WiFi-direct to improve video streaming quality. Recent work, like CWC [38], proposed to harness the computational power of many smartphones to establish a distributed computing platform that offload tasks from costly infrastructures. MobiUS [39] split one video frame between two nearby smartphones to enable better-together viewing experience. Many participatory sensing applications [40] have been developed to collect data and reveal interesting context patterns from a large number of distributed smartphones. Compared with the above research, a key distinguishing challenge in Dia is the granularity of cooperation — it requires tight I/O synchronization between close-by smartphones. To our knowledge, Dia represents the first system to tackle this challenge.

# 9. CONCLUSION

In this paper, we have designed Dia, a novel system that leverages ad-hoc smartphone cooperation to achieve autodirective audio capturing. Dia employs a two-level synchronization framework that can synchronize the audio I/O of distributed smartphones with sample-level accuracy. With this framework, we develop a practical speaker tracking and audio beamforming module that achieves similar performance as a perfectly-synchronized microphone array. Since Dia runs on COTS smartphones and supports spontaneous setup, it has potential to enable a wide range of distributed acoustic sensing and microphone-array based signal processing systems. Our immediate next step to explore Dia's potential is to expand it into a cooperative audio and visual recording system as proposed in Section 2.

# 10. ACKNOWLEDGMENT

We sincerely thank Dr. Wen Hu for shepherding our paper, as well as the anonymous reviewers for their valuable comments and feedback. This work was partly supported by NSF grant CNS-1318292 and CNS-1350039.

# 11. REFERENCES

[1] Polycom Inc., "CX5000," 2013. [Online]. Available: http://www.polycom.com/products-services/products-for-microsoft/lync-optimized/cx5000-unified-conference-station.html

[2] Microsoft Corporation, "Microsoft RingCam / Roundtable," 2006. [Online]. Available: http://microsoft.blognewschannel.com/2006/05/18/microsoft-ringcam-microsoft-roundtable/

[3] D.-S. Lee, B. Erol, J. Graham, J. J. Hull, and N. Murata, "Portable Meeting Recorder," in *Proc. of ACM Multimedia*, 2002.

[4] Z. Yu and Y. Nakamura, "Smart Meeting Systems: A Survey of State-of-the-Art and Open Issues," *ACM Computing Survey*, vol. 42, no. 2, 2010.

[5] Ingeniero Marketing Tecnologia, "Fully Steerable Wireless Micro-Camera," 2013. [Online]. Available: http://www.imtsrl.it/wireless.html

[6] Polycom Inc., "Eagle Eye Director Datasheet," 2013. [Online]. Available: http://www.polycom.com/content/dam/polycom/common/documents/data-sheets/eagleeye-director-ds-enus.pdf

[7] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[8] J. Benesty, J. Chen, and Y. Huang, *Microphone Array Signal Processing*. Springer, 2008.

[9] M. Vorlander, *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*. Springer, 2008.

[10] M. Laner, S. Caban, P. Svoboda, and M. Rupp, "Time Synchronization Performance of Desktop Computers," in *IEEE International Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, 2011.

[11] The NTP newsgroup, "Network Time Protocol," 2013. [Online]. Available: http://www.ntp.org/

[12] R. Myers, *Classical and modern regression with applications*. Duxbury Press Belmont, CA, 1990, vol. 2.

[13] P. J. Rousseeuw, "Least median-of-squares regression," *American Statistical Association*, vol. 79, 1984.

[14] Broadcom Corporation, "brcmfmac (sdio) drivers," 2013. [Online]. Available: http://wireless.kernel.org/en/users/Drivers/brcm80211

[15] Texas Instrument Inc., "OMAP4460 Multimedia Device - Technical Reference Manual," 2013.

[16] J. J. M. V. de Sande, "Real-time Beamforming and Sound Classification Parameter Generation in Public Environments," Master's thesis, Delft University of Technology, 2012.

[17] "A High-Accuracy, Low-Latency Technique for Talker Localization in Reverberant Environment," Ph.D. dissertation, Brown University, 2000.

[18] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan, "BeepBeep: a High Accuracy Acoustic Ranging System Using COTS Mobile Devices," in *Proc. of ACM SenSys*, 2007.

