MNTP: Enhancing Time Synchronization for Mobile Devices

Sathiya Kumaran Mani[†], Ramakrishnan Durairajan[†], Paul Barford[†+], Joel Sommers[*]
{sathiya,rkrish,pb}@cs.wisc.edu, jsommers@colgate.edu

[†]University of Wisconsin - Madison [+]comScore, Inc. [*]Colgate University

ABSTRACT

Clock synchronization between Internet hosts is important in a variety of applications including gaming, finance and measurement. While clock synchronization issues in wireline networks have been well studied, mobile hosts present challenges that have not received as much attention. In this paper, we describe a study of clock synchronization in mobile hosts, which often implement a simplified version of the Network Time Protocol (NTP), known as SNTP, due to resource constraints typical of mobile devices. We begin by reporting an analysis of logs from NTP servers that highlights the significant differences in synchronization behavior of wireline vs. wireless hosts. This analysis motivates a laboratory-based study of the details of clock synchronization on mobile hosts, which reveals the causes and extent to which synchronization can become misaligned. We then describe a new protocol that we call Mobile NTP (MNTP), which is designed to be simple, efficient and easy to deploy. We implement MNTP on a wireless laptop and demonstrate its capability over a range of operating conditions. We find that MNTP maintains clock synchronization to within 25ms of a reference clock, which is over 12 times better than standard SNTP.

CCS Concepts

•**Networks** → **Network protocol design; Mobile networks;**

Keywords: SNTP; Wireless; Mobile; Time; Measurement

1. INTRODUCTION

The notion of time and the invention of mechanisms to track it have a long history. One perhaps inevitable aspect of time was the need to synchronizing clocks in separate geographic locations. This capability first emerged in earnest in the 19th century to enable complex railway schedules to be managed [48]. Modern computing and communication systems rely intrinsically on time synchronization for basic operation and in a wide variety of distributed applications.

The basic operation of time synchronization in a distributed environment assumes the presence of a high precision reference clock, which serves as the source of accurate time for other systems. Remote systems interact with the reference clock periodically to synchronize (discipline) their local clock. The primary challenges in such a distributed environment are *(i)* the variation in crystal oscillator quality and environmental conditions (which determine clock drift[1]), *(ii)* characteristics of the network paths that separate the remote host from the reference clock (which can limit synchronization accuracy [21]), and *(iii)* the protocol that is used to discipline the clock on the remote host.

The efforts of Mills led to standardization of the Network Time Protocol (NTP) in 1985 [37]. NTP is widely used in the Internet today. At its core are high precision reference clocks (typically atomic clocks or GPS-based clocks) that form the foundation of a widely distributed hierarchy of public time servers. The protocol mechanisms are based on periodic exchanges of information between clients and servers. However, the specific behavior of any particular client and its level of synchronization with a server can vary widely [52]. Furthermore, for devices that do not require *all* the performance and accuracy benefits of NTP, a subset of NTP called Simple NTP (SNTP) [39] is used.

While the use of mobile computing devices, including smartphones, has exploded over the past decade, to the best of our knowledge, very little attention has been paid to their particular challenges for time synchronization. We argue that the need for tight time synchronization for applications such as gaming, finance and measurement will become increasingly important as mobile handset use continues to grow. We also argue that several of the key challenges for time synchronization noted above—in particular, the possibility of dramatic changes in environmental conditions and highly variable path conditions between clients and servers—require detailed consideration in order to understand the efficacy of standard approaches.

[1]In this study, we follow the definitions described in §2 of Paxson *et al.* [45] for the terms *drift*, *offset*, *skew* and *accuracy*.

IMC 2016, November 14-16, 2016, Santa Monica, CA, USA
 ISBN 978-1-4503-4526-2/16/11…$15.00
DOI: http://dx.doi.org/10.1145/2987443.2987484

In this paper, we address the issue of time synchronization in mobile systems. Our first objective is to understand the time synchronization behavior of mobile hosts in the Internet today. Our specific objective is to assess the time synchronization protocols and configurations that are used by mobile clients and in particular the one-way delay latencies that they experience, which can limit synchronization. This aspect of our study serves as a baseline for the second part of our investigation. Specifically, our second objective is to understand the range of behaviors of mobile time synchronization under a range of wireless operating conditions. This leads to the final component of our study, which is the specification and examination of Mobile NTP (MNTP), a new protocol for time synchronization of mobile clients.

We begin our study by examining logs gathered from 19 NTP servers located in the US over a period of 24 hours. Our focus is on the wireless hosts that are identified in the logs using simple heuristics based on their IP addresses. The logs include time synchronization requests from over 86,312 unique US mobile hosts. The logs reveal that over 95% of the mobile hosts use the SNTP simplifications to NTP for synchronization. The logs also reveal that typical one-way delays (OWD) to mobile hosts vary between 1ms to 997ms. The combination of the use of SNTP and highly variable OWDs suggests that tight time synchronization in mobile hosts is likely to be quite challenging.

We expand our examination by conducting a series of controlled laboratory experiments. The focus of these efforts is to assess the impact of SNTP and variable path conditions on the ability of mobile hosts to synchronize with a reference server. To facilitate repeatability of our laboratory experiments, we develop a scriptable tool which is able to modify the transmission power of a wireless base station as well as flexibly inject different forms of cross-traffic in order to create a wide range of wireless network conditions. From our experiments using this tool, we find that the time *offsets*—the measured difference between our clock's time and the 'true' time according to the national standards—relative to a reference server are highly variable in *all* wireless settings compared with similar experiments carried out using a wired network. A wireless host running SNTP with the same clock hardware under the same ambient temperature conditions, achieves time offsets with a mean of 4ms and standard deviation of 7ms on a wired network compared to time offsets with a mean of 31ms and standard deviation of 47ms on a wireless network.

Motivated by the findings from our laboratory experiments, we next describe Mobile NTP (MNTP). MNTP is designed for mobile hosts, which are assumed to experience highly variable and lossy channel conditions that present challenges for standard time synchronization protocols. As a result, the core idea in MNTP is cross-layer-awareness of wireless channel conditions. MNTP is designed to be simple and easily deployable, with minimal modifications to the SNTP implementation of mobile hosts. MNTP is resistant to both clock drift and variability and lossy channel conditions, while still operating in a lightweight manner, thereby enabling accurate time synchronization for mobile devices.

We demonstrate the capabilities of MNTP in a laboratory setting over a range of operating conditions. Using the same laboratory testbed as our initial experiments, we compare MNTP's synchronization accuracy with SNTP[2] by comparing the offset measurements that would be used to discipline the local clock. In our evaluations, we find that MNTP offers dramatic improvement over SNTP. In one particular experiment in which we allow the local NTP process to update the system clock, we find that MNTP's maximum offset is 23ms, and its average is 12 time better than for SNTP.

The remainder of this paper is organized as follows. In §2 we provide background on NTP and the SNTP. We present our empirical study of NTP based on NTP server logs and laboratory-based experiments, which focuses specifically on wireless hosts, in §3. This is followed by a description of MNTP in §4. We demonstrate the capabilities of MNTP and compare it to SNTP in §5. Next, we provide a review of related literature in §6. Finally, we summarize and discuss future work in §7.

2. BACKGROUND

Various applications and services hosted on telecommunication networks require accurate time synchronization for correct and consistent operation. Achieving this requirement is complicated by *clock drift*, which is caused by differences in environmental conditions or crystal oscillator quality and leads to clocks advancing at different rates. To maintain clock synchronization among independently running networked-hosts, the Network Time Protocol (NTP) was created in 1985 (RFC 958 [37]). Due to the heterogeneity of computing and network devices (*e.g.*, mobile phones vs. routers vs. desktops) and their varying requirements (*e.g.*, tolerance to less accurate time vs. bandwidth vs. battery life), several variants of NTP including the Simple Network Time Protocol (SNTP) (RFC 1769 [39]) and Precision Time Protocol (PTP) [27] have been developed.

Precise time estimates from a hierarchy of time sources (also known as *stratum servers*) can be obtained using NTP, SNTP or PTP. At the top-level of the hierarchy are stratum 0 servers. GPS receivers and/or atomic crystals are used by these high-quality time sources to correct clock drifts. Stratum 0s offer highly-precise time estimates to stratum 1 servers, *i.e.*, to servers in the next level of the hierarchy. Stratum 1s are also referred to as *primary* servers. *Secondary* or stratum 2 servers connect to stratum 1s, and so on, all the way down to stratum 15. Stratum servers typically respond with time estimates for synchronization requests from a wide variety of hosts supporting any of the three protocol variants. Hosts in the Internet synchronize time with multiple stratum servers, typically at level 2 or higher.

Wired Internet hosts traditionally run the `ntpd` daemon in order to synchronize their clock using NTP. This software implements several NTP-standard filtering heuristics to select the best time samples from the stratum servers. `ntpd` operates by exchanging timestamps with its reference servers (in

[2]We do not compare against NTP (which uses an exhaustive filtering and selection pipeline for accepting packets) but plan to do so in future work.

a process called *polling*). The polling behavior is governed by the *clock discipline algorithm* [7] and, as part of this procedure, `ntpd` determines the optimal polling interval from the measured round-trip delay, jitter and oscillator frequency [9].

