A SOFTWARE-DEFINED RADIO IMPLEMENTATION OF TIMESTAMP-FREE NETWORK SYNCHRONIZATION

Mitchell W.S. Overdick[⋆] *Joseph E. Canfield*[⋆] *Andrew G. Klein*[⋆] *D. Richard Brown III*[†]

[⋆] Western Washington University
Department of Engineering and Design
516 High St, Bellingham, WA 98225
{overdim, canfiej2, andy.klein}@wwu.edu

[†] Worcester Polytechnic Institute
Department of Electrical and Computer Engineering
100 Institute Rd, Worcester, MA 01609
drb@wpi.edu

ABSTRACT

This paper describes a real-time implementation of timestamp-free network synchronization using RF signaling between a master and slave node. Rather than conventional approaches of exchanging digital timestamps through a dedicated synchronization protocol, timestamp-free synchronization is performed implicitly at the physical layer through timing of a master node's responses to a slave node. This approach was implemented in C++ on a pair of Ettus USRP E310 software defined radios, and extends a previous implementation at audio frequencies using acoustic hardware. Experimental results using modulated sinc pulses demonstrate synchronization accuracy less than 12% of the sampling period. The results confirm previous theoretical studies suggesting that the timestamp-free synchronization approach accurately accounts for propagation delay, frequency offset, and stochastic drift.

***Index Terms*—** synchronization, oscillator dynamics, real-time signal processing, software-defined radio (SDR), Universal Software Radio Peripheral (USRP)

1. INTRODUCTION

Synchronization is necessary in communication networks to enable coordination among nodes, scheduling of communication resources, interference avoidance, event detection/ordering, data fusion, and coordinated wake/sleep cycles. Most synchronization approaches (e.g., [1–3]) employ application-layer or MAC-layer exchanges of digital timestamps, which inherently require a significant amount of overhead to be able to resolve sufficiently small time increments. Recently, a bidirectional timestamp-free synchronization approach was proposed in [4], which conveys implicit timing information in the physical layer through the timing of the responses between nodes. This approach was shown to be attractive because the synchronization functions can be embedded in existing network traffic, the scheme accounts for propagation delay, and there is no need for exchanging digital timestamps.

This paper describes a real-time implementation and experimental results for the bidirectional timestamp-free synchronization protocol using Ettus USRP software defined radios. This work builds upon the original protocol described in [4] and illustrated in Fig. 1. Previously, this protocol was implemented using acoustic hardware on the Texas Instruments TMS320C6713 DSK platform, the results of which are described in [5]; this work expands on the prior experimental study by implementing the synchronization approach at radio frequencies using software-defined radios. Besides the obvious differences in testbed (Ettus USRP vs. TI DSK) and transmission medium (RF vs. acoustic), the use of high RF carrier frequencies required an implementation employing analog downconversion prior to sampling; thus, as opposed to the previous fully digital acoustic implementation [5], this RF implementation was required to be designed to operate on baseband as opposed to passband signals. Finally, this work demonstrates the feasibility of the timestamp-free synchronization algorithm for use in modern wireless networks.

This work was supported by a Research Experience for Undergraduates (REU) Supplement to National Science Foundation award CCF-1319458.

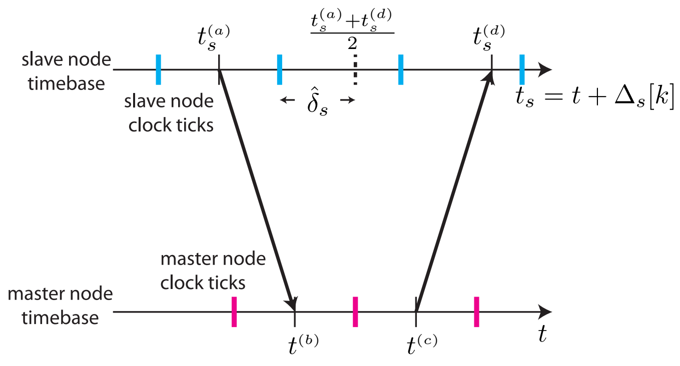

Fig. 1. Timestamp-free synchronization bidirectional signal exchange.

