Fault Tolerant Clock Synchronization in Distributed Network using Weighted Average

P D Sharma Rameshwar Nath Tripathi Y N Singh
{pdsharma, rameshn, ynsingh}@iitk.ac.in
Department of Electrical Engineering
Indian Institute of Technology, Kanpur - India

***Abstract*—To perform a task in coordinated fashion in distributed environment, we need the common notion of time. To achieve this purpose, clock synchronization is conceptualized. Clock drifting is inherent property of a clock which necessitates the synchronization. A network consists of nodes which may behave properly or misbehave. The misbehaving node may pose problem in clock synchronization. We require a good mechanism to mitigate the effect of misbehaving nodes. In this paper, we present a weighted average clock synchronization algorithm to perform coordinated activity in fault tolerant manner. We use behavior of the node to calculate normalized weight in localized fashion which lies between unit intervals. This weight assignment enables us to suppress effect of misbehavior up to some extent. This algorithm offers improved precision while tolerating misbehaving nodes. The upper bound of tolerance limit is one third of network size.**

***Index Terms*—Clock, synchronization, Fault Tolerant, distributed, drift, weighted average**

I. Introduction

A distributed network consists of spatially dispersed heterogeneous nodes. These nodes are required to carry out collective functions in close coordination to one another. This is possible only when they share some common notion of time. This ensures temporal precedence order across the network. Generally time is maintained by an instrument which ticks periodically at every fixed small interval of time. This instrument is the physical clock. As introduced by Newton, real time exists and progresses at a consistent pace throughout the Universe. A perfect physical clock can maintain a consistent time without any deviation i.e. it can keep real time. Every physical clock designed fails to do this because of the physical properties of the material along with external factors like temperature, pressure and aging. This means every physical clock value will deviate from the real time. The deviation of a clock with respect to the perfect clock is the drift of the clock. The rate at which the clock is drifting is the drift rate, ρ. Mathematically drift rate can be represented as $\frac{dC(t)}{t} = \rho$ where t is the real time. For perfect clock$\frac{dC(t)}{t} = 1$. Different physical clocks will have different drift rates and hence all clocks will drift with respect to one another. The measure of clocks drift with regard to each other is clock skew. Clock drift is a special case of clock skew with respect to the perfect clock. Figure 1 depicts the notion of perfect, fast and slow clocks.

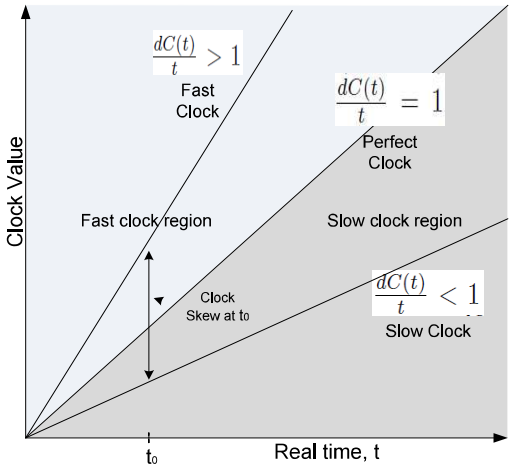

Figure 1: Perfect, fast and slow clocks

We ideally want all nodes to share the exact notion of time but due to drifting this is not possible. So we relax the meaning of 'exactness'. We define the 'Relax exactness' in sense of a interval which is ensure by clock synchronization. Clock synchronization ensures that these drifting clocks are within a thin envelope of time. This agreed bound is the maximum deviation, δ, within which all the good clocks should lie. Clock synchronization is done periodically to maintain the clock synchrony. Accuracy of a network is the measure of closeness of the clocks in the network to a reference clock outside the network. Precision is the closeness among the clocks of the network. Here we use concept of logical clock [1] in synchronization algorithm for distributed network. Each of the nodes has a physical clock. Each node derives their logical clock from its physical clock. We achieve synchronization using the logical clock which is software defined.

The network consists of nodes, communication links and clocks. There are various failure possible in a network namely node failure, communication link failure and clock failure. In this paper our concern is clock failure. The various possible scenario of clock failure in the network are as follows:

(i) All clocks in the network are good i.e. they all behave. This is the ideal case and clock synchronization is easily achievable.