Unlike the hosts in the wired Internet (*e.g.*, routers, desktops, and traditional laptops), wireless hosts such as mobile devices typically use SNTP to acquire time estimates from stratum servers. SNTP sets all fields in an NTP packet to zero except the first octet and does not employ the sophisticated clock correction and filtering algorithms of NTP. Hence the accuracy of the time estimates obtained using SNTP are always lower [39]. Besides SNTP, wireless devices also support a mechanism called Network Identity and Time Zone (NITZ) to update clocks in a one-off fashion [5,6]. NITZ is a weaker mechanism to obtain time information as the estimates are not obtained in a periodic fashion like NTP and are dependent on the device crossing a network boundary.

In addition to the issues in SNTP and NITZ, commodity operation systems (OS) shipped with wireless hosts inhibit accurate clock synchronization due to a variety of vendor-specific implementations, including differing frequencies at which the synchronization requests are emitted and number of retries on prior request failures. Specifically, upon detailed analysis of code from Android's codebase [1], we found that Android SNTP implementations poll once a day if data from NITZ are unavailable. To further aggravate the issue, Android performs only three retries upon error and updates the system time *only* if the estimate differs by more than 5000ms. Similarly, the Windows Mobile OS updates the system clock once every 7 days. Even if the synchronization request fails, no further retries are sent.

3. CLOCK SYNCHRONIZATION OVER WIRELESS LINKS

In this section, we analyze the problem of accurate clock synchronization in mobile hosts. We begin by examining NTP server logs[3] and classifying clients as wired or wireless. We then discuss the basic characteristics of latencies for the various hosts identified in the logs. Next, we describe a series of laboratory-based experiments conducted with wired and wireless hosts that are designed to quantify the extent of time *skew*, which is defined as the measured frequency difference between our clock and the 'true' time clock, in these devices. We conclude by discussing the results from the logs and experiments, which motivate the need for a new scheme for mobile host clock synchronization.

3.1 NTP Server Logs

Dataset. In this study, we rely on a new source of network latency measurements—passively collected traffic from NTP servers. We reached out and explained our research objectives[4] to several NTP server administrators. Eight administrators responded positively with *tcpdump* traces from a total of 19 NTP servers across 9 states in the US. We developed a light-weight tool based on `netdissect.h` and `print-ntp.c`, available as part of [15], to process the logs and extracted relevant information such as total number of unique clients, one-way delays (OWD) to and from NTP servers and the time synchronization protocol used using the filtering heuristic described in Durairajan *et al.* [23]. The heuristic is necessary to infer the state of synchronization of the client's clock with respect to the NTP server, in order to eliminate invalid latency measurements.

Basic Characteristics. Table 1 summarizes the basic statistics and some key attributes of the 19 NTP servers such as a server's stratum number, IP version, total number of measurements collected, and number of distinct clients. The NTP servers used in our study include 5 stratum 1's and 14 stratum 2's with a combination of both IPv4 and IPv6 support. This dataset, collected over a period of one day, include a total of 209,447,922 OWD measurements to 17,823,505 unique clients, as indicated by the unique number of IP addresses seen in server logs.

Wired vs. Wireless. To classify the hosts seen in the NTP logs into wired or wireless, we use Team Cymru's IP-to-ASN mapping service [16] and group hosts based on AS number and provider name in hostnames. We follow a simple process that leverages keywords and provider names (*e.g.*, mobile, cloud, Amazon, Sprint, etc.) present in hostnames to classify clients into various service provider categories (described below). Even though the keyword-based classification is fairly rudimentary, we argue that it is sufficient enough to highlight wired vs. wireless service providers.

Figure 1-(left) shows a comparison of the minimum OWDs of clients belonging to various service providers that are listed based on the average of minimum OWDs as seen in logs of three NTP servers[5]. Four categories of latency characteristics are evident: *(1)* cloud and hosting providers (SP 1–3) exhibit very low minimum OWDs with a median of 40ms, *(2)* Internet service providers (SP 4–9) show a medium trend with a median of 50ms minOWDs, *(3)* broadband providers (SP 10–21) exhibit high latency characteristics with median delays of about 250ms, and *(4)* mobile providers (SP 22–25) exhibit very high latencies with high interquartile ranges and median latencies as high as 550ms. To complement the latency characteristics shown in Figure 1-(right), we plot the distribution of minimum OWDs of clients belonging to top 25 service providers, which are ranked based on the number of unique IP addresses, as depicted in Figure 1-(left). One of the striking characteristic of the extracted latencies is the linear trend of the mobile providers (note SP 22–25). We hypothesize that this characteristic is related to the broader geographic distribution of mobile wireless clients compared with fixed-location wireline clients. For all servers, 50% of the hosts from the three mobile providers exhibit a latency of more than 400ms and this observation is consistent across all the 19 NTP servers.

[3]To preserve anonymity, NTP server and service provider names (in results) are removed.

[4]We carefully selected a number of NTP administrators from pool.ntp.org based on server location and stratum, and explained our goals here: http://ntp-study.cs.wisc.edu/

[5]The remaining 16 NTP servers exhibited similar characteristics and are not shown here due to space constraints.

Table 1: **Summary of client statistics seen in the NTP logs.**

Server ID	AG1	CI1	CI2	CI3	CI4	EN1	EN2	JW1	JW2	MW1	MW2	MW3	MW4	MI1	SU1	UI1	UI2	UI3	PP1
Unique Clients	639,704	606	359	335	262	228	232	12,769	35,548	2,746	9,482,918	1,141,163	2,525,072	1,078,308	21,101	36,559	18,925	1,77,957	128,644
Server Stratum	2	2	2	2	2	2	2	1	1	1	2	2	2	1	1	2	2	2	2
IP Version	v4	v4/v6	v4/v6	v4/v6	v4/v6	v4/v6	v4/v6	v4	v4	v4	v4	v4	v4	v4	v4/v6	v4	v4	v4	v4/v6
Total Measurements	9,988,576	1,480,571	1,268,928	812,104	7,63,847	4,11,253	4,37,440	3,54,530	8,69,721	1,97,900	46,232,069	10,948,402	11,126,121	63,907,095	16,404,882	18,426,282	14,194,081	9,254,843	2,369,277

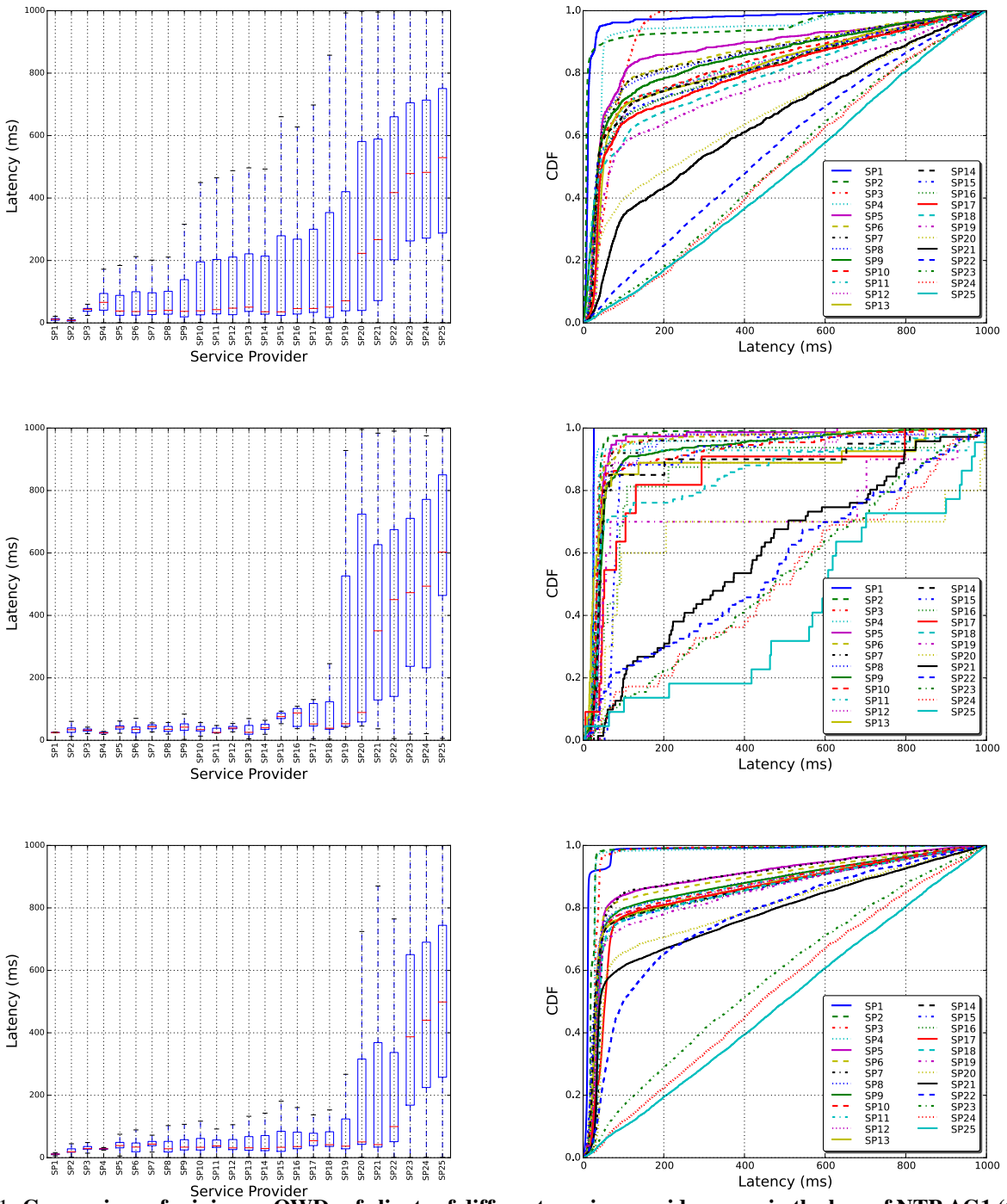

Figure 1: **Comparison of minimum OWDs of clients of different service providers seen in the logs of NTP AG1 (top-left), JW2 (middle-left) and SU1 (bottom-left). CDF of minimum OWDs of clients of different service providers seen in the logs of NTP servers AG1 (top-right), JW2 (middle-right) and SU1 (bottom-right).**

