Research Article

A Cluster-Based Secure Synchronization Protocol for Underwater Wireless Sensor Networks

Ming Xu,1,2 Guangzhong Liu,1 Daqi Zhu,1 and Huafeng Wu3

1 College of Information Engineering, Shanghai Maritime University, 1550 Haigang Avenue, Shanghai 201306, China
2 Shanghai Key Lab of Intelligent Information Processing, Fudan University, 220 Handan Road, Shanghai 200433, China
3 Merchant Marine College, Shanghai Maritime University, 1550 Haigang Avenue, Shanghai 201306, China

Correspondence should be addressed to Ming Xu; mingxu@shmtu.edu.cn

Received 11 October 2013; Revised 27 February 2014; Accepted 19 March 2014; Published 8 April 2014

Academic Editor: Marimuthu Palaniswami

Copyright © 2014 Ming Xu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Secure time synchronization is one of the key concerns for Underwater Wireless Sensor Networks (UWSNs). This paper presents a CLUster-based Secure Synchronization (CLUSS) protocol that ensures the security of synchronization under harsh underwater environments against various attacks, including Sybil attack, replay attack, message manipulation attack, and delay attack. The time synchronization process of CLUSS is divided into three phases: authentication phase, intercluster synchronization phase, and intracluster synchronization phase. During the authentication phase, cluster heads are authenticated to beacons and ordinary nodes are authenticated to cluster heads. During the process of intercluster and intracluster synchronization phases, CLUSS distinguishes the propagation delay of downlink from that of uplink caused by node movement in order to improve time synchronization accuracy. Moreover, part of these two phases can be executed concurrently for reducing the number of messages generated in synchronization. We demonstrate through simulations that CLUSS can reduce synchronization errors as well as energy consumption compared to traditional protocols in various circumstances.

1. Introduction

Underwater Wireless Sensor Networks (UWSNs) have a lot of potential application areas such as oceanographic data collection, disaster prevention, pollution monitoring, offshore exploration, and military surveillance [1–3]. UWSN is a new area of wireless sensor network in underwater environments which has challenges to be overcome such as long propagation delay, severely limited bandwidth, and unpredictable node movement [4–6]. All the above distinct features of UWSNs give birth to new challenging areas for every level of the network protocol suite. Among them, providing clock synchronization is a well-known fundamental issue due to the unique characteristics of UWSNs [7, 8]. A number of important network functionalities require that nodes have a common notion of time, including time stamping of events, distributed data aggregation, MAC, and localization. Meanwhile, numerous time synchronization protocols have been proposed for terrestrial wireless sensor networks, such as RBS [9], TPSN [10], and FSTP [11]. However, they cannot be directly applied to UWSNs, because most of them assume negligible propagation delay among sensor nodes, which is not true in UWSNs. On the other hand, as UWSNs are often deployed in an inaccessible and hostile environment, those unattended sensor nodes are likely to incur many critical security attacks, including Sybil attack, replay attack, message manipulation attack, and delay attack [12, 13]. Therefore, time synchronization must resist modification attempts by attackers, especially in applications such as military surveillance and oceanographic data collection.

Time synchronization attacks have great effects on a set of sensor network applications and services since they heavily rely on accurate time synchronization to perform their respective functions. All attacks on time synchronization protocols of UWSNs aim to deceive some nodes that their neighbor’s clocks are at a different time than they actually are. In Sybil attack, a malicious node pretends to have multiple identities. In some cases, an attacker can simply create
fabricated identities. If the name space is intentionally limited to prevent attackers from fabricating identities, they can also destroy or temporarily disable the impersonated nodes and assign stolen identities to malicious nodes. In replay attack, a valid data transmission is maliciously or fraudulently repeated. An attacker can take part in distributing timing messages among its neighbor nodes. As a result, old timing messages could be deceitfully issued as legitimate messages to its neighbor nodes for disrupting the timing synchronization process. In message manipulation attack, an attacker may drop, modify, or even forge the exchanged timing messages to interrupt the process of time synchronization. In a delay attack, an attacker intentionally delays some of the time messages in order to disrupt the time synchronization process and cause it to fail. If this attack infiltrates into the system, it cannot be prevented using traditional cryptographic techniques.

In this paper, we propose a novel CLUster-based Secure Synchronization (CLUSS) protocol that concerns security as well as accuracy in synchronization. Before time synchronization, CLUSS needs to execute the process of cluster formation securely by means of cluster consistency checking. After that, CLUSS performs the process of time synchronization, which can be divided into three phases: authentication phase, intercluster synchronization phase, and intracluster synchronization phase. During the authentication phase, all sensor nodes need to be authenticated to each other and the identified malicious nodes will be removed from the network. During the intercluster synchronization phase, cluster heads synchronize themselves with beacons in sender-receiver mode. And then, ordinary nodes synchronize themselves with cluster heads in receiver-receiver mode during the intracluster synchronization phase. Part of intercluster synchronization phase and intracluster synchronization phase can be executed concurrently in order to reduce the number of messages generated in synchronization. In order to improve time synchronization accuracy, CLUSS detects abnormal end-to-end delay data and identifies malicious nodes. Additionally, CLUSS also distinguishes the propagation delay of downlink from that of uplink caused by node movement during the process of calculating clock skew and offset. Numerical simulations confirm the suitability of our protocol compared with the current schemes.

The remainder of the paper is organized as follows: Section 2 presents a brief overview of related work. Section 3 presents the technical details of our time synchronization protocol. Performance evaluation is described in Section 4. Finally, we conclude the paper in Section 5.

