The Verifier Bee: a Path Planner for Drone-Based Secure Location Verification

Pericle Perazzo*, Kanishka Ariyapala†, Mauro Conti‡, Gianluca Dini*
* Department of Information Engineering, University of Pisa, Italy. Email: {name.surname}@iet.unipi.it
† Department of Mathematics and Informatics, University of Florence, Italy. Email: {name.surname}@math.unifi.it
‡ Department of Mathematics, University of Padua, Italy. Email: {name.surname}@math.unipd.it

***Abstract*—Many dependable systems rely implicitly on the integrity of the positions of their components. For example, let us consider a sensor network for pollution monitoring: it is sufficient that a hostile actor physically moves some sensors to completely disrupt the monitoring. In such scenarios, a key question is: how to securely verify the positions of devices? To answer this question, researchers proposed several solutions. However, these generally require several fixed stations (anchors) with trusted positions.**

In this paper, we explore the possibility to use the emerging drone technology in order to overcome the limitation of using several fixed anchors. In particular, our approach is to replace all the fixed anchors with a single drone that flies through a sequence of waypoints. At each waypoint, the drone "acts like" an anchor and securely verifies the positions of the devices. The main challenge here is to find a convenient path for the drone to do this. The problem presents novel aspects, thus existing path planning algorithms cannot be used. We present VerifierBee: a path planning algorithm that allows a drone to perform a secure location verification of a set of devices. VerifierBee finds a good approximation of the shortest path, and at the same time it respects a set of requirements about drone controllability, localization precision, and communication range.

I. Introduction

The dependability of many distributed systems relies (often implicitly) on knowing the positions of the component devices. If the system believes that a device is in a position different from the real one, then it could infer wrong information and take wrong decisions. An adversary capable of changing the position of one or more devices (*displacement attack*) can deeply affect the system behavior with little effort. As an example, let us consider a sensor network deployed for pollution monitoring. The sensors could measure the density of dioxin in the air at different positions and report it to a centralized gateway, which eventually decides whether it should raise an alarm. An adversary willing to mask a pollution event could simply move some sensors in different positions, in a way such that it will avoid the detection (as illustrated in Figure 1). This attack is simple to carry out and difficult to detect.

Periodically measuring the positions of the devices is not enough to guarantee security. In fact, the majority of the positioning methods are vulnerable to attacks in which an adversary falsifies the position measurement [19], [11]. For example, if the position is inferred from the strength of a received beacon message, the adversary can confuse the measurement by sending fake beacons from wrong positions. Authenticating the beacons does not solve the issue, since the adversary could listen and replay authenticated beacons from different positions (*wormhole attack* [11]). Providing secure measurements of positions has shown to be a non-trivial problem [19], [20], [18]. A promising approach is *verifiable multilateration* [19]. Verifiable multilateration is a secure positioning technique that determines a position by measuring the distances from (at least) three anchors by means of *distance bounding protocols* [2]. A distance bounding protocol is a cryptographic protocol able to measure a secure upper bound to the distance between two devices. Verifiable multilateration has the drawback that it requires many fixed anchors. The number of necessary anchors grows roughly linearly with the size of the area in which the nodes are deployed [19]. Another problem is that the fixed anchors must be truly "fixed". Otherwise an adversary could simply move an anchor to jeopardize the security of the system.

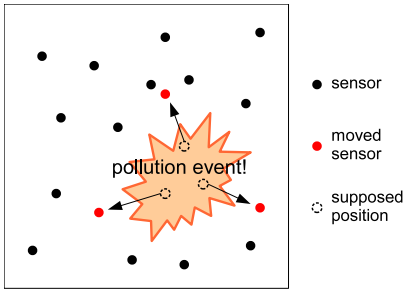

Fig. 1. Displacement attack against an environment monitoring system. A hostile actor changes the positions of some sensors, and then she can pollute without being detected.