SNTP vs. NTP. Next, we classify the hosts into SNTP or NTP based on the protocol used for clock synchronization, as seen from the tcpdump traces of the servers. Figure 2-(left) shows the percentage of SNTP vs. NTP hosts seen across the 19 NTP servers, where a majority of hosts seen across all servers (except CI1-4 and EN1-2, which are ISP-specific

NTP servers) use SNTP to synchronize time. For each server, we also calculate the percentage of SNTP vs. NTP clients for the top 25 service providers observed in the NTP logs and find that over 95% of the clients of mobile providers use SNTP—a result consistent with other information (*e.g.*, Android codebase [1]). Because of simplifications in SNTP compared with NTP, we can expect these hosts not to be able to synchronize with the same level of accuracy as full NTP clients. We seek to quantify this limitation, next.

3.2 Wireless Experiments

In this section, we seek answers to the question: what is the combined impact of wireless effects such as channel fading, interference due to adjacent channels, signal attenuation, etc. on time synchronization procedure used by mobile hosts? To address this question, we use the laboratory-based wireless testbed shown in Figure 3. The architecture of our testbed is designed to provide control over network delay characteristics, *e.g.*, to repeatably introduce large and variable delays, with the goal of evaluating the presence of wireless hop(s) on SNTP-based time synchronization between hosts and their corresponding references. Our testbed consists of three types of nodes: *(1)* a wireless access point, *(2)* a target node, and *(3)* a monitor node.

For the wireless access point (WAP) in our testbed, we turn a laptop with a 802.11 b/g/n wireless adaptor into a WiFi hotspot. The WAP is connected to the wired Internet and shares its connection to the other nodes in the testbed. Both the monitor node (MN) and the target node (TN) are associated with the WAP, making the last hop completely wireless. The WAP has the ability to programmatically increase or decrease the transmission power (within legal limits) of the wireless adaptor upon receiving commands from the MN.

SNTP and NTP measurements are all launched from the target node (TN) in the testbed. When TN is booted, it is configured to use its default OS-specific NTP server such that TN's system clock is in tight synchronization throughout the experiments. In our experiments, the TN is a Macbook pro laptop and the default NTP server used is *time.apple.com*. Once the TN has achieved synchronization with its OS-specific NTP server, we record TN's system clock offset—which we call the *true time offset*—obtained by using the *ntpq* utility as the baseline in our evaluations. We launch SNTP requests to `0.pool.ntp.org` (also known as a pool server) to get the SNTP-based time estimates from a remote reference clock over the wireless network[6]. Every SNTP request to the pool server is randomly assigned to a new NTP time reference enabling unbiased time server selection. All responses from the pool servers include exact time at the remote NTP server and TN's clock offset with respect to remote server's clock. We compare these reported offsets with the true time offset to quantify the effect of the wireless hop on clock synchronization. The TN also sends statistics collected through active measurement to the MN using tools like `ping`, which are targeted to a user-configured probe destination. To characterize the impact of the wireless hop on the time estimates obtained from reference servers, all experiments were repeated by connecting the TN to the wired Internet.

The monitor node (MN) in the testbed manipulates the delay and variability characteristics in the wireless last hop which are required for our experiments. In order to achieve the required characteristics, the MN employs two strategies: *(i)* the WAP's outgoing Internet connection is occupied intermittently by downloading a large file at random intervals from a fixed download destination (*i.e.*, `ubuntu.com/download`), and *(ii)* control commands are sent to the WAP to randomly increase or decrease the WAP's transmission power and file download frequency. The channel occupancy is determined by the file download pattern and duration of the traffic and are automatically tuned at the MN based on channel statistics reported by the TN. Specifically, if the latencies of ping probes reported by TN increases, as observed from the number of packet losses in ping probes, the file download frequency is decreased and the transmission power value is increased thereby making the channel less lossy and dynamic. Otherwise, the frequency of downloads and transmission power are increased and decreased respectively. Once the channel stabilizes, as denoted by no packet losses in ping traffic, our tool automatically responds by a decrease in transmission power and increase in download frequency, making the channel conditions variable and lossy at random intervals.

We note that wireless experiments that include interference using our scriptable tool are difficult to repeat *exactly* since we are experimenting in a live environment. However, we argue that repeating the same set of experimental steps will lead to results that have the similar statistical properties to those we report. In addition, the RTS/CTS feature which is used to help cope with hidden-terminal situations was disabled for the experiments described below. Given the introduction of additional variable delays due to RTS/CTS, we would expect the performance of SNTP to be even worse with this feature enabled.

Figure 4 shows the SNTP time offsets reported by the TN with (left) and without (right) reference to a NTP time source from `0.pool.ntp.org`. Specifically, the figures depict the clock offsets reported from SNTP when the TN is connected to the *(1)* wired network with and without NTP time as baseline and *(2)* wireless network with and without NTP time reference. From these figures, the impact of a variable and lossy wireless channel on the time synchronization is clearly evident. In particular, the mean and standard deviation of the offset for the wireless experiments with NTP clock correction[7] are 31ms and 47ms, and for the experiments without clock correction they are 118ms and 133ms. In contrast, on a wired network, when TN's system clock is corrected using NTP, the offsets reported by SNTP requests with respect to

[6]We note that the use of an NTP pool server is commonly employed in mobile wireless devices, such as Android-based devices.

[7]We use the term 'NTP clock correction' to refer to the scenario where TN's system clock is corrected using NTP—specifically, using NTP's sophisticated sample filtering and clock selection heuristics. Furthermore, SNTP only reports the offset of TN's system clock with respect to the NTP time source (`0.pool.ntp.org`). This helps us measure the error in the clock offsets reported by SNTP relative to an expected clock offset of 0ms.

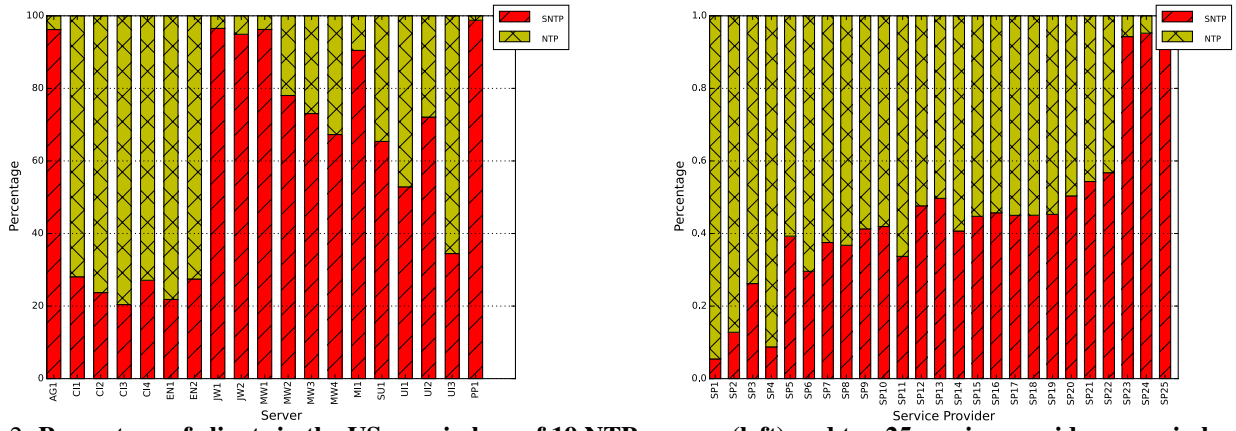

Figure 2: **Percentage of clients in the US seen in logs of 19 NTP servers (left) and top 25 service providers seen in logs of SU1 NTP server (right) using NTP and SNTP protocol.**

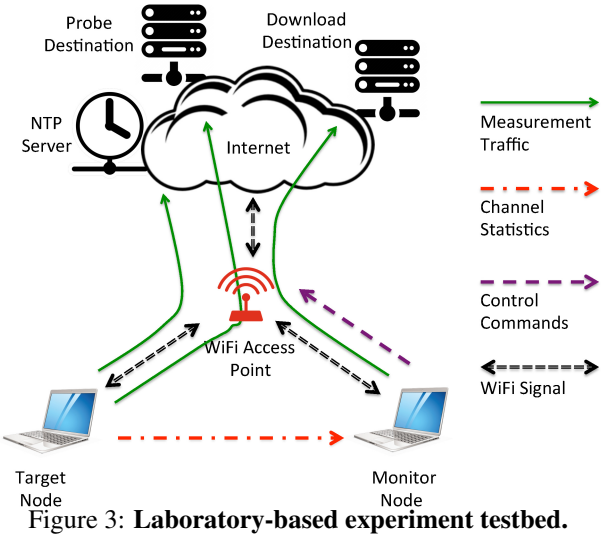

Figure 3: **Laboratory-based experiment testbed.**

the reference clocks are always close to 0ms. Even when the clock correction is suspended on a wired network, the drift is steady and is dependent on the temperature of the vendor-specific oscillator present in the device.

