Analysis of time synchronization of OpenSky sensor nodes

***Abstract*—This project was mainly done with the purpose of finding the synchronization error in OpenSky network. For this, we checked the messages sent by the aircraft to two receivers and checked the offset in their reception time. To check the correctness of our system we compare the calculated measurements to the ground truth value, ie; the difference in the timestamps between two receivers should be equal to the difference in the message propagation delay between the two receivers to the aircraft.**

I. Introduction

Time synchronization is an important aspect in many of the mobile security systems. Time synchronization can be internal or external time synchronization[1]. External synchronization means that all nodes in the network are synchronized with an external source of time (e.g., UTC). Internal synchronization means that all nodes in the network are synchronized with one another, but the time is not necessarily accurate with respect to UTC. Synchronization can be defined as the process of correcting the clock value of a system to the clock value of the comparing system. For example, in external synchronization, the system clock value is corrected by comparing to the UTC global time and in internal synchronization, the system clock is corrected by comparing to the clock value of a system inside the network, a master node. In synchronized network, the clock values of the systems in the network will be same. Synchronizing the nodes at the right time is a difficult job. Huge offset in the synchronization between the nodes can be a threat to different types of attacks on the system. So it is always important to check the correctness of the synchronization in the network.

OpenSky Network is a community-based receiver network and synchronization should be done with the data received from the sensors. i.e not every sensor in the are not GPS synchronized. Even though there exist several existing methods to synchronize the networks, none of them provides the solution for the problem. OpenSky requires a time synchronization within an error standard deviation of 100 nanoseconds. Apart from those, ADS-B protocol is vulnerable against identifying false positioning reports from aircraft. Researchers have already found this vulnerability in ADS-B based air traffic control system [2]. To tackle this issue, OpenSky uses Multilateration navigation technique to verify the correctness of location claims [3]. For multilateration, three or more synchronized sensor nodes are required.

In this project, we perform an analysis on the synchronization of OpenSky network. Next in Section(II) we will define our system model. Then we will give a brief overview about the reason for this offset. In Section(III) we discuss the factors for errors and how we solve those issues. Then in Section(IV), we will explain about the representation of data, our methods and in Section(V), we will show results of the analysis, and follows the conclusion.

II. Problem Description

The system model is motivated by Air Traffic Monitoring (ATM) systems. In our model, aircrafts accordingly broadcast a sequence of location claims, let $S = \{i, j, k....\}$ be the set of location claim events sent to stationary receivers R= $\{R_A$,R_B,....$\}$ using a wireless communication channel. The location claims are assumed to be three-dimensional Euclidean coordinates. The position of the receiver R_A represented as $P_A = (x_A, y_A, z_A)$

where x_A, y_A, z_A denotes the Geographic coordinates latitude, longitude, and altitude of the receiver R_A respectively.

A. Time Notations

In our model, we denote the global and real time as T. Each receiver has a local clock value called timestamps which are a 30 bit rolling counter value with nanosecond precision. For a receiver R_A the timestamp value is represented as t^A. Similarly, each receiver in our model has independent timestamps which give their local clock values. The time at which a location claim i sent by any aircraft in the model at time t is represented as t_i. Similarly, position claims i sent by an aircraft in our model received by a receiver R_A at its timestamp is denoted as t_i^A. But each receiver has no information about the time t_i at which the message was sent. Each location claims are sent by the aircraft is received at each receiver with a specific propagation delay. Propagation delay can be calculated easily since we have the information about the position of receivers and an aircraft. So if the receivers are synchronized, then location claim messages should be with an offset in timestamps respect to their propagation delays. But due to various factors, the offset between the receivers varies and is unknown. Since we can find the offset between the receivers by using the information in a location claim and timestamps of each message. Hence finding the unknown offset is the primary target of this project. This task of finding this unknown offset can helps to synchronize the network. This can lead the way to find the accuracy of time synchronization in our model. But phenomenons like clock drift and measurement errors make our task more difficult and resynchronizing the network.

Let $\Omega_i^{AB} = t_i^A - t_i^B$ be the offset of an event where t_i^A and t_i^B denotes the reception timestamp of a location claim event i, sent by an aircraft that received by receiver R_A and receiver R_B respectively. In synchronized network, the offsets of the event should be equal to the difference in their propagation delays. ie, Let $\theta_i^{AB} = \Delta_i^A - \Delta_i^B$ denotes the difference in propagation delay for that event i received by the receivers R_A and R_B, where

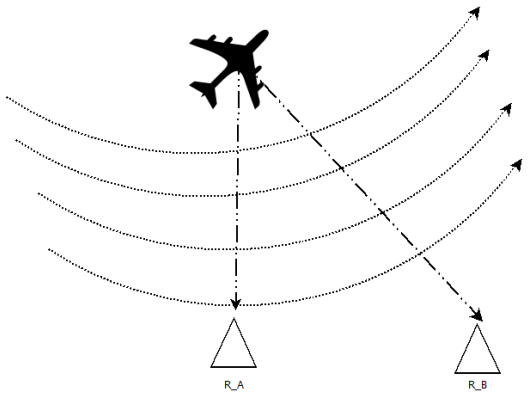


Figure 1. A graphical view of the scenario

Δ_i^A and Δ_i^B denotes the propagation delay between receiver R_A and aircraft, receiver R_B and aircraft respectively. Then synchronized network holds the condition,

$$\Omega_i^{AB} = \theta_i^{AB} = \Delta_i^A - \Delta_i^B = t_i^A - t_i^B \qquad (1)$$

but this is not the case in an unsynchronized network. So our task is to find this unknown offset and analyze the effects considering the different common phenomenons that make the network unsynchronized and prevent frequent resynchronization of the network. So let,

$$\begin{aligned}\Phi_i^{AB} = \Omega_i^{AB} - \theta_i^{AB} &= (t_i^A - t_i^B) - (\Delta_i^A - \Delta_i^B) \\ &= (t_i^A - \Delta_i^A) - (t_i^B - \Delta_i^B)\end{aligned} \qquad (2)$$

denotes the unknown offset. Since the message should have been sent at the same time but differences between their reception time will gives offset in synchronization. So as mentioned before, the primary aim of this project is to find and analyze about this unknown offset ϕ. Methods for finding this offset is explained in the later section.

B. Mathematical Approach

Consider an aircraft and any two receivers in the range of the aircraft. Let the two receivers be R_A and R_B among the whole receivers within the range of an aircraft F as in Fig:1. The positions of the two receivers denoted as $P_A = (x_A, y_A, z_A)$ and

$P_B = (x_B, y_B, z_B)$ and a position message i from the aircraft is denoted as $F_i = (x_i^F, y_i^F, z_i^F)$. For our purpose, we need to find the propagation delay between each receiver and the message send from the aircraft. In order to calculate the propagation delay, it is necessary to calculate the distance between receivers and the aircraft when it sent the location claim using the distance formula. Let D_i^A and D_i^B denotes the distance between receiver R_A and F, receiver R_B and F respectively for each location claim event i. To calculate the distance between two geographic coordinates(R_A and F),

1) Calculate the central angle between the coordinates using following equations,

$$lat1 = x_A * \pi/180$$

$$lat2 = x_i^F * \pi/180$$

$$lon1 = y_A * \pi/180$$

$$lon2 = y_i^F * \pi/180$$

$$tmp1 = (\cos(lat1) * \cos(lat2) * \sin(\frac{(lon2 - lon1)}{2}))^2 \quad (3)$$

$$tmp2 = (\sin(\frac{(lat2 - lat1)}{2}))^2 \quad (4)$$

From equation (3) and (4)

$$cAngle = 2 * \arcsin(\sqrt{(tmp1 + tmp2)})$$

2) Using central angle calculated between two coordinates, the distance is calculated using following equation,

$$radius = 6371000.8$$

$$a = radius + z_A * 0.3048$$

$$b = radius + z_i^F * 0.3048$$

$$D_i^A = \sqrt{a**2 + b**2 - 2 * a * b * \cos(cAngle)} \quad (5)$$

radius denotes the radius of the earth. The altitude is considered in feet and the calculated distance is in meters.

