2009 American Control Conference
Hyatt Regency Riverfront, St. Louis, MO, USA
June 10-12, 2009

Distributed Triangulation in the Presence Faulty and Byzantine Beacons in Aircraft Networks with ADS-B Technology

Daniel Uhlig, Negar Kiyavash and Natasha Neogi

Abstract— **This paper explores two different algorithms designed for quick triangulation in the face of numerous incorrect measurements. The incorrect measurements can be randomly faulty or maliciously converging to an incorrect answer. Both algorithms require the number of correct measurements to exceed a user defined consensus threshold. Both algorithms will correctly terminate in an environment possessing more than 50% faulty beacons, as long as the number of correct measurements exceed the consensus threshold.**

I. INTRODUCTION

Automated Dependent Surveillance Broadcast (ADS–B) is the state of the art technology employed for inter-aircraft communication in today's National Airspace System. With many ADS-B equipped aircraft in an area, the aircraft form a sensor network continually broadcasting the position of ADS-B equipped aircraft.

Location information, based on the position of sensor nodes, is useful in networks with varying node capabilities. Distributive networks must handle different types of incorrect nodes. The research application studied in the following paper is the ADS-B aircraft communication network. The network was selected because of the available data (accurate time, position, speed) and the fact that there is some level of government oversight over this emerging technology.

GPS has seemingly solved the positioning problem for many users, but shortcomings still exist [1]. A variety of methods relying on alternative broadcast data have been proposed to overcome these shortcomings [2]. Triangulation based on existing signals must undergo more robust metrics to filter out faulty or seemingly malicious data. ADS-B information is available anywhere there is significant commercial air traffic, including most large urban areas. Beyond ADS-B, the ideas within this paper can be applied to numerous triangulation applications, such as undersea buoy navigation systems [3].

In this paper, we will outline two algorithms that are fault tolerant to both malicious and accidental faulty aircraft broadcast messages. In the face of purely accidental faulty aircraft (non-colluding), the algorithms are able to terminate correctly *even when the number of faulty aircraft significantly outnumbers correct aircraft*. The algorithms are also able to terminate correctly when the number of colluding byzantine beacons remains less than the correct beacons. Both algorithms have regimes where one is more efficient than the other, depending on the outliers model and user requirements. The execution time of each algorithm is explored with respect to inputs and model of the incorrect beacons.

D. Uhlig is a Research Assistant at the Coordinated Science Laboratory University of Illinois Urbana-Champaign duhlig2@illinois.edu

N. Kiyavash is a Assistant Professor with the Department of Computer Science at University of Illinois Urbana-Champaign kiyavash@illinois.edu

N. Neogi is a Assistant Professor with the Department of Aerospace Engineering at University of Illinois Urbana-Champaign noegi@illinois.edu

II. POSITION ESTIMATION AND CONSENSUS

Several popular position estimation algorithms that do not use GPS-like infrastructure are presented in [4], [5], [6]. Li et al. propose the use of robust outlier detection statistical models to achieve robust position estimation [7]. They propose a probabilistic approximation to the least median of squares (LMS) approach [8] in order to circumvent computational complexity. Liu et al. presented a greedy algorithm to filter out the attacker's data on the basis of a consistent minimum mean square error (MMSE) criterion between received measurements from multiple beacons [9]. As shown in [10], the approach of Kiyavash and Koushanfar removes the anomalies in a shorter runtime than both the greedy algorithm of [9] and LMS with superior accuracy. In the context of sensor networks, randomized consensus has been applied to distributed object tracking [11] and time-synchronization [12].

The Byzantine generals problem is one of the most studied scenarios in computer science [13],[14], [15], [16]. The seminal paper of Lamport [17] demonstrated that consensus under the presence of failures could be reached in a distributed synchronous system only if the number of faulty agents was less than one third of the correct agents.

