Clock Synchronization in Mobile Ad Hoc Networks
based on an Iterative Approximate Byzantine Consensus Protocol

Chuanyou Li, Yun Wang
School of Computer Science and Engineering, Southeast University
Key Lab of Computer Network & Information Integration, Ministry of Education
Nanjing, China
Email: chuanyou.li@gmail.com, yunwang@seu.edu.cn

Michel Hurfin
Inria
Rennes, France
Email: michel.hurfin@inria.fr

Abstract—We consider the clock synchronization problem in wireless mobile ad hoc networks in the presence of Byzantine nodes. The communication topology is dynamic: nodes move randomly within a geographical area. We propose a clock synchronization protocol which is based on the linear approximate consensus method. Periodically each correct node broadcasts its current timestamp and gathers the timestamps provided by its current neighbors. To cope with the malicious nodes (and to improve the performance when the node density is low), each node keeps the collected timestamps in a local log. As a log may contain values received more or less recently, a transformation technique is introduced to refresh the outdated values. The accuracy of the synchronization depends on the connectivity among the moving nodes. We use a matrix and vector based representation to model the behavior of the synchronization process and to analyze its accuracy. We show that the deviation between the different clock values can converge towards zero when a particular condition is satisfied infinitely often. The frequency at which the condition is satisfied also impacts the synchronization accuracy. Based on a particular mobility scenario, performance simulations are conducted.

Keywords—clock synchronization; approximate consensus; mobility; ad-hoc network;

I. INTRODUCTION

We consider the clock synchronization problem in a distributed system composed of wireless mobile devices. The nodes form an ad hoc network and move randomly within a geographical area. At a given time, a node can communicate only with the nodes located within its current communication range. The system is unreliable. Messages may be lost. Nodes may suffer from Byzantine failures. A Byzantine node, called also a malicious node, may stop its activity or execute arbitrary statements. The non-faulty nodes are called correct nodes. In such a network, synchronized clocks are crucial for many applications. For example, an accurate synchronization can be required by a sleep/wake protocol executed by energy-constrained sensor nodes. Monitoring tasks may also require to identify a total order among events observed on different nodes. As the local clocks of nodes may drift away from each other in time, a clock synchronization protocol aims at correcting the divergence between the clocks of the correct nodes. In this work, we assume that there is no predefined clock source (no external reference clock and no reference node). All the nodes participate to the clock synchronization protocol on an equal basis. The goal of the correct nodes is to reduce the difference between their clocks. Obviously, this objective is more difficult to achieve when the topology is dynamic and when the network contains malicious nodes.

The clock synchronization strategy proposed in this paper is based on a solution to the approximate consensus problem [1]. Approximate consensus is a paradigm which does not require to achieve exact agreement on a single outcome value. Each node starts the execution with a local numerical value called its initial value. Convergence is reached as soon as all the correct nodes have obtained values that are different from each other within a maximum predefined limit. To solve this problem, a linear iterative method is often adopted [2]. At each iteration, a node broadcasts its current value, gathers values from its neighbors and updates its own value. To ensure the convergence in spite of a few Byzantine nodes, [3], [4] define a sufficient and necessary condition in static multi-hops networks and [5] considers the same problem in mobile ad hoc networks. When the condition is not satisfied, convergence is not necessarily ensured. Indeed the node’s mobility may prevent some correct nodes to receive enough values.

Different works consider the clock synchronization problem in wireless sensor networks and mobile ad hoc networks. Many works [6], [7], [8], [9] aim at improving the synchronization accuracy while minimizing the overheads. [10], [11] propose clock synchronization algorithms based on the linear iterative consensus method. Yet these works do not address security issues. [12], [13], [14] propose robust solutions to cope with Byzantine nodes. [12] relies on the existence of deployed tamper-proof nodes. Such a node will destroyed itself once it is compromised. These trusted nodes do not exist in our context. [13] focuses on the delay attack and uses statistical methods to detect or to filter outliers. Our work is rather inspired by fault-tolerant techniques: we do not try to identify the malicious nodes but we synchronize the clocks in spite of them. Based on a firefly synchrony
method, [14] also proposes a Byzantine fault-tolerant clock synchronization algorithm for ad hoc networks. However, it tolerates only up to $1/5$ of faulty nodes and requires a $5f + 1$ connected network. We do not require such a strong connectivity in the network. Moreover we tolerate that up to less than $1/3$ of the nodes are malicious.

We propose a clock synchronization protocol which follows the strategy described in [5]. Each node broadcasts its clock value periodically and collects values from other nodes while it moves around. To tolerate the Byzantine nodes, the local clock updating operation is carried out by a correct node only when enough values have been received. Collecting values from a sufficient number of nodes may require a non-negligible time. Indeed, a node can be temporarily isolated or be surrounded by the same neighbors. Consequently, when a node has finally collected enough clock values, many of them are already out of date. To cope with this problem, a clock transformation technique is introduced. It allows to transform the obsolete values (received at different instants in the past) into fresh temporary values as if they have been just received, all at the same time. There exists several differences between our work and the related works. First, no global scheduler or common pulse generated system is assumed. The nodes perform asynchronous actions (e.g., when a node updates its clock value, other nodes are perhaps still moving to collect more clock values). Second, in an approximate consensus scenario, the proposed value of each correct node is only updated by the running consensus algorithm. In the clock synchronization problem, besides the synchronization algorithm, the clock value of each node is also updated by the underlying hardware clock system: the local clock value is progressing forward thanks to two independent mechanisms. The modeling of the node’s behavior proposed herein integrates this aspect. To analyze the accuracy of the synchronization, the whole process is modeled with matrices and vectors. We demonstrate that if a particular condition is satisfied infinitely often, the deviation from different initial clock values will converge. We also point out that when this condition is satisfied more frequently, a lower deviation of the synchrony outcome is achieved.