In a network with varying load and delay characteristics, which is not atypical in wireless deployments [18], the offsets obtained from SNTP requests often exhibit high variance as shown in Figure 4-(left) (in green), despite NTP correcting TN's system clock. For example, the offset was as bad as 600ms compared to the NTP reference clock during a highly-varying and lossy channel condition (at x=245), which was absent in the wired counterpart of the experiment. Furthermore, the presence of a wireless hop and the impact of channel effects on clock synchronization is more pronounced in the absence of an NTP-based reference clock as shown in Figure 4-(right), where the reported SNTP offsets were as bad as 1.58s (at x=522). We also considered different hardware platforms and repeated the experiments at different times of the day to verify this behavior. We do not show results of these additional configurations due to space constraints, but what we found in our experiments was that the time synchronization mechanism is susceptible to wireless effects, regardless of hardware platforms and/or time of the day.

3.3 Cellular Network Experiments

We repeat the experiments described in §3.2 by replacing the TN with a mobile phone—specifically, Samsung Galaxy S4 with Android OS v4.4.4 (KitKat)—and ran the experiments for 3 hours on a 4G network (of SP 22) without MN and download traffic. To establish a baseline time for comparisons, we use the SmartTimeSync app [13] to correct the device's system clock. To obtain SNTP offsets from the time source (`0.pool.ntp.org`), we use SNTP Time app [10]. Since the SNTP Time app does not support logging capability, we modified the codebase given here [11] to log the SNTP offsets computed against the system clock that is corrected using GPS fixes. Figure 5 shows the clock offset reported by SNTP Time app after GPS correction. We see in the figure that the reported SNTP offset can vary significantly, and was as high as 840ms. Overall, the mean offset was 192ms with a standard deviation of 55ms. Similar to §3.2, we repeated the experiments multiple times at different times of the day and found the poor synchronization behavior to hold across the runs.

3.4 Discussion

A natural question is whether GPS or NTP might be used to correct major clock offsets in mobile and wireless devices. We argue that using GPS-based time synchronization is not a widely applicable solution, as many mobile and wireless devices have minimal or no support for GPS, especially in the developing world. Even if the devices have a built-in GPS receiver or support for a GPS module, vendor-specific OS implementations (*e.g.*, iOS) often prohibit GPS-based time synchronization [3, 4]. Moreover, GPS availability can depend on location (*e.g.*, GPS valleys such as building and tunnels) and has been observed to be power-hungry in mobile devices [43, 51]. Should these limitations resolve at some point in the future, the GPS would be an attractive option.

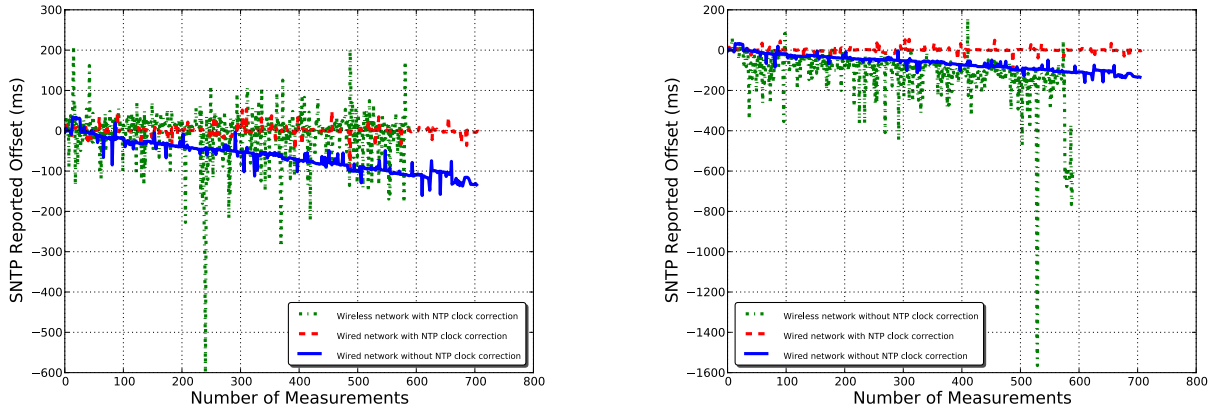

Figure 4: **Comparison of SNTP clock offsets in wired vs. wireless environments with (left) and without (right) NTP clock correction.**

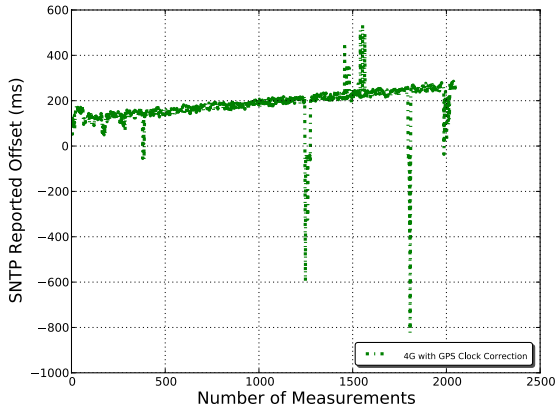

Figure 5: **SNTP clock offsets reported by mobile host on a 4G network.**

Using NTP for time synchronization is also a less attractive solution for mobile devices because of its heavier-weight communication characteristics, including periodic polling behavior and requirement to maintain state at the devices. We posit that these characteristics are ill-suited for mobile devices and would have a negative impact on battery life due to the following two reasons. First, an outcome of NTP's polling algorithm is that a time source is polled intermittently with ∼128B of data by the client. A measurement study by Balasubramanian *et al.* has shown that a few 100B transfers periodically on mobile phones with 3G/GSM technology can consume more energy than bulk one-shot transfers [19]. Second, another effort by Haverinen *et al.* showed that UDP-based protocols (such as NTP) require frequent keep-alive messages to maintain state at the devices, which in turn can lead to short battery lifetimes [26].

Main Findings and Implications. SNTP uses clock offset to update the local clock directly and none of the time-tested filtering algorithms as would be used in a full NTP client. As a result, the significant and highly variable offsets we observe have a direct impact on the synchronization of the client's clock. Because of the disadvantages of using a GPS- and NTP-based clock synchronization, SNTP has been widely deployed on mobile devices, but as our experiments show, it provides poor time synchronization. These findings call for a new clock synchronization mechanism or the need to modify an existing mechanism to support mobile/wireless devices.

4. MNTP DESIGN

To enhance the accuracy of the time synchronization under varying and lossy conditions described in §3, we contend that mobile devices should dynamically adapt to the channel conditions and pace their requests to reference servers based on the collected channel information. In order to decide when to emit SNTP requests to servers during *ideal* channel conditions, we must answer the following three questions:

- What wireless channel information should devices use to detect an ideal/stable portion of the channel?
- Over what timescale should the wireless environment be monitored to identify an ideal channel condition?
- At what interval (across ideal conditions) should requests be sent to synchronize devices with reference(s)?

In this section, we attempt to answer these questions and use the answers to guide the design of Mobile NTP, a lightweight, simple and easy-to-deploy modification of SNTP, which improves clock synchronization for mobile devices.

4.1 Measured Channel Information

Similar to §3.2, we monitor the following channel information (or *wireless hints*)—specifically, Received Signal Strength Indication (RSSI) (in decibels) and noise level (in decibels)—to calculate the Signal-to-Noise Ratio (SNR) margin (defined as RSSI - Noise) and guide the establishment of ideal channel conditions (§4.2) at the host, which needs its clock synchronized with a time reference. We obtain these hints directly from the wireless adaptor of the wireless device. We note that all these hints, among other information, can easily be obtained for a variety of hardware platforms. For

example, the wireless hints for mobile devices can be measured using techniques and tools available as part of Zhang *et al.* [54]. Similarly, for Mac OS-, Windows OS-, and linux-based laptops, the wireless hints from the channel can be obtained using the *airport* utility (for Mac), *iwconfig* utility (for linux) and other free surveying tools [17].

