Time Synchronization for High Performance Internet of Things Applications

Jessica Ko and Vasuki Narasimha Swamy

The "Tactile Internet" promises to introduce several new emerging technology market opportunities like immersive and interactive applications such as robotics, gaming, smart-health-care, and smart grid. To enable these high-performance IoT applications, ultra-reliable communication with latencies of about 1ms is crucial. This domain is largely unexplored and the techniques used by existing standards are fundamentally ill suited for low-latency and high-reliability.

An existing domain which parallels these requirements is high performance industrial control systems which are supported by wired communication protocol. They support short messages (10s of bytes) to/from closeby sensors/actuators (10s -100s) delivered regularly (100s - 1000s of times per second). The protocols are ultra-reliable (probability of a single packet delivery failure of 10^{-8}) with very low latency (a couple of ms). Synchronized cooperative communication based wireless protocols proposed in [1], [2] achieves QoS (high-reliability and low latency) similar to wired fieldbus systems by exploiting multi-user diversity and distributed space-time coding to achieve. One of the main requirements for protocols aiming at high-performance IoT applications is tightly synchronized clocks to avoid collision and thus avoiding any decoding errors which lead to violation of latency requirements.

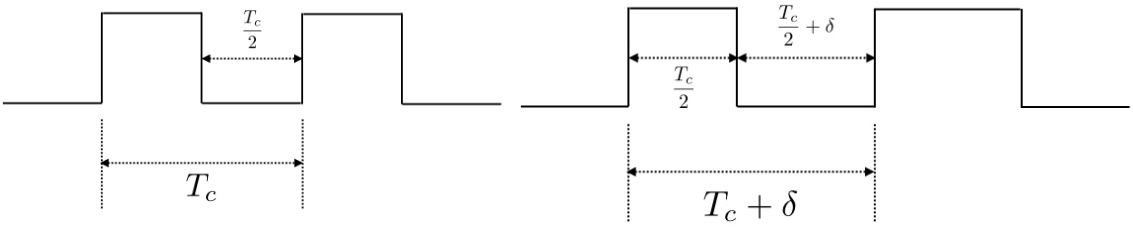
To understand the kind of tight synchronization required we do the following back of the envelope calculations. Consider the "Occupy CoW" protocol described in [1]. Each downlink and uplink messages are flooded into the network and all the nodes which can aid the origin node in getting the packet through to the destination help at the same time by the means of a distributed space time code (DSTC) - either a deterministic one like Alamouti [3], or a random one like a random linear combination of possible codewords [4] or something even more simpler like delay diversity [5]. The main objective of any DSTC is to enable full diversity. Thus each message gets two chances to reach the destination. If there are 25 nodes in the network and the total time available is 2ms, then the uplink packets get approximately 20μs. To ensure that uplink messages are decodable and everything happens in a timely fashion, the clocks need to be synchronized upto to a precision of 100ns. The reason we can tolerate errors up to 100ns is because an OFDM-like technique is employed can compensate for such offsets.

We studied some of the commonly used techniques for clock synchronization. GPS clocks have great accuracy — about $40ns$. But there are two main reasons for why GPS clocks are not useful for our applications: (a) they work only outdoors (b) they have a very high convergence time. The Internet generally synchronizes in a hierarchical fashion based on the seminal work of Mills "Network Time Protocol"(NTP) [6]. The offsets achieved by this protocol is in the order of hundreds of milliseconds and is clearly not good enough for us. Reference broadcast based protocols [7], [8] are a significant improvement over NTP as the offsets are in the order of microseconds and also deal well with skews (difference of clock frequencies). Thus none of these already established methods were good enough for our application which requires accuracy in the order of $100ns$. So we explored some simple methods inspired by these existing application. Section I presents different clock models, section II describes the

problem in detail, section III describes the methods explored for synchronization and IV present the results.

I. How do clocks work?

In order to develop a synchronization protocol, we need to first understand the behavior of typical clocks. Clocks are made up of oscillators and a counter. Every time the oscillator finishes one oscillation it increments (or decrements) the counter. This is the smallest time that can be measured by the clock. For example if the oscillator was ticking at 10 GHz, then the smallest time kept would be $\frac{1}{10\times10^9}$ which is 100ps. More generally clocks tick at a frequency F_c leading to a cycle time $T_c = \frac{1}{F_c}$. This is illustrated in Fig. 1a.



