AirSync: Time Synchronization for Large-scale IoT Networks Using Aircraft Signals

Shaopeng Zhu, Xiaolong Zheng, Liang Liu, Huadong Ma
Beijing Key Laboratory of Intelligent Telecommunication Software and Multimedia
Beijing University of Posts and Telecommunications, Beijing, China
{zhushaopeng,zhengxiaolong,liuliang,mhd}@bupt.edu.cn

***Abstract*—The prosperity of Internet of Things (IoT) brings forth the deployment of large-scale sensing systems such as smart cities. The distributed devices upload their local sensing data to the cloud and collaborate to fulfill the large-area tasks such as pollutant diffusion analysis and target tracking. To accomplish the collaboration, time synchronization is crucial. However, due to the long range and device heterogeneity, accurate time synchronization for a large-scale IoT network is challenging. Existing GPS or NTP solutions either require an outdoor environment or only have low and unstable accuracy. In this paper, we propose AirSync, a novel synchronization method that leverages the widely existed aircraft signals, ADS-B, to synchronize large-scale IoT networks with nodes even in indoor environments. But ADS-B messages have no time stamp and cannot provide a reference time. We leverage the continuity of aircraft movements to estimate the aircraft traveling time. Then devices that observe common aircraft moving segments can calculate their time offset. To obtain the time skew, we propose a combined aircraft linear regression method. We also design a transitive synchronization for devices that cannot observe common aircraft. We implement a prototype of AirSync and evaluate its performance in various real-world environments. The results show that AirSync can obtain the sub-ms accuracy.**

***Index Terms*—Time synchronization, large-scale, aircraft signal**

I. Introduction

With the expansion of network scale and the increase of node types, IoT networks can collect data from a large-scale scenario. Various applications can be realized in large-scale sensing systems such as earthquake hypocentre estimation [1], pollution monitoring [2] and mobile secure authentication [3]. Taking earthquake hypocentre localization as an example, a common method for hypocentre localization is Double Difference (DD) [4], which needs to synchronize multiple stations to calculate the arrival time difference between P-wave and S-wave for hypocentre localization. Because of lacking time synchronization method for large-scale heterogeneous devices, data from multiple sources is hard to align, hindering their cooperations to extend the coverage for large-scale earthquake warning and analysis.

To support the cooperation of large-scale IoT applications with heterogeneous sensing devices, an accurate synchronization method that can apply in heterogeneous devices is desired. We also desire the method works for both indoors and outdoors to adapt to different scenarios. Besides, the method is desired to provide continuous synchronization service with high precision. Taking hypocentre localization as an example, if the time offset of two nodes reaches to 100 *ms*, the error for hypocentre positioning will increase to 3 *km*, hindering the prediction and quick response in the early rescue [5].

978-1-7281-6630-8/20/$31.00 © 2020 IEEE

TABLE I: Comparison of common methods

	GPS	NTP [7]	FTSP [6]	**Desired**
Heterogeneity	✓	✓	✗	✓
Indoor	✗	✓	✓	✓
Synchronous range	Global	Internet reached	Dependes on network	Tens of kilometers
Accuracy	*ns*˜*ms*	1˜100 *ms*	*us*	sub-*ms*˜*ms*

However, existing synchronization methods cannot meet the requirements. We compare existing methods with the requirements in Table I. FTSP [6] is a representative of the time exchange based methods that broadcast time reference by active communication in homogeneous networks. The methods dedicated to homogeneous networks usually cannot cover a large area because the multihop broadcasting brings accumulative synchronization errors. Network Time Protocol (NTP) [7] is a widely used method to synchronize heterogeneous devices with Internet connectivity. It leverages network delay between the devices and the server to calculate the local time offset. But NTP requires stable network connection when synchronizing. When the network jitters, network delay estimating becomes difficult, causing errors during synchronization. Global Positioning System (GPS) time service leverages satellites to obtain accurate UTC time, which is accurate and valid globally for heterogeneous devices. But unfortunately, GPS signal cannot reach indoors, which greatly limits its application scenarios. From the comparison in Table I, we can find none of the existing methods can fulfil all the desired requirements.

In this paper, instead of using the weak GPS signal from satellites, we take notice of the aircraft signal, which can be a time source for large-scale time synchronization. The so-called Automatic Dependent Surveillance-Broadcast (ADS-B) signal is a broadcast message with the aircraft flight state. The carrier frequency of the ADS-B signal is 1090MHz. Although it is only a little bit lower than GPS, the ADS-B signal can easily reach indoors because the altitude of aircraft is much lower than the satellite. Besides, the ADS-B signal can be received

dozens and even hundreds of kilometers away, which can satisfy the basic large-scale time synchronization requirements. Besides, according to the open data from OpenSky network [8], ADS-B signal can cover a wide area and can be received all day long to support our continuous synchronization.

Although the ADS-B signal has these promising characteristics, there are challenges when using it for synchronization: *(1) ADS-B signal does not have any time information.* We try to take ADS-B packets as an event trigger, but received packets are not strictly periodic, directly synchronize will produce accumulative error caused by time drift. *(2) Different aircraft contribute uneven quality information for synchronization results*. For each aircraft, the signal noise is different. How to utilize all aircraft information wisely is challenging. *(3) how to synchronize the nodes cannot receive the same packet is a challenge.* Although ADS-B signal can cover large-scale scenario, signal will be lost caused by block or other reasons, so nodes in long-range are hard to get the same packet event. How to synchronize the nodes without common received packets should be considered.