Algorithm 1: MNTP clock synchronization algorithm

input : *warmupPeriod* = time to estimate clock offsets
input : *warmupWaitTime* = interval between requests in warmupPeriod
input : *regularWaitTime* = interval between requests in regularPeriod
input : *resetPeriod* = duration of warm-up plus regular periods

```
1  inWarmup = True
2  recordedOffsets = {}
3  driftEst = None
4  if inWarmup then
       // Acquire offset only when channel is stable
5      wait(favorableSNRCondition())
6      offset = getOffsetUsingMultipleSources()
7      if accept(offset) then
8          measureSystemClock(offset)
9          recordedOffsets.add(offset)
10     wait(warmupWaitTime)
11     if exitingWarmup then
12         driftEst = estimateDrift(recordedOffsets)
13         inWarmup = False
14     goto Step 4
15 else
16     correctSystemClockDrift(driftEst)
       // Acquire offset only when channel is stable
17     wait(favorableSNRCondition())
18     offset = getOffsetUsingSingleSource()
19     if accept(offset) then
20         correctSystemClock(offset)
21         recordedOffsets.add(offset)
22     wait(regularWaitTime)
23     if elapsed(resetPeriod) then
24         goto Step 1
25     else
26         goto Step 4
27 Function estimateDrift(recordedOffsets)
28     trendLine = leastSquaresFit(recordedOffsets)
29     driftEst = getSlope(trendLine)
30     return driftEst
```

4.2 MNTP Overview and Algorithm

Overview. In this section, we describe a new clock synchronization protocol called MNTP. MNTP is sensitive to varying channel conditions and reuses/sends SNTP requests in a channel-aware fashion based on wireless hints from the channel. Using the wireless hints that are captured from the device (as described above), we *monitor* the channel for stable and less-dynamic portions and *pace* the clock synchronization such that we send the SNTP requests *only* during those identified stable channel conditions—the twin goals of MNTP.

Algorithm. The key steps of MNTP are shown in Algorithm 1. The algorithm starts with identification and establishment of ideal channel conditions and needs the following: *(1)* baseline thresholds for the wireless hints to filter bad offsets and *(2)* time period over which the channel should be observed so that SNTP requests can be emitted.

To create a baseline threshold for the wireless hints, we set the values of RSSI, noise and SNR margin to be -75 dB, -70 dB and 20 dB respectively. That is, RSSI value should be greater than -75 dB, noise level should be lesser than -70 dB and the SNR margin should be greater than or equal to 20 dB. These values are not arbitrary, rather they emerged through an iterative process of refining our experiments and information given in online forums [2,12,14]. In our experiments, we find that the baseline thresholds work effectively, but we expect that these parameters would likely need fine tuning for other external or outdoor settings.

We follow a two-part approach in MNTP's design to determine the time period over which the channel should be monitored and requests can be sent to reference clocks. The first part of the algorithm is the warm-up phase (steps 4 to 14), where the wireless host is put on a test period for *warmupPeriod* time at the start of the device and/or the algorithm. In this test period, the channel condition is determined based on the measured wireless hints (step 5) and SNTP requests are emitted to three reference clocks. Specifically, the SNTP requests are emitted to `0.pool.ntp.org`, `1.pool.ntp.org` and `3.pool.ntp.org`, on every *warmupWaitTime* interval, in parallel, till the *warmupPeriod* (step 6). These requests are emitted *only* when the measured wireless hints satisfy the established baseline thresholds. Otherwise the synchronization requests are deferred. For the offset responses received from time sources, we follow the philosophy of NTP's clock selection heuristic [8]. We calculate the mean and standard deviation of the offsets and classify the time sources whose offsets exceed the mean plus one standard deviation as *false tickers*. We reject the false tickers to ensure very tight clock synchronization.

Since a network could be completely lossy at the start of clock synchronization, we may not be able to emit any SNTP requests. To address this issue, we wait until the network conditions are favorable and record 10 offset values reported by SNTP to create a trend line for the clock skew or clock drift (steps 7 to 9). The waiting time is determined by the *warmupPeriod* and *warmupWaitTime* variables, which are user-tunable parameters and which depend on the level of accuracy needed by users of the apps installed on hosts. It is possible that our algorithm never perceives the network conditions to be favorable, or at least not over a very long

period of time. In this work, we do not consider such perpetually unstable network conditions and plan to address such scenarios in future work.

Once the initial trend is created, the direction of the subsequent reported offsets is determined based on the established trend line and an accept or reject decision is made. Specifically, we find the squared error of each of the reported offset with respect to the fitted trend line and then extend the trend line to get an estimate of where the next sample should be. Next, we calculate the error of the reported offsets with respect to this estimate. If the square of that error is one standard deviation above or below the mean, then we reject the reported offset. Otherwise the offset is accepted and the trend line is extended (steps 11 to 14). MNTP's filtering capability ensures accurate clock synchronization by rejecting large offsets as we show in §5. Since the clock skews are not always linear and the constant skew factor of the clock dominates its variable counterpart [42], we fit a trend line using least squares polynomial fit with a first degree polynomial. We note that our algorithm does not make any assumptions about linearity in clock skew.

Finally, once the trend lines are determined and when the warm-up phase is complete, MNTP starts the regular phase for clock synchronization. Note that the actual clock update happens *only* during the second part of the algorithm (steps 16 to 26). This step is similar to the warm-up phase with the exception of two things: *(i)* the interval at which the requests are emitted to references, which is based on the *regularWaitTime* variable; and *(ii)* SNTP requests are emitted to a single reference clock only. The actual clock update and drift correction mechanisms vary, depending on vendor-specific system calls available to MNTP.

Advantages. MNTP has the following advantages over standard SNTP for clock synchronization. First, MNTP is *efficient* in terms of how the synchronization requests are paced and emitted based on the wireless channel conditions. In addition, simple examination of MNTP's CPU utilization during tests indicate loads less than 0.5%. Our initial focus on efficiency has been from a network load perspective and even though MNTP's CPU impact is low, we intend to consider the efficiency of our implementation from a system load perspective in future work. Next, MNTP algorithm (Algorithm 1) is implemented in about 274 lines of python code[8], which conveys the fact that MNTP is *simple*. Next, MNTP is also easily *deployable* with minimal support from the end hosts. Specifically, the *only* support needed form the wireless hosts is that the wireless host should allow MNTP to measure wireless hints. Finally, MNTP can also offer *accurate* clock synchronization to wireless/mobile devices as explained in §5.

5. MNTP EVALUATION

In this section, we evaluate the clock synchronization accuracy of MNTP compared with an unmodified SNTP implementation. We begin by conducting a set of baseline laboratory-based experiments with SNTP and MNTP in which synchronization requests are emitted every 5 seconds for one hour during varying channel conditions using the same experimental setup (including noise generation) as in §3. In these experiments, we do not consider warmup and regular periods, and we switched off the drift correction feature in MNTP to create a head-to-head comparison with SNTP. For the similar reason, in all our experiments, we *only* compare MNTP against SNTP and not NTP, because of the NTP's exhaustive filtering overhead. Our focus for these one-hour experiments is to highlight the reduction in the reported clock offsets by MNTP versus SNTP, and to experiment in a wide variety of operating conditions including tests on *(1)* wireline and wireless networks and *(2)* with and without NTP for system clock correction.

Next, we conduct experiments for 4 hours to demonstrate the efficacy of MNTP. All the experiments were run on a Macbook Pro laptop (with Intel core i5 processor and 4GB RAM) and the wireless hints are measured using the *airport* utility. Finally, we analyze the effect of MNTP's parameters on its synchronization accuracy through a trace-driven analysis on the logs collected in the longer experiments.

5.1 Baseline experiments

We begin by considering a simple wireless network scenario with variable channel conditions and compare the clock offsets reported by MNTP and SNTP. Throughout this experiment, we use NTP to correct the laptop's system clock from which the clock synchronization requests are emitted. Using NTP to establish a baseline is simply a design choice and can be easily replaced with a GPS receiver, which we plan to investigate in future work. Figure 6 shows the clock offsets reported by both SNTP and MNTP when the laptop is connected on a wireless network along with the large offsets that are rejected by MNTP's filter. We can see that the offsets reported by SNTP are susceptible to the varying channel conditions and are as much as 292ms from the system clock. However, the offsets reported by MNTP are very close to the system clock with a maximum offset value of 23ms, which is a 12-fold improvement over standard SNTP on a wireless network with lossy conditions. From this figure, it is also evident that all the outlier offsets are effectively discarded by our MNTP filter, thus enabling much tighter clock synchronization.

To explain the gains achieved using MNTP versus SNTP, we plot the measured wireless hints in Figure 7. Both the reported and rejected offsets by the MNTP filter are also shown. We note that the advantages of MNTP are due to the following two properties of MNTP's filtering and channel-aware request pacing heuristics. First, many of the synchronization requests are deferred due to either RSSI, or noise, or SNR margin not meeting the established baseline thresholds. Second, many of the large reported offsets are effectively rejected by the MNTP filter allowing only those offsets that are close to the clock drift trend line.