The paper is organized as follows. Section II sketches out some related works. Section III introduces the system model and the problem. In Section IV, we present our algorithm based on the linear iteration method. A matrix-based presentation of the synchronization algorithm is provided in Section V. Based on this model, Section VI analyzes the accuracy of the clock synchronization. A condition which impacts the synchrony outcome is proposed. In Section VII, we evaluate the performance through simulations.

II. RELATED WORKS

Kay Romer proposes a time synchronization algorithm in ad hoc networks [6]. His goal is not to synchronize the local device clocks: he aims at generating timestamps by using the unsynchronized local clocks. When these timestamps are passed between devices, they are transformed into the local time of the receiving device. By using this technique, nodes in the network can have a consistent view on the sequence of events that have occurred (construction of a total order). Methods proposed in [7], [8], [9], [10], [11] focus on the synchronization of the local clocks of the devices. In [7], Jana van Greunen et al. first design a single-hop synchronization scheme and then they extend this single-hop solution to a multi-hop one. A spanning tree is constructed and the multi-hop approach performs the single-hop method for each edge of the spanning tree. Instead of using a fixed spanning-tree based approach, Miklos Maroti et al. propose a synchronization method which is based on flooding [8]. The cost for constructing a spanning tree is saved, but flooding entails new costs. To avoid both network-wide flooding and rigid structures like the spanning tree, [9] proposes an opportunistic and on-demand synchronization method. The algorithm is executed only when necessary, such that the overhead is reduced. [10], [11] address their clock synchronization algorithms in a different manner. Their approach follows the linear control theory. [10] estimates the clock skew and the offset while [11] tackles the problem of noisy measurements of the skew and offset.

The works aforementioned do not address security issues. Even if a single node suffers from Byzantine failures, the synchronization process may be compromised. Hui Song et al. propose an attack-resilient clock synchronization algorithm which is based on statistical methods [13]. By computing a threshold, compromised values are detected and suppressed. [14] proposes a Byzantine fault-tolerant synchronization algorithm based on the reach back firefly algorithm. None of these works provides a global convergence analysis. In [12], Qun Li et al. also propose a Byzantine fault-tolerant clock synchronization scheme and prove global convergence by using matrix theory. However their method is based on tamper-proof nodes and the deployment of these nodes is characterized by strong constraints: 1) each normal node must be one hop away from a tamper-proof node; 2) any two tamper-proof nodes must communicate via a route on which no two consecutive nodes are normal nodes; 3) among the shared neighbors of two tamper-proof nodes, at most one third can be compromised. These requirements are difficult to satisfy in the case of a mobile sensor network.

III. SYSTEM MODEL AND PROBLEM DEFINITION

A. Mobile Ad Hoc Networks

We consider a mobile network composed of $n$ nodes represented by a set $N = \{p_1, p_2, ..., p_n\}$. Nodes communicate with each other only by exchanging messages. Each node has three basic modules: a mobility control module, a radio communication module and a calculation module. The mobility control module is in charge of the problems
related to the motion. The duty of the radio communication
module is to send (receive) messages to (from) other nodes.
The calculation module is responsible for operating the
computing tasks (in particular, updating the local clock).

The communication range is limited: a node can only
communicate with the neighbors located within its communica-
tion range. As a wireless communication can easily be
interfered, messages can be lost. \( P_{\text{lost}} \) is used to denote
the loss probability. \( P_{\text{lost}} \) is not a constant. Yet it always
satisfies the condition \( 0 < P_{\text{lost}} < 1 \). If a message is not
lost, its passing delay between two neighbors is denoted \( \sigma \).
The notation \( \sigma_{ij} \) is used when the message is received by
\( p_j \) from \( p_i \). The passing delay depends on many parameters
(channel condition, scheduling algorithm, arrival rate, etc)
and thus it may vary. When an ad hoc network is not heavy
loaded, the message passing delay between two neighbors is
in the order of a few milliseconds [15]. We assume that
there exists a maximum bound \( \sigma_{\text{max}} \) for all \( \sigma_{ij} \).

Among the \( n \) nodes of the set \( N \), some are correct nodes
while others are Byzantine nodes. Correct nodes (belonging
to the subset \( N_C \)) always follow the protocol’s specification,
while Byzantine nodes (belonging to the subset \( N_F \)) can
collude together and behave arbitrarily. A malicious node
can send fake values, delay and drop messages, or stop
the computation. However, the total number of Byzantine
nodes is limited by \( f \) and satisfies \( n \geq 3f + 1 \). Moreover
each correct node is able to identify the real sender of any
received message. In other words, a faulty node can not use
fake identities to communicate with others.