2. TIMESTAMP-FREE SYNCHRONIZATION PROTOCOL

Figure 1 shows the interactions between a slave node and the master node using the timestamp-free synchronization protocol. The time-varying clock offset at the slave node with respect to the master node is denoted as $\Delta_n[k]$ and local time at the slave node is denoted as

$$t_n = t + \Delta_n[k]$$

where t is the reference time corresponding to the local clock at the master node. The timestamp-free synchronization protocol begins with the slave node transmitting a signal to the master node at arbitrary local time $t_n^{(a)}$. The signal arrives at the master node at local time

$$t^{(b)} = t_n^{(a)} - \Delta_n[k] + \tau_n$$

where $\Delta_n[k]$ is the current clock offset of the slave node with respect to the master node and τ_n is the propagation delay between the slave node and the master node. The master node then transmits a signal back to the slave node at time $t^{(c)}$ where $t^{(c)}$ is selected such that

$$\frac{t^{(b)} + t^{(c)}}{2} \pmod{T_0} = 0 \tag{1}$$

where T_0 is master node *clock tick period*. Note that, unlike the usual sender/receiver synchronization protocol, e.g. [6], no timestamps are exchanged between the nodes. Implicit timing information is embedded in the master node's response to the slave node by selecting $t^{(c)}$ so that a local clock tick the master node is centered between $t^{(b)}$ and $t^{(c)}$. Assuming a reciprocal channel, the slave node receives the reply signal from the master node at local time

$$t_n^{(d)} = t^{(c)} + \Delta_n[k] + \tau_n.$$

The slave node can now estimate its clock tick offset with respect to the master node by calculating

$$\hat{\delta}_n = \left(\frac{t_n^{(a)} + t_n^{(d)}}{2}\right)_{T_0} \tag{2}$$

where the notation $(z)_{T_0}$ corresponds to wrapping z to the interval $[-T_0/2, T_0/2)$. The offset estimate in (2) can be used directly for immediate clock offset correction at the slave node or as an input to a filtering algorithm to correct both clock offsets and drifts.

This timestamp-free synchronization technique inherently accounts for propagation delay, and its accuracy is only limited by the fundamental bounds of delay estimation and the accuracy of the channel reciprocity assumption. In the related work that implemented the timestamp-free approach at acoustic frequencies [5], passband sampling was used due to the low rate of acoustic frequencies relative to modern A/D converter speeds. At RF frequencies, however, sampling rates are generally much *lower* than the carrier frequency, so that analog downconversion must be performed prior to sampling; in this case, delay estimation must generally be performed on baseband signals, and it is well-known that the inherent phase ambiguity limits the accuracy of delay estimation performed on baseband signals relative to passband signals [7, 8]. The maximum likelihood delay estimator is non-coherent in the sense that it ignores phase when processing the baseband waveform, and with sufficient signal to noise ratio (SNR) the Cramer-Rao lower bound for delay estimation can be written as [7, 8]

$$\mathrm{var}(\hat{\tau}) \geq \frac{24\pi}{W^3 T \cdot \mathrm{SNR}} \tag{3}$$

where W and T are the signal bandwidth and integration time, respectively, and SNR is the pre-integration SNR at the receiver.

3. DELAY ESTIMATOR

A fundamental building block of the timestamp-free synchronization protocol is accurate delay estimation. This section provides an overview of the delay estimator used in our real-time implementation of the timestamp-free synchronization protocol.

We assume the signals exchanged between the nodes are modulated sinc pulses. To avoid LO leakage during down conversion, we employ a "low IF" scheme where the sinc pulses in the complex baseband are frequency-shifted by a frequency Ω_0. The complex baseband modulated sinc pulse received by a node with unknown delay $\tau \in \mathbb{R}$ can be expressed as

$$r_\tau[k] = e^{j\omega_0(k-\tau)}\mathrm{sinc}(\eta(k-\tau))$$

for $k = 0, \dots, K-1$ where $\omega_0 = \frac{\Omega_0}{f_s}$ is the normalized intermediate frequency and $\eta = \frac{W}{f_s}$ is the normalized bandwidth of the sinc pulse. Cross-correlation with the template waveform $r_0[k]$ can be used to generate a nearest-sample delay estimate $k_0 \in \mathbb{Z}$.

