Faster GPS via the Sparse Fourier Transform

Haitham Hassanieh Fadel Adib Dina Katabi Piotr Indyk
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{haithamh, fadel, dk, indyk}@mit.edu

ABSTRACT

GPS is one of the most widely used wireless systems. A GPS receiver has to lock on the satellite signals to calculate its position. The process of locking on the satellites is quite costly and requires hundreds of millions of hardware multiplications, leading to high power consumption. The fastest known algorithm for this problem is based on the Fourier transform and has a complexity of $O(n \log n)$, where n is the number of signal samples.

This paper presents the fastest GPS locking algorithm to date. The algorithm reduces the locking complexity to $O(n\sqrt{\log n})$. Further, if the SNR is above a threshold, the algorithm becomes linear, i.e., $O(n)$. Our algorithm builds on recent developments in the growing area of sparse recovery. It exploits the sparse nature of the synchronization problem, where only the correct alignment between the received GPS signal and the satellite code causes their cross-correlation to spike.

We further show that the theoretical gain translates into empirical gains for GPS receivers. Specifically, we built a prototype of the design using software radios and tested it on two GPS datasets collected in the US and Europe. The results show that the new algorithm reduces the median number of multiplications by 2.2× in comparison to the state of the art design, for real GPS signals.

Categories and Subject Descriptors C.2 [**Computer Systems Organization**]: Computer-Communications Networks

General Terms Algorithms, Design, Performance, Theory

Keywords GPS, Synchronization, Sparse Fourier Transform

1. INTRODUCTION

The global positioning system (GPS) is one of the most pervasive wireless technologies. It is incorporated in more than one billion smartphones world-wide [17], and embedded in a wide variety of devices, including personal navigation systems [37], sensors [5], digital cameras [26], and even under-the-skin bio-chips [13]. The key functionality of a GPS receiver is to calculate a position, called a fix. Computing a fix involves locking on the GPS satellite signals and decoding satellite orbit and time data. Most GPS receivers, however, are embedded with some other radio (e.g., WiFi, cellular, or ZigBee) and, hence, can download the content of the GPS signal from assisted GPS (A-GPS) servers instead of decoding it from the satellite signals [19].[1] With assisted GPS used widely in phones and other GPS-capable devices [9], the bulk of what a GPS receiver does is to lock on the satellite signal (i.e., synchronize with it). This allows the receiver to calculate the sub-millisecond synchronization delay necessary for computing its position [10]. The importance of locking is further emphasized by the fact that current GPS receivers are typically duty-cycled [34, 2]; hence, they need to re-synchronize with the satellite signals regularly. Synchronizing with the satellite signal, however, is a costly process that requires tens of millions to a few billion digital multiplications [36]. Many GPS-enabled devices (e.g., mobile phones, sensors, etc.) have strict power limitations and would significantly benefit from reducing the complexity of this process.

In this paper, we aim to reduce the cost of synchronizing with weak signals like GPS. At a high level, GPS synchronization works as follows: each satellite is assigned a CDMA code. For each satellite, the receiver needs to align the corresponding CDMA code with the received signal. The process is complicated because GPS signals are very weak (about 20 dB *below* the noise level [31]). To find the right alignment of each satellite, a GPS receiver conducts a search process. It computes the correlation of the CDMA code with the received signal for all possible shifts of the code with respect to the signal. The correct shift is the one that maximizes the correlation.

So, how does a GPS receiver compute all these shifted correlations? The traditional approach *convolves* the received signal with the CDMA code of each satellite in the time domain. The correct alignment corresponds to the one that maximizes this convolution. This approach has a computational complexity of $O(n^2)$, where n is the number of samples.[2] More recent GPS receivers lock on the satellite using frequency domain computation. This approach lever-

[1] The data includes almanac, ephemeris, reference time. AGPS may also provide other optional assistance data [19].

[2] The CDMA code consists of 1023 chips transmitted at 1.023 MHz. For a GPS receiver that samples at 5 MHz, the computational complexity of the shifted correlation is $(1023 \times 5/1.023)^2$, which is about 25 million multiplications of complex signal samples. The GPS receiver has to repeat this process for multiple satellites (between 4 to 12 satellites) and multiple Doppler shifts (between 21 to 41 shifts) for each satellite, which brings the number of multiplications to over a billion. Further, correlating with one block of the signal may not be sufficient. For weak signals, the receiver may need to repeat this process and sum up the output [20].

MobiCom'12, August 22–26, 2012, Istanbul, Turkey.
Copyright 2012 ACM 978-1-4503-1159-5/12/08 ...$15.00.

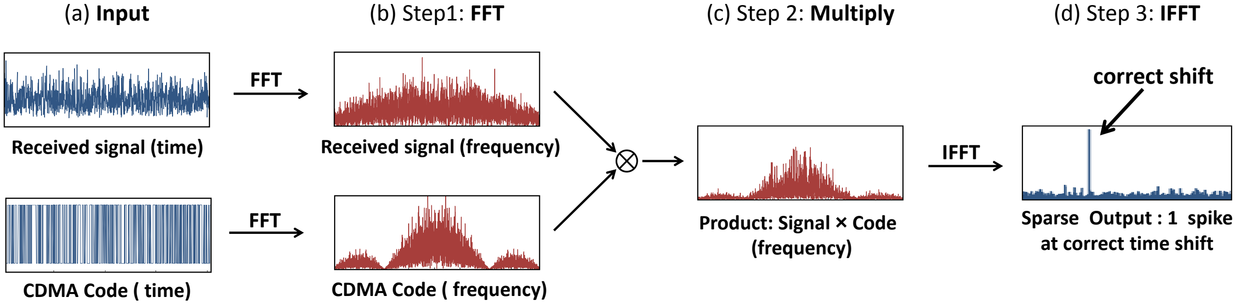

Figure 1—The steps performed by the FFT-based synchronization algorithm. The algorithm multiplies the FFTs of the received signal with the FFT of the code, and takes the IFFT of the resulting signal. The output of the IFFT spikes at the shift that correctly synchronizes the code with the satellite signal.

ages the fact that convolution in the time domain corresponds to multiplication in the frequency domain. It proceeds in the following three steps, shown in Fig. 1: 1) The receiver takes the FFT of the received signal; 2) It multiplies the output of this Fourier transform by the FFT of the CDMA code; and 3) It performs the inverse FFT on the resulting signal. This 3-step process is mathematically equivalent to convolving the signal with the code; thus, the output of the inverse FFT will spike at the correct shift that synchronizes the code with the received signal, as shown in Fig. 1d. The computational complexity of this approach is $O(n \log n)$. For the past two decades, this has been the algorithm with the lowest computational complexity for synchronizing a GPS receiver [36].

This paper introduces the lowest complexity GPS synchronization algorithm to date. Our synchronization algorithm is based on the following observations:

- First, we note that since the output of the synchronization process has a single major spike at the correct shift, as shown in Fig. 1d, the inverse FFT is very sparse. The problem of sparse FFT has recently received much attention in the computer science theory community, which resulted in new algorithms that can compute the FFT (or inverse FFT) of a sparse signal in sublinear time [14, 15, 11].[3] We build on these advances to significantly reduce the runtime of the GPS synchronization algorithm. However, existing sparse FFT algorithms use relatively complex filters (e.g., Dirichlet [11], Gaussian [14], or Dolph-Chebyshev filters [15]) to deal with the interaction of multiple potential spikes at the output of the transform. In contrast, we exploit the fact that the synchronization problem produces only one spike, and design a simple sublinear algorithm that uses only aliasing to filter the signal. This allows us to reduce the complexity of the IFFT step in Fig. 1d to sublinear time.
- Although the output of the inverse FFT is sparse and can be quickly computed, the GPS signal in the frequency domain is not sparse (Fig. 1b) and, hence, the runtime of the forward FFT cannot be reduced by applying a sparse transform. Thus, simply using sparse inverse FFT does not reduce the overall complexity of the problem (which is still $O(n \log n)$ due to the forward FFT). To address this issue, we note that the FFT in Fig. 1b is just an intermediate step that will be used as an input to the sparse IFFT. Since sparse IFFT algorithms (including ours) operate only on a subset of their input signal, we do not need to compute the values of all frequencies at the output of the forward FFT. We leverage this property to compute only a subset of the frequencies and reduce the complexity of the FFT step.

We provide an algorithm that, for any SNR, is as accurate as the original FFT-based (or convolution-based) algorithm, but reduces the computational complexity from $O(n \log n)$ to $O(n\sqrt{\log n})$. Further, when the noise in the received signal can be bounded by $O(n/\log^2 n)$, we prove that the same algorithm has a linear complexity, i.e., $O(n)$.[4]

We implement our design and test it on two datasets of GPS signals: We collected the first dataset in the US using software radios. The second dataset was collected in Europe.[5] The datasets cover both urban and suburbun areas. We compare our design against an FFT-based synchronization algorithm. Our design reduces the number of multiplications for detecting the correct shift by a median of 2.2×. Since a large fraction of GPS power is consumed by the synchronization process (30% [28] to 75% [29] depending on the required accuracy), we expect the new design to produce a significant reduction in GPS power consumption.