All nodes walk on a predefined finite graph \( G = (V, E) \).
Initially the \( n \) nodes are distributed randomly among all the
vertices of set \( V \). Let us consider the mobility behavior of
a correct node \( p_i \) located on vertex \( v_x \). To move, it selects
randomly the destination \( v_y \) that satisfies \( \langle v_x, v_y \rangle \in E \).
Then it walks along the edge \( \langle v_x, v_y \rangle \) until it reaches
the destination vertex \( v_y \). Once arrived in \( v_y \), the node \( p_i \) can
decide to stay during a limited period of time in this place.
In any case, it will leave this place and move later to another
vertex \( v_z \). In that sense, all the correct nodes walk infinitely.
We assume that the velocity of each node (correct or faulty)
is limited. The velocity of a correct node is a constant but
two correct nodes may have different velocities. Every node
is assumed to be a point: they can move independently and
no moving collision is considered. A Byzantine node
can plan special routings or remain motionless. While it
remains in the graph, it has to follow the predefined paths.
The assumption that correct nodes can move randomly
and indefinitely in a limited graph allows to propose the
following first claim.

Claim 1: At any time and for any two correct nodes \( p_i \)
and \( p_j \), the probability that \( p_i \) and \( p_j \) become neighbors in
a bounded time approaches to 1.

In other words, we assume that the trajectories of any two
correct nodes intersect infinitely often. As the message loss
probability always satisfies \( 0 < P_{\text{lost}} < 1 \) and as \( p_i \) can
meet \( p_j \) infinitely often, the following claim holds also.

Claim 2: At any time and for any two correct nodes \( p_i \)
and \( p_j \), the probability that \( p_i \) receives in a bounded time a
new timestamp from \( p_j \) approaches to 1.

B. Computer Clocks

Computer devices are equipped with a hardware oscillator
which can be used to implement an approximation \( C(t) \) of
real time \( t \). The value of \( C(t) \) is equal to \( k \int_{t_0}^{t} w(t)dt + C(t_0) \),
where \( w(t) \) is the angular frequency of the oscillator,
k is a coefficient for that oscillator and \( t \) is the real time [6].

In a perfect scenario, at a real time instant \( t \), \( C(t) = t \)
which implies \( \frac{dC}{dt} = 1 \). Unfortunately all hardware clocks
are imperfect. They are subject to clock drift. The skew rate
depends on the environment changes (temperature, pressure,
power voltage, etc). However in an isomorphic system, it is
possible to know the maximum skew rate \( \rho \) of a device:

\[
1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho
\]

An acceptable \( \rho \) satisfies \( 1 \gg \rho > 0 \). Today, hardware
techniques allow to obtain a value of \( \rho \) equal to \( 10^{-6} \) [6].

We assume that there is no external central clock or pulse
generated system. If \( t_0 \) is the initial real time instant, we
assume that every node \( p_i \in N \) has an initial local clock
value \( C_i(t_0) \). When the network is deployed, it is difficult
to ensure that all nodes are rigorously synchronized. Thus
two nodes \( p_i \) and \( p_j \) may have different initial values \( C_i(t_0) \)
and \( C_j(t_0) \) (even if both nodes are correct).

C. Problem Description

In a mobile ad hoc network, where up to \( f \) Byzantine
nodes may exist, the correct nodes move randomly and
indeinitely on a predefined limited graph. Each correct
node tries to synchronize its local clock only by exchanging
messages with its neighbors. Without a reference clock, the
synchronization process does not converge towards the real
time \( t \), but tries to eliminate the deviation between any two
correct nodes’ local clock values. Thus, the problem is to
decrease the value \( |C_i(t) - C_j(t)| \), \( \forall p_i, p_j \in N_C \).

IV. THE SYNCHRONIZATION ALGORITHM

Each correct node broadcasts periodically its timestamp
(i.e., its current clock value). When a correct node receives
a new timestamp from a neighbor, it launches the execution
of the synchronization algorithm. In a nutshell, this algo-
rithm performs three major tasks. First, it transforms the
timestamps already contained in a local log into temporary
values as if all these values have been received at the same
time than the last received one. Then, if enough values are
available, it identifies among the gathered timestamps those
that may be fake and unsafe. Finally, it calculates the average
of the remaining timestamps and updates its clock value.
Figure 1 shows the pseudo-code executed by a correct node $p_i$. It initializes its local clock value $C_i$ (line 1) and a log buffer $Neb_i$ (line 2). Remember that $C_i$ can be changed by two independent mechanisms: the synchronization algorithm currently described and the underlying hardware clock system which can only increase the value of $C_i$. The log $Neb_i$ stores timestamps that have been received by $p_i$ from other nodes. Each value contained in $Neb_i$ is represented by a pair $(C_j, C'_j)$. The value $C_j$ represents the timestamp received by $p_i$ from the node $p_j$ and the value $C'_j$ is the time (provided by $p_j$’s local clock) at which $C_j$ was received. Depending on the context, a value of the log refers either to such a couple or to a value $C_j$ contained in a couple.

When a new timestamp $C_j$ is received, $p_i$ begins to execute the clock updating procedure (lines 3-14). Of course, $p_i$ can not determine if the sender $p_j$ is a correct node or a Byzantine node. Consequently, $p_i$ can not immediately update its clock by using just the new received value. The log allows to circumvent this problem [5]. First, $p_i$ stores the pair $(C_j, C'_j)$ into the log $Neb_i$ (line 4). $p_i$ will not update its clock value (lines 10-12) if it has not yet received enough timestamps (test of line 9). During the execution of line 4, If a timestamp also provided by $p_j$ is already in the log, this old value is suppressed and replaced by the new one. Thus the log may contain at most $n - n$ elements.

