# Towards a Theory for Securing Time Synchronization in Wireless Sensor Networks

Murtuza Jadliwala
Laboratory for computer Communications and Applications (LCA)
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, CH-1015, Switzerland
murtuza.jadliwala@epfl.ch

Qi Duan, Shambhu Upadhyaya, and Jinhui Xu
Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY 14260, USA
{qiduan, shambhu, jinhui}@cse.buffalo.edu

## ABSTRACT

Time synchronization in highly distributed wireless systems like sensor and ad hoc networks is extremely important in order to maintain a consistent notion of time throughout the network and to support the various timing-based applications. But, cheating behavior by the participating nodes in the network can severely jeopardize the accuracy of the associated time synchronization process. Despite recent advances in this direction, a key fundamental question still remains unanswered: Is it theoretically feasible to secure distributed time synchronization protocols, given complete (or global) time and time difference information in the network?

In this paper, we attempt to answer this question with the help of sound mathematical modeling and analysis. We first formulate the problem of distributed time synchronization as a *Constraint Satisfaction Problem (CSP)* in a graph-based model of the network. Then, we prove that efficiently eliminating cheating behavior in distributed time synchronization protocols is combinatorially hard ($NP$-hard), i.e., it is highly unlikely that there exists an algorithm that solves, or even approximates, this problem in polynomial (in terms of total number of nodes) time. Due to this negative result for the general case, we focus on studying the problem for a special case of the graph-based model of the network, namely completely connected graphs. We derive an upper bound on the best possible solution quality for this problem, propose two polynomial-time approximation strategies, and present an empirical evaluation of their performance.

## Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Network problems*

## General Terms

Algorithms, Security, Theory

## Keywords

Approximation Algorithms, Time Synchronization, Wireless Sensor Networks


*WiSec'09,* March 16–18, 2009, Zurich, Switzerland.


## 1. INTRODUCTION

Wireless ad hoc networks such as sensor networks are fast gaining popularity for a variety of outdoor monitoring and emergency response applications including environment and climate monitoring, forest fire monitoring, target tracking, monitoring enemy movement during wars and intrusion detection systems. Due to the critical nature of these applications, knowledge of the exact order and time of occurrence of the monitored events is extremely essential. In order to maintain time, each mote is normally equipped with an on-chip local clock (generally, a quartz crystal oscillating at a specific frequency), which at the time of deployment is synchronized with the other motes in the network. Thus, at the beginning all the nodes[1] in the network have the same notion of time. But over a period of time, motes in the network develop varying notions of time due to various network and mote dependent factors such as fading battery power, longer sleep periods and mote crystals oscillating at different frequencies.

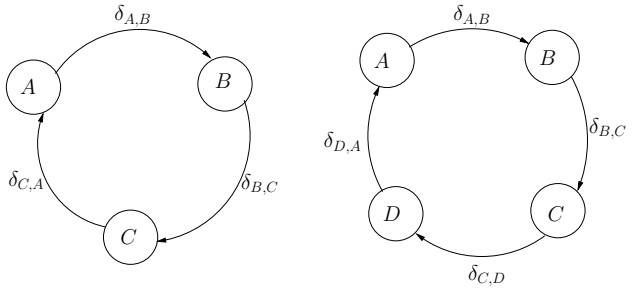
The process of updating the local clocks of each mote in the network such that all the motes have the same notion of time is referred to as *time synchronization*. Time synchronization can be either *absolute* or *relative*. In absolute synchronization, the clocks of all the motes are adjusted to a real-time standard like UTC (Universal Coordinated Time) or some other well-known global value. In relative synchronization, such a global or standard value of time is not known, and the nodes have to be synchronized relative to each other. Several efficient protocols for time synchronization in infrastructure-based computer networks exist in the

[1]The term mote and node are used interchangeably

literature [32]. Cristian's Remote Clock Reading method [4], Arvind's Time Transmission Protocol (TTP) [1], Set-valued Estimation method [18] by Lemmon et al. and Mill's Offset Delay Estimation method employed by the Network Time Protocol (NTP) [22] are a few example of such protocols. These protocols estimate the time offset between the target machine (that needs to synchronize its time) and a source machine (generally a time-keeping server or a machine that has the correct notion of time) using simple message transmissions and time stamping, and use this estimated time difference to determine the local time of the target. But, due to factors such as lack of infrastructure, limited power, limited bandwidth, limited hardware and non-deterministic delays at the MAC layer in sensor motes, the above schemes cannot be directly applied to sensor networks.

Depending on the application, either relative or absolute synchronization may be required by the network. But, for most sensor network applications relative synchronization is generally sufficient. Relative time synchronization algorithms for wireless sensor networks can be further divided into two broad types: *sender-receiver* synchronization and *receiver-receiver* synchronization [32]. In sender-receiver synchronization [10, 25, 20] the nodes synchronize themselves with respect to a sender or beacon node. The sender node periodically sends a message (beacon) with its local time to the receiver. The receiver then synchronizes with the sender using the local time-stamp it receives from the sender. The message delay between the sender and the receiver is calculated by measuring the total round-trip time, from the time a receiver requests a time-stamp until the time it actually receives a response. In the receiver-receiver synchronization [5, 29, 21, 33], instead of interacting directly with a sender, receivers exchange with each other the time at which they received the same message from the same sender, and compute the time offset between them based on the difference in reception times. One thing that is common to all the time synchronization techniques mentioned above is that each mote that wants to synchronize itself, attempts to estimate the time offset (difference) to its neighboring nodes. The number and type of neighboring nodes that the target node estimates time differences to, and the technique used to estimate these time offsets vary and depend on the individual schemes.

In the case when all the nodes behave honestly during time synchronization, all the estimated time differences in the network should follow the *triangle law*, as shown in Figure 1. Figure 1(a) shows that if nodes $A$, $B$ and $C$ honestly




(a) $\delta_{A,B} + \delta_{B,C} + \delta_{C,A} = 0$ (b) $\delta_{A,B} + \delta_{B,C} + \delta_{C,D} + \delta_{D,A} = 0$

**Figure 1: Triangle law for time offset**

compute their time offsets and if $\delta_{A,B}$ is the time offset of node $A$ with respect to node $B$, and $\delta_{B,C}$ is the time offset of node $B$ with respect to node $C$, and so on, then the sum of the time differences $\delta_{A,B}$, $\delta_{B,C}$ and $\delta_{C,A}$ is zero. This observation was also used by Ganeriwal et al. [9] in their design of secure group synchronization protocol. (In practice, this sum should be less than some constant $\sigma$, which denotes an upper bound on the measurement error in the network. Currently, we do not focus on modeling the effects of measurement errors on time synchronization. Thus, the current exposition is simplified by assuming no measurement error. Although the proposed network model assumes a zero measurement error, the related results are general enough and also hold for a non-zero measurement error.) In other words, if the sum of the time differences is non-zero for a set of nodes in the network then it implies that at least one of the nodes in this set cheats or drastically deviates from correctly executing the protocol. Such *inconsistencies* in time difference estimates between nodes can severely jeopardize the accuracy of the time synchronization protocol associated with the network. In order to secure time synchronization protocols, one needs to eliminate in an efficient fashion such inconsistent time offset data that is introduced by the cheating or byzantine behavior of the participating nodes. To decide if this problem is feasible or not, is the crux of this paper.

In order to gain a better understanding of the problem of eliminating inconsistent time offset data, a rigorous mathematical formulation of the time synchronization problem itself is extremely crucial. Such a formulation is useful in understanding both the combinatorial and the security related properties of the time synchronization problem. In this direction, we first formulate the time synchronization problem as a *Constraint Satisfaction Problem (CSP)* in a special type of graph-based model of the network, called the *time difference graphs*. We next prove an intuitive but fundamental result, which states that the time synchronization problem has a solution (in this model) if and only if the associated time difference graph is consistent, i.e., there exist no inconsistent time offset estimates in the graph. We refer to a time difference graph containing inconsistent time offset values as a *partially consistent time difference graph*. The problem of eliminating inconsistent time offset values (or cheating behavior) can then be modeled as an optimization problem that determines the largest consistent subgraph of the corresponding partially consistent time difference graph of the network. We also refer to this problem as the *Maximum Consistent Time Difference Graph* or *MCTD*. We prove that it is combinatorially infeasible to solve the MCTD problem and show that it is unlikely to even have a polynomial-time approximation algorithm. Due to this negative result for the general case, we study the MCTD problem for a restricted version of time difference graphs, namely completely connected time difference graphs. We prove that even in this case, the MCTD problem is combinatorially hard. But unlike the previous case, there exist polynomial-time approximation algorithms for the completely connected graphs. We prove that the best possible solution quality of these algorithms is bounded by $n^{\frac{1}{2}-\epsilon}$, for some constant $\epsilon > 0$, where $n$ is the number of nodes. We also propose two heuristics for this problem. The first heuristic is based on a greedy selection strategy, while the second is a linear programming based optimization technique. In order to verify the practical efficiency and performance of these heuristics, an empirical evaluation is also presented towards the end.