To solve these challenges, we propose AirSync, a novel time synchronization method for large-scale heterogeneous IoT networks. Fig.1 shows the framework of AirSync. AirSync leverages the widely existing ADS-B signal as a common event. Although we can not obtain time information from packets, we leverage the common traveling distance of aircraft observed by different nodes to calculate the interval of matched packets. By this way, we can get the time information without explicit timestamps. Integrated ADS-B antennas, nodes can record ADS-B messages with a local timestamp for each receiving ADS-B message, then upload the messages to the cloud periodically. To reduce the transmission overhead, AirSync designs a preprocessing at the end nodes to filter retransmission packets and unnecessary information. Then the cloud server will align the ADS-B messages to get the time offsets of nodes. After obtaining multiple time offsets, AirSync can establish a linear model to synchronize two nodes. To avoid the inaccuracy caused by uneven noise from different aircraft, we propose a CAR algorithm that smartly combines the information from multiple aircraft. We also propose a transitive synchronization method that uses relay nodes to synchronize the nodes that cannot receive the same packets.

We implement the prototype of AirSync on laptops and Raspberry Pi3 with commercial RTL2832U receiver and ADS-B antenna. We deploy 5 nodes in the real-world environments to evaluate AirSync. We run the experiments for 60 hours and collect about two million ADS-B messages. The experimental results show that the median of time offset error is 0.426 *ms* and 0.882 *ms* for short- and long-range nodes, respectively. The median of time skew error of AirSync is 4.56 ppm, which is much smaller than the 30-50 ppm clock drift for IoT sensor devices [9].

The contribution of this paper is summarized as follows.

- We propose a novel IoT time synchronization method which leverages ADS-B signal to achieve a sub-ms accuracy for large-scale IoT networks.
- We address the challenges of using ADS-B for synchronization. AirSync uses the common travelling time of aircraft to eliminate the dependency on explicit time stamp as reference. We design a CAR synchronization algorithm to cope with the noises from different aircraft and obtain more accurate estimation on the time skew. AirSync also integrates the transitive method to synchronize remote nodes.
- We implement AirSync and evaluate its performance by real-world experiments. The results show that AirSync can achieve sub-ms synchronization precision for the heterogeneous IoT devices 22.9 *km* apart.

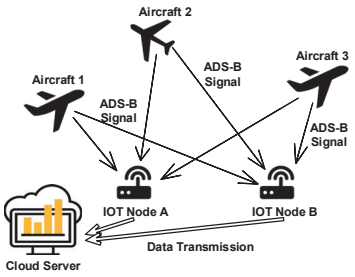


Fig. 1: AirSync uses the commonly observed ADS-B signal as the synchronization event for long-range IoT nodes.

The rest of this paper is organized as follows. We discuss the related work in Section II and introduce design details of AirSync in Section III. We present the evaluation in Section IV and conclude our paper in Section V.

II. Related work

Synchronization methods can be divided into two categories: external time source synchronization and internal message exchange synchronization. External time synchronization means the reference time is provided by the external accurate time source. Message exchange time synchronization means all the nodes in network are synchronized by message exchange, eventually all the nodes have the same clock.

External time synchronization source mainly includes Global Positioning System (GPS) time server and timekeeping radio station synchronization (e.g., WWVB [10] in the US and JJY [11] in Japan). GPS time synchronization lets nodes get UTC from GPS time servers, which can synchronize global-range heterogeneous devices. However, the signal of GPS and WWVB have poor penetration through walls [12].

For message exchange synchronization, NTP is a common method for nodes without resource constraint. A client estimates network delay to compensate error after getting time from a time server. However, the estimation error of NTP will increase due to network jitter, even reaching hundreds of milliseconds. For the low-power IoT nodes with limited resources [13], there are many RF communication synchronization methods like RBS [14], FTSP [6], Glossy [15], CESP [16], R-Sync [17]. RBS [14] synchronizes by sending a message to the target devices, then devices add

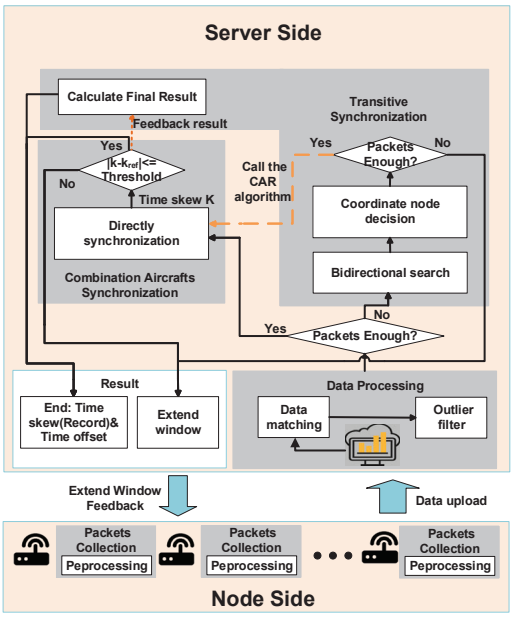


Fig. 2: Overview of AirSync.

timestamp when they receive the message, the node calculates time offset by exchanging timestamp. Compared with one-hop synchronization of RBS, FTSP [6] can achieve large-scale time synchronization of multi-hop network. All these methods need devices to communicate with each other, but they cannot realize in heterogeneous devices with incompatible radios. Recently, some event-driven synchronization methods were proposed for IoT device like WizSync [12] and PSync [18]. WizSync leverages Wi-Fi beacon as a trigger to synchronize two ZigBee nodes. PSync leverages visible light in building to synchronize multiple ZigBee nodes. However, these methods have a limit synchronization range which cannot satisfy the large-scale synchronization requirement.

Besides, there are some related works based on ADS-B signal: Calvopalomino.et.al realize Nanosecond-precision time-of-arrival estimation by SDR receiver [19]. Manuel Eichelberger.et.al use ADS-B signal to realize indoor localization [20]. Localization also requires time synchronization, compared with their work, our design is for a completely different scenario. In our setting, we have no UTC for reference, so it's impossible to build a linear model directly. Besides, nodes can not exchange messages directly and we do not have node location information to calculate the propagation time delay.