Collecting enough timestamps (while moving through several places) may require a lot of time. The logged values become progressively “out of date”. The call to the procedure $timeTransform$ (line 5) aims at transforming all the timestamps stored in $Neb_i$ into “recent” temporal values as if they have been received together simultaneously at time $C_i(t)$. The notation $Neb_i^{tr}$ represents the set containing the timestamps after this transformation. Note that a similar idea is proposed in [6]. Indeed, time transformation is an estimation process. In accordance to the progress of its own local clock value, a correct node $p_i$ tries to estimate the corresponding increments of the clock values of other nodes. Figure 2 provides an example. Suppose that at time $t_3$, $p_i$ needs to transform a timestamp $C_j(t_1)$ received from $p_j$ at time $t_2$ into a new temporary value $C_j^{tr}(t_1)$ as if it is received by $p_i$ at time $C_i(t_3)$. Because nodes have different skew rates, a perfect transformation is impossible. However with the known maximum skew rate $\rho$ aforementioned, it is possible to obtain a lower and an upper bound. Suppose $\Delta_{c_j} = C_i(t_3) - C_i(t_2)$ and $\Delta_{c_j} = C_j^{tr}(t_1) - C_j(t_1)$. In a first step, we consider the time interval $t_3 - t_2$ whose definition is based on real time and we conclude that $t_3 - t_2 \in [\frac{\Delta_{c_j}}{1+\rho}, \frac{\Delta_{c_j}}{1-\rho}]$. In a second step we consider $\Delta_{c_j}$ whose definition is based on the local clock of $p_i$ and we conclude that $\Delta_{c_j} \in [\frac{\Delta_{c_j}}{1+\rho}, \frac{\Delta_{c_j}}{1-\rho}]$.

After the procedure $timeTransform$, the computed temporary timestamps are sorted into ascending order (line 6). The operations from line 7 to 10 are used to identify a few timestamps which are potentially risky. Line 7 (resp. 8) computes the number of timestamps within $Neb_i^{tr}$, whose value is bigger (resp. smaller) than or equal to $C_i(t)$. The test evaluated by $p_i$ at line 9 defines two favorable cases: either $p_i$ has received timestamps that are greater than or equal to $C_i(t)$ from at least $f + 1$ different nodes, or $p_i$ has received timestamps that are smaller than or equal to $C_i(t)$ from at least $f + 1$ different nodes. If the test in line 9 is passed, $p_i$ starts to execute the procedure $reducing$ (line 10) to suppress risky timestamps. The suppressing process respects the following rules: if there are less than $f$ timestamps greater than $C_i(t)$, then $p_i$ suppresses all of these particular timestamps that are greater than its own clock. Otherwise, $p_i$ suppresses all of these timestamps that are smaller than or equal to $C_i(t)$ from at least $f + 1$ different nodes. If a malicious node $p_j$ still remains in $Neb_i^{tr}$, some correct node have necessarily proposed either the same value or both a smaller and an higher value: the value of $p_j$ is safe. If the test in line 9 is evaluated to false, $p_i$ can not safely adjust its clock value. It must continue to collect more timestamps.

After reducing, the $average$ procedure is called (line 11). To get a new value, the node $p_i$ takes into account its own local clock $C_i(t)$, all the remaining timestamps $C_j^{tr}$ contained in $Neb_i^{tr}$ and the compensation factor $\sigma_{max}/2$ which is corresponding to an estimation of the message.
transmission delay. After each computation of a new clock value, \( N_{eb_i} \) is reset to \textit{null} (line 12). This operation is necessary to avoid bad phenomena. For example, without this reset operation, after updating the clock value, the log may contain a timestamp \((C_j, C_i^m)\) which has been received in the future (if \( C_i^m > C_i \)). Moreover, keeping very old values decrease the accuracy of the estimations obtained during the time transformation process.

V. MATRIX REPRESENTATION

A. Preliminaries

In the following, a boldface upper case letter denotes a matrix and the corresponding lower case letter denotes an element of this matrix. Lower case letter (or Greek letter) with an arrowhead is used to represent a vector. For example, \( \mathbf{A} \) denotes a matrix, \( \mathbf{a}_i \) is the \( i^{th} \) row of \( \mathbf{A} \) and \( a_{ij} \) denotes the element at the position of the \( i^{th} \) row and the \( j^{th} \) column of \( \mathbf{A} \). \( \vec{v} \) is a vector while \( v_i \) is one of its components.

**Definition 1:** A matrix is a row stochastic matrix if it is non-negative (all elements are bigger than or equal to 0) and, for each row, the sum of its elements is equal to 1.

For a row stochastic matrix \( \mathbf{A} \), the coefficients of ergodicity \( \delta(\mathbf{A}) \) and \( \lambda(\mathbf{A}) \) are defined as follows [16]:

\[
\delta(\mathbf{A}) \doteq \max_j \max_{i_1<i_2} [a_{i_1j} - a_{i_2j}]
\]

\[
\lambda(\mathbf{A}) \doteq 1 - \min_{i_1<i_2} \sum_j \min(a_{ij}, a_{ij})
\]

Because of \( a_{ij} \in [0, 1] \), it is easy to see that \( 0 \leq \delta(\mathbf{A}) \leq 1 \) and \( 0 \leq \lambda(\mathbf{A}) \leq 1 \). If \( \delta(\mathbf{A}) = 0 \), then the rows of \( \mathbf{A} \) are all identical. Moreover, \( \delta(\mathbf{A}) = 0 \) if and only if \( \lambda(\mathbf{A}) = 0 \). The following theorem reveals an important relationship between \( \delta \) and \( \lambda \). Its proof can be found in [17].