In this paper, we explore the possibility of using the emerging *drone technology* to solve these issues. Drones, or Unmanned Aerial Vehicles (UAV), are aircraft with no human pilot. They can enjoy different levels of autonomy [14]: ranging from being remotely piloted to being completely autonomous in movements and decisions. In practice, our idea is to replace many fixed anchors with a single mobile drone. The drone follows a path that passes though a series of waypoints. At each waypoint, the drone "acts like" an anchor by executing a distance bounding protocol with a node. At the end of the path, each node has been measured from three

978-1-4799-8461-9/15/$31.00

different waypoints, and the position can be securely computed by means of verifiable multilateration. We thus completely eliminate the need for many expensive fixed anchors.

The problem is how to determine a convenient path for the drone. We cannot use existing path planning algorithms, because a valid path for verifiable multilateration must respect additional geometric constraints. In particular, the triangle formed by the waypoints must contain the node, otherwise the computed position will not be secure. Furthermore, other specific issues must be addressed, like the imprecision on the control of the drone movements.

Contribution

- We explore the approach of using drones to securely verify a set of positions by means of verifiable multilateration.
- We formally state the Traveller Location Verifier Problem (TLVP), that regards finding the shortest path for a drone to securely verify a set of nodes.
- We propose VerifierBee, a path planning algorithm that finds an approximate solution to TLVP.
- We run a thorough set of experimental evaluation of VerifierBee. The results of our experiments show that VerifierBee improves the path length of some 50% with respect to a simple solution.

Organization The rest of the paper is organized as follows. Section II compares relevant related work. Section III introduces the basic concepts. Section IV introduces the idea of drone-based verifiable multilateration. Section V formalizes the Traveller Location Verifier Problem. Section VI introduces VerifierBee. Section VII reports the results of our experimental evaluation. Finally, Section VIII concludes the paper.

II. Related Work

Secure positioning aims at measuring the position of a device in the presence of an adversary that wants to falsify such a measurement. Researchers proposed many methods [13], offering different levels of security (provable or only statistical), and defending against different adversaries (external or internal). Čapkun and Hubaux [19] proposed a secure positioning method called *verifiable multilateration*. In this proposal, the system measures the distances from a set of trusted anchors by means of distance bounding protocols. The position is computed by means of multilateration, and it is considered secure if it lies inside the polygon formed by the anchors. In this paper, we approach the problem of performing verifiable multilateration not with many fixed anchors, but with a single mobile drone.

Čapkun et al. [18] proposed a drone-based approach for secure location verification. In their system, the adversary is a malicious node that lies about its position. The drone sends a challenge message to the nodes, and then moves to a different random position. After an agreed period of time, the nodes respond with a response message, by which the drone infers their positions. Assuming that the malicious node ignores the drone's new position, it cannot falsify its own position in a coherent manner. Our approach is radically different, because it is based on verifiable multilateration, which is provably secure. As a consequence, we do not need to suppose that the drone's position is unknown by the adversary.

A similar problem to ours is drone-based data gathering from sensors [1], [7], [10], [16]. In this case, a robot (either aerial, terrestrial, or under-water) must collect data from a set of sparse and unconnected sensors. These papers propose path planning algorithms that solve generalized forms of the Traveller Salesman Problem (TSP). The objective is usually to minimize the path length while respecting particular constraints. In this paper, we propose a path planning algorithm to securely verify the positions of a set of nodes. This problem can be viewed as a generalization of the TSP as well. However, our problem is radically different from data gathering, because the path must respect a completely different set of constraints. For example, we must range each node from three distinct waypoints, whereas a single waypoint per node is sufficient for data gathering.

Another problem related to ours is drone-based (insecure) localization of ground devices [3], [4], [5], [17]. All these works do not have security in mind, and their position measurements cannot be considered trusted in a hostile environment. One of the simplest approaches is the one given by Corke et al. [4], in which a robot sweeps the entire area and periodically broadcasts its GPS position. The nodes collect such messages, called *position broadcasts*. They finally infer their own position by averaging all the received position broadcasts. Such a method is not secure, since an adversary could simply send fake position broadcasts, in such a way to confuse the nodes. Authenticating the position broadcasts does not solve the issue, since an adversary could listen to a legitimate broadcast and replay it on different