Using the calculated distance, we can calculate the propagation delay using the following equation. i.e.

$$\Delta_i^A = \frac{D_A}{c}$$
$$\Delta_i^B = \frac{D_B}{c} \quad (6)$$

where c denotes the speed of light.
Using the above information, we can calculate the offset between two receivers. Even we find the offset to resynchronize the network, it is necessary to tackle the issues of clock drift and measurement error. So the purpose of this project is to find the precise offset along with finding the clock drift which helps from frequently resynchronizing the network. We will talk about it more in the coming section. Also for the analysis, data should be filtered from the noise.

III. Common factors for errors and the approach

In this section, we will discuss the factors that lead to error and its effect. We mainly discuss clock drift, noise, multiple receptions etc.

A. Time synchronization and effects of clock drift

Time synchronization is one of the most basic building blocks for many applications in computer science and engineering. It is vital for any mobile distributed network where security is the key feature. Time synchronization between two mobile nodes can be affected due to different clock speeds on each system.The clock speed depends on the clock's quality, sometimes the stability of the power source, the ambient temperature, and other subtle environmental variables.Thus, the same clock can have different speed at different occasions which result in different offset. This phenomenon is defined as *clock drift*. So synchronizing the nodes at the right time is important, but synchronizing the network frequently is not a cost efficient task. So the time synchronization must be performed at the proper interval. i.e. when the clocks of the receivers run at different speed due to different factors, the timestamp of the location claim messages received varies accordingly.
The error due to clock drift ϵ_{drift} linearly depends

on the duration between two time measurements. It can be modeled by a drift coefficient t^x_{drift} for an entity X. Assuming that X wants to measure a period of time $\Lambda_{l,m} = t_m - t_l$, the clock drift error ϵ^x_{drift} of X^s measurement $\Lambda^X_{l,m}$ is given by

$$\epsilon^x_{drift} = \Lambda_{l,m} \cdot t^x_{drift} = (t_m - t_l) \cdot t^x_{drift}$$

Next we will discuss about possible causes for the measurement errors in our calculation.

B. Noise

Noise is a common issue in every wireless based communication systems. It can be either external noise due to any undesired signal in a communication circuit or unwanted disturbances. Here we consider the statistical noise. A statistical noise is a colloquialism for recognized amounts of unexplained variation in a sample, ie errors and residuals in statistics. The error of an observed value is the deviation of the observed value from the true value of a quantity of interest and the residual of an observed value is the difference between the observed value and the estimated value of the quantity of interest. Here it can be due to the inability to find the correct timestamp of the reception event since it is not able to find the accurate point in time where the location claim messages are received. But we can model these noise in the data which depends on the factors like bandwidth, signal energy, and sampling of the signal which helps to find other errors in the system. Apart from that, the noise in data might be due to some error in the aircraft transponder, which results in sending the same message multiple times or multiple receptions. We will talk about the multiple receptions in detail when we discuss data extraction.

So to remove the measurement errors, we assume that measurement errors are independent for each location claim and each of its associated timestamps. We summarize all sources of noise in a zero-mean Gaussian random variable $\epsilon \sim N(0, \sigma^2)$. The variance σ^2 depends on the accuracy of the system components involved in the process. For instance, if clocks with higher rates are used, σ^2 becomes smaller.

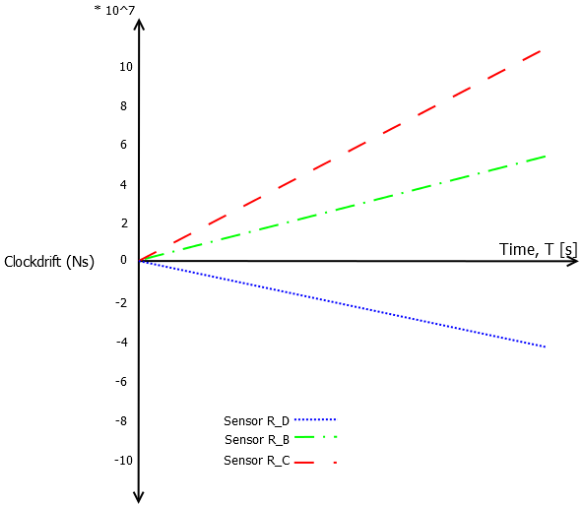


Figure 2. Drift of the internal clocks of three sensors compared to a reference sensor R_A

C. Approach

In this section, we describe the method. As mentioned earlier, the goal is to predict the synchronization error at any point in time with good precision. For this, we analyze the position messages received over an interval $\Lambda_{l,m} = t_m - t_l$ for different receivers. For each message over this time interval, the clock offset is calculated with respect to one receiver. Receivers have different clock drift depending on the surrounding factors and by finding the clock offset over a large interval, we can predict the clock drift for any message in the future.
The method is as follows:

1) Let R_A be the referenced sensor and $\forall R_x \in R - \{R_A\} \mid x = \{B, C, D..\}$ be the other sensors. Next, consider the position messages for the time interval $\Lambda_{l,m}$ for each sensor.
2) Let $E \subseteq S$ be the subset of events happened between the interval $\Lambda_{l,m}$. For an event $i \in E$, the timestamp for the event at receiver R_A and for a receiver R_x is t^A_i & t^x_i. Find the offset of the timestamp for all the sensors R_x with respect to reference sensor R_A. i.e $\forall i \in E$, find the Φ^{Ax}_i using equation 2 for all sensors R_x.
3) Calculate the Φ^{Ax}_i, such that i is the first event

during interval $\Lambda_{l,m}$ and let this be $\Phi^{Ax}_{initial}$, the initial offset. Calculate the offset deviation for every event $j \in E \mid T_j > T_i$.
Let β^{Ax}_j be difference between the initial offset $\Phi^{Ax}_{initial}$ and Φ^{Ax}_j , i. e

$$\beta^{Ax}_j = \Phi^{Ax}_{initial} - \Phi^{Ax}_j$$

Now it is important to consider the upper bound of the local timestamp counter of each receiver during the calculation. The timestamp counter runs from 0 to 999999999, ie 1 Sec. Let γ^{Ax} be the overflow parameter for offset comparison between R_A & R_x. Its is incremented by 1 when the offset is more than 1 Sec. There is two case, either the clock of the receiver R_x is running faster or running slower compare to the reference clock R_A.

a) For a receiver R_x running faster and if

$$tmp^A_j + \theta^{Ax}_j + \beta^{Ax}_k > 999999999$$

where k is an event happened before the event i then calculate,

$$\Omega^{Ax}_j = (999999999 - t^A_j) + t^B_j$$

b) For a receiver R_x running slower and if

$$tmp^A_j + \theta^{Ax}_j - \beta^{Ax}_k < 0$$

where k is an event happened before the event i then calculate,

$$\Omega^{Ax}_j = (999999999 - t^B_j) + t^A_j$$

Plot the offset β^{Ax}_i in a Clock Offset graph vs Time(β^{Ax} vs T) as show in fig(2).

4) A robust linear regression over the offset points of each sensor predicts the clock drift for that particular sensor. In the graph the x-axis represents the real time T in seconds and y-axis represents the clock drift in nanoseconds. For the ease of explanation, we consider three receivers R_B, R_C, R_D. In order to predict the offset for a sensor R_x at time T, find the slope of the line representing the clock drift of the sensor R_x. Drift coefficient t^x_{drift} (also the slope of the line) for a receiver R_x,

$$t^x_{drift} = \frac{\beta^{Ax}_i - \beta^{Ax}_j}{T^x_i - T^x_j} \tag{7}$$

where T^x_i & T^x_j is the time at which the event i & j occurs. So the equation will be

$$y = m * x + b$$
$$\beta^{Ax} = t^x_{drift} * T + 0$$

b=0 since the clock drift at the start of analysis is zero.