**Theorem 1:** For any \( m \) square row stochastic matrices \( \mathbf{A}(1), \mathbf{A}(2), ..., \mathbf{A}(m) \), their product satisfies:

\[
\delta(\mathbf{A}(1)\mathbf{A}(2)\ldots\mathbf{A}(m)) \leq \Pi_{i=1}^m \lambda(\mathbf{A}(i))
\]

B. Modeling the Updates with Matrices and Vectors

We use matrices and vectors to express the iterative synchronization algorithm described in Figure 1. The set \( N_F \) contains all the Byzantine nodes. Let its cardinality be \( |N_F| = \tau \) (0 \( \leq \tau \leq j \)). Without loss of generality, if \( \tau > 0 \), we suppose that nodes 1 through \( n - \tau \) are correct and nodes \( n - \tau + 1 \) through \( n \) are malicious. At real time \( t \), the local clock values of all the correct nodes are represented by a vector \( \vec{h}(t) \). Inside \( \vec{h}(t) \), the element \( h_i(t) \) (1 \( \leq i \leq \tau \)) denotes the clock value \( C_i(t) \) of node \( p_i \). If an update of the clock value of at least one correct node occurs at time \( t_k \) (execution of line 11 of the algorithm), the vector just before the update occurs is denoted \( \vec{h}(t_k) \) while \( \vec{h}(t_k) \) denotes the vector just after. Formula (2) shows the relationship between the two vectors. The matrix \( \mathbf{A}[t_k] \) and the vector \( \vec{\omega}[t_k] \) characterize the computation performed by the correct nodes (at least one) that have updated their local clocks at time \( t_k \).

\[
\vec{h}(t_k) = \mathbf{A}[t_k] \vec{h}(t_k) + \vec{\omega}[t_k]
\]

The square matrix \( \mathbf{A}[t_k] \) is a state transition matrix and its dimension is \((n - \tau) \times (n - \tau)\). When a correct node \( p_i \) updates its value at time \( t_k \), its new value depends on its previous clock value (corresponding to \( h_i(t_k) \)). The node \( p_i \) uses also transformations of the collected values contained in its log. Neither a timestamp \( C_j \) contained in the log nor its transformation \( C_j^m \) is equal to the value of \( h_j(t_k) \). To compensate the gap between the used estimation values and the real value \( h_j(t_k) \), a compensation vector \( \vec{\omega}[t_k] \) is introduced. Of course, the different values \( a_{ij}[t_k] \) must reflect the weight of the process \( p_i \) in the computation performed by \( p_i \). The Claim 3 states that the definition of such a weighting is possible.

**Claim 3:** Formula 2 can express the clock updating process of a correct node \( p_i \) (1 \( \leq i \leq \tau \)) and \( \mathbf{A}[t_k] \) is a row stochastic matrix with a positive diagonal.

Here we just explain how to construct the matrix \( \mathbf{A} \). We focus on the \( i^{th} \) row \( \mathbf{A}_i \). There are two cases depending on whether \( p_i \) is a correct node which has modified its clock value at time \( t_k \) or not. If \( p_i \) has made no update, we have \( a_{ii}[t_k] = 1 \) and \( a_{is}[t_k] = 0 \) (for all \( s \in [1, \tau] \) and \( s \neq i \)). If \( p_i \) has executed line 11 and changed its value, we have to consider two sub-cases depending on the content of \( N_{eb_i} \). First sub-case: after the reducing operation, all the remaining timestamps in \( N_{eb_i} \) have been provided by correct nodes. Suppose that, after the reducing operation, \( |N_{eb_i}| = n_i \). We use subscript \( s \) (\( s \neq i \)) to represent those correct nodes whose timestamps are still remained and use subscript \( r \) (\( r \neq i \)) to represent other correct nodes. Thus, \( a_{is}[t_k] = \frac{1}{n_i + \tau} \), \( a_{sr}[t_k] = \frac{1}{n_i + \tau} \) and the remaining entrances \( a_{ir}[t_k] \) are all equal to 0. Second sub-case: after reducing some of the remaining timestamps have been provided by Byzantine nodes. As the matrices and vectors focus only on the correct nodes, any value of a Byzantine node used to update the clock value of \( p_i \) has to be expressed as a mix of some values provided by correct nodes. Suppose \( C_{ir}^{max} \) and \( C_{ir}^{min} \) are respectively the maximum and the minimum value among all the timestamps provided by correct nodes and contained in \( N_{eb_i} \) before reducing. Define \( C_{max} = \max\{C_{ir}^{max}, h_i(t_k)\} \) and \( C_{min} = \min\{C_{ir}^{min}, h_i(t_k)\} \). Assume that \( p_i \) is a Byzantine node whose timestamp remains after reducing. Even if \( p_i \) is faulty, its value is safe because \( C_{min} \leq C_{ir}^{b} \leq C_{max} \). Thus \( C_{r}^{b} \) can always be considered as a mix of the values of two correct nodes: \( C_{r}^{b} = \beta_b C_{ir}^{max} + \gamma_b C_{min} \) with \( \beta_b \geq 0 \), \( \gamma_b \geq 0 \) and \( \beta_b + \gamma_b = 1 \). Thanks to these two coefficients, we can now
define \( A_i \). First we have \( a_{ii}[t_k] = \frac{\alpha_i}{n_i+1} \). Three different situations have to be examined to fix the value of \( \alpha_i \): (1) if \( \tilde{h}_i[t_k] \) is equal to \( C_{\text{max}} \), then \( \alpha_i = 1 + \sum_i \beta_i \); (2) if \( \tilde{h}_i[t_k] \) is equal to \( C_{\text{min}} \), then \( \alpha_i = 1 + \sum_i \gamma_i \); (3) if \( \tilde{h}_i[t_k] \) is neither \( C_{\text{max}} \) nor \( C_{\text{min}} \), then \( \alpha_i = 1 \). Second we have \( a_{is}[t_k] = \frac{1}{n_i+1} (s \neq i) \), if the timestamp from \( p_s \) is remained in \( N eb^+ \) after reducing. Third if \( \tilde{h}_j[t_k] (j \neq i) \) is suppressed in line 10, but it is however equal to \( C_{\text{max}} \) (resp. \( C_{\text{min}} \)), then \( a_{ij}[t_k] = \sum_i \beta_i/(n_i+1) \) (resp. \( a_{ij}[t_k] = \sum_b \frac{n_i}{n_i+1} \)). All the other entries are equal to 0. After obtaining the state transfer matrix \( A \), we can easily get the value of the compensation vector \( \vec{\omega} \) (not provided in this article).

