# Network Synchronization in Wireless Ad Hoc Networks

Carlos H. Rentel and Thomas Kunz
Department of Systems and Computer Engineering, Carleton University
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6 Canada
{crentel, tkunz@sce.carleton.ca}

**Abstract**. This report presents a brief overview of network synchronization in the context of Wireless Ad Hoc networks. A network synchronization algorithm is proposed and computer performance evaluations presented that show the feasibility of a mutual network synchronization algorithm compatible with an IEEE 802.11 complaint station. The algorithm proposed seeks to synchronize the clocks of the nodes in a Wireless Ad Hoc network in which single-hop (Wireless LAN) or multi-hop communication is supported. No use is made of a master clock such as the one provided by the Global Positioning System (GPS) or by a fixed cluster of broadcasting stations. The algorithm takes advantage of the simplicity of the clock-sampling technique to exchange timing information among the nodes, and the truly distributed characteristic of a mutual network synchronization approach. The beacon messages specified in the IEEE 802.11 standard are used to periodically send the timestamps by each node. A steady-state inter-node time-deviation error less than ten microseconds is obtained after a transition period during which the clocks in the network learn about their drift differences. Performance results for the single-hop and multi-hop case are presented using computer simulations that show a considerable improvement in terms of scalability and maximum inter-node time deviation error with respect to the timing synchronization function (TSF) of the IEEE 802.11 independent basic service set (IBSS) network. Additional results are presented for a modification of the original algorithm in which beacon transmission permission probabilities are computed per node in order to improve the spreading of the synchronization information in a multi-hop network. The contribution of this work is, 1) The performance evaluation of the TSF in a multi-hop scenario, 2) the performance evaluation and approximate analysis of simple extensions to the TSF algorithm, and 3) The introduction of a more scalable and accurate network synchronization algorithm for wireless Ad Hoc networks.

## 1. Introduction

A Wireless Ad Hoc network is a distributed communication network comprised of geographically separated radio terminal units or nodes (mobile or fixed) that perform the management and regular operation of the network over a wireless transmission medium. A Wireless Ad Hoc network is autonomous and operates by the shared responsibility of the entities that provide the communication service itself, all or part of the nodes implement the required applications to be host, router and transmission medium interface. In this respect a wireless Ad-Hoc network differs from a cellular radio network in that it does not make use of a centralized and fixed infrastructure. One of the most interesting advantages of Wireless Ad Hoc networks are the possibility of multi-hop mode of communication, and the lack of a fixed centralized infrastructure. These advantages make them attractive for battlefield and disaster relief applications in which the deployment of a fixed infrastructure represents a high cost. Wireless Ad Hoc networks have been traditionally related to wireless local area networks (wireless LANs), however, recent advances in semiconductors have opened the venue for networks of tiny sensors that can perform data collection and analysis. In the context of network synchronization we are mainly concern with the performance of a distributed wireless computer network such as a

wireless LAN, a multi-hop LAN, mesh network (multi-hop WAN), and also a network of sensors. However, we are less concern with the synchronization approaches that seek to find time-relationships of events for data fusion in sensor networks, such as the works in [1], and [2]. The Wireless Ad Hoc network concept is not new, and it embodies the same principles originally envisioned for the packet radio networks.

The evolution of wireless Ad Hoc networks is mainly dependent on the efficient support of multimedia applications. Users want to be able to simultaneously work or play with video, audio, and data applications through existing and new services. The spectrum of applications and services offered by wireless networks is expected to increase and mechanisms to support them are needed. It is imperative therefore to focus on quality of service (QoS) mechanisms that can support the performance requirements of different types of traffic simultaneously. Efforts are underway to provide QoS in the Internet through the Integrated Services and Differentiated Services architectures. Efforts must therefore follow to make wireless Ad Hoc networks QoS aware as well.

Real-time applications such as voice, streaming video, and audio have specific requirements in terms of end-to-end delay, delay jitter and packet loss ratio. Providing support for these applications is particularly challenging in a wireless Ad Hoc network due to the lack of infrastructure and the broadcasting and highly random nature of the wireless medium. Most present day telecommunication networks rely on a fixed infrastructure to provide some level of QoS support; examples of this are the cellular radio networks. The infrastructure in a cellular radio network alleviates the complexity of supporting different mobile user applications by providing time synchronization, and a central processing point through which important information can be extracted from all mobile users simultaneously and in the same location. The centralized architecture is not available to a pure wireless Ad Hoc network; therefore it is more challenging to implement distributed QoS support to a group of nodes scattered in space yet sharing a common transmission medium. One mechanism common to many telecommunication networks is that of network synchronization. Network synchronization is utilized to simplify and enhance the implementation and performance of medium access control protocols, security protocols, and management operations to name just a few. Network synchronization is a key function for instance in the IEEE 802.11 IBSS Ad Hoc mode to perform power management of the nodes, and support of the medium access control (MAC) protocol in the Frequency Hoping Spread Spectrum version of the physical (PHY) layer. Network synchronization is crucial in the recently standardized 802.16a [3] mesh network option since the MAC protocol is based on Time-Division-Multiple-Access (TDMA). Network synchronization can play a fundamental role in the support of QoS for future Wireless Ad Hoc networks, particularly for real-time applications, and more understanding of its advantages, and associated challenges is needed.

In this report we present a novel distributed network synchronization algorithm. The performance of the proposed algorithm is evaluated and compared with that of the timing synchronization function (TSF) in the IEEE 802.11 standard [4]. The remainder of this report is organized as follows: In Section 2 we present some background and motivation on network synchronization along with the description of some related work. In Section 3 the TSF is presented along with analysis and discussion of some simple extensions to the original TSF algorithm. Section 4 describes the proposed algorithm in detail. The performance evaluation for the single-hop or Wireless LAN (WLAN) case is presented in Section 5. The multi-hop variation of the proposed algorithm is discussed in Section 6. Performance evaluation of the multi-hop case is presented in Section 7, and finally Section 8 concludes the report.

## 2. Network Synchronization

The synchronization of geographically separated clocks has intrigued researchers for a long time. The first *accurate* clocks started to appear in the 17th century after the scientific work pioneered by Galileo and expanded by Christian Huygens [5]. These scientists helped develop a better clock based on their theories of the motion of pendulums. More recent advances in the area of clocks include the development of the atomic clocks that are used in numerous applications including the Global Positioning System (GPS), and in the digital telephone communication network in order to provide timing reference [6]. In the area of clock synchronization we will present a summary of some important related works later on this section, but first some background on the basic theory of clocks.

A clock is a time measurement device. It fundamentally comprises an oscillator and an accumulator (Figure 1). The oscillator's task is to generate periodic events and the accumulator (i.e., integrator) adds-up these events in order to obtain the measured time. In Figure 1 the oscillator produces a sinusoidal waveform at its output; in this case the zero-crossing events of the sinusoidal waveform are detected, and the accumulator adds-up the number of zero-crossing events. *Time* represents the measured *time* in Figure 1. The output of the accumulator in Figure 1 is an ideal straight line with a unit slope ($\partial Time / \partial time = 1$), in practice this is not the case and several problems can be identified that cause the measured time (*Time*) to differ from the absolute or reference time (*time*).

The most difficult errors to correct in a clock are those due to the oscillator imperfections. These imperfections are due to ambient conditions such as temperature, relative humidity, pressure, and also imperfections in the construction and materials used that cause frequency instabilities. In this work an accurate oscillator is considered to be one based on quartz crystals up to atomic standards with frequency accuracies in the order of tens of ppm. The IEEE 802.11 standard specifies a clock-accuracy for its Pseudo-Noise (PN) sequence generator not greater than ±25ppm [4]. Clocks in personal computers utilized quartz-crystal oscillators with frequency accuracy in the range of 10 to 100ppm. Frequency stability requirements for cellular radios are very tight in the range of 1 or 2ppm [7]. A common model for the phase error of many free-running accurate oscillators can be described by the following equation [8]

$$\Phi_e(t) = \Phi_{e0} + w_{e0}t + \frac{1}{2}Dt^2 + \xi(t) \qquad (1)$$

The phase error is measured using a phase detector that compares the phases of the oscillating waveform under test and the waveform produced by a reference oscillator. $\Phi_e(t)$ is the phase error as a function of real time, $\Phi_{e0}$ is the initial phase error, $w_{e0}$ is the fixed frequency offset of the oscillator with respect to its nominal frequency $w_0$, $D$ is the frequency drift coefficient, and $\xi(t)$ is a random process that models the short-term oscillator imperfections. In synchronization parlance the $D$ coefficient contributes to the *phase or frequency wander* of the oscillating waveform with respect to a reference, and the $\xi(t)$ term contributes to its *jitter*. The $\xi(t)$ term is a random process that has been characterized with a power-law spectra, in other words the Fourier transform of the frequency or phase errors between an oscillator and its reference is computed, and it has been found to be of the form $f^b$, where $b$ identifies the power of the process,

if $b = 0$, the process is a white noise process. All accurate clocks exhibit this random behavior, for details see [8].

An ideal oscillator will have a zero frequency drift (i.e., $D$ = 0 Hz/day), and no short-term variations ($\xi(t) = 0$), assuming the initial phase ($\Phi_{e0}$) and frequency offset ($w_{e0}$) errors can be corrected. Frequency is the derivative of phase, therefore the frequency error between the test and reference oscillator is given by

$$w_e(t) = w_{e0} + Dt + \dot{\xi}(t) \tag{2}$$

Equation (2) shows the instability of non-ideal oscillators due mainly to the frequency drift coefficient. As time progresses the frequency of the test oscillator drifts from the frequency of the reference oscillator linearly with respect to the reference time. It is possible however to estimate the drift coefficient and adjust the phase of the oscillator accordingly, but no estimation is perfect and therefore drifting of the frequency will dominate the frequency error of the clock again. Equation (1) is still an imperfect model since the oscillator's frequency and phase also depend on ambient variables such as temperature, humidity, and pressure. For these reasons, it is common practice to adjust the clock timing process utilizing a combination of several methods such as drift coefficient estimation, temperature compensation, and periodic refresh coming from a more accurate clock. Using equation (2), the frequency of the oscillator is

$$w(t) = w_o + w_{e0} + Dt + \dot{\xi}(t) \tag{3}$$

Where $w_0$ is the nominal frequency of the oscillator. If equation (3) is integrated in time we obtain the phase of the oscillator

$$\Phi(t) = \int_0^t \left(w_0 + w_{e0} + Dt + \dot{\xi}(t)\right)dt = w_0 t + w_{e0} t + \frac{D}{2}t^2 + \xi(t) - \xi(0) + \Phi(0) \tag{4}$$

Where $\Phi(0)$ is the initial phase. Ideally a perfect clock's phase equation would be

$$\Phi_{ideal}(t) = w_0 t \tag{5}$$

Therefore dividing the phase in (4) by the nominal frequency of the oscillator we obtain the measured time of the clock modeled according to equation (1)

$$T(t) = t + \frac{w_{e0}}{w_0}t + \frac{D}{2w_0}t^2 + \frac{\xi(t)}{w_0} + \Theta(0) \tag{6}$$

The measured time *T(t)* is a random process.

More formally, several important terms are defined for a clock based on the time process *T(t)* [9].

The *offset* of a clock *a* with respect to a clock *b* or reference time *t* is $T_a(t) - T_b(t)$ or $T_a(t) - t$ respectively. The *skew* is the difference between the rate of change of time of two

clocks, $T_a'(t) - T_b'(t)$ or $T_a'(t) - 1$, where $T'(t) = dT(t)/dt$, and the *drift* is defined as $T_a''(t) - T_b''(t)$. Taking model (6), and assuming that $\xi(t) = 0$ and $\Theta(0) = 0$, we have for two clocks *a* and *b* with the same nominal frequency

$$\text{Offset} = \left(\frac{w_{ae0} - w_{be0}}{w_0}\right)t + \left(\frac{D_a - D_b}{2w_0}\right)t^2 \quad (7)$$

$$\text{Skew} = \left(\frac{w_{ae0} - w_{be0}}{w_0}\right) + \left(\frac{D_a - D_b}{w_0}\right)t \quad (8)$$

$$\text{Drift} = \left(\frac{D_a - D_b}{w_0}\right) \quad (9)$$

Offset indicates the difference in units of time of the two time processes as real time evolves, skew can be interpreted as the rate of change of offset in units of time over time, and drift is the rate of change of skew in units of frequency over time. A more general equation modeling a clock will include multiple derivatives of the offset beyond the second derivative (see [10]), but this accurate analysis is beyond our interest in this paper.

According to our oscillator model in equation (1), time offset increases as a quadratic function of real time, and skew increases linearly with respect to real time. Figure 2 shows the time offset of an oscillator with respect to real time. The oscillator has a nominal frequency of 2GHz, a maximum fixed frequency error of 10Hz, and a frequency drift coefficient of +100Hz/min. The offset of *T(t)* with respect to *t* after one day is approximately +3 seconds. Note that this is a worst case scenario since an oscillator will not drift its frequency indefinitely in one direction except very slowly due to aging. The most common scenario for a crystal oscillator used in telecommunication radios is to be at an approximately constant frequency offset from its nominal frequency (e.g., accuracy in the range 10-100 ppm [7]), therefore, the time offset will be approximately a linear function of reference time.

In two perfectly synchronized clocks the offset in equation (7) is zero. In practice however, this is not the case, and ways to synchronize different clocks must be found if a useful measure of time is desired in a different location to that in where the more accurate clock is located. In this study we are interested in synchronizing clocks located in geographically separated nodes of a wireless Ad Hoc network. Network synchronization is a difficult task particularly due to the real characteristics of the oscillators in the clocks as described previously, and also due to the delay and delay variation in the links used to transfer the timing information among the nodes comprising the network.