2. Related Work

In the literature, there are various time synchronization protocols for distributed systems like terrestrial radio sensor networks. RBS (Reference Broadcast Synchronization) [9] is a well-known receiver-receiver synchronization protocol that synchronizes a set of receivers with one another by periodically sending beacon messages to their neighbors. Recipients use the message’s arrival time as a point of reference for comparing their clocks. However, RBS requires extra message exchange to communicate the local timestamps between any two nodes which intend to become synchronized. TPSN (Timing-sync Protocol for Sensor Networks) [10] is based on the sender-receiver time synchronization, which employs a two-way message exchange for synchronization. Although TPSN takes care of propagation delays, it does not take the clock skew into account during synchronization period. FTSP (Flooding Time Synchronization Protocol) [11] supports network topology changes including mobile nodes. The applied flood-based communication protocol provides a very robust network, but it is not applicable to underwater sensor network mainly because it also assumes instant reception of messages. TSHL (Time Synchronization for High Latency) [14] is the first time synchronization algorithm designed for high latency networks specifically. It uses one-way communication to estimate the clock skew and two-way communication to estimate the clock offset. TSHL assumes that underwater sensor networks are static and therefore suffers from sensor nodes mobility, especially when nodes move fast. D-Sync [15] provides an indication of the relative motion between nodes. In D-Sync, Doppler shift is used in order to infer the propagation delay and an ordinary least square estimator is applied to provide an estimate for both clock skew and offset. However, in deriving the estimators, communication channel variability is not taken into account, which reduces the accuracy of synchronization. MU-Sync [16] runs two times of linear regression to estimate the clock skew and clock offset for cluster based UWSNs. MU-Sync assumes that the one-way propagation delay can be estimated as the average round trip time, which causes extra errors and has relative high message overhead. Moreover, none of these protocols take security as one of their goals. Consequently, an adversary can easily attack any of these time synchronization protocols by capturing a fraction of the nodes and have them distribute faulty time synchronization messages. In effect, the nodes in the entire network will be out-of-sync with each other. WATERSync [17] is a correlation-based time synchronization protocol specifically for shallow underwater sensor networks. WATERSync integrates the time synchronization procedure with the tree-like network routing topology in vertical direction (the surface station is the tree root), which consists of Gradual Depth Timing (GDT) phase and Level 1 (i.e., between the surface station and first depth nodes) Skew Compensation (LSC) phase. To make the time synchronization dependable, WATERSync adopts a correlation-based security model to detect outlier timestamp data and identify malicious nodes. However, horizontal direction is neglected during the process of time synchronization, which results in high offset errors.

3. Proposed Scheme

3.1. Network Topology and Cluster Formation. We consider that a UWSN is composed of a large number of nodes uniformly scattered in monitoring fields and represented by $G = (V, E)$, where $V$ denotes the set of vertexes (nodes) and $E$ is the set of communication links. We assume that the set
of \( n \) safe nodes is represented as a set \( V_s \) and the set of \( m \) malicious nodes is represented as a set \( V_m \), where \( V = V_s \cup V_m \). Each node only knows about its direct neighboring nodes and communicates with them when necessary without any advanced knowledge about whether they are safe nodes or malicious nodes. All sensor nodes are assumed to be static, or they have low mobility with respect to signal propagation speed. Every sensor node has the same transmission range and is able to communicate with other sensor nodes within its range. We also assume that each node is assigned with a triplet of coordinate \((x, y, z)\), where each coordinate represents the hop distance of the node from one anchor. All sensor nodes have the same communication range of \( r \), which is represented as a sphere volume of radius \( r \) in a UWSN.

**Definition 1.** The function \( \delta(u, v) \) defines the distance between two nodes \( u \) and \( v \) in a 3D Euclidean space as

\[
\delta : N \times N \rightarrow \Gamma : \delta (u, v) = \sqrt{(u_x - v_x)^2 + (u_y - v_y)^2 + (u_z - v_z)^2}.
\]

Two nodes \( n_u \) and \( n_v \) are neighbors and connected by a link if \( \delta(u, v) < r \) and the link between \( n_u \) and \( n_v \) is denoted by \( e(u, v) \). The node degree of \( n_u \), which is denoted by \( D_u \). We construct the network topology with RNG (Relative Neighborhood Graph) [18], and then two nodes become neighbor nodes if and only if for any arbitrary node \( n_p \), \( \delta(u, v) \leq \max\{\delta(p, u), \delta(p, v)\} \). For a three-dimensional Euler space embedded, if the arcuate area formed by the intersection of two spheres centered at \( n_u \) and \( n_v \) (with radius \( \delta(u, v) \)) is empty, then \( n_u \) and \( n_v \) are adjacent nodes.

RNG algorithm is simple and is easily built in a distributed way. There is no crossing edges in a RNG because at least one edge in any pair of crossing edges must be removed according to their definitions and the time complexity is \( O(n^2) \). While constructing from a Delaunay Triangulation Graph structure, its complexity of lower bound is \( O(n \log(n)) \) [19]. In addition, a computational method with the complexity of \( O(n^2) \) for the RNG in a three-dimensional space is given in [20]. As underwater sensors float with currents, their movements are constrained in different horizontal planes and they are likely to maintain a steady position relative to each other. The construction of RNG does not require that the exact positions of nodes and their neighbors are known. For each node, only the corresponding mutual distances to its neighbors are required. Therefore, RNG is expected to be more suitable in modeling UWSN, which achieves more accurate results and behaves more consistently than other models. In fact, we model a UWSN with a RNG, not form a RNG overlay on top of a random geometric graph. The deployment of a UWSN is deemed to be sparser than that of a terrestrial sensor network due to the cost involved and to the challenges associated to the deployment itself in the underwater environment. Therefore, modeling a UWSN with a random geometric graph is not suitable for selection in view of the connectivity especially when sensor nodes are not evenly distributed.

CLUSS is a CLUSTER-based Secure Synchronization protocol. Given a UWSN, the whole network is composed of three types of nodes: beacons, cluster heads, and ordinary nodes. Beacons have unlimited energy resources and perfect timing information. For example, they can synchronize to UTC (Universal Time Coordinated) time constantly using GPS services without recalibrating their atomic clocks or performing any synchronization algorithms. In this regard, they provide the time reference for the sensors positioned underwater. If there are two or more beacons, they can communicate with each other through radio frequency (RF) links. Beacons communicate with cluster heads and ordinary nodes through acoustic links. Each cluster has and only has one cluster head. All ordinary nodes connect to their cluster head via single hop. The cluster heads of different clusters connect to a beacon through multiple hops. Figure 1 shows the topology of a UWSN with 8 sensor nodes and one beacon. The beacon is placed on the water surface and is equipped with GPS to obtain UTC time. Each sensor node is assigned a unique identifier (ID). The sensor nodes make autonomous decision about cluster formation without any centralized control.