III. DESIGN

A. Overview

Figure 2 shows the overview of AirSync, which contains 4 modules: packets collection, data processing, combination aircraft synchronization, and transitive synchronization. Packets collection module aims to receive ADS-B packets and preprocesses them, the module receives messages by picking out the interesting location and speed packets to reduce the transmission overhead. Then packets are sent to a cloud server and Airsync enters the data processing module. Uploading data is time window based and the initial window length is 5 minutes in our system. Window length will be extended if received packets are not enough to synchronize. The server will match data, calculate system time difference by local timestamp and remove outlier. If the number of matched packets from the same aircraft is greater than 3, we judge the number of packets can support direct synchronization. AirSync will enter combination aircraft synchronization module. Otherwise, our system will go into the transitive synchronization module. In transitive synchronization module, we will search for all possible synchronized paths by bidirectional search and use the score-based method to pick the relay node to optimize accuracy. If the number of packets is not enough for transitive synchronization, the server will give feedback to the device to extend the next uploading time window. If the synchronized time skew is an outlier compared with the historical record, AirSync will also extend the uploading window for collecting more ADS-B messages.

B. Data Preprocessing

Figure 3 shows the structure of ADS-B packet. The first key message is 24-bit ICAO, which can uniquely identify one aircraft. We use ICAO to classify the information from different aircraft. ADS-B adds flight state into a 56-bit data field. In the data field, the first 5 bits are used to indicate the type of data. The type includes speed, location, altitude and so on [21]. That means we cannot receive all the interesting information from a single packet. To verify the reception of ADS-B signal, we use homogenous and heterogeneous devices to receive signal at different distances. Figure 4 shows the number of matched packets and the match rate during 5 minutes. As shown in Figure 4, in the worst case, heterogeneous devices in long distance still have 75 matched packets which can provide enough information for our method.

Raw ADS-B data is highly redundant for our method, including the caution packets and retransmission packets. Besides, local timestamps may also cause jitter because receiver just receives retransmission packets or system schedule delay. Therefore, the preprocessing of the original data is necessary to improve the synchronization accuracy of AirSync. There are two parts in AirSync data preprocessing:

1) Local Data Preprocessing: In the IoT node side, we first filter out all signal except position and speed according to type code in the data field. Then if two packets from one aircraft have the same location, the later packet should be a retransmission packet and we filter out it. Signal preprocessing at the nodes not only improves signal quality, but also reduces the transmission overhead to the server.

2) Cloud Server Data Preprocessing: After local data preprocessing, collected data are sent to the server. The server will first match packets from the same aircraft with the same location, then delete the aircraft information matches only one packet which cannot provide effective information. Then it calculates the system time difference according to

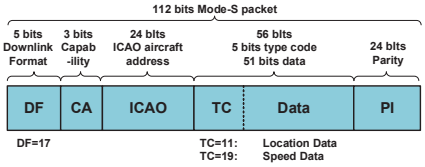


Fig. 3: Structure of the Mode-S packet.

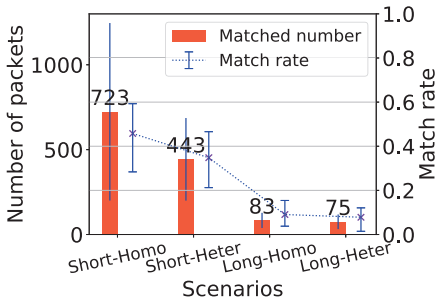


Fig. 4: Measurement of the matched packets in different scenarios.

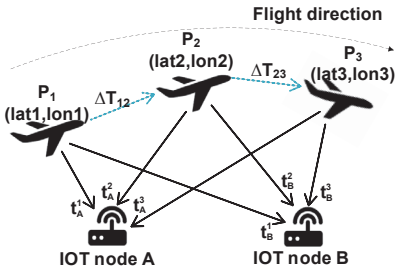


Fig. 5: Two nodes are synchronized by one aircraft.

each matched packet local timestamp. Finally, we use StatisticalOutlierRemoval filter [22] to remove outliers.

C. ADS-B Linear Regression Model

In AirSync, we build a linear model in order to describe the time difference between two nodes. The basic regression equations can be written as:

$$t_A^n = k_a(T_n - T_1) + b_a + T_{da} + e_a \tag{1}$$

$$t_B^n = k_b(T_n - T_1) + b_b + T_{db} + e_b \tag{2}$$

where t_A^n and t_B^n are receiving time of the n-th matched message at nodes A, B respectively. k_a, b_a, k_b, b_b are time skew and time offset relative to UTC time of node A and node B. T_n is the transmit time of n-th match message. T_{da}, T_{db} are the propagation time of signal from aircraft to nodes A and B. e_a, e_b are noise. The receiving time difference between A and B can be expressed as:

$$t_A^n - t_B^n = k(T_n - T_1) + b + T_d + e \tag{3}$$

where k, b is the time skew and time offset between two nodes. T_d represents the signal propagation time difference between nodes A and B. e is noise error in data. If we can obtain T_n directly from packets, we can estimate k, b easily by linear regression. However, ADS-B signal does not have time information about when the signal is sent. Therefore, it is impossible to capture regression results directly from Equation (3). To solve this problem, we leverage the common aircraft flight time between two packets. As shown in Figure 5, we can obtain relative flight time by the position and speed decoded from the packet, i.e.

$$T_n - T_1 \doteq \frac{\|P_n - P_1\|_2}{\bar{V}_{1,n}} \tag{4}$$

Where p_n is the position where the n-th matched message transmits. $\bar{V}_{1,n}$ is the average flight speed between two receiving packets. The equation has the limitation that the aircraft route should be a straight line. Besides, using the average speed will reduce the accuracy of flight time. So we divide the whole flight time into multiple segments. Each segment is the time between two matched position packets. Then we accumulate these segments as flight time. In this way, the flight path of the aircraft can be considered as a straight line in a short time. Even if the aircraft is hovering in the air, the accurate flight distance can be obtained. So Equation (4) can tranform into Equation (5):

$$T_n - T_1 = \sum_{i=1}^{n-1} \frac{\|P_{i+1} - P_i\|_2}{V_{i,i+1}} \tag{5}$$

Besides, the error caused by propagation time difference T_d will have some influence on our synchronization. If propagation distance difference is 30 km, the propagation time difference will be 0.1 ms. Because we do not know the location of nodes in most real scenes, we regard this error as noise and try to reduce this effect in the later design. So the final regression model is Equation (6).