There are two commonly known approaches for clock synchronization [10], centralized and decentralized. The centralized synchronization approach is also known as master-slave synchronization and it is the most common method encountered in practice in civilian applications, there is one or more accurate clocks (the master(s)) to which all the rest of the clocks listen and adjust their frequency and phases accordingly.

The decentralized synchronization approach is also known as mutual synchronization, in this approach there is no master clock, but instead all clocks cooperate to achieve synchronization in a distributed manner. In mutual synchronization the clock of a node tries to achieve synchronization by reducing its phase or timing error with respect to a weighted average of the other clocks' phases.

In practice it is necessary to exchange timing information messages among the different nodes comprising the network. There are several approaches for doing this and some of the most important are:

- Burst position measurement
- Continuous correlation of timing signals and,
- Clock-sampling

In the burst position method each node schedules the periodic transmission of a burst or pulse. At each receiving node the positions of the incoming bursts are compared with the position of the local burst and the difference is used to correct the local clock period according to a many-to-one mapping. This mapping usually takes the form of a weighted average of the errors. The transmission of pulses has the disadvantage of requiring a large bandwidth and possibly a dedicated channel if a wireless medium is used.

In the continuous correlation method each node continuously transmits a signal that is tracked at the receiving node. At each receiving node the sequence is compared with a replica generated by the local clock and a sliding correlation is performed in order to compute the phase offset. For instance, a clock can drive a pseudo-noise (PN) sequence generator, and this sequence can be transmitted to other nodes. As in previous cases a many-to-one mapping is needed to extract the correction term used to adjust the local clock.

In the clock-sampling method each node reads the time of its clock and transmits it to other neighbor nodes. At each receiving node the timing errors are computed as the difference between the local and neighbor nodes' clocks. The errors are used in a many-to-one mapping rule to determine the correction applied to the local clock. The main advantage of the clock-sampling technique is its relative simplicity of implementation. The TSF in the IEEE 802.11 standard is a clock-sampling method. In all these methods the timing information exchanged can be corrupted during the time it travels from transmitter to receiver. Some of the most detrimental factors include link delay, signal fading, signal delay spread, and collision of timing messages due to the broadcasting nature of the wireless medium.

In the context of wireless Ad Hoc networks our goal is to achieve inter-node time synchronization of clocks utilizing a combination of clock-sampling and mutual network synchronization. The use of the master-slave approach is feasible by utilizing the reference time provided by the atomic clocks in the GPS satellite system. Accuracy in the order of a few hundred nanoseconds is possible [7]. However, it is of interest to study the feasibility of a distributed inter-node time synchronization scheme due to its higher robustness, deployment flexibility and other factors that will become clearer later. Known disadvantages of a GPS approach include the fact that GPS signals can be jammed, and are only available in outdoor areas. For small networks of devices or computers in conference rooms, classrooms, and secluded places it is also desirable to achieve synchronization regardless of the operating environment (i.e., indoor or outdoor). Additionally, the IEEE 802.11 IBSS network [4] takes advantage of synchronization for power and channel management through the TSF, therefore a mechanism that improves over TSF can also translate into improvements of network management that further translate into better QoS

support in an already standardized wireless local area network technology. Recent work [11] has identified the necessity to improve the TSF mechanism due to its lack of scalability.

The algorithm proposed in this report is based on a mutual synchronization approach of which the work by Gersho and Karafin [12] is one of the most representatives. In [12], stability of a mutual network synchronization scheme is proved through a mathematical analysis based on classical control theory. Geographically separated oscillators are directly controlled in a distributed manner through a multiple-input phase-locked-loop (PLL) [13] approach. Each input of the multiple-input PLL located in every oscillator is the timing information exchanged with neighbor oscillators utilizing the available channels (i.e., wired or wireless). One important parameter for the stability of the approach is the latency of the communication links. Link latency affects the validity of the timing information exchanged among the oscillators and ways to estimate it and compensate for it are required in case it is not negligible. The work in [14] proposes a similar approach to the one in [12] for the time synchronization of time-division multiple access (TDMA) cellular base stations. Our work differs from these early works in that there is no direct physical control of the clocks or oscillators in every node. Other novelties of our approach are the use of the simple clock-sampling technique to exchange timing information in combination with a mutual network synchronization approach, and the introduction of a simple mechanism to enhance the synchronization performance in the case of a multi-hop network.

Mutual network synchronization approaches are proposed for a wireless Ad Hoc network of automobiles in [15] and [16]. However, no study is made of the performance of these approaches in a multi-hop network, and in one case the timing information is exchanged using very short pulses that use rather large bandwidth resources. In [11] and [17] the lack of scalability of the TSF of IEEE 802.11 is first analyzed and new methods are proposed to improve it. Again, no study is made of the multi-hop network case (The authors mention the interest in pursuing this study later). A network synchronization scheme is proposed with multi-hop support in [2]. In this approach the synchronization of multi-hop neighborhoods is achieved virtually through the exchange of messages with an intermediate node (i.e., a node in between separated neighborhoods). The purpose is to obtain a logical ordering of time events rather than achieving real synchronization of the clocks, which is fundamentally the same approach of the work in [1] for data fusion in sparse sensor networks. Our approach requires the connectivity of the network at least momentarily in order to achieve clock synchronization; only partial connectivity of the network is needed. Our algorithm does not require knowledge of the maximum drift of any clock in the network. In practice a clock's maximum drift is specified within a positive and negative range such as ±25ppm, this represents a maximum error of 50ppm between the real drift and the assumed drift for two given clocks. Therefore, it is not accurate enough for our purposes to use the maximum drift value for our network synchronization algorithm. We assume no link delay since the maximum separation of neighboring nodes is in the order of few hundred meters as it is the common case in a WLAN (i.e., 200m $\approx 0.6\mu secs$).

## 3. IEEE 802.11 Timing Synchronization Function (TSF)

The TSF in the IEEE 802.11 standard is a network synchronization algorithm that utilizes the clock-sampling method to exchange timing information. The TSF is summarized next for the IBSS (Ad Hoc mode) case.

1. Each node sends a beacon periodically at a *Target Beacon Transmission Time* (TBTT) with period *aBeaconPeriod* (e.g., 0.1*sec* [4]). At each TBTT each node shall:

2. Suspend the back-off timer of any pending non-beacon transmission.
3. Calculate a random delay uniformly distributed in the range between zero and $2 \cdot aCW\ \min \cdot aSlotTime$ . Table 1 shows the values of $aCW\ \min$ and $aSlotTime$ for the IEEE 802.11 standard with different PHY layer versions.
4. Wait for the period of random delay before transmitting the beacon.
5. Cancel the remaining random delay and the pending beacon transmission, if a beacon arrives before the random delay timer has expired.
6. Send a beacon if the random delay has expired and no beacon has arrived during the delay period.

Upon reception of a beacon a node will adjust the received timestamp to take into account its physical layer delay. The node will set its clock to the value of the adjusted timestamp if it is later than the timestamp value of the given node's clock. The TSF clock is a 64 bit counter with a $1\mu sec$ resolution.

| Parameter | FHSS | DSSS | OFDM |
|---|---|---|---|
| $aCW\ \min$ | 15 | 31 | 15 |
| $aSlotTime$ | 50 | 20 | 20 |

**Table 1. Beacon contention window parameters for IEEE 802.11**

An approximate analysis of TSF in the single-hop case was first attempted in [11]. The probability of sending one beacon successfully regardless of the node that sent it ( $P_{any}$ ), and the probability of sending a beacon successfully by a given node ( $P_{given}$ ) were found under the assumption that perfect synchronization has been achieved. That is, the beacon contention window of every node starts at the same time for all the nodes in the network. However, the analysis proves the inefficiency of TSF to scale even to a moderate number of nodes. The reason for the lack of scalability of TSF is blamed to the beacon collisions, which make $P_{given}$ small as the number of nodes in the network increases. Figure 3 repeats the curve shown in [11] for $P_{given}$ versus the number of nodes in a network. The parameters used in Figure 3 correspond to those used in an IEEE 802.11 FHSS network (see Table 1). The beacon transmission takes 11 slots (i.e., $550\mu secs$). As seen in Figure 3, for a network of 20 nodes the probability of a given node to transmit its beacon is approximately 0.05. This is equivalent to say that the probability of receiving a beacon from the node with the fastest clock is 0.05 when the number of nodes in the network is equal to 20. The low probability of sending a beacon successfully by the node with the fastest clock translates into severe a-synchronism when the clocks of the network drift with respect to one another. This affects power management, and the channel hoping procedure in the IEEE 802.11 standard, furthermore, it proves the lack of scalability of the TSF algorithm.

A simple way to try to improve TSF is to allow beacon transmissions even after successfully receiving a beacon (hereafter called secondary beacon transmissions). That is, modify steps 5 and 6 above in the description of TSF in order to allow a node to transmit its beacon even after successfully receiving one. This approach is not the most ideal however, since we are improving TSF at the expense of increasing overall network energy consumption and overhead. Energy consumption will be increased since more beacons will be transmitted in average, and overhead increases since it is more likely that the contention window will be extended further by the extra beacon transmissions. A larger beacon contention window implies smaller bandwidth for actual data transmission.

The modified TSF is analyzed extending the approximate analysis in [11]. The probability of a given station to transmit its beacon in the modified TSF ( $\hat{P}_{given}$ ) is given by

$$\hat{P}_{given}(n,W) = \frac{1}{W+1}\sum_{k=0}^{W} \hat{P}^{k}_{given}(n,W,k) \tag{10}$$

Where $\hat{P}^{k}_{given}(n,W,k)$ is the conditional probability that the given node successfully transmits a beacon given that it is scheduled to transmit in slot *k*; $W+1$ is the contention window size (there are $W+1$ slots labeled 0 through $W$), and *n* is the number of nodes in the network. $\hat{P}^{k}_{given}(n,W,k)$ can be computed based on the same events outlined in [11] plus an additional one allowing one node to transmit even after a successful beacon reception. $\hat{P}^{k}_{given}(n,W,k)$ is given by

$$\hat{P}^{k}_{given}(n,W,k) = \begin{cases} \left(\frac{W-k}{W+1}\right)^{n-1} & for\ k < b, \forall n\ or\ k \geq b, n = 0,1 \\ \left(\frac{W-k}{W+1}\right)^{n-1} + \sum_{i=0}^{k-b}\left\{\left(\frac{1}{W+1}\right)\cdot \hat{P}^{k}_{given}(n-1,W-i-b,k-i-b)\right\} & for\ k \geq b, n = 2 \\ \left(\frac{W-k}{W+1}\right)^{n-1} + \sum_{i=0}^{k-b}\sum_{x=1}^{n-1}\sum_{y=0}^{n-1-x}\left\{C_x^{n-1}C_y^{n-1-x}\left(\frac{1}{W+1}\right)^{x}\left(\frac{b-1}{W+1}\right)^{y}\right. & \\ \left.\cdot\left(\frac{W-i-b+1}{W+1}\right)^{n-1-x-y}\hat{P}^{k}_{given}(n-x-y,W-i-b,k-i-b)\right\} & for\ k \geq b, n \geq 3 \end{cases} \tag{11}$$

The boundary conditions in equation (11) are $\hat{P}^{k}_{given}(0,W,k) = \hat{P}^{k}_{given}(n,0,k) = (\hat{P}^{k}_{given}(n,W,k)$ $\forall\ W < k) = 0$, and $C_x^n = \frac{n!}{x!\cdot(n-x)!}$. The first expression in equation (11) is the event that no beacon transmissions occurred before slot *k*. The second expression is the event that a single successful transmission or no transmissions occurred before slot *k* (at slot *i*) when only 2 nodes are in the network. The third term is the event that exactly *x* beacon transmissions occurred in slot *i*<*k*, where $(1 \leq x \leq n-1)$, and that exactly *y* nodes $(0 \leq y \leq n-1-x)$ are scheduled to transmit in slots $i+1$ through $i+b-1$. The nodes scheduled to transmit during the latter interval will defer their transmissions because of the beacon transmission that started at slot *i* (Carrier sense). A beacon takes *b* slots to transmit.

Equation (11) is plotted in Figure 4 along with $P_{given}$ of TSF and simulation points of the modified TSF. The simulation points were obtained after 30 minutes of real-time simulation. As can be seen, equation (11) corresponds to the simulation data quite well. The first thing to notice about the modified TSF is that, as expected, it achieves a better probability of beacon transmission than TSF, however, it still suffers from severe degradation when the number of nodes in the network increases. Therefore, although in a lesser degree, the modified TSF suffers from the same scalability problems of the original TSF. One could try to further improve the modified TSF and allow secondary beacon transmissions only from those nodes that had a larger timestamp than the timestamp received in the previous beacon. We performed simulations of this approach in a network of 10 and 20 nodes in which the fastest clock drift at +25ppm (i.e., gains

2.5$\mu secs$ with respect to real time every *aBeaconPeriod* = 0.1 seconds), and the rest of the clocks drift at –25ppm, with the FHSS parameters. Figure 5 shows the cumulative distribution function (c.d.f) of the maximum time difference among the nodes of the network. The modified TSF achieves better performance since the maximum time difference is smaller; however, it is still unsatisfactory because the small accuracy is gained at the expense of almost twice as much beacon transmissions. Note that the accuracy of the modified TSF with 20 nodes is roughly the same as the original TSF with 10 nodes (Figure 5). This is approximately what we observe in Figure 4 if a horizontal line is drawn from the point of 10 nodes in the TSF curve, to the intersecting point with the curve of the modified TSF.