III. PROBLEM STATEMENT

The vast majority of aircraft, particularly general aviation are not ADS-B enabled. Since ADS–B is, at present, an unencrypted broadcast, general aviation aircraft can access the messages broadcast by ADS–B enabled aircraft. If a general aviation aircraft is equipped with an ADS–B receiver it can theoretically eavesdrop on all ADS-B traffic within broadcast range. Based on these eavesdropped messages, it becomes possible for general aviation aircraft to triangulate their position, as long as there are sufficient ADS–B enabled aircraft within their proximity.

The ADS–B broadcast contains a number of different parameters that are updated for each new packet. The parameters included are still being finalized, but the position, velocity and time (based off GPS) form the most basic layer [18]. When many aircraft are broadcasting ADS–B

978-1-4244-4524-0/09/$25.00 ©2009 AACC

information within range of a receiver, they form a sensor network of beacons that can be utilized by receivers. By combining time stamp and location information the receivers can triangulate their own location within the sensor network of aircraft.

A. Limitations and Ordering of ADS-B Broadcast

The ADS–B protocol is designed to broadcast at a frequency of 1 Hz. There is an implicit assumption of synchronicity, as all ADS–B aircraft are assumed to be broadcasting at the same rate

The order that the messages are received, as well as the unique identifier that is associated with a beacon, cannot be altered. Thus, under ADS-B, the aircraft implement a totally ordered broadcast protocol. Messages are processed in the order in which they arrive.

B. Triangulation

Existing triangulation methods [19] are implemented to resolve the GPS positioning problem. They take the location of each broadcast beacon (satellite) along with a time stamp and calculate a travel time between the beacon and receiver. The travel time is proportional to the distance between the two nodes. The intersection of four spheres can uniquely define a single point in three dimensional space and is used to define a GPS location [20]. Triangulation in two dimensions is done using the intersection of three circles to uniquely define a single point [21]. There are a number of degenerate cases where the intersection does not result in a point (i.e. two of the circles are identical, etc.). The geometry is shown in figure 1 and the triangulation algorithm [22] is outlined below.

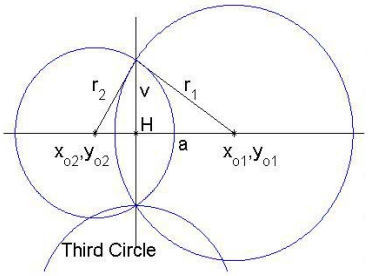

Fig. 1. The geometry of circle-circle intersection

The two circles centered at x_{01}, y_{01} and x_{02}, y_{02} with radii r_1 and r_2 are defined by equations:

$$(x-x_{o1})^2+(y-y_{o1})^2 = r_1^2 \quad (1)$$

$$(x-x_{o2})^2+(y-y_{o2})^2 = r_2^2 \quad (2)$$

First, the distance between the circle centers is found

$$x_d = x_{o2}-x_{o1}, \quad y_d = y_{o2}-y_{o1} \quad (3)$$

$$d = \sqrt{(y_d^2+x_d^2)} \quad (4)$$

The points of interest are on a line perpendicular to the line that connects the circle centers. The two lines intersect at a point labeled H. This point is distance a from the center of the first circle.

$$a = (r_1^2-r_2^2+d^2)/(2d) \quad (5)$$

$$H_x = x_{o1}+(x_d a/d), \quad H_y = y_{o1}+(y_d a/d) \quad (6)$$

The length of v (see figure 1) can be found using a and the radius of the first circle. The points of intersection are:

$$p_x = H_x \pm v(y_{o2}-y_{01}), \quad p_y = H_y \pm v(x_{o2}-x_{01}) \quad (7)$$

The pair of points of intersection, referred to as p and p', are used extensively in one of the algorithms as outlined in Section IV-B. Once the two points are found, the correct point can be selected by verifying the distance from each potential point and the third circle. The point that lies on the third circle is the point of intersection. If the three circles do not result in an intersection, then the triangulation fails to find a common point and generates an error with no point of intersection.