To refine the estimate, we employ quadratic interpolation [9] by fitting a parabola to the cross-correlation peak and its two adjacent samples. We define $c[k]$ as the discrete-time cross-correlation between the template waveform $r_0[k]$ and the received pulse with unknown delay $r_\tau[k]$, so $c[k_0]$ is the cross-correlation peak. Then, the refined delay estimate can be computed as

$$\hat{\tau} = k_0 + \frac{1}{2}\frac{c[k_0-1] - c[k_0+1]}{c[k_0-1] - 2c[k_0] + c[k_0+1]}.$$

4. REAL-TIME IMPLEMENTATION

Three E310s were used in the implementation: one as the master node, a second as the slave node, and a third as the measurement device for determining the resulting clock offset. The E310s include an AD9361 2x2 RF Agile Transceiver, tunable from 70 MHz to 6.0 GHz. Each of the node's software was implemented in C++ using the USRP Hardware Driver (UHD) provided by Ettus. The driver is configured to transmit and receive buffers of 1000 samples at a time.

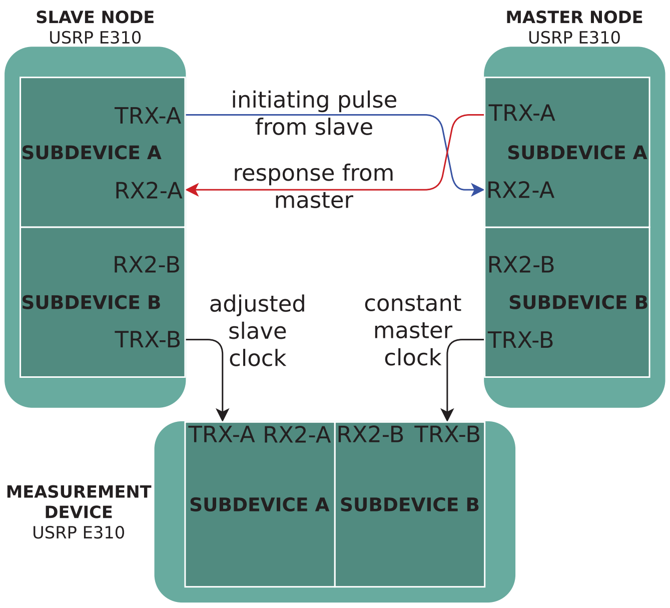

Fig. 2. Block diagram of the connections between the E310s used for the master and slave nodes. Each E310 has an independent local oscillator.

The test setup is shown in Fig. 2. The E310s have two subdevices "A" and "B" where each subdevice has one TX/RX channel (labeled TRX on the device) and one RX channel (labeled RX2 on the device). On both the master and slave node, subdevice A is used for the synchronization protocol and subdevice B is used to transmit clock signals. The slave node transmits a modulated sinc pulse on TRX-A and listens for a response on RX2-A. The master node listens for a pulse from the slave on RX2-A; when one is detected, it transmits a time-reversed recording of the received signal after a specified delay. For this implementation, the pulses were transmitted with a carrier frequency of 900 MHz and a sampling rate of 150 kHz.

While a higher sampling rate would have been preferable, the computational power of the ARM Cortex A9 on the Ettus E310 limited the number of clock cycles available for such operations as correlation, and 150 kHz was the highest sampling rate that did not result in buffer under/over-runs due to excessive processing time.

A state-diagram of the master node is shown in Fig. 3. Subdevice A facilitates all communication to and from the slave node and subdevice B simply facilitates the clock output. Both devices are controlled synchronously within the software, but are displayed on the diagram asynchronously for simplicity. On subdevice A, cross-correlation is used to detect the presence of a pulse. When the magnitude of the cross-correlation breaks a predefined threshold, the master node schedules the time reversed transmission of the received waveform to the slave node. The transmitter is set to a constant delay to allow for the buffers to be queued. The delay is used to guarantee to the slave node the presence of a clock tick at the half-way point between transmitting and receiving.