(a) Ideal clock (b) Real clock

Fig. 1: Ticking of different clocks

But these are ideal clocks. Most clocks are marked to tick at a frequency F_c but they do not necessarily tick at that rate. The error in the actual ticking frequency of a 'moderately good' clock is about 300 ppm (parts per million). Thus if we are looking at a clock which is marked to tick at 10GHz and if off by 300 ppm, then it would be actually ticking at 10 + $10 \times 300 \times 10^{-6}$ GHz which is 10.003 GHz. While an ideal clock would report that 1 second is 1 second, the fast clock would report that the elapsed time is 1 second and 30 microseconds instead of 1 second. The deviation from the ideal ticking rate is called the 'skew' (denoted by α) and this is the ratio between actual ticking rate and ideal ticking rate. Thus a clock $C(t)$ can be modeled as $C(t) = \alpha t + \beta$ where t is the time at the ideal clock, α is the skew and β is some initial offset.

What we described above is still a bit ideal. Why? We've completely ignored any environmental factors (like temperature, humidity, pressure) that might affect the oscillator's ticking rate [9]. These are important factors to be considered and actual oscillators behave differently. If the ideal frequency of oscillation is F_c (leading to a period of T_c) then each period of oscillation in reality is $T_c + \delta$ where δ can be modeled as a Gaussian random variable & this is called the jitter. This is illustrated in Fig. 1b. How high is the jitter going to be in one cycle? Typically, the standard deviation of δ is 50ppm of T_c. In 1ms, the number of cycles recorded by a 10GHz oscillator (T_c = 100ps is $10 \times 10^9 \times 10{-}3 = 10^7$. Thus, if the standard deviation per cycle is 50ppm of 100ps, the standard deviation time taken to register 1ms is $\left(\sqrt{10^7} \times 100 \times 50 \times 10^{-6}\right)$ps which is approximately 158ps. This essentially means that the time needed to record 1ms is a Gaussian random variable centered at 1ms with a variance os 158ps.

II. Network Setup

We consider a network with a central controller (C) that wishes to send and receive separate messages to and from each client in a set of N client, denoted by the set $\mathcal{S}$ (see Fig. 2). Distinct messages flow

in a star topology from the central controller to individual clients, and in the reverse direction from the clients to the controller within a "cycle" of length T (here $T = 2$ms). This cycle of communication must be achieved with a very small outage probability (on order of 10^{-9}).

The Occupy CoW protocol described in [1] uses multi-user diversity to overcome bad fading events. The basic idea is to use a flooding strategy where the controller broadcasts a packet that includes messages for all clients. As this is a broadcast message, the intended set of recipients are all clients. Clients with good channels act as relays for other clients. This is done in a carefully scheduled, phased manner with dedicated time slots for each client to talk.

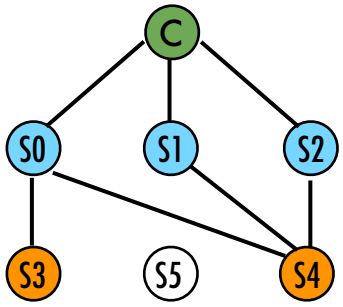


Fig. 2: Topology schematic for the protocols.

We first consider an aggressive synchronization protocol for the intial ideal clock model (where there is no jitter). As the applications we envision are mostly indoors (else we could just use GPS), most of the clients are in each other's range. We first consider a multiple beaconing reference clock model where there are several dedicated clock nodes say on the ceiling of each room which serve as reference clocks. These reference clocks are all connected by a wire so that they have the same reference time t. The clocks in the N clients are essentially marked to be ticking at the same rate as the reference clock but due to the imperfect nature of clocks, each client clock has a skew α_i. The time of client i at time t of the reference clock is $C_i(t) = \alpha_i t + \beta_i$ where α_i is the skew and β_i is the initial offset. Clients will listen to the messages of the closest reference clock. Figure 3 shows the model with reference clocks and clients where $d_i(t)$ is the distance of client i from the nearest reference clock at time t.