[19] Texas Instruments Inc., "cpuloadgen," 2013. [Online]. Available: http://github.com/ptitiano/cpuloadgen

[20] "Dhrystone Benchmarking for ARM Cortex Processors," 2011. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0273a/index.html

[21] E. Weinstein, K. Steele, A. Agarwal, and J. Glass, "A 1020-Node Modular Microphone Array and Beamformer for Intelligent Computing Spaces," 2004.

[22] Y. Rui, A. Gupta, J. Grudin, and L. He, "Automating Lecture Capture and Broadcast: Technology and Videography," *ACM Multimedia Systems Journal*, vol. 10, 2004.

[23] C. Busso, S. Hernanz, C.-W. Chu, S. il Kwon, S. Lee, P. Georgiou, I. Cohen, and S. Narayanan, "Smart Room: Participant and Speaker Localization and Identification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2005.

[24] R. Cutler, Y. Rui, A. Gupta, J. Cadiz, I. Tashev, L.-w. He, A. Colburn, Z. Zhang, Z. Liu, and S. Silverberg, "Distributed Meetings: a Meeting Capture and Broadcasting System," in *Proc. of ACM Multimedia*, 2002.

[25] T. Takagi, H. Noguchi, K. Kugata, M. Yoshimoto, and H. Kawaguchi, "Microphone Array Network for Ubiquitous Sound Acquisition," in *Proc. of IEEE ICASSP*, 2010.

[26] M. Jeub, C. Herglotz, C. Nelke, C. Beaugeant, and P. Vary, "Noise Reduction for Dual-Microphone Mobile Phones Exploiting Power Level Differences," in *Proc. of IEEE ICASSP*, 2012.

[27] Y. Huang, J. Benesty, and J. Chen, *Acoustic MIMO Signal Processing*. Springer, 2006.

[28] H. Luo, J. Wang, Y. Sun, H. Ma, and X.-Y. Li, "Adaptive Sampling and Diversity Reception in Multi-hop Wireless Audio Sensor Networks," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2010.

[29] D. Sun and J. Canny, "A High Accuracy, Low-latency, Scalable Microphone-array System for Conversation Analysis," in *Proc. of ACM UbiComp*, 2012.

[30] L. Girod, M. Lukac, V. Trifa, and D. Estrin, "The Design and Implementation of a Self-calibrating Distributed Acoustic Sensing Platform," in *Proc. of ACM SenSys*, 2006.

[31] M. Chen, Z. Liu, L.-W. He, P. Chou, and Z. Zhang, "Energy-Based Position Estimation of Microphones and Speakers for Ad Hoc Microphone Arrays," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2007.

[32] A. M. Ali, K. Yao, T. C. Collier, C. E. Taylor, D. T. Blumstein, and L. Girod, "An Empirical Study of Collaborative Acoustic Source Localization," in *Proc. of ACM/IEEE IPSN*, 2007.

[33] J. Sallai, P. Völgyesi, A. Lédeczi, K. Pence, T. Bapty, S. Neema, and J. R. Davis, "Acoustic Shockwave-Based Bearing Estimation," in *Proc. of ACM/IEEE IPSN*, 2013.

[34] F. Sivrikaya and B. Yener, "Time Synchronization in Sensor Networks: a Survey," *IEEE Network*, vol. 18, no. 4, 2004.

[35] D. Budnikov, I. Chikalov, I. Kozintsev, and R. Lienhart, "Distributed Array of Synchronized Sensors and Actuators," in *Proc. of European Signal Processing Conference (EUSIPCO)*, 2004.

[36] Y. Jia, Y. Luo, Y. Lin, and I. Kozintsev, "Distributed Microphone Arrays for Digital Home and Office," in *Proc. of IEEE ICASSP*, 2006.

[37] L. Keller, A. Le, B. Cici, H. Seferoglu, C. Fragouli, and A. Markopoulou, "MicroCast: Cooperative Video Streaming on Smartphones," in *Proc. of ACM MobiSys*, 2012.

[38] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy, "Computing While Charging: Building a Distributed Computing Infrastructure Using Smartphones," in *Proc. of ACM CoNext*, 2012.

[39] G. Shen, Y. Li, and Y. Zhang, "MobiUS: Enable Together-viewing Video Experience Across Two Mobile Devices," in *Proc. of ACM MobiSys*, 2007.

[40] D. Estrin, "Participatory Sensing: Applications and Architecture," *IEEE Internet Computing*, vol. 14, no. 1, 2010.