Time Synchronization for Underwater Sensor Networks Based on Multi-Source Beacon Fusion

Feng Hong Bozhen Yang Yuliang Zhang Ming Xu Yuan Feng Zhongwen Guo
College of Information Science and Enigeering, Ocean University of China, Qingdao, China
{hongfeng, mingxu, fengyuan, guozhw}@ouc.edu.cn, {yangbozhen, yuliangzhang0030}@gmail.com

Abstract—Time synchronization is the foundation for collaboration among sensor nodes. Acoustic communication and restricted mobility of sensor nodes are two unique characteristics of underwater sensor networks (UWSNs), negating the effect of terrestrial synchronization schemes. Existing UWSN time synchronization protocols focus on point to point synchronization, incompatible for large-scale networks. Moreover, some of them require special hardwares and deployment conditions. In this paper, we propose MulSync, a scalable synchronization protocol for multi-hop UWSNs. MulSync includes the synchronization communication scheme to exploit acoustic communication nature of broadcast. The skew and offset are estimated by performing linear regression three times over a set of time stamp pairs gathered through message exchange. Three linear regressions are exploited to make full use of the time reference information delivered from multi-source beacons. Simulation results demonstrate that MulSync achieves high accuracy at low message overhead and time cost.

Keywords: Time synchronization, underwater sensor networks, fusion, acoustic

I. INTRODUCTION

Time synchronization is the fundamental problem in distributed sensor networks [1]. Specifically, time stamp is an important element of sensing data. As and as the basis for sensor networks to collaborate, time synchronization is required by TDMA [2], localization [3], duty cycling [4] and information fusion [5].

Time synchronization for Underwater Sensor Networks (UWSNs) is more challenging because of two unique characteristics. First, UWSNs communicate through acoustic signals with low propagation speed, approximately 1500 m/s, leading to significant propagation delay comparing to RF signals [6]. Second, the restricted mobility of sensor nodes caused by ocean current, wind or tide brings new challenge to propagation delay estimation. It creates difference in propagation delay between two-way message exchanging and among multiple message transmissions from one node to another. Time synchronization protocols for terrestrial sensor networks, such as TPSN [7], FSTP [8] and RBS [9], cannot be directly applied to UWSNs, since all of them assume that the propagation delay is negligible.

Existing UWSN time synchronization protocols target on the problem of long propagation delay of acoustic communication. According to the need for special hardware or deployment condition, they can be further divided into two categories: 1) protocols requiring no extra hardwares, like TSHL [10] and MU-Sync [11]. However, TSHL assumes that the propagation delay is of negligible difference among multiple time message transmissions from one node to another. MU-Sync makes the assumption that the propagation delay difference is little between two-way message exchanging. 2) Mobi-Sync [12], E2DTS [13], D-Sync [14] and TSMU [15] exploit special hardwares or deployment conditions. For Mobi-Sync, beacon nodes should be equipped with speed sensors, and there should be three beacons in the communication range of every ordinary node. E2DTS uses Autonomous Underwater Vehicle (AUV) as its beacon. Doppler meter should be equipped for both D-Sync and TSMU.

Moreover, all existing UWSN synchronization protocols focus on single-hop synchronization. For multi-hop scenarios, all these protocols can only be applied in hop by hop manner. The reason is that it is difficult to deal with time reference packets from different beacons. Even the ordinary node is just one hop away from two beacons, the different propagation delays for these two links make estimation on the local skew and offset difficult. It is, needless to say, harder when time reference packets coming through diverse paths from multiple beacons.

Multi-hop scenario and low density of beacons (just a few) bring out some new challenges for synchronization design as follows: how to distribute the time reference packets from beacons to ordinary nodes? How to exploit time reference packets coming through diverse paths from multiple beacons? Last but not least, how to limit the number of synchronization messages exchanging while keeping the synchronization accuracy?

In this paper, we propose the time synchronization approach, called MulSync, for multi-hop UWSNs with only a little number of beacons. Without any additional hardware and deployment condition requirements, MulSync estimates the skew and offset for ordinary nodes, by performing linear regression three times over a set of time stamp pairs gathered through different neighbors, traced back from multiple beacons. The time stamp set for estimation is constructed via the synchronization message communication scheme of MulSync, which is initiated by ordinary nodes and exploits
acoustic communication nature of broadcast to limit the communication overhead. Major contributions of this paper are as follows:

- Through exploiting time reference packets from different links traced back from multiple beacons, MulSync presents three times of linear regression to estimate the local skew and offset.
- In order to reduce the communication overhead on synchronization messages, MulSync provides the communication scheme, combining the synchronization request for beacons and response for ordinary nodes through one broadcast.
- The simulation results demonstrate that MulSync achieves high-level synchronization precision, with low message overhead and time cost, comparing with MultiSync.