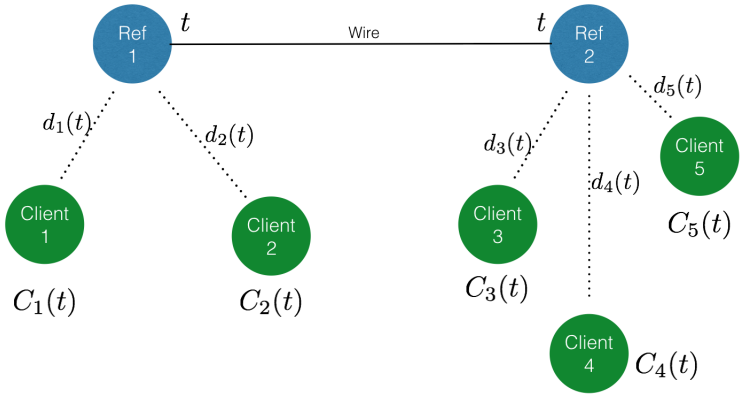


Fig. 3: Clock Model

The key feature of our synchronization protocol is that the reference clocks transmit a beacon indicating the time *every* t_0 seconds. The clients are in promiscuous listening mode so they anticipate a beacon to arrive every t_0 seconds. The t_0 selected effects how fast the method converges to an accurate α and β which we will discuss in detail in section IV. After k time intervals or at time kt_0, the reference clock will broadcast the time kt_0. The client i will receive the message not exactly at kt_0 but after some time $n_i(kt_0)$ which is the combination of speed-of-light lag (which is inevitable) and recording lag (after all

it takes time for the radio to figure out that it is receiving a packet). With a slight abuse of term we will call this 'noise'.

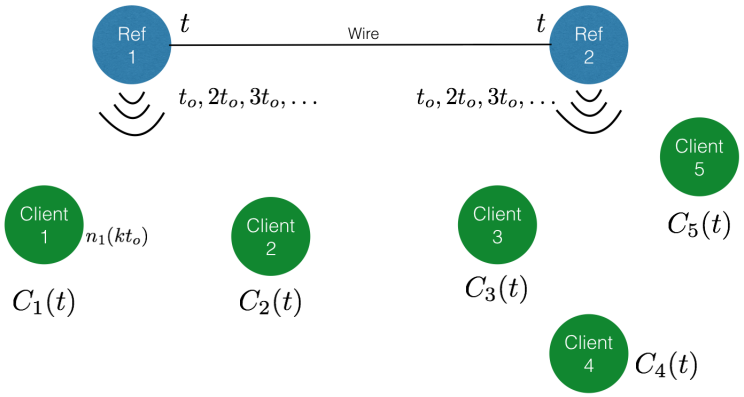


Fig. 4: Clock Model with Broadcasts and Noise

Noise is inevitable when the client receives the message from the reference clock. The noise for client i is represented by $n_i(kt_0) := n_i(k)$. The delay d_i is determined by the speed of light constraints on the distance between the client and the closest reference clock. In addition, there will be noise from the environment which is denoted by $N_i(k)$ and is chosen to be a normal distribution, $N_i(k) \sim \mathcal{N}(0, \sigma^2)$. Combining the two components, the noise is $n_i(k) = d_i + N_i(k)$. This also captures the minute movements that the clients have made in the interval of t_0. Figure 4 shows clock model with the noise and reference clock broadcasts.

III. Synchronization Methods

The client i can estimate an α_i by using the times from consecutive intervals. When the reference clock broadcasts at fixed time intervals, the clients will recover their own time and the messages received contains t_0 to be used by the client for finding the skew. To simplify the notation, consider $C_i(k) = C_i^{(rec)}(kt_0)$, which is the time the client records that it received the synchronization message. When there is no noise or $n_i(k) = 0$, the skew value can be obtained correctly from consecutive synchronization messages using the following equations.

$$\begin{aligned} C_i(k+1) - C_i(k) &= \alpha_i(k+1)t_0 - \alpha_i(k)t_0 \\ &= \alpha_i \times t_0 \end{aligned}$$

Therefore we get,

$$\hat{\alpha_i} = \frac{C_i(k+1) - C_i(k)}{t_0}$$

However, noise is always present, so $\hat{\alpha_i} \neq \alpha_i$. A cleverer method to minimize the effect noise has on the skew is to consider the first synchronization packet arrival time (or an appropriate older packet) instead of the consecutive times. The reason we mentioned an appropriately older packet is because nodes are constantly moving and the d_i would vary widely if we consider the *very* first packet. So an older packet

such that the movement is not too pronounced is good enough. The revised expression for the estimate of α is

$$\hat{\alpha_i} = \frac{C_i(k) - C_i(1)}{(k-1)t_0} = \frac{\alpha_i((k-1)t_0 + n_i(k) - n_i(1))}{(k-1)t_0}$$