[8]We have open-sourced our MNTP implementation and it is available for download at https://github.com/satkum/mntp

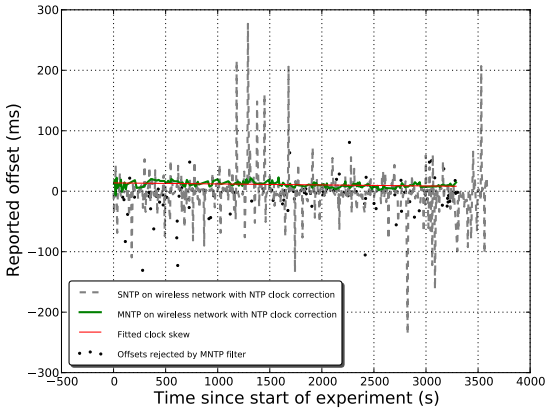

Figure 6: **Reported SNTP vs. MNTP offsets on wireless network with NTP clock correction.**

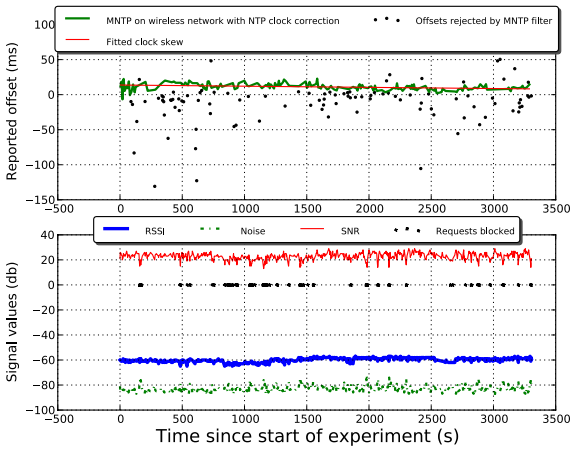

Figure 7: **Signals and selection plot.**

Next, we repeat the above experiment without allowing NTP to correct the system clock. Figure 8 depicts the offsets reported by SNTP and MNTP on wireless network without NTP-based clock correction. The figure highlights the disadvantages of using SNTP on a wireless network, with reported clock offsets as high as 450ms. On the other hand, the clock offsets reported by MNTP were always close to the fitted trend line for the clock skew, which we believe will easily be corrected using our estimated drift available as part of the MNTP filter. The maximum offset reported by MNTP is 24ms, which is, on average, within 4.5ms of the reference clock and 17 times more accurate than standard SNTP.

Third, we test the clock synchronization accuracy of MNTP when the host is connected to a wireless network compared with SNTP when the same host is connected to a wired network. In this experiment, NTP is turned on and the clocks are synchronized with the OS-specific NTP time source before starting the experiments. Figure 9 depicts the clock offsets reported by SNTP and MNTP on a wired and wireless network respectively. The figure shows that even when the SNTP offsets were obtained from a wired network, SNTP can be as high as 50ms compared to MNTP on a wireless network, where reported offsets are about 20ms. The gains we see with MNTP are, again, due to emitting requests only during favorable network conditions, whereas SNTP emits requests blindly, and rejection of outliers, whereas SNTP does no such rejection.

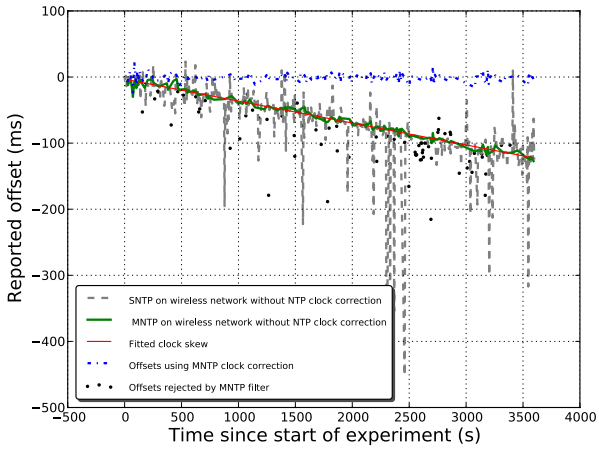

Figure 8: **Reported SNTP vs. MNTP offsets on wireless network without NTP clock correction.**

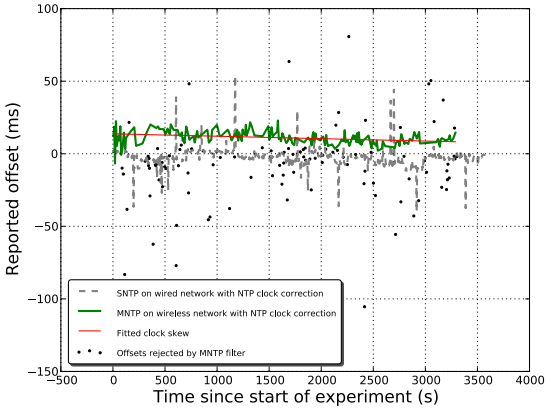

Figure 9: **Reported SNTP offsets on wired network vs. MNTP offsets on wireless network with NTP clock correction.**

Similarly, we compare the offsets reported by MNTP on a wireless network against offsets reported by SNTP when connected on a wired network without NTP clock correction. Figure 10 plots the clock offsets reported by SNTP and MNTP on a wired and wireless network respectively without NTP clock correction. Similar to the third experiment, we observe that the SNTP offsets reported were as high as 50ms despite the host being connected to a wired network.

5.2 Longer experiment

Next, we demonstrate the efficacy of MNTP by extending the baseline clock synchronization experiment for a longer duration. In this experiment, we follow the setting described in §5.1 and we emit synchronization requests every 5 seconds for a duration of 4 hours. Specifically, we record the

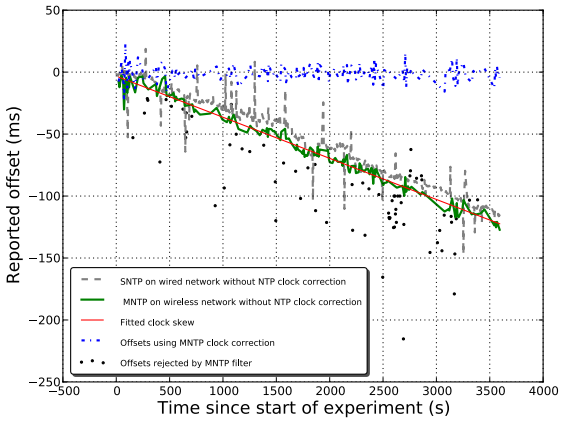

Figure 10: **Reported SNTP offsets on wired network vs. MNTP offsets on wireless network without NTP clock correction.**

offsets reported by SNTP client connected to a wireless host and simultaneously collect the offsets reported by MNTP. Throughout the experiment, the wireless host's clock is allowed to drift, *i.e.*, the clock correction using NTP is turned off.

Figure 12 shows the comparison of the offsets reported by standard SNTP and MNTP on a wireless network without NTP's clock correction. The fitted trend line for clock skew and the clock correction feature capabilities of MNTP along with the large offsets rejected by MNTP are also shown. We again observe high offsets with SNTP—as high as 392ms—whereas the offsets reported by MNTP are always less than 20ms, as seen from MNTP's clock corrected drift values (shown in blue). Further, we see that many of the large offsets reported by SNTP are effectively rejected by MNTP's filtering logic producing a robust and accurate clock synchronization for wireless devices despite the presence of lossy channel conditions.

5.3 Tuning MNTP

In order to tune the four parameters of MNTP—specifically, warmupPeriod, warmupWaitTime, regularWaitTime and resetPeriod—shown in Algorithm 1 for various indoor and outdoor settings, we built a stand-alone tool called the *MNTP tuner*. At the core of the MNTP tuner tool is the ability to perform trace-driven analysis on the recorded clock offset values. Using the MNTP tuner tool, our goal is to evaluate the impact of various input parameter choices and the suitability of parameters in a given network condition.

MNTP tuner consists of three components: *(a)* a logger, *(b)* an emulator, and *(c)* a searcher. First, the logging component runs on the TN of our testbed and emits SNTP requests to multiple reference clocks every 5 seconds and records the responses in the form of traces. It also records the corresponding wireless hints from the channel every time an SNTP request is emitted. Next, the emulator is capable of running the MNTP algorithm using the captured traces and wireless hints and prints the offsets reported by MNTP.

To run the emulator, the four input parameters required by MNTP need to initialized, which is satisfied by the search component. When provided with a range of values for the input parameters, *e.g.* the duration of the warm-up or regular waiting times, the search component generates all possible values of the parameters and invokes the emulator for each generated combination. The search component then calculates the accuracy of MNTP for the given combination of input parameters using the offsets reported by the emulator. Specifically, the search component measures the Root Mean Square Error (RMSE) of the MNTP offsets with respect to a perfectly synchronized clock (*i.e.*, offset value of 0 ms) and outputs the various configurations. A sample list of configurations, along with the values for MNTP parameter and RMSE of clock offsets, are shown in Table 2. The corresponding achievable clock offset values for the six sample configurations are depicted in Figure 11. The values indicate that while the RMSE can indeed be reduced as the number of tuning requests grows, relatively speaking, MNTP performs well with only modest tuning.