Contributions: This paper makes algorithmic and system contributions, which can be summarized as follows:

- The paper presents the fastest algorithm to date for synchronizing GPS receivers with satellite signals. The algorithm has multiple features: 1) It is adaptive, i.e., it can finish faster if the SNR is higher; 2) it continues to work at very low SNRs; and 3) it is general, i.e., it can be used to synchronize any signal with a random (or pseudo random) code.
- The paper provides an implementation and an empirical evaluation on real GPS signals, demonstrating that the algorithmic gains translate into a significant reduction in the number of operations performed by a GPS receiver.

2. GPS PRIMER

The key functionality of a GPS receiver is to calculate its position using the signal it receives from the GPS satellites. To do so, the receiver computes the time needed for the signal to travel from each satellite to itself. It then multiplies the computed time by the speed

[3] Sparse FFT algorithms are designed for the case where the output of the Fourier Transform contains only a small number of spikes. They are applicable to both sparse FFTs and sparse IFFTs. For a more detailed description see section 3.

[4] Note that n is not a constant and varies across GPS receivers. Specifically, different receivers sample the GPS signal at different rates, hence obtaining a different number of samples per codeword. For example, for a receiver whose sampling rate is 5MHz, n=5000, whereas for a 4MHz receiver, n=4000.

[5] The Europe dataset is courtesy of the GNSS-SDR team [8]. at the Centre Tecnologic de Telecomunicacions de Catalunya (CTTC).

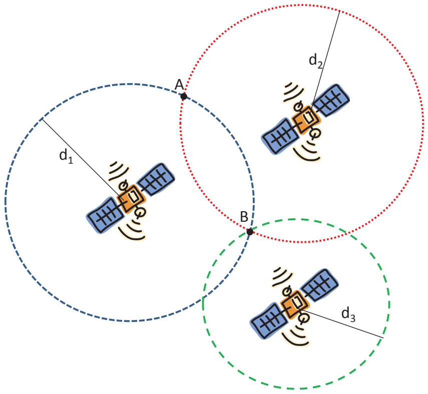

Figure 2—Trilateration: After determining the distance to different satellites, the receiver can draw spheres centered at each of the satellites and whose radii are the respective distances. These spheres should intersect at the receiver's position. A GPS receiver needs four satellites to uniquely determine its position [20]. Extra satellites can be used to correct for the lack of very tight synchronization between the receiver's clock and those of the satellites.

of light to obtain its distance from each satellite. As a result, the receiver knows that it lies on a sphere centered at that satellite and whose radius is the computed distance. It then determines its position as the intersection of several such spheres through a method called trilateration [20] shown in Figure 2.

But how does the receiver compute the propagation time from the satellites? The propagation time is obtained using a synchronization algorithm that allows the device to lock on the received signal. Specifically, each satellite has its own CDMA code, called the C/A code, which consists of 1023 chips [20]. Assuming the receiver's and satellites' clocks are perfectly synchronized, a GPS receiver generates the satellites' codes at the same time as the satellites. Due to propagation delay, however, the signal arrives in a shifted version at the receiver by exactly the amount of time it took the signal to travel from the satellite. By correlating with shifted versions of the satellite's code, the receiver calculates the propagation time as the shift at which the correlation spikes [36]. In practice, the receiver's clock is not fully synchronized with that of the satellites; this, however, can be compensated for by increasing the number of satellites used in the trilateration process.[6]

The motion of the satellites introduces a Doppler shift in the received signal. The signal does not correlate with the C/A code unless the Doppler shift is corrected. To deal with this issue, a GPS device typically performs a 2-dimensional search on the received signal [20]: one for time (code shifts), and one for Doppler shifts. Specifically, the receiver tries all possible code shifts, and 41 equally spaced Doppler shifts within +/-10 kHz of the center frequency [36], as shown in Fig. 3. Finally, the GPS satellites repeat the code 20 times for each data bit to enable the GPS receiver to decode very weak signals. The receiver tries to use one code repetition to synchronize. However, if the signal is too weak, the receiver repeats the 2D-search for multiple codes and sums the result [20].

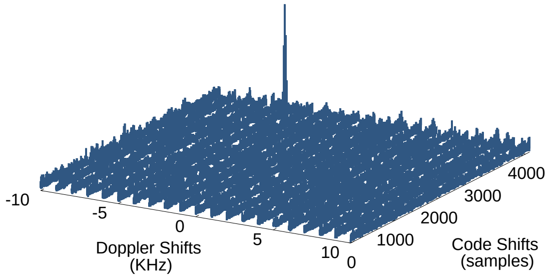

Figure 3—2D search for correlation. The plot shows the result of correlating with a C/A code for a satellite whose signal is present in the received signal. On the x-axis, we search 4000 different code shifts and on the y-axis 21 different Doppler shifts.

3. QuickSync

We describe QuickSync, a synchronization algorithm for GPS receivers. The algorithm works in the frequency domain similar to the FFT-based algorithm described in §1. QuickSync, however, exploits the sparse nature of the synchronization problem, where only the correct alignment between the received GPS signal and the satellite code causes their cross-correlation to spike. QuickSync harnesses this property to perform both the Fourier and inverse Fourier transforms in a time faster than $O(n \log n)$, therefore reducing the overall complexity of GPS synchronization.

The next subsections formalize the problem and detail the algorithm.

3.1 Problem Formulation

The synchronization problem can be formulated as follows: Given a spreading code $\mathbf{c} = c_0, \ldots, c_{n-1}$ of size n and a received signal $\mathbf{x} = x_0, \ldots, x_{n-1}$, find the time shift $\hat{t}$ that maximizes the correlation between $\mathbf{c}$ and $\mathbf{x}$, i.e., compute:

$$\hat{t} = \arg\max \mathbf{c}_{-n} \circledast \mathbf{x}, \qquad (1)$$

where $\circledast$ is a circular convolution and $\mathbf{c}_{-n}$ is the time reversed code; i.e. $\mathbf{c}_{-n} = c_{n-1}, \ldots, c_0$. Computing this convolution in the time domain requires performing n correlations each of size n and thus has complexity $O(n^2)$. However, convolution in the time domain corresponds to element-by-element multiplication in the frequency domain. Therefore computing the convolution in Eq. 1 can be done more efficiently by performing FFT on each of the code and the signal, multiplying those FFTs, then performing an inverse FFT (IFFT) as shown below:

$$\arg\max_{t} \mathbf{c}_{-n} \circledast \mathbf{x} = \arg\max_{\mathbf{t}} \mathcal{F}^{-\mathbf{1}}\{\mathcal{F}\{\mathbf{c}\}^* \cdot \mathcal{F}\{\mathbf{x}\}\}, \qquad (2)$$

where $\mathcal{F}(.)$ is the FFT, $\mathcal{F}^{-1}(.)$ is the IFFT, $*$ is the complex conjugate and t is any time sample in the output vector of the convolution. This reduces the complexity of the synchronization process to $O(n \log(n))$. Accordingly, in the remainder of this paper, we only consider the FFT-based synchronization algorithm as a baseline for evaluating QuickSync's performance.

3.2 Basics

Before introducing our synchronization algorithm, we remind the reader of a basic property of the Fourier transform, which we rely on in our design: *Aliasing a signal in the time domain is equivalent*

[6] All GPS satellites use atomic clocks and are fully synchronized with each other [20]. Hence, a GPS receiver will have the same clock skew with respect to all satellites and all the estimated propagation delays will have the same error ϵ. However, trilateration needs only 4 satellites to estimate the position and thus extra satellites can be used to to estimate and correct ϵ.

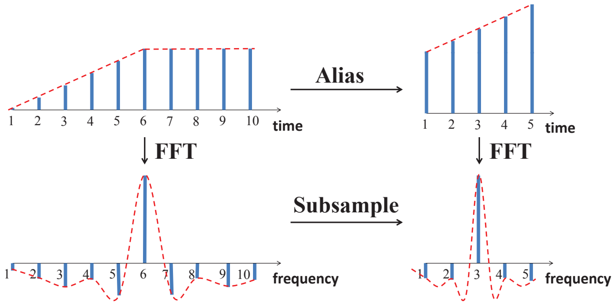

Figure 4—The duality of aliasing and subsampling. Aliasing in the time domain corresponds to subsampling in the frequency domain and vice versa. Folding (aliasing) the time domain signal in the top left results in the signal in the top right; specifically, time samples 1 and 6 add into sample 1 in the aliased signal, samples 2 and 7 into sample 2, etc. In the Fourier domain, the FFT of the aliased signal is a subsampled version of the FFT of the initial signal; namely, sample 1 in the bottom right signal corresponds to sample 2 in the bottom left, sample 2 corresponds to sample 4, etc.

to subsampling it in the frequency domain, and vice versa. Figure 4 illustrates this property.

Formally, let $\mathbf{x}$ be a discrete time signal of length n, and $\mathbf{X}$ its frequency representation. Let $\mathbf{x}'$ be a version of $\mathbf{x}$ in which adjacent windows of size B (where B divides n) are aliased on top of each other (i.e., samples that are $p = n/B$ apart are summed together). Then, for $t = 0 \ldots B-1$:

$$\mathbf{x}'_t = \sum_{j=0}^{n/B-1} \mathbf{x}_{t+jB}. \tag{3}$$

Thus, $\mathbf{X}'$, the FFT of $\mathbf{x}'$ is a subsampled version of $\mathbf{X}$, and for $f = 0 \ldots B-1$

$$\mathbf{X}'_f = \mathbf{X}_{pf}, \tag{4}$$

where $p = n/B$, and the subscript in $\mathbf{X}_{pf}$ refers to the sample whose index is $p \times f$.