$$t_A^n - t_B^n = k(\sum_{i=1}^{n-1} \frac{\|P_{i+1} - P_i\|_2}{V_{i,i+1}}) + b + e \tag{6}$$

Because speed changes less in a short time, $V_{i,i+1}$ is set to the speed decoded by the speed packet closest to the $i+1$-th matched packet. To investigate the speed and location packet reception, we show the packet reception result from 7 aircraft in 2 minutes in Figure 6. It shows that it is uncertain whether it will receive the speed message after location message because receiving ADS-B signal is uncontrollable. To overcome this problem, we leverage greedy algorithm to design a discarding mechanism, it searches for speed information from the same aircraft before and after the target position information, gradually expanding search until the first speed message is found or reaches the time threshold (10s). Finally, if we can not find speed information, this matched position will be discarded. By this way, each position message can find a valid speed.

D. Combination Aircraft Synchronization

For a large-scale indoor scenario, the number of matched packets from one aircraft may not be enough to obtain high accuracy. Hence, we try to leverage packets from multiple aircraft. However, because of the location difference of aircraft, information from different aircraft may have inconsistent regression results. If we just choose one of them as a final result, the result will become unstable and may have a large

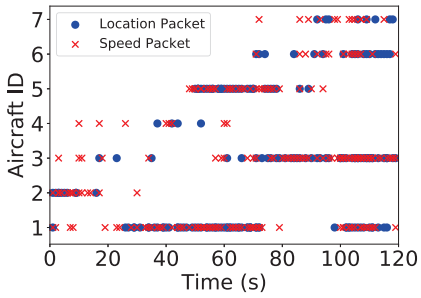


Fig. 6: The reception of location and speed packets in two minutes.

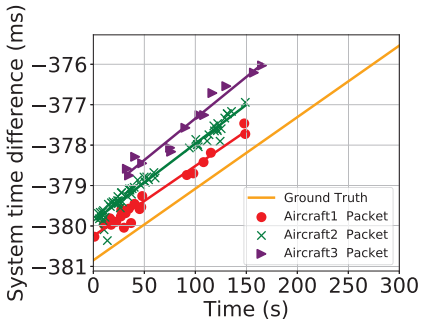


Fig. 7: The results without CAR algorithm.

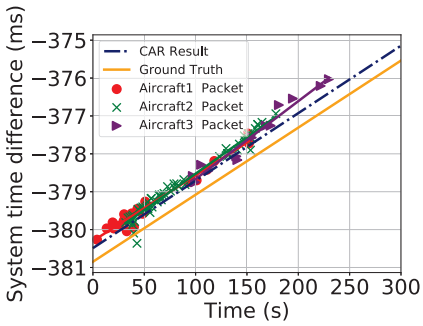


Fig. 8: The results of CAR algorithm.

synchronization error. Figure 7 shows the original regression results of three aircraft directly synchronized.

All of these factors drive us to combine multiple aircraft information. That means we regard multiple aircraft as one by combining these matched packets of the aircraft. There are some advantages of combining multiple aircraft: (1) When the number of matched packets from one aircraft gets smaller, combining information from multiple aircraft can take full use of matched packets. (2) As Figure 6 shows, one aircraft is hard to cover entire time horizon, combining aircraft can extend the coverage of information on the time horizon which can promote the accuracy of time skew estimation. To utilize packets from multiple aircraft, we need to shift packets relative time of aircraft into the same time axis. We use the packet receiving timestamp as time information between aircraft. Using packets timestamp will cause cumulative errors for time skew in theory. In the evaluation session, we prove that this cumulative error will not have a big impact on time skew estimation.

We design an algorithm for combining multiple aircraft called Combine Aircraft Regression (CAR) algorithm. There are two inputs for this algorithm, one is a set $PI = \{pi_{f1}, pi_{f2}, ...\}$. The set includes packets information pi_{fi} consisting of matched packets location P_i, valid speeds V_i and both local timestamps TS_i^1, TS_i^2, which is grouped by aircraft (ICAO) fi. The other one is the interval from the system start to first matched message for each flight $t_s = \{t_{f1}, t_{f2}, ...\}$, we call this duration time the first matched time. The output is the time skew and time offset.

In the CAR algorithm, we first pre-calculate each aircraft subset to find the time relation inside matched packets of single aircraft. then we can get the result set: $PreSynResult = \{PreSync(sd_{f1}), PreSync(sd_{f2}), ...)\}$, where $PreSync()$ function is the calculation of packets relative time and system time difference. Each aircraft packets set pre-calculates result $PreSync(sd_{fi})$ including the packet relative time T_{pj}^i and the system time difference calculated by timestamp Sys_{diffj}^i for each packet. Then we build a summary set $ComRes$ including every aircraft information. Each matched packet relative time T_{pj}^i from each aircraft set pi_{fi} will add corresponding first matched time t_{fi}. By this way, all the packets will be arranged into the same time axis.