Based on the previous analysis, we argue that a synchronization algorithm that truly improves over the IEEE 802.11 TSF should:

1. Abandon the idea of giving the responsibility of network synchronization to a node with particular characteristics (e.g., fastest clock, node with largest cardinality etc). This will improve the chances of spreading the timing information, increase the robustness of the algorithm to network dynamics, and increase the speed of convergence of the algorithm (more about this later).
2. Take full advantage of the exchanged timestamps in order to avoid the need to continuously refresh the clocks in the network.

The latter goals should be achieved with small increase in energy expenditure and overhead over the already standardized algorithm. We show that the algorithm presented in the next section achieves a good performance in terms of accuracy, and scalability, with negligible extra energy expenditure and overhead cost.

## 4. Clock-sampling Mutual Network Synchronization: WLAN case

The root of the TSF problems is that it differentiates among the nodes in the network based on the drift of the clocks, which is an unknown parameter (i.e., only the maximum and minimum drift values might be known). The TSF and the algorithms presented in [11] and [17] make the node or a subset of nodes with fastest clocks the most important nodes in the network in terms of synchronization. Giving explicit or implicit priority to a subset of the node population makes the algorithm less distributed and therefore less resilient to node failure and network dynamics. The argument to give priority to the fastest node is to avoid backward leaps in time, which is a necessary requirement for some applications. Furthermore, a mechanism that tries to discover the node with the fastest clock increases the latency of the network synchronization approach (time is needed to find the node with the fastest clock). Our algorithm departs from the idea of giving priority to any node, yet the time can be ensured to go forward. This algorithm is in principle a mutual network synchronization scheme in which all nodes participate equally in the overall synchronization of the network and therefore no time is spent discovering nodes with particular characteristics.

The probability of transmitting a beacon successfully regardless of the node that transmitted it ( $P_{any}$ ) can give an indication of the improvement of a mutual network synchronization algorithm over the more centralized TSF algorithm. The authors in [11] found a recursive formula for $P_{any}$, which matches our simulation shown in Figure 6 for the parameters of the FHSS and DSSS PHY versions of IEEE 802.11. Figure 6 suggests that a mutual network synchronization algorithm

based on the same beacons transmitted in the IEEE 802.11 standard can greatly improve over the TSF and the modified TSF algorithm presented earlier (see Figure 4). Therefore, our efforts should be directed towards more distributed network synchronization approaches that exploit the information carried by every beacon transmitted.

Our algorithm is as follows:

1. We define a controlled clock and a real clock in each node. The real clock can be the same timer used in the TSF of IEEE 802.11 (64 bits @ $1\mu sec$ resolution). The controlled clock reads from the real clock and adjusts the value read by a correction factor we call $\boldsymbol{s}$. Synchronization information for any purposes (e.g., management, security, MAC support, space-time event relationships etc) is taken from the controlled clock. We see $\boldsymbol{s}$ as a control parameter that adjusts the speed of the controlled clock regardless of the real clock's drift. If $\boldsymbol{s} = 1$, the controlled clock is no different from the real clock except for a negligible difference caused by the processing time of $\boldsymbol{s}$. Figure 7 shows the relationship between the controlled and real clocks. Without loss of generality we assume that the physical layer and processing delays have been taken into account and the controlled timestamp has been adjusted accordingly if necessary.
2. A node must scan beacons for some period of time in order to acquire synchronization before joining the network. The node listens for beacons and sets the timestamps of the controlled and real clocks to the value of the timestamp received. The value of $\boldsymbol{s}$ is set to 1 at initialization. The requirement to acquire some information at the beginning of a session when a node enters a network is common for other protocols and other types of networks, such as routing protocol information in wireless Ad Hoc networks or system information parameters in cellular radio networks. It is, for instance, a requirement in the IEEE 802.16a standard to acquire coarse synchronization at initialization. We assume that initial coarse synchronization of the nodes has been achieved before the main algorithm is run.
3. All nodes contend at specific intervals of time in order to send their beacons with their controlled timestamps.
4. When a node receives a beacon successfully it will not content in the next T_DELAY TBTTs to send its beacon. Each node that received a beacon successfully less than T_DELAY TBTTs ago will decrement the value of a T_DELAY counter by one in every TBTT regardless of the successful reception of a beacon. Once the T_DELAY counter reaches one, the node will store the value of the controlled timestamp in the real timestamp register, set $\boldsymbol{s}$ =1, and contend again to send its beacon. If a node does not receive any beacon successfully it will continue to send its beacons in every TBTT. We refer to the node that wins the contentions as the *reference node*, and the process of changing the reference node after some time dependent on the value of the T_DELAY counter as *reference node hoping*. We also refer to the maximum value of the T_DELAY counter as T_DELAY parameter or T_DELAY, with the understanding that the T_DELAY counter starts counting down from a value equal to T_DELAY in steps of one until one.
5. If the $i^{th}$ node successfully receives a beacon, it will adjust $\boldsymbol{s}$ based on the error $e_i$ computed as the difference between the received timestamp and the $i^{\text{th}}$ node controlled timestamp. The value of $\boldsymbol{s}$ for the $i^{th}$ node at the $n^{\text{th}}$ TBTT is then computed as follows

$$s_i(n) = s_i(n-1) + \frac{K_p e_i(n)}{controlled\ time\ stamp_i(n)} \qquad (12)$$

Where $K_p$ is the proportional design gain of the algorithm. Figure 8 depicts equation (12) in a block diagram. The information of the maximum allowed drift of the clocks is not necessary for the correct operation of the algorithm since it is effectively an automatic proportional controller.

Note that when a new value of $s$ is smaller than the previous value, the new controlled timestamp will be smaller than the previous one during a short period of time. In order to avoid this it is necessary to adjust the real timestamp by increasing it before the multiplication by the new smaller value of $s$ is performed. Assume the value of $s$ is updated from $s_{old}$ to a new smaller value $s_{new}$ just after receiving a new beacon, then the time span that needs to elapse before the controlled timestamp shows a greater than or equal value to the previous one is given by

$$T_{interval} \geq \frac{controlled\ \ timestamp}{s_{new}} - \frac{controlled\ \ timestamp}{s_{old}} \tag{13}$$

Where the *controlled timestamp* is the value of the timestamp in the controlled clock when $s_{new}$ is computed. The first ratio in (13) is the value that the real timestamp must have in order to obtain the same controlled timestamp after being multiplied by $s_{new}$, and the second ratio is the value of the real timestamp right before the change to $s_{new}$. Therefore, equation (13) is the time that has to elapse before the controlled timestamp reaches its original value. Using equation (12) (see also Figure 8) we have

$$s_{new} = s_{old} + \frac{K_p\left(rx\_timestamp - controlled\ \ timestamp\right)}{controlled\ \ timestamp} \tag{14}$$

Substituting (14) in (13) and after some algebra we obtain

$$T_{interval} \geq \frac{-K_p\left(rx\_timestamp - controlled\ \ timestamp\right)}{s_{new}s_{old}} \tag{15}$$

Note that $s_{new} < s_{old} \Rightarrow controlled\ timestamp > rx\_timestamp$, which implies that $T_{\text{interval}}$ in (15) is positive. Also, as will be shown in Section 6, $s_{new} \approx s_{old} \approx 1$, therefore equation (15) can be simplified to

$$T_{interval} \geq -K_p\left(rx\_timestamp - controlled\ \ timestamp\right) \tag{16}$$

The time during which the controlled timestamp goes backward in time is proportional to the time difference between the received timestamp and the controlled timestamp. It is shown through simulations in Section 5 and 6 that the time difference in (16) can be in the order of tens of microseconds. If the real timestamp is adjusted to $controlled\ timestamp / s_{new}$ before the real timestamp is multiplied by $s_{new}$, then no backward leaps in time will be observed. Note that the change to a new $s$ value occurs after the contention window has finished because the last beacon received successfully is taken as the reference beacon (i.e., more than one beacon can be received in the same contention window due to beacons received in error).

In case the reference node fails, the other nodes will wait until their corresponding T_DELAY counter is equal to one and will contend again for beacon transmission, therefore, the re-

acquisition of synchronization is ensured since any node can be a reference node regardless of its drifting characteristics. Giving any node the possibility of being a reference node means that we have increased the probability of successful (*and useful*) beacon transmission from being approximately equal to $P_{given}$ in TSF, to $P_{any}$ in the proposed algorithm. This is a considerable improvement, however more improvement is possible that makes this algorithm more scalable. The fully distributive characteristic of the algorithm ensures that not all nodes need to contend for beacon transmission. In order to improve the algorithm we require all nodes to have a *rough* estimation of the number of nodes in their transmission range. Based on the number of neighbors, a node will compute and periodically update a permission probability value $P_{per}$ which is used to determine whether the node is going to transmit its beacon or not in every TBTT. Equation (17) is a possible mapping between $P_{per}$ and the estimated number of nodes $\hat{N}$.

$$P_{per} = \begin{cases} \dfrac{K}{\hat{N}} \; ; \; \hat{N} > K \\ 1 \quad ; \; 0 < \hat{N} < K - 1 \end{cases} \tag{17}$$

Figure 9 shows equation (17) as solid line with $K = 40$. Since only a rough estimation of the number of neighbors is required, we can approximate (17) as the step function shown as dotted line in Figure 9. The performance of this additional mechanism is evaluated in Section 5. The latter procedure represented by equation (17) only suggests a possible improvement to the original algorithm, but it is not necessary for its correct operation.

It is important to note that the proposed algorithm described in this section is practically memoryless. Each time a node reaches one in its T_DELAY counter, it resets its parameter ***s***, stores the controlled timestamp in the real clock's register, and contends again. This is done in order to avoid large differences between the real and controlled timestamps in each node, which could cause instability. If more memory is incorporated into the approach through the increase in the resetting period of ***s*** (i.e., setting ***s*** = 1 less often), a better performance is achieved as will be shown through simulations in Section 5. However, the memoryless approach is more robust to small disturbances in the value of ***s*** (i.e., when the real and controlled timestamps differ substantially, a small deviation of ***s*** from the desired value can result in a large difference in the controlled timestamp).

## 5. Performance Evaluation: WLAN case

We used Matlab for the computer simulations. The general parameters used in the simulations are shown in Table 2. A simplified flow diagram of the code can be found in the appendix. Where not specified differently the following values were used: $K_p = 1$, the number of nodes in the network are 80, and 150, T_DELAY=10, and beacon error rate = 1%. Throughout our simulations we assume the clocks drift with respect to real-time according to equation (6) with maximum drifts values given in Table 2, and assuming negligible values for $D$, $\xi(t)$, and $\Theta(0)$. Other values for the maximum drift of the clocks are possible, however we assume in our algorithm reasonably accurate clocks as the ones used in quality transceivers or computers (e.g., Max. Drift = [10, 100ppm]). Less accurate clocks will result in larger inter-node time difference. We added in some cases a 1% probability of loosing beacons per node additional to the simulation of beacon collisions; this emulates beacons received in error due to wireless medium

impairments and interference. A more sophisticated model for beacons in error will only increase the complexity of the simulation without adding new insights to our discussion.

| *ABeaconPeriod* | 0.1*secs* |
|---|---|
| Beacon error rate | 1% |
| Max. clock drifts | ±25 ppm |
| Time for Beacon transmission | 550$\mu$*secs* |
| Total Real-time | 30 *min*. |

**Table 2. General simulation parameters**

Some results for the TSF were already presented in Figures 4, 5, and 6. The results showed are for up to 20 nodes since the network becomes a-synchronous for a greater number of nodes (i.e., time difference greater than 1000$\mu$*sec*). Note that the results in Figures 5 and 6 are for the worst case scenario for TSF since there is only a single node with the fastest clock (+25ppm) that has to contend with the other clocks in the network with completely opposite drifts (-25ppm). In practice this might or might not be the case, but as was previously mentioned, the drift of the clocks is an unknown parameter. Our algorithm adjusts automatically the controlled variable (i.e., controlled timestamp) without knowledge of the maximum drift of the clocks. Figure 10 shows simulation results of $P_{any}$ for the FHSS and the DSSS PHY versions of IEEE 802.11. The results for FHSS show a worse performance due to the fact that the contention window is smaller (see Table 1 for the additional parameter values of FHSS and DSSS).

Figure 11 shows a comparison of $P_{any}$ using the proposed algorithm and the TSF. The result shows a higher probability for the proposed algorithm in all cases. The reason is that the instantaneous value of the T_DELAY counter is not always equal for each node since not all nodes receive the beacons exactly on the same TBTT from its reference node due to wireless medium losses. This effectively spreads the beacon transmissions in time and reduces the beacon collision probability. Furthermore, in the proposed algorithm the nodes do not contend in every TBTT as in the TSF, which greatly reduces energy consumption and overhead. The beacon error rate is 0% for both algorithms in Figure 11 therefore the larger value of $P_{any}$ for the proposed algorithm is due exclusively to the latter point. The TSF in Figure 11 shows smaller probabilities than the ones in Figure 10 because in the latter there is only a single fastest clock and the rest of the clocks are all drifting with the same value and much slower. The result in Figure 11 however, is for drifting clocks with values uniformly distributed in the range [–25, +25ppm] (except when there are two nodes only, in which case one node drifts at +25ppm and the other one at –25ppm). When there is a single fastest clock that is much faster than the rest of the clocks (the case in Figure 10) its contention window is going to start before any other node's contention window, making its beacon transmission successful if it chooses a small slot to transmit. When the clocks drift more uniformly however, the opportunity for the fastest node to transmit its beacon successfully is not as high.