## 1.1 Background and Related Work

The problem of time synchronization in the presence of malicious motes was first studied by Ganeriwal et al. [9]. They outlined a few attacks on existing time synchronization schemes, including the pulse delay attack, where an adversary deliberately delays the transmission of synchronization messages in order to magnify the time offset between it and the neighboring nodes. The authors also proposed secure single-hop, multi-hop and group time synchronization protocols for wireless sensor networks. Song et al. [30] also proposed two approaches to detect and accommodate the delay attack. The authors proposed schemes that make use of the fact that if there are no malicious motes then the time offsets among the sensor motes should follow the same (or similar) distribution. Sun et al. [31] also proposed secure and resilient pairwise and global time synchronization protocols that use authenticated MAC layer time-stamping and the $\mu$-TESLA broadcast authentication protocol to overcome attacks by malicious motes. The authors claim that their scheme is secure against compromised nodes and external attacks such as sybil attacks. Li et al. [19] proposed a secure time synchronization protocol that verifies the synchronization process between each synchronizing node pair using the node's neighbors as verifiers. Recently, Xianglan et al. [34] proposed a secure light-weight scheme that eliminates the requirement of loose synchronization in the $\mu$-TESLA broadcast authentication protocol.

In summary, all the above schemes only viewed the problem of secure time synchronization from a local or a per-node perspective, which is completely different when observed from a network-wide standpoint. For a large wireless sensor network consisting of thousands of nodes, a set of time offset values might be consistent within a neighborhood (or group) of nodes, but might not be consistent with the rest of the network (or other groups). This issue is more pronounced in de-centralized or infrastructureless time synchronization protocols such as the ones used for sensor networks. We could not find any concrete feasible solution in the literature for securing time synchronization from a network-wide perspective. Although Ganeriwal et al. [11] proposed a group synchronization protocol, their solution works well only for small groups of nodes with a limited number of adversarial nodes. Thus, it is very important to solve the secure time synchronization problem from a global perspective in order to achieve network-wide security. In addition to these issues, we discovered that an elegant formal treatment of the time synchronization problem itself was missing in the literature. Such a formal treatment is very useful in deriving guarantees and fundamental limits on the level of security that can be achieved, and the cost for achieving that. In this work, we attempt to address these unanswered questions.

## 1.2 Paper Organization

In Section 2, we outline the network and adversary model, introduce the notion of time difference graphs and formulate the problem of time synchronization as a CSP. In Section 3, we state the MCTD problem for general time difference graphs and derive the related combinatorial results. In Section 4, we analyze the MCTD problem for completely connected time difference graphs and present two approximation strategies for it. In Section 5, we discuss the initial simulation results. We conclude the paper with a summary of results in Section 6.

# 2. MATHEMATICAL FORMULATION

We begin by providing a brief overview of some important concepts and terminology in complexity theory that will be used throughout this paper. More details on these topics can be found in [13].

## 2.1 Preliminaries: Complexity Theory

$NP$ is the class of decision problems that have efficiently *verifiable proof systems*. A decision problem $S \subseteq \{0,1\}^*$ has an efficiently verifiable proof system if there exist a polynomial $p$ and a polynomial-time verification algorithm $V$ such that the following two conditions hold:

- Completeness: For every $x \in S$, there exist $y$ of length at most $p(|x|)$ such that $V(x,y) = 1$.
- Soundness: For every $x \notin S$ and every $y$, it holds that $V(x,y) = 0$.

A polynomial-time computable function $f$ is called a *karp-reduction* of $S$ to $S'$ if, for every $x$, it holds that $x \in S$ if and only if $f(x) \in S'$. In this case, $S$ is said to be *many-one reducible* (or karp reducible) to $S'$ in polynomial time. A set $S$ is *$NP$-complete* if it is in $NP$ and every set in $NP$ is karp-reducible to it. A set $S$ is *$NP$-hard* if every set in $NP$ is karp-reducible to it, but its membership within $NP$ is not known. It is not known whether every problem in $NP$ can be efficiently solved (in polynomial time). But, if any single problem in the set of $NP$-complete problems can be solved efficiently, then every problem in $NP$ can also be solved efficiently. Thus, $NP$-complete problems are considered "harder" than $NP$ problems in general, and are believed to have no polynomial-time (efficient) exact solutions. Algorithms for such hard problems, also called *optimization problems*, that run in polynomial time and produce a near-exact or sub-optimal solution are called *approximation algorithms*. Approximation algorithms that can guarantee that the solution output by it can be no more (if minimization problem) or less (if maximization problem) than a factor $\sigma$ times the optimum solution is called a *$\sigma$-approximation algorithm* for that problem.

In this paper, we model the problem of securing distributed time synchronization protocols in wireless sensor networks as an optimization problem. In the following section, we first outline the network model for this problem.

## 2.2 Network Model

Let $N = \{1, 2, \ldots, n\}$ denote the set of $n$ motes in the network. Let $P = (p_1, p_2, \ldots, p_n)$ be the *local clock vector* such that each $p_i$ is a function of time $p_i : t \rightarrow \mathbb{R}^+$, and gives the local clock value on the mote $i$ at any instant in time $t$. At the time of network deployment, i.e., at $t = 0$, $p_i(t) = p_j(t)$, $\forall i, j \in N$, but this equality ceases to hold with time due to factors such as clock drift, clock skew, etc. [32, 23]. Now, a time difference graph, $\mathbb{G} = (\mathbb{V}, \mathbb{E}, w, \delta)$, for the network at any instant in time can be defined as follows (see Figure 2). The set $\mathbb{V} = \{v_i | i \in N\}$ contains a vertex corresponding to each *operating* mote in the network at that instant. A *directed edge* $(v_i, v_j) \in \mathbb{E}$ exists between two vertices $v_i$ and $v_j$ in the graph $\mathbb{G}$ if and only if $i$ and $j$ are *neighbors* of each other, and the direction of the edge depends on the role of the nodes during time difference estimation. This is discussed shortly. But first, by "neighbors" we mean that both $i$ and $j$ are in the radio range of each other, and can

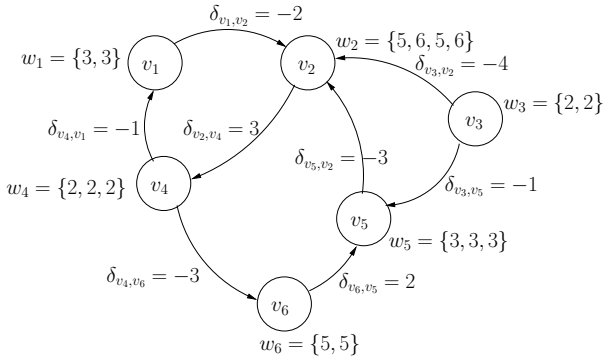



**Figure 2: Time difference graph**

communicate with each other directly. We assume that each mote is able to securely determine its neighborhood [24].

Assuming that the degree (combined in and out degree) of a vertex $v_i$ is denoted by $d_i$, the graph $\mathbb{G}$ is associated with a *clock function* $w$ that assigns a vector $w_i = \{\mathbb{R}^+\}^{d_i}$ of $d_i$ real values to each vertex $v_i$ in the graph. Each scalar value $w_{v_i,v_j}$ in this vector signifies the local clock value *advertised* by $i$ to each of its neighbors $j$ at time $t$. Ideally, for every vertex $v_i \in \mathbb{V}$, $w_i = \{p_i(t)\}^{d_i}$, i.e., the node should advertise its actual local clock value to each of its neighboring node. But in reality, this equality may or may not hold depending on whether the node is honest or not about advertising its actual local clock value, i.e., every scalar value in the clock function vector $w_i$ may not be identical and not necessarily equal to the node $i$'s actual local clock value $p_i(t)$. For example, as shown in Figure 2, the clock function vector for node 2 is $w_2 = \{w_{v_2,v_1} = 5, w_{v_2,v_3} = 6, w_{v_2,v_4} = 5, w_{v_2,v_5} = 6\}$, i.e., node 2 advertises a clock value of 5 to nodes 1 and 4, while a clock value of 6 to nodes 3 and 5. We shall discuss this in detail later in Section 2.4. It is important to note here that for any node $i$, $w_{v_i,v_j}$ is known only to $j$ and not to any other neighbors of $i$. Now, in each pair of neighboring nodes, one node acts as a *sender* node and the other node is a *receiver*, and the receiver node always computes its time difference with respect to the sender. Thus, if $i$ is a sender node and $j$ is a receiver node then the edge $(v_i, v_j)$ is directed from $v_i$ to $v_j$. Intuitively it may seem that such a representation only models the sender-receiver type of time synchronization protocols, but the current graph-based model is very general and also captures the receiver-receiver type of protocols. In the receiver-receiver type of protocols, two nodes estimate the time offset with respect to each other by exchanging the receipt times of a packet from a common node. So in the current model, these two nodes will be modeled both as a sender and a receiver, and there will be a directed edge between the nodes in both directions.