(ii) Good clocks are in majority and bad clocks are in minority. Here clock synchronization is achievable by using like majority voting function and various convergence func-

tions. These bad clocks may include malicious clocks[2] with arbitrary behavior.

(iii) Good clocks are in minority and bad clocks are in majority. Clock synchronization difficult but possible. Clock synchronization is achieved by non-averaging convergence function.

Our algorithm provide the solution of the first and the second case.

In this paper we propose an algorithm which use weighted averaging method as a convergence function to achieve better precision. The algorithm has a significant advantage. It offers improved Precision. The algorithm will tolerate just under one third faulty nodes.

The remainder of this paper is organized as the following. Section 2 describes some of the related works. In section 3 we defined various definition which will be used subsequently in the paper. In Section 4 we explain the system model and give the statement of the problem in hand. Our approach to the problem is explained in section 5. Section 6 is devoted to result and evaluation. We finally conclude in section 7.

II. Related Work

Clock synchronization in a distributed system have been extensively studied in the last few decades. Many synchronization algorithm used averaging technique as convergence function will varying degrees of efficiency and complexities. One of the earlier algorithm[3] uses Egocentric averaging function where the mean of the clocks value are returned as the new clock value of the node. For calculation of the mean all clocks value are taken except for those which are more than δ distance from it where the node replace such with own value. This method suffers from clustering if δ is very small and the quality of the precision will reduced if δ is very large. In a method proposed in [4] the new clock value is the midpoint of the range which is calculated by discarding the highest and the lowest clocks value. The under laying assumption for the algorithm is good clocks lie in between bad clocks which are faster or slower than the good clocks. Two variants of sliding window method for clock synchronization are given in [5], here first a window of appropriate size chosen and mean or the median of the same window is return as the new clock value. Another method for calculation is given in [6-7] where the average is calculated for all clocks value within δ for at least $n - f$ other nodes for total of n nodes and f faulty nodes. There are also many algorithms based on non-averaging convergence technique [8-10] for clock synchronization. Some of this technique guaranteed optimal precision.

Motivation: Our motivation for writing this paper is to design a novel clock synchronization algorithm which offers a near optimal precision for clocks in a distributed network even in presence of malicious clocks.

III. Preliminaries

In this section, we are giving some basic definitions which helps to develop our system model and synchronization algorithm.

Definition 1. Correct Behavior: Correction Behavior in context of distributed network is the performance of various function by the network component as expected according to the designed or algorithm.

Definition 2. Failure: A failure is a deviation from a correct behavior. Any component or subsystem which deviates from their correct behavior is considered to be failed.

Definition 3. Faulty Component: Any component which fails is faulty.

Definition 4. Malicious Behavior: Malicious behavior is an arbitrary fault that occurs during the execution of an algorithm by a distributed system. It is a serious type of failure which give inaccurate, untimely and conflicting information. Malicious behavior includes 'dual-faced' clocks which give different time to different nodes at the same real time.

IV. System Model and Problem Statement

In this section we give a detailed description of the system model. We also define the problem statement of the paper subsequently.

A. System Model

The system model is explained in two parts, firstly the we describe the network design and then the failure model of the network.

1) Network, Nodes and Clock: We consider a fully connected distributed static network where all the nodes can communicate to one another directly by passing messages. Each of the nodes has a local physical clock from where the logical clock of the node is derived. We assumed that the physical clocks run continuously with respect to the real time. This physical clock can drift at rate ρ apart from the real time due to factors like temperature, pressure or aging. We make the following assumptions in our paper:

Drift-Bound*: The Physical Clock for any node* n_i, $P_{n_i}(t)$ is ρ-bounded for all real time t as following:

$$(1+\rho)^{-1} \leq \frac{dP_{n_i}(t)}{dt} \leq (1+\rho) .$$

The logical clock is related to the physical clock as L(t) = P(t) + $A(t)$ where $A(t)$ is the called the adjustment. This adjustment can be done discretely or linearly.