The remainder of the paper is as follows. We discuss the special characteristics of UWSNs on time synchronization in Section II. The design details of MulSync is provided in Section III. Section IV demonstrates the simulation results. We then describe the previous works on time synchronization for UWSNs in Section V. Finally, Section VI concludes the paper.

II. UWSN CHARACTERISTICS

Figure 1 shows the sketch architecture of UWSNs. Comparing to terrestrial sensor networks, three characteristics have the great impact on synchronization, listed as follows:

- Long propagation delay. Acoustic signal has the propagation speed of approximately 1500 m/s in underwater scenario, five orders of magnitude slower than radio-frequency signal. And the deployment distance between sensor nodes is about 0.5km to 2km [6]. So the propagation delay on one link is about 666ms when the distance between two nodes is 1km.
- Restricted mobility of nodes. Sensor nodes usually float in the water tied by cables, and anchored at ocean bottom as depicted in Fig. 1. Due to ocean current, wind or tide, sensor nodes move in a restricted area, making the propagation delay between a pair of nodes dynamic. The largest variance on two-way propagation delay can reach 3ms, when the moving speed is 1.67m/s with the link about 1km. The movement of sensor nodes brings diversity in propagation delay for the same link, too. Moreover, the variation of propagation speed caused by temperature, density and salinity, makes the delay difference even more complicated.
- Poor acoustic channel quality. Acoustic channel suffers a lot of signal loss in the water, such as spreading and absorption loss; multi-path reflection from the surface, obstacles, and the bottom; noise due to natural and artificial sources [6]. To improve network efficiency, multiple base stations are usually deployed in UWSNs. For the reason that each base station can play the role of time beacon, this may help in solving the synchronization problem for multi-hop UWSNs.

III. DESIGN

MulSync is designed to carry out synchronization in multi-hop UWSNs, where most nodes cannot communicate directly with beacon nodes. This section first provides an overview of MulSync, then describes the design details separated into three phases of hop detection, synchronization message communication and synchronization calculation.

Hop detection is to discover the local topology for each node, especially to find the neighbors closer to or farther from beacons. The second phase is for communication of synchronization request and response among ordinary nodes and beacons. Broadcasting will be fully exploited in this phase. All ordinary nodes will get enough synchronization time stamp pairs after this phase. The last phase intersects with the second phase. When one ordinary node has enough synchronization information, it will estimate both skew and offset and broadcast its estimation result out.

In order to illustrate the design clearly, we give an example deployment as shown in Fig. 2. It consists of two beacon nodes and 23 ordinary nodes. Each node’s id is labelled close to the node.
A. Hop Detection

In the first phase, all beacons broadcast one notification message to announce their existence. The notification message is a tuple of two items, containing beacon node id, hop of 1. When an ordinary node first receives one notification message, it records the hop number as its hop. Figure 2 shows every node’s hop in the brackets. The ordinary node records id and hop in the packet in its low hop neighbor list. After replacing id to its own id and adding one hop number, it rebroadcasts out this message immediately.

Upon receiving other notification messages, the ordinary node first checks whether the hop number in the packet is lower than its hop. If the message is broadcasted by its lower hop neighbor, it adds id to its low hop neighbor set. If higher, it records in its high hop neighbor set. Otherwise, it ignores the message.

The whole network will be covered by broadcasting of notification messages. After this phase, every ordinary node knows not only its hop number to beacon nodes, but also the neighbors with their hop information. For example, node N_3 determines its hop = 2, and has a low and high hop neighbor list of \{N_4, N_8\} and \{N_2, N_6\} in Fig. 2.