3.3 The QuickSync Algorithm

We describe how QuickSync operates on a received GPS signal to synchronize it with an internally generated C/A code. For simplicity, we assume that the input signal neither exhibits a carrier frequency offset nor a Doppler shift; in later sections, we extend the algorithm to deal with these frequency offsets. Furthermore, in this section, we describe the algorithm in the context of synchronizing the GPS receiver with the signal of only one satellite; the algorithm can be easily adapted for synchronizing with multiple satellites.

The key insight to our algorithm is that the IFFT performed in step 3 of the FFT-based synchronization algorithm is sparse in the time domain, i.e., it has only one spike and, hence, can be performed in sub-linear time [14]. Further, a sub-linear time algorithm for computing the sparse IFFT would require a sub-linear number of samples as input; thus, there is no need to perform a full $n \log n$ FFT on the received GPS signal and obtain all of its n frequency samples. Rather, we only need to compute the frequency samples that will be used to perform the sparse inverse FFT.

Below, we explain how we exploit these ideas to reduce the complexity of both the IFFT and FFT performed to synchronize the signal with the code. We then put these components together in a complete algorithm.

(a) Sparse IFFT.

Inspired by recent work on sparse Fourier (inverse Fourier) transform [14, 11], we develop a simple algorithm to efficiently perform the IFFT step of GPS synchronization and quickly identify the spike of the correlation between the received signal and the CDMA code. To do so, our algorithm uses a sublinear number of samples of the signal.

The sparse IFFT algorithm proceeds as follows. It first subsamples the frequency domain signal of size n by a factor of p. It then computes the IFFT over these n/p frequency samples. Recall that subsampling in the frequency domain is equivalent to aliasing in the time domain. Thus, the output of our IFFT step is an aliased version of the output in the original IFFT step shown in Fig 1. Aliasing here can be viewed as a form of hashing, where the n original outputs samples, i.e. time shifts, are hashed into n/p buckets. Time shifts which are n/p apart will be summed and hashed together in the same bucket at the output of our IFFT. Since there is only one correlation spike in the output of the IFFT, the magnitude of the bucket it hashes into will be significantly larger than that of other buckets where only noise samples hash to. Hence, the algorithm chooses the bucket with the largest magnitude among the n/p buckets at the output of our IFFT.

Out of the p time shifts that aliased (or hashed) into this chosen bucket, only one is the actual correlation spike. To identify the spike among these p candidate shifts, the algorithm correlates the received signal with each of those p shifts of the CDMA code. The shift that produces the maximum correlation is the right spike.

Using aliasing as a form of hashing in our synchronization algorithm, as opposed to other forms of hashing used in sparse FFT algorithms (such as Gaussian and Dirichlet bucketization [15, 11]), has two major benefits:

- **Simplicity:** Subsampling can be implemented simply by taking a small IFFT of size n/p on a subset of the input and does not require any additional complex processing such as random hashing.
- **No Leakage:** In subsampling, each non-zero sample is hashed into exactly one bucket and does not leak power into any other bucket. In alternative forms of hashing [14, 15], each non-zero sample is hashed into a bucket where it is dominant but still leaks power into other buckets, which reduces the performance of the sparse IFFT algorithm.

(b) Subsampled FFT.

With the sparse IFFT step in place, the algorithm does not need the whole n-point FFT of the signal. Specifically, all the IFFT requires is a subsampled version of this signal. Thus, rather than taking a full n-point FFT, QuickSync aliases the received signal in the time domain before taking its FFT, as in Eq. 3 (Said differently, QuickSync sums up blocks of size n/p and then computes a smaller FFT of size n/p.) The output of this FFT, expressed in Eq. 4, is exactly the samples we need at the input of the sparse IFFT, described above.

A subsampled input to the IFFT (as described in §3.3(a)) results in an output spike of smaller magnitude relative to the noise bed. To compensate for this loss, we alias $p \times n$ samples instead of n into blocks of size n/p before performing the FFT.

(c) Full Algorithm.

The QuickSync algorithm proceeds in the following steps:

1. **Aliasing:** Alias $p \times n$ samples of the GPS signal into $B = n/p$ samples as described in Eq. 3, where $p = \sqrt{\log n}$.

2. **Subsampled FFT:** Perform an FFT of size n/p on the aliased time signal. This is effectively equivalent to performing an FFT of size pn and subsampling the output by p^2 according to Eq. 4.

3. **Multiplying with the code:** Subsample the FFT of the satellite CDMA code of length n by p, and multiply the resulting samples by the n/p samples at the output of step 2, above. Note that the algorithm can precompute the FFT of the CDMA code and store it in the frequency domain.

4. **Sparse IFFT:** Perform an IFFT on the n/p resulting samples. Since the input of this IFFT was subsampled, its output is aliased in the time domain. Specifically, each of the n/p buckets at the output of this stage is effectively the sum of p aliased time samples [7] as described in §3.3a.

5. **Find the unique solution:** Find the bucket with the maximum magnitude among the n/p buckets. Then, check the correlation of each of the p possible time shifts which are aliased into this bucket, and pick the shift that gives the maximum correlation. Checking the correlation can be done using only n/p samples as per Lemma A.7; therefore, it takes a total of $p \times n/p = n$ to perform the correlation of the p shifts and pick the one that maximizes the correlation.

(d) Runtime.

The running time of the QuickSync algorithm may be computed as follows. Step 1 performs np additions. Step 2 performs an FFT which takes $n/p \log(n/p)$. Step 3 performs n/p multiplications. Step 4 takes $n/p \log(n/p)$ to perform the IFFT, and finally Step 5 performs n operations to compute the correlations and find the solution. Thus, the complexity of QuickSync is $O(pn+(n/p)\log(n/p))$. To minimize this complexity, we set $p = \sqrt{\log n}$ which makes the overall running time of QuickSync $O(n\sqrt{\log n})$.

(e) Scaling with the SNR.

If the signal is too weak, GPS receivers repeat the synchronization algorithm on subsequent signal samples and sum up the output to average out the noise [36]. This approach allows the receiver to scale the computation with the SNR of the signal. The approach can be applied independent of the algorithm; hence, we also adopt it for QuickSync. However, QuickSync operates on blocks of size pn whereas the traditional FFT-based algorithm operates on blocks of size n. Both QuickSync and the traditional FFT-based algorithm compare the magnitude squared of the largest spike to the noise variance in the received signal. If the largest spike's squared magnitude exceeds the noise variance by a desired margin, the algorithm terminates the search and declares the time shift corresponding to the largest spike as the correct alignment. Otherwise, the algorithm repeats the same process on the subsequent signal samples, and sums the new output with the previous one. Since the spike corresponding to the correct synchronization is at the same time shift in each run, it becomes more prominent. In contrast, noise spikes are random and hence they tend to average out when combining the output of multiple runs of the synchronization algorithm.

(f) Linear Time Algorithm.

The algorithm described in §3(c) above can be made linear-time by modifying Step 1: instead of taking pn samples, we take only n samples and alias them into n/p buckets, where $p = \log n$. The rest of the steps are unmodified. This reduces the complexity of Step 1 to n, and the total complexity of the algorithm to $O(n + (n/p)\log(n/p)) = O(n)$.

This linear-time algorithm has weaker guarantees than the above super-linear algorithm and may not always work at very low SNRs, as detailed in §4. One can try this algorithm first. If a spike is detected with the required margin, the algorithm terminates. Otherwise, one can fall back to the super-linear algorithm in §3(c).

4. GUARANTEES

In this section, we analyze the performance of the baseline and QuickSync algorithms (both the linear and super-linear variants), under natural probabilistic assumptions about the input signal $\mathbf{x}$. In particular, we show that both the baseline and the super-linear QuickSync are correct under the same asymptotic assumptions about the variance of the noise in the signal $\mathbf{x}$. At the same time, the running time of our algorithm is equal to $O(pn + (n/p)\log(n/p))$, where p is the number of blocks used. This improves over the baseline algorithm which has $O(n \log n)$ runtime as long as the term pn is smaller than $(n/p)\log(n/p)$. In particular, by setting $p = \sqrt{\log n}$, we achieve the running time of $O(n\sqrt{\log n})$.

(a) Assumptions.

Recall that we use $\mathbf{c} = c_0 \dots c_{n-1}$ to denote the spreading code. We use $\mathbf{c}^{(t)}$ to denote the code $\mathbf{c}$ shifted by $t = 0 \dots n-1$, i.e., $c_i^{(t)} = c_{t+i \mod n}$. We have that $\mathbf{x} = \mathbf{c}^{(t)} + \mathbf{g}$ for some shift t, where $\mathbf{g}$ denotes the noise vector. We make the following assumptions:

1. The coordinates $g_0 \dots g_{n-1}$ of the noise vector $\mathbf{g}$ are independent and identically distributed random variables that follow a normal distribution with zero mean and variance σ. That is, we assume additive white Gaussian noise (AWGN).