Algorithm 1: CAR Algorithm

Input :
Packets information grouped by aircraft:
$PI = \{pi_{f1}, pi_{f2}, ...\}$ $(|PI| = N)$;
$pi_{f1} = \{\{P_1, V_1, TS_1^1, TS_1^2\}, ...\}$;
First matched time:
$t_s = \{t_{f1}, t_{f2}, ...\}$;

Output:
Combined regression results:
CAR_k, CAR_b;

$PreSynResult = \{PreSync(pi_{f1}), PreSync(pi_{f2}), ...)\}$
$PreSync(pi_{f1}) = \{(T_{p1}^1, Sys_{diff1}^1), (T_{p2}^1, Sys_{diff2}^1)...\}$
if *PreSynResult!=∅* **then**
  for *i in N* **do**
    for *j in* $|PreSync(pi_{fi})|$ **do**
      ComRes add $(T_{pj}^i + t_{fi}, Sys_{diffj}^i)$
    end
  end
end
else
  return *Exit, Cannot synchronized directly*
end
$CAR_k = Sync(ComRes)_{-k}$
$CAR_b = Sync(sd_{f\theta} \quad |\theta = \arg\max_\theta |\mathrm{sd}_{f\theta}|)_{-b}$

Then the algorithm synchronizes $ComRes$, where $Sync()$ function is the Equation (6) synchronization model. After synchronization, AirSync uses the regression result k as CAR algorithm time skew k (Equation (7)). Because we think that combining aircraft set can extend information time horizon and promote the accuracy of time skew.

$$CAR_K = Sync(ComRes)_{-k} \tag{7}$$

For time offset estimation, the propagation time difference is the main error source, so we should reduce the error as much as possible. Empirically, if the matched number is large, the aircraft should locate at a more balance place between two nodes and both nodes can receive enough ADS-B messages

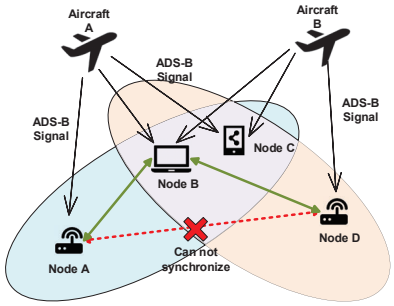


Fig. 9: Overveiw of transitive synchronization.

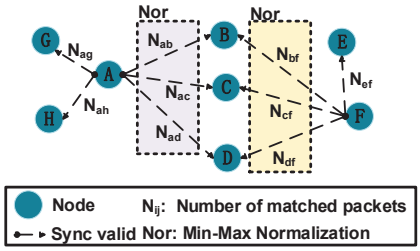


Fig. 10: Bi-directional synchronization path search algorithm.

to match. Therefore, there is less propagation time difference for the aircraft which has more matched packets. So in CAR algorithm, we synchronize the aircraft which has the most number of matched packets, then we use the synchronization time offset as B of CAR algorithm as Equation (8). Figure 8 shows the CAR algorithm synchronization result of 3 aircraft in Figure 7. We can find that CAR algorithm synchronization can get a better result than the synchronization result of any one of the three.

$$CAR_B = Sync(sd_{f\theta} \quad |\theta = \arg\max_{\theta} |\mathrm{sd}_{f\theta}|)_{-b} \tag{8}$$

E. Transitive Synchronization

In the real scenario, nodes may have few matched packets to synchronize because of node's biased position or signal missing. In this situation, we try to leverage the transitivity of synchronization to synchronize target nodes. As Figure 9 shows, if node A and node D tend to synchronize but do not match enough packets, nodes B and C both have some matched packets with them, B and C will become the relay nodes. The server will choose one of them to synchronize target nodes indirectly. The server uses a heuristic optimization algorithm to construct a criterion to choose the best relay node. The process of transitive synchronization shows in Figure 10, if two nodes A, F have less matched messages to synchronize, we use bidirectional search algorithm [23] to find all feasible nodes as relay nodes. Bidirectional search can reduce the searching space and find the alternative synchronization path rapidly. Then we can find three paths: $A-B-F, A-C-F, A-D-F$. The server will traverse all alternative nodes between them and calculate the number of matched packets, normalize the number of matched packets on each side, add normalized

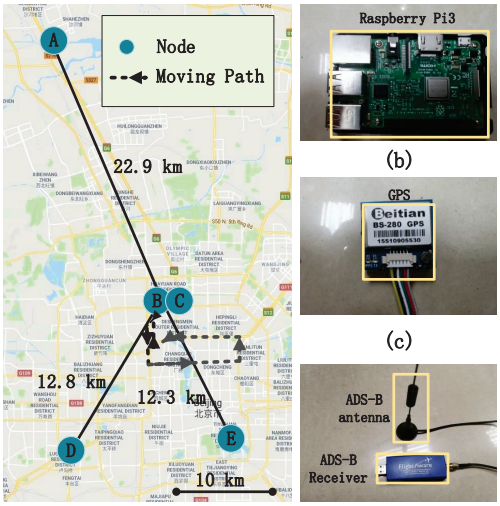


Fig. 11: Experiment settings: (a) Deployed nodes of AirSync. (b) Raspberry Pi3. (c) GPS module. (d) ADS-B antenna and receiver.

results on each path to calculate scores for all relay nodes. For instance, the score of node B is:

$$Score_B = N(N_{ab}) + N(N_{bf}) \tag{9}$$

where function $N()$ is Min-Max normalization. N_{ab}, N_{bf} are the numbers of matched packets for nodes A, B and nodes B, F. By calculating and comparing final scores, server can choose the node with the highest score as the relay node. Besides, it's hard to ask for time alignment for two sides of the path. If the start time difference of two synchronization sides is less than $\frac{WindowLength}{2}$ (150 s), these two sides are considered to be capable to transitively synchronize. If the start time difference is too large, an accumulated error will affect our accuracy.