Figure 12 shows the c.d.f of the maximum time deviation observed with the proposed algorithm when there is a single fastest node drifting at +25ppm and the rest of the nodes drift at –25ppm. The results are shown for FHSS and DSSS parameters with $P_{per}$ = 1 in all cases. The maximum time deviation observed for DSSS is 60$\mu$*secs* and for FHSS is 413$\mu$*secs* for 150 nodes with 98.6% of the values below 240$\mu$*secs*. DSSS with 150 nodes shows slightly better performance than FHSS for 80 nodes. Also note in Figure 11 that 80 nodes in FHSS have less chance of sending beacons successfully than 150 nodes in DSSS. While the TSF is a-synchronous when

more than 20 nodes are in the network, the new algorithm still shows good performance for 150 nodes. The maximum time deviation observed when 80 nodes are in the network is even smaller for the new algorithm compared to the TSF with 2 nodes. Figure 13 shows a sample of the difference between the controlled timestamps of the fastest node and one of the other nodes during the first 10 seconds of real-time in a network of 150 nodes. The peaks in Figure 13 are due to the beacon collisions and/or beacons received in error (i.e., 1% beacon error rate is assumed). The collision of messages is inevitable if the beacon messages of the IEEE 802.11 standard are used to exchange the time information. Other mutual network synchronization algorithms proposed in the literature make use of pulses that are used to correct the phase deviation of the geographically separated oscillators in every clock of the network [15]. These methods can have greater accuracy than our proposed algorithm, however, the large bandwidth required, and the incompatibility with the IEEE 802.11 standard make then unattractive for present day Wireless Ad Hoc networks. One of the main contributions of our work is the realization of a mutual network synchronization algorithm that incorporates the simplicity of the clock-sampling method used in the IEEE 802.11 standard. This is not the optimum is terms of performance, however, we have choose a balance between performance and simplicity of implementation.

Figure 14 shows a sample of the variation of the parameter $\boldsymbol{s}$ as time evolves for the fastest node. Note that during the first 5 TBTTs (0.5*secs*) node 1 has a value of $\boldsymbol{s}$ =1, this is due to the initial beacon collisions that lasted the first 5 TBTTs which resulted in a deviation of 25$\mu$*secs* with respect to node 2. The other peaks are less noticeable due to their short duration, however the effect of the second largest peak at around 62 TBTTs can also be observed as a small flat value for $\boldsymbol{s}$ = 1 in Figure 14. The continuous resetting of the parameter $\boldsymbol{s}$ is evident from the plot since the value of $\boldsymbol{s}$ returns to 1 every T_DELAY TBTTs. Note that the algorithm converges quite rapidly as it is also evident in Figure 3 at the sixth TBTT, where the time difference drops to zero. However, the continuous resetting of the parameter $\boldsymbol{s}$ makes the convergence of the algorithm noisier. As explained previously in Section 4, the reset of the parameter $\boldsymbol{s}$ plus the update of the real timer are cautionary measures used to avoid large differences between the real and controlled timestamps.

Figure 15 shows the distribution of reference nodes for the 1st node in a network of 150 nodes during 30 minutes of real-time simulation. Note that the distribution is uniform, therefore all nodes have equal chance to become the reference node. Figure 16 shows a closer look at the reference node hopping for three different nodes in a network of 10 nodes during 10 seconds of real-time. Note that when one of the nodes in question (i.e., 1, 2, or 3) is the reference, it has a value of zero, while the other two nodes have the corresponding value of the given reference node. The T_DELAY counter in Figure 16 has a maximum value of 10 (i.e., That is, T_DELAY = 10), in practice every T_DELAY counter will reach one at different points in the time scale due to beacons received in error, therefore spreading the beacon transmissions in time.

Figure 17 shows the c.d.f of the maximum time deviation when all the clocks drift with different values uniformly distributed in the range [–25, +25ppm] for 150 and 200 nodes using equation (17), for FHSS and DSSS. The clocks remain synchronous with a maximum time deviation of 39$\mu$*secs* for DSSS and 264$\mu$*secs* for FHSS with 99.97% of the values falling below 240$\mu$*secs* and 200 nodes. For 150 nodes the maximum time deviation value is 60$\mu$*secs* for FHSS. This result shows the scalability potential of the proposed algorithm.

Figure 18 shows the c.d.f of maximum time deviation with 150 nodes and all clocks drifting differently in the range [–25, +25ppm], with maximum values of 10, 40 and 60 TBTTs in every T_DELAY counter, for FHSS and $P_{per}$ =1 for all nodes. Note the considerable improvement by

using a larger T_DELAY, however with a larger T_DELAY the reference nodes hop less frequently and the maximum time deviation will be larger in case the reference node fails. By indefinitely increasing T_DELAY we are effectively converging to the same performance of a master-slave network synchronization approach. In Figure 18 the resetting of the parameter ***s*** is done less frequently when T_DELAY is increased therefore the synchronization approach does not need to re-learn about the clock differences as often. It is possible however, to hop faster the reference nodes and yet introduce more memory into the algorithm by not resetting the value of the parameter ***s*** every time the T_DELAY counter reaches one. Figure 19 shows the c.d.f of maximum time deviation when the parameter ***s*** is never reset with 150, and 200 nodes; all clocks drifting differently in the range [–25, +25ppm], and T_DELAY = 10 TBTTs. The result shows a considerable improvement of the performance even without the mechanism represented by equation (17).

The results presented so far are for the WLAN scenario (i.e., all nodes are able to hear each other transmissions). The potential for the proposed algorithm in terms of scalability, reduced extra overhead, energy consumption, and synchronization accuracy makes it an excellent solution for the improvement of the IEEE 802.11 TSF. Note that we have made use of IEEE 802.11 standard parameters and beacon messages to achieve the results presented with the proposed algorithm. Other advantages of the algorithm are the independence from the PHY layer used, the fact that control of time is not exerted over the real clocks through, for instance, the use of Phase-Locked-Loops as in previous studies, and the resilience to node failure thanks to the reference node hopping feature. In the next sections we present a discussion, and performance evaluation of the proposed algorithm in a multi-hop Ad Hoc network.

## 6. Clock-sampling Mutual Network Synchronization: Multi-hop case

In a multi-hop network it is important to spread the synchronization information as quickly as possible among all the nodes in the network, this is because the network latency is higher (i.e., it takes longer to send information from one extreme of the network to the other). A node that received a beacon successfully can not afford to wait too long before sending its newly acquired synchronization information. Therefore, the T_DELAY parameter should be smaller in a multi-hop network in comparison to a single-hop network of the same size, where T_DELAY is not a critical parameter (i.e., synchronization still works well with large values of T_DELAY for a single-hop network as observed in Section 5). The original algorithm is not optimum for a multi-hop network since as soon as a node receives a beacon successfully it will wait for T_DELAY TBTTs before trying to transmit its own timestamp, delaying unnecessarily the spreading of the synchronization information over the network. In order to avoid this it is important not to have a large value for the maximum value of the T_DELAY counter. Additional to this, we add the following mechanism to improve the efficiency of the original algorithm:

1. Define a $P_p$ parameter per node. This parameter is used to determine whether the node is going to transmit its beacon once the T_DELAY counter reaches 1.

   $min\_permission \le P_p \le 1$.

   Where $min\_permission > 0$.

2. At initialization, a node will set $P_p = 1$.

3. If a node receives a beacon successfully, it will increase its $P_p$ value by an amount equal to $\alpha$. If a node does not receive a beacon in the current TBTT, it will decrement its $P_p$ by $\beta$.

   If a beacon is received successfully, then

$$P_p = \min\{1, P_p + \alpha\} \tag{18}$$

   Otherwise,

$$P_p = \max\{min\_permission, P_p - \beta\} \tag{19}$$

The next time the T_DELAY counter reaches 1, the node will determine whether it is allowed to transmit its beacon by comparing its new value of $P_p$ with a uniform random number *rand*, $0 \le rand \le 1$. If $rand < P_p$, then the node is allowed to transmit, otherwise it is not allowed. Note that being allowed to transmit does not necessarily mean the node is actually going to successfully transmit a beacon since it is contending with other nodes in the network. In order to increase the speed of response of the additional mechanism and avoid beacon starvation when the number of nodes is large (i.e., most of the nodes having a low value of $P_p$), we choose $\alpha > \beta$. Therefore, the additional mechanism will promptly increase the probability of beacon transmission when a successful beacon is detected, and it will be more conservative otherwise.

## 7. Performance Evaluation: Multi-hop case

The additional parameters used in the multi-hop simulations are shown in Table 3. Throughout the simulations we assume the clocks drift with respect to real-time linearly with maximum drifts values equal to ±25ppm.

| | |
|---|---|
| Transmission range | 150m |
| $\alpha$, $\beta$ | 0.4, 0.1 |
| Detection range | 300m |

**Table 3. Additional simulation parameters for the multi-hop case**

In order to evaluate the performance of the new algorithm, we use a grid topology as shown in Figure 20. In the particular case of Figure 20 we are showing a 4x4 network; for a specific simulation we will refer to this topology by its size (e.g., 2x2, 5x5, 10x10 etc) which will implicitly refer to a grid topology. The ordering of the nodes will follow the same pattern as the one in Figure 20 (i.e., node 1 at lower left corner and incrementing towards the up-right direction). The vertical and horizontal distances are equal to 150m. The nodes do not move, however, in order to simulate some adverse conditions, we are going to shut down some of the nodes temporarily in order to observe the response of the network synchronization algorithm. The links joining the nodes in Figure 20 correspond to our assumption that the transmission range is equal to 150m. That is, a node is able to correctly decode a beacon if it is within transmission range of the transmitter. A beacon will not be

decoded successfully however, if there is a collision, or the beacon is received in error (i.e., a 1% beacon error rate is added in the simulations) even if it is within transmission range.

In a multi-hop network we have more detrimental effects due to the hidden and exposed node problems, this is also taken into account in the simulation. The hidden and exposed node problems are also present in single-hop networks, but to a lesser degree and due to wireless medium impairments; the topology of a multi-hop network increases the detrimental effects of these two problems over the already present wireless medium impairments. An exposed node will defer its beacon transmission if another node within its detection range (<300m) is transmitting; a hidden node will cause a collision if it is within transmission range of a receiving node, but out of the detection range (>300m) of the corresponding transmitting node.

We compare the new algorithm with the TSF of IEEE 802.11 with contention window and beacon parameters for frequency hoping spread spectrum (FHSS) since this is the worst case scenario. The FHSS has a smaller beacon contention window in comparison with the direct sequence spread spectrum (DSSS) version of the standard. Figure 21 shows the cumulative distribution function (c.d.f) of the maximum time difference between the nodes of a 5x5 network using TSF. Node 1 has the fastest clock with a drift $D_1$ = +25ppm, and the rest of the nodes all drift at $D_i$ = -25ppm, $i = 2, 3, ..., 25$. The simulation was run for 30*min* of real-time and a sample of the time difference between nodes 1,7, and 25 is shown in Figures 22 and 23.

The differences between the nodes never fall below 5$\mu$*secs* in Figure 23. This is because the differences are taken in discrete steps every *aBeaconPeriod* (see Table 1), therefore the clock in node 1 will always be at least 5$\mu$*secs* ahead of any other clock in the network. Note also that, as expected, the time difference between node 1 and node 25 is larger (average = 310$\mu$*secs*) than the one between nodes 1 and 7 (average = 10$\mu$*secs*) due to their larger geographical separation. TSF requires the fastest clock information to be relayed over the whole network through the intermediate nodes in a continuous way because the rest of the nodes never learn what is the time difference (i.e., the slower clocks will deviate if the information is not refreshed continuously). By introducing a learning mechanism into our algorithm we show that it is possible to refresh the synchronization information less often, and in fact use the refreshment exclusively to adapt to topology variations and dynamics in the network. Even in an extreme situation of static network (i.e., a network without mobility, no wireless medium impairments, and clocks that always drift with a constant value), TSF still requires continuos refreshing to maintain synchronization, this is also true for a single-hop network. Also note that the result in Figure 20 is optimistic since link delay was assumed negligible. Link delay in TSF is more detrimental than in the proposed algorithm since the timing information is extracted from a single point in space. It takes time to deliver the timing information of the fastest clock to the clocks on the opposite boundaries of the network. In a mutual synchronization approach however, the timing information is distributed in space, therefore only the inter-node link delay has to be considered, which is negligible considering the range of terrestrial WLANs (less than 1$\mu$*sec*).

Figure 24 shows the c.d.f of the maximum time difference when the clocks of the nodes drift with different values chosen from a uniform distribution in the range [-25, 25] ppm (i.e., each clock always drifts with the same value but different to other clocks' drifts). We assume in our simulations that a clock will drift with the same value *D* throughout the entire simulation; only aging and sudden atmospheric variations will change this value substantially. The rest of the parameters are equal to the ones used for Figure 21. Note that Figure 24 shows a better result than Figure 21 (smaller time difference). This is because the result in Figure 21 is a worst case scenario for TSF since there is only a single fastest clock that runs much faster than the rest of the

clocks in the network, whereas in Figure 24 the drift differences among the clocks are not as drastic. The curves shown in Figures 21-24 can serve as basis of comparison between the TSF and the proposed algorithm (to be presented next).