2. The coordinates $c_0 \dots c_{n-1}$ of the spreading code $\mathbf{c}$ are independent and identically distributed random variables with values in $\{-1, 1\}$, such that $\Pr[c_i = 1] = \Pr[c_i = -1] = 1/2$. This assumption models the fact that the CDMA code, $\mathbf{c}$, is *pseudorandom*.

(b) Combining Multiple Runs.

As described in §3(e), both the baseline and our algorithm can sum the output of multiple runs to average out the noise and increase the probability of identifying the correct spike. The analysis of such multi-run scenario can be derived directly from a single run. Specifically, say the algorithm runs L times and sum up the outputs of these L runs. This is equivalent to reducing the noise variance to $\sigma' = \sigma/L$. Therefore, the L-run scenario can be analyzed by reducing it to the case of a single run, with variance divided by L.

(c) Guarantees.

We start by stating the sufficient condition for the baseline algorithm to work with probability approaching 1.

THEOREM 4.1. *Assume that $\sigma \leq c(n)n/\ln n$ for $c(n) = o(1)$. Then the baseline algorithm is correct with probability* $1 - o(1)$.

The above condition is tight. Specifically,

[7] Note that we only get p candidate shifts (and not p^2) because the actual code is of size n; hence, all shifts mod n are the same. Thus, although the total number of samples is np and they are aliased into n/p buckets, we only have p distinct shifts per bucket.

THEOREM 4.2. *There exists a constant $c > 0$ such that for $\sigma \geq cn/\ln n$, the baseline algorithm is incorrect with probability $1 - o(1)$.*

We then proceed with the analysis of the two variants of the QuickSync algorithm. The first statement holds for the super-linear variant, and shows that the algorithm works with probability approaching 1 under the same condition as the baseline algorithm, while being faster.

THEOREM 4.3. *Assume that $\sigma \leq c(n)n/\ln n$ for $c(n) = o(1)$, and that $p = o(n^{1/6})$. Then, the QuickSync algorithm that aliases p blocks of size n into n/p buckets is correct with probability $1 - o(1)$. The running time of the algorithm is $O(pn + (n/p)\log(n/p))$, which is $O(n\sqrt{\log n})$ for $p = \sqrt{\log n}$. Moreover, the algorithm performs only $O(n + (n/p)\log(n/p))$ multiplications, for any p.*

Finally, we analyze the linear-time variant of the QuickSync algorithm.

THEOREM 4.4. *Assume that $\sigma \leq c(n)\frac{n}{p\ln n}$ for $c(n) = o(1)$, and that $p = o(n^{1/6})$. Then, the QuickSync algorithm that aliases one block of n samples into n/p buckets is correct with probability $1 - o(1)$. The running time of the algorithm is $O(n + (n/p)\log(n/p))$, which is $O(n)$ for $p > \log n$.*

See Appendix for the proofs.

5. DOPPLER SHIFT & FREQUENCY OFFSET

GPS satellites orbit the Earth at very high speeds. Consequently, the GPS signal arrives at the receiver with a Doppler shift. This shift is modeled as a frequency offset f_d which is a function of the relative speed of the satellite (see Chapter 2 in [12] for exact calculations). Furthermore, the discrepancy between the RF oscillators of the GPS satellite and the GPS receiver induces a carrier frequency offset Δf_c. The total frequency offset $\Delta f = f_d + \Delta f_c$ typically ranges from -10 kHz to 10 kHz [36] and is modeled as a phase shift in the received samples. Formally, if $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are respectively the signal without and with a frequency offset then:

$$\tilde{x}_t = x_t e^{j2\pi\Delta ft}, \tag{5}$$

where t is time in seconds.

Like past synchronization algorithms, QuickSync must search and correct for the frequency offset in the received GPS signal in order for the correlation to spike at the correct code shift. However, since QuickSync processes $p \times n$ samples as opposed to n samples in past algorithms(see §3), it needs to deal with larger phase shifts that accumulate over pn samples. In order to overcome this limitation, QuickSync performs a finer grained frequency offset search, which introduces an overhead to the 2D search. This overhead, however, is amortized across all satellites in the GPS signal since correcting for this frequency offset is done on the received signal before it is multiplied by each satellite's C/A code. In §7.2, we show that despite this overhead, QuickSync still provides a significant reduction in the computational complexity of GPS synchronization. Furthermore, the frequency offset changes slowly (see §7.2); hence, the receiver can cache its value from recent GPS readings, and does not need to search for it for every GPS synchronization event.

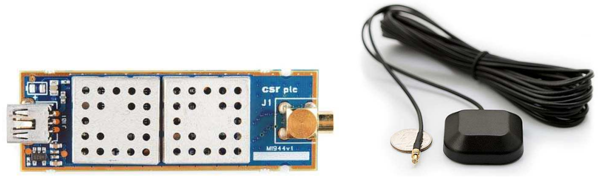
(a) Sampler (b) Antenna

Figure 5—The SciGe GN3S Sampler. The sampler is used to collect raw GPS data. It downcoverts the received signal and delivers the I and Q samples to the computer.

6. TESTING ENVIRONMENT

(a) Data Collection. We test our algorithm on a data set consisting of 40 GPS signal traces captured from urban and suburban areas in US and Europe. The traces in US are collected using the SciGe GN3S Sampler v3 [7] shown in Fig. 5(a). The GN3S is a form of software radio that collects raw complex GPS signal samples. We set the sampling rate of the GN3S to 4.092 MHz and its carrier frequency to 1575.42 MHz. The traces from Europe are collected using the USRP2 software radio [18] and the DBSRX2 daughterboard, which operates in the 1575.42 MHz range and is capable of powering up active GPS antennas [18]. The Europe traces are collected with a sampling frequency of 4 MHz. We also use a 3V magnetic mount active GPS antenna shown in Fig. 5(b). These datasets allow us to test the performance of QuickSync in different geographical areas and for different GPS receiver hardware.

(b) Baseline Algorithm. We compare our algorithm agains a baseline that uses the traditional FFT-based synchronization [39]. The baseline algorithm operates on blocks of size n. If a spike is not detected after processing the first block, the algorithm repeats the computation on the next block, i.e., the next set of n samples, and sums up the output of the IFFTs. The algorithm keeps processing more blocks until the magnitude of the peak crosses a certain threshold (as described in §7.1). Note that the algorithm must sum up the magnitudes of the output of the IFFTs rather than the actual complex values; otherwise, samples would combine incoherently due to the accumulated phase caused by the doppler shift (see §5).

(c) Implementation. We implement both QuickSync and the FFT-based algorithm in Matlab and run them on the collected GPS traces. Both algorithms use the FFTW [1] implementation internally to compute the Fourier transform (though the baseline computes an n-point transform while QuickSync computes an n/p-point transform).

(d) Metrics. We use two metrics for comparing the algorithms: number of multiplications, and number of floating point operations (FLOPs). We mainly focus on the number of real multiplications executed until an algorithm finds the synchronization offset. This metric is particularly important for hardware-based GPS synchronization, where multiplications are significantly more expensive than additions [32], and serve as standard metric to estimate the complexity of a potential hardware implementation [4, 35].

Some GPS-enabled devices do not have a full fledged GPS receiver hardware to reduce cost and form factor [23]. They use a GPS radio to collect signal samples, but offload the synchronization algorithm to the main CPU of the device, where it is done in software. To evaluate QuickSync's performance on software-based GPS receivers, we count the number of FLOPs executed by both QuickSync and the baseline. FLOPs is a standard metric

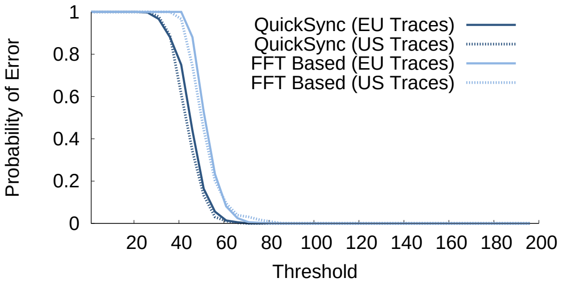

Figure 6—Probability of error versus threshold. The plot shows that the probability of error decreases sharply for both algorithms, and that a threshold of 90 for QuickSync and 100 for the baseline produce a zero error probability.

used to evaluate software implementations of algorithms, including FFTW [1]. It includes both multiplications and additions.

We count the FLOPs using OProfile, a standard profiler for Linux systems [27]. We run the code in Matlab R2011b under Ubuntu 11.10 on a 64-bit machine with Intel i7 processor. We run OProfile from within Matlab in order to profile the part of the code executed by each algorithm, and get a more accurate estimate of the number of FLOPs. We program OProfile to log the counter INST_RETIRED (the number of executed floating point operations on the Intel i7 processor [27]).