5) The clock drift β^{Ax}_i for any event i for a receiver R_x with respect to receiver R_A in future will be approximately ,

$$\beta^{Ax}_i = T \cdot t^x_d rift \tag{8}$$

6) So the clock drift error ϵ^x_{drift} for a receiver x of a time period $\Lambda_{l,m}$,

$$\epsilon^x_{drift} = \Lambda_{l,m} \cdot t^x_d rift \tag{9}$$

In the figure(2), R_B, & R_C are receivers with positive slop or having local clocks running faster than the reference clock and R_D have negative slop or local clocks running slower.
In this project, we perform an analysis on our network to find the error in clock synchronization between the receivers and after effects due to it. For this purpose, we use the messages in the OpenSky network, sent by the aircrafts. We consider two receiver Radarcape sensors for the analyze purpose which provides nanosecond precision but has an error deviation up to 15 nanoseconds. Next, we explain data representation, the methods used for extracting the data from the database, statistics were done on it and then the results.

IV. Details about the data and network

A. OpenSky Network and ADS-B

The OpenSky Network [4] is a research sensor network, which collects real-world air transportation communication data at a large scale and then provides the data to researchers. In particular, it records the messages broadcast by airplanes that support the Automatic Dependent

SurveillanceBroadcast (ADS-B) protocol, an upcoming communication standard for air traffic surveillance. Currently, OpenSky consists of 30 ADS-B receivers, mostly deployed in Central Europe. In ADS-B, the aircrafts continuously transmit their positions fetched by GPS to the ground station.

In ADS-B, each position messages are Compact Position Reporting format data, which need to have two data frame (one odd, and one even) to calculate one position. Compact Position Reporting (CPR) is a way of reducing the number of bits needed to transmit position whilst maintaining high position resolution ($\approx$ 5.1 meters for airborne encoding) with 35 bits. For this, the world is divided up into a number of zones and for calculation, a lookup table of so-called Latitude Zone is used. The position reports in ADS-B is in the GPS coordinates latitude, longitude, and altitude. The precision of the location sent by the aircraft depends on the GPS receiver on-board on the aircraft which has a maximum error rate up to 15 meters. So in worst case, a position message can have an error up to 20.1 meters due to the inaccuracy in GPS and CPR. For successful decoding of one position location, the receiver should receive an even and an odd encoded message within 10 seconds from each other. The positions which are successfully received are considered for the analysis. The messages received from the aircrafts are decoded using the decoder and later represented into a tabular form with needed information. For example, a decoded message contains the information as in table 1.

B. Data Representation

In this section, we give a brief overview about the data obtained for OpenSky network. In the decoded data, the information which are interested for the purpose are *SensorType, SensorLatitude, SensorLongitude, SensorAltitude, TimeAtServer, Timestamp* and *RawMessage*. Table 1 shows the data received by a sensor and *SensorType* is the hardware model which receives the message, in out case it can be either SBS-3 or Radarcape receivers. Similarly,

$SensorType$	Radarcape
$SensorLatitude$	51.758894
$SensorLongitude$	-1.256654
$SensorAltitude$	200.0
$TimeAtServer$	1.429617600176156E9
$Timestamp$	1.32164453E8
$RawMessage$	8f4ca2c999119393c0041782f9ed
$sensorSerialNumber$	Oxford
$RSSIPacket$	24.0

Table I. Information in a message

SensorLatitude, SensorLongitude and *SensorAltitude* denotes the GPS position of the receiving sensor. *TimeAtServer* denotes the server time in UNIX timestamp when the particular message i been received by the server. *Timestamp* denotes the local timestamp of the receiver when the message has been received at the sensor and *RawMessage* denotes the raw message which contains either velocity, location, identification or any other values of the aircraft's, but for the moment we are only interested in the messages which provide location information.

For the current purpose, messages which have *SensorType* value Radarcape are taken into consideration. Its because the Radarcape sensors can provide better precision than SBS-3 values. Radarcape timestamps are 30 bit rolling timestamps with nanosecond precision. *Timestamp* values can be between 0 - $(2^{30} - 1)$.

For the analysis, let the location claim i received by a sensor R_A, be

$$\{t_i^A, ser\_tmp_i^A, \Delta_i^A\}$$

t_i^A denotes the *Timestamp*, $ser\_tmp_i^A$ denotes the *TimeAtServer* and Δ_i^A as the propagation delay.

Since the information inside the message will not directly provide those information, we need to calculate it. In order to calculate Δ_i^A, use the equation(2) and following the methods mentioned in section III C.

V. Data Analysis And Results

In this section, we discuss the results after analyzing the data from the network. For time syn-

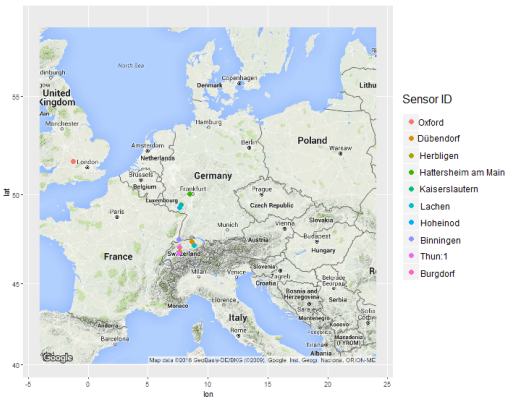


Figure 3. Position of each sensors for dataset 1

chronization, a common event observed by sensors is required. In our case, the common events are the duplicates. *Duplicates* are position messages sent from an aircraft that are received by two or more sensors. Beforehand, the data should be filtered from wrong position reports and multiple receptions. Multiple receptions are the messages that are received by the receiver more than once. It can be due to the sampling errors in the receiver. Due to this, we might get the same position message more than once. Hence, a similar location claims i can be received at two different timestamps, ie an event t_i^A can have two different timestamp value. To mitigate this, we consider the timestamp of the first message received.

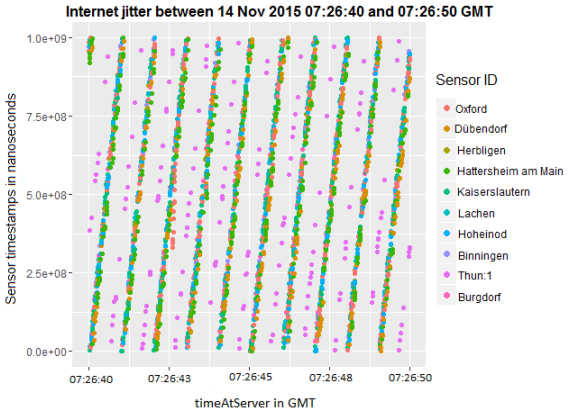
Our main goal is to accurately predict the timestamp of a duplicate received by one sensor from the timestamp of another sensor. As mentioned earlier, in a synchronized network the difference between the timestamp sensors is equal to the difference in propagation delay. We can achieve it either by synchronizing to the point at which the message reaches the server or to the time at which the message reaches the sensors. For the former, it necessary to consider the Internet Jitter (IJ) and should be able to predict the IJ. Due to high IJ and unpredictability of IJ for each sensor, we chose the second method. We will discuss the results later.

To analyze synchronization of the network, we considered the sensors as a group of two. i.e consider four sensors *A,B,C,D* and groups as (*A,B*), (*B,C*), (*C,D*). In this case, if we can predict or synchronize the offset for the sensor pair (*A,B*) and (*B,C*) results in synchronizing or correct prediction offset for the sensor pair (*A,C*). In such a way, we group the sensors in the network. Then we performed the following steps;

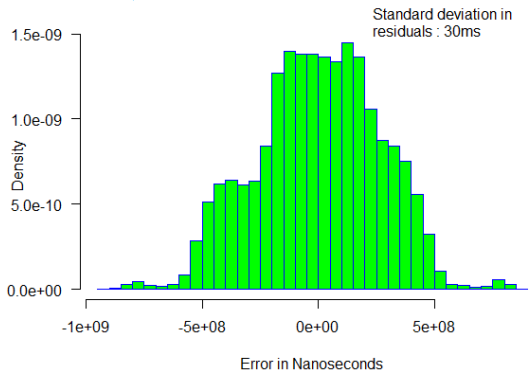
1) We checked the IJ for every message received by the sensor pair to reach the server. We were able to find that for each sensor pair the IJ was in the range of microseconds.
2) Since the IJ is in the range of micro seconds, we can find the synchronization error by comparing value of local timestamp counter of each sensor when a duplicate message is received. ADS-B messages contain the position of the aircraft when it sent the message. Using the position information of the aircraft, we can calculate the propagation delay for a message to reach the sensor. This helps to compare to our ground truth, i.e. the difference reception time in the clock values of the sensors should be equal to the difference in the propagation delay of the message, Equation (1).
3) We compare every pair of sensors to check conditions 1 and 2 mentioned above.