Figure 25 shows the maximum time difference among nodes 1, 7, and 25 in a 5x5 network using the proposed algorithm. The clocks of the nodes drift with different values chosen from a uniform distribution in the range ±25ppm, T_DELAY = 10, $Kp$ = 0.5, and the permission algorithm of equations (18) and (19) is not used. Since the algorithm learns after an initial transition of approximately 100 seconds what is the required parameter $\boldsymbol{s}$ in all the clocks, future adjustments are rarely needed. Only in a highly dynamic network will the algorithm try to adjust its parameters to cope with variations more frequently (limited by the *aBeaconPeriod* value). Note that Figure 25 only shows values for the first 200 seconds of real-time and the final time variation remains approximately around $\pm 3\mu secs$. This is a considerable improvement in terms of steady-state behavior with respect to the TSF (see Figures 4-7). Note that the worst case scenario for TSF (Figures 4-7) is actually quite favorable for the new algorithm since most of the clocks are already synchronous to one another at –25ppm. Figure 26 shows the same scenario of Figure 8 with the permission algorithm of equations (18) and (19) enabled. In this case the time of convergence, defined as the time at which the maximum time difference between two given nodes falls and remains below $10\mu secs$, is reduced from approximately 100 seconds down to 40 seconds.

Figure 27 shows the same scenario of Figure 26, but a uniform random error equal to $\pm 100\mu secs$ is added to the initial clock value in every clock instead of starting with all clocks synchronized. In this case the time of convergence is increased, but the algorithm still shows good convergence properties. This models more accurately a real scenario since it is unlikely the clocks will be exactly synchronized when first joining the network (usually a coarse synchronization is achieved at initialization).

Figure 28 shows the plot of the maximum time difference of the nodes in a 10x10 network using the new algorithm with permissions enabled for the first 500 seconds of simulation. The plot in Figure 28 is the maximum time difference observed in the network regardless of the node-pairs considered. This result shows promising scalability performance for the proposed synchronization algorithm. The time of convergence is approximately 250 seconds when T_DELAY = 10 and $Kp$ = 0.5 is used, 147 seconds for T_DELAY = 2 and $Kp$ = 0.5, and 80 seconds for T_DELAY = 2 and $Kp$ = 0.8. Decreasing T_DELAY improves the convergence time; however, it increases the overhead since beacons are sent more frequently. The increase in the proportional gain $Kp$ makes the algorithm faster as well, however, $Kp$ can not be increased indefinitely since the system can become unstable. Note that the result in Figure 11 is a worst case scenario since all nodes start to exchange beacons at the same time without any previous knowledge. In a more realistic scenario, the nodes will scan for beacons before joining the network, and the transition will be smoother.

We simulated the failure of nodes 5, 9, 13, 17, and 21 for 200 seconds in a 5x5 network in order to show the response of the new algorithm to network dynamics. Figure 29 shows the time difference of nodes 1, 7, 19 and 25. The former nodes failed at 200 seconds of real-time for an interval of 200 seconds. The initial conditions are random ($\pm 100\mu secs$), and all the clocks drift with different values (in the range of ±25ppm). Note that this failure is quite drastic since it effectively divides the original network into two isolated sub-networks. Nodes 19 and 25 remain synchronous to each other, and the same occurs to nodes 1 and 7 since they are in the same sub-net, however, inevitably the two sub-networks start drifting apart from each other since they can not communicate. After 200 seconds the nodes are back and the algorithm converges to its

original state. Note that the clocks do not drift too much during this period since we assume good quality clocks, furthermore the resultant drift is upper and lower bounded by the maximum and minimum real drift of the clocks in every sub-net as will be shown later. The real drift of the clock refers to the actual drift, and not the absolute maximum drift, which is equal to ±25ppm.

We are interested in observing the drift with respect to real-time of the whole network once it has achieved synchronization. We simulated a symmetric 2x2 network in which nodes 1 and 3 drift with $D$ = +25ppm, and nodes 2 and 4 drift with $D$ = -25ppm. Figure 30 shows one sample of the difference with respect to real-time of the nodes in this network utilizing $Kp$ = 0.5 and T_DELAY = 10 during the first 100 seconds of real-time with zero initial conditions. Figure 31 shows the equivalent result with $Kp$ = 0.01 and T_DELAY = 1. By increasing the rate at which the reference node is hopped (see Section IV) and by reducing the proportional gain of the algorithm, the overall drift of the network tends to be closer to the average of the individual drifts in the network (0ppm). However, the time of convergence increases as can be observed in Figure 31. By reducing T_DELAY and $Kp$ we are effectively giving more nodes the opportunity to influence the final value of the network drift, which in turn reduces the deviation around the average drift value (0ppm). This is a crude approximation of the behavior of this complex system.

We performed 100 simulations similar to the ones of Figures 30 and 31 in order to characterize better the drift of the network after synchronization. The ensemble average was close to 0ppm (approx. –2ppm) after different trials and different values of $Kp$. The simple proportional controller used in every node consistently shows an offset on its final response. Additionally, it is important to realize that the dynamics of this system correspond to a MIMO (Multiple-Input-Multiple-Output) system which has inputs that are corrupted by random collisions and errors. Therefore, a more powerful control strategy is needed for the purpose of synchronizing with real time (the existence of such strategy is unknown to us). We simplified the system even further and performed 100 simulations with a 2x1 network (2 nodes) drifting at equal and opposite drift values (+25ppm and –25ppm). In this simpler case the proportional control law does achieve an ensemble average closer to the average of the individual drifts. Figure 32 shows 200 sample network drifts (100 per node) obtained when using $Kp$ = 0.03 (also shown a case when node 1 and node 2 have unequal drifts equal to +25ppm and -5ppm respectively). Note that even a small drift such as -2ppm might not be attainable in practice because the final drift of the network will also be determined by the link latencies, which we assumed to be 0$\mu secs$ in our simulation due to the relatively short distances involved. However, recall that our goal is not to synchronize with real-time (build an accurate clock).

Figure 33 shows the parameter $\boldsymbol{s}$ corresponding to Figures 30 and 31. As expected, nodes 1 and 4 converge to a value of $\boldsymbol{s}$ smaller than 1 and nodes 2 and 3 converge to a value higher than 1. When using a larger value for $Kp$ (Figure 30) the convergence is much faster and the influence of the nodes that transmit first are larger in determining the final drift of the network.

Finally, we performed 100 simulations of the new algorithm in a 5x5 network and recorded the time of convergence in *secs* (this is done for the time difference of nodes 1-2, 1-25, and 7-19), and maximum time difference observed (in $\mu secs$) between any pair of nodes. Table 4 shows the ensemble average and standard deviation of these two performance parameters with and without permission as defined in equations (18) and (19).

| | Permission OFF | | Permission ON | |
|---|---|---|---|---|
| | Std. Dev | Aver. | Std. Dev | Aver. |
| Max. time difference | 203 | 933 | 41 | 431 |
| Time of convergence 1-2 | 37 | 85 | 18 | 44 |
| Time of convergence 7-19 | 40 | 114 | 23 | 65 |
| Time of convergence 1-25 | 42 | 122 | 35 | 75 |

**Table 4. Performance of the new algorithm in a 5x5 network for 100 simulations of 200 seconds real-time each**

The results show an improvement of roughly 50% by using the permission algorithm, both in terms of maximum (or peak) time difference, and time of convergence. This is due to the combined effects of beacon contention reduction and the more effective way in which synchronization is spread throughout the network (nodes that received beacons successfully will have higher probability of beacon transmission since they have captured more recent information about the clock's drift in the network). The algorithm without the permission mechanism still achieves the same steady state performance, but its transient response has poorer qualities.

Increasing the aBeaconPeriod time can also reduce the overhead of sending beacons further. Once a network has achieved synchronization through the initial learning transition, there is no need to send beacons frequently. If the network dynamics do not change often through, for instance, nodes joining and leaving the network frequently, then the beacon period can be increased without a major impact. Note that this is implicitly shown through the results using equations (18) and (19), and through the T_DELAY counter, since not all nodes are required to send their beacons at every TBTT

.

## 8. Conclusion

We have shown through extensive computer simulations the feasibility to achieve mutual network synchronization in a WLAN or in a multi-hop wireless Ad Hoc network by utilizing the beacon messages in the IEEE 802.11 standard. Different from the IEEE TSF algorithm, which requires continuous refresh of the synchronization in every node, the new algorithm learns after a transition period from all the clocks in the network, and maintains very accurate synchronization accuracy afterwards (approx. less than $\pm 10 \mu secs$). Node failure is emulated and the results for the new algorithm show good performance. The new algorithm requires the connectivity of the network at least temporarily in order to synchronize all the clocks in a given location. One of the most attractive features of the new algorithm is its compatibility with the beacon messages used in the IEEE 802.11 standard and its independence from the physical layer since no direct control of clocks is performed. A more suitable mechanism is added to the algorithm represented by equations (18) and (19) that better copes with the peculiarities of multi-hop communications with promising results. Note that in our simulation set-up it is assumed that a beacon is received properly if it is sent by a node within transmission range (only if collisions and wireless medium impairments do not affect its reception). However, in a more realistic scenario the combination of the signals transmitted by distant nodes (nodes farther than detection range) might lower the signal-to-interference plus noise ratio to a level that could affect the reception of the given beacon. This degrades the performance of any algorithm that requires exchanging timing messages, such as TSF. After converging to a synchronous state, the network clocks drift with an ensemble average that is close to the average of the individual clock's drift. Obtaining very accurate real-time synchronization (although not our main purpose in this work) is very difficult due to the finite resolution of the clocks, the link latencies, and the spatial distribution of the

clocks in the network. Furthermore, it is important to realize that this is a distributed MIMO control problem with noisy inputs, therefore more powerful control techniques are required in order to achieve a better result. Future work includes further computer performance evaluations with a more realistic wireless medium model, and the implementation of this algorithm in a wireless Ad Hoc network test-bed currently under development.