7. RESULTS

7.1 Setting the Synchronization Threshold

As explained in §3(e), both QuickSync and the FFT-based synchronization algorithm check that there is a sufficient margin between the detected maximum spike and the noise level, before accepting the spike as the one that identifies the correct alignment. Specifically, they check that the ratio of the spike's magnitude squared to the noise variance exceeds a particular threshold. This threshold defines how large the spike has to be in comparison to the bed of noise to ensure enough confidence that the spike is not due to noise and is indeed due to the code matching. Hence, the threshold is a measure of the SNR of the spike and is not dependent on the data. In particular, if the GPS data is noisy as in an urban area, the algorithm will continue processing more data until the threshold is crossed (as discussed in section §6(b)). In contrast, if the GPS data is less noisy as in an open suburban area, the algorithm will terminate early on since the spike will cross the threshold after processing one or two blocks.

In this section, we aim to verify that there is such a threshold that works for all datasets. Thus, we perform the following experiment. We vary the threshold between a value of 1 and 200, and for each of those values, we run both algorithms on a subset of the GPS traces from both datasets. We define the probability of error as the ratio of runs that output a false positive (i.e., in which the algorithm terminates by returning an invalid shift) to the total number of runs at a given threshold.

Fig. 6 plots the probability of errors versus the preset threshold. The plot shows that setting the threshold to 90 for QuickSync and 100 for the baseline produces a zero error probability. The baseline has a slightly higher error probability than QuickSync. This is because the baseline takes an n-point IFFT and, hence, has to ensure that none of the $n-1$ noise spikes exceeds the correct spike that corresponds to the proper alignment. In contrast, QuickSync takes an n/p-point IFFT and hence has fewer noise spikes that have to be kept below the threshold.

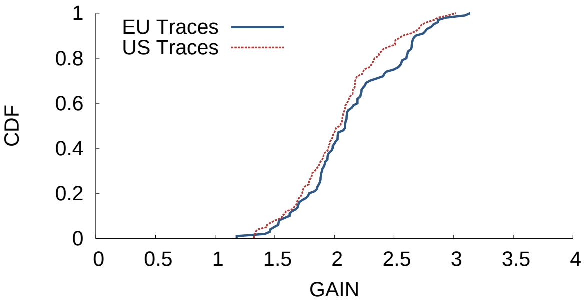

Figure 7—Gain of QuickSync over the FFT-based algorithm in number of multiplications. The two curves show the CDFs of the QuickSync's gains for the US and Europe datasets. QuickSync achieves a median gain of around 2.2× and a maximum gain of 3.3×.

The figure also shows that the used metric is stable, i.e.: (1) the metric is consistent across traces captured from two continents, and (2) the probability of error decreases monotonically as the threshold increases. This shows that the threshold is independent of the location of the GPS receiver.

In the experiments that follow, we set the thresholds to 90 and 100 for QuickSync and the baseline respectively. We also use a different set of traces from those used in testing for this threshold to ensure separation between testing and training.

7.2 Performance in Terms of Hardware Multiplications

We start by evaluating the performance gain of QuickSync over FFT-based synchronization in terms of the number of hardware multiplications. We run each of QuickSync and the FFT-based algorithm on both traces collected in US and Europe. We run the experiment 1000 times; each time taking a different subset of samples from these datasets. We compare the total number of multiplications required by each of the algorithms to synchronize with the signals of satellites present in the GPS traces. Fig. 7 shows a CDF of the gain. The gain is calculated as the number of multiplications needed by the FFT-based algorithm divided by the number of multiplications required by QuickSync. The figure shows that QuickSync always outperforms the FFT-based synchronization on both the US and EU traces with a median gain of 2.2×. This means that QuickSync can save on average twice the number of hardware multiplications.

To better understand the performance of QuickSync we zoom in on the number of multiplications required by each algorithm for each of the satellites. Specifically, each point in the CDFs in Fig. 7 corresponds to a full GPS reading with all satellites. However, because different satellites have different Doppler shifts and signal strengths, we expect the gains to vary from one satellite to another. Specifically, for each of the satellites detected in the Europe traces, and for each GPS reading, we measure the number of multiplications required by both algorithms to perform the synchronization. We repeat this experiment 1000 times on different subset of the samples and plot the average results in Fig. 8.

Fig. 8 shows that each of the satellites, on average, requires less multiplications using QuickSync. However, the gains vary considerably among those satellites. For example, satellites 5 and 18 have an average gain of 4× whereas satellites 14 and 15 have an average gain of only 1.5×. Examining the Doppler shifts of these satel-

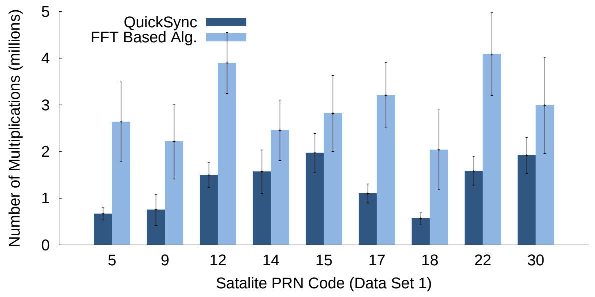

Figure 8—Number of multiplications on a per satellite basis for the Europe trace. The gain of QuickSync over the FFT-based algorithm varies among different satellites and ranges between and 1.5× and 4×.

Satellite Code:	9	12	14	15	18	21	22	25	27
Mean (Hz):	75	100	150	175	75	75	175	25	125
Max (Hz):	300	200	300	300	300	200	300	100	300

Table 1—Variation in the Doppler Shift in the US traces. For a given satellite, the Doppler shift of the received signal varies very little over a period of 2 hours and in an area of 2-mile diameter.

lites we find that satellites 5 and 18 have Doppler shifts of 6000 Hz and 1000 Hz respectively while satellites 14 and 15 have Doppler shifts of 600 Hz and 6800 Hz. This shows that the latter require a finer grain Doppler search as explained in §5. However, because QuickSync is opportunistic, it first attempts to search at courser grain shifts (the same as the baseline), but falls back to finer resolutions when it fails to detect a peak that passes the threshold. Even in such scenarios, however, it consistently outperforms the baseline as the figure shows.

In many scenarios, the receiver knows the Doppler shift a priori. The reason for this is that the Doppler shift varies only slightly between nearby locations and over a time period of about an hour. In order to test how much the Doppler shift varies, we measure the Doppler shift of satellite signals in the GPS traces captured at different locations within a 2-mile diameter geographical area over a period of 2 hours. For each of those satellites, we calculate the mean and the maximum variation in the Doppler shift of all those signals and record them in Table 7.2. The mean change is around 100 Hz and the maximum is 300 Hz. Accordingly, since GPS receivers are duty cycled, whenever the receiver wakes up, it may use the Doppler shift it calculated before going to sleep rather than performing an exhaustive search for it. Alternatively, assisted GPS receivers may download the measured Doppler shift from an adjacent base station [6]. In both of these situations, the GPS receiver can significantly reduce the overhead of searching for the right Doppler shift.

In order to measure the gains of QuickSync without the Doppler search, we repeat the first experiment but this time by providing each of the synchronization algorithms with the correct Doppler shift for each satellite. Fig. 9 shows a CDF of QuickSync's gain over the FFT-based algorithm in terms of number of multiplications over all the runs on both traces. For both traces, QuickSync achieves a median gain of 4.8×. This shows that QuickSync's gains increase when the receiver caches the correct Doppler shift across readings. We note that some of the traces benefit from QuickSync much more than others; the reason is that these runs have higher SNRs such that QuickSync can synchronize to their signals using the linear-time algorithm without falling back to the super-linear variant.

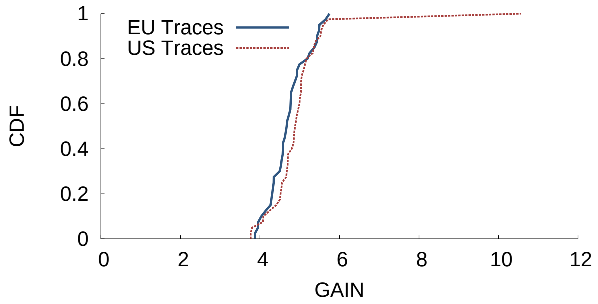

Figure 9—Gain of QuickSync over the FFT-based algorithm when the Doppler shift is known. The two curves show the CDFs of the gain in number of multiplications for both of our GPS traces. QuickSync achieves a median gain of 4.8×.

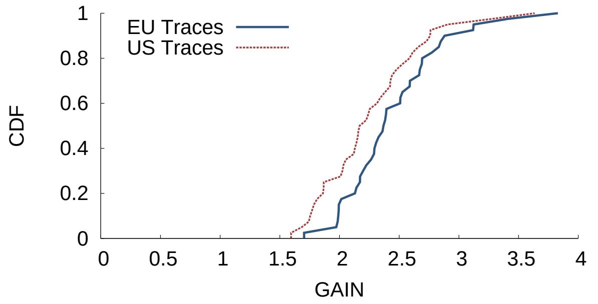

Figure 10—Gain of QuickSync over the FFT-based algorithm in FLOPs. This metric illustrates the gains of QuickSync for a software based implementation. The CDFs show a median gain about 2.2× and a maximum gain of around 3.7×.