We performed our method in two different date sets from two different times. The number of Radarcape sensors in the first data set was less compare to the second data set. To increase the readability the sensor ID's are mapped to its location. Table 3 shows the mapping of the sensor ID to its location. We used the first data set to test our concepts. Figure 3, shows the position of each sensor under consideration for the first set of data. The data set is from 05 Dec 2015 which has data from 13 Radarcape sensors.

As the first step, we checked the IJ for the sensor pairs. Figure (4a) shows the plot for the timestamp of the local counter with respect to the time at which message reached in the server from a sensor. We repeated the process of findings the error in local timestamp counter with respect to the *TimeAtServer*. From the plot, we can observe that the sensor at *Thun:1* have lower an IJ compared to other sensors.



(a) Plot showing the timestamp and timeAtServer of each message received by the sensor's in data set 1



(b) Internet Jitter residual for the sensor at Thun:1 with respect to the sensor at Oxford

Figure 4. Observation on Internet jitter

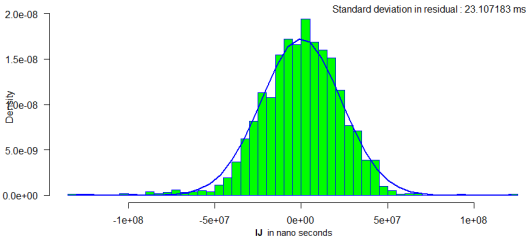
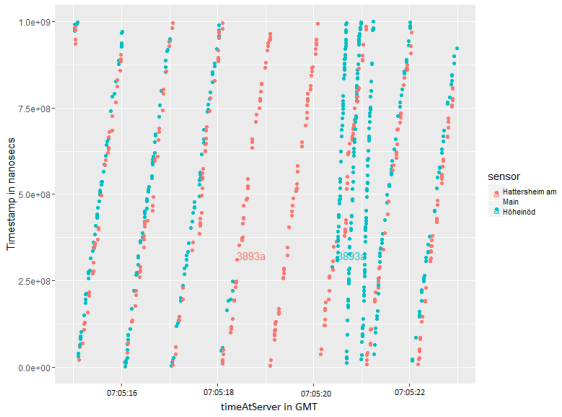


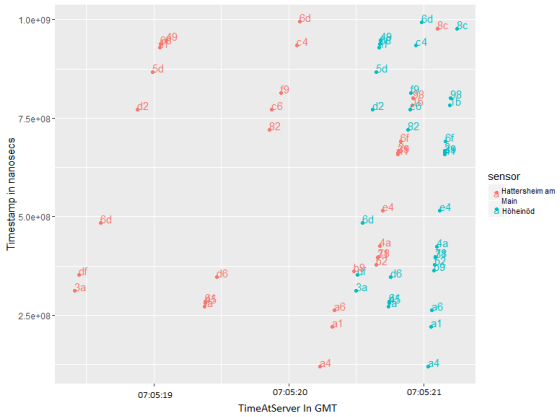
Figure 5. Internet Jitter residual for the sensor at Höheinöd with respect to sensor at Hattersheim am Main

The reason for the low jitter is because the sensor at *Thun:1* located inside the local network of the server. So the data from the sensor is not delayed by Internet traffic. Figure (4b) shows the internet jitter for the sensor at *Oxford* for the time period 1447486000 to 1447487000 with respect to the sensor at *Thun:1*. The sensor pair has IJ within a standard deviation error of 30 ms. Figure (5) shows the histogram plot for the IJ residuals of the sensor at *Höheinöd* with respect to the sensor at *Hattersheim am Main*. The standard deviation in the error for the sensor pair *Höheinöd vs Hattersheim am Main* is 23.107 183 ms. We repeated the process for each sensor and found that the sensor nodes are synchronized within the range of milliseconds. Thus, we can analyze the local timestamp counter of sensors which are in the nanosecond precision range for more precise results. But we found a different behavior in IJ for certain sensors while processing the duplicate messages. Next, we explain this behavior shown by the sensors.

Let us consider the sensors at *Hattersheim am Main* and *Höheinöd*. While parsing through the duplicate messages, we found an inappropriate behavior for the sensor at *Höheinöd*. When a duplicate message is received by the sensor at *Höheinöd* and by some other sensor, the time at which the message from the sensor at *Höheinöd* arrives in server with some extra jitter. For analyzing this, we checked the time at which several groups of duplicate message arrived at the server. Figure (6) and Figure (7) shows the *TimeAtServer* for certain messages along with duplicate messages during two different time period. Blue and red dots denotes the messages from sensor at *Hattersheim am Main* and *Höheinöd* sensor respectively. Figure (6a) and (7a) clearly shows that the duplicate messages from *Höheinöd* has an initial delay over 1.8 sec to reach the server and delay decrease rapidly for that set

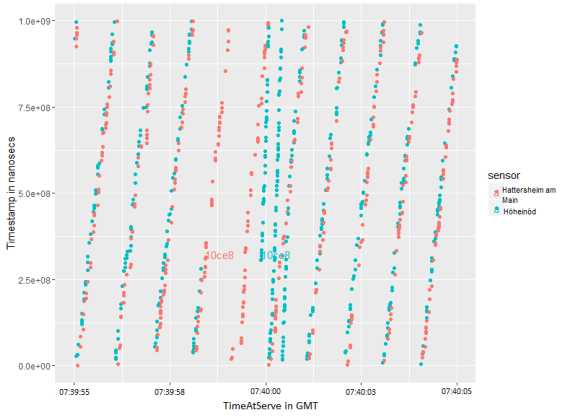


(a) Plot includes all messages received by both sensors during the interval

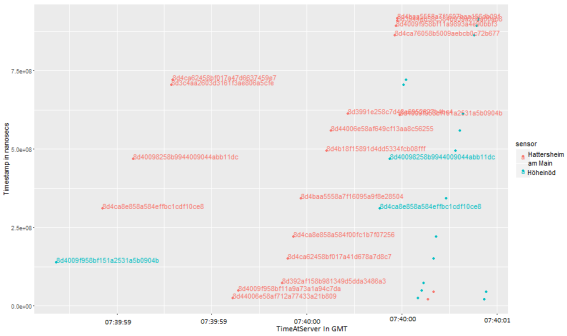


(b) Plot includes only the duplicates received by both sensors during the interval

Figure 6. Messages received by the sensor at Höheinöd and Hattersheim am Main between unix time 1447484715 and 1447484725



(a) Plot includes all messages received by both sensors during the interval

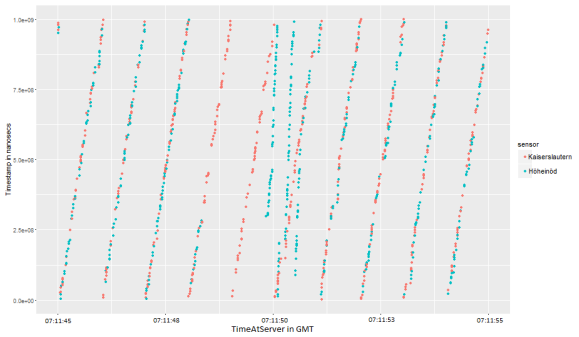


(b) Plot includes only the duplicates received by both sensors during the interval

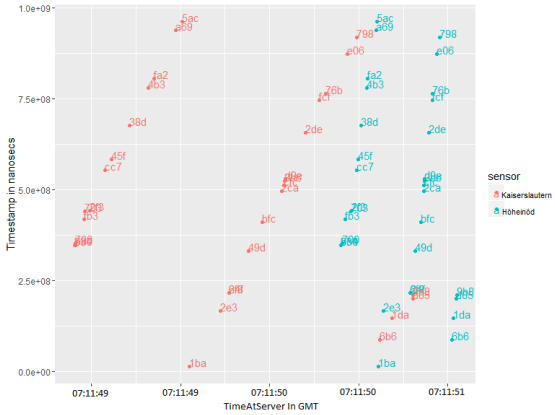
Figure 7. Messages received by the sensor at Höheinöd and Hattersheim am Main between unix time 1447486795 and 1447486805

of messages. This high delay occurs only when the sensor wants to handle a set of duplicate message. Figure (6b) and (7b) shows the deviation for each duplicate message during both periods. We repeated the process for another sensor pair at *Kaiserslautern* and *Höheinöd* and obtained similar results as in figure 8. Also did the same process for the sensor pair at *Kaiserslautern* and *Hattersheim am Main* as shown in figure 9.