Initial synchronization: *The logical clock of any two nodes* n_i *and* n_j *at* t_0 *where* t *is the real time at the start of the synchronization algorithm will differ by real time* α.

$$\mid L_{n_i}(t_0) - L_{n_j}(t_0) \mid \leq \alpha$$

This is basically the initial synchronization assumption. Here we are assuming that all the nodes are initially synchronized at real time t_0.

2) Failure Model: In our system failures can due to failure of nodes, their clock or due to link failure between the nodes or a combination of some or all these failure. A clock is considered failed if the *Drift-Bound* assumption is violated. We are basically considering two types of clock failures namely timing failure and malicious clock. If a clock gives inaccurate, untimely and conflicting information or behave like a 'dual-faced' clock then we assumed the clock is a malicious clock. A clock is a good clock if it is running without any failure. We also assumed that the maximum number of bad clock is f.

Maximum Clock Failure: *There are at maximum f number of faulty clock in the network for 3f+1 nodes.*

B. Problem Statement

Our algorithm uses a weighted averaging method for attaining synchronization of the nodes in a distributed network. The synchronization algorithm will satisfy the following properties:

Agreement Property*: For good clocks of nodes n_i and n_j at real time t immediately after re-synchronization, $C_{n_i}(t)$ and $C_{n_j}(t)$ will satisfy :*

$$|C_{n_i}(t) - C_{n_j}(t)| \leq \Pi,$$

where $\Pi \leq (\delta + \epsilon)$ is the precision of our synchronization algorithm.

Incremental Step property: *The correct clocks will change every time by some amount ξ at each synchronization.*

Accuracy property*: The correct clocks of node n_i, for some constants a, b, c and d will hold for real time $t_1 < t_2$ the following:*

$$(t_2 - t_1)/a - b \leq C_{n_i}(t_2) - C_{n_i}(t_1) \leq (t_2 - t_1)c + d$$

We would like to make Π as small as possible so that clocks are close to one another. The validity of the *Agreement property* and *Incremental Step* property implies that our good clocks are running linearly with respect to real time. We would also like to make $(t_2 - t_1)$ as close as possible to $C_{n_i}(t_2) - C_{n_i}(t_1)$ for any node n_i. This will ensure the time elapsed by the clock of any node n_i is same to the real time elapsed in the interval. For optimal accuracy a and c should equal to one and b and d should be zero in expression of property 3.

V. Weighted Average Synchronization Algorithm (WASA)

We present a synchronization algorithm which achieves clock synchronization by using a weighted averaging as a convergence function. This algorithm is called WASA and it utilizes the concept of sliding window to find the minimum variance data set upon which the convergence function is used.

A. Distribution of Bad Clocks in the Network

In a network each node exchanges their clock value and store in an array in a order of their clock values. The clock values distributed in the array can be visualized in in the following fashion: n clock values are distributed in the array $i.e.$ we can visualize this as n slots to be filled with $n - f$ good clock values and f bad clocks. The good and bad clocks values can be anywhere in the n slots. There will be ${}_nP_f$ distinct possibility as depicted in figure 2(a) below.

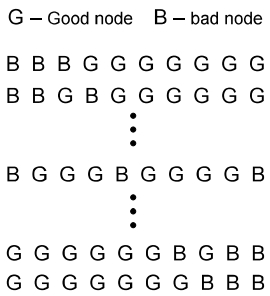

Figure 2(a): Distribution of good and bad clock values

The two extreme cases are as shown where all the bad clocks have accumulated in the left or the right corner as shown below in figure 2(b).

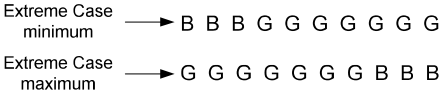

Figure 2(b): Extreme distribution of good and bad clock values

B. Minimum Variance Window

The good clocks in a network are spread around the $\pm\delta$ from the mean. Bad clocks can be anywhere in the array. As most of the world values are normally distributed, approximately sixty eight percent of these n clock values are within $\pm\sigma$ of the mean, where σ is the standard deviation of the distribution of all the clocks. σ is a measurement of deviation of the values from the mean. Now let us take a window of size 2σ. Initially the window contain the left most clock values of the array as its left most member. We slide this window from the left a clock value ahead along the array. The sliding will start from left corner of the array and move toward right. We find the variance of the window at each window instant. The minimum variance window is the window containing clock values within $\pm\sigma$ from the mean. If the size of the window is reduce around two-third i.e. sixty six percent of the total clocks, then also the member clock values are same with few reduction in the number of clock values. This is particularly true if there is no malicious clocks are present which may give different clock values to different nodes. In presence of malicious clocks the member of the window may change by up to f clock values. But since the window must maintain minimum variance the means calculated by the different arrays stored by different nodes will vary within a bound. We use this concept in our algorithm.