7.3 Performance on software based GPS receivers

In this section, we test the performance of QuickSync on software based GPS receivers in terms of the number of floating point operations (FLOPs). We run QuickSync and the FFT-based algorithm on the US and Europe traces and use OProfile to count the number of FLOPs as described in §6. We run the experiment 1000 times with a different subset samples of the traces and calculate the gain as the ratio of the number of FLOPs required by the FFT-based algorithm to the number of FLOPs required by QuickSync. We do not assume in this experiment that the Doppler shift is known and we let both algorithms search for the right Doppler shift. Fig. 10 shows a CDF of the gains. QuickSync achieves a median gain of 2× and 2.3× over the FFT-based algorithm for the US and Europe traces respectively. This shows that QuickSync can reduce the number of CPU computation on average by half in software based GPS receivers.

8. RELATED WORK

FFT-based GPS synchronization was first proposed by Nee et al. [39] who showed that it reduces synchronization complexity from $O(n^2)$ to $O(nlog(n))$, where n is the number of samples per C/A code. QuickSync builds on this technique and recent advances in sparse FFT to further reduce the synchronization complexity.

Our approach is related to past work on GPS block-averaging [36, 24], which sums up consecutive signal blocks before performing the FFT. QuickSync however differs from that work along two axes: First, on the algorithmic front, past work performs a full size FFT of n points. One cannot simply replace this n-point

FFT with sparse FFT because, as explained in §1, the output of the FFT is not sparse. In contrast, QuickSync introduces a design that can harness sparse FFT. This enables QuickSync to operate with a smaller FFT of size n/p, which provides faster synchronization. Second, past work on block-averaging focuses on weak GPS signals and does not provide an adaptive design that works for the whole range of GPS SNRs. Applying their approach to the whole SNR range can incur unnecessary overhead. This is because they average and pre-process many blocks independent of the SNR. As a result, these schemes increase the synchronization delay for scenarios in which only one (or a few) blocks are sufficient for detecting the spike. In contrast, our algorithm adaptively processes more data when the spike is too low for detection, and hence gracefully scales with the SNR of the received GPS signal.

Signal synchronization is a general problem that applies to other wireless technologies, e.g., WiFi and cellular. However, synchronization in these technologies is simpler because the noise level in the received signals is much lower than in GPS. For example, WiFi receivers can lock on the signal simply by detecting an increase in the received power [16]. This is not possible in GPS since the signal is received at 20 dB *below* the noise floor [31]. Cellular systems also operate at much higher SNRs than GPS, which allows them to synchronize with relatively low overhead [21].

QuickSync is also related to the general class of work on reducing GPS power consumption. The most common approach uses assisted-GPS [19, 33], which involves connecting to an assisted GPS server through a WiFi or cellular network. The server provides the GPS receiver with the GPS data decoded from each satellite signal, which allows the receiver to avoid decoding the GPS signal. The device can also offload GPS computations to the server after acquiring the GPS signal [9]. The latter approach, however, reduces the complexity of the device but still consumes much power because it requires the device to transmit the undecoded noisy GPS signal to the cellular tower (thus consuming transmission power and even bandwidth [9]). Other approaches for reducing GPS power consumption leverage WiFi, sensors, or the cellular signal to localize the receiver [38, 3, 25]. These schemes typically are less accurate than GPS and are more constrained in terms of where they can be deployed. Our work contributes to this effort by tackling the complexity of the GPS receiver itself.

Finally, QuickSync builds on recent advances in sparse FFT algorithms [11, 14, 15]. These algorithms are designed for the case where the output of the Fourier Transform contains only a small number of spikes k, and use a variety of signal filters (Dirichlet, Gaussian, or Dolph-Chebyshev filters) to isolate and identify each of the spikes. The running times range from $O(\log n\sqrt{nk\log n})$ [14] to $O(k\log^4 n)$ [11] and $O(k\log n\log(n/k))$ [15]. The synchronization problem differs from the sparse Fourier problem in that only the IFFT is sparse. Hence, simply applying past sparse FFT algorithms does not reduce synchronization complexity.

9. CONCLUSION

This paper presents the fastest synchronization algorithm for GPS receivers to date. The gains of the algorithm are also empirically demonstrated for software GPS implementations as well as potential hardware architectures. Because synchronization consumes a significant amount of power, we expect the reduced complexity to decrease GPS power consumption. Further, we believe that the sparse synchronization algorithm we introduced has other applications in signal processing and pattern matching. We plan to explore those applications in future work.

Acknowledgments: We thank Jue Wang, Nate Kushman, Hariharan Rahul, Lixin Shi, Arthur Berger, Swarun Kumar, the reviewers and our shepherd Kyle Jamieson for their insightful comments. We thank Jonathan Eastep for explaining the proper profiling. We also thank Carles Fernandez and Javier Arribas from the GNSS-SNR project for providing us with the Europe dataset. This research is funded by NSF.

10. REFERENCES

[1] FFTW 3.2.3. http://www.fftw.org.

[2] B. Buchli, F. Sutton, and J. Beutel. GPS-equipped wireless sensor network node for high-accuracy positioning applications. In *EWSN 2012*, Trento, Italy.

[3] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low-cost outdoor localization for very small devices. *IEEE Personal Communications*, Oct. 2000.

[4] C. Cheng and K. Parhi. Low-cost fast VLSI algorithm for discrete fourier transform. *IEEE Transactions on Circuits and Systems*, April 2007.

[5] Dexter Industries. dGPS for LEGO MINDSTORMS NXT. http://dexterindustries.com.

[6] G. Djuknic and R. Richton. Geolocation and assisted GPS. *Computer*, 34(2):123 –125, feb 2001.

[7] S. Electronics. Sige gn3s sampler v3. http://www.sparkfun.com.

[8] C. Fernandez-Prades, J. Arribas, P. Closas, C. Aviles, and L. Esteve. GNSS-SDR: an open source tool for researchers and developers. In *ION GNSS Conference*, 2011.

[9] G. Fleishman. How the iphone knows where you are. Macworld, Aug. 2011.

[10] G. Fleishman. Inside assisted GPS: helping GPS help you. Arstechnica, Jan. 2009.

[11] A. C. Gilbert, S. Muthukrishnan, and M. Strauss. Improved time bounds for near-optimal sparse fourier representations. In *SPIE Wavelets XI*, 2003.

[12] A. Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[13] S. Gossett. GPS implant makes debut. WND, May 2003.

[14] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Nearly optimal sparse fourier transform. In *STOC 2012*.

[15] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Simple and practical algorithm for sparse fourier transform. In *SODA'12*.

[16] J. Heiskala and J. Terry. *OFDM Wireless LANs: A Theoretical and Practical Guide*. Sams Publishing, 2001.

[17] A. Hepburn. Infographic: Mobile stats & facts 2011. Digital Buzz, April 2011.

[18] E. Inc. Universal software radio peripheral. http://ettus.com.

[19] J. L. Jani Jarvinen, Javier DeSalas. Assisted GPS: A low-infrastructure approach. GPSWorld, March 2002.

[20] E. D. Kaplan. *Understanding GPS Principles and Applications*. Artech House Publishers, Feb. 1996.

[21] M. Karim and M. Sarraf. *W-CDMA and CDMA2000 for 3G mobile networks*. McGraw-Hill, 2002.

[22] J. Matousek. On variants of the johnson-lindenstrauss lemma. *Random Structures & Algorithms*, 33(2), 2008.

[23] Maxim IC. MAX2745 single-chip global positioning system front-end downconverter. http://www.maxim-ic.com.

[24] R. J. L. Mohamed Sahmoudi, Moeness G. Amin. Acquisition of weak gnss signals using a new block averaging pre-processing. In *IEEE/ION Position, Location and Navigation Symposium 2008*.

[25] D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *IEEE GLOBECOM*, 2001.
[26] Nikon USA. Coolpix p6000. http://www.nikonusa.com.
[27] OProfile. Linux profiler. http://oprofile.sourceforge.net.
[28] OriginGPS. ORG447X series datasheet. http://www.acaltechnology.com.
[29] Perthold Engineering LLC. SkyTraq Venus 6 GPS Module. http://www.perthold.de.
[30] G. Pisier. *The Volume of Convex Bodies and Banach Space Geometry*. Cambridge University Press, May 1999.
[31] D. Plausinaitis. GPS receiver technology mm8. Danish GPS Center, http://kom.aau.dk.
[32] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital integrated circuits*. Prentice-Hall, 1996.
[33] H. S. Ramos, T. Zhang, J. Liu, N. B. Priyantha, and A. Kansal. Leap: a low energy assisted GPS for trajectory-based services. In *ACM Ubicomp 2011*, pages 335–344, Beijing, China.
[34] D. Raskovic and D. Giessel. Battery-Aware embedded GPS receiver node. In *IEEE MobiQuitous*, 2007.
[35] T. Tan, G. Bi, Y. Zeng, and H. Tan. Dct hardware structure for sequentially presented data. *Signal Processing*, 81(11), 2001.
[36] A. C. Team. *Fundamentals of Global Positioning System Receivers: A Software Approach*. Wiley-Interscience, 2000.
[37] Ted Schadler. GPS: Personal Navigation Device. Texas Instruments. http://www.ti.com.
[38] A. Thiagarajan, L. Ravindranath, H. Balakrishnan, S. Madden, and L. Girod. Accurate, low-energy trajectory mapping for mobile devices. In *NSDI 2011*.
[39] D. Van Nee and A. Coenen. New fast GPS code-acquisition technique using FFT. *Electronics Letters*, 27(2), Jan 1991.