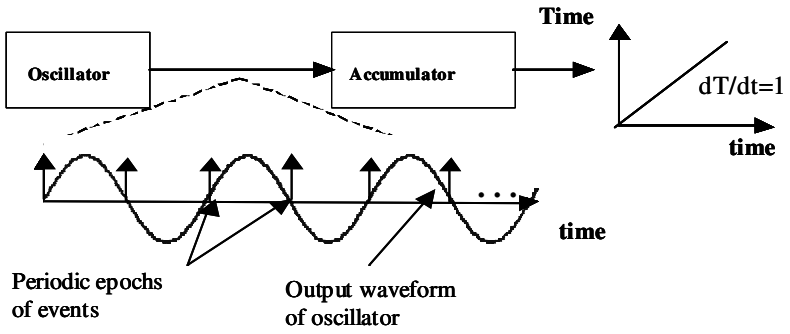


**Figure 1. A simple clock structure**

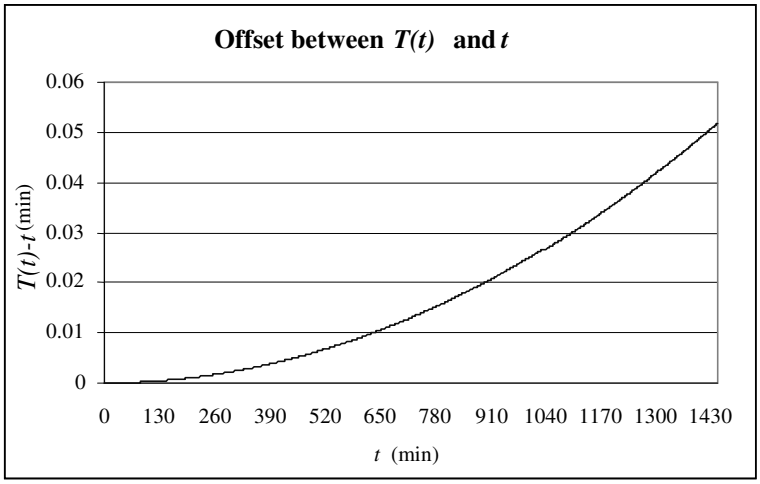


**Figure 2. Time process example**

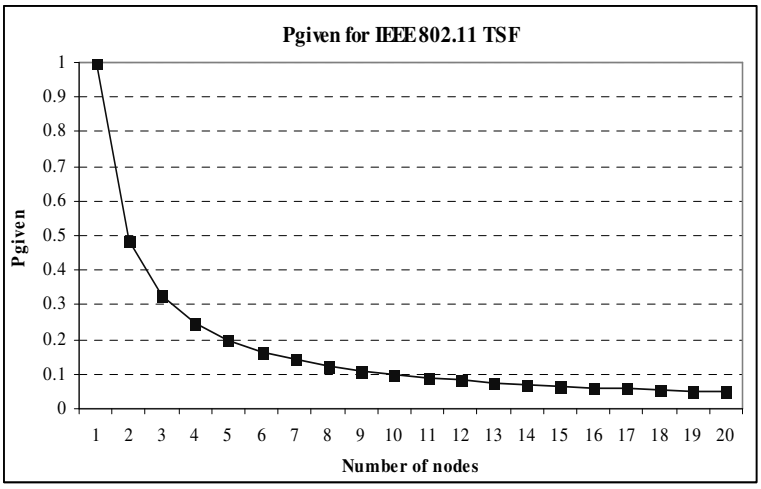


**Figure 3. $P_{given}$ in IEEE 802.11 TSF**

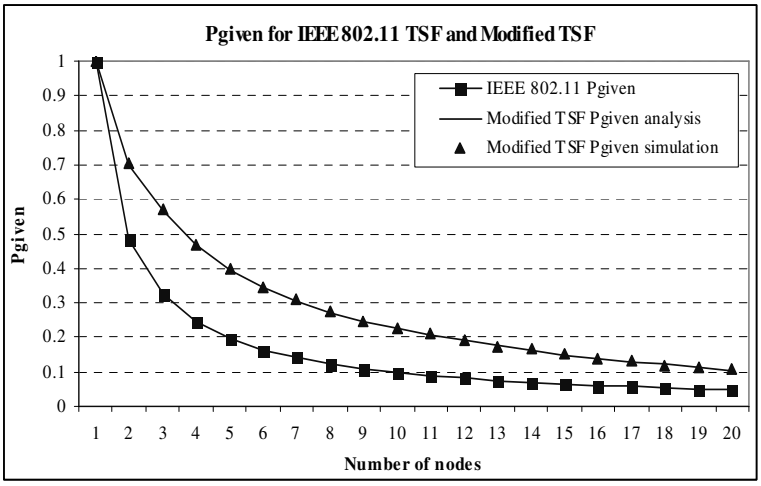


**Figure 4. $\hat{P}_{given}$ for IEEE 802.11 and modified TSF**

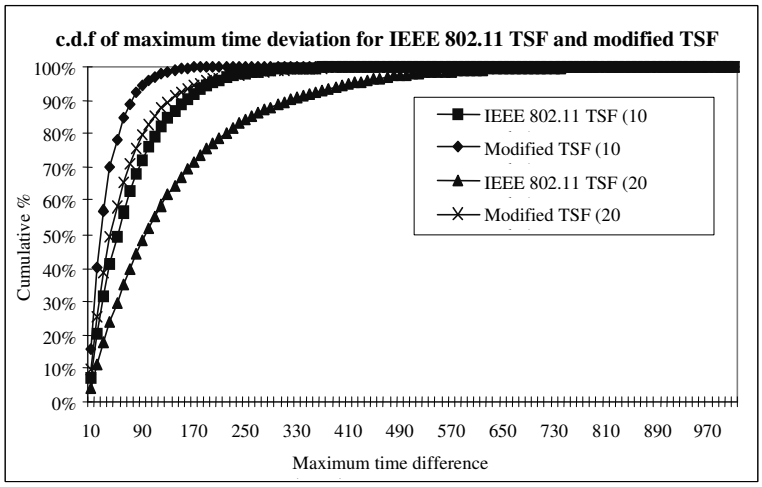


**Figure 5. IEEE 802.11 and modified TSF c.d.f of maximum time deviation**

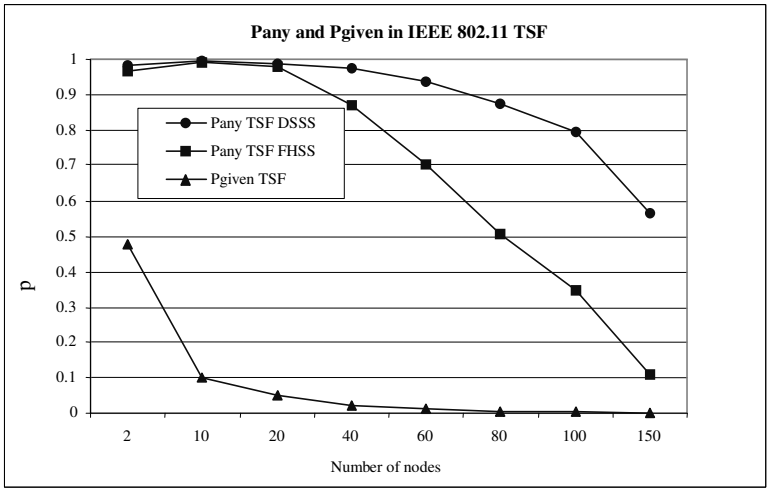


**Figure 6. $P_{any}$ and $P_{given}$ for the IEEE 802.11 TSF**

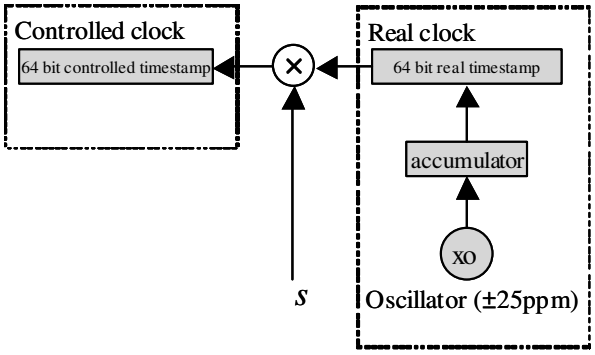


**Figure 7. Controlled and real clocks in each node**

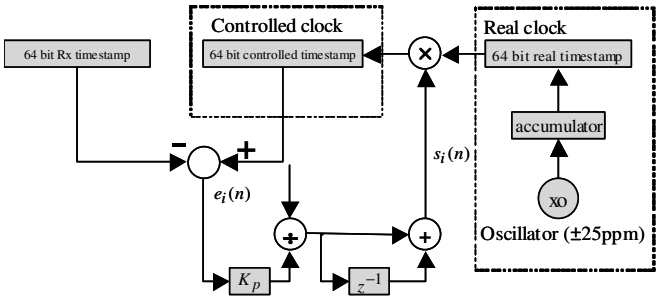


**Figure 8. New algorithm block diagram**

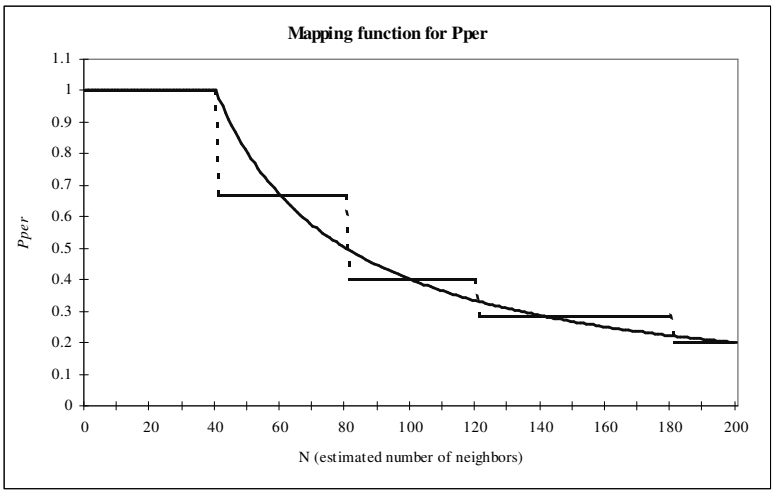


**Figure 9. Mapping between *Pper* and $\hat{N}$**

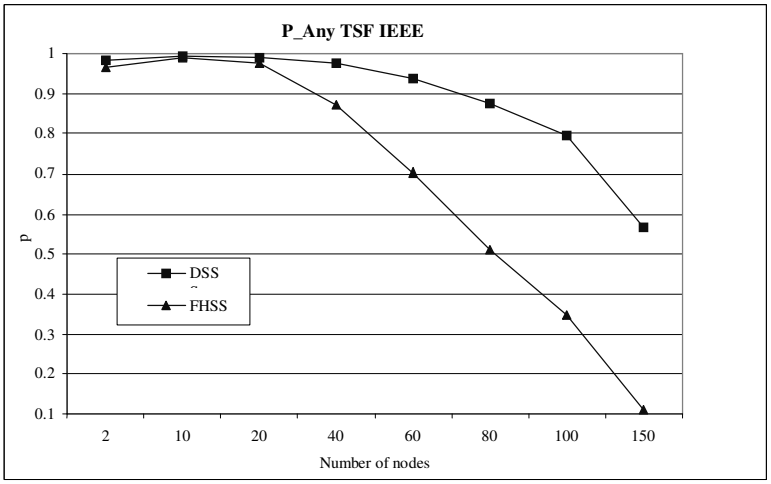


**Figure 10. $P_{any}$ for the IEEE 802.11 TSF (FHSS vs DSSS)**

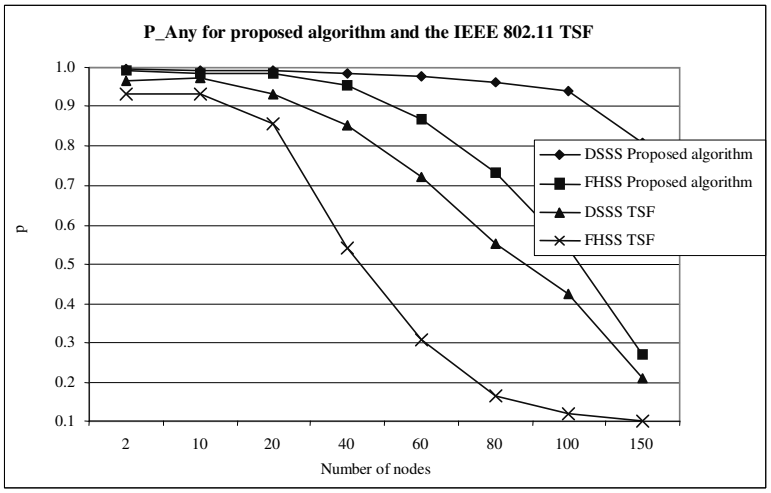


**Figure 11. $P_{any}$ for IEEE 802.11 TSF and proposed algorithm with uniformly distributed drifts**

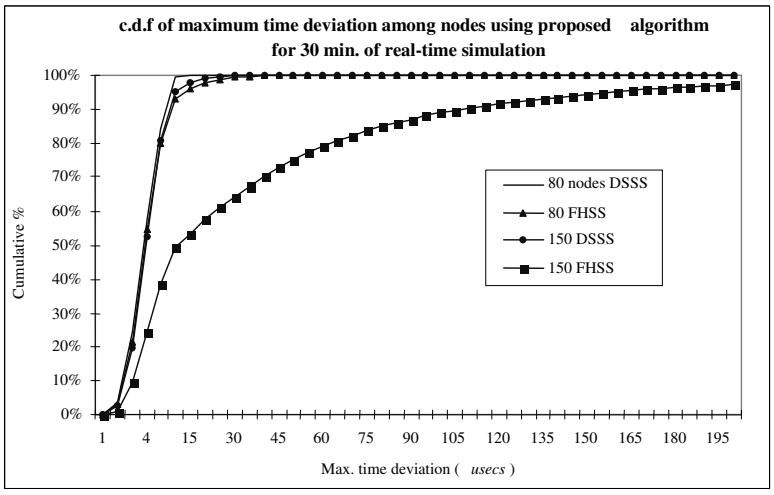


**Figure 12. c.d.f of maximum time deviation during first 30 minutes of real time with proposed algorithm**

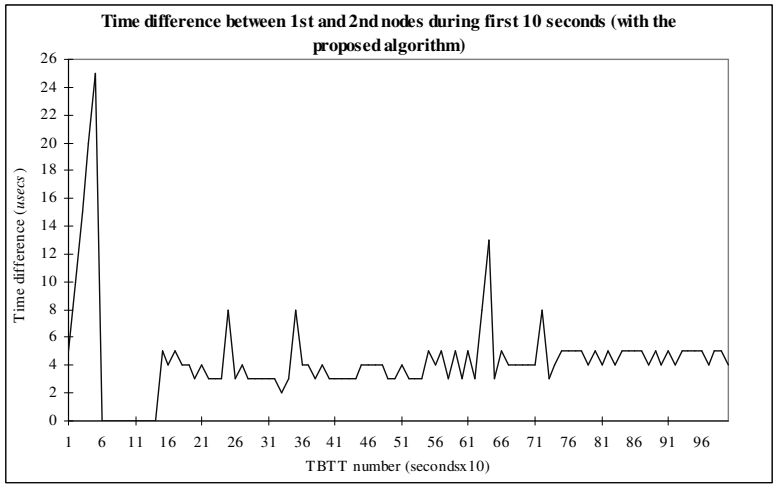


**Figure 13. Time difference between two nodes using the proposed algorithm**

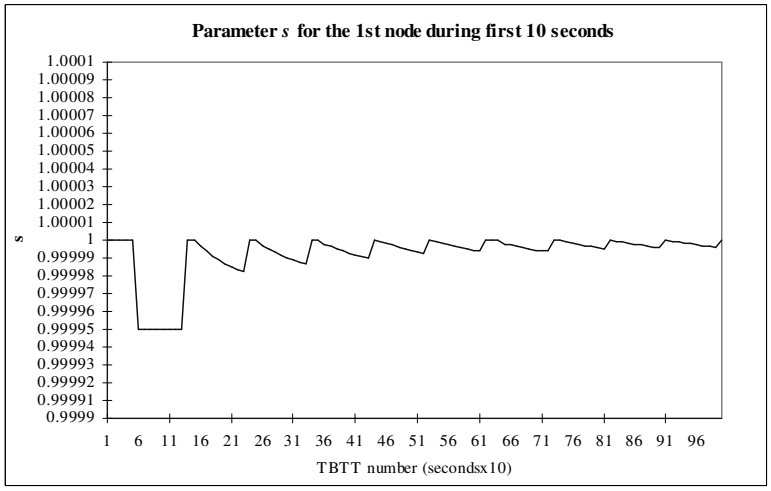


**Figure 14. Sample of the parameter s for a single node in a network of 150 nodes**

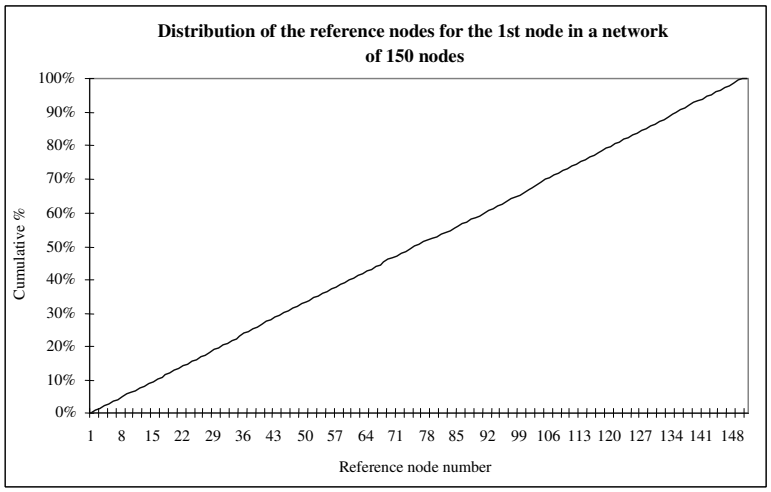


**Figure 15. Sample distribution of reference nodes of the 1st node**

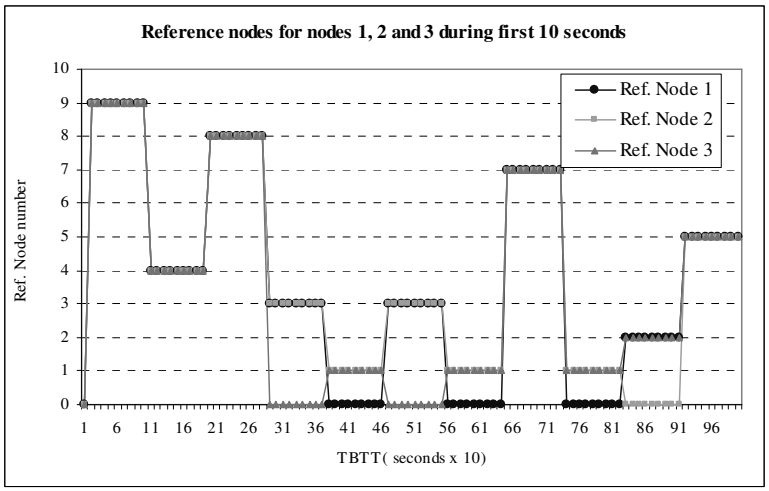


**Figure 16. Reference node hopping**

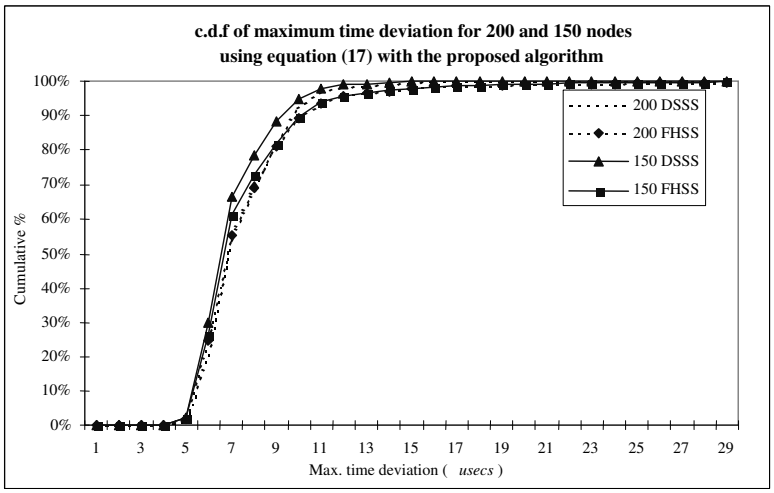


**Figure 17. c.d.f of maximu time deviation using equation (17)**

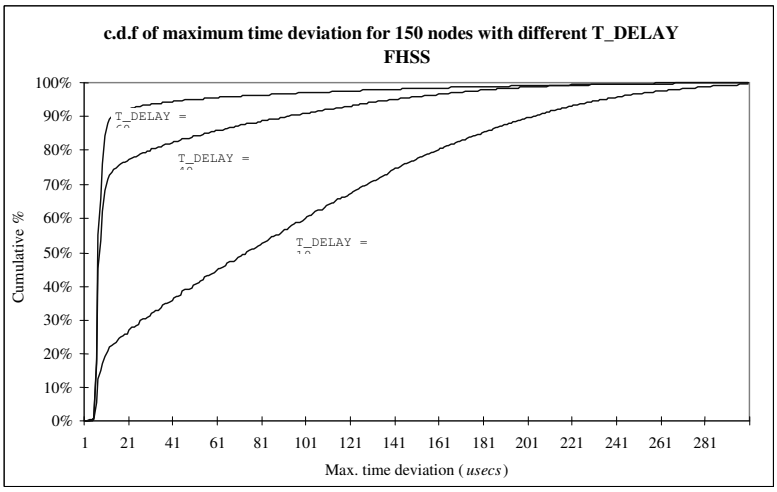


**Figure 18. c.d.f of maximum time deviation for different values of the T_DELAY parameter**

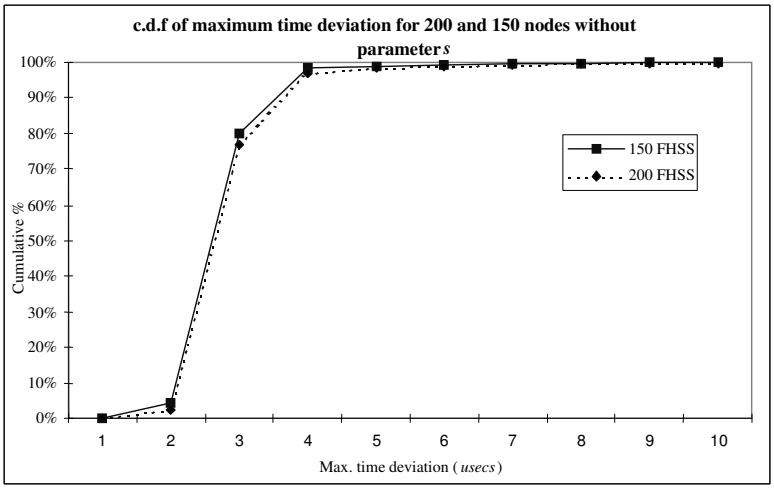


**Figure 19. c.d.f of maximum tme deviation with memory**

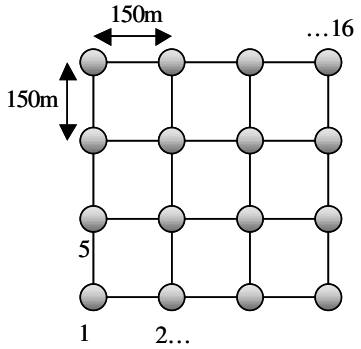


**Figure 20. Network topology (example of a 4x4 grid)**

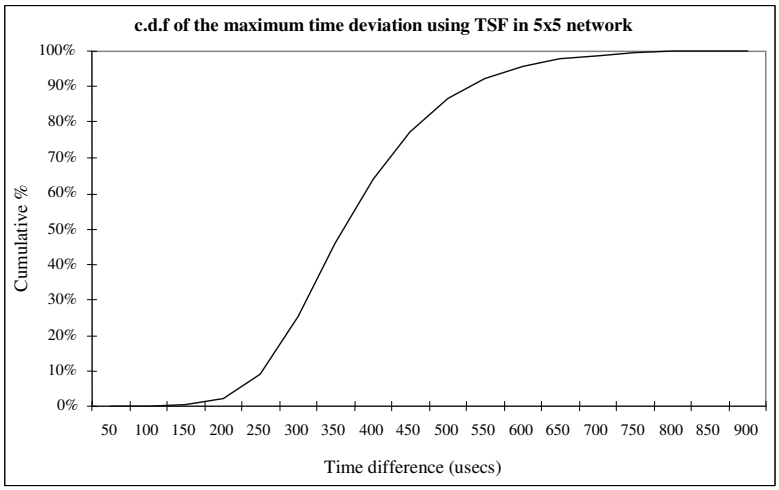