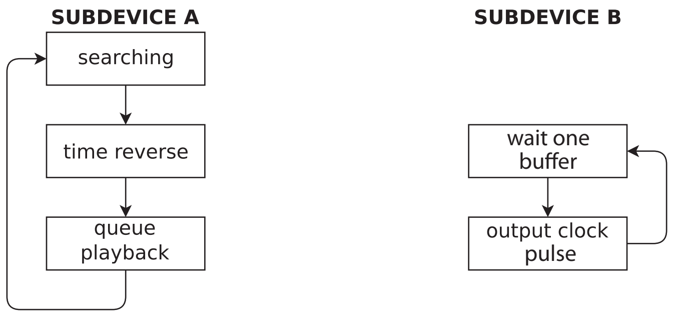

Fig. 3. Flow diagram of master node operation.

A state-diagram of the slave node is shown in Fig. 4. As with the master node, subdevice A facilitates all communication to and from the master node and subdevice B simply facilitates the synchronized clock output. Because the slave node is adjusting its clock based on the master's timed response, the implementation complexity of the slave node is significantly higher. When searching, the slave

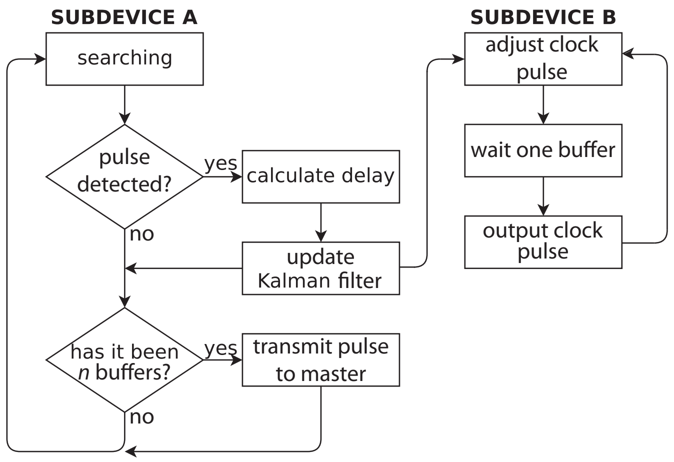

Fig. 4. Flow diagram of slave node operation.

node performs cross-correlation on every receive buffer in order to locate the precise sample that received the peak of the sinc pulse. As with the master, the slave node computes the magnitude of the cross-correlation when searching for a pulse, to allow the threshold to be well out of the noise margin. When the detection threshold is broken and a pulse is detected, the slave node sets a flag for calculation, saves the position of the peak, and saves the three points about the peak of the normalized cross-correlation. The slave node initiates the synchronization between the two nodes by transmitting a pulse every n buffers. The n-buffer delay is computed depending on the propagation between the master and slave nodes to ensure that the response from the master node is received before initiating another synchronization.

When the code enters the calculation block, the three points saved from the searching block are input to the quadratic interpolator. The result of the interpolator is used to calculate a prediction of the master node's clock using a two-state Kalman filter. The Kalman filter smooths and tracks the master node's relative rate and time with respect to the slave node. Using these rate and time estimates, the Kalman filter generates an improved prediction of when future master node clock pulses will occur. The filter output is used to generate a new, delayed sinc pulse that is transmitted on subdevice B. The fractional sinc pulse delay is adjusted before each transmission to compensate for the slight difference in clock rate of the master node and slave node.

5. EXPERIMENTAL RESULTS

This section summarizes our experimental results in a high-SNR over-the-wire setting. Modulated sinc pulses were transmitted from the slave with a period of 153.3 ms. The master node and slave node clocks were both recorded with a third E310 labeled "Measurement Device" as shown in Fig. 2.