C. Overview of WASA

In a network, two kind of player are there with conflicting goals. One set of player which has objective to stabilize the system is called good nodes. The other set of player which aims to destabilize the system is called bad nodes. Our goal is to synchronize the network even in presence of bad nodes. The most serious type of bad nodes is those nodes which give inaccurate, untimely and conflicting information. They may also give different time to different nodes at the same real time. This malicious behavior of nodes can be of classified as follows: *Consistent misbehavior* - Nodes will exhibit the same malicious behavior in temporal domain. *Inconsistent misbehavior* - Nodes display malicious behavior is inconsistent fashion in temporal domain. A node can misbehave in individual capacity or can collectively collude. These misbehavior can be categorized as follows: *Individual misbehavior* : A node is displaying misbehavior in individual manner without consulting other bad nodes. *Collaborative malicious behavior*: Bad nodes may consult each other and decide their strategy in collective fashion.

In this paper our focus is to counter the individual misbehavior which includes consistent and inconsistent behavior both. We have devised a sliding window technique to find the set of values which shows minimum deviation among the values. We know that Gaussian distribution is widely present in nature. We capitalized this knowledge to design the weight function. This weighted approach is used to mitigate the misbehavior.

D. Detail View of WASA

Clock working on time line is divided into equal rounds. T_r time duration before the end of each round is called re-synchronization time. During re-synchronization period each node executes WASA. The duration of a round is constant R which is determined by the deviation range. Hence R is decided according to the requirement of the system design. WASA can counter consistent maliciioius nodes. It also works in presence of inconsistent malicious behavior provide the malicious nodes remain consistent during T_r. The figure 3 depicts the relationship of R and T_r.

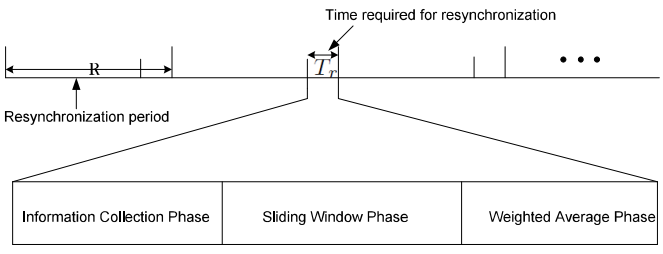

Figure 3: Working of a Clock and re-synchronization period

WASA consists of three phases namely Information Collection phase, Sliding Window phase and Weighted Average phase.

1) Information Collection phase: In this phase every nodes will share their clock with one another. Node stores these clock values in an array. The clock values in the array is then arrange in an ordered fashion as per their values.

2) Sliding Window phase : We introduced the notion of window. The size of window is $n-f$ where n network size and f is allowed bad nodes. Initially we placed the left boundary of the window at the beginning of the array. So first window consists of first $n-f$ values of the array. Thereafter we slide the window one step in the right direction and this consists of next $n-f$ values. We continue this process until the right boundary of the window reaches end of the array. In this process we find the f data set of size $n-f$. Now out of f data sets we find out the data set with minimum variance.

3) Weighted Average phase: We use the minimum variance data set calculated in the Sliding Window phase for finding the weighted average with some modifications. Since the size of the minimum variance window is $n-f$, it contains at least $n-2f$ good clocks and at most f bad clocks. For the worst case the minimum variance window contains $n-2f$ good clocks and f bad clocks. There can be also at maximum f good clock outside the minimum variance window.

We now push these good clocks outside the minimum variance window inside the window by assigning the value of the good clock nearest to them inside the window. Now data set values inside the minimum variance window are assigned weights as per weight function and the weighted average is calculated. These weighted average value, θ, is the new clock.