We noticed that this error occurs more often for the sensor at *Höheinöd*. The reason for such incidents might be due to the buffering of duplicate messages on the server. But the reason for occurring of such delays only for duplicates is still an open question. The delay starts when a new group of duplicates is received by a group of sensors and linearly decrease for that group. Figure 10 shows the results after analyzing the jitter for sensors at *Höheinöd* and

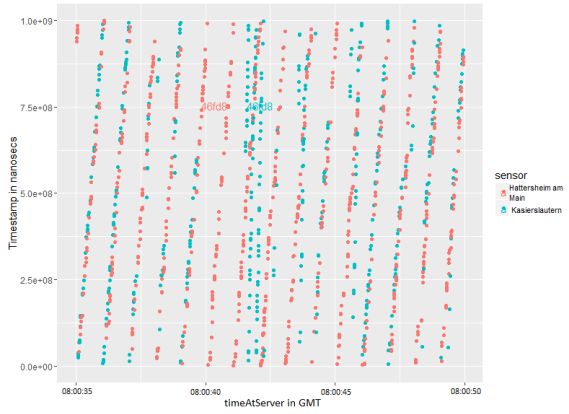


(a) Plot includes all messages received by both sensors during the interval

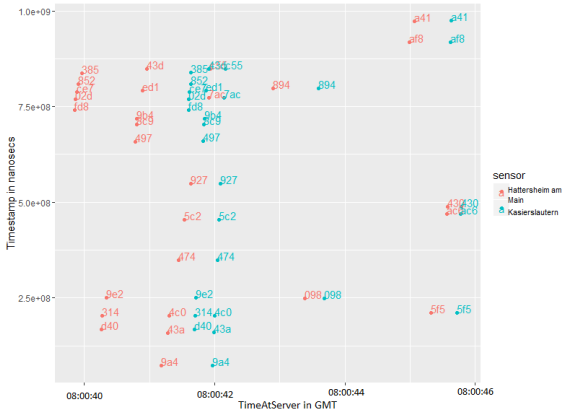


(b) Plot includes only the duplicates received by both sensors during the interval

Figure 8. Messages received by the sensor at Höheinöd and Kaiserslautern between unix time 1447485105 and 1447485115



(a) Plot includes all messages received by both sensors during the interval

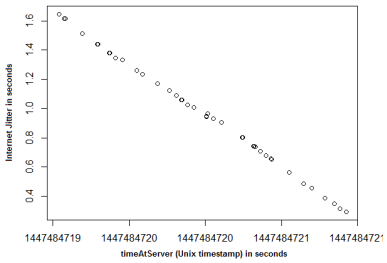


(b) Plot includes only the duplicates received by both sensors during the interval

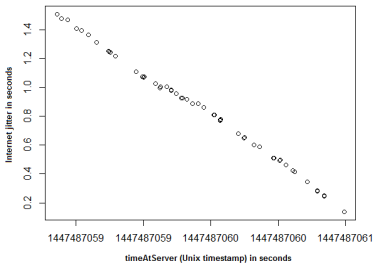
Figure 9. Message received by the sensor at Kaiserslautern and Hattersheim am Main between unix time 1447488035 and 1447488050

Kaiserslautern at different time. It is also the reason to drop the initial idea of synchronizing to the point at which the duplicate reaches the server. The IJ are unpredictable and standard deviation in error for sensors are in the range of milliseconds. Since the IJ are in milliseconds, it is impossible to perform NTP synchronization on sensors nodes. Synchronization error in the range of milliseconds can result in multilateration error of several kilometers.

After finding those problems, we started to work on the new set of data. The new dataset has more Radarcape sensors (total 15) with few of them are GPS clock synchronized and have an additional information "*timeAtSensor*". Apart from one node, all other nodes were GPS synchronized for the data set 2. Figure 11 show the position of sensors considered in the new dataset. We performed the same process of evaluating the IJ for the sensor



(a) Between 14^{th} Nov 2015 07:05:19 and 07:05:22 GMT



(b) Between 14^{th} Nov 2015 07:44:19 and 07:44:21 GMT

Figure 10. Internet jitter for the sensor at Höheinöd while receiving the duplicates with respect to the sensor at Kaiserslautern

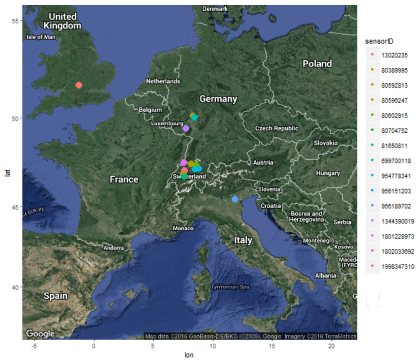


Figure 11. Position of each sensors for dataset 2

pairs in the data set 2 and obtained a similar result. *timeAtSensor* was supposed to provide the second of each day, i.e. *timeAtSensor* resets every 24h at time 23:59:59. But we didn't use this information since we found a lot of false readings for some sensors and found that some sensors resets at different points in time.

As mentioned earlier, the next step is to compare reception time of the duplicate messages received by each sensor. For this, we considered each group of sensors and evaluate the duplicate messages. Figure 12 shows a number of duplicate messages for each pair i.e. thicker the link, more the number of duplicates and vice versa. As we can observe from the figure, the sensor at *Zug* has a strong connection with others sensors in the network.

For analyzing the synchronization of the sensors, we repeat the processes mentioned before for each pair of sensors. Figure 17 shows the graph of pairs used for the purpose. Figure 18 shows the result of the sensor pair at *Thun:1* and *Thun:2* which are close enough (0.016 km apart). Figure 19 shows the sensor pair which is far apart (192.16 km apart). The results show that the sensors can have offset in synchronization with standard deviation of $144\,276\,\mathrm{ns}$.

Figure 20 shows that the clock of the sensor at *Uni-KL* is drifting apart from the network synchronization. Figure 20a and 21a shows that the sensor at *Uni-KL* has a negative clock drift. i.e

- Figure 20a shows that local clock of the sensor *Uni-KL* has negative clock drift with respect to clock of the sensor at *Hattersheim am Main*,the
- Figure 21a shows that the sensor at *Uni-KL* has negative clock drift with respect to the lock of the sensor at *Zug*.

From the above observation, we can conclude that the sensor at *Uni-KL* is not GPS clock synchronized.