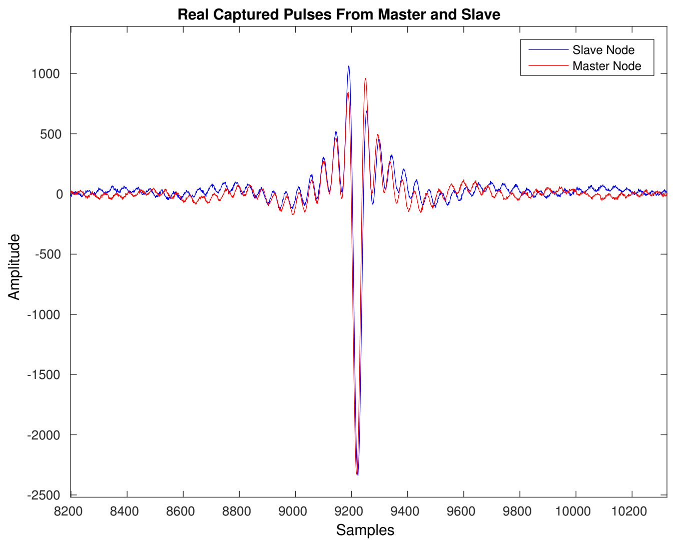

Fig. 5. Example excerpt recording of synchronized USRP clocks, sampled at 3 MHz.

The E310 used for measurement recorded the two clocks for 30s and saved the recordings to a file. The file was then transferred to a host PC for analysis in MATLAB. The accuracy of the "Measurement Device" was determined by sending pulses of known offset from one E310 to receiver-configured TRX channels on another E310. To increase the resolution of the measurement device the receiver was set to sample at 3 MHz, and all processing was performed offline. The E310 serving as the measurement device recorded each TRX channel and the time difference between the pulses was calculated using MATLAB and compared with the intended known offset. We found that the standard deviation of this test was approximately 32 ns. This precision is one of the limiting factors in our experimentation, as the E310 could not reliably measure an offset of less than 32 ns for our chosen pulse, sampling rate, and carrier rate.

Figure 5 shows a 354 μs recording of the real components of the fixed master node clock (shown in blue) and the adjusted slave node clock (shown in red). The master and slave node clocks were implemented with the same modulated sinc pulse as in the synchronization routine for simplicity. The clocks were set to occur every 6.67 ms, the duration of one buffer's worth of transmission.

To determine the effectiveness of the synchronization protocol, we ran 3 sets of tests. For each test, every combination of E310 was tested, each labeled A, B, or C. The M/S column of Tables 1, 2, and 3 indicates which E310 was used for the master and slave nodes, with the unlisted E310 used as the recording device.

The first test determined the Kalman Filter's (KF) precision and accuracy for predicting future samples. We allowed the Master and Slave nodes to synchronize for several minutes to ensure the KF had converged to steady values, using the last five minutes for recording and analysis. Table 1 shows the mean and standard deviation of the KF's prediction error, and the oscillator drift estimate measured in parts per million.

M/S	Run	mean$[\tilde{y}[k]]$	std$[\tilde{y}[k]]$	Drift Estim.
A/B	1	-0.1595 μs	0.1296 μs	+0.4124 ppm
A/B	2	0.1556 μs	0.2401 μs	+0.2529 ppm
B/A	1	0.1555 μs	0.1931 μs	-0.3805 ppm
B/A	2	0.0668 μs	0.1316 μs	-0.3753 ppm
A/C	1	-0.4451 μs	0.1519 μs	+2.0829 ppm
A/C	2	-0.1879 μs	0.2061 μs	+2.0054 ppm
C/A	1	0.2485 μs	0.1484 μs	-2.0171 ppm
C/A	2	0.1610 μs	0.1485 μs	-2.0054 ppm
B/C	1	-0.3019 μs	0.1336 μs	+1.6056 ppm
B/C	2	-0.4883 μs	0.1526 μs	+1.6388 ppm
C/B	1	0.4540 μs	0.1556 μs	-1.6347 ppm
C/B	2	0.0222 μs	0.1378 μs	-1.6113 ppm

Table 1. Summary of Kalman filter prediction error and drift estimates

The second test let the master and slave synchronize until KF convergence, and recorded the last 30s of clock pulses to analyze the offset between clocks. Each combination of E310's had a fixed bias and converged to a value not equal to zero. The data without bias calibration is shown in Table 2.

The third test was conducted identically to the previous test as presented in Table 2, however the average of the mean offsets from the two runs of each test was used to calibrate the offset towards zero. The calibrated data is shown in Table 3. Runs 1 and 2 often have significantly different mean offsets, which we conjecture is due to the temperature difference between runs 1 and 2; in run 1, the E310's were started cold, though in run 2 the E310's started warm from run 1.