IV. Evalution

A. Experiment setting

In order to evaluate the performance of AirSync, we conduct experiments in the real world. We deploy 5 IoT nodes where the location of nodes shows in figure 11(a) and nodes detail shows in Table II. nodes A, B and C are deployed in campus and node D and E deploy in residential zones. We modify open source project Dump1090 [24] for nodes to obtain accurate local timestamps by reducing the data length in each batch. We use the commercial RTL2832u receiver and ADS-B antenna (shown in Figure 11(d)) to receive ADS-B signal, the total costs of devices are less than 60 dollars which supports possibilities of AirSync extensive deployment on IoT nodes. We realize the prototype on laptop and Raspberry Pi3 (Figure 11(b)). To evaluate AirSync in a long range, we synchronize node A and node C in a distance of 22.9 *km*. Node C is Raspberry Pi3 as a heterogeneous node. Nodes B and C are used to evaluate AirSync in a short range scenario. Nodes D and E are set as chance nodes which have unbalanced data size

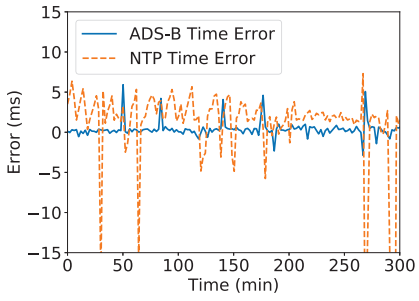
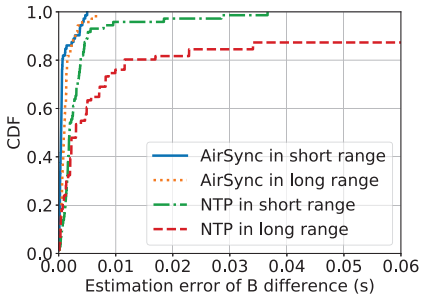
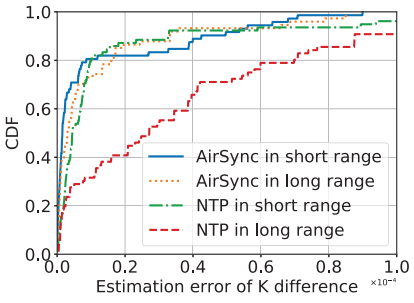


Fig. 12: Synchronization error of AirSync and NTP during 5 hours.



(a) Time Offset



(b) Time Skew

Fig. 13: CDF of the error of time offset and time skew of AirSync and NTP in the short and long range.

TABLE II: Summary of experimental nodes

Node	Device	CPU Frequency	Total time	Total number of packets
A	Dell Latitude5290	1.6GHz	43.2h	493800
B	Dell-G3	2.2GHz	60.1h	626796
C	Raspberry Pi3	1200MHz	60.1h	726128
D	ThinkPad T430	2.6GHz	16.4h	8433
E	Lenovo F41	1.8GHz	21.1h	10050

comparing with the other three nodes to evaluate transitivity design. To evaluate AirSync in a mobile scenario, we moved node B about 17 *km* on the ring road of Beijing according to the moving path in Figure 11(a) and synchronize with node A. We choose GPS time with PPS signal (shown in Figure 11(c)) as our ground truth which accuracy reaches microsecond level. Because GPS can not reach indoors, so we leverage extension cord to stick GPS module out of the window.

B. Performance of AirSync

To evaluate the performance of AirSync, we first evaluate AirSync time error in a short range by using node B and node C. Both of the two nodes connect to the same Wi-Fi node which setting expected to have less network jitter and get a better NTP synchronization result. Figure 12 shows the time error in 300 minutes for AirSync and NTP. AirSync average time error is about 0.513 *ms* and NTP is 3.672 *ms*, NTP has a larger average time error and has more jitter. So AirSync result is more accurate and stable.

Figure 13(a) shows cumulative distribution function (CDF) of time offset estimation error. In the experiment, we set node B and node C as short-range devices, node A and node C as long-range devices and synchronize about 70 times. The result shows that the worst error for AirSync is less than 8 *ms*, the error median of short range is 0.426 *ms* and 0.882 *ms* for long range, both of them reach sub-millisecond level. AirSync in long range will bring some extra errors for time offset estimation. The main reasons include 1) Propagation time differences bring some effects on the time offset estimation. 2) The reduction of the number of matched packets affects the accuracy of synchronization.

NTP in a short range and long range both have small errors when connecting to the Wi-Fi network. But when connecting to the mobile network, time error will increase obviously because of the network jitter. To evaluate the NTP in the 4G network, we connect node B to the 4G network and synchronize with node A. As Figure 14 shows, NTP synchronization average time error increases to 20 *ms* in the 4G network.

Figure 13(b) shows the estimation error of time skew. The median error in long range is 4.456×10^{-6} (4.456 ppm). This error is less than NTP time skew in short range. Comparing with 30-50 ppm time skew for IoT devices, this error is significantly reduced.

On the other hand, AirSync not only has the advantage of accuracy comparing with NTP. NTP can only synchronize when the network is available, but AirSync can also capture data without network and align these data when the network is available. The recorded ADS-B messages can also transmitted by LEO satellite network [25], but it costs too much.

C. Performance of CAR algorithm

To evaluate the effectiveness of CAR algorithm, we design experiments to evaluate time offset and time skew respectively. We synchronize two nodes 70 times, compare the 3 regression methods in Figure 15. Where overall regression is to regress all matched points. Weighted average refers to assigning different weights to different aircraft based on the number of matched packets. The single best means the regression result of aircraft which has the most matched packets will become AirSync's final result. Figure 15(a) shows that the most single method has the best result comparing with the other two regression methods. So using the aircraft with the most matched packets can reduce the propagation time difference effect. Figure 15(b) shows the time skew estimation errors for three methods. It shows that the overall regression has the best result for time skew estimation. So for time skew estimation, we need to expand the time range of packets, in case the matched packets are concentrated in a short period of time and reduce time skew accuracy.