The next thing observed was the error in calculating the propagation delay. The two sensors at *Thun:1* and *Thun:2* are placed in same position which means the difference in propagation delay for a message between the two sensors is zero. i.e. according to the ground truth equation (1), $t_i^{Thun:2} = t_i^{Thun:1}$ for any event i. Figure 18b shows the residual distribution for the offset in timestamps. Since the sensors at *Thun:1* and

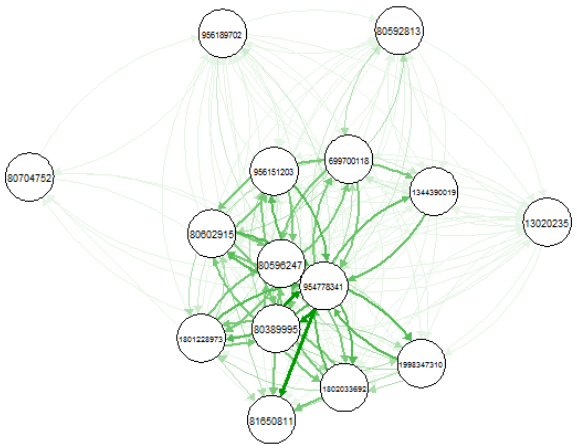


Figure 12. Amount of duplicate pairs

Thun:2 are placed in similar position, they are synchronized with a standard deviation for residuals $40.184\,670\,67$ ns. Thus, we can conclude that error in propagation delay estimation also affects our result. Also, while observing the plot for the offset of timestamps with respect to *TimeAtServer* for a sensor pair, we observed some patterns. For example, Figure 13 shows the offset for the sensor pair at *Dübendorf* and *Zug*. Offsets can seen as waveforms with some rapid jumps for a certain point in time. This continuous waveforms shows the error in calculation of propagation delay estimation. Also Figure 18 gives the ideas about how the timestamp counter works in Radarcape. The jumps in offset shows that the counter increments the value every 15 seconds (figure shows 6 lines in every 100 seconds). We found the same jumps every other plots too.

As you can observe from the Figure 12, a sensor at *Zug* overlapping lot of other sensors and thus we have the most duplicate message for this sensor. We analyzed the error in the synchronization for those sensors overlapping *Zug*'s coverage area and analyzed the standard deviation in their errors. The intention is to compare the variation of error with respect to difference in distance, and difference in bearings between the sensor at *Zug*. As mentioned

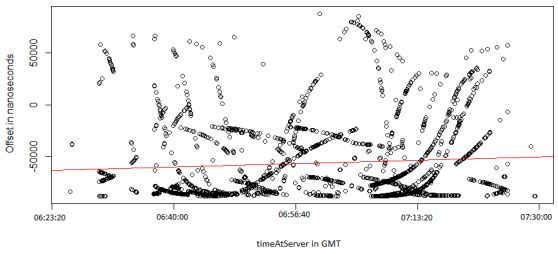


Figure 13. Offset in timestamp for the sensor at Dübendorf with respect to the sensor at Zug

earlier, Figure 13 shows the offset of local clock for the sensor at *Dübendorf* with respect to sensor at *Zug*. From the plot, we can observe that for a certain *TimeAtServer* there is two different offset value for different duplicate message. As per the Equation 1, offset can either due to the difference in the local clocks or due to the error in calculating the propagation delay. In this case, we can conclude that the error in calculation of propagation delay is also included in synchronization error. The next step is to find this error and reduce this error.

As mentioned earlier, to calculate the distance between an aircraft and a ADS-B receiver, the

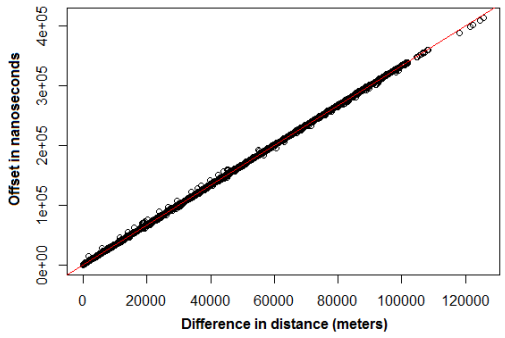


Figure 14. Propagation delay with respect to difference in distance for sensor at Hattersheim am Main with respect to the sensor at Zug

haversine model was used. In order to estimate the error in the calculation of propagation delay, we tried different propagation delay models. First the results were compared with results of Geodetic Reference System 1980 (GRS 80) [5] model which considers the earth as the ellipsoid and with World Geodetic System 1984 (WGS 84) model which is an improvement of the GRS 80 model. WGS 84 defines two models, a geometric model that describes earth as an ellipsoid of revolution characterized by the semi-major axis and earth flattening, and a physical model that relates to the average level of the oceans [6]. An upgrade to WGS 84 model is the Earth Gravitational Model 1996 (EGM 96) defines a geoid based on the gravitational force of the Earth. The altitude provides by GPS is the height over the referential ellipsoid. Next we compare the distance measures obtained using haversine model with GRS 80 and WGS 84.

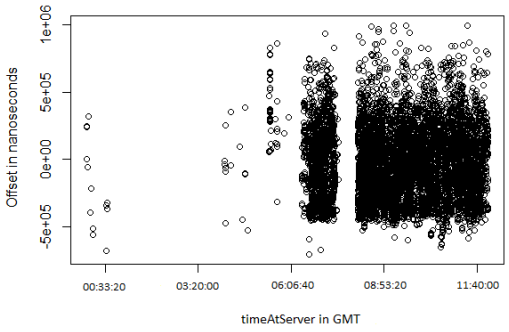
In order to remove the error due to propagation delay calculation, we first analyzed the offset in the synchronization error for each sensor pair with respect to the difference in the distance between sensors and aircraft, i.e. the distance between sensor 1 and aircraft minus the distance between the sensor 2 and aircraft. For synchronized sensor pairs, the time taken by the message to travel this distance should be the difference in their arrival time. Thus, we can predict the error in calculating the propagation delay and reduce for each duplicate messages.

For example, consider the sensor pair at *Hattersheim am Main* and *Zug* with haversine propagation delay model. The sensors are 198.717 Kms apart. Figure 15a shows the offset in the synchronization with respect to the time at which message reaches the server. Figure 16a shows the histogram for distribution of the error. As the next step, we calculate the error due to propagation delay with respect to the difference in distance and Figure 14 shows the result. As we can observe from the plot, the offset increases linearly with respect to the difference in distance. So using the equation of the line, we can predict the error in offset with respect difference in distance. We reduced this error due to the calculation of propagation delay and Figure 15b shows the result. Figure 16b shows the histogram for distribution of the error and we can observe the improvement of residual standard error from 229408.5283ns to 1096.2766ns. We repeat this process for each sensor pair in the network and observed similar improvement. For example, Table II gives the equation for estimation of error each sensor pair depending on the distance difference, *differenceInDistance*. The distance should be in meters and *Error* to be reduced is in nanoseconds. Thus, we can modify the equation 2 as,

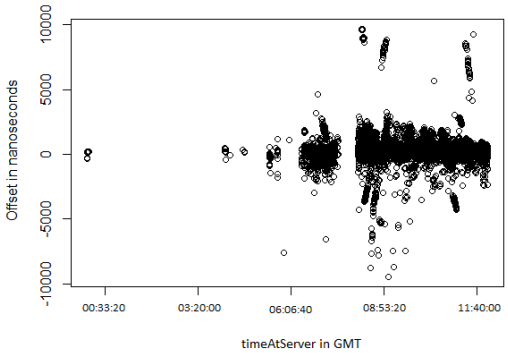
$$\Phi_i^{AB} = \Omega_i^{AB} - \theta_i^{AB} = (t_i^A - t_i^B) \\ -(\Delta_i^A - \Delta_i^B) - Error_d \quad (10)$$

where $Error_d$ is error for the difference in distance, i.e. D_i^A - D_i^B.