The estimated skew can be used to find the estimate of β along with speed of light lag (combined), denoted by $\hat{\beta}_c$. The reason we cannot distinguish between β and the combined value β_c as the rank of the matrix we are dealing with is 2 and the number of variables is 3 [10]–[13]. With these estimated values, the client synchronizes it's time as described by the following equation.

$$\text{virtual clock}_i = \frac{C_i(k) - \hat{\beta}_c}{\hat{\alpha}}.$$

We also modeled the noise as an exponential random variable which would offset better for changing distance. We proved bounds for when the noise is exponential.

A. *Find* $P(|\hat{\alpha} - \alpha_1| \leq \epsilon)$

First find $\hat{\alpha}$ an estimation of α_1.

$$c_1 = \alpha_1 t + \beta$$

If there are no errors, then

$$c_1(t_1 + jt_0) - c_1(t_1) = j\alpha_1 t_0$$
$$\hat{\alpha} = \alpha_1 = \frac{c_1(t_1 + jt_0) - c_1(t_1)}{jt_0}$$

With noise st $n_k \sim exp(\lambda)$,

$$c_1(t_1 + jt_0 + n_{j+1}) - c_1(t_1 + n_1) = j\alpha_1 t_0 + \alpha_1(n_{j+1} - n_1)$$

Using the above method of dividing by jt_0.

$$\hat{\alpha} = \frac{j\alpha_1 t_0 + \alpha_1(n_{j+1} - n_1)}{jt_0} = \alpha_1 + \frac{\alpha_1(n_{j+1} - n_1)}{jt_0}$$

Finding the probability,

$$P(|\hat{\alpha} - \alpha_1| \leq \epsilon) = P(\hat{\alpha} - \alpha_1 \leq \epsilon) - P(\hat{\alpha} - \alpha_1 \leq -\epsilon)$$

$P(\hat{\alpha} - \alpha_1 \leq \epsilon)$ = ?

$$P(\hat{\alpha} - \alpha_1 \leq \epsilon) = P\left(n_{j+1} - n_1 \leq \frac{jt_0\epsilon}{\alpha_1}\right)$$
$$P\left(n_{j+1} - n_1 \leq \frac{jt_0\epsilon}{\alpha_1}\right) = 1 - \frac{1}{2}e^{-\lambda\frac{jt_0\epsilon}{\alpha_1}} \tag{1}$$

For $P(\hat{\alpha} - \alpha_1 \leq -\epsilon)$, we get

$$P(\hat{\alpha} - \alpha_1 \leq -\epsilon) = \frac{1}{2}e^{-\lambda\frac{jt_0\epsilon}{\alpha_1}} \tag{2}$$

Putting Eq. (1) and (2) together, we get

$$\begin{aligned} P(|\hat{\alpha} - \alpha_1| \leq \epsilon) &= 1 - \frac{1}{2}e^{-\lambda\frac{jt_0\epsilon}{\alpha_1}} - \frac{1}{2}e^{-\lambda\frac{jt_0\epsilon}{\alpha_1}} \\ &= 1 - e^{-\lambda\frac{jt_0\epsilon}{\alpha_1}} \end{aligned} \tag{3}$$

B. Find $E[|\hat{\alpha} - \alpha_1|^2]$

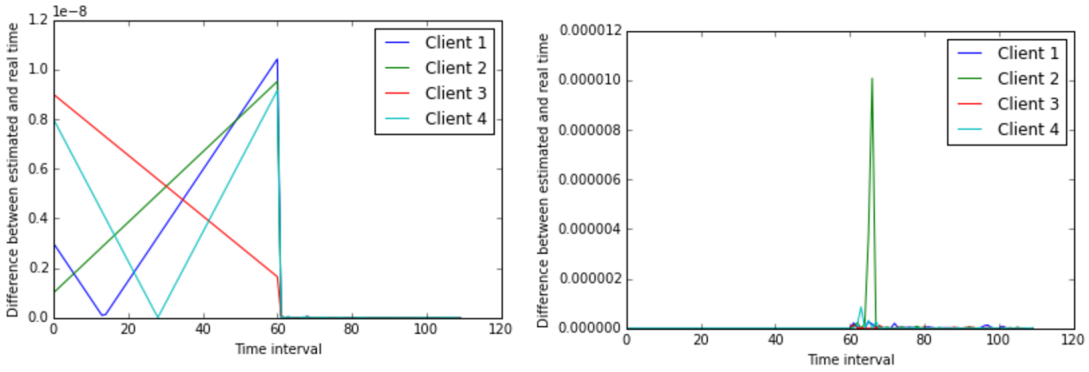
Assume α_1 to be a constant.