**Definition 2.** Suppose that the cluster for node \( n_w \) is denoted as \( C_w \); then for each node, \( n_w \in C_w \); if \( C_w = C_u \), then cluster consistency is satisfied; else if there exists a node \( n_w \in C_u \) such that \( C_w \neq C_u \), then it is called cluster inconsistency.

**Definition 2** gives the meanings of cluster consistency and cluster inconsistency. According to Definition 2, each sensor node belongs to one and only one cluster. Different clusters cannot share any common sensor nodes. It is important for sensor nodes to perform cluster consistency checking during the process of cluster formation.

The process of cluster formation for a UWSN is described as in the following step-by-step instructions.

**Step 1.** All sensor nodes with 1-hop distance to a beacon are set to cluster heads.

**Step 2.** Each cluster head will try to add its neighboring nodes as ordinary nodes that belong to its cluster in order of their
degrees from high to low. If a sensor node happens to be the neighboring node of multiple cluster heads, it will be assigned to the cluster head with the smallest ID as an ordinary of that cluster.

**Step 3.** Each ordinary node will try to find its neighboring nodes with 2-hop distance to the cluster head of its current cluster for the purpose of forming new clusters. If there is such a sensor node, the ordinary node will update itself to be the cluster head of a new cluster and add other neighboring nodes with 2-hop distance to its former cluster head into the new cluster as ordinary nodes.

**Step 4.** Repeat Step 2 and Step 3 until all sensor nodes have identities of either cluster heads or ordinary nodes.

**Step 5.** Each sensor node performs cluster consistency checking. If it identifies malicious nodes, it removes them from the network and restarts the cluster formation process from Step 1.

Figure 2 shows the process of cluster formation for a UWSN with 8 sensor nodes. Each circle denotes an ordinary node and each concentric circle denotes a cluster head in the UWSN. The concrete steps are given as follows.

**Step 1.** Node 000 and node 001 are set to cluster heads as they are 1-hop distance to the beacon as shown in Figure 2(a).

**Step 2.** Node 000 tries to add its neighbors as ordinary nodes of its cluster in order of their degrees from high to low, that is, node 011 ($D_{011} = 5$), node 101 ($D_{101} = 4$), and node 010 ($D_{010} = 3$). Likewise, node 001 tries to add its neighbors as ordinary nodes of its cluster in order of node 011 ($D_{011} = 5$), node 110 ($D_{110} = 4$), and node 100 ($D_{100} = 2$). As the ID of node 000 is smaller than that of node 001, node 001 will become an ordinary node in the cluster of node 000 as shown in Figure 2(b).

**Step 3.** The ordinary node 101 in the cluster of node 000 finds that its neighboring node 111 is located 2-hop away from the cluster head node 000. In the same way, the ordinary node 110 in the cluster of node 001 also finds node 111 with 2-hop distance to the cluster head node 001. Since the ID of node 101 is smaller than that of node 110, node 101 will update itself as the cluster head of a new cluster and add node 111 into its cluster as shown in Figure 2(c).

Hereafter, all sensor nodes are assigned identities of either cluster heads or ordinary nodes. After cluster consistency checking, the process of cluster formation is completed. An attacker may participate in the process of cluster formation using malicious nodes. These malicious nodes can launch different attacks in order to introduce cluster inconsistency. However, as each sensor node has a unique ID, all unicast messages exchanged between nodes can be authenticated with unique pair-wise keys shared with other nodes and broadcast messages can be authenticated with public key based digital signatures. By such means, the process of cluster formation can be executed securely.

### 3.2. Time Synchronization Protocol

The time synchronization of CLUSS can be divided into three phases: authentication phase, intercluster synchronization phase, and intracluster synchronization phase. During the authentication phase, all sensor nodes need to be authenticated to each other. Cluster heads are authenticated to beacons and ordinary nodes are authenticated to cluster heads. During the intercluster synchronization phase, cluster heads synchronize themselves with beacons in sender-receiver mode. During the intracluster synchronization phase, ordinary nodes synchronize themselves with cluster heads in receiver-receiver mode. In order to reduce the number of messages generated in synchronization, part of intercluster synchronization phase and intracluster synchronization phase can be executed concurrently. Although many existing sender-receiver and receiver-receiver based clock synchronization protocols can be interpreted by common statistical signal processing methods, it is still too early to say that these techniques can get rid of cluster or tree structure especially in those wireless sensor networks with characteristics of long propagation delay and high mobility. The basic idea of our synchronization protocol is simple but effective in realizing time synchronization even when the network is attacked by malicious nodes. To the best of our knowledge, this is the first work that describes secure synchronization protocol for underwater wireless sensor networks.

#### 3.2.1. Authentication between Nodes

We adopt the following notations throughout the description of the protocol as shown in Notations Section.