C. Colluding Strategies

Correct beacons generate position and distance information that is consistent with the receiver's true position. Incorrectly performing beacons do not generate information that is consistent with the receiver's true position. Incorrect beacons fall into two subcases. In the first case, faulty beacons have independent random errors that result in the beacon's broadcasts not conforming with receiver's true position. The second case involves multiple beacons being consistent with a point other than the beacon's true position. Multiple beacons may be collusive in nature and select a single coherent estimated position that is consistent with all of their messages. Colluding beacons do not have to be malicious, but may simply be faulty in a similar manner. For instance, an airline wide mistake or aircraft specific bug may cause sets of ADS-B beacons to experience identical faults resulting in behavior that looks malicious.

IV. Outline of Algorithms

Unlike the previous approaches to fault tolerant position estimation under consensus [8], [7], [9], which use as much data as possible to estimate the unknown coordinates, our approach starts by picking a small (but sufficient) subset of the data and incrementally increasing the subset.

The first algorithm, referred to as Random Selection, involves drawing three beacons in sets (called triples) until a set of three correct beacons is drawn. The second algorithm, Intelligent Redraw, processes beacons one at a time until the correct estimate is found. Both algorithms run in a distributed fashion on all triangulating aircraft, receiving data from a buffer that ensures the totally ordered broadcast. The algorithms process one time step of data at a time, calculating the receiver's true position.

A. Random Selection Algorithm

The Random Selection algorithm starts by selecting a small (but sufficient) subset of the data and subsequently augments it with consistent data. The algorithm arbitrarily selects an initial subset and employs a randomized algorithm to determine the set of consistent measurements. The pseudocode is formally stated in Algorithm 1.

Algorithm 1. Random Selection

Input: set S of N messages,
CI: the δ-consistency interval
C_t: the consensus threshold,
i_{max}: maximum number of iterations .

1. Initialize i=1, $L = \emptyset$;
2. **While** $(i < i_{max})$ {
3. Randomly draw a unique subset Si of size 3 from $S \backslash L$[1];
4. Use Si to estimate the position $\hat{s}_0$;
5. Calculate K, the number of δ-consistent beacons with respect to the estimate $\hat{s}_0$ in $S \backslash Si$;
6. **If** $(K > C_t)$ {
7. Form new estimate $\hat{s}_0$ from K consistent points;
8. Terminate the program and return $\hat{s}_0$;}
9. Increment $L \leftarrow Si$; }
10. Increment i; }
11. Terminate program by announcing failure

In light of the algorithm's minimalist methodology, the approach first estimates the position of the unknown node $\hat{s}_0$ from some randomly selected unique subset of beacons, Si (Steps $3-4$). The motivation for keeping record of already picked triplets (Step 3) is to save on computational complexity. Thus, every subsequent triple drawn is compared against the list of stored triples, and discarded if it already appears, leading to another drawing, without incrementing i. If a lexicographical order is applied to each triple as it is stored, then comparison and storage can be reduced to the act of inserting unique items into an ordered list. The triangulation process outlined in subsection III-B is then applied (Step 4), and an estimate for the position, $\hat{s}_0$, is calculated, and stored in a linked fashion. Using the new position, $\hat{s}_0$, the consensus with all beacons is checked (Step 5). The check compares the δ-consistency of each beacon's reported measurement with the estimated $\hat{s}_0$. To check if a consensus is reached, the number of beacons , K, in consensus with the estimated position, is compared against a threshold C_t. Once a consistency set has been identified, the algorithm uses all points in the consensus set to form the final estimate of $\hat{s}_0$, then terminates (Steps 7-8). If the algorithm performs i_{max} iterations and does not find a consensus set of at least size C_t, it declares a failure and terminates (Step 11).

On the surface, i_{max} and C_t are fixed input quantities, and the computational complexity of the algorithm is a function of these two variables. The storage complexity is a function of i_{max}. These quantities depend on the size of S (i.e. N).

[1]With abuse of notation, we will use $S \backslash L$ to denote the set of beacons excluding the triplets already chosen. More precisely, this does not exclude picking an already selected node, but prevents picking an already selected triplet.