**Figure 21. c.d.f of maximum time deviation for TSF in a 5x5 grid**

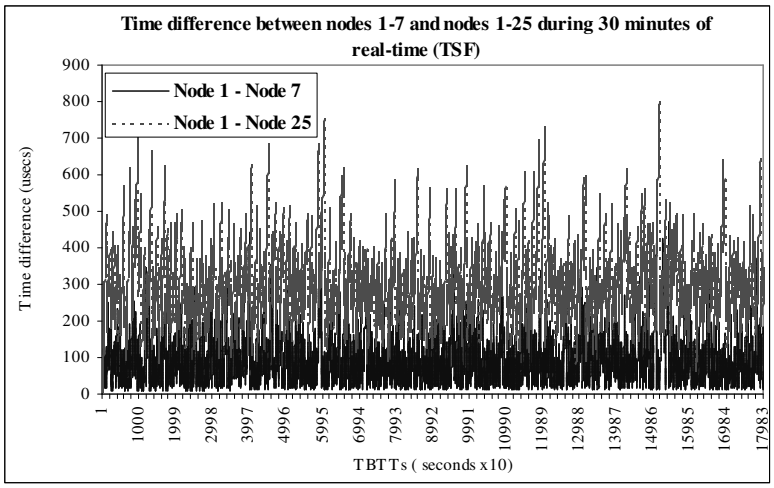


**Figure 22. Time difference of nodes 1, 7, and 25 in a 5x5 network using TSF**

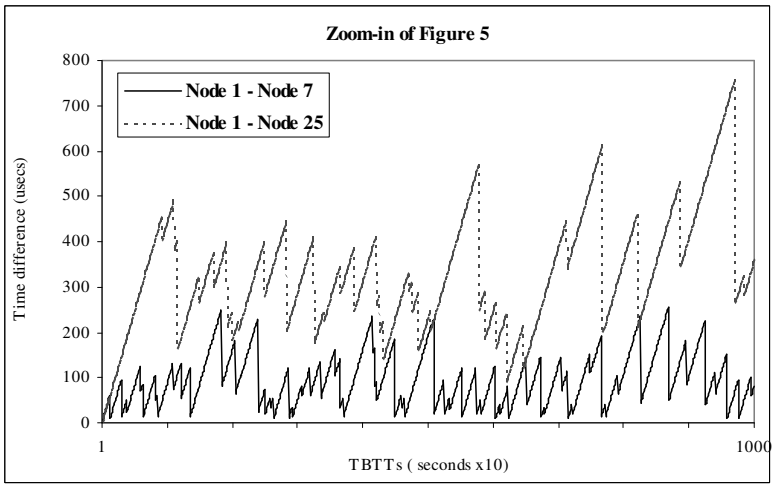


**Figure 23. Time difference of nodes 1, 7, and 25 in a 5x5 network using TSF (first 100 seconds).**

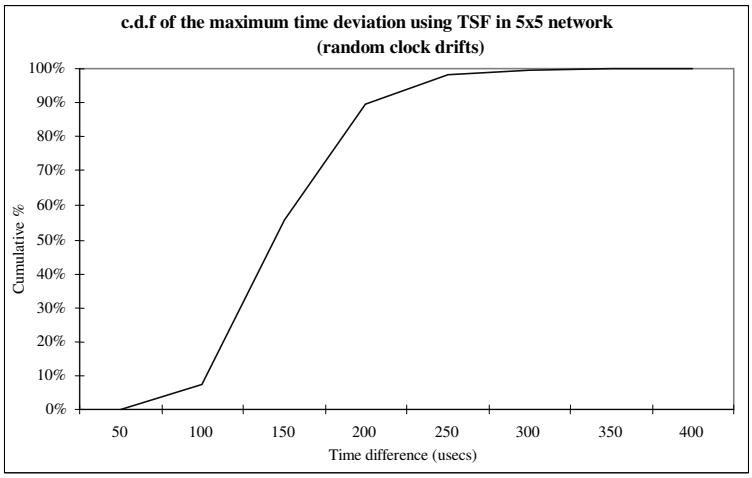


**Figure 24. c.d.f of maximum time deviation in a 5x5 network using TSF and different drifts per clock**

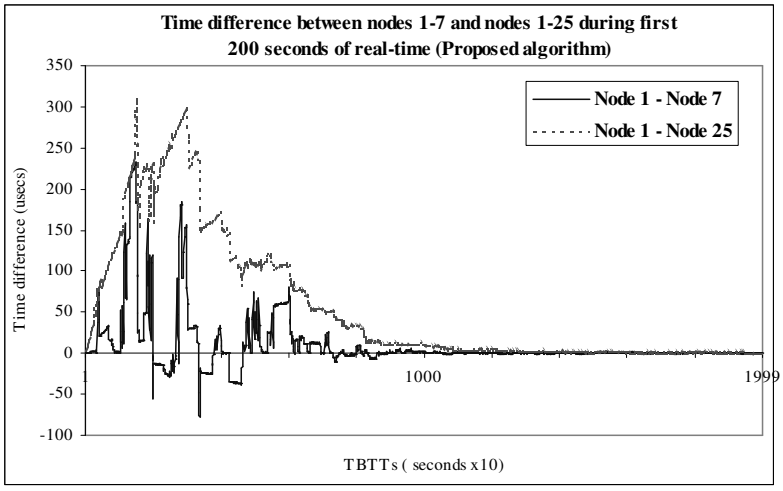


**Figure 25. Time difference among clocks of nodes 1,7, and 25 with new algorithm**

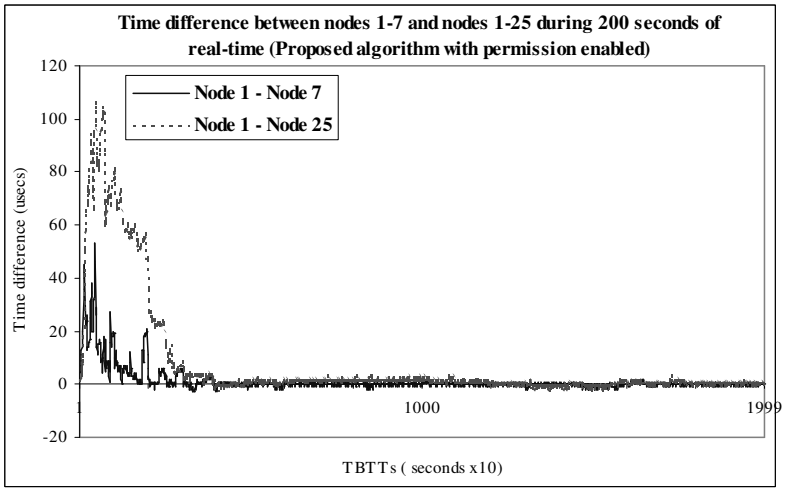


**Figure 26. Time difference among nodes 1, 7 and 25 with the permission algorithm enabled**

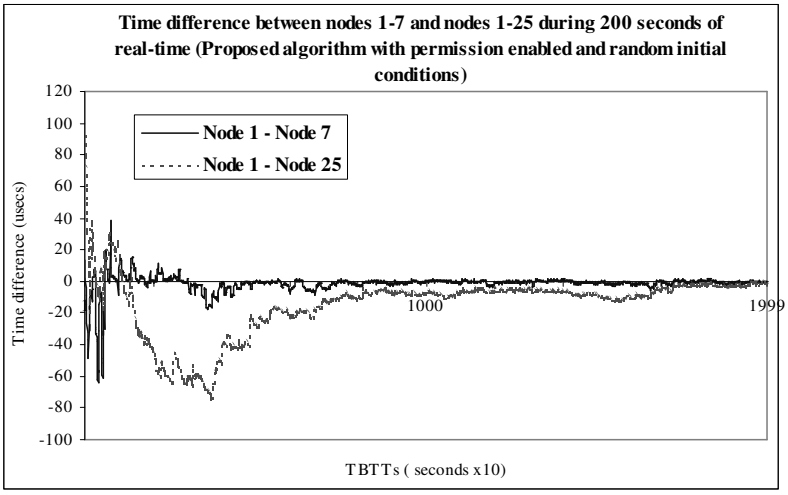


**Figure 27. Time difference among nodes 1, 7, and 25 with random initial conditions**

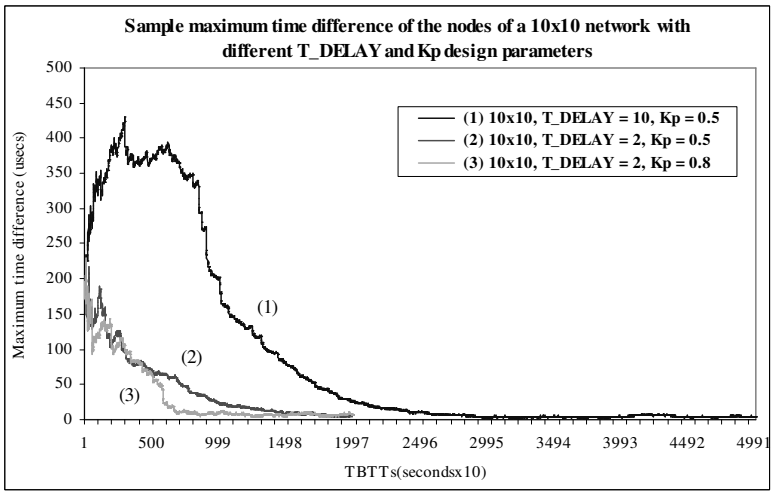


**Figure 28. Maximum time difference in a 10x10 network**

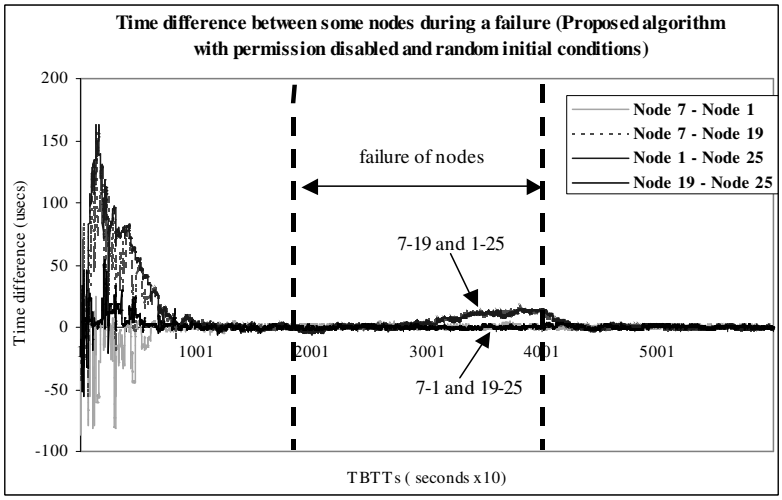


**Figure 29. Time difference during a failure.**

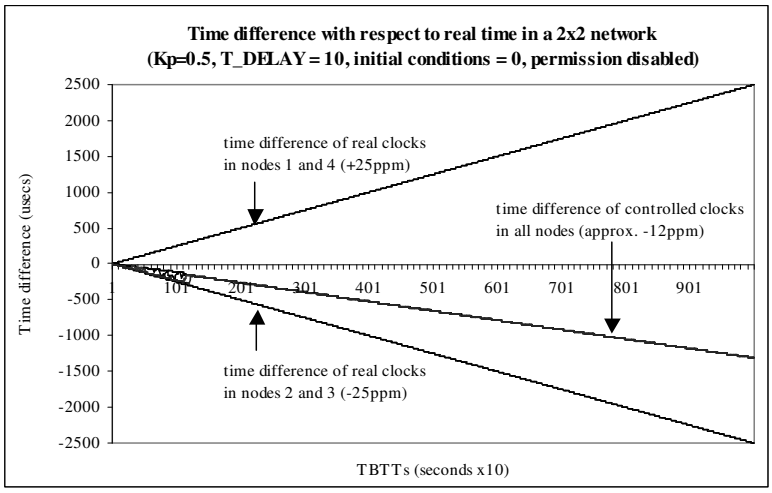


**Figure 30. Time deviation with respect to real-time (*Kp* = 0.5 and T_DELAY = 10)**

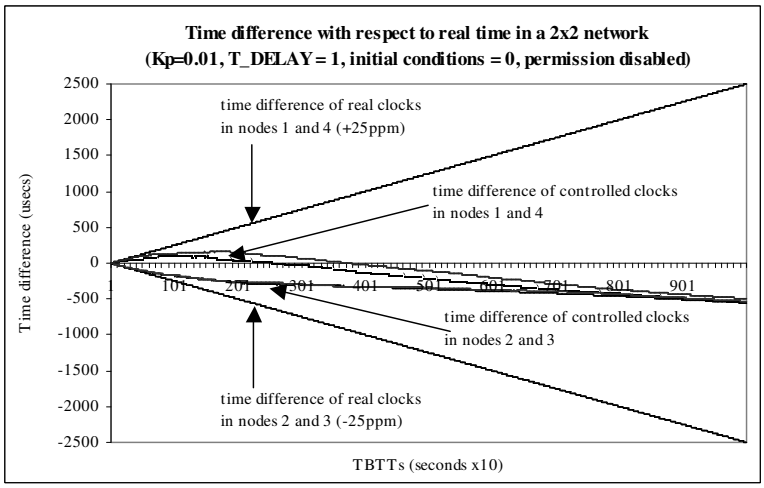


**Figure 31. Time deviation with respect to real-time (*Kp* = 0.01 and T_DELAY = 1)**

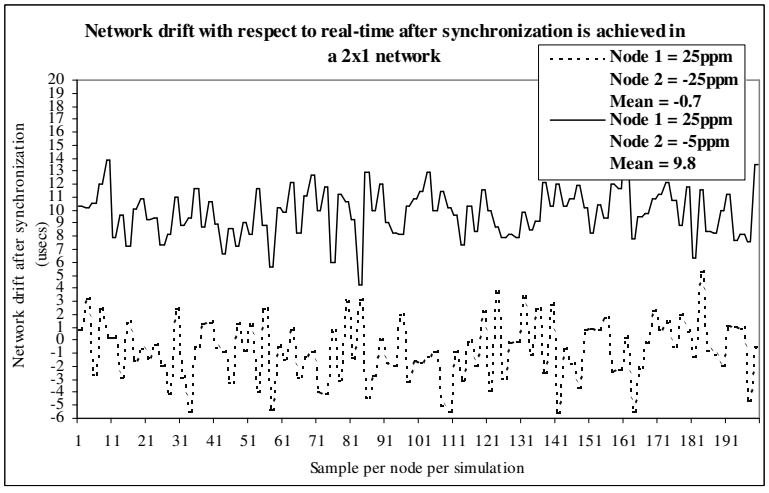


**Figure 32. Network drift with respect to real-time in a 2x1 network for 100 simulations of 100 seconds each with equal and unequal drifts.**

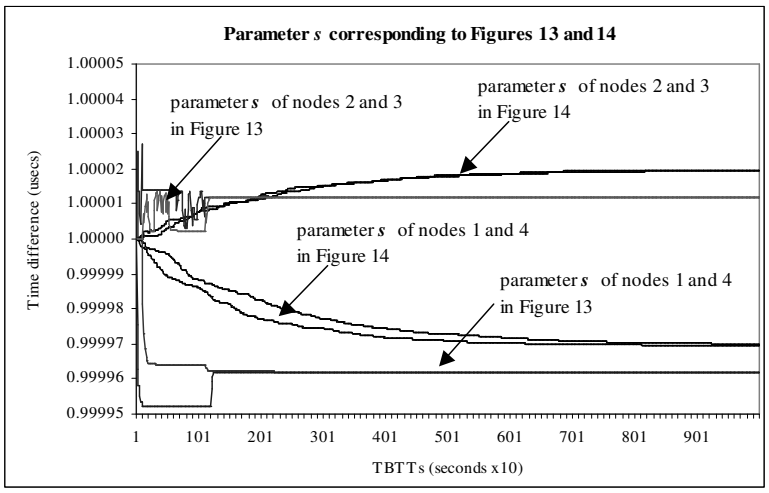


**Figure 33. Parameter *s* corresponding to Figures 30 and 31**

## References

[1] Kay Römer, " Time synchronization in Ad Hoc networks," Mobihoc. 2001, pp. 173-182

[2] J. Nelson, L. Girod, and D. Estrin, "Fine-Grained network time synchronization using reference broadcast," Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02). December 2002.

[3] IEEE Std. 802.16a Amendment to IEEE 802.16-2001 "Air Interface for Fixed Broadband Wireless Access Systems- Amendment 2: MAC modifications and Additional PHY specifications for 2-11GHz," 2003