For assignment of the weights, clock value within one $\pm\sigma$ distance from μ are assigned weight ω_1, where μ is the mean and σ is the standard deviation of the window. Clocks within $\pm\sigma$ to $\pm 2\sigma$ are assigned weight ω_2 and clocks within $\pm 2\sigma$ to $\pm 3\sigma$ are assigned weight ω_3. Finally clocks beyond $\pm 3\sigma$ are assigned zero weight. A pictorial description of working of WASA is given below at figure 4.

VI. Evaluation of WASA

Here we try to evaluate the performance of WASA algorithm. Initially all clocks are synchronize in some sense. At the start of the first synchronization period all the correct clock are within α of the real time t since it is α synchronous *(Initial synchronization*). The distance between any two correct clocks will be equal to or less than δ at the end of R. If ϵ is the maximum message delay then width of the window will be at most $\delta + \epsilon$. Now we discuss the performance of WASA for the ideal and worst case scenario.

A. Ideal Case Scenario

In the ideal case scenario there is no bad node in the network. Hence the data set of clocks are the same for all the clocks. The minimum variance of these windows is the same. Therefore the weighted average, θ is identical across all the clocks.

Lemma 1. The precision of WASA in an ideal condition where there is no bad clock is $\Pi = \epsilon$, which is optimal.

Proof: The size of the minimum variance window is $n-f$ but for a where case $f = 0$, hence the size is now n. This mean there will be only one window and a single weighted average, θ value which will the new clock. Hence the precision will be optimal. ■