B. Synchronization Message Communication

In order to compensate the restricted mobility of sensor nodes, MU-Sync employs two-way synchronization message exchanging similar to MU-Sync [11]. The process for the ordinary node one-hop away from a beacon is shown in Fig. 3. The ordinary node sends out one synchronization request to the beacon at T_1. Upon receiving the synchronization request, the beacon marks the time t_2 and responds to the request node at t_3, informing the time stamps of t_2 and t_3. The ordinary node records the time T_4 when receiving the response. It gets \( < T_1, t_2, t_3, T_4 > \), called synchronization tuple. T_1, T_4 are the ordinary node’s local time, and t_2, t_3 are the reference time from the beacon. In order to guarantee certain synchronization accuracy, this process will be iterated for n times. The duration between two requests is called Req_INT. And the whole process should be repeated upon some period, for the time difference will accumulate to make last synchronization lose efficacy.

However, most ordinary nodes cannot connect to beacon nodes in one hop, under the scenario of multi-hop UWSNs with a few number of time beacons. MulSync communication scheme deals with the process of exchanging of synchronization messages, and exploits the broadcast nature of acoustic communication to eliminate message number.

The node first starts the timer of Req_INT. Then it determines whether it is of the highest hop in its neighborhood, by checking its high hop neighbor list. If so, it broadcasts the synchronization request. The packet of synchronization request is the tuple of \(< m_{id}, s_{id}, hop, flag >\). m_{id} is the locally unique identifier for the message. s_{id} and hop are assigned with the current node information. flag indicates whether its lower hop neighbor should response to this request, labeling true for the origin request. Such requests will be broadcasted periodically with the interval Req_INT. The total request number is \( \lceil \frac{n}{m} \rceil \), instead of n requests in MU-Sync, where m is the number of low hop neighbors. The reason will be discussed later. In Fig. 2, node N_1 of hop = 4 will broadcast its request of \(< 1, N_1, 4, true >\) first.

Other ordinary nodes waits for the synchronization requests from its high hop neighbors. If it receives no request during the first period of Req_INT, it will broadcast its own synchronization request and start the timer again. For example, node N_3 may not receive any request in its first period of Req_INT in Fig. 2. Upon receiving a synchronization request, an ordinary node first checks the request from the neighbor of lower hop, higher hop or equal hop.

If from the higher hop neighbor, the ordinary node checks flag, if it is true, current node constructs a new synchronization request of \(< m_{id}, s_{id}, hop, flag, pm_{id}, r_{id}, t_2, t_3 >\). The first four items have the same contents as illustrated above. The next two \( pm_{id}, r_{id} \) are copied from the request message’s \( m_{id} \) and \( s_{id} \). t_2 is the local time on receiving the request and t_3 is the local time on broadcasting out the new request.

The new request message will be broadcasting out when timer Req_INT is up. If the synchronization requests from other higher hop nodes arrive before firing of timer Req_INT, the node will also add \( < pm_{id}, r_{id}, t_2 > \) of the request to the end of its new request. t_3 will not be changed for it depends on the timer Req_INT. e.g. node N_1 may send out a request \(< 2, N_3, 2, true, 1, N_2, t_2(N_3, 1), t_3, 1, N_6, t_2(N_6, 1) >\). The broadcasting of this new request plays two roles, as the synchronization request of current node and as the response to its high hop neighbors.

If the ordinary node receives some requests from its lower hop neighbors, it first examines whether this message contains a response to its previous request. If a pair of \( < pm_{id}, r_{id} > \) matches previous request, it records a synchronization tuple of \(< T_1, t_2, t_3, T_4, s_{id} >\), where s_{id} labels the transmitted node, and T_4 marks the receive time. Although t_2 and t_3 are not provided by some beacon, they
will play some role as time reference for synchronization calculation.

As one request will be answered by all of its lower hop neighbors, the number of requests needs only \( \left\lfloor \frac{n}{m} \right\rfloor \) for one ordinary node to get enough \( n \) tuples of \( <T_1, t_2, t_3, T_4, s_{id}> \) for skew and offset estimation. Taking packet dropping or error into consideration, the node keeps broadcasting at every \( \text{Req}_{INT} \) cycle until it collects \( n \) synchronization tuples.

If the message does not contain any response to current node, it still cannot estimate its own skew and offset. For example, node \( N_8 \) in Fig. 2, it uses one-hop neighbors, the number of requests needs only \( \left\lfloor \frac{n}{m} \right\rfloor \) for one ordinary node to get enough \( n \) tuples of \( <t_3, t_4, s_{id}> \), called broadcast tuple, for future use in estimating its skew as described in next subsection. \( T_3 \) also indicates the time upon receiving. If the request is from the node of the same hop number, current node just ignores the request. Beacons act as ordinary nodes in MulSync, but do not initiate synchronization request.