B. Intelligent Redraw Algorithm

The Intelligent Redraw algorithm starts by selecting a pair of messages, then incrementally augments this set with additional messages until an estimated position reaches consensus. The pseudocode is formally stated in Algorithm 2.

Algorithm 2. Intelligent Redraw

Input: set S of $m_{i:N}$ messages,
CI: δ-consistency interval,
C_t: number of delta consistent points to achieve consensus.
Internal variables: $ListofPoints$ list containing estimated, positions
m_i message contains a unique Beacon ID, (x,y) position and distance.

1. Initialize $ListofPoints$;
2. For $i = 2 : N$ {
3. If m_i agrees with no elements in $ListofPoints$[2]{
4. For $j = 1 : i-1$; {
5. Calculate p and p' from intersection of m_i and m_j
6. Append $listofPoints$ with p and p';
7. Calculate K, the number of δ-consistent beacons with respect to the estimate p;
8. **If** $(K > C_t)$ {
9. Form new estimate $\hat{s}_0$ from K consistent beacons;
10. Terminate the program and return $\hat{s}_0$;}
11. Calculate K, the number of δ-consistent beacons with respect to the estimate p';
12. **If** $(K > C_t)$ {
13. Form new estimate $\hat{s}_0$ from K consistent beacons;
14. Terminate the program and return $\hat{s}_0$;}
} } }
15. Terminate program by announcing failure;

This algorithm uses a data structure that contains all estimated positions. The Euclidean distance between a beacons's position and an estimated position is defined as d_e. A notion of agreement is used in this algorithm to define when d_e is within a margin of error of the beacon's reported distance. The margin of error is the δ-consistency metric. The algorithm processes each beacon incrementally from the list of all beacons (Step 2). If the selected beacon agrees with any previously estimated positions then Steps 4-14 can be skipped, since position has already been checked. The algorithm then returns to step 2.

If the beacon agrees with none of the previously estimated positions, the selected beacon is intersected with all previously drawn beacons. For all intersections resulting in estimated positions p and p', these two points are appended to $ListofPoints$ (Steps 5–6) and the consistency metric K is calculated for both estimated positions (Steps 7–11). If K exceeds C_t, meaning enough beacons agree with the estimated position to achieve the required consensus threshold, the algorithm terminates returning that estimated position. If not, i is incremented and the process is repeated with another beacon being selected (Step 2).

[2]Agreement(m_i, $ListofPoints_k$)=$False$ for all $k, 1 \leq k \leq$ cardinality of $ListofPoints$

C. Parameters

The δ-consistency metric, CI, allows the user to specify the distance that an estimated position and beacon can deviate and still be in agreement. In the real world, all measurements include noise; the δ-consistency metric is an Euclidean distance that should be large enough so all correctly behaving beacons are δ-consistent with the correct location.

Consensus is defined as having greater then a threshold, C_t, number of beacons in agreement with an estimated position. Each estimated position will have at least two beacons in agreement, and the true position will have all correct beacons in agreement. Byzantine beacons can select a byzantine position and coordinate their effort to be consistent. This results in the preselected position being in agreement with all byzantine beacons.

The user can select for the consensus threshold, C_t, any value from 3 to the total number of beacons, N. Values above $N/2+1$ ensure that all cases with byzantine beacons numbering less than $N/2-1$ terminate with the correct position. With the threshold value (C_t) set between 3 and $N/2+1$ the algorithm terminates correctly in cases where at least C_t beacons are correct, and no more than C_t-2 beacons are byzantine. Simulations has shown the threshold can be set as low as 10% in cases of 80% faulty beacons (with no byzantine beacons). When the user selects values less then $N/2+1$, the algorithm can no longer differentiate between byzantine groups of beacons numbering more than C_t and the correct beacons. When setting the threshold below $N/2+1$ the user must keep the threshold above the maximum number of anticipated byzantine beacons or risk selecting the byzantine position.