$$\begin{aligned}
E[|\hat{\alpha} - \alpha_1|^2] &= E[\hat{\alpha}^2] - 2E[\hat{\alpha}\alpha_1] + E[\alpha_1^2] = E[\hat{\alpha}^2] - 2\alpha_1 E[\hat{\alpha}] + \alpha_1^2 \\
&= E\left(\alpha_1 + \frac{\alpha_1(n_{j+1} - n_1)}{jt_0}\right)^2 - 2\alpha_1^2 + \alpha_1^2 \\
&= \alpha_1^2 + \alpha_1^2 E\left[\left(\frac{(n_{j+1} - n_1)}{jt_0}\right)^2\right] - 2\alpha_1^2 E\left[\frac{(n_{j+1} - n_1)}{jt_0}\right] - \alpha_1^2 \\
&= \alpha_1^2\left(E\left[\left(\frac{n_{j+1}}{jt_0}\right)^2\right] + E\left[\left(\frac{n_1}{jt_0}\right)^2\right] - 2E\left[\frac{n_{j+1}}{jt_0}\right]E\left[\frac{n_1}{jt_0}\right] - 2E\left[\frac{n_{j+1}}{jt_0}\right] + 2E\left[\frac{n_1}{jt_0}\right]\right) \\
&= \alpha_1^2\left(\frac{2}{\lambda^2} + \frac{2}{\lambda^2} - 2\frac{1}{\lambda}\frac{1}{\lambda}\right) + \alpha_1^2\left(\frac{2}{\lambda} - \frac{2}{\lambda}\right) \\
&= \frac{2\alpha_1^2}{\lambda^2}
\end{aligned} \tag{4}$$

These seem well enough for the ideal clock model with constant skew but what about the case when the clocks take a random walk along the line of constant skew? Does this method give us good enough results or do we have to do something more intelligent? Before answering that question, we explored how the above method would perform for ideal clocks.

IV. Results

We performed some initial simulations with one reference clock and four clients. Synchronization starts at the 60th time interval, so the times of the client clocks before that time are not very accurate.

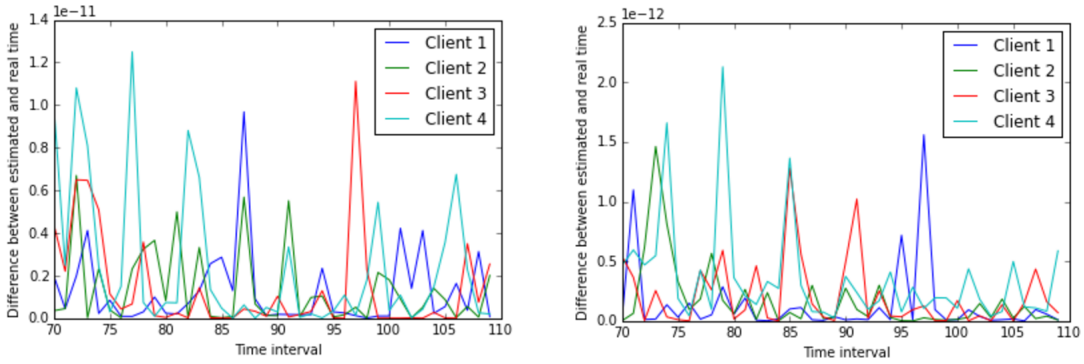


(a) Results when $t_0 = 1$ms

(b) Results when $t_0 = 1\mu s$

Fig. 5: Simulated results for different synchronization time intervals

For the clients, the skew is between -300 to 300 ppm, which is an accurate representation of most clocks. The offset is picked randomly and the distance from the reference clock is picked randomly between 2 to 10 meters (this is fine as the applications being targeted are indoors and mostly in the line-of-sight of each other). The standard deviation of the noise that we used in our simulations was 100ns. This is a good estimate of any movement or wrong estimate from mean position + radio delay.