APPENDIX

A. PROOFS

A.1 Analysis of the baseline algorithm

The baseline algorithm computes $a_k = \mathbf{c}^{(k)} \cdot \mathbf{x}$ for all shifts $k = 0 \ldots n-1$. The probability that the algorithm reports an incorrect output is equal to

$$P(\sigma) = \Pr[a_t \le \max_{k \neq t} a_k]$$

To estimate the probability of success as a function of σ, we derive the distribution of the coordinates a_k. From our assumptions we have that $a_k = \mathbf{c}^{(k)} \cdot (\mathbf{c}^{(t)} + \mathbf{g}) = v_k + u_k$, where $v_k = \mathbf{c}^{(k)} \cdot \mathbf{c}^{(t)}$ and $u_k = \mathbf{c}^{(k)} \cdot \mathbf{g}$. Note that u_k has normal distribution with zero mean and variance

$$\mathrm{Var}[u_k] = \mathrm{Var}[\sum_{i=0}^{n-1} c_i^{(k)} g_i] = n\mathrm{Var}[c_1^{(k)}]\mathrm{Var}[g_1] = n\sigma$$

Regarding v_k, we have the following two cases:

- If $k = t$, i.e., for the correct value of the shift, we have $v_t = \mathbf{c}^{(t)} \cdot \mathbf{c}^{(t)} = n$.
- If $k \neq t$, i.e., for an incorrect value of the shift the expectation of v_k is 0.

We need to bound $P(\sigma) = \Pr[n + u_t \le \max_{k \neq t} v_k + u_k]$. The following theorem establishes the sufficient (and as we show later, necessary) condition for the baseline algorithm to be correct with probability $1 - o(1)$, i.e., $P(\sigma) \to 0$ as $n \to \infty$.

LEMMA A.1. *Assume that $\sigma \le c(n)n/\ln n$ for $c(n) = o(1)$. Then $P(\sigma) = o(1)$.*

PROOF: We will bound the probabilities of the following events: E_1: $\exists_{k \neq t} u_k \ge n/3$; E_2: $u_t \le -n/3$; E_3: $\exists_{k \neq t} v_k \ge n/3$. If none of the events hold then the algorithm output is correct. We will show that $\Pr[E_1] + \Pr[E_2] + \Pr[E_3] = o(1)$.

To analyze E_1 and E_2 recall the following fact:

FACT A.2. *Let $\Phi(s)$ be the c.d.f. of the normal distribution with zero mean and unit variance. Then for $s > 0$*

$$1 - \Phi(s) \le e^{-s^2/2}/s$$

We can now bound

$$\Pr[E_1] \le n \Pr[u_k \ge n/3] = n(1 - \Phi(\frac{n/3}{\sqrt{\mathrm{Var}[u_k]}}))$$

where k is any index distinct from t. Since $\mathrm{Var}[u_k] = n\sigma \le c(n)n^2/\ln n$, we have

$$\Pr[E_1] \le n(1 - \Phi(\sqrt{\ln n/(9c(n))})) \le e^{\ln n} \cdot e^{-\frac{\ln(n)}{18c(n)}} = o(1)$$

The probability $\Pr[E_2]$ can be bounded in the same way.

To bound $\Pr[E_3]$, assume without loss of generality that $t = 0$. In this case

$$\begin{aligned} v_k &= \mathbf{c}^{(k)} \cdot \mathbf{c} \\ &= (c_k c_0 + c_{k+1} c_1 + \ldots + c_{n-1} c_{n-k-1}) \\ &+ (c_0 c_{n-k} + \ldots + c_{k-1} c_{n-1}) \\ &= S_k + S_k' \end{aligned}$$

The terms in the sum $S_k + S_k'$ are in general *not* independent. In particular, if $k = n/2$, then $S_k = S_k'$. However, we observe the following.

CLAIM A.3. *Each of S_k and S_k' is a sum of independent random variables taking values in $\{-1, 1\}$ with probability $1/2$.*

The claim enables us to bound each sum separately. We will bound $\Pr[S_k' \ge n/6]$ first. If $k < n/6$ then the probability is zero. Otherwise, by applying the Chernoff bound we have

$$\Pr[S_k' \ge n/6] \le e^{-(n/6)^2/(2k)} \le e^{-n/72}$$

The probability $\Pr[S_k \ge n/6]$ can be bounded in the same way. Hence $\Pr[E_3] \le ne^{-n/72} = o(1)$. □

Theorem 4.1 follows from Lemma A.1.

A.2 Tightness of the variance bound

In this section we show that the assumption on the noise variance used in Theorem 4.1 is asymptotically tight. Specifically, we show that if $\sigma \ge cn/\ln n$ for some large enough constant $c > 0$, then there with probability $1 - o(1)$ the output of the baseline algorithm is incorrect. This will prove Theorem 4.2.

Recalling the notation in Section A.1, we have $a_t = n + u_t$, $a_k = v_k + u_k$ for $k \neq t$. We need to show that $P(\sigma) = \Pr[a_t \le \max_{k \neq t} a_k]$ approaches 1 for c large enough. To this end, we first observe that (i) $v_k \ge -n$ holds always and (ii) $\Pr[u_t \ge n] = o(1)$ since u_t is a normal variable with variance $n\sigma = O(n^2/\ln n)$, so the desired bound holds e.g., by Chebyshev inequality. Hence it suffices to show that

$$\Pr[\sup_{k \neq t} u_k \le 3n] = o(1) \tag{6}$$

The main difficulty in proving Equation 6 is the fact that the random variables u_k are *not* independent. If they were, a simple calculation would show that the expected value of the maximum of n independent normal variables with variance $n\sigma$ is at least $\sqrt{n\sigma \ln n} = cn$, which is larger than $a_t = n$ for a large enough c. This would then ensure that the reported shift is distinct from t with constant probability. Unfortunately, the independence could be guaranteed only if the shifted codes $c^{(k)}$ were orthogonal, which is not the case.

Instead, our argument utilizes the fact that the shifted codes $\mathbf{c}^{(k)}$ are "almost" orthogonal. Specifically, let $C = \{\mathbf{c}^{(k)} : k \neq t\}$. Since (as shown in the earlier section in the context of the event E_3) the probability that for any pair $\mathbf{c} \neq \mathbf{c}' \in C$ we have $\mathbf{c} \cdot \mathbf{c}' \leq n/3$ is $o(1)$, it follows that $\|\mathbf{c} - \mathbf{c}'\|_2^2 \geq n$ for all such $\mathbf{c}, \mathbf{c}' \in C$ with probability $1 - o(1)$.

We can now use a powerful inequality due to Sudakov [30] to show that the random variables u_k are "almost" independent, and thus the expected value of the maximum is still $\sqrt{n\sigma \ln n}$. In our context, the inequality states the following.

FACT A.4. *There exists a constant $c_2 > 0$ such that if D is the Euclidean distance between the closest pair of vectors in C, then:*

$$E = E[\max_{\mathbf{c} \in C} \mathbf{c} \cdot \mathbf{g}] \geq c_2 D \sqrt{\sigma \ln n}$$

Since $D = \sqrt{n}$, we obtain that

$$E \geq c_2 \sqrt{n} \sqrt{cn/\ln n \cdot \ln n} = c_2 n$$

The lower bound on the expected value of the maximum can be then converted into an upper bound on probability that the maximum is much lower than its expectation (this follows from simple but somewhat tedious calculations). This leads to Equation 6 and completes the proof of Theorem 4.2.

A.3 Analysis of the QuickSync algorithm

In this section we show that the probability of correctness for the QuickSync algorithm that aliases into n/p buckets exhibits a similar behavior to the baseline algorithm, albeit with the bucket variance larger by a factor of $O(p)$. At the same time, the running time of our algorithm is equal to $O(pn + (n/p)\log(n/p))$. This improves over the baseline algorithm which has $O(n \log n)$ runtime as long as the term pn is smaller than $(n/p)\log(n/p)$.

Recall that the algorithm first computes the aliased spreading code $\mathbf{c}(p)$ and signal $\mathbf{x}(p)$, defined as

$$c(p)_i = \sum_{q=0}^{p-1} c_{i+qn/p} \text{ and } x(p)_i = \sum_{q=0}^{p-1} x_{i+qn/p}$$

for $i = 0 \ldots n/p - 1$. The aliased noise vector $\mathbf{g}(p)$ is defined in an analogous way.

The application of FFT, coordinate-wise multiplication and inverse FFT computes

$$a(p)_k = c(p)^{(k)} \cdot x(p)$$

for all shifts $k = 0 \ldots n/p - 1$. The algorithm then selects $a(p)_k$ with the largest value. The last step of the algorithm fails if for $t' = t \mod n/p$ we have $a(p)_{t'} \leq \max_{k \neq t'} a(p)_k$. Let $P'(\sigma)$ be the probability of this event. We will show that as long as $\sigma = o(pn/\ln n)$ we have $P'(\sigma) = o(1)$.