V. Discussion of Performance

The two triangulation algorithms have different computation and storage costs as outlined below. The algorithm that performs best depends on the scenario for incorrect beacons. The two scenarios explored to benchmark algorithmic performance are as follows: (1) Up to 50% byzantine beacons all in collusion with one another and (2) high percentages of accidently faulty beacons in excess of 50% (ie 50%–80%).

A. Scenario Initialization

For basic triangulation, N beacons were randomly distributed in a 100 by 100 grid in a uniform manner. The correct position and the byzantine position, when necessary, for triangulating aircraft, were placed at different points. Simulations were run varying N at 30, 50 and 100 beacons. Correct beacons reported a position and distance that was in agreement with the true position. Accident faulty beacons, which were varied from 0% to 80%, reported random positions and distances. Byzentine beacons reported correct positions, but adjusted their distance to be consistent with the byzantine position for the triangulating beacon. Byzentine beacons all colluded towards one position and were varied from 0% to 49% in number. The consensus threshold, C_t, was set at $N/2+1$ in the Byzantine case and it was decreased for cases involving only accidentally faulty beacons. Every point on the graphs represents the average over 100 independent runs.

B. Accidentally Faulty Scenario

In the case of solely faulty beacons, the time to terminate with the correct position increases with the percentage of faulty beacons. The algorithms require more draws to select pairs or triples of correct beacons and estimate the correct position when there are more faulty beacons. The cost of the algorithm is measured via execution time.

For the Random Selection Algorithm two main computational steps contribute to the overall execution time: (1) Triangulation and (2) Checking if an estimated position reaches consensus. The primary contribution to execution time comes from the triangulation operation. On average, triangulating a triple takes 0.273 milliseconds. Thus, as the percentage of faulty beacons increases, the number of incorrect triangulations increase, causing this component of the execution time to scale with the percentage of faulty beacons. The secondary contribution to the cost comes while checking if an estimated position, $\hat{s}_0$, is consistent with at least C_t beacons. As C_t increases, so do the number of comparisons that need to be made. On average, for $C_t = N/2+1$ and $N = 50$, a consensus check takes 0.24 milliseconds, for 20% faulty beacons. This portion of execition costs scales linearly with C_t and percentage of faulty beacons. Figure 2 illustrates how the overal execution time scales with N and percentage of faulty beacons.

The execution time of Intelligent Redraw has three contributing factors: (1) Pairwise triangulation (2) Checking if an estimated position reaches the consensus threshold (3) Checking if a beacon agrees with any previously estimated positions.

The primary contribution comes from the pairwise triangulation, which requires on average 0.171 milliseconds to complete. Note, this is shorter than the full triangulation required for Random Selection. Again, as the percentage of faulty beacons increases, the required number of pairwise triangulations increases, causing this component of execution time to scale with the percentage of faulty beacons. As before, checking if an estimated position achieves the consensus threshold scales the execution time linearly with the number of beacons. The final component scales with the number of valid intersections generated by the beacons. This component of cost is most important in the byzantine case and will be discussed in Section V-C. Figure 4 shows how the execution time scales with N and the percentage of faulty beacons.

Figure 3 shows the execution time for Intelligent Redraw becomes faster then Random Selection when the percentage of faulty beacons becomes high enough. The result for $N = 30$ is shown in figure 3. The percentage of faulty beacons that results in this cross over depends on N. The consensus check component of execution time depends on N, and since Intelligent Redraw must perform two checks per pairwise

triangulation, it scales with N. The cross over points are tabulated for $N = 30, 50, 100$ in Table I.

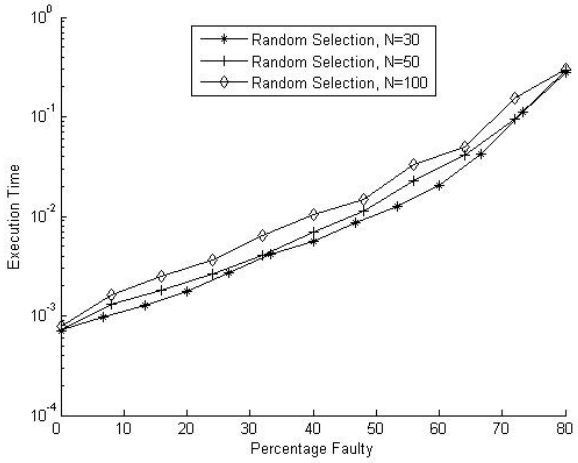

Fig. 2. The Random Selection algorithm with differing N values and the percentage of faulty beacons varied from 0–80%. The threshold is set at $C_t = 0.1N\%$.