C. Synchronization Calculation

The synchronization message broadcasts are initiated by the nodes of highest hop number, but the synchronization calculation begins at the ordinary node close to beacons. Because even if an ordinary node collects enough synchronization tuples \( <T_1, t_2, t_3, T_4, s_{id}> \), more than \( n \), it still cannot estimate its own skew and offset. For \( t_2 \) and \( t_3 \) are the local time stamp of node \( s_{id} \), most of which are not beacons. There are some differences in the estimation process between ordinary nodes one hop and multiple hops away from beacon nodes. We illustrate the calculation process of ordinary nodes close to beacon nodes first.

For the node labelled as \( x \) just one hop away from the beacon node, all its synchronization tuples come from one beacon i.e. \( s_{id} \) indicates the same beacon in all its synchronization tuples. After collecting \( n \) synchronization tuples, it begins the calculation process. The estimation contains twice of linear regression. Node \( x \) performs the first linear regression, over the set of \( <t_3, T_4, s_{id}> \) not only extracting from its synchronization tuples, but also from broadcast tuples. The target is the first estimation on skew \( \alpha_x \). For node \( N_4 \) in Fig. 2, it uses two kinds of time tuples in estimation, getting from \( B_1 \) responding to its own requests and to node \( N_8, N_9 \).

The value of \( \alpha_x \) is then used to estimate one way propagation delay to node \( s_{id} \) for each synchronization tuple as in Equation (1).

\[
D_{s_{id}}^{t_{id}} = \frac{1}{2} \left[ \frac{T_4 - T_3}{\alpha_x} + (t_2 - t_3) \right] \tag{1}
\]

The estimation of propagation delay between current node and its neighbor, will be further removed for the set of \( <t_3, T_4, s_{id}> \). The new time pairs of \( <t_3, T_4 - D_{s_{id}}^{t_{id}}> \) are used for the second linear regression. The final estimated skew and offset of node \( x \) is calculated, and denoted by \( \alpha_x \) and \( \beta_x \). Once completing synchronization estimation, the ordinary node broadcasts its estimation results.

Other ordinary nodes wait for the estimation results from its lower hop neighbor. For node \( N_3 \), both synchronization and broadcast tuples are constructed from broadcast by node \( N_4 \) and \( N_8 \). Hence the sets of \( <t_3, T_4, N_3> \) and \( <t_3, T_4, N_8> \) cannot be used to estimate \( \alpha_3 \) and \( \beta_3 \), unless node \( N_4 \) and \( N_8 \) broadcast their estimation results. Upon receiving \( N_4 \)’s estimation notification, it will recompute all the time stamps concerning \( N_4 \) with Equation (2).

\[
t_4 = \frac{t_4 - \tilde{b}_1}{\tilde{a}_4} \tag{2}
\]

For example, node \( N_3 \) changes the synchronization tuple \( <t_1, t_2, t_3, T_4, N_4> \) to \( <t_1, t_2 - \frac{t_3 - t_4}{\alpha_4}, t_3 - t_4, T_4, N_4> \), called time stamp correction. When counting more than \( n \) corrected synchronization tuples, the ordinary node will begin the synchronization estimation process. For the ordinary node multiple hops away from beacon nodes, their synchronization estimation consists of three linear regressions. The reason for one more linear regression is to eliminate the propagation delay difference between current node and different neighbors, when estimating the first \( \alpha_x \).

Figure 4 describes the synchronization data for three linear regressions with the example of node \( N_3 \). The one-way message time pairs of \( <t_3, T_4, s_{id}> \) on node \( N_3 \) are sketched in Fig. 4(a). The propagation delay from node \( N_3 \) to \( N_4 \) and \( N_8 \) are different, reflected by intercepts around the two regression lines, labelled with \( b_{3,4} \) and \( b_{3,8} \) in Fig. 4(a). Note that the intercept contains two parameters, one-way propagation delay and the offset of node \( N_3 \). These
intercepts and skews of the two regression lines are the results of first linear regression, conducted on the time pair set of different $s_{id}$ separately.

After the first linear regression, all time pairs remove the corresponding estimation intercept, shown in Fig. 4(b). These new time pairs can be further used to estimate the local skew, which is the output of second linear regression, $\hat{a}_3$.