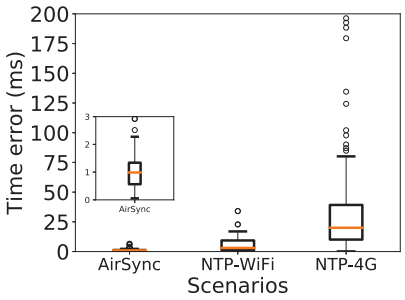
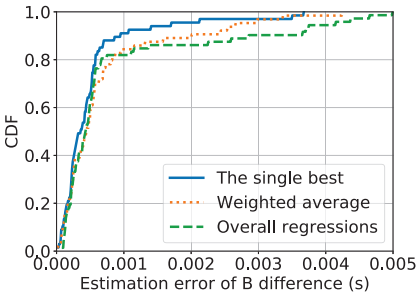
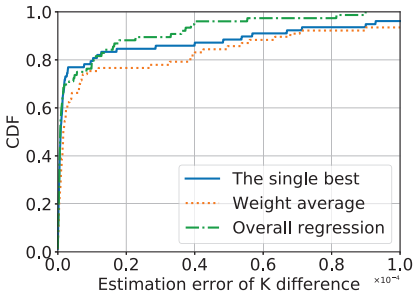


Fig. 14: Performance of AirSync with NTP in different network connections.



(a) Time Offset



(b) Time Skew

Fig. 15: CDF of the estimation error of different regression methods.

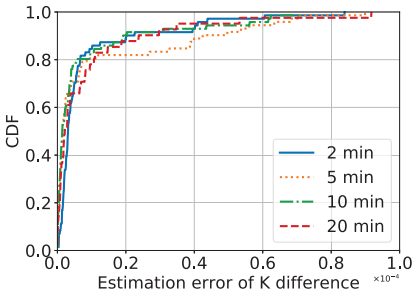


Fig. 16: CDF of time skew estimation error when using different synchronization window lengths.

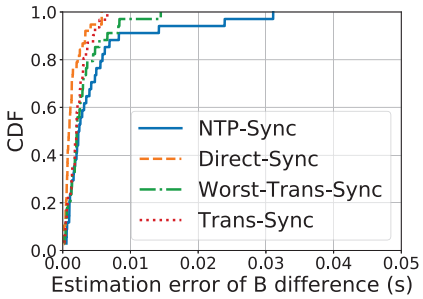


Fig. 17: CDF of the time offset error for transitive synchronization.

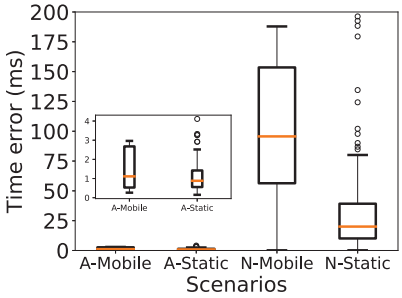


Fig. 18: The results in mobile scenarios. A is AirSync, N is NTP.

Besides, when designing CAR algorithm, we can only get the time relationship between packets from the same aircraft, so when taking aircraft into the same time axis, we can only use the local timestamp, it will affect time skew accuracy. In theory, it has more effects when extending signal reception time. In Figure 16, we set AirSync synchronization signal at different time scales to see the time skew errors. The result shows time skew regression results in different time scales have a similar estimation error. Therefore shifting aircraft to uniform time axis will not affect time skew estimation a lot.

D. Performance of Transitive Synchronization

To evaluate transitive synchronization, we suppose node A and node C in Figure 11(a) cannot directly synchronize, then choose node B as the main relay node. Node D and node E have unbalanced data size, and bring some intermittent synchronization chances. The server needs to pick the best relay node according to the node score when multiple paths appear. In our experiment, we compare the time offset synchronization error of direct synchronization, transitive synchronization, worst transitive synchronization and NTP synchronization. The worst transitive synchronization is that when there multiple relay nodes, we choose the node which has the lowest node score. Figure 17 shows the experiment result, transitive synchronization result median is 2.057 *ms*, it's still better than NTP method, whose synchronization median is 2.368 *ms*. Comparing with the worst transitive result, transitive synchronization can promote accuracy. Transitive synchronization has more errors comparing with direct synchronization, there are two main error sources: 1) Accumulated errors for each side synchronization of a transitive path. 2) Because the two sides synchronization is not strict alignment, the difference in synchronization start time will cause a time offset error. Because of these errors in transitive synchronization, AirSync will not attempt to transitively synchronize if the nodes can directly synchronize.

E. Mobile scenario

In the real world, nodes may work in different scenarios, we try to simulate these scenarios to evaluate AirSync performance. Firstly, some nodes are mobile when working like the nodes on the vehicle or on the buoy. So we evaluate AirSync synchronization in a mobile scenario. In the experiment, node B in Figure 11(a) connects to the 4G network and node A connects to the Wi-Fi network, then node B moves along the moving path in Figure 11(a) and synchronizes with node A. The result shows in Figure 18. The time offset error median of AirSync in mobile scenario is 1.113 *ms*, close to the static scenario error median 0.882 *ms*. So AirSync has a stable synchronization accuracy in a mobile scenario. But the mobile scenario has an obivous effect on NTP synchronization in a 4G network because of the network jitter caused by base stations handover. The synchronization error increases from 20 *ms* to 95.6 *ms*. Therefore, AirSync has better performance than NTP in a mobile scenario.

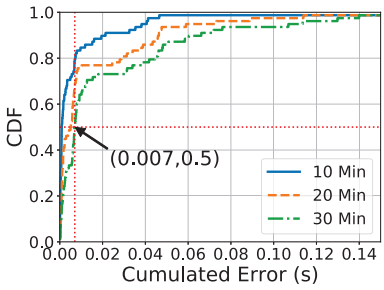


Fig. 19: Cumulated error without synchronization chance.