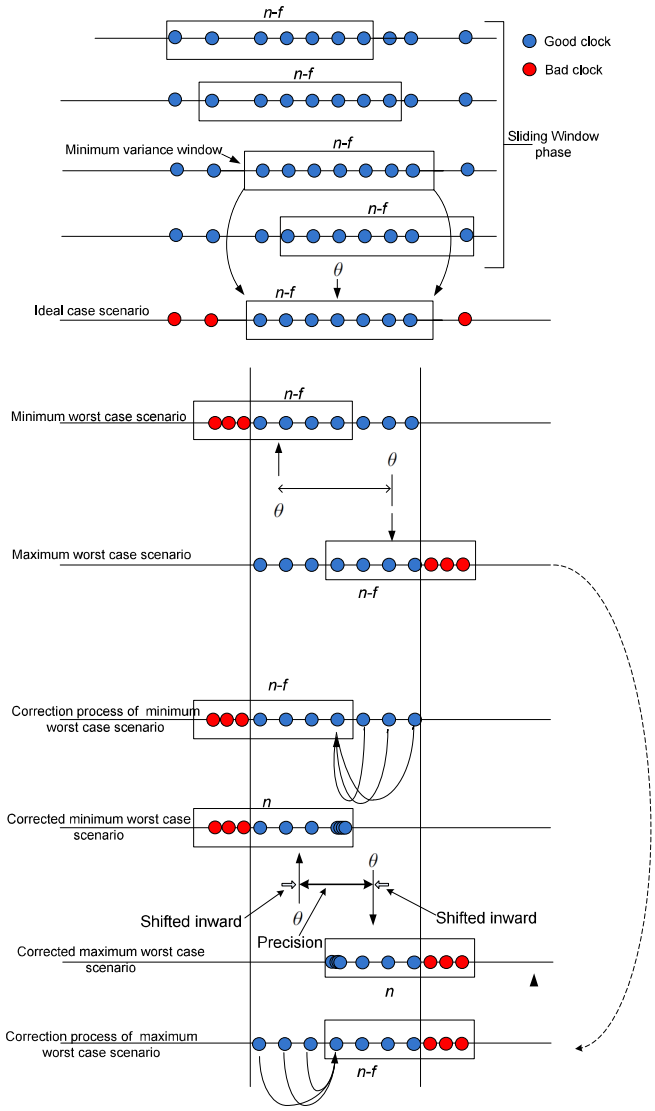

Figure 4: Pictorial description of working of WASA

B. Worst Case Scenario for Proof of Correctness

We know that if an algorithm is validated for the worst case then it holds for all possible cases. Here we do worst case analysis to establish correctness of WASA. The worst case is the condition when WASA will give the worst possible precision. This condition is obtained where all the bad nodes are malicious nodes. From *Maximum Clock Failure assumption* the size of the network is $n = 3f + 1$. Let C_{min} and C_{max} be the minimum and the maximum clock created by WASA immediately after R. Also let C_{min} $window$ be the window from where C_{min} is calculated and similarly C_{max} $window$ for C_{max}. Here the malicious clocks are all within the minimum variance window and make C_{min} the least and C_{max} the largest. This happens when all the malicious clocks are smaller than or equal to the smallest good clock. The good clocks will have minimum drift and delay. In case of C_{max} all the malicious clocks will be larger than or equal to the largest good clock. The drift and delay will be highest. A distribution pattern of the clocks for the worst case is depicted in the figure 5 below. As shown any good clocks are in either of the window. The C_{min} $window$ contains $f + 1$ good clocks out of which f good clocks are exclusively within it. Similarly the same distribution for C_{max} $window$. One good clock is common to both the windows. The difference between C_{min} and C_{max} will give the worst case precision, Π, of WASA.

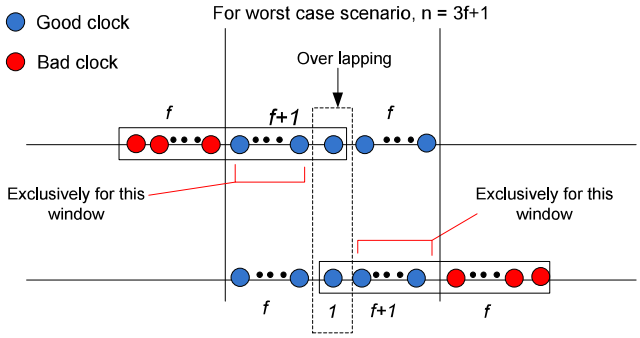

Figure 5: Clock values distribution for worst case scenario

Theorem 1 **.** In the worst case the precision of WASA , Π, if the upper bound of bad clocks is one-third of the total number of nodes, n, with a maximum window width $\delta + \epsilon$ is given by

$$\Pi \leq \epsilon + 2(\delta + \epsilon)\left(\frac{f}{n}\right) = \epsilon + 2(\delta + \epsilon)\left(\frac{f}{3f+1}\right)$$

Proof: Here Π is the distance between C_{min} and C_{max}. From *Lemma 1* we know that for ideal condition all the clocks are within ϵ of θ. With the presence of bad clocks θ is shifted to either direction. In case of C_{min} all the malicious clocks is to the left of the left most good clock. The maximum deviation of the $C_{min}window$, $\delta_{min} \leq \delta$ because it is the minimum variance window. C_{min} is obtain by shifting θ by an amount upto $f\frac{\delta}{(n-f)}$ since there are f malicious clocks. The shift is made lesser by pushing the left out good clocks into the window. Now the number of clocks is increased to n from $n - f$. Hence the distance shifted is up to $f\frac{\delta}{n}$.

Similarly the shifting due to malicious clocks on the right side of the good clock is up to $f\frac{\delta}{n}$ on the right side of θ. This gives value of C_{max} w.r.t. θ. Hence considering ϵ, the distance between C_{min} and C_{max} is given by

$$\Pi \leq \epsilon + \{\theta + (\delta + \epsilon)\left(\frac{f}{n}\right)\} - \{\theta - (\delta + \epsilon)\left(\frac{f}{n}\right)\}$$

$$\Pi \leq \epsilon + +2(\delta + \epsilon)\left(\frac{f}{n}\right)$$

but $n = 3f + 1$, therefore

$$\Pi \leq \epsilon + 2(\delta + \epsilon)\left(\frac{f}{3f+1}\right)$$

Hence proved. ■

Proposition 1. The worst possible precision of WASA for any value of f is within :

$$\Pi \leq \epsilon + \frac{2(\delta + \epsilon)}{3}$$

Proof: From *Theorem 1* we have

$$\Pi \leq \epsilon + 2\left(\delta + \epsilon\right)\left(\frac{f}{3f+1}\right)$$

If the number of malicious clocks is very large i.e. $f \gg 1$ then $3f + 1 \simeq 3f$ hence

$$\mathit{\Pi} \leq \epsilon + \frac{2(\delta + \epsilon)}{3}$$

This is the worst possible Π for any value of f. ■

Our aim of designing WASA is to ensure that the good clocks never deviates from $\delta + \epsilon$ as defined in *Property 1*.