(a) Results when $t_0 = 1$ms for time past 70ms

(b) Results when $t_0 = 10$ms for time past 70ms

Fig. 6: Closer look at the simulated results for different synchronization time intervals

The distance covered by a radio signal in 100ns is 30m and any moving radio would typically cover than distance in 10 seconds or so. Given we are considering applications in indoor environment, it is safe to assume that at most 1ns (or 30cm) error is contributed due to inaccuracy in distance measurement in every cycle lasting about 1ms. In addition to that, 100ns noise also takes into account any delay in measurement and decoding. If we had a 10Ghz oscillator then 100ns corresponds to almost 1000 symbols of error which is roughly the maximum delay in a radio circuit to declare that a packet is being transmitted.

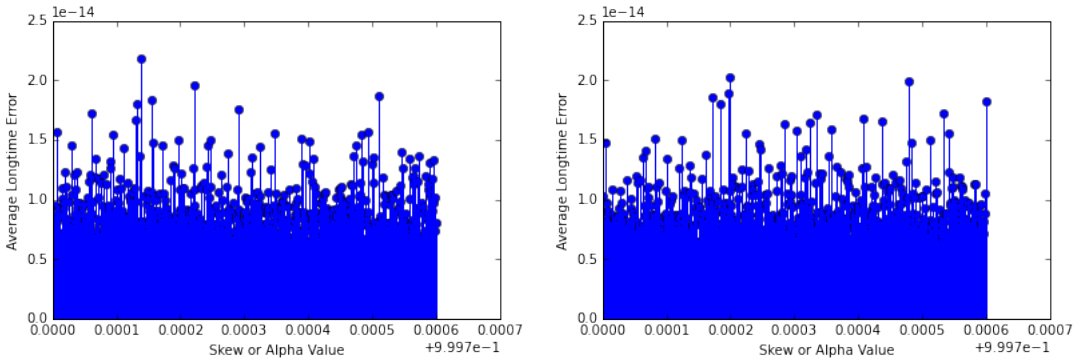
When $t_0 = 1$ms as shown in figure 5a, the method performs extremely well because the difference between the estimate time and real time gets very close to zero within the first few time intervals.

From Eq (3) it is plenty clear that considering the length of the time interval is necessary because the results are influenced by what value is picked. When a shorter time interval is used, the accuracy of the estimated time is worse as shown in figure 5b with $t_0 = 1\mu s$. This is supported by Eq. (3) which states that the probability that we get too far from the true value of skew is an increasing function of t_0 so the simulation results agree with the theory. If a longer time interval like 10ms is used, the accuracy of the estimated time improves. When looking at time intervals greater than 70, figures and easily show the difference between accuracies. However, an accurate estimated time takes longer to reach because the time intervals are now longer.

Does the 'value' of the skew affect the performance of the synchronization method? To learn that, we performed another simulated experiment where the synchronization was allowed to run for 100 seconds for all values of skews ranging from $\left(1 - 300 \times 10^{-6}, 1 + 300 \times 10^{-6}\right)$ and we calculated the long term average error in the virtual clock and reference clock time. For the ideal clock model with constant skew and synchronization interval of 1ms, the performance is depicted in figure 7a.

As we see there is no discernible pattern which could make the performance dependent on the skew. But all the above performance results were for the ideal clock model. What happens to the performance of this method in the real clock model? We performed the same simulated experiment for the real clock model and observed the long term error average between the virtual clock and reference clock which is depicted in figure 7b.

We were initially surprised with this very positive result! Why is the performance of the synchronization method good enough even for real clocks? The answer lies in the amount of noise real clocks add. In section I, we presented a back of the envelope calculation which showed that for a 10GHz oscillator with 50ppm variance, the variance of error in time to register 1ms is about 158ps. And the variance noise we



(a) Performance of synchronization method for ideal clocks with synchronization interval of 1ms

(b) Performance of synchronization method for real clocks with synchronization interval of 1ms

Fig. 7: Long term average for synchronization interval of 1ms

add for various delays is 100ns which is about 1000 times larger than the error due to skew variations. Due to the fact that variation in skew adds negligible noise compared to what is already being considered, the performance of the synchronization scheme is as good as the performance for ideal clocks.