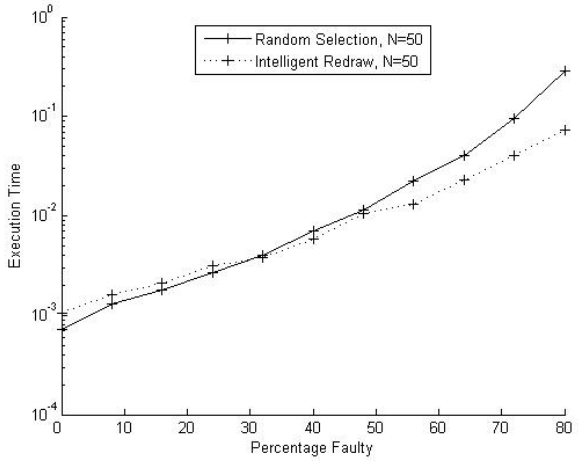

Fig. 3. The Random Selection algorithm compared with the Intelligent Redraw algorithm with the percentage of faulty beacons varied from 0–80%. Note the cross over at 30%.

C. Collusive Case

Unlike faulty beacons which do not generate a consistent point, byzantine beacons generate a consistent point upon triangulation. Random Selection still must draw triples until three correct beacons are selected. As with the faulty case the number of draws required to achieve a correct triple increases with the percentage of byzantine beacons (see fig. 5). The Random Selection algorithm is insensitive to the fact of whether an incorrect beacon may be accidentally faulty or byzantine. Figure 6 (with $N = 50$, $C_t = N/2 + 1$) shows the execution of the Random Selection algorithm essentially the same for colluding byzantine beacons and accidentally faulty beacons.

TABLE I

PERCENTAGE OF FAULTY BEACONS ABOVE WHICH INTELLIGENT REDRAW IS FASTER THAN RANDOM SELECTION

Number of Beacons	Percentage
$N = 30$	23%
$N = 50$	30%
$N = 100$	50%

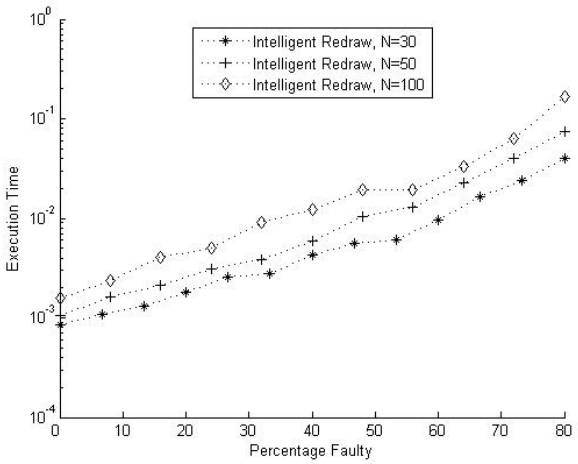

Fig. 4. The Intelligent Selection algorithm with differing N values and the percentage of faulty beacons varied from 0–80%.

The Intelligent Redraw algorithm is more sensitive to the byzantine case. Intelligent Redraw is designed to take advantage of the fact that it is easier to draw two correct beacons simultaneously than three. By generating a pair of points for each combination of considered beacons, Intelligent Redraw implicitly leverages the fact that intersections involving any accidentally faulty beacons do not result in estimated positions. If this is no longer the case it is clear the Intelligent Redraw is combinatorial with respect to each additional beacon considered. Thus Intelligent Redraw performs poorly in the byzantine case (see fig. 7) and is always dominated by the Random Selection algorithm.