With the one round of calibration we were able to achieve an accuracy within $\pm$0.8 μs. At our relative low sampling rate of 150 kHz – necessary to allow sufficient time for computation within the E310 software-defined radio – we note that the synchronization accuracy is within 12% of the sampling period, and higher sampling rates would yield much better synchronization accuracy.

M/S	Run	mean$[\epsilon[k]]$	std$[\epsilon[k]]$
A/B	1	-3.2789 μs	0.1047 μs
A/B	2	-3.3084 μs	0.0685 μs
B/A	1	-3.4708 μs	0.0717 μs
B/A	2	-3.4288 μs	0.0499 μs
A/C	1	3.6044 μs	0.1343 μs
A/C	2	3.5940 μs	0.0923 μs
C/A	1	3.2397 μs	0.0394 μs
C/A	2	3.3967 μs	0.0218 μs
B/C	1	3.3682 μs	0.0238 μs
B/C	2	3.3499 μs	0.0209 μs
C/B	1	3.2833 μs	0.0203 μs
C/B	2	3.3145 μs	0.0236 μs

Table 2. Mean and standard deviation of uncalibrated clock offsets

M/S	Run	mean$[\epsilon[k]]$	std$[\epsilon[k]]$
A/B	1	-0.2445 μs	0.0404 μs
A/B	2	0.0661 μs	0.0671 μs
B/A	1	0.3143 μs	0.0214 μs
B/A	2	0.0212 μs	0.0251 μs
A/C	1	-0.7754 μs	0.0197 μs
A/C	2	-0.6381 μs	0.0355 μs
C/A	1	0.3947 μs	0.0203 μs
C/A	2	0.2167 μs	0.0191 μs
B/C	1	-0.4351 μs	0.0236 μs
B/C	2	-0.4185 μs	0.0258 μs
C/B	1	0.6130 μs	0.0222 μs
C/B	2	0.0439 μs	0.0203 μs

Table 3. Mean and standard deviation of calibrated clock offsets

6. CONCLUSION

This work demonstrates the feasibility of the timestamp-free synchronization algorithm for use in modern wireless networks. We presented a real-time implementation of timestamp-free synchronization using RF signaling and an Ettus USRP software-defined radio platform. Experimental results demonstrated that the slave node is able to predict the clock pulse observations from the master node to within 12% of the sampling period over 0.153s prediction intervals.

Source code for this experiment is available from:
https://github.com/agklein1/tsfreesync.git

7. REFERENCES

[1] D.L. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. Commun.*, vol. 39, no. 10, pp. 1482–1493, Oct. 1991.

[2] "IEEE 1588 Home Page," http://www.ieee1588.com/.

[3] W. Lewandowski, J. Azoubib, and W.J. Klepczynski, "GPS: primary tool for time transfer," *Proceedings of the IEEE*, vol. 87, no. 1, pp. 163–172, Jan 1999.

[4] D.R. Brown III and A.G. Klein, "Precise timestamp-free network synchronization," in *Conf. Inf. Sciences and Systems (CISS2013)*, Mar. 2013.

[5] M. Li, S. Gvozdenovic, A. Ryan, R. David, D. R. Brown, and A. G. Klein, "A real-time implementation of precise timestamp-free network synchronization," in *2015 49th Asilomar Confer-*

ence on Signals, Systems and Computers, Nov 2015, pp. 1214–1218.

[6] S. Ganeriwal, R. Kumar, and M.B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings ACM SenSys 2003*. ACM New York, NY, USA, Nov. 2003, pp. 138–149.

[7] A. Weiss and E. Weinstein, "Fundamental limitations in passive time delay estimation–Part I: Narrow-band systems," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 31, no. 2, pp. 472 – 486, April 1983.

[8] E. Weinstein and A. Weiss, "Fundamental limitations in passive time-delay estimation–Part II: Wide-band systems," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 32, no. 5, pp. 1064 – 1078, October 1984.

[9] Giovanni Jacovitti and Gaetano Scarano, "Discrete time techniques for time delay estimation," *IEEE Transactions on Signal Processing*, vol. 41, no. 2, pp. 525–533, 1993.