Proposition 2. WASA guaranteed satistifaction of *Property 1., the Agreement Property*.

Proof: The worst possible precision guaranteed by WASA is given by *Property 1* which is

$$\mathit{\Pi} \leq \epsilon + \frac{2(\delta + \epsilon)}{3}$$

Since $\epsilon \ll \delta$,

$$\mathit{\Pi} \leq \epsilon + \frac{2(\delta + \epsilon)}{3} < (\delta + \epsilon)$$

Hence WASA guaranteed *Property 1.* ■

From *Proposition 2* we can establish that WASA ensures all good clocks are consistently within $\mathit{\Pi} < (\delta + \epsilon)$. This complete the proof of correctness.

The worst case precision by SWA^{det}_{mean}[5] is given as for $n \geq 4f$

$$\Pi_S \leq \epsilon + f\frac{(\delta + \epsilon)}{n - f} + f\frac{(2\delta + \epsilon)}{n - f + 1}$$

If $f \gg 1$ and taking $\delta \gg \epsilon$, WASA is atleast 33% tighter.

VII. Conclusion

In this paper, we devise and present a WASA algorithm for clock synchronization which is suitable for fully connected network. WASA achieve at least 33% tighter precision in the worst case scenario. Even the worst case scenario occurrence is very less but our study provide meaningful analytically insight. WASA guarantee clock synchronization in presence of malicious clocks if the upper bound of malicious nodes is just under one-third of the network size. In our current work we provide the analysis for static network. In future work we will try to achieve synchronization for dynamic and partial connected network.

References

[1] C. Lenzen, T. Locher, P. Sommer, and R. Wattenhofer. Clock synchronization: Open problems in theory and practice. In Proc. 36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 10), pp. 61-70, 2010.

[2] L. Lamport, R. Shostak, and M. Pease, The Byzantine Generals problem, ACM Trans. On Prog. Lang. and Systems. Vol. 4, No. 3,pp. 382-401 July 1982.

[3] L. Lamport and P. M. Melliar-Smith. Synchronizing Clocks in the Presence of Faults. Journal of the ACM, Vol. 32, No. I,pp. 52-78, January 1985.

[4] Jennifer Lundelius Welch and Nancy Lynch. A New Fault-Tolerant Algorithm for Clock Synchronization. Information and Computation 77, pp. 1-36, 1988.

[5] M.J.> Pfluegl and D. M. Blough. A New and Improved Algorithm for Fault-Tolerant Clock synchronization. Journey of Parallel and distributed Computing 27 pp. 1-14, 1995

[6] Riccardo Gusella and Stefano Zatti. An Election Algorithm for a Distributed Clock Synchronization Program. In Proc. of 6th international Conference on Distributed Computng Systems, pp. 364-373, 1986.

[7] Stephen R. Mahaney, Fred B. Schneider. Inexact Agreement:Accuracy, Precision, and Graceful Degradation. In Proc. of 4th International Symposium on Principles of Distributed Computng , pp. 237-249, August 1985.

[8] Joseph Y. Halpern, Barbara Simons, Ray Strong, Danny Dolce. Fault-Tolerant Clock Synchronization. In Proc. of 3rd International Symposium on Principles of Distributed Computng , pp. 89-102, 1984.

[9] F. Cristian, H. Aghili, and R. Strong, "Clock Synchronization in the Presence of Omission and Performance Faults, and Processor Joins," Proc. 16th Int'l Symp. Fault-Tolerant Computing Systems, pp. 218-223, 1986.

[10] T. K. Srikanth and Sam Toueg. Optimal Clock Synchronization. Journal of the ACM, Vol. 34, No. 3 ,p p. 626-645, July 1987.