F. Low-power scenario

For some low-power consumption devices, periodic dormancy is the basic characteristic [26]. For these devices, we evaluate cumulated errors of time after an interval without synchronization, we try to simulate periodical node synchronization in this way. we set the duration as 10, 20 and 30 minutes and find the time errors after these durations. Figure 19 shows the results of 80 synchronizations. The cumulated error will increase with the duration extending, there are 84.6%, 75.6%, and 64.1% of errors less than 10 *ms* in 10, 20, 30 minutes, the cumulated error median in 30 minutes is 7 *ms*, which is acceptable in most data fusion applications.

V. Conclusion

In this paper, we realize time synchronization for large-scale, heterogeneous devices inspired by ADS-B signal. we use data in ADS-B signal to build a linear regression model. Then we design a CAR algorithm to fully utilize packets from different aircraft to promote regression accuracy. For the nodes that cannot directly synchronize, we design transitive synchronization to promote the robustness of AirSync. Finally, AirSync realizes sub-millisecond level synchronization accuracy for long-range, heterogeneous nodes. In evaluation, we set 5 nodes in the real world and receive different time ADS-B signals to test AirSync. We compare AirSync with NTP and GPS in different scenarios. Besides, we evaluate the performance of our CAR algorithm and the transitive module. For special scenario evaluation, we test AirSync in mobile and low-power consumption scenarios, we find the node move will not affect AirSync a lot, and the low-power situation will bring acceptable cumulated errors.

Acknowledgment

This work is supported in part by the Funds for Creative Research Groups of Chinaunder No. 61921003, the Natural Science Foundation of China (NSFC) under No. 61720106007, No. 61722201, and No. 61632008, and the 111 Project (B18008).

References

[1] J. Spetzler and B. Dost, "Hypocentre estimation of induced earthquakes in groningen," *Geophysical Journal International*, vol. 209, no. 1, pp. 453–465, 2017.

[2] L.-J. Chen, Y.-H. Ho, H.-H. Hsieh, S.-T. Huang, H.-C. Lee, and S. Mahajan, "Adf: An anomaly detection framework for large-scale pm2. 5 sensing systems," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 559–570, 2017.

[3] P. Xie, J. Feng, Z. Cao, and J. Wang, "Genewave: Fast authentication and key agreement on commodity mobile devices," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1688–1700, 2017.

[4] F. Waldhauser and W. L. Ellsworth, "A double-difference earthquake location algorithm: Method and application to the northern hayward fault, california," *Bulletin of the Seismological Society of America*, vol. 90, no. 6, pp. 1353–1368, 2000.

[5] M. Lukac, P. Davis, R. Clayton, and D. Estrin, "Recovering temporal integrity with data driven time synchronization," in *Proceedings of IEEE IPSN*, 2009.

[6] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of ACM Sensys*, 2004.

[7] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[8] "Opensky network," https://opensky-network.org/, accessed January 2, 2020.

[9] Z. Li, W. Chen, C. Li, M. Li, X.-Y. Li, and Y. Liu, "Flight: Clock calibration using fluorescent lighting," in *Proceedings of ACM MobiCom*, 2012.

[10] "WWVB," https://www.nist.gov/pml/time-and-frequency-division/radio-stations/wwvb, accessed January, 2020.

[11] "JJY," http://jjy.nict.go.jp/jjy/index-e.html, accessed May, 2019.

[12] T. Hao, R. Zhou, G. Xing, M. W. Mutka, and J. Chen, "Wizsync: Exploiting wi-fi infrastructure for clock synchronization in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 6, pp. 1379–1392, 2014.

[13] X. Zheng, Z. Cao, J. Wang, Y. He, and Y. Liu, "Interference resilient duty cycling for wireless sensor networks under co-existing environments," *IEEE Transactions on Communications*, vol. 65, no. 7, pp. 2971–2984, July 2017.

[14] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *USENIX operating systems design and implementation*, vol. 36, pp. 147–163, 2002.

[15] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Proceedings of ACM/IEEE IPSN*, 2011.

[16] F. Gong and M. L. Sichitiu, "Cesp: A low-power high-accuracy time synchronization protocol," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 4, pp. 2387–2396, 2015.

[17] T. Qiu, Y. Zhang, D. Qiao, X. Zhang, M. L. Wymore, and A. K. Sangaiah, "A robust time synchronization scheme for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3570–3580, 2017.

[18] X. Guo, M. Mohammad, S. Saha, M. C. Chan, S. Gilbert, and D. Leong, "Psync: Visible light-based time synchronization for internet of things (iot)," in *Proceedings of IEEE INFOCOM*, 2016.

[19] R. Calvo-Palomino, F. Ricciato, B. Repas, D. Giustiniano, and V. Lenders, "Nanosecond-precision time-of-arrival estimation for aircraft signals with low-cost sdr receivers," in *Proceedings of ACM/IEEE IPSN*, 2018.

[20] M. Eichelberger, K. Luchsinger, S. Tanner, and R. Wattenhofer, "Indoor localization with aircraft signals," in *Proceedings of ACM SenSys*, 2017.

[21] S. Junzi, H. Jacco, and E. Joost, "ADS-B decoding guide." https://adsb-decode-guide.readthedocs.io/en/latest/content/introduction.html.

[22] "Statisticaloutlierremoval filter," http://pointclouds.org/documentation/tutorials/statistical_outlier.php, accessed May, 2019.

[23] G. Nannicini, D. Delling, L. Liberti, and D. Schultes, "Bidirectional a search for time-dependent fast paths," in *Proceedings of Springer International Workshop on Experimental and Efficient Algorithms*, 2008.

[24] "Dump1090 project," https://github.com/mutability/dump1090.

[25] D. Xia, X. Zheng, P. Duan, C. Wang, L. Liu, and H. Ma, "Ground-station based software-defined leo satellite networks," in *Proceedings of ICPADS*, 2019.

[26] X. Zheng, Z. Cao, J. Wang, Y. He, and Y. Liu, "Zisense: Towards interference resilient duty cycling in wireless sensor networks," in *Proceedings of ACM, SenSys*, 2014.