To establish the relationship between two consecutive updating operations performed at time \( t_{k-1} \) and \( t_k \), we transform formula (2) into an iterative form. Obviously, the vector value after the first update is not equal to the vector value just before the next update. The underlying clock system also modifies the vector value in-between. Vector \( \vec{d}[t_{k-1}] \) is introduced to represent the increments of the local clock value of each correct node between time \( t_{k-1} \) and \( t_k \). For each correct node \( p_i \), we have \( d_i(t_{k-1}+1) = (t_k - t_{k-1})/(1 \pm \rho_i) \) (where \( \rho_i \) is \( p_i \)'s skew rate). At time \( t_k \), just before the update, \( \tilde{h}[t_k] \) is equal to \( \hat{h}[t_{k-1}] + \vec{d}[t_{k-1}] \). To obtain the relationship between two consecutive updating operations, we transform formula (2):

\[
\tilde{h}[t_k] = A[t_k](\hat{h}[t_{k-1}] + \vec{d}[t_{k-1}]) + \vec{\omega}[t_k]
\]

Now, we derive the relationship between \( \tilde{h}[t_k] \) and \( \hat{h}[t_0] \).

\[
\tilde{h}[t_k] = \hat{h}[t_k] + \vec{d}[t_k] + \vec{\omega}[t_k]
\]

(3)

with the following definitions of the three vectors

\[
\vec{d}[t_k] = \prod_{q=1}^{k} A[t_q] \vec{d}[t_0]
\]

\[
\vec{\omega}[t_k] = \prod_{q=2}^{k} A[t_q] \vec{\omega}[t_1] + \prod_{q=3}^{k} A[t_q] \vec{\omega}[t_2] + ... + \vec{\omega}[t_k]
\]

VI. ANALYSIS

Formula (3) shows that for any pair of correct nodes the deviation of their clock values may have three origins: different initial values (item \( \vec{\omega} \)), different clock drifts (item \( \vec{d} \)) and extra bias due to the estimation process (item \( \vec{\omega} \)). The property of the product of the matrices impacts the final synchrony outcome. Let us first analyze the item \( \vec{\omega} \) and the consequences of having different initial values.

**Lemma 1**: For any vector \( \vec{x} \) of dimension \( \tau \), if \( A[t_q] \) and \( \vec{\omega}[t_k] = \prod_{q=1}^{k} A[t_q] \vec{x}[t_0] \) are two row stochastic matrices then, while \( k \) increases, \( \max_{i,j} |\hat{x}_i - \hat{x}_j| \) is monotone non-increasing.

To prove Lemma 1, we use the property of a row stochastic matrix. Suppose \( x_i[t_k] \) is an element of vector \( \sum_{q=1}^{k} A[t_q] \vec{x}[t_0] \). Thus \( x_i[t_k] = \sum_{q=1}^{k} a_{iq}[t_k] x_q[t_{k-1}] \). Due to \( a_{iq}[t_k] = 1 \) and \( x_q[k-1] \leq x_{\text{max}}[k-1] \), we obtain \( x_i[t_k] \leq x_{\text{max}}[k-1] \). Similarly, \( x_i[k] \geq v_{\text{min}}[t_{k-1}] \). Lemma 1 holds.

**Corollary 1**: \( \max_{i,j} |\hat{h}_i - \hat{h}_j| \) is monotone non-increasing.

Due to Claim 3, all the state transfer matrices are row stochastic. So Lemma 1 allows to conclude that Corollary 1 holds. Due to Corollary 1, the maximum deviation from initial clock values is monotone non-increasing. But this is not sufficient to conclude that this deviation will converge.