D. Conclusions

The performance of triangulation in the presence of faulty and malicious beacons depends directly on the percentage of incorrect beacons along with the number of beacons. The Random Selection algorithm presented scales linearly with the number of beacons and exponentially in the percentage of faulty or byzantine beacons. The Random Selection algorithm outperforms the Intelligent Redraw algorithm in cases where there are low percentage of faulty beacons, as well as the byzantine case. The Intelligent Redraw algorithm is more resilient to higher percentages of faulty beacons and executes more quickly in these cases than the Random Selection algorithm.

VI. ACKNOWLEDGMENTS

The authors wish to acknowledge funding provided under grants CCR 0311616 and CRR 0325716 by the NSF. The

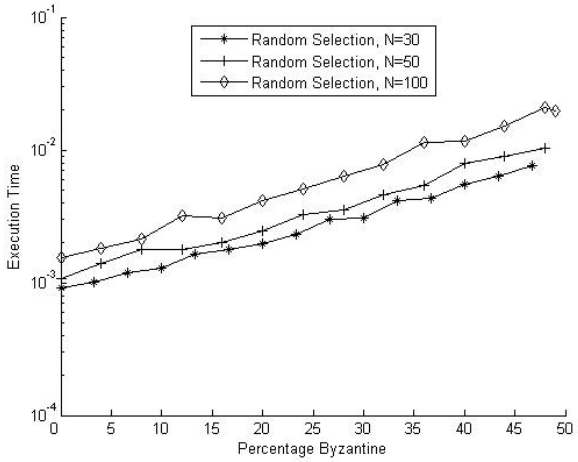

Fig. 5. The Random Selection algorithm with differing N values and the percentage of byzantine beacons varied from 0–50%.

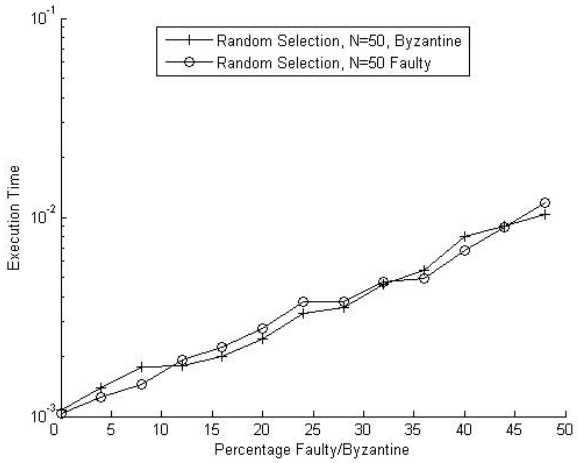

Fig. 6. The Random Selection algorithm with $N = 30$ plotted for both byzantine and faulty beacons from 0–50%.

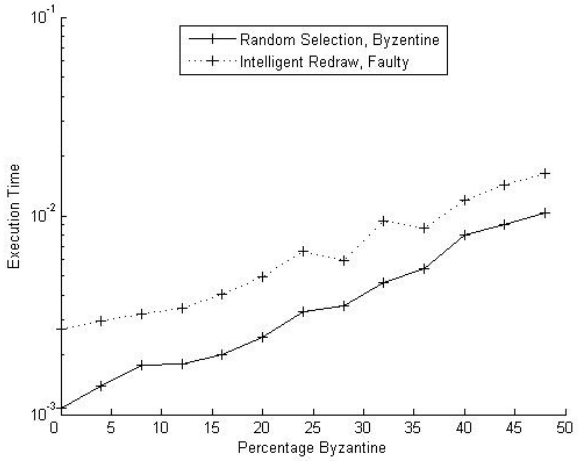

Fig. 7. The Intelligent Redraw and Random Selection algorithm's execution time compared for the byzantine case (0–50%).

authors wish to thank Dr. Helen Gill for her support. The authors would also like to thank the Boeing Corporation for their support under the grant Boeing ITI RPS #15, and Dr. Jae Kim.