Apart from tuning the input parameters, the tool can also be used to gain useful insights about the factor that influences the accuracy of MNTP for a given network condition. For example, using the analysis tool, we discovered that sometimes the MNTP filter did not accept any offsets during the regular period for some values of warmupWaitTime. Investigating this further, we discovered that the number of samples were too low causing the MNTP to underestimate the clock drift value. And because of this underestimation of the clock drift, the MNTP filter was too conservative in accepting the offsets resulting in all the offsets being rejected in the regular phase. This insight led us to change our MNTP filter to reestimate the clock drift value with every new offset sample.

6. RELATED WORK

Accurate clock synchronization of networked systems has been a subject of interest in the research community for over three decades. The pioneering work by Lamport [30], Marzullo [33], and Mills [35, 36] paved the way for the development of synchronization protocols such as the near-ubiquitous Network Time Protocol [34, 37, 40]. While NTP and the related SNTP have been deployed widely in ordinary wireline and wireless networks, time synchronization protocols have also been developed for more specialized deployments such as wireless sensor networking [22, 24, 25], underwater sensor networks [20, 31, 32] and acoustic networks [50].

There has been substantial work in analyzing the NTP's synchronization accuracy, which has led to improvements to the protocol and to new protocols *e.g.*, [21, 38, 46, 47, 52]. Our work on MNTP relates to this research in that we design mechanisms to improve the accuracy of clock synchronization using SNTP through wireless channel-aware pacing of synchronization requests, and we propose improvements to the filtering and clock correction heuristics. An issue related to clock synchronization is that of correctly processing packet trace data containing timestamps from two in-

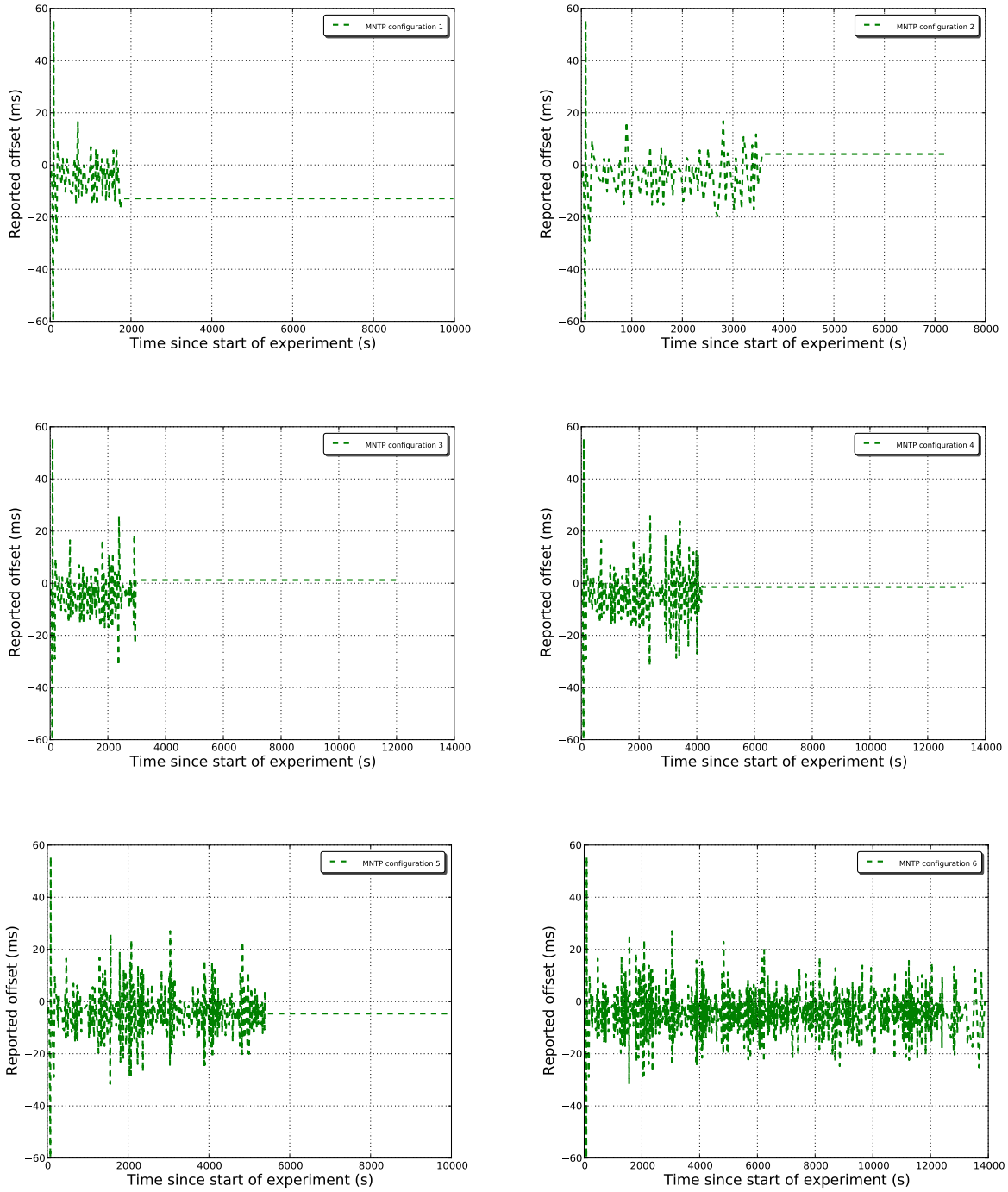

Figure 11: **Outputs produced by our MNTP tuning emulator.**

Table 2: **Sample list of values for parameters, RMSE of offsets, and the number of requests generated by MNTP tuner.**

Configuration no.	warmupPeriod (min)	warmupWaitTime (min)	regularWaitTime (min)	resetPeriod (min)	RMSE (ms)	Number of requests
1	30	0.25	15	240	13.08	239
2	40	0.25	15	240	11.66	316
3	50	0.25	15	240	11.09	387
4	70	0.25	30	240	10.86	534
5	90	0.084	15	240	9.27	1210
6	240	0.084	15	240	8.9	2913

dependent clocks by correcting for clock skew and other artifacts [41, 44]. Key findings from these studies that inform the design of MNTP are that clock skew can be non-linear and highly erratic.

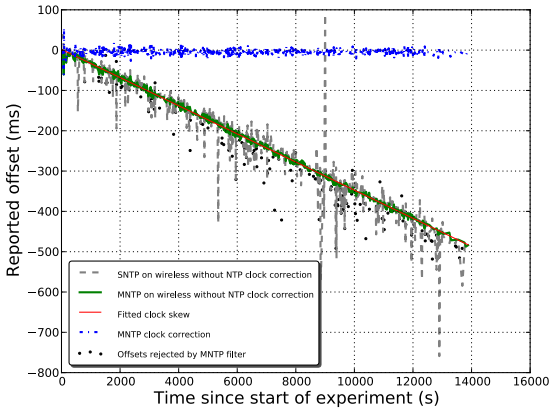

Figure 12: **Reported SNTP versus MNTP offsets on wireless network without NTP clock correction. The experiments were ran for 4 hours.**

The idea of measuring and exposing wireless hints to higher protocol layers is similar to other efforts in the area of cross-layer design of mobile systems [28, 29, 49, 53]. In particular, MNTP considers wireless conditions when emitting synchronization requests in order to avoid collecting clock offsets that may be skewed due to channel contention. To the best of our knowledge, MNTP exploits cross-layer information for enhancing clock synchronization in mobile and wireless devices in a way that has not been considered in prior studies.

7. SUMMARY AND FUTURE WORK

Synchronizing independent clocks in a network setting is important for a wide variety of applications and has been extensively studied. The problem of clock synchronization on mobile wireless devices poses particular challenges and has received little attention in prior work. Our study seeks to address this gap, with the goal of improving clock synchronization on mobile wireless hosts and with low overhead.

We begin by examining the one-way delay characteristics of likely wireless hosts through analysis of packet traces collected at 19 NTP servers in the US, observing high average delays as well as high variance. In particular, for three predominantly wireless service providers, we see median latencies of 550 ms and high variance, whereas for other service providers we see median latencies of 40–50ms with low variance. Through further analysis of the traces, we attribute these characteristics to the fact that the majority of these clients implement SNTP, and thus do not employ any of the mechanisms used in NTP that would generally eliminate poor latency samples.

In a set of laboratory testbed experiments, we further examine the degree to which SNTP exhibits poor synchronization. We create a tool that enables wireless interference to be programmatically altered through manipulation of base station transmit power and introduction of cross-traffic. We find that the mean time offset with respect to a reference clock of the latency samples observed with SNTP is significantly higher in experiments over a wireless network (100's of ms) than in experiments with a wired network (close to zero).

Based on the results of our laboratory experiments, we describe MNTP, a set of modifications to SNTP that improve its behavior by *(1)* exposing wireless link-layer information to pace out polling request to NTP servers when wireless channel conditions are favorable, and *(2)* adding lightweight filtering heuristics which result in discarding outlier latency samples. We evaluate MNTP in a laboratory setting, comparing it with SNTP. Our results show that MNTP is able to maintain synchronization with a reference clock to within 25 ms, which is a 12-fold improvement over SNTP.