Note that convergence is ensured if \( \delta(\prod_{q=1}^{\infty} A[t_q]) = 0 \)

**Lemma 2**: The following implication holds:

\( \delta(\prod_{q=1}^{\infty} A[t_q]) = 0 \Rightarrow \lim_{t_k \rightarrow +\infty} \max_{i,j} |\hat{h}_i[t_k] - \hat{h}_j[t_k]| = 0 \)

The proof of Lemma 2 rely on the results described in Section V-A. When \( \delta(\prod_{q=1}^{\infty} A[t_q]) = 0 \), the product \( \prod_{q=1}^{\infty} A[t_q] \) converges to a matrix in which all the rows are identical. Thus the deviation from different initial clock values will converge towards 0. With the communication and mobility models described in Section III, the following Theorem 2 holds.

**Theorem 2**: \( \delta(\prod_{q=1}^{\infty} A[t_q]) \) approaches 0 with high probability.

Due to the associative law of matrix multiplication, the product can be rewritten as follows:

\( \prod_{q=1}^{\infty} A[t_q] = \prod_{i=1}^{\infty} \Theta_i \)

Each \( \Theta_i \) is a sub product of some state transfer matrices. From Theorem 1, we know that if each \( \Theta_i \) satisfies \( \lambda(\Theta_i) < 1 \), then Theorem 2 holds. Moreover, from the definition of \( \lambda \), the fact that \( \Theta_i \) has a positive column is a sufficient condition to ensure that the property \( \lambda(\Theta_i) < 1 \) holds. Without loss of generality, suppose \( \Theta_i = A[t_{x+k_1}] ... A[t_{x+k}] \). Thanks to the property that each \( A[t_{x+y}] \) \((0 \leq y \leq k_i)\) is a stochastic matrix and has a positive diagonal, it can be inferred that if one \( a_{ij}[t_{x+y}] > 0 (0 \leq y \leq k_i) \), then \( \theta_{ij}[t_x,t_{x+k_i}] > 0 \). We will define a necessary condition such that the property \( a_{ij}[t_{x+y}] > 0 \) is satisfied if the condition holds. First we define the notions of direct use and indirect use of a timestamp. If at time \( t_{x+y} \), \( p_i \) uses the value of timestamp \( C_j \) (or more precisely the corresponding value obtained after transformation) to compute a new value, we say that \( p_i \) uses \( p_j \)'s timestamp directly. \( p_i \) uses \( p_j \)'s timestamp indirectly if \( p_j \)'s information is relayed by one or several correct nodes before \( p_i \) receives a timestamp impacted by
the original timestamp $C_j$. For example, a correct node $p_z$ uses $p_z$’s timestamp directly and after that $p_z$ receives and uses $p_z$’s timestamp directly. Due to the property of the matrix multiplication, it can be inferred that if at time $t_{x+y}$ $p_i$ uses $p_z$’s timestamp indirectly, then $a_{ij}[t_{x+y}]$ is positive. Based on these observations, we define event $\ell$.

**Definition 2:** Event $\ell$ occurs during a sequence of updates if there exists one correct node $p_j$ whose timestamp is used directly or indirectly by all the other correct nodes.

Note that when event $\ell$ happens we obtain a sub product $\Theta$ which has a positive column. Thus $\lambda(\Theta) < 1$.

**Lemma 3:** The fact that event $\ell$ can happen infinitely often is a sufficient condition to ensure that Theorem 2 holds.

**Claim 4:** The probability that event $\ell$ happens infinitely often approaches to 1.

The number of Byzantine nodes is less than $n/3$. Consider the correct nodes whose clock values do not belong to the maximal and minimal values that are discarded during a reducing operation. At most $2f$ values are suppressed during each reducing operation. As the number of correct nodes $n - f$ is greater than or equal to $2f + 1$, there exist at least one correct node which will ensure that event $\ell$ may happen.

Claim 4 implies that the differences between initial clock values have a limited impact on the synchrony outcome. However, as $\hat{d}[t_k]$ and $\overline{d}[t_k]$ are series, Claim 4 is not sufficient to analyze these two types of deviation. Yet, for each item $\hat{d}[t_{1,k-1}]$ (and $\overline{d}[t_k]$), we make the following observations. First, due to Lemma 1, when $t_k$ increases, the maximum deviation is monotone non-increasing. Second if pre-multiply by a sub product $\Theta$ with a positive column, then the maximum deviation significantly decreases. Third the number of matrices contained in $\Theta$ affects this decrement. Each time an event $\ell$ happens, a corresponding $\Theta$ with a positive column can be identified. The size of a $\Theta$ (i.e. the number of matrices it contains) and the frequency at which event $\ell$ happens are strongly related. The frequency of event $\ell$ impacts the deviation from $\hat{d}[t_k]$ and $\overline{d}[t_k]$.

Now let us estimate a possible upper bound for $\hat{d}[t_k]$ and then for $\overline{d}[t_k]$. First, for all $k \geq 1$, under our model, with a high probability there exists a upper bound $d_{\text{max}}$ such that $\max_{i,j} |d_{i}[t_{k,k-1}] - d_{j}[t_{k,k-1}]| \leq d_{\text{max}}$. $d_{\text{max}}$ depends on the frequency of the clock updating operations. With a higher frequency, a lower $d_{\text{max}}$ can be obtained. A similar result can be obtained for $\overline{d}[t_k]$ . With a high probability $\max_{i,j} |\omega_{i}[t_k] - \omega_{j}[t_k]| \leq \omega_{\text{max}}$. $\omega_{\text{max}}$ depends on the frequency of the clock updating operations and the deviation of the initial clock values. In order to cover both cases $\hat{d}[t_k]$ and $\overline{d}[t_k], \supseteq \Theta[t_k, t_2] = \prod_{q=2}^{k} A[t_q]$ and event $\ell$ happens $m$ times from time $t_2$ to time $t_k$. Thus, $\Theta[t_k, t_2] = \prod_{i=1}^{m} \Theta_i$.

Consider the worst case characterized by the maximum deviation of each item $\hat{d}[t_{k,k-1}]$ ($\overline{d}[t_k]$) is $d_{\text{max}}$ ($\omega_{\text{max}}$).

Meanwhile assume $\xi$ is the maximum number of matrices contained in $\Theta_i$ (1 $\leq i \leq m$) and $\zeta$ ($\zeta < 1$) denotes the proportional decrement factor of a pre-multiply by a product $\Theta_z$. For $\hat{d}[t_k]$, starting from item $\hat{d}[t_{k,k-1}]$ to $\hat{d}[t_{1,0}]$, at most after each $\zeta$ items the deviation decreases to $d_{\text{max}}$. Such that the worst case is $\max_{i,j} |\hat{d}[t_{k}] - \hat{d}_{j}[t_{k}]| \leq \frac{1}{1 - \zeta} d_{\text{max}}$. A similar result $\max_{i,j} |\overline{d}[t_{k}] - \overline{d}_{j}[t_{k}]| \leq \frac{1}{1 - \zeta} \omega_{\text{max}}$ can also be obtained on $\overline{d}[t_k]$.

**VII. SIMULATION**

We consider a simulation scenario where the $n$ nodes move on a predefined graph $G = (V, E)$ represented by a square grid of size $s \times s$ ($|V| = s \times s$). Initially each node is located on an vertex randomly selected. According to our mobility model (See Section III-A), each node moves independently. When a node arrives in a vertex $v_x$, it chooses its next destination randomly its next destination $v_y$ such that $(v_x, v_y)$ belongs to $E$. We suppose that Byzantine nodes have no stronger moving abilities. Moving outside the graph is forbidden, when a node is on a vertex which is at a corner or on a border, it has fewer possibilities for its next destination. Otherwise, it has always four choices.

The simulation is based on an open source platform The one [18] version 1.4.1, which is currently prevalent in DTN simulations. Each correct node receives an initial clock value and all the nodes start the simulation at the same time. When two nodes $p_i$ and $p_j$ are located within the same communication range, they can exchange messages. Correct nodes follow exactly the protocol. A Byzantine node $p_b$ also follows the algorithm to update its clock time, but it can send out a fake timestamp, which is computed by $C_b \pm r$ ($r$ is a random value). The drift rate $\rho_i$ of node $p_i$ is selected randomly from the interval $[9 \times 10^{-6}, 9 \times 10^{-6}]$ and the message passing delay between two neighbors is selected randomly from the range $[1 \times 10^{-3}, 1 \times 10^{-2}]$. The losses of messages have not been simulated. Each simulation ends after 43200 steps. During the simulations two major results are measured. One is the maximum differences among all the correct nodes’ clocks. And the other is the occurrence time interval of event $\ell$: when $\ell$ happens, we log the time that has passed since the previous occurrence of $\ell$.

We analyze the convergence speed when different initial clock values exist. The grid size is fixed to $10 \times 10$ (100 vertexes) and the length of each edge is set to 70 (thus, the size of the area is $700 \times 700$). First we progressively increase the broadcasting period from 10 time units to 40 time units. 7 nodes (including 2 Byzantine nodes) are moving in the grid. The communication range is equal to 50. All nodes have a same moving speed which is fixed to 25 per time unit. The maximum difference between initial clock values is set to 80. When a Byzantine node sends out its timestamp, it adds a random number $r$ ($C_b + r, r \in [-30, 30]$). The simulation outcome is in Figure 4. As the broadcasting period increases,
correct nodes have fewer chances to compute a new clock value, such that the convergence procedure slows down.

![Figure 4. Impact of the broadcasting period](image)

Other factors such as the moving speed, the size of the communication range and the number of nodes have also a strong impact. In particular, when the total number of nodes increases while the number of byzantine nodes remains stable, the convergence is more faster. Other simulations show that when event $\ell$ happens more frequently, the synchrony outcome is also more accurate.

**VIII. CONCLUSION**

We have addressed the clock synchronization problem in wireless mobile ad hoc networks in the presence of Byzantine nodes. Based on the linear consensus method, a synchronization algorithm has been proposed. The proposed sufficient condition impacts the accuracy of the synchrony outcome. The theoretical analysis and the simulation results show the interest of the event $\ell$ defined in this paper.

**ACKNOWLEDGMENT**

This work is partially supported by Natural Science Foundation, China under grant 60973122, by National 863 Hi-Tech Program, China under grant 2011AA040502 and by the ANR INS French program (grant #ANR-11-INSE-010, project AMORES).

**REFERENCES**