The idealism that we had to yet account for was dropping of packets which would result in clients not receiving the synchronization message. Wireless channels constantly vary. Coherence time is the time duration over which a channel's impulse response is considered to be not varying — in other words, a good channel remains good for one coherence time period and a bad channel remains bad for one coherence time period. For indoor environments where we use say a 5GHz spectrum for transmission and the clients are moving at a relative velocity of 10m/s, the coherence time is between 15ms and 30ms (see [14] for coherence time calculation). We modeled the packet drop of a synchronization message (sent every 1ms) using the coherence time of the indoor channels as if messages are received only once in every 30ms. The performance of the synchronization method is shown in figure 8.

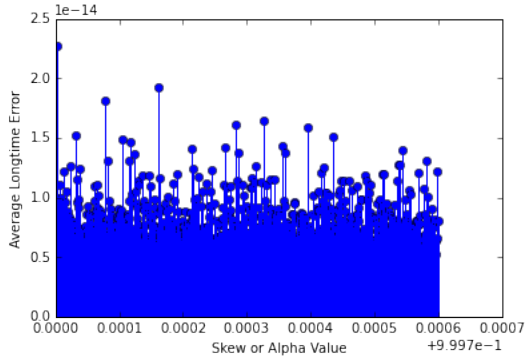


Fig. 8: Performance of synchronization method for real clocks with synchronization interval of 30ms

We see that again the performance is well within the target! Our next step was to add feedback from other

clients but given that the performance is good, we decided to skip it. But this very positive performance got us wondering why we were outperforming most wireless sensor network synchronization techniques. The answer is due to three reasons:

1) Advancements in electronics has made cheap but good clocks and as we considered 'good' clocks in our work (not atomic clocks but still good), we get good performance.
2) The nodes are most of the time in talking range with each other as the applications being considered are indoors. The biggest factor is comes from the fact that we eliminate the 'critical' path effect mentioned in [8] by actually minimizing it by considering an indoor environment.
3) A huge difference between wireless sensor networks (WSN) and the applications we consider are energy requirements. Nodes in WSN cannot afford to stay up all the time and be in promiscuous listening mode but the applications we target can (and infact have to as shown in [1]). This is a big contributor to this simple scheme having a good performance.

This concludes our theoretical analyses of the synchronization protocols for high-performance IoT applications. The implementation of this scheme will be done on software defined radios over the summer and Fall semester!

REFERENCES

[1] V. Narasimha Swamy *et al.*, "Cooperative communication for high-reliability low-latency wireless control," in *IEEE International Conference on Communications (ICC)*, 2015.

[2] ——, "Network coding for High-Reliability Low-Latency wireless control," in *IEEE Wireless Communications and Networking Conference: Workshop on 5G & Vertical Industry*, Doha, Qatar, Apr. 2016.

[3] S. M. Alamouti, "A simple transmit diversity technique for wireless communications," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 8, pp. 1451–1458, 1998.

[4] B. Sirkeci-Mergen and A. Scaglione, "Randomized space-time coding for distributed cooperative communication," *Signal Processing, IEEE Transactions on*, vol. 55, no. 10, pp. 5003–5017, 2007.

[5] M. Bossert *et al.*, "On cyclic delay diversity in ofdm based transmission schemes," in *OFDM workshop*, vol. 2, 2002.

[6] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, 1991.

[7] J. Elson *et al.*, "Global synchronization in sensornets," in *LATIN 2004: Theoretical Informatics*, 2004.

[8] ——, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, 2002.

[9] M. Frerking, *Crystal oscillator design and temperature compensation*. Springer Science & Business Media, 2012.

[10] N. M. Freris *et al.*, "Fundamentals of large sensor networks: Connectivity, capacity, clocks, and computation," *Proceedings of the IEEE*, vol. 98, no. 11, 2010.

[11] N. M. Freris and P. Kumar, "Fundamental limits on synchronization of affine clocks in networks," in *46th IEEE Conference on Decision and Control*, 2007.

[12] A. Giridhar and P. Kumar, "Distributed clock synchronization over wireless networks: Algorithms and analysis," in *2006 45th IEEE Conference on Decision and Control*, 2006.

[13] N. M. Freris *et al.*, "Fundamental limits on synchronizing clocks over networks," *Automatic Control, IEEE Transactions on*, vol. 56, no. 6, 2011.

[14] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. New York, NY, USA: Cambridge University Press, 2005.