The last linear regression is carried on the synchronization tuple set after correction. Note that the synchronization tuples are also from different low hop neighbors, indicating different one-way propagation delays. The estimation on one-way propagation delay between current node and its neighbor, can be improved through Equation (1) with the estimation of $\hat{a}_3$. For node $N_3$, two propagation delays will be calculated, $D^3_{1\to T_4}$ and $D^3_{3\to T_4}$. All time stamp pairs $<t_3,T_4,s_{id}>$ will remove their corresponding one-way propagation delay. Such time pairs of node $N_3$ are plotted in Fig. 4(c). As the link propagation delay has been eliminated, these time pairs are only concerning with the skew and offset of current node. Hence, $\hat{a}_3$ and $b_3$ will be computed with the last linear regression. After node $N_3$ finishing estimation, it informs all its neighbors. And the estimation process will be iterated until each node calculates its own skew and offset.

In summary, MulSync fully exploits broadcasting in synchronization message communication. It plays two kinds of roles for ordinary nodes to rebroadcast synchronization request, as forwarding request from high hop neighbor to the direction of beacon nodes, and as responding to the requested neighbor for building one synchronization tuple. Moreover, MulSync fuses all the time reference information for synchronization estimation. During synchronization estimation process, the ordinary node make use of not only synchronization tuples from two-way request and response, but also broadcast tuples from its low hop neighbors.

IV. SIMULATION RESULTS

The simulations are carried out on UWSN-Sim, developed by our team to simulate the underwater environment and acoustic communication. The mobility behavior of nodes is characterized by kinematic model proposed in [16]. The propagation speed of acoustic signal is 1500 m/s. Req_INT is set to 10s. The skew of the embedded timer for ordinary nodes is set randomly around 50ppm, and the clock offset is initialized randomly between (0.0, 0.08)s. Aloha [17] is exploited to deal with MAC problems. The random backoff is chosen between (0.0, 1.0)s. If the synchronization messages fail, it will not be retransmitted. In the simulations, 100 ordinary nodes are deployed randomly with 4 beacons. Unless specified, the simulation results are the average over 100 runs. We show one random network topology in Fig. 5.

The performance of MulSync are compared to MU-Sync [11]. Other UWSN synchronization protocols, such as E²DTS [13], Mobi-Sync [12], D-Sync [14] and TSMU [15], exploit additional hardwares. TSHL [10] has been proven even worse than no synchronization mechanism through simulations in [11]. For multi-hop scenarios, MU-Sync synchronizes the network hop by hop i.e. the beacons firstly synchronize the ordinary nodes within one hop range. After that, these synchronized ordinary nodes play the role of beacon nodes to synchronize other nodes of next hop and so on.

As MU-Sync chooses the number of two-way message exchange as 25, we first let the tuple count number $n$ as 25. Figure 6(a) shows the average time error on simulation time $10^{5}$s after one iteration of synchronization. It seems the synchronization errors are similar between MU-Sync and MulSync. The average synchronization error values are 0.601s and 0.659s for MU-Sync and MulSync after 10^{5}s.

We compare the mean synchronization error on every hop number in Fig. 6(b). The results shows that MulSync is of lower error for one hop ordinary nodes. This is because MulSync exploits the extra broadcast tuples for the first linear regression on $a$. However, the errors on two to four hops are higher than MU-Sync. This comes from that many MulSync’s synchronization tuples are of the same $T_1$ value. These tuples of same $T_1$ are constructed on the same broadcast, sent out by the current ordinary node.

Although MulSync does not outperforms MU-Sync on synchronization error under $n$=25, superiority of MulSync exists in synchronization message overhead and time cost, comparing in Table I. MulSync exploits broadcasting to construct $n$ synchronization tuples, while every ordinary node in MU-Sync should response 25 times for two-way message exchanging. This makes MulSync message cost 70% lower than MU-Sync. On time cost, MU-Sync conducts in hop by hop manner, while MulSync works in parallel manner through its communication scheme, saving 87% time to complete one whole iteration of synchronizing for all ordinary nodes.

We further carry on simulations for MulSync with the same total message overhead of MU-Sync. $n$ is set to 44 and
the total message number is 3707, a little lower to 3822 of MU-Sync. The average time errors on simulation time $10^5$ s for this scenario is shown in Fig. 7(a). MulSync outperforms MU-Sync 41% under this scenario, with the average time error of 0.354s. The time cost of MulSync 455s, is 9% shorter than MU-Sync. We further examine the error result on each hop in Fig. 7(b), showing that the synchronization accuracy of MulSync is better than MU-Sync on every hop. This comes from that though the total message number are of the same level, MulSync takes more messages into its synchronization calculation process.