REFERENCES

[1] Y. Cui and S. S. Ge, "Autonomous Vehicle Positioning With GPS in Urban Canyon Environments," *IEEE Transactions on Robotics and Automation*, vol. 19, pp. 15–28, 2003.

[2] J. Dittmer, "Solving the GPS Urban Canyon Problem," *Frost & Sullivan Market Insight*, 2005.

[3] A. P. A. Alcocer, P. Oliveira, "Underwater Acoustic Positioning Systems Based on Buoys with GPS," *Proceedings of the Eighth European Conference on Underwater Acoustics*, 2006.

[4] N. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket Location-Support System," in *Proceedings of ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2000, pp. 32–43.

[5] A. Savvides, C. Han, and M. Strivastava, "Dynamic fine-grained localization in ad-hoc networks of sensors," in *Proceedings of AC International Conference on Mobile Computing and Networking (MobiCom)*, 2001, pp. 166–179.

[6] D. Niculescu and B. Nath, "Ad hoc positioning system (APS) using AoA," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, 2003, pp. 1734 – 1743.

[7] Z. Li, W. Trappe, Y. Zhang, and B. Nath, "Robust statistical methods for securing wireless localization in sensor networks." in *Proceedings of The International Symposium on Information Processing in Sensor Networks (IPSN)*, 2005, pp. 91–98.

[8] P. Rousseeuw and A. Leroy, *Robust Regression & Outlier Detection*. New York, NY: John Wiley & Sons, 1987.

[9] D. Liu, P. Ning, and W. Du, "Attack-resistant location estimation in sensor networks," in *Proceedings of International Symposium on Information Processing in Sensor Networks (IPSN)*, 2005, pp. 99–106.

[10] N. Kiyavash and F. Koushanfar, "Anti-Collusion Position Estimation in Wireless Sensor Networks," in *IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, Pisa, Italy, 2007, pp. 1–9.

[11] T. Roosta, M. Meingast, and S. Sastry, "Distributed reputation system for tracking applications in sensor networks," in *International Workshop on Advances in Sensor Networks (IWASN)*, 2006.

[12] M. Manzo, T. Roosta, and S. Sastry, "Time synchronization attacks in sensor networks," in *Proceedings of the ACM workshop on Security of ad hoc and sensor networks (SASN)*, 2005, pp. 107–116.

[13] M. Barborak, A. Dahbura, and M. Malek, "The consensus problem in fault-tolerant computing," *ACM Comput. Surv.*, vol. 25, no. 2, pp. 171–220, 1993.

[14] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, "Byzantine fault tolerance, from theory to reality," in *Computer Safety, Reliability, and Security, Lecture Notes in Computer Science*, 2003, pp. 235–248.

[15] M. J. Fischer and N. A. Lynch, "A lower bound for the time to assure interactive consistency," *Information Processing Letters*, vol. 14, pp. 183–186, 1982.

[16] R. Turpin and B. A. Coan, "Extending binary Byzantine Agreement to multivalued Byzantine Agreement," vol. 18, no. 2, pp. 73–76, Feb. 1984.

[17] R. S. L. Lamport and M. Pease, "The byzantine generals problem." *ACM Trans. Prog. Lang. Sys.*, vol. 4, pp. 382–401, 1982.

[18] E. Valovage, "Enhanced ADS-B Research," *IEEE A&E Systems Magazine*, pp. 35–39, 2006.

[19] J. L. Awange and E. W. Grafarend, "Algebraic Solution of GPS Pseudo Ranging," *GPS Solutions*, vol. 5, pp. 20–32, 2002.

[20] B. W. Parkinson and S. W. Gilbert, " NAVSTAR: Global positioning systemTen years later," *Proceedings of the IEEE*.

[21] E. W. Weisstein, "Circle-Circle Intersection," *From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Circle-CircleIntersection.html*, 2008.

[22] P. Bourke, "Intersection of two circles," website: http://local.wasp.uwa.edu.au/ pbourke/geometry/2circle/, 1997.