In future work, we intend to extensively test and enhance our MNTP implementation toward the goal of broader deployment. Unfortunately, there are no known implementations of NTP for any of the popular mobile operating systems, which complicates this effort. However, we plan to build a reference NTP implementation and perform an exhaustive benchmarking of MNTP against SNTP and NTP in terms of metrics like processor and battery performance on various mobile platforms.

We also plan to investigate self-tuning of parameter settings and to evaluate MNTP in a wider variety of cellular and WiFi settings and to evaluate the trade-offs between MNTP's performance and the tuning of its parameters. We intend to carry out longer-term *in situ* experiments in order to evaluate not only the trade-offs but also MNTP's effectiveness in day-to-day operation.

Acknowledgements

We thank the NTP operators for providing NTP server logs, and our shepherd Sharon Goldberg and the anonymous reviewers for their valuable comments. This material is based upon work supported by the NSF under grant CNS-1054985, DHS grant BAA 11-01 and AFRL grant FA8750-12-2-0328. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF, DHS or AFRL.

References

[1] Android Codebase. https://source.android.com/source/.
[2] Cisco wireless site survey FAQ. http://www.cisco.com/c/en/us/support/docs/wireless-mobility/wireless-lan-wlan/68666-wireless-site-survey-faq.html#qa23.
[3] Getting time from GPS on iOS. http://stackoverflow.com/questions/31920829/getting-time-from-gps-on-ios.
[4] Is it possible to get the atomic clock timestamp from the iPhone GPS. http://stackoverflow.com/questions/1444456/is-it-possible-to-get-the-atomic-clock-timestamp-from-the-iphone-gps .
[5] NITZ functionality for non-3gpp devices. http://goo.gl/xQUECH.
[6] NITZ Service Description. http://www.3gpp.org/DynaReport/22042.htm.
[7] NTP Clock Discipline Algorithm. https://www.eecis.udel.edu/~mills/ntp/html/discipline.html.
[8] NTP Clock Select Algorithm. https://www.eecis.udel.edu/~mills/ntp/html/discipline.html.

[9] NTP Polling Interval. http://www.eecis.udel.edu/~mills/ntp/html/poll.html.

[10] NTPSync app. https://github.com/Free-Software-for-Android/NTPSync.

[11] NTPSync code snippet. https://github.com/Free-Software-for-Android/NTPSync/blob/master/NTPSync/src/main/java/org/ntpsync/service/NtpSyncRemoteService.java.

[12] Relationship between SNR and RSSI. http://community.arubanetworks.com/t5/Controller-Based-WLANs/What-is-the-relationship-between-data-rate-SNR-and-RSSI/ta-p/178312.

[13] Smart GPS Time app. https://play.google.com/store/apps/details?id=com.pautinanet.smartgpstime&hl=en.

[14] SNR and RSSI values. https://supportforums.cisco.com/discussion/10954591/snr-and-rssi-values.

[15] Tcpdump Tool. https://github.com/the-tcpdump-group/tcpdump.

[16] Team Cymru whois lookup service. http://www.team-cymru.org/IP-ASN-mapping.html#whois.

[17] Windows and Mac Stumbler and Survey Tools. http://www.networkworld.com/article/2925081/wi-fi/7-free-wi-fi-stumbling-and-surveying-tools-for-windows-and-mac.html .

[18] AKELLA, A., JUDD, G., SESHAN, S., AND STEENKISTE, P. Self-Management in Chaotic Wireless Deployments. *Wireless Networks* (2007).

[19] BALASUBRAMANIAN, N., BALASUBRAMANIAN, A., AND VENKATARAMANI, A. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *ACM IMC* (2009).

[20] CHIRDCHOO, N., SOH, W.-S., AND CHUA, K. C. MU-Sync: A Time Synchronization Protocol for Underwater Mobile Networks. In *WuWNeT* (2008).

[21] D. VEITCH AND K. VIJAYALAYAN. Network Timing and the 2015 Leap Second. In *PAM* (2016).

[22] DAI, H., AND HAN, R. TSync: A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks. *SIGMOBILE CCR* (2004).

[23] DURAIRAJAN, R., MANI, S., SOMMERS, J., AND BARFORD, P. Time's Forgotten: Using NTP to Understand Internet Latency. In *ACM HotNets* (2015).

[24] ELSON, J., GIROD, L., AND ESTRIN, D. Fine-grained Network Time Synchronization Using Reference Broadcasts. *OSDI* (2002).

[25] ELSON, J., AND RÖMER, K. Wireless Sensor Networks: A New Regime for Time Synchronization. *SIGCOMM CCR* (2003).

[26] HAVERINEN, H., SIREN, J., AND ERONEN, P. Energy Consumption of Always-on Applications in WCDMA Networks. In *IEEE VTC* (2007).

[27] IEEE. IEEE 1588 Precision Time Protocol (PTP), Version 2 Specification, March 2008.

[28] KHAN, S., PENG, Y., STEINBACH, E., SGROI, M., AND KELLERER, W. Application-driven Cross-layer Optimization for Video Streaming over Wireless Networks. *IEEE Communications Magazine* (2006).

[29] KUMAR, S., CIFUENTES, D., GOLLAKOTA, S., AND KATABI, D. Bringing Cross-layer MIMO to Today's Wireless LANs. In *ACM SIGCOMM* (2013).

[30] LAMPORT, L. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of ACM* (1978).

[31] LIU, J., ZHOU, Z., PENG, Z., CUI, J.-H., ZUBA, M., AND FIONDELLA, L. Mobi-Sync: Efficient Time Synchronization for Mobile Underwater Sensor Networks. *IEEE TPDS* (2013).

[32] LU, F., MIRZA, D., AND SCHURGERS, C. D-sync: Doppler-based Time Synchronization for Mobile Underwater Sensor Networks. In *WuWNet* (2010).

[33] MARZULLO, K. *Loosely-coupled distributed services: a distributed time service*. PhD thesis, Stanford University, 1983.

[34] MILLS, D. Network Time Protocol (Version 3): Specification, Implementation and Analysis. https://www.ietf.org/rfc/rfc1305.txt.

[35] MILLS, D. DCNet Internet Clock Service. https://tools.ietf.org/html/rfc778, April 1981.

[36] MILLS, D. Internet Delay Experiments. https://tools.ietf.org/html/rfc889, December 1983.

[37] MILLS, D. Network Time Protocol (NTP). https://tools.ietf.org/html/rfc958, September 1985.

[38] MILLS, D. Measured Performance of the Network Time Protocol in the Internet System. https://tools.ietf.org/html/rfc1128, October 1989.

[39] MILLS, D. Simple Network Time Protocol (SNTP). https://tools.ietf.org/html/rfc1769, March 1995.

[40] MILLS, D., MARTIN, J., BURBANK, J., AND KASCH, W. Network Time Protocol Version 4: Protocol and Algorithms Specification. https://tools.ietf.org/html/rfc5905, June 2010.

[41] MOON, S. B., SKELLY, P., AND TOWSLEY, D. Estimation and Removal of Clock Skew from Network Delay Measurements. In *IEEE Infocom* (1999).

[42] MURDOCH, S. J. Hot or Not: Revealing Hidden Services by their Clock Skew. In *ACM CCS* (2006).

[43] PAEK, J., KIM, J., AND GOVINDAN, R. Energy-efficient Rate-adaptive GPS-based Positioning for Smartphones. In *Proceedings of the MobiSys* (2010).

[44] PAXSON, V. *Measurements and Analysis of End-to-end Internet Dynamics*. PhD thesis, University of California, Berkeley, 1997.

[45] PAXSON, V. On Calibrating Measurements of Packet Transit Times. In *ACM SIGMETRICS* (1998).

[46] RIDOUX, J., AND VEITCH, D. Principles of Robust Timing over the Internet. *Queue* (2010).

[47] RIDOUX, J., VEITCH, D., AND BROOMHEAD, T. The Case for Feed-forward Clock Synchronization. *IEEE/ACM TON* (2012).

[48] SCHIVELBUSCH, W. *The Railway Journey: The Industrialization of Time and Space in the Nineteenth Century*. 2014.

[49] SHAKKOTTAI, S., RAPPAPORT, T. S., AND KARLSSON, P. C. Cross-layer Design for Wireless Networks. *IEEE Communications Magazine* (2003).

[50] SYED, A. A., HEIDEMANN, J. S., ET AL. Time synchronization for high latency acoustic networks. In *IEEE Infocom* (2006).

[51] THIAGARAJAN, A., RAVINDRANATH, L., LACURTS, K., MADDEN, S., BALAKRISHNAN, H., TOLEDO, S., AND ERIKSSON, J. VTrack: Accurate, Energy-aware Road Traffic Delay Estimation using Mobile Phones. In *Proceedings of the SenSys* (2009).

[52] VIJAYALAYAN, K., AND VEITCH, D. Rot at the Roots? Examining Public Timing Infrastructure. In *IEEE Infocom* (2016).

[53] VUTUKURU, M., BALAKRISHNAN, H., AND JAMIESON, K. Cross-layer Wireless Bit Rate Adaptation. *ACM SIGCOMM* (2009).

[54] ZHANG, T., PATRO, A., LENG, N., AND BANERJEE, S. A Wireless Spectrum Analyzer in Your Pocket. In *ACM HotMobile* (2015).