LEMMA A.5. *Assume that $\sigma \leq c(n)\frac{n}{p \ln n}$ for $c(n) = o(1)$, and that $p = o(n^{1/6})$. Then $P'(\sigma) = o(1)$.*

PROOF: We start by decomposing each term $a(p)_k$:

$$\begin{aligned}
& a(p)_k \\
= & \sum_{i=0}^{n/p-1} \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(k)} \right) \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(t')} + \sum_{q=0}^{p-1} g_{i+qn/p} \right) \\
= & \sum_{i=0}^{n/p-1} \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(k)} \right) \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(t')} \right) \\
& + \sum_{i=0}^{n/p-1} \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(k)} \right) \left(\sum_{q=0}^{p-1} g_{i+qn/p} \right) \\
= & v_k + u_k
\end{aligned}$$

Consider the following events: E_0: $v_{t'} \leq n - n/4$; E_1: $\exists_{k \neq t'} u_k \geq n/4$; E_2: $u_{t'} \leq -n/4$; E_3: $\exists_{k \neq t'} v_k \geq n/4$. If none of the events hold, then the algorithm output is correct. We need to show that $\Pr[E_0] + \Pr[E_1] + \Pr[E_2] + \Pr[E_3] = o(1)$.

Events E_1 and E_2.

Let $\hat{c}_i = \sum_{q=0}^{p-1} c_{i+qn/p}$, $\hat{g}_i = \sum_{q=0}^{p-1} g_{i+qn/p}$ and $m = n/p$. Observe that $|\hat{c}| \leq p$ and $\hat{g}_i$'s are i.i.d. random variables chosen from the normal distribution with mean zero and variance $p\sigma$. Conditioned on the choice of $\hat{c}_i$s, the random variable $u_k = \sum_{i=0}^{m-1} \hat{c}_{i+k}\hat{g}_i$ has normal distribution with variance $\mu p \sigma$, where $\mu = \sum_{i=0}^{m-1} (\hat{c}_{i+k})^2$. We first show that $\mu \leq 4pm$ with very high probability, and then bound the tail of a normal random variable with variance $4pm$.

The following fact is adapted from [22], Theorem 3.1.

FACT A.6. *Let R_{ij}, $i = 0 \ldots m-1$ and $j = 0 \ldots p-1$, be i.i.d. random variable taking values uniformly at random from $\{-1, 1\}$. Let $T_i = 1/\sqrt{m} \sum_{j=0}^{p-1} R_{ij}$. There is an absolute constant C such that*

$$\Pr[\sum_{i=0}^{m-1} T_i^2 \leq 4p] \geq 1 - 2e^{-m/C}$$

Applying the fact to $R_{ij} = c_{i+jn/p}^{(k)}$, we conclude that with probability $1 - o(1)$ we have $\mu \leq m \sum_{i=0}^{m-1} T_i^2 \leq 4pm = 4n$. In that case $\mathrm{Var}[u_k] \leq 4np\sigma \leq 4npc(n)\frac{n}{p \ln n} = 4n^2 c(n)/\ln(n)$. We then follow the proof of Lemma A.1.

Event E_0.

Observe that $v_{t'}$ is a sum of terms $q_i = (\sum_{q=0}^{p-1} c_{i+qn/p}^{(t')})^2$, where each q_i is a square of a sum of p independent random variables taking values in $\{-1, 1\}$ with probability $1/2$. It follows that $E[q_i] = p$, and therefore $E[v_{t'}] = n/p \cdot p = n$. To bound the deviation of $v_{t'}$

from its mean n, we first compute its variance.

$$\begin{aligned}\mathrm{Var}[v_k] &= \mathrm{Var}[\sum_{i=0}^{n/p-1} q_i] \\ &\leq n/p \cdot E[q_1^2] \\ &= n/p \cdot E[(\sum_{q=0}^{p-1} c_{1+qn/p}^{(t')})^4] \\ &= n/p \cdot (\binom{4}{2} \sum_{q \neq q'} E[(c_{1+qn/p}^{(t')})^2 (c_{1+q'n/p}^{(t')})^2] \\ &\quad + \sum_q E[(c_{1+qn/p}^{(t')})^4]) \\ &= n/p \cdot (6p(p-1)+p) \ \leq 7pn\end{aligned}$$

We can now use Chebyshev's inequality:

$$\Pr[|v_{t'} - n| \geq n/4] \leq \mathrm{Var}[v_{t'}]/(n/4)^2 \leq 7pn/(n/4)^2 = o(1)$$

Event E_3.

It suffices to bound $\Pr[E_3]$. To this end we bound $\Pr[v_k \geq n/4]$. Without loss of generality we can assume $t = 0$. Then $v_k = \sum_{i=0}^{n/p-1} \hat{c}_i \hat{c}_{i+k}$, where $i+k$ is taken modulo n/p.

We first observe that in each term $\hat{c}_i \hat{c}_{i+k}$, the random variables $\hat{c}_i$ and $\hat{c}_{i+k}$ are independent (since $k \neq 0$). This implies $E[\hat{c}_i \hat{c}_{i+k}] = E[\hat{c}_i] E[\hat{c}_{i+k}] = 0$, and therefore $E[v_k] = 0$.

To bound $\Pr[v_k \geq n/4]$, we compute the fourth moment of $v_{t'}$ (using the second moment does not give strong enough probability bound). We have

$$\begin{aligned}E[v_k^4] &= E[(\sum_{i=0}^{n/p-1} \hat{c}_i \hat{c}_{i+k})^4] \\ &= \sum_{i_1,i_2,i_3,i_4=0}^{n/p-1} E[\hat{c}_{i_1} \hat{c}_{i_1+k} \cdot \hat{c}_{i_2} \hat{c}_{i_2+k} \cdot \hat{c}_{i_3} \hat{c}_{i_3+k} \cdot \hat{c}_{i_4} \hat{c}_{i_1+k}]\end{aligned}$$

Observe that the expectation of any term in the above sum that contains an odd power of $\hat{c}_i$ is zero. Hence the only remaining terms have the form $E[\hat{c}_{j_1}^2 \hat{c}_{j_2}^2 \hat{c}_{j_3}^2 \hat{c}_{j_4}^2]$, where $j_1 \ldots j_4$ are not necessarily distinct. Let I be a set of such four-tuples (j_1, j_2, j_3, j_4). We observe that for (j_1, j_2, j_3, j_4) to belong in I, at least two disjoint pairs of indices in the sequence $i_1, i_1+k, i_2, i_2+k, i_3, i_3+k, i_4, i_4+k$ must be equal. This means that $|I| = C(n/p)^2$ for some constant C. Since $|\hat{c}_i| \leq p$, we have $E[v_k^4] \leq C(n/p)^2 p^8 \leq Cn^2p^6$. Thus

$$\Pr[v_k \geq n/4] \leq E[v_k^4]/(n/4)^4 \leq \frac{Cn^2p^6}{(n/4)^4} = C \cdot 4^4 \cdot p^6/n^2$$

This implies $\Pr[E_3] \leq n \Pr[v_k \geq n/4] \leq C \cdot 4^4 \cdot p^6/n$ which is $o(1)$ if $p = o(n^{1/6})$. □

We now show that if the noise variance σ is "small" then one can check each shift using few time domain samples.

LEMMA A.7. *Assume that* $\sigma \leq c(n)\frac{n}{p \ln n}$. *Consider an algorithm that, given a set* K *of* p *shifts, computes*

$$a'_k = \mathbf{c}^{(k)}[0 \ldots T-1] \cdot \mathbf{x}[0 \ldots T-1]$$

for $T = n/p$, *and selects the largest* a_k *over* $k \in K$. *Then*

$$P''(\sigma) = \Pr[a_t \leq \max_{k \neq t} a_k] = o(1)$$

PROOF: The argument is similar to the proof of Lemma A.1. We verify it for an analog of the event E_1; the proofs for E_2 and E_3 are straightforward syntactic modifications of the original arguments.

First, observe that $\mathbf{c}^{(t)}[0 \ldots T-1] \cdot \mathbf{c}^{(t)}[0 \ldots T-1] = T$. Let $u'_k = \mathbf{c}^{(k)}[0 \ldots T-1] \cdot \mathbf{g}[0 \ldots T-1]$. Consider the event $E'_1 : \exists_{k \in K} u'_k \geq T/3$. We can bound

$$\Pr[E'_1] \leq p \Pr[u'_k \geq T/3] = n(1 - \Phi(\frac{T/3}{\sqrt{\mathrm{Var}[u'_k]}}))$$

Since $\mathrm{Var}[u'_k] = T\sigma \leq c(n)\frac{n^2}{p^2 \ln n}$, we have

$$\Pr[E'_1] \leq p(1 - \Phi(\sqrt{\ln n/(9c(n))})) \leq e^{\ln n} e^{-\frac{\ln(n)}{18c(n)}} = o(1) \quad \square$$

Proofs of Theorem 4.3 and Theorem 4.4.

To prove Theorem 4.3, recall that given a signal $\mathbf{x}$ consisting of p blocks, each of length n and with noise variance σ, QuickSync starts by aliasing the p blocks into one. This creates one block of length n, with noise variance σ/p (after normalization). We then apply Lemmas A.5 and A.7.

Theorem 4.4 follows from the assumption that $\sigma = c(n)\frac{n}{p \ln n}$ and Lemma A.5.