As mentioned earlier, to calculate the distance between an aircraft and a ADS-B receiver, the haversine model was used. For further improvement in the estimation of error due to propagation delay, we tried the same calculation by considering other models. We calculated difference in the distance calculation for a set position report among each model. Figure 22, 23 and 24 shows the results. As we can see, using either GRS 80 or WGS 84 will give a better result compare to using haversine

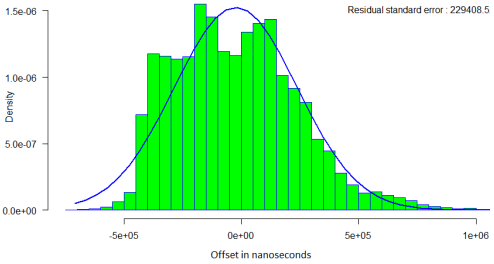


(a) Including the error in propagation delay calculation

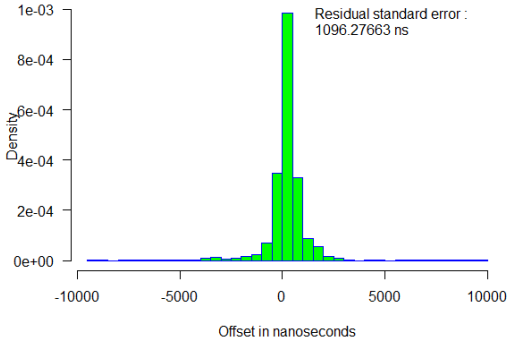


(b) Excluding the error in propagation delay calculation

Figure 15. Synchronization offset for the sensor pair Hattersheim am Main vs Zug



(a) Including the error in propagation delay calculation



(b) Excluding the error in propagation delay calculation

Figure 16. Histogram showing residual distribution Hattersheim am Main vs Zug

Table II. Equation for estimating the error during the calculation propagation delay WGS 84 model

Sensor Pair	Equation
Belp vs Zug	Error = 2.998491e+03 + (3.340290e+00 * differenceInDistance)
Hofheim am Taunus vs Zug	Error = -5.299840e+02 + (3.342852e+00 * differenceInDistance)
Lupfig vs Zug	Error = 2.785623e+02 + (3.246888e+00 * differenceInDistance)
Dübendorf vs Zug	Error = -1.332355e+02 + (3.299196e+00 * differenceInDistance)
Thun:2 vs Zug	Error = 1.077133e+03 + (3.267332e+00 * differenceInDistance)
Hattersheim am Main vs Zug	Error = 2.492813e+02 + (3.329933e+00 * differenceInDistance)
Lachen vs Zug	Error = 3.533047e+02 + (3.280929e+00 * differenceInDistance)
Binningen vs Zug	Error = 2.641018e+02 + (3.280574e+00 * differenceInDistance)
Thun:1 vs Zug	Error = 4.377751e+02 + (3.296593e+00 * differenceInDistance)
Burgdorf vs Zug	Error = 2.825037e+02 + (3.306096e+00 * differenceInDistance)

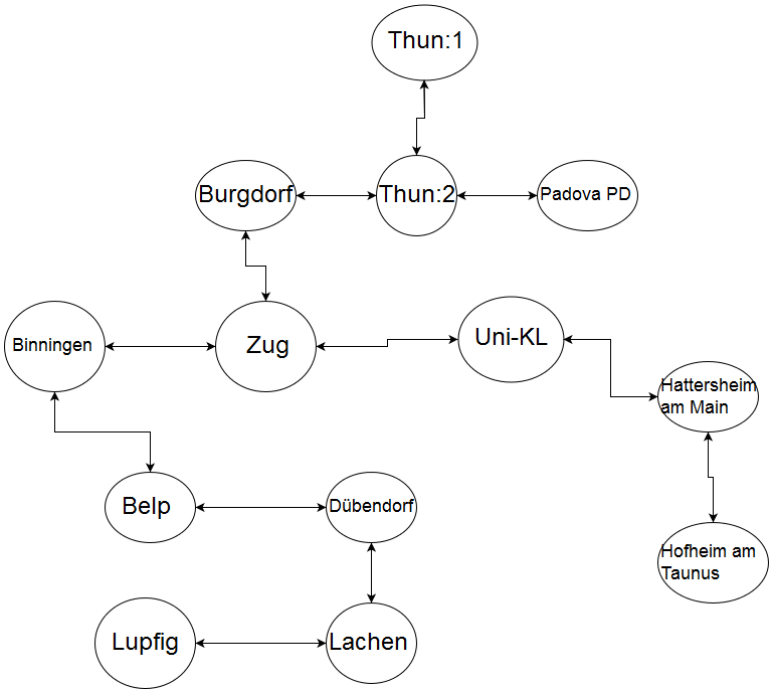
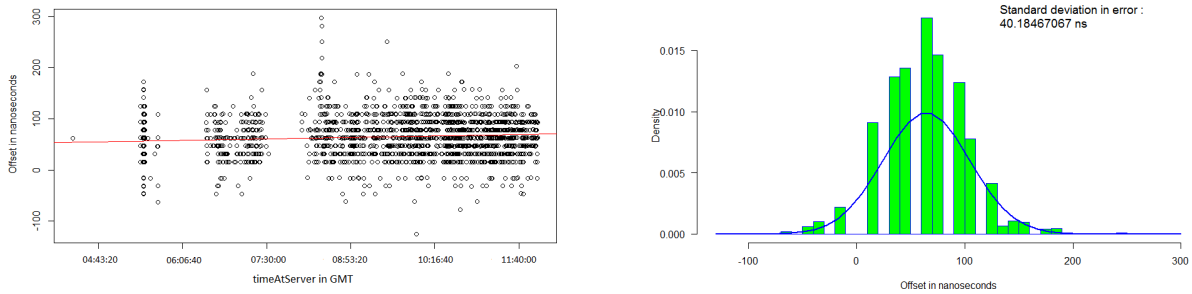


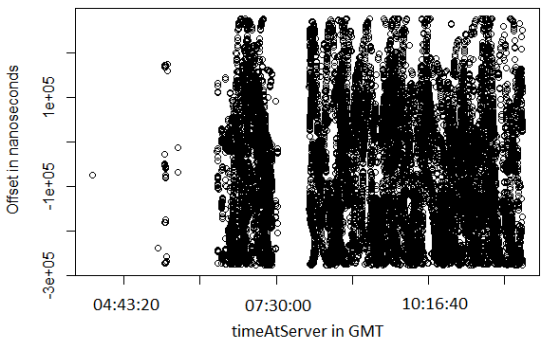
Figure 17. Plot representing the pairs



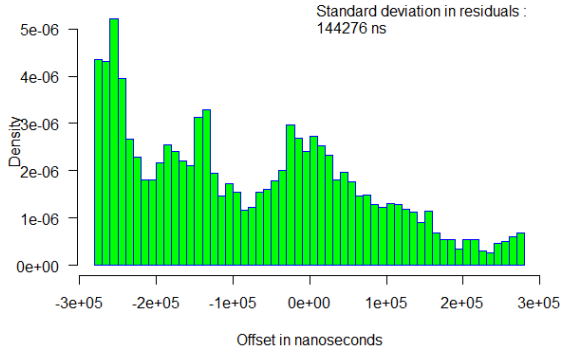
(a) offset vs timeAtServer

(b) Histogram showing the offset

Figure 18. Sensor pair Thun:1 vs Thun:2

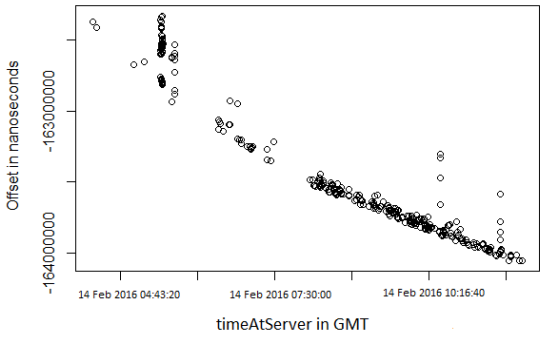


(a) offset vs timeAtServer

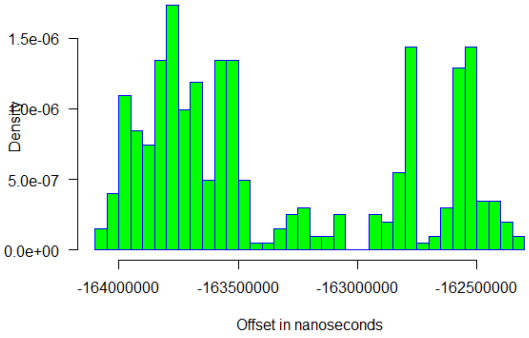


(b) Histogram showing the offset

Figure 19. Sensor pair Zug vs Binningen



(a) offset vs timeAtServer

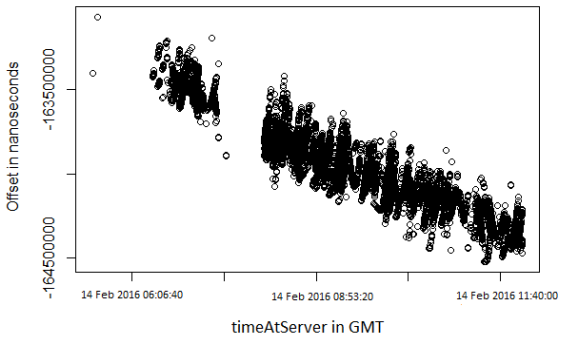


(b) Histogram showing the offset

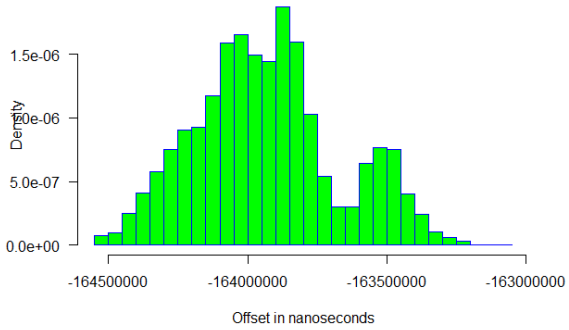
Figure 20. sensor pair Hattersheim am Main vs Uni-KL