[4] IEEE Std. 802.11. "Wireless LAN medium access control (MAC) and physical layer specification," 1999.

[5] Douglas M. Considine, Editor, "Van Nostrand's scientific encyclopedia," Eight ed. 1995.

[6] *Kartaschoff, P,* **"**Synchronization in digital communications networks," Proceedings of the IEEE ,Volume: 79 , Issue: 7 , July 1991

[7] John R. Vig, " Quartz crystal resonators and oscillators for frequency control and timing applications," U.S Army communications-electronics command. http://www.ieee-uffc.org/freqcontrol/tutorials/vig/vig_tutorial1_files/frame.htm

[8] D.W. Allan, "Clock characterization tutorial**,**" Proceedings of the 15th Annual Precise Time and Time Interval (PTTI) Applications and Planning Meeting, 1983.

[9] S.B Moon, P. Skelly, and D. Towsley, "Estimation and removal of clock skew from network delay measurements," 18th Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM '99. Volume: 1 , 21-25 March 1999, pp. 227 - 234 vol.1.

[10] W.C Lindsey, F. Ghazvinian, W. Hagmann, and K. Dessouky, “Network synchronization,” Proceedings of the IEEE, vol. 73, No. 10, October 1985

[11] L. Huang, T-H Lai, “On the scalability of IEEE 802.11 Ad Hoc networks,” Mobihoc. June 2002, pp. 173-182

[12] A. Gersho and B. J. Karafin, "Mutual synchronization of geographically separated oscillators," Bell Syst. Tech. J., vol. 45, December 1966, pp.1689-1904

[13] John G. Proakis, "Digital Communications," Mc-Graw Hill, 3rd edition, 1995

[14] Y. Akaiwa, H. Andoh, and T. Kohama, “Autonomous decentralized inter-base station synchronization for TDMA microcellular systems,” IEEE Vehicular Technology conference. May 1991, pp. 257-262

[15] E. Sourour, and M. Nakagawa, “Mutual decentralized synchronization of intervehicle communications,” IEEE Transactions on Vehicular Technology, vol. 48, No. 6, November 1999, pp. 2015-2027

[16] A. Ebner, H. Rohling, M. Lott, R. Halfmann, "Decentralized slot synchronization in highly dynamic Ad Hoc networks," www.fleetnet.de

[17] T-H Lai, and D. Zhou, " Efficient and scalable IEEE 802.11 Ad-Hoc-Mode timing synchronization function," Proc. of the 17th International Conf. on Advanced Information Networking and Applications (AINA'03). March 2003, pp. 1-6