After the process of cluster formation, all sensor nodes know ID and nonce information of their neighboring nodes. The compromised nodes may attempt to provide falsified clock values, which in turn make the clock adjustment deviate from its desired value. Therefore, each sensor node must authenticate the trustworthiness of synchronization messages it receives from its neighboring nodes before using them for synchronization. In order to realize authentication in the whole network, beacons play an important role during the whole process of authentication. Firstly, cluster heads will be authenticated to beacons. Suppose that $C$ is a cluster head with 1-hop distance to a beacon $B$. For the purpose of authentication with the beacon $B$, the cluster head $C$ generates its secret key $SK_C$ and signs its nonce $N_C$ with $SK_C$, which then becomes $SK_C(N_C)$. Moreover, the cluster head $C$ also generates a hash value $H_{CBO} = H(ID_C, ID_B, N_C)$ and signs $H_{CBO}$ with $SK_C$, which then becomes $SIG_{SK_C}(H(ID_C, ID_B, N_C))$. Both $SK_C(N_C)$ and $SIG_{SK_C}(H(ID_C, ID_B, N_C))$ are sent to the beacon $B$. On the other hand, the beacon $B$ attempts to decrypt $SK_C(N_C)$ using $C$’s public key $PK_C$ and get the original $N_C$. Furthermore, the beacon $B$ also attempts to decrypt $SIG_{SK_C}(H(ID_C, ID_B, N_C))$ using $C$’s public key $PK_C$ and get the original $H_{CBO}$. After that, the beacon $B$ generates a hash value $H_{CB1} = H(ID_C, ID_B, N_C)$ and compares $H_{CB1}$ with $H_{CBO}$. If $H_{CB1}$ matches $H_{CBO}$, then the beacon $B$ generates another hash value $H_{BCO} = H(ID_B, ID_C, N_B, N_C)$ and signs $H_{BCO}$ with its secret key $SK_B$, which then becomes $SIG_{SK_B}(H(ID_B, ID_C, N_B, N_C))$. In addition, the
The beacon $B$ also signs its nonce $N_B$ with $SK_B$, which then becomes $SK_B(N_B)$. In turn, the beacon $B$ sends $SK_B(N_B)$ and $SIG_{SK_B}(H(ID_B, ID_C, N_B, N_C))$ to the cluster head $C$. After receiving $SK_B(N_B)$ and $SIG_{SK_B}(H(ID_B, ID_C, N_B, N_C))$, the cluster head $C$ decrypts them using $B$'s public key $PK_B$ and get the original $H_{BC0}$ and $N_B$. Moreover, the cluster head $C$ generates a hash value $H_{BC1} = H(ID_B, ID_C, N_B, N_C)$ and compares $H_{BC1}$ with $H_{BC0}$. If $H_{BC1}$ matches $H_{BC0}$, then the authentication is successful; otherwise, the authentication is failed. Likewise, ordinary nodes can be authenticated to cluster heads. After the execution of authentication phase, all neighboring nodes are authenticated with each other and they can communicate with each other securely.

In UWSNs, sensor nodes are usually equipped with crystal oscillators for providing local times. Due to aging and environmental factors (such as temperature, pressure, and...
vibration), these oscillators will gradually exhibit deviations as shown in Figure 3.

Assuming the existence of clock skew and drift, the clock value \( T_A(t) \) can be defined as the following Taylor series [21]:

\[
T_A(t) = \alpha_a t + \beta_a + \gamma_a t^2 + \cdots ,
\]

where \( T_A(t) \) stands for the measured time of the sensor node \( A \); \( t \) is the reference time (UTC time); the linear parameter \( \alpha_a \) denotes the clock skew; the coefficient \( \beta_a \) can be associated with the initial offset of clock; and the quadratic parameter \( \gamma_a \) represents nonlinear deviation from the reference time. However, a nonlinear quadratic model will lead to high computational complexity, which is not suitable for resource constrained UWSNs. Therefore, most synchronization protocols rely on linear model that only takes into account the drift and skew. As shown in Figure 3, vibration), these oscillators will gradually exhibit deviations from ideal UTC time (initial offset equal to 0).

\[
\begin{align*}
T_A(t_1) & = (ID_C, N_C, T_C(t_1), Sync-Req) \\
T_A(t_2) & = (ID_C, N_C, T_C(t_2), Sync-Ack, SIG_{SK_C}(H(ID_B, N_B, T_C(t_1), t_2, t_3, Sync-Ack))) \\
\end{align*}
\]

The basic idea of time synchronization is that the unsynchronized nodes exchange messages with the reference node (already synchronized) in order to estimate clock skews and offsets in local clocks, so that they can correct their local times with that of UTC time and maintain time synchronization by periodical updating their local clocks.

Figure 4 illustrates the process of intercluster and intracluster synchronization in CLUSS.

At first, the cluster head \( C \) generates a synchronization request packet \( Sync-Req \), which consists of destination address, source address, identifier, hops, and the number of iterations. After that, the cluster head \( C \) creates a synchronization request message \( M_1 \) containing its identifier \( ID_C \), nonce \( N_C \), timestamp \( T_C(t_1) \), \( Sync-Req \), and a signature \( SIG_{SK_C} \) over the hash of \( ID_C, N_C, T_C(t_1), Sync-Req \). If the verification is positive, the cluster head \( C \) records its arrival time \( T_C(t_1) \) and the beacon \( B \) sends the synchronization acknowledgement message \( M_2 \) to the cluster head \( C \).

On receiving the synchronization acknowledgement message \( M_2 \), the cluster head \( C \) records its arrival time \( T_C(t_4) \) and verifies its authenticity using \( B \)'s public key \( PK_B \). If the verification is positive, the cluster head \( C \) also needs to calculate the message end-to-end delay. Let \( d_{CB} \) be the deterministic part of the message end-to-end delay between the cluster head \( C \) and the beacon \( B \). It can be precomputed in the deployment phase of the network or estimated via an extra message exchange between the cluster head \( C \) and the beacon \( B \). There are many existing techniques that can also estimate the deterministic delay while estimating the clock skew and offset [22].

3.2.2. Abnormal End-to-End Delay Detection. In order to detect abnormal end-to-end delay data and identify malicious nodes, we consider centered hyperellipsoidal support vector machine (CESVM) [23] as an appropriate scheme for outlier detection in a distributed manner. The basic idea of CESVM is to map the data vectors in the input space to a higher dimensional space, which are centered by means of a
A hyperellipsoid centered at the origin with minimal effective radius is fitted around the majority of the data vectors in the higher dimensional space. The vectors that fall on or inside the hyperellipsoid are classified as normal and those falling outside the hyperellipsoid are classified as outlier values.

Consider a data set \( X = \{ x_i ; i = 1, \ldots, n \} \) of \( p \) variant data vectors \( x_i = \{ x_{i1}, x_{i2}, \ldots, x_{ip} \} \), \( i = 1, \ldots, n \), in the input space which is mapped into a feature space \( \mathbb{R}^f \) using some non-linear function \( \phi(.) : \mathbb{R}^p \rightarrow \mathbb{R}^f \), resulting in a set \( X_\phi = \{ \phi(x_i) ; i = 1, \ldots, n \} \). The aim is to enclose the majority of mapped data vectors in the feature space by fitting a hyperellipsoid centered at the origin with a minimum effective radius \( R(>0) \). Thus, the optimization problem in CESVM classifier is represented as

\[
\min_{R \in \mathbb{R}^+, \xi \in \mathbb{R}^n} \quad R^2 + \frac{1}{n} \sum_{i=1}^{n} \xi_i \\
\text{subject to: } \quad \phi(x_i) \Sigma^{-1} \phi(x_i)^T \leq R^2 + \xi_i, \\
\xi_i \geq 0, \quad i = 1, \ldots, n,
\]

where \( \{ \xi_i ; i = 1, \ldots, n \} \) are the slack variables that allow some of the data vectors to lie outside the hyperellipsoid. The parameter \( v \in [0, 1] \) is used to control the fraction of data vectors that can be outliers. \( \Sigma^{-1} \) is the inverse of the sample covariance matrix \( \Sigma \) of mapped data vectors \( X_\phi \), which is calculated as follows:

\[
\Sigma = \frac{1}{n} \sum_{i=1}^{n} (\phi(x_i) - \mu) (\phi(x_i) - \mu)^T,
\]

\[
u = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i),
\]

where is the sample mean of \( X_\phi \). Using Mercer Kernels [24], the inner products of mapped data vectors in the feature space can be computed in the input data space without knowledge of the nonlinear function \( \phi(.) \) [25]. Each sensor node \( u \) models its own CESVM for \( n \) sequential end-to-end delay values and then calculates the effective radius \( R_u \) as well as the corresponding parameters of the median and the covariance matrix of centered data. Then the local outliers at each sensor node \( u \) can be determined in real time by comparing the distances between new arriving end-to-end delay values and the origin using (4).

3.2.3. Clock Skew and Offset Estimation. In CLUSS, the random delays in uplink and downlink are modeled as independent and identically distributed zero mean Gaussian random variables \( X_{CB} \) and \( Y_{CB} \), respectively, with common variance equal to \( \sigma^2 \). The UTC time \( t_2 \) can be calculated as

\[
t_2 = t_1 + d_{CB} + X_{CB}.
\]

The timestamp of cluster head \( C \) at UTC time \( t_1 \) can be calculated as

\[
T_C(t_1) = \alpha_c (t_2 - d_{CB} - X_{CB}) + \beta_c.
\]

Similarly, the UTC time \( t_4 \) can be calculated accordingly as

\[
t_4 = t_3 + d_{CB} + Y_{CB}.
\]

Moreover, the timestamp of cluster head \( C \) at UTC time \( t_4 \) can be calculated as

\[
T_C(t_4) = \alpha_c (t_3 + d_{CB} + Y_{CB}) + \beta_c.
\]

Assuming that \( t_3 = t_2 + \epsilon \), \( z_c = [(T_C(t_1) - t_2) + (T_C(t_1) - t_3)]/2 \), then we get

\[
z_c = \alpha_c \left[ t_2 + \frac{\epsilon}{2} + \frac{Y_{CB} - X_{CB}}{2} \right] + \beta_c + \frac{Y_{CB} - X_{CB}}{2}.
\]

Let \( u_c = (Y_{CB} - X_{CB})/2 \), as \( t_2 \gg \epsilon/2 + (Y_{CB} - X_{CB})/2 \); then we get

\[
z_c = \alpha_c t_2 + \beta_c + u_c,
\]

where \( u_c \) obeys Gaussian distribution of \( N(0, \sigma^2/2) \).

After \( k \) iterations of (II) with MAC timestamps, cluster head \( C \)'s clock skew and offset can be estimated.

And then, cluster head \( C \) generates a Sync-Ack packet and broadcasts the synchronization acknowledgement message \( M_3 \) to all ordinary nodes in its local cluster. After receiving and verification of the authenticity \( M_3 \), these ordinary nodes can synchronize themselves with cluster head \( C \). For example, the ordinary node \( A \) in the cluster of \( C \) can calculate its estimated time \( t_1 \) as

\[
\tilde{t}_1 = \frac{T_C(t_1) - \tilde{\beta}_c}{1 + \tilde{\alpha}_c}.
\]

Let \( d_{AC} \) denote the deterministic part of the message end-to-end delay between ordinary node \( A \) and cluster head \( C \). Also, let the random portion of the downlink delay \( X_{AC} \) be a zero mean Gaussian random variable with variance \( \sigma^2 \); then the estimated UTC time \( t_1 \) of ordinary node \( A \) can be calculated as

\[
\tilde{t}_2 = \tilde{t}_1 + \tilde{d}_{AC} + X_{AC}.
\]

The timestamp of ordinary node \( A \) at UTC time \( t_2 \) can be calculated as

\[
T_A(t_2) = \alpha_a (\tilde{t}_2 + \tilde{d}_{AC} + X_{AC}) + \beta_a.
\]

Applying (12) to (13), we can get

\[
T_A(t_2) = \alpha_a (\tilde{t}_1 + \tilde{d}_{AC} + X_{AC}) + \beta_a.
\]

Applying (12) to (15), we can get

\[
T_A(t_2) = \alpha_a \left( T_C(t_1) - \tilde{\beta}_c + d_{AC} + X_{AC} \right) + \beta_a.
\]

Similarly, we can also get the linear model of \( T_A(t_2) \) and \( T_C(t_3) \), which can be used to calculate ordinary node \( A \)'s clock skew and offset together with (16).

In the same way, cluster head \( C \)'s neighboring clusters can synchronize their clock times by treating cluster head \( C \) as their reference node. This process is repeated until all sensor nodes in the network are synchronized.
3.3. Security Analysis. In this section we discuss how different attacks are prevented by the proposed protocol. We assume that beacons cannot be captured by an attacker as they are physically protected.

In Sybil attack, an attacker tries to create multiple fabricated identities. In our protocol, all sensor nodes along the communication path are authenticated using Elliptic Curve Cryptography (ECC) public-key algorithm, which is feasible for low-end underwater sensors [26]. The sender signs the hash of a synchronization message with its secret key and broadcasts the signature along with the message. The receivers verify the correctness of the signature by using the sender’s public key. As a result, an attacker cannot disguise as a legitimate node and cannot exchange any false information between legitimate nodes.

Replay attack is prevented by using a random nonce, which is updated every time the synchronization message which is broadcasted. Thus, an attacker cannot pass the authentication phase even if a malicious node makes a copy of the previous synchronization message.

In order to perform message manipulation attack, an intruder tries to drop, modify, or even forge the exchanged messages to interrupt the process of synchronization. In the proposed protocol, all unicast messages exchanged between nodes can be authenticated with unique pair-wise keys shared with other nodes and broadcast messages can be authenticated with public key based digital signatures. Thus, message manipulation attack is not effective with this protocol.

In a delay attack, an attacker intentionally delays some of the timing messages in order to disrupt the synchronization process and cause it to fail. Our protocol exploits centered hyperellipsoidal support vector machine (CESVM) in order to detect abnormal end-to-end delay data and identify malicious nodes in a distributed manner. All detected abnormal end-to-end delay data will be aborted without using in computation of clock skew and offset. Thus, this attack cannot take place if the proposed protocol is in use.

Of course, obtaining absolutely secure and labeled data for training a classifier could be an impossible task. That is the reason that we consider CESVM as an appropriate scheme for detecting abnormal end-to-end delay data. CESVM belongs to one-class SVM that can model the normal profile of the monitored data while automatically ignoring the errors or anomalies present in the data using unsupervised anomaly detection techniques.

3.4. Complexity Analysis. In order to execute time synchronization protocol completely, each sensor node consumes a certain amount of memory space for storing the information of its neighboring nodes and propagates a finite number of timing messages for synchronization. Moreover, each sensor node should also calculate its clock skew and offset, which will incur nonignorable computational cost. Let \(|V|\) denote the total number of sensor nodes in the network. \(D_{\text{max}}\) and \(D_{\text{avg}}\) denote the maximum degree and the average degree of the nodes in the network, respectively. Then the computational complexity is proportional to the square of the maximum degree of the node according to the definition of Breadth-First Search, which can be represented as \(O(D_{\text{max}}^2)\).

The number of clusters in the network is calculated as \(|V|/D_{\text{avg}}\). Suppose that the number of messages generated in synchronization is \(M\); then the communication complexity is calculated as \(O(M \cdot |V|^2/D_{\text{avg}}^2)\). As a distributed time synchronization protocol, each sensor node only stores the information of its neighboring nodes within 1-hop distance in CLUSS. Therefore, the memory complexity is \(O(D_{\text{max}})\) for each sensor node and \(O(|V| \cdot D_{\text{max}})\) for the whole network in the worst case.

### Table 1: Simulation parameters and their default values.

| Simulation parameters | Default values |
|-----------------------|----------------|
| Number of beacons     | 3              |
| Number of sensor nodes| 50             |
| The size of region    | 1km \times 1km \times 1km |
| Average node degree   | 4.6            |
| Average node speed    | 1 m/s          |
| Initial clock skew    | 50 ppm         |
| Initial clock offset  | 80 ppm         |
| Time granularity      | 1 \(\mu\)s     |
| Jitters in receivers  | 10 \(\mu\)s    |
| Time interval of messages | 1 s           |

4. Performance Evaluation

4.1. Simulation Settings. We use Aqua-Sim [27] as simulation framework to evaluate our approach. The data packets are also generated by the simulator. Aqua-Sim is an ns-2 based underwater sensor network simulator developed by underwater sensor network lab at University of Connecticut. Aqua-Sim can simulate the attenuation and propagation of acoustic signals. It can also simulate packet collision in underwater sensor networks.

In simulations, we assume that beacons own UTC time and have unlimited energy resources. Other sensor nodes have their own internal clocks and the errors for processing and exchanging of messages among sensor nodes obey Gaussian distribution [28]. The R-MAC [27] protocol is used as the underlying MAC protocol in order to avoid data packet collision and produce timestamp during the process of simulation. The sensor nodes follow the random-walk mobility pattern, which randomly select a direction and move to the new position with a speed between 1 m/s and 5 m/s. Other simulation parameters and their default values are listed in Table 1.

CESVM is less sensitive to the choice of parameter values than other SVMs. For example, CESVM can achieve an AUC (Area Under roc Curve) in excess of 0.8 for a wide range of kernel parameter values which vary from 0.08 to 41943.04 [22]. Moreover, the effective radius for the CESVM is computed either using any border support vector or using the mean of the support vectors if such border support vector not exists. Therefore, we do not individually discuss the effect of parameters for CESVM on the final results in this paper.
We compared the performance of CLUster-based Secure Synchronization (CLUSS) protocol with that of MU-Sync [16] and WATERSync [17]. Without loss of generality, every data point is obtained as the average of 1000 simulation runs.

4.2. Simulation Results. In the first set of simulations, we compared the errors with time elapsed since synchronization in different synchronization protocols without malicious nodes. The number of sensor nodes is set to 50. As shown in Figure 5, the errors of three synchronization protocols are proportional to the time after synchronization. CLUSS achieves lower errors than that of MU-Sync and WATERSync in the same circumstances. The average error of CLUSS is only 42.4% of MU-Sync and 28.8% of WATERSync after synchronization of $10^5$ seconds. Moreover, compared with Figure 5, the errors of all three protocols increase to some extent. The curve of MU-Sync rises sharply because it has no prevention measures against malicious nodes. As a result, the precision of WATERSync overtakes that of MU-Sync as WATERSync can detect outlier timestamp data and identify some attacks generating by malicious nodes.

Next, we compared the errors with time elapsed since synchronization in different synchronization protocols with 10% malicious nodes. As shown in Figure 6, the errors of three synchronization protocols are proportional to the time after synchronization. CLUSS achieves lower errors than that of MU-Sync and WATERSync in the same circumstances. The average error of CLUSS is only 15.4% of MU-Sync and 24.1% of WATERSync after synchronization of $10^5$ seconds. Moreover, compared with Figure 5, the errors of all three protocols increase to some extent. The curve of MU-Sync rises sharply because it has no prevention measures against malicious nodes. As a result, the precision of WATERSync overtakes that of MU-Sync as WATERSync can detect outlier timestamp data and identify some attacks generating by malicious nodes.

And then, we compared the errors with average node speed in different synchronization protocols with 10% malicious nodes. The synchronization time is set to 10 seconds. As shown in Figure 7, the errors of three synchronization protocols are proportional to the average node speed. MU-Sync obtains the highest synchronization error among the three synchronization protocols when their average node speeds are same. Moreover, with the increase of average node speed, the curve of MU-Sync shows a rapid rising compared with other synchronization protocols. The curve of WATERSync indicates that the error is approximately linear with the average node speed. The error of CLUSS is obviously lower than that of MU-Sync and WATERSync. Furthermore, with the increase of average node speed, the accumulated errors of CLUSS display a mild growth. On average, the error of CLUSS is only 25.6% of MU-Sync and 57.6% of WATERSync.

After that, we compared the errors with the number of nodes in different synchronization protocols with 10% malicious nodes. The synchronization time is also set to 10
seconds. As shown in Figure 8, the errors of three synchronization protocols are proportional to the number of nodes. CLUSS achieves the lowest errors among the three protocols in the same circumstances. The average error of CLUSS is only 34.3% of MU-Sync and 72.5% of WATERSync.

In addition, we also compared the errors with the rate of malicious nodes in different synchronization protocols. The number of sensor nodes is set to 50 and the average node speed is set to 1 m/s as default. As shown in Figure 9, the errors of three synchronization protocols are proportional to the rate of malicious nodes. CLUSS achieves lower errors than that of MU-Sync and WATERSync in the same circumstances. The curve of CLUSS is also gentler than that of MU-Sync and WATERSync. When the rate of malicious nodes is not above 6%, the error of MU-Sync is slightly lower than that of WATERSync. When the rate of malicious nodes is not less than 10%, the error of MU-Sync becomes higher than that of WATERSync. Overall, the average error of CLUSS is only 34.4% of MU-Sync and 62.3% of WATERSync.

In the second set of simulations, we compared the number of messages with time elapsed since synchronization in different synchronization protocols with 10% malicious nodes. The synchronization time is set to 10 seconds. As shown in Figure 10, the number of messages is proportional to the time elapsed since synchronization. CLUSS produces the minimum number of messages, while WATERSync generates the maximum number of messages. Moreover, the curve of CLUSS indicates gentler slope compared with that of MU-Sync and WATERSync. Overall, CLUSS decreases 71% of the number of messages than that of MU-Sync and 32.7% of the number of messages than that of WATERSync on average after time synchronization. This is because part of intercluster synchronization phase and intracluster synchronization phase in CLUSS can be executed concurrently, which in turn reduces the number of messages generated in synchronization.
After that, we compared the number of messages with the number of nodes in different synchronization protocols with 10% malicious nodes. As shown in Figure 11, the number of messages is proportional to the number of nodes in three synchronization protocols. CLUSS produces fewer messages than that of MU-Sync and WATERSync with the same number of nodes. Moreover, the slope of the curve in CLUSS is gentler than that of MU-Sync and WATERSync. Overall, CLUSS decreases 14.9% of the number of messages than that of MU-Sync and 35.4% of the number of messages than that of WATERSync on average.

In the last set of simulations, we compared the number of iterations with error tolerance in different synchronization protocols as shown in Figure 12. The amount of energy consumed by resynchronization is represented by the number of iterations over a period of $10^5$ seconds of operating time for various values of error tolerance. The number of iterations is inversely proportional to error tolerance in three synchronization protocols. Under the same condition, CLUSS outperforms MU-Sync and WATERSync due to its ability to estimate skew more accurately. On average, CLUSS only needs 46.5% of iterations compared to MU-Sync and 22.1% of iterations compared to WATERSync.

4.3. Discussion. When an existing sensor node leaves the network due to battery outage, if this sensor node is not the cluster head of any cluster, then there is nothing to do. If this sensor node is a cluster head, then the system will execute the process of cluster formation and assign new cluster heads for the newly formed clusters if needed. Furthermore, the new sensor node should also synchronize its clock with the whole network. Therefore, the cost for cluster maintenance cannot be neglected in highly dynamic and unstable networks where nodes frequently join and leave. However, in a UWSN, each sensor node is attached
to a surface buoy by a wire that can be adjusted for proper and suitable length or anchored to the ocean bottom using weights. Moreover, the network battery life is sufficiently long to avoid recharging or replacing node batteries frequently in harsh underwater environments. Hence, the topology of the network does not change frequently after initial deployment and the cost for cluster maintenance only has a trilling impact on the proposed method in a UWSN.

5. Conclusion

This paper proposed a CLUster-based Secure Synchronization (CLUSS) protocol that ensures the security of synchronization under harsh underwater environments against various attacks, including Sybil attack, replay attack, message manipulation attack, and delay attack. The time synchronization process of CLUSS is divided into three phases: authentication phase, intercluster synchronization phase, and intracluster synchronization phase. During the authentication phase, cluster heads are authenticated to beacons and ordinary nodes are authenticated to cluster heads. During the process of intercluster and intracluster synchronization phases, CLUSS distinguishes the propagation delay of downlink from that of uplink caused by node movement in order to improve time synchronization accuracy. Moreover, part of these two phases can be executed concurrently for reducing the number of messages generated in synchronization. We demonstrate through simulations that CLUSS can reduce synchronization errors as well as the number of synchronization messages than traditional protocols even when the network is attacked by malicious nodes. In future work, we will investigate the influence of MAC layer activities to the performance of time synchronization protocols, such as packet loss and retransmission. Moreover, we will analysis the adaptability of our protocol and evaluate its performance through real ocean experiments.

Notations

| ID:      | The identifier                  |
| SK:      | The secret key                  |
| PK:      | The public key                  |
| $H(M)$:  | The hash function on message $M$|
| $N_A$:   | A nonce generated for node $A$  |
| $\text{Sig}_{SK_A}(M)$: | The signature of $M$ signed by SK of node $A$. |

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was sponsored by the National Nature Science Foundation of China (61202370, 51279099), the Innovation Program of Shanghai Municipal Education Commission (14YZ110), the Shanghai Pujiang Program from Science and Technology Commission of Shanghai Municipality (11PJ1404300), and the Open Program of Shanghai Key Laboratory of Intelligent Information Processing (IIPJ-2011-008).

References

[1] R. B. Manjula and S. M. Sunilkumar, “Issues in underwater acoustic sensor networks,” International Journal of Computer and Electrical Engineering, vol. 3, no. 1, pp. 1793–8163, 2011.
[2] Y. G. Chen, X. M. Xu, and L. Zhang, “Design of RC-LDPC codes and its application for shallow water acoustic communications,” Journal of Convergence Information Technology, vol. 7, no. 12, pp. 177–185, 2012.
[3] W. Shen, H. Sun, E. Cheng, and Y. Zhang, “SNR estimation algorithm based on pilot symbols for DFT-spread OFDM systems over underwater acoustic channels,” Journal of Convergence Information Technology, vol. 6, no. 2, pp. 191–196, 2011.
[4] S. Basagni, C. Petrioli, R. Petroccia, and M. Stojanovic, “Optimized packet size selection in underwater wireless sensor network communications,” IEEE Journal of Oceanic Engineering, vol. 37, no. 3, pp. 321–337, 2012.
[5] J. M. Jornet, M. Stojanovic, and M. Zorzi, “On joint frequency and power allocation in a cross-layer protocol for underwater acoustic networks,” IEEE Journal of Oceanic Engineering, vol. 35, no. 4, pp. 936–947, 2010.
[6] C. Detweiler, M. Doniec, I. Vasilescu, and D. Rus, “Autonomous depth adjustment for underwater sensor networks: design and applications,” IEEE/ASME Transactions on Mechatronics, vol. 17, no. 1, pp. 16–24, 2012.
[7] I. F. Akyildiz, D. Pompili, and T. Melodia, “Underwater acoustic sensor networks: research challenges,” Ad Hoc Networks, vol. 3, no. 3, pp. 257–279, 2005.
[8] J. Liu, Z. Zhou, Z. Peng, and J.-H. Cui, “Mobi-sync: efficient time synchronization for mobile underwater sensor networks,” in Proceedings of the 4th ACM international Workshop on Underwater Networks, pp. 1–5, 2009.
[9] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” in Proceedings of the 5th Symposium on Operating Systems Design and Implementation, pp. 147–163, 2002.
[10] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync protocol for sensor networks,” in Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys ’03), pp. 138–149, November 2003.
[11] M. Marotti, B. Kusy, G. Simon, and Á. Lédeczi, “The flooding time synchronization protocol,” in Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys ’04), pp. 39–49, November 2004.
[12] A. S. Uluagac, R. A. Beyah, and J. A. Copeland, “Secure source-based loose synchronization (S0BAS) for wireless sensor networks,” IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 4, pp. 803–813, 2013.
[13] Y. Liu, J. Li, and M. Guizani, “Lightweight secure global time synchronization for wireless sensor networks,” in Proceedings of the IEEE Wireless Communications and Networking Conference: Mobile and Wireless Networks, pp. 2312–2317, 2012.
[14] A. A. Syed and J. Heidemann, “Time synchronization for high latency acoustic networks,” in Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM ’06), pp. 1–6, April 2006.
[15] F. Lu, D. Mirza, and C. Schurger, “D-Sync: doppler-based time synchronization for mobile underwater sensor networks,” in Proceedings of the 5th ACM International Workshop on UnderWater Networks (WUWNet ’10), pp. 1–8, October 2010.

[16] N. Chirdchoo, W.-S. Soh, and K. C. Chua, “MU-Sync: a time synchronization protocol for underwater mobile networks,” in Proceedings of the 3rd ACM International Workshop on Underwater Networks (WUWNet ’08), pp. 35–42, September 2008.

[17] F. Hu, S. Wilson, and Y. Xiao, “Correlation-based security in time synchronization of sensor networks,” in Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC ’08), pp. 2525–2530, April 2008.

[18] G. T. Toussaint, “The relative neighbourhood graph of a finite planar set,” Pattern Recognition, vol. 12, no. 4, pp. 261–268, 1980.

[19] P. Bose, L. Devroye, W. Evans, and D. Kirkpatrick, “On the spanning ratio of gabriel graphs and $\beta$-skeletons,” SIAM Journal on Discrete Mathematics, vol. 20, no. 2, pp. 412–427, 2006.

[20] K. J. Supowit, “The relative neighborhood graph with an application to minimum spanning trees,” Journal of Association for Computing Machinery, vol. 30, no. 3, pp. 428–448, 1983.

[21] J. A. Fernandez, J. M. Ortega, A. G. Hernandez et al., “Performance analysis of time synchronization protocols in wireless sensor networks with regular topologies,” International Journal on Advances in Security, vol. 3, no. 1-2, pp. 13–26, 2010.

[22] X. Cao, F. Yang, X. Gan et al., “Joint estimation of clock skew and offset in pairwise broadcast synchronization mechanism,” IEEE Transactions on Communications, vol. 61, no. 6, pp. 2508–2521, 2013.

[23] S. Rajasegarar, C. Leckie, J. C. Bezdek, and M. Palaniswami, “Centered hyperspherical and hyperellipsoidal one-class support vector machines for anomaly detection in sensor networks,” IEEE Transactions on Information Forensics and Security, vol. 5, no. 3, pp. 518–533, 2010.

[24] V. N. Vapnik, Statistical Learning Theory, John Wiley & Sons, 1998.

[25] V. N. Vapnik, “An overview of statistical learning theory,” IEEE Transactions on Neural Networks, vol. 10, no. 5, pp. 988–999, 1999.

[26] H. Yan, Z. J. Shi, and Y. Fei, “Efficient implementation of elliptic curve cryptography on DSP for underwater sensor networks,” in Proceedings of the 7th Workshop on Optimizations for DSP and Embedded Systems, pp. 7–15, 2009.

[27] P. Xie, Z. Zhou, Z. Peng et al., “Aqua-sim: an NS-2 based simulator for underwater sensor networks,” in Proceedings of the MTS/IEEE Marine Technology for Our Future: Global and Local Challenges (OCEANS ’09), pp. 1–7, Biloxi, Miss, USA, October 2009.

[28] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” in Proceedings of the 5th Symposium on Operating Systems Design and Implementation, pp. 147–163, 2002.