Above comparisons between MulSync and MU-Sync bring up one question that though 25 is optimal for the number of two-way message exchanging in MU-Sync, what is the optimal number of $n$ for MulSync? We carry out experiments on the different number of $n$ ( from 20 to 44 ).

Figure 8 compares the performance on synchronization error and message cost for different $n$. The message overhead increases linearly with parameter $n$. MulSync applies aggregation in response from low hop nodes to its neighbors of high hop. And it uses a parameter of flag in its synchronization request to indicate its low hop neighbor whether the response is needed. These two techniques keeps every node carry out $n$ broadcasts, except the node of highest hop number in its neighborhood, whose broadcast number is lower than $n$. The synchronization error seems to decrease in a logarithmic manner as $n$ increasing. The choose of parameter $n$ depends on the requirement of certain sensor network applications. We can choose the intersect of two performance lines as an optimal value of $n = 28$, when no specific requirements on synchronization precision or message cost predefined.

We further compare MulSync with the optimal $n$ value to MU-Sync. The results of average time error on $10^5$ s after one iteration of synchronization, message overhead and time cost of one iteration are compared in Table II. The reason that MulSync outperforms MU-Sync in all three performance metric, 7% on average synchronization time error, 54% on message overhead, and 68% on time cost, comes from two aspects: (1) the communication scheme saves message cost and provides the calculation scheme enough time synchronization tuples. (2) The calculation scheme makes use of all the time reference information that the nodes receive via three linear regressions.

V. RELATED WORK

Because of the unique characteristics of underwater environment, several time synchronization protocols for UWSNs have been proposed in recent years.

TSHL [10] is the first mechanism designed to deal with high propagation latency. Two phases are adopted to estimate both clock skew and offset. In the first phase, the
beacon node broadcasts REF packets for many times, and common nodes perform linear regression using REF packets to estimate clock skew. In the second phase, clock offset is calibrated by two-way message exchange. TSHL assumes that the propagation delay between one common node and the beacon node is constant.

MU-Sync [11] is proposed to synchronize nodes in cluster-based UWSNs, aiming to solve the mobility of nodes. Twice linear regression is executed on the cluster head to estimate clock skew and offset of common nodes. In the first run, cluster head estimates the clock skew by assuming constant propagation delays. Then the propagation delay is calculated using the estimated skew value. The second linear regression calibrates the clock skew and offset by subtracting the propagation delay out. MU-Sync exploits the average of the round trip time to compute the one-way propagation delay, assuming that there is only little difference between two-way REF packet exchanging.

Mobi-Sync [12] utilizes spatial correlation and geometrical relationship to estimate propagation delay with beacon nodes equipped with speed sensors. It also requires that ordinary nodes can communicate to at least 3 beacon nodes in one hop. Twice linear regression is executed to estimate skew and offset.

E2DTS [13] exploits the AUV as its beacon node to synchronize ordinary nodes. The procedure also contains two phases. In the first phase, AUV broadcasts beacons to ordinary nodes. After received enough beacons, ordinary nodes execute the linear regression to estimate clock skew and the relative speed between AUV and unsynchronized node. In the second phase, E2DTS uses two-way message exchange to calculate offset.

Both D-Sync [14] and TSMU [15] exploiting Doppler shift to estimate the propagation latency between ordinary and beacon nodes. Doppler meter are required for both beacon and ordinary nodes. Linear regression is used to calculate the clock skew and offset.

VI. CONCLUSION

In this paper, we have presented a synchronization approach for multi-hop underwater sensor networks with only a few number of beacons, known as MulSync. Unlike those existing schemes synchronized in hop by hop manner, MulSync proposes its communication scheme to exploit the acoustic signal nature of broadcast, and its calculation scheme of three linear regressions to estimate local skew and offset. Moreover, none of special hardware and deployment condition is assumed by MulSync. Simulation results confirm that MulSync has achieved comprehensive performance improvement on synchronization accuracy, message overhead and time cost.

ACKNOWLEDGMENT

This research is supported by National Natural Science Foundation of China under Grant #61379128.

REFERENCES