model. We repeat this each sensor observed similar improvement. For example Figure 25 shows the difference in the standard deviation in residual error with respect to the difference in distance, depending on the propagation delay model. Figure 26 shows the difference with respect to the difference in bearings between sensors. Even after removing the error due to the calculation of propagation delay, error exists within the limit of 2000 ns or 2 µs. This can be due to several reasons and next we discuss it.

One of the reason may be the error in the position reports sent by the aircraft. The altitude sent by aircrafts are pressure altitudes which are different from true altitude [7]. In order to calculate the true altitude, information such as the atmospheric pressure of that particular position and others are required. In our calculation, we assume pressure altitude as the true altitudes, i.e. the height from referential ellipsoid. The altitude value sent from aircraft set the atmospheric pressure as 29.92 inHg/1013 hPa for its calculation. This difference between pressure



(a) offset vs timeAtServer



(b) Histogram showing the offset

Figure 21. sensor pair Zug vs Uni-KL

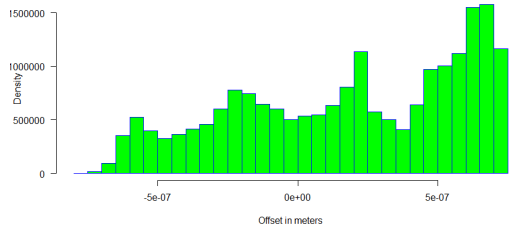


Figure 22. Histogram showing the difference in distance calculation for GRS80 with respect to WGS84 model.

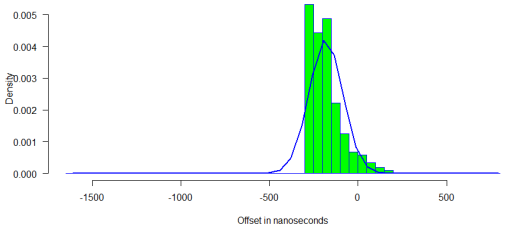


Figure 23. Histogram showing the difference in distance calculation for GRS80 with respect to haversine model.

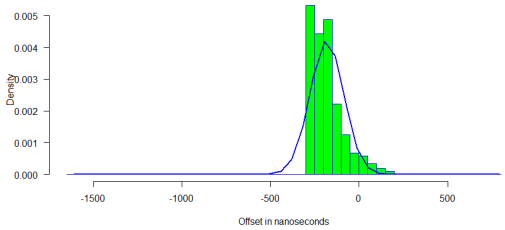


Figure 24. Histogram showing the difference in distance calculation for WGS84 with respect to haversine model

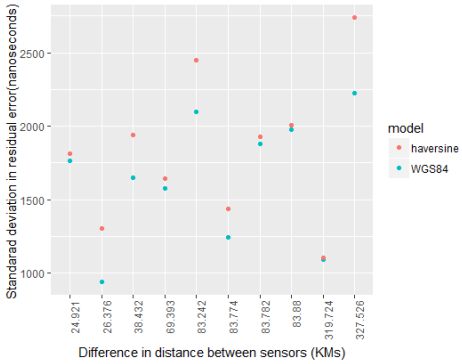


Figure 25. Plot showing the difference in synchronization with respect to the difference in distance, depending the propagation delay model.

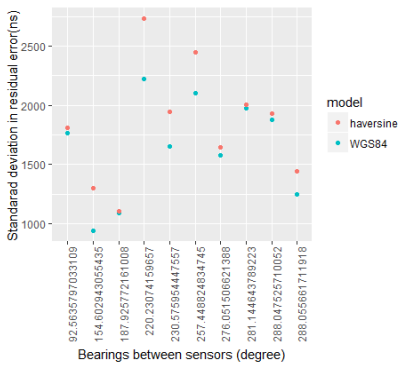


Figure 26. Plot showing the difference in synchronization with respect to the difference in bearings, depending the propagation delay model.

altitude and true altitude adds error in the distance calculation. Another reason for the error is the inaccuracy in the position information of the receivers. The GPS position of a node in the OpenSky network is given by the user. This inaccuracy can also contribute the error in calculating the distance between an aircraft and receiver.

VI. Conclusion

The goal of the project was to analyze the time synchronization of the sensor nodes in the OpenSky network. To achieve the target we analyzed two data sets from a different point in time. To use the data sets, the data sets were filtered from wrong position reports, multiple receptions and from other errors. While analyzing the data, we found out the faults in the network such as the unpredictable Internet Jitter for some sensors while processing ”duplicates”, the incorrect values for *timeAtSensor*, unsynchronized nodes in the network and so on. We also checked the precision of Radarcape timestamp counters. To improve the accuracy in calculating the distance between an aircraft and receiver, and to reduce the error in the calculation of difference in propagation delays, we analyzed the error rate with respect to the difference in the distance traveled between aircraft and receivers. Even though it was able to remove the error in the calculation of propagation to a certain limit, the timestamp at which a duplicate received by a sensor can be predicted with a timestamp of another sensor with an upper limit in error of 2000 ns or 2 µs. We also tried different methods to synchronize the network, but found that those methods will not give synchronization with in the range of nanoseconds. For further improvement, the position reports of the sensors need to be precise.

References

[1] “Time Synchronization,” http://www.cs.usfca.edu/~srollins/courses/cs686-f08/web/notes/timesync.html, accessed: 2016-01-23.

[2] M. Schfer, V. Lenders, and I. Martinovic, “Experimental analysis of attacks on next generation air traffic communication,” in *Applied Cryptography and Network Security (ACNS ’13)*, ser. Lecture Notes in Computer Science, vol. 7954. Springer Berlin Heidelberg, Jun. 2013, pp. 253–271. [Online]. Available: discofiles/publicationsfiles/SLM13.pdf

[3] M. Schäfer, V. Lenders, and J. Schmitt, “Secure track verification,” in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 199–213.

[4] M. Schfer, M. Strohmeier, V. Lenders, I. Martinovic, and M. Wilhelm, “Bringing Up OpenSky: A Large-scale ADS-B Sensor Network for Research,” in *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, ser. IPSN ’14, Berlin, Germany, April 2014, pp. 83–94.

[5] H. Moritz, “Geodetic reference system 1980,” *Journal of Geodesy*, vol. 54, no. 3, pp. 395–405, 1980.

[6] J. Schreuders, “Tracking aircraft with parsax,” Ph.D. dissertation, TU Delft, Delft University of Technology, 2014.

[7] “The Different Types of Altitudes,” http://www.meretrix.com/~harry/flying/notes/altitudes.html, accessed: 2016-04-23.

Sensor ID	Location
1998347310	Burgdorf
1801228973	Binningen
956151203	Lachen
699700118	Hattersheim am Main
80704752	Herbligen
1802033692	Thun:1
13020235	Oxford
820995539	Kaiserslautern
1344380651	Höheinöd
80602915	Dübendorf
1344390019	Uni-KL
80389995	Belp
80592813	Hofheim am Taunus
80596247	Lupfig
954778341	Zug
81650811	Thun:2
956189702	Padova PD

Table III. Table showing the sensor ID of the sensor's in a location