Each edge $(v_i, v_j) \in \mathbb{E}$ is associated with a real weight $\delta_{v_i,v_j}$, which is nothing but the estimated time offset (difference) computed by the receiver to the sender. Formally, the graph $\mathbb{G}$ is associated with a *time difference function* $\delta$, $\delta : \mathbb{E} \rightarrow \mathbb{R}$, such that $\delta$ assigns a weight to each edge in the graph signifying the *estimated* value of the time difference between the two vertices (motes) connected by that directed edge. The weight $\delta_{v_i,v_j}$ is *positive* ($> 0$) if the receiver node *lags* the sender node, $\delta_{v_i,v_j}$ is *negative* ($< 0$) if the receiver node *leads* the sender node, and $\delta_{v_i,v_j}$ is zero if there is no time difference between the receiver and the sender node. We also assume here that the time difference function $\delta$ can be efficiently computed. An example of one such efficient implementation of the time difference function was given by Ganeriwal et al. [10], where the receiver computes the time difference by receiving time stamped messages from the sender. In order to simplify the current exposition, we assume here that the graph $\mathbb{G}$ is a simple, connected, directed graph, i.e., there are no self loops and every vertex is reachable from every other vertex through a sequence of edges. In the following section, we present a formulation of the time synchronization problem, given the time difference graph model of the network.

## 2.3 The Time Synchronization Problem

Given a time difference graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, w, \delta)$ of the network, as defined above, the problem of time synchronization can be formulated as a Constraint Satisfaction Problem (CSP). Suppose that $\|\mathbb{V}\| = n$, i.e., all motes are operational. We define a set of $n$ variables $x_1, x_2, \ldots, x_n$, one for each vertex $v_i \in \mathbb{V}$ in $\mathbb{G}$. These variables are also referred to as *adjustment variables* [28]. Each adjustment variable $x_i$ has a non-empty domain, which is the set of real numbers $\mathbb{R}$. Each directed edge $(v_i, v_j) \in \mathbb{E}$ defines two constraints. The first constraint, denoted as $C_{1(v_i,v_j)}$, gives the relationship between $x_i$, $x_j$ and the time difference function $\delta$, and is given as

$$C_{1(v_i,v_j)} \equiv x_j - x_i = \delta(v_i, v_j) \tag{1}$$

The second constraint, denoted as $C_{2(v_i,v_j)}$, gives the relationship between $x_i$, $x_j$ and the clock function $w$,

$$C_{2(v_i,v_j)} \equiv w_{v_i,v_j} + x_i = w_{v_j,v_i} + x_j \tag{2}$$

Thus, there are a total of $2 \cdot \|E\|$ constraints in the system. The *state* of the above CSP is defined by an *assignment* $x_1 = m_1, x_2 = m_2, \ldots, x_n = m_n$, where $m_i \in \mathbb{R}$, to some or all the adjustment variables. An assignment that does not violate any constraints is called a *consistent* or *legal* assignment. A *complete assignment* is one in which every variable is mentioned, and a *solution* to the CSP is a complete assignment that satisfies all the constraints. Given a time difference graph, $\mathbb{G} = (\mathbb{V}, \mathbb{E}, w, \delta)$, the problem of time synchronization then reduces to determining a complete consistent assignment to the adjustment variables $x_1, x_2, \ldots, x_n$. Intuitively, $x_1 = m_1, x_2 = m_2, \ldots, x_n = m_n$ represents the values by which the clocks of the nodes $1, 2, \ldots, n$ must be adjusted such that all the nodes have the same notion of time. The problem of time synchronization, as formulated above, is *feasible* or *solvable* if such a complete assignment exists. If such a complete assignment does not exist then it implies that some of the constraints cannot be satisfied, and time synchronization is *infeasible* or *partially feasible*. This infeasibility can be due to the cheating or byzantine behavior of the participating nodes, as discussed in the following section.

## 2.4 Adversary Model and Inconsistencies in Time Synchronization

From the point of view of time synchronization, an honest node always accurately advertises its own local clock values and accurately estimates the time difference to its neighboring nodes. On the other hand, nodes can also cheat during time synchronization, as discussed below.

1. Advertise incorrect local clock value: As discussed before, each node advertises its local clock value to each of its neighboring nodes during time difference estimation. A node can cheat by advertising incorrect local clock information to its neighboring nodes. Such a cheating behavior translates to the clock function $w$ assigning incorrect weight vectors to the vertices in the corresponding time difference graph model $\mathbb{G}$ of the network, i.e., $w_{v_i,v_j} \neq p_i(t)$ for some node $i$ and its neighbor $j$.

2. Manipulating time difference estimation: During time synchronization, a cheating node can also manipulate message transmissions, for example, by introducing unnecessary delays or changing packet time stamps. This may affect the time difference estimation process and translate to the time difference function $\delta$ assigning incorrect weights to the edges in the corresponding time difference graph model $\mathbb{G}$.

There are some important observations that we make at this point. First, the advertised local clock vector $w_i$ of an honest node $i$ always follows the equality $w_i = \{p_i(t)\}^{d_i}$, while the local clock values advertised by a cheating node are arbitrarily chosen by the adversary. Also, the time difference function $\delta$ depends on the clock function $w$ for time difference estimation, for example [10]. As a result, any cheating by a node in the advertised local clock value also translates to an incorrect time difference value between the node and its corresponding neighbors. Moreover, from Equations (1) and (2), we can see that in order to successfully mislead the time synchronization protocol, the cheating node has to maintain consistency between its advertised local clock value and the estimated time offset between it and its neighboring node. Otherwise, it is trivial for a neighboring node to observe the inconsistency between the advertised clock value and the estimated time difference, and as a result, detect the cheating node. In this work, we assume that the adversary is smart and wants to avoid trivially being detected by its neighbors. The adversary will make sure that its advertised local clock value is consistent with the time difference estimate with each neighbor. In other words, cheating on the local clock value translates into a corresponding manipulation of the time difference estimate and vice versa. In either case, if a node cheats with some neighboring node, its association with the other nodes will fail the triangle law verification, as discussed in Section 1. We refer to this as an inconsistency and is described more formally next.

Before moving ahead, we would like to review some important definitions.

*Definition 1. A Cycle or Circuit* in a graph is an alternating sequence of vertices and edges, with each edge being incident to the vertices immediately preceding and succeeding it in the sequence such that all the vertices in the sequence are distinct except the first and the last.

A *directed cycle* is a directed version of the cycle, with all the edges being oriented in the same direction. In this paper, whenever we refer to a cycle, it will always imply a directed cycle. Also, here we assume *simple cycles*, i.e., a cycle with no repeated vertices except the first and the last vertex. Now, recall from Section 1 that the triangle law for time offset (time difference) outlined the necessary condition for time difference consistency for a group of three nodes. Let us present a more general notion for this condition, called a *consistent cycle*.

*Definition 2.* Given a time difference graph, $\mathbb{G} = (\mathbb{V}, \mathbb{E}, w, \delta)$, any cycle of $\mathbb{G}$ consisting of three or more vertices is called a *consistent cycle* if and only if the sum of the time difference function values $\delta_{v_i,v_j}$ of all the edges $(v_i, v_j)$ in the cycle is exactly zero.

A cycle of three or more vertices in which the sum of the time difference function values for all the edges is anything except zero (positive or negative) is called an *inconsistent cycle*. A time difference graph that contains no inconsistent cycles is called a *consistent time difference graph*. A time difference graph that contains inconsistent cycles is called an *inconsistent* or *partially consistent time difference graph*. A similar notion was used by Jadliwala et al. in [15] to represent inconsistencies in localization systems.

Cheating behavior by nodes during time synchronization can lead to an inconsistent or partially consistent time difference graph. But, one problem with the current definition of time difference graphs is the presence of *connected acyclic loops*, as defined in the following section, which are also possible in such directed time difference graphs. Graph consistency, as defined above, considers the consistency of only all (directed) cycles, and none of the connected acyclic loops are verified for consistency. We overcome this problem by adding redundancy, as discussed in the following section. A time difference graph that does not have any directed cycles or connected acyclic loops is always consistent.

## 2.5 Time Difference Graphs - Revisited

In a directed graph such as the time difference graph (from Section 2.2), it is possible that some vertex combinations are connected by edges but do not constitute a (directed) cycle because all these edges are not oriented in the same direction. We call such vertex combinations as *connected acyclic loops*. An example of such an acyclic loop $(v_2, v_3, v_5)$ in a time difference graph is shown in Figure 2. Due to such acyclic loops, it is not possible to determine the consistency of the time difference graph just based on the consistency of all the directed cycles. In order to overcome this problem, we add some redundancy to our initial definition of time difference graphs. Given a time difference graph, $\mathbb{G} = (\mathbb{V}, \mathbb{E}, w, \delta)$, we define a new graph $\mathbb{G}' = (\mathbb{V}', \mathbb{E}', w', \delta')$ as follows. The vertex set $\mathbb{V}'$ and the clock function $w'$ of $\mathbb{G}'$ are the same as that of $\mathbb{G}$. For each edge $(v_i, v_j) \in E$, $(v_i, v_j) \in E'$ and $(v_j, v_i) \in E'$. Also, if $(v_i, v_j) \in E$ and $\delta(v_i, v_j) \neq 0$ then $\delta'(v_i, v_j) = \delta(v_i, v_j)$ and $\delta'(v_j, v_i) = -1 \times \delta(v_i, v_j)$. If $\delta(v_i, v_j) = 0$ then $\delta'(v_i, v_j) = \delta'(v_j, v_i) = 0$. This new graph does not add any new information to the time difference graph, and is referred to as a *Redundant Time Difference Graph*. A redundant time difference graph for the time difference graph of Figure 2 is shown in Figure 3. As we can see from Figure 3, for each edge in the time difference graph, a new edge between the same pair of nodes in the opposite direction is added in the corresponding redundant time difference graph. This edge is called the *redundant edge* as its time difference value is just opposite in sign to the time difference value of the actual edge and it does not provide any new information. This is also intuitive because if a node, say $A$, lags another node, say $B$, by $m$, then the same thing can also be interpreted as $B$ leads $A$ by $m$. Redundant time difference graphs eliminate acyclic loops in time difference

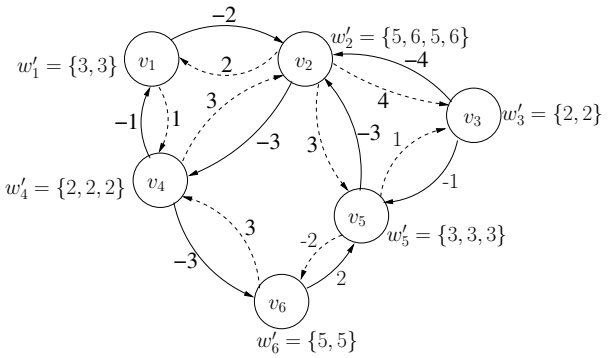



**Figure 3: Redundant time difference graph**

graphs by converting each such loop into a directed cycle that can be verified for consistency. This brings us to our first result that gives the relationship between the solution of the time synchronization problem and the consistency of the redundant time difference graph.

PROPOSITION 2.1. *The time synchronization problem for a redundant time difference graph $\mathbb{G}' = (\mathbb{V}', \mathbb{E}', w', \delta')$ has a solution if and only if $\mathbb{G}'$ is consistent.*

PROOF. We first prove the reverse direction. We will show that if the graph $\mathbb{G}'$ is consistent then the time synchronization problem for $\mathbb{G}'$ has a solution. In other words, if the graph $\mathbb{G}'$ is consistent then the CSP for the time synchronization problem has at least one assignment to the adjustment variables $x_1, x_2, \ldots, x_n$ such that all the constraints (Equations (1) and (2)) are satisfied. We prove this by a contradiction argument. Since the graph $\mathbb{G}'$ is consistent, by definition of consistency all simple cycles in $\mathbb{G}'$ are consistent, i.e., sum of time difference values of all edges in each cycle is zero. Now, let $x_1 = m_1, x_2 = m_2, \ldots, x_n = m_n$ be one assignment to the adjustment variables such that there is at least one constraint that is not satisfied. Let this constraint (which is not satisfied) be on the edge $(v_i, v_{i+1})$, i.e., $x_{i+1} - x_i \neq \delta(v_i, v_{i+1})$. Thus, $\delta(v_i, v_{i+1}) \neq m_{i+1} - m_i$. Without loss of generality, assume that $\delta(v_i, v_{i+1}) < m_{i+1} - m_i$. Also, let this edge be on some cycle $\mathbb{C}' = (v_1, v_2), (v_2, v_3), \ldots, (v_i, v_{i+1}), (v_{i+1}, v_{i+2}), \ldots (v_{i+r}, v_1)$. Now, since $\mathbb{G}'$ is consistent, the cycle $\mathbb{C}'$ is consistent. Thus,

$$
\begin{aligned}
& \delta(v_1, v_2) + \delta(v_2, v_3) + \ldots + \delta(v_{i-1}, v_i) + \\
& \delta(v_i, v_{i+1}) + \delta(v_{i+1}, v_{i+2}) + \ldots + \delta(v_{i+r}, v_1) = 0 \\
\implies \quad & (x_2 - x_1) + (x_3 - x_2) + \ldots + (x_i - x_{i-1}) + \\
& \delta(v_i, v_{i+1}) + (x_{i+2} - x_{i+1}) + \ldots + (x_1 - x_{i+r}) = 0 \\
\implies \quad & (m_2 - m_1) + (m_3 - m_2) + \ldots + (m_i - m_{i-1}) + \\
& \delta(v_i, v_{i+1}) + (m_{i+2} - m_{i+1}) + \ldots + (m_1 - m_{i+r}) = 0 \\
\implies \quad & m_i - m_{i+1} + \delta(v_i, v_{i+1}) = 0 \\
\implies \quad & \delta(v_i, v_{i+1}) = m_{i+1} - m_i
\end{aligned}
$$

which is a contradiction. This proves the reverse direction.

Now for the forward direction, let us assume that the time synchronization problem for the graph $\mathbb{G}'$ has a solution, i.e., the CSP formulation has at least one assignment to the adjustment variables, $x_1 = m_1, x_2 = m_2, \ldots, x_n = m_n$, such that all the constraints (Equations (1) and (2)) are satisfied. Also, assume that $\mathbb{G}'$ is not consistent. Without loss of generality, let $\mathbb{C}' = (v_1, v_2), (v_2, v_3), \ldots, (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{i+1}, v_{i+2}), \ldots (v_{i+r}, v_1)$ be the cycle such that

$$
\begin{aligned}
& \delta(v_1, v_2) + \delta(v_2, v_3) + \ldots + \delta(v_{i-1}, v_i) + \\
& \delta(v_i, v_{i+1}) + \delta(v_{i+1}, v_{i+2}) + \ldots + \delta(v_{i+r}, v_1) \neq 0
\end{aligned}
$$

But, this implies that,

$$
\begin{aligned}
& (x_2 - x_1) + (x_3 - x_2) + \ldots + (x_i - x_{i-1}) + \\
& (x_i, x_{i+1}) + (x_{i+2} - x_{i+1}) + \ldots + (x_1 - x_{i+r}) \neq 0
\end{aligned}
$$

i.e.,

$$
\begin{aligned}
& (m_2 - m_1) + (m_3 - m_2) + \ldots + (m_i - m_{i-1}) + \\
& (m_i, m_{i+1}) + (m_{i+2} - m_{i+1}) + \ldots + (m_1 - m_{i+r}) \neq 0
\end{aligned}
$$

which is a contradiction. Thus, there can be no such cycle. Thus, $\mathbb{G}'$ is consistent. Thus, the proof. □

Readers should note that from this point onwards the term time difference graph in this paper would always imply a redundant time difference graph. Given the time difference graph model for time synchronization and the adversary model, we now focus on the problem of securing time synchronization in the following sections.

# 3. ELIMINATING INCONSISTENCIES

Up to this point, we have formulated the distributed time synchronization problem as a constraint satisfaction problem in a graph-based model of the network, called time difference graphs, and proved that a solution to the time synchronization problem exists if and only if the corresponding time difference graph is consistent. We have also established that cheating behavior by nodes during time synchronization results in inconsistencies in the corresponding time difference graph model of the network. Thus, in order to secure the time synchronization process in the network, the first step would be to eliminate these inconsistencies in an efficient fashion. In other words, we need to address the problem that given a partially consistent time difference graph, how to obtain the largest consistent subgraph of this graph. This can be formulated as a (graph-based) optimization problem, as formally stated next.

## 3.1 Maximum Consistent Time Difference Graph (MCTD) Problem

A consistent subgraph of a partially consistent time difference graph is obtained by eliminating vertices (and the corresponding edges) until the resulting induced subgraph is consistent, i.e., all simple cycles in the induced subgraph are consistent. The *size* of the consistent subgraph is the cardinality of its vertex set. The *edge size* is the cardinality of its edge set. A consistent subgraph is *maximal* if its vertex set is not a proper subset of the vertex set of any other consistent subgraph of that time difference graph. A *maximum* consistent subgraph is a maximal consistent subgraph with maximum size. Now, given a time difference graph $\mathbb{G}$, the problem of obtaining the largest consistent subgraph can be formulated as an optimization problem that finds the maximum consistent subgraph of $\mathbb{G}$. We refer to this problem as the *Maximum Consistent Time Difference Graph* problem or MCTD. A decision (or parametrized) version of MCTD can be stated as,

**Input:** A partially consistent time difference graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, w, \delta)$ and a positive integer $k$ s.t. $k \leq \|V\|$.

**Question:** Does $\mathbb{G}$ contain a consistent time difference subgraph of size $k$ or more?

## 3.2 Hardness of the MCTD Problem

Intuitively, the MCTD problem appears to be hard. Actually, MCTD does belong to the class of highly intractable problems. Currently, we do not have sufficient proof that it even belongs to $NP$, i.e., the class of non-deterministic polynomial-time algorithms. This is because, it is highly unlikely that MCTD even has a *polynomial-time verifier*. Given a time difference graph, $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \delta)$[2] and an integer $k \leq \|\mathbb{V}\|$, it is not possible to verify in polynomial time whether a subgraph of $\mathbb{G}$ (of size $k$) contains only consistent cycles. In order to verify the consistency of the subgraph, all the simple cycles in the subgraph have to be verified for consistency. The total number of cycles in the subgraph itself could be exponential, in the worst case. But, we do show that MCTD is *NP-hard*, i.e., it is at least as hard as every problem in $NP$. We prove this result by a straightforward polynomial-time many-one (hardness preserving) reduction from a well-known $NP$-complete problem, the VERTEX-COVER [16] problem. A vertex cover of a directed graph $\tilde{G} = (\tilde{V}, \tilde{E})$ is a subset of vertices $C \subseteq \tilde{V}$ that contains at least one vertex of every directed edge $(\tilde{u}, \tilde{v}) \in \tilde{E}$, and the (minimum) VERTEX-COVER problem is to find such a subset $C$ of the smallest cardinality. The hardness of the MCTD problem is given by Theorem 3.1.

THEOREM 3.1. *The Maximum Consistent Time Difference Graph (MCTD) problem is NP-hard.*

PROOF. We show that VERTEX-COVER $\leq_m^P$ MCTD, i.e., VERTEX-COVER many-one ($m$) reduces in polynomial time to the MCTD problem.
Construction: We describe a polynomial-time construction that maps an instance $\tilde{G} = (\tilde{V}, \tilde{E})$ of the VERTEX-COVER problem to an instance $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \delta)$ of the MCTD problem such that $\tilde{G}$ has a vertex cover of size $\leq k$ ($k \leq \|\tilde{V}\|$) if and only if $\mathbb{G}$ has a consistent subgraph of size $\geq \|\tilde{V}\| - k$. Since, consistency can be verified using just the time difference function $\delta$, we can ignore the clock function $w$. Let, $a > 1$ be some small constant and $\|\tilde{V}\| = n$. The construction is shown in Figure 4.

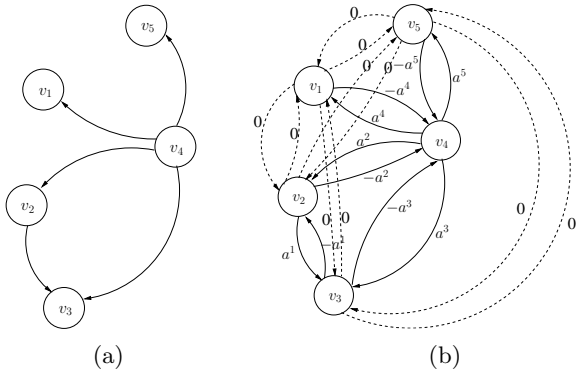



**Figure 4: Reduction from VERTEX-COVER TO MCTD (a) Input graph $\tilde{G} = (\tilde{V}, \tilde{E})$ for VERTEX - COVER (b) Input graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ for MCTD**

[2]We can ignore $w$ from this point onwards since it is not used in the consistency check of the time difference graph

1. For each vertex $v$ in the vertex set $\tilde{V}$ of $\tilde{G}$, place a vertex $v$ in the vertex set $\mathbb{V}$ of $\mathbb{G}$.
2. For each vertex pair $u, v \in \tilde{V}$ such that edge $(u, v) \notin \tilde{E}$ and $(v, u) \notin \tilde{E}$, add edges $(u, v)$ and $(v, u)$ in $\mathbb{E}$ of $\mathbb{G}$. For each such edge, define $\delta(u, v) = 0$ and $\delta(v, u) = 0$. These edges are shown as dotted lines in Figure 4(b).
3. Initialize some counter $\beta = 1$. Select a directed edge $(u, v) \in \tilde{E}$. Add edges $(u, v)$ and $(v, u)$ in $\mathbb{E}$, if they are not already present in $\mathbb{E}$. These edges are shown as solid lines in Figure 4(b). Assign $\delta(u, v) = a^\beta$ and $\delta(v, u) = -a^\beta$. Increment $\beta$ by 1. Repeat this procedure till every directed edge $(u, v) \in \tilde{E}$ is covered.

It is clear that the above construction can be completed in polynomial time. We now show that $\tilde{G}$ has a vertex cover of size $k$ if and only if $\mathbb{G}$ has a consistent time difference graph of size $\|\tilde{V}\| - k$.

Suppose the graph $\tilde{G}$, as shown in Figure 4(a), has a vertex cover $C$ ($C \subseteq \tilde{V}$) of size $k$ ($\|C\| = k$). Since $C$ is a vertex cover, $\forall (u, v) \in \tilde{E}$, either $u$ or $v$ or both are in $C$. By our construction, $\forall (u, v) \in \tilde{E}$, $\delta(u, v) \neq 0$ and $\delta(v, u) \neq 0$ in $\mathbb{G}$. In other words, $C$ covers all edges with non-zero time difference values in $\mathbb{G}$. This implies that $\mathbb{V} - C$ would contain only edges with time difference values 0, i.e., $\mathbb{V} - C$ would contain no cycle such that the sum of the time difference values of all the edges in that cycle is not equal to 0. Thus, the subgraph induced by $\mathbb{V} - C$ is a consistent time difference graph of $\mathbb{G}$ of size $\|\mathbb{V}\| - k$.

Now, we prove the other direction. From the above construction, it is clear that the solid edges in $\mathbb{G}$ are assigned time difference ($\delta$) values in a geometric progression. From the property of the progression, the only way the sum of edge weights or time difference values $\delta$ in any simple cycle can be zero is if all the edge weights $\delta$ in the cycle are zero. Now, let $C'$ be the vertex set representing the consistent time difference subgraph of $\mathbb{G}$ of size $k$ ($k \leq \|\mathbb{V}\|$). Thus, $C'$ should contain no edges with a non-zero $\delta$ values, i.e., there can be no edges in $C'$ with time difference values either $a^\beta$ or $-a^\beta$, where $\beta \geq 1$ and $a > 1$. Otherwise there will be at least one inconsistent cycle in $C'$. This implies, $\mathbb{V} - C'$ covers all edges in $\mathbb{E}$ with time difference values $a^\beta$ or $-a^\beta$. From our construction, it is clear that the edge set $\tilde{E}$ of $\tilde{G}$ is a subset of all edges with time difference values $a^\beta$ or $-a^\beta$ in $\mathbb{E}$. Thus, $\mathbb{V} - C'$ covers all the edges in $\tilde{G}$ and is a vertex cover of $\tilde{G}$ of size $\|\mathbb{V}\| - k$ i.e., $\|\tilde{V}\| - k$ since $\|\tilde{V}\| = \|\mathbb{V}\|$.

Thus, VERTEX-COVER many-one reduces in polynomial time to the MCTD problem. Since VERTEX-COVER is $NP$-complete, MCTD is $NP$-hard. □

Theorem 3.1 implies that it is unlikely that MCTD will have a deterministic polynomial-time algorithm. Whether it has an efficient approximation algorithm is still an open question. The next result shows the equivalence of the MCTD problem to another combinatorially hard problem, namely the *Feedback Vertex Set* problem or FVS, which further undermines the possibility of the existence of an efficient polynomial-time approximation algorithm for the MCTD problem.

## 3.3 Solving the MCTD Problem

Before outlining the next result, we need to introduce another graph-based optimization problem, namely the Feedback Vertex Set problem or FVS [12]. The FVS problem is

defined as: given a directed or undirected graph, $G = (V, E)$, find a subset $F \subseteq V$ of vertices in the graph such that $G - F$ is acyclic. In simpler words, the FVS problem is to find a (minimum) subset of vertices that covers all the cycles in $G$. The set $F$ is called the feedback vertex set or FVS of $G$. Our next proposition gives the relationship between the MCTD problem and the minimum FVS problem.

PROPOSITION 3.1. *The MCTD problem for a time difference graph $\mathbb{G}$ is equivalent (or corresponds in a one-to-one way) to finding the minimum feedback vertex set of all the simple negative cycles in $\mathbb{G}$.*

A negative cycle is a cycle such that the sum of all the edge weights of the cycle is strictly less than 0. We skip the details of the proof for Proposition 3.1, as it follows from a very straightforward polynomial time reduction from the MCTD problem. One point to note here is that in redundant time difference graphs, finding the minimum feedback vertex set of all the positive simple cycles (strictly greater than zero) would also give a solution to the MCTD problem. But, that is not going to affect the hardness of the problem because the construction of the redundant time difference graph guarantees exactly same number of positive and negative simple cycles. Proposition 3.1 implies that (efficient) algorithms for the negative cycle enumeration problem and the minimum FVS problem in weighted directed graphs can be used to obtain a (efficient) solution for the MCTD problem. Based on this result, a simple algorithm for the MCTD problem can be outlined as shown in Algorithm 1.

1: Compute the set of all negative cycles $C$ in $\mathbb{G}$.
2: Compute the feedback vertex cover $F$ of $C$.
3: **return** $\mathbb{G} - F$ as the maximum consistent time difference graph of $\mathbb{G}$

**Algorithm 1:** Calculating maximum consistent subgraph

Both the negative cycle enumeration and the minimum FVS problems are known $NP$-complete problems. Although, deciding the existence and finding a negative cycle in a weighted directed graph is polynomially solvable, and all cycles of a directed or undirected graph can be enumerated efficiently by a simple backtracking algorithm [27], Kachiyan et al. [17] proved using Gallai's results [8] that (directed) negative cycles in a graph cannot be generated in polynomial time, unless $P = NP$. Similarly, Karp et al. [16] proved the $NP$-completeness of the minimum FVS problem. For undirected graphs, there are numerous results for the FVS problem, for example, there are exact algorithms that run in time $O(1.9053^n)$ [26] and in time $O(1.7548^n)$ [7], where $n$ is the number of vertices. There also exist a polynomial-time 2-approximation algorithm for it that was proposed by Bafna et al. [2]. In directed graphs, the FVS problem becomes harder and there has been only a limited progress on it, since Karp [16] proved that to find an FVS in a directed graph of size bounded by some constant $k$ is $NP$-complete. No exact algorithms with running time within $O(c^n n^{O(1)})$, where $c < 2$, and no polynomial time approximation algorithms with constant ratio have been found [6]. From Proposition 3.1 and the infeasibility result of Kachiyan et al. [17], we can conclude that it is highly unlikely that the MCTD problem will have a polynomial-time (or efficient) approximation algorithm, let alone a constant ratio polynomial time algorithm.

These negative results for the MCTD problem in connected time difference graphs, prompted us to investigate the combinatorial properties of the MCTD problem for a restricted type of time difference graph, namely a *completely connected time difference graph*. In the next few sections, we present some analysis and results for this special case.

# 4. COMPLETELY CONNECTED TIME DIFFERENCE GRAPHS

A completely connected (or complete) time difference graph is a special type of redundant time difference graph in which there is an (directed) edge between every pair of vertices in the graph (in both the directions). Although it is very difficult to model existing sensor networks (except extremely dense networks spread over a small area) using such completely connected graphs, it will be worthwhile to study such graphs in the context of the MCTD problem.

## 4.1 Properties

A complete time difference graph possesses the following two properties.

PROPERTY 1. *A complete time difference graph has a polynomial number of exactly three node (vertex) cycles. Specifically, there are a total of $2 \cdot \binom{n}{3}$ three vertex cycles, where $n$ is the total number of vertices in the graph.*

PROPERTY 2. *A complete time difference graph is consistent if and only if all the three node (vertex) cycles in the graph are consistent.*

Property 1 follows from the definition of the complete time difference graph, while Property 2 follows from Proposition 4.1, as discussed below.

PROPOSITION 4.1. *In a complete time difference graph, two $k$-node ($k \geq 3$) simple cycles with two or more common vertices are independently consistent if and only if the simple cycle containing all the vertices of these two $k$-node cycles, and formed by the edges of these cycles, is consistent.*

PROOF. First we prove this result for the simple case of $k = 3$. We then argue that the result holds for $k > 3$.

Let us first prove the forward direction for this case. Without loss of generality let us assume that {1,2,3,1} and {1,3,4,1} are the two 3-node independently consistent cycles with the common nodes 1 and 3, as shown in Figure 5. Thus, by definition of consistency, $\delta_{12} + \delta_{23} + \delta_{31} = 0$ and $\delta_{13} + \delta_{34} + \delta_{41} = 0$. The cycle containing all the four nodes and formed by the edges of the above two 3-node cycles is $\{1, 2, 3, 4, 1\}$. Now, let cycle $\{1, 2, 3, 4, 1\}$ be not consistent. Thus,

$$\begin{aligned} & \delta_{12} + \delta_{23} + \delta_{34} + \delta_{41} \neq 0 \\ \implies \quad & \delta_{12} + \delta_{23} + \delta_{31} - \delta_{31} + \delta_{34} + \delta_{41} \neq 0 \\ \implies \quad & -\delta_{31} + \delta_{34} + \delta_{41} \neq 0 \quad (\delta_{12} + \delta_{23} + \delta_{31} = 0) \\ \implies \quad & \delta_{13} + \delta_{34} + \delta_{41} \neq 0 \quad (\delta_{13} = -\delta_{31}) \\ \implies \quad & \textit{Cycle } \{1, 3, 4, 1\} \textit{ is inconsistent} \end{aligned}$$

which is a contradiction. Thus, cycle $\{1, 2, 3, 4, 1\}$ is consistent. Similarly, the reverse direction is also very straightforward.

Now, we argue that this also holds for $k > 3$. It is easy to see that two $k$-node cycles with two common vertices can

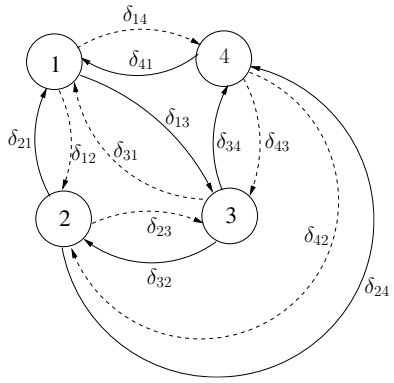



**Figure 5: Illustration of property 2**

be combined into a single cycle using the same set of edges of the individual cycles if and only if both cycles have the same orientation (i.e., either clockwise or anti-clockwise). As a result, the first $k$-node cycle has one of the edges (in one direction) between the two common nodes, while the second $k$-node cycle has the other edge (in the other direction). Since each $k$-node cycle is consistent, the $\delta$ value of the edge in the cycle that is between the common nodes is numerically equal but opposite in sign to the sum of the $\delta$ values of all the other edges in the cycle. In other words, the sum of $\delta$ value of every edge except the edge between the common nodes in the first $k$-node cycle is numerically equal in value but opposite in sign to the corresponding sum of $\delta$ values of the edges of the other k-node cycle. When the two $k$-node cycles are combined, the only edges not included in the combined cycle are the ones between the common nodes. Thus, the sum of the $\delta$ values of the edges of the resulting cycle will always be zero, which implies that the combined cycle will always be consistent. Similarly, the reverse direction can be proved for $k > 3$. □

## 4.2 Theoretical Results

Contrary to the initial intuition, theoretical analysis, as discussed next, has shown that the MCTD problem is computationally hard even for the completely connected case, although, there exist polynomial-time approximations for it.

THEOREM 4.1. *MCTD problem for completely connected time difference graph is NP-complete.*

This theorem follows from Theorem 4.3 and Lemma 4.1 discussed ahead. Before proceeding ahead, we need the following result from Johan Ḣastad (2002) [14] for the CLIQUE problem.

THEOREM 4.2. *For any $\epsilon > 0$, unless $NP = P$ there is no polynomial-time algorithm that approximates CLIQUE within a factor $n^{\frac{1}{2}-\epsilon}$.*

CLIQUE, a well-known $NP$-hard problem, for a given graph with $n$ vertices is to find a maximum size *clique* in it, i.e., a complete subgraph of maximum size [16]. The best known algorithm for approximating CLIQUE has a factor of $O(\frac{n}{log^2 n})$ [3]. Our next result, proves the approximability of the MCTD problem for complete time difference graphs.

THEOREM 4.3. *For any $\epsilon > 0$, it is NP-hard to approximate the MCTD problem for a completely connected time difference graph within a factor $n^{\frac{1}{2}-\epsilon}$ ($\epsilon > 0$).*

PROOF. From Theorem 4.2, it is clear that CLIQUE, in general, is not approximable to within $n^{\frac{1}{2}-\epsilon}$, assuming $P \neq NP$. We describe a reduction from CLIQUE to MCTD for complete time difference graphs such that the approximation ratio is preserved, and with only a quadratic blow-up in the input size.

Suppose, we have an instance of CLIQUE, $\tilde{G} = (\tilde{V}, \tilde{E})$, as shown in Figure 6(a). We can construct an instance $\mathbb{G} =$

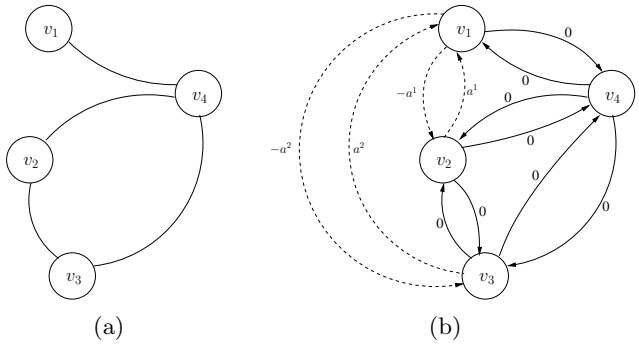



**Figure 6: Reduction from CLIQUE to MCTD**

$(\mathbb{V}, \mathbb{E}, \delta)$ of MCTD for a complete time difference graph as follows. Let, $a > 1$ be some small constant and $\|\tilde{V}\| = n$. Refer to Figure 6 for an illustration of the reduction. The vertex set of the graph $\mathbb{G}$ is the same as the vertex of $\tilde{G}$, i.e., for each $v \in \tilde{V}$, add a vertex $v$ to $\mathbb{V}$. Now, for each edge in the edge set of $\tilde{E}$, i.e., for each $(v_i, v_j) \in \tilde{E}$, add two directed edges $(v_i, v_j)$ and $(v_j, v_i)$ in $\mathbb{E}$. Also, for each such edge in $\mathbb{E}$, $\delta(v_i, v_j) = 0$ and $\delta(v_j, v_i) = 0$. These edges are shown as solid lines in Figure 6(b).

Initialize some counter $\beta = 1$. Select a vertex pair $v_i, v_j$ such that both edges $(v_i, v_j)$ and $(v_j, v_i) \notin \tilde{E}$. Add two directed edges $(v_i, v_j)$ and $(v_j, v_i)$ in $\mathbb{E}$, if they are not already present in $\mathbb{E}$. These edges are shown as dotted lines in Figure 6(b). Assign $\delta(v_i, v_j) = a^\beta$ and $\delta(v_j, v_i) = -a^\beta$. Increment $\beta$ by 1. Repeat this procedure till every pair of vertices $v_i, v_j$ for which edges $(v_i, v_j), (v_j, v_i) \notin \tilde{E}$ is covered. It is easy to see that $\mathbb{G}$ is a complete time difference graph. Moreover, we can observe that the above construction can be done efficiently, i.e., in polynomial time (in terms of the number of vertices) with only a quadratic increase in the size of the input. This increase is due to the extra edges that are added to the input, which is of the order of $O(n^2)$.

Next, we need to show that the above construction is approximation preserving, given by Lemma 4.1 below. This completes the proof. □

LEMMA 4.1. *For any positive integer $p \leq \|\tilde{V}\|$, the graph $\tilde{G} = (\tilde{V}, \tilde{E})$ has a CLIQUE of size at most p if and only if there is a consistent subgraph of the time difference graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \delta)$ of size at most p.*

PROOF. From the construction in Theorem 4.3, it is clear that the dotted edges in $\mathbb{G}$ are assigned time difference ($\delta$) values in a geometric progression. From the property of the progression, the only way the sum of edge weights or time difference values $\delta$ in any simple cycle can be zero is if all the edge weights $\delta$ in the cycle are zero. Now, let $C \subseteq \mathbb{V}$ be the vertex set representing the consistent time difference subgraph of $\mathbb{G}$ of size $p$ ($p \leq \|\mathbb{V}\|$). Thus, $C$ should contain no edges with a non-zero $\delta$ values, i.e., there

can be no edges in $C$ with time difference values either $a^\beta$ or $-a^\beta$, where $\beta \geq 1$ and $a > 1$. Otherwise there will be at least one inconsistent cycle in $C$. Moreover since $\mathbb{G}$ is a complete graph, $C$ is also a complete graph. This implies, $C$ is a clique of $\tilde{G}$ of size $p$.

Similarly for the forward direction, let us assume that the graph $\tilde{G} = (\tilde{V}, \tilde{E})$ has a CLIQUE $C' \subseteq \tilde{V}$ of size $p$. From the construction in Theorem 4.3, it is clear that $C'$ is also a consistent subgraph of the time difference graph $\mathbb{G}$, and there can be no other consistent graph larger than it. □

## 4.3 Heuristics

We now propose two heuristics for the MCTD problem in complete time difference graphs. The first heuristic is based on a greedy strategy, while the second is based on an Integer Programming formulation of the MCTD problem.

### 4.3.1 Greedy Strategy

This heuristic greedily selects a vertex for inclusion into the solution based on its *Consistency Index (CI).*

*Definition 3.* The *Consistency Index (CI)* of a vertex in a complete time difference graph is the total number of consistent 3-node cycles of the graph that the vertex is a part of.

The heuristic begins by computing the CI for each vertex in the graph. It then sorts the vertices based on their CI values from the highest to the lowest. If there are no vertices with CI greater than zero then there are no consistent three-node cycles, and the heuristic concludes that there is no consistent subgraph. Otherwise, it selects the vertex with the highest CI and places it in the partial solution. It continues to select vertices in the order of their CI's and adds them to the partial solution if and only if they are consistent with all the previous vertices in the partial solution, i.e., all the 3-node cycles after adding the new vertex should be consistent. If there is at least one inconsistent 3-node cycle then the vertex is discarded and not added to the partial solution. During the vertex selection step, if there are more than one vertex with the same CI value then the heuristic picks a vertex at random from such a group. The pseudo-code for this heuristic is shown in Algorithm 2.

**Require:** Time Difference Graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \delta)$, $\|\mathbb{V}\| = n$
1: Compute CI for each vertex, $v_i \in \mathbb{V}$
2: Sort vertices based on CI from largest to smallest. Let the sorted list be $\mathbb{V}' = \{v_1, v_2, \ldots, v_n\}$
3: **if** there is no vertex $v_i \in \mathbb{V}'$ with CI $> 0$ **then**
4: **print** "No consistent subgraph"
5: **return** $\phi$
6: **else**
7: Let $\mathbb{T} = v_1$
8: **for** each vertex $v_i \in \mathbb{V}'$ ($i = 2$ to $n$) **do**
9: **if** $v_i$ is consistent with every vertex in $\mathbb{T}$ (i.e., all 3-node cycles in $\mathbb{T}$ after adding $v_i$ are consistent) **then**
10: Add $v_i$ to $\mathbb{T}$
11: **end if**
12: **end for**
13: **return** $\mathbb{T}$
14: **end if**

**Algorithm 2:** Greedy heuristic

The CI computation step of the greedy algorithm takes $O(n^4)$ time, where $n$ is the number of vertices. This is because there are at most $O(n^3)$ three-vertex cycles in the complete time difference graph and it takes an extra $O(n)$ time to assign a CI for every vertex in the graph. Sorting takes $O(n.logn)$ and the verification and addition steps (4 through 8) take $O(n^4)$ in the worst case. Thus, the total running time of the greedy algorithm is bounded by $O(n^4)$.

### 4.3.2 Integer Programming (IP) Formulation

One drawback of the greedy algorithm is that it makes locally optimal decisions due to which it can get stuck in a local optima, thus resulting in a poor solution quality. To overcome this issue, we propose an alternative heuristic that uses convex optimization to find a globally optimal solution. More specifically, we formulate the MCTD problem for a complete time difference graph as an Integer Program (IP), also called a 0-1 Program, as shown below:

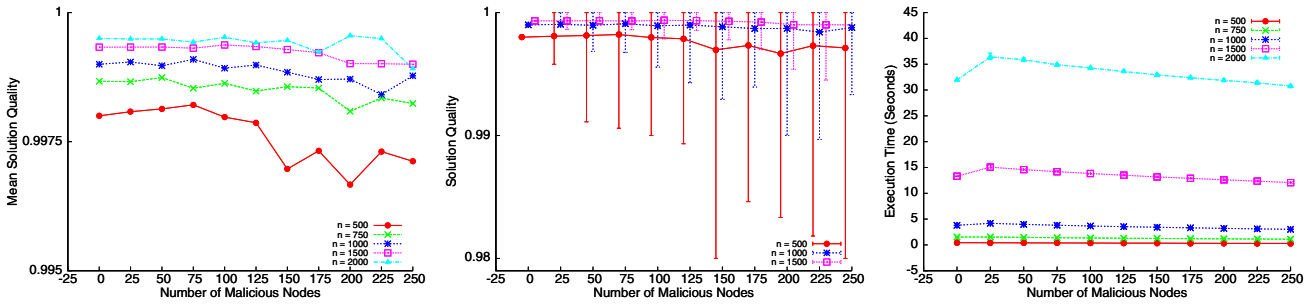
$$\begin{aligned} &\textit{Maximize} && f = \sum_{i=1}^{n} v_i \\ &\textit{Subject to} && (v_i + v_j + v_k - 2) \cdot \delta_{i,j,k} \leq 0 \\ &&& \qquad \forall i,j,k \;\; s.t. \;\; i,j,k \in \{1,2,\ldots,n\} \; and \\ &&& \qquad \delta_{i,j,k} \equiv \delta_{i,j} + \delta_{j,k} + \delta_{k,i} \geq 0 \\ &&& and \; v_i, v_j, v_k \in \{0,1\} \end{aligned}$$

Now, solving an Integer Program is a well-known NP-hard problem [16]. But, a Linear Program (LP) relaxation for the above Integer program can be solved in polynomial time using efficient techniques such as *simplex*. If the LP relaxation has an integral solution then that can also be the solution for the above IP. But due to the hardness of the MCTD problem, as proved earlier, getting an integral solution is highly unlikely. In that case approximation strategies such as rounding and branch and bound can be used to obtain a feasible solution.

# 5. EMPIRICAL EVALUATIONS

In this section, we discuss the results of some initial simulation experiments conducted for the greedy and the LP-based heuristics. The greedy heuristic is implemented using the $C$ programming language, while the LP-based heuristic is implemented in MATLAB using the *linprog* LP solver of the MATLAB Optimization toolbox. These implementations are executed on a Pentium dual processor/2GHz/2GB specification machine and running Debian Linux operating system.

We execute the implementation of the greedy heuristic with randomly generated complete time difference graphs as input; the size ($n$) of these graphs are varied from 500 up to 2000 nodes during the simulations. In each simulation run, we randomly choose $k$ number of nodes as malicious, and assign inconsistent $\delta$ values to randomly chosen edges out of each malicious node. The value of $k$ is varied from 0 up to 250 nodes in steps of 25 nodes for each value of $n$. The distance estimate function $\delta$ assigns values only from the set of integers ($\mathbb{Z}$). It is easy to see that if we remove all the malicious nodes and their corresponding edges then the resulting subgraph becomes consistent. This subgraph may or may not be the optimal solution, and we refer to such a subgraph as the *sub-optimal* solution. Since it is computationally infeasible to get the true optimal solution,




(a) (b) (c)

**Figure 7: Experimental results for the greedy heuristic**

especially for large graphs, we measure the *solution quality* of the greedy heuristic by computing the *ratio* of the size of the solution returned by the heuristic to the size of the sub-optimal solution. Simulations for each set of parameters is repeated 100 times. The simulation results for the greedy heuristic are shown in Figure 7.

Figure 7(a) shows the variation of the mean or average (over the 100 runs) value of the solution quality as the number of malicious nodes increase. We can see that the solution quality decreases as the number of malicious nodes increases, which is pretty intuitive. Moreover, as the graph becomes larger in size, i.e., the ratio of malicious to honest nodes decreases, the solution quality improves. In Figure 7(b), we have plotted the solution quality and the confidence intervals for $n = 500$, $n = 1000$ and $n = 1500$. We can see that the confidence interval is much larger for $n = 500$ and becomes smaller as the value of $n$ increases. This shows that when the ratio of malicious to honest nodes in the graph is high, the greedy heuristic is much less predictable in its solution quality. This improves when the ratio of malicious to honest nodes decreases. To summarize, we can say that despite the negative result in Theorem 4.3, the solution quality of the greedy heuristic is good for lower values of $n$ and that the solution quality is never better than the sub-optimal solution. Figure 7(c) shows the execution time of the greedy heuristic for each value of $n$ as the number of malicious nodes increases. From the plot, we can see that there is not much variation in the execution time as the number of malicious nodes increases. But, the execution time increases sharply as the size $n$ of the complete time difference graph increases.

Next, we attempt to simulate the LP-based heuristic under similar simulation parameters. Unfortunately, we are unable to get similar extensive results for the LP-based heuristic because of the inability to simulate the current implementation of the heuristic for very large values of $n$. From Section 4.3.2, we can see that the total number of constraints in the IP formulation is around $2 \cdot \binom{n}{3}$. This makes the LP solver very very slow even for reasonably large values of $n$. For lower values of $n (n < 50)$, the trends for the solution quality of the LP-based heuristic is similar to the greedy heuristic, i.e., it decreases with increase in the number of malicious nodes, and the average solution quality is always above 0.95. It would be interesting to study the efficiency of the LP-based heuristic for very large complete time difference graphs using a better implementation. We plan to undertake this exercise as a part of future work.

# 6. CONCLUSION

In this paper, we have provided a formal treatment for the problem of eliminating cheating behavior in time synchronization protocols for highly distributed systems like wireless sensor networks. We have modeled the time synchronization problem as a constraint satisfaction problem using a graph-based representation of the network, called the time difference graph. The problem of eliminating cheating (or inconsistent) behavior was then formulated as an optimization problem, called MCTD, in such graphs. We have proved that MCTD for the general case is a combinatorially hard problem. We have also showed that a restricted case of MCTD, namely for complete time difference graphs, is also hard ($NP$-complete) and that there is no algorithm that can approximate it within a factor $n^{\frac{1}{2}-\epsilon}$, unless $P = NP$. We have also outlined two simple heuristics for this case and presented some analysis based on initial empirical results.

These hardness results for securing time synchronization may assume less significance in small sensor networks containing only a few nodes, as even exhaustive approaches are feasible in such networks. But, most practical sensor network applications usually consist of thousands of sensor nodes spread over a vast area. In such large networks, the significance of these results can no longer be ignored, as securing network-wide time synchronization using exhaustive techniques can quickly become infeasible.

# 7. REFERENCES

[1] K. Arvind. Probabilistic Clock Synchronization in Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):474–487, 1994.

[2] V. Bafna, P. Berman, and T. Fujito. A 2-Approximation Algorithm for the Undirected Feedback Vertex Set Problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.

[3] R. Boppana and M. Halldórsson. Approximating Maximum Independent Sets by Excluding Subgraphs. *BIT*, 32:180–196, 1992.

[4] F. Cristian. Probabilistic Clock Synchronization. *Distributed Computing*, 3:146–158, 1989.

[5] J. Elson, L. Girod, and D. Estrin. Fine-grained Network Time Synchronization using Reference Broadcasts. *ACM SIGOPS Operating Systems Review*, 36:147–163, 2002.

[6] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating Minimum Feedback Sets and

Multicuts in Directed Graphs. *Algorithmica*, 20(2):151–174, 1998.

[7] F. V. Fomin, S. Gaspers, and A. V. Pyatkin. Finding a Minimum Feedback Vertex Set in Time $o(1.7548^n)$. In *IWPEC 2006: Proceedings of the* $2^{nd}$ *International Workshop on Parameterized and Exact Computation*, pages 184–191, 2006.

[8] T. Gallai. Maximum-minimum sätze über graphen. *Acta Mathematicae, Academiae Scientiarum Hungaricae*, 9:395–434, 1958.

[9] S. Ganeriwal, S. Capkun, C.-C. Han, and M. B. Srivastava. Secure Time Synchronization Service for Sensor Networks. In *WiSe '05: Proceedings of the* $4^{th}$ *ACM Workshop on Wireless Security*, pages 97–106, 2005.

[10] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync Protocol for Sensor Networks. In *SenSys '03: Proceedings of the* $1^{st}$ *International Conference on Embedded Networked Sensor Systems*, pages 138–149, 2003.

[11] S. Ganeriwal, C. Pöpper, S. Čapkun, and M. B. Srivastava. Secure Time Synchronization in Sensor Networks. *ACM Transactions on Information and System Security (TISSEC)*, 11(4):1–35, 2008.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[13] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge, 2008.

[14] J. Ḣastad. Clique is Hard to Approximate within $n^{1-\epsilon}$. *Acta Mathematics*, 182:105–142, 2002.

[15] M. Jadliwala, Q. Duan, J. Xu, and S. Upadhyaya. On Extracting Consistent Graphs in Wireless Sensor Networks. *International Journal of Sensor Networks (IJSNet)*, 2(3/4):149–162, 2007.

[16] R. Karp. *Complexity of Computer Computations*, chapter Reducibility Among Combinatorial Problems, pages 85–104. Plenum Press, 1972.

[17] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, and V. Gurvich. Generating all Vertices of a Polyhedron is Hard. In *SODA '06: Proceedings of the* $17^{th}$ *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 758–765, 2006.

[18] M. D. Lemmon, J. Ganguly, and L. Xia. Model-based Clock Synchronization in Networks with Drifting Clocks. In *PRDC '00: Proceedings of the 2000 Pacific Rim International Symposium on Dependable Computing*, pages 177–184, 2000.

[19] H. Li, Y. Zheng, M. Wen, and K. Chen. *A Secure Time Synchronization Protocol for Sensor Network*, chapter Emerging Technologies in Knowledge Discovery and Data Mining, pages 515–526. Springer Berlin / Heidelberg, 2007.

[20] Q. Li and D. Rus. Global Clock Synchronization in Sensor Networks. *IEEE Transactions on Computers*, 55(2):214–226, 2006.

[21] M. Maróti, B. Kusy, G. Simon, and Ákos Ĺdeczi. The Flooding Time Synchronization Protocol. In *SenSys '04: Proceedings of the* $2^{nd}$ *International Conference on Embedded Networked Sensor Systems*, pages 39–49, 2004.

[22] D. Mills. Internet time synchronization: The network time protocol. In *Global States and Time in Distributed Systems, IEEE Computer Society Press*. 1994.

[23] S. B. Moon, P. Skelly, and D. Towsley. Estimation and Removal of Clock Skew from Network Delay Measurements. Technical Report UM-CS-1998-043, 1998.

[24] P. P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade, D. Basin, S. Capkun, and J.-P. Hubaux. Secure Neighborhood Discovery: A Fundamental Element for Mobile Ad Hoc Networking. *IEEE Communications Magazine*, 46(2), 2008.

[25] S. Ping. Delay Measurement Time Synchronization for Wireless Sensor Networks. *Intel Research, IRB-TR-03-013*, June 2003.

[26] I. Razgon. Exact Computation of Maximum Induced Forest. In *SWAT '06: Proceedings of the* $10^{th}$ *Scandinavian Workshop on Algorithm Theory*, pages 160–171, 2006.

[27] R. Read and R. Tarjan. Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees. *Networks*, 5:237–252, 1975.

[28] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, chapter Constraint Satisfaction Problems, pages 137–160. 2 edition, 2002.

[29] M. Sichitiu and C. Veerarittiphan. Simple, Accurate Time Synchronization for Wireless Sensor Networks. In *WCNC 2003: Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 1266–1273, 2003.

[30] H. Song, S. Zhu, and G. Cao. Attack-Resilient Time Synchronization for Wireless Sensor Networks. In *MASS 2005: Proceedings of the* $2^{nd}$ *IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 765–772, 2005.

[31] K. Sun, P. Ning, and C. Wang. TinySeRSync: Secure and Resilient Time Synchronization in Wireless Sensor Networks. In *CCS '06: Proceedings of the* $13^{th}$ *ACM Conference on Computer and Communications Security*, pages 264–277, 2006.

[32] B. Sundararaman, U. Buy, and A. D. Kshemkalyani. Clock Synchronization in Wireless Sensor Networks: A Survey. *Ad-Hoc Networks*, 3(3):281–323, 2005.

[33] J. van Greunen and J. Rabaey. Lightweight Time Synchronization for Sensor Networks. In *WSNA '03: Proceedings of the* $2^{nd}$ *ACM International Conference on Wireless Sensor Networks and Applications*, pages 11–19, 2003.

[34] Y. Xianglan, Q. Wangdong, and F. Fei. ASTS: An Agile Secure Time Synchronization Protocol for Wireless Sensor Networks. In *WiCom '07: Proceedings of the* $3^{rd}$ *International Conference on Wireless Communications, Networking and Mobile Computing*, pages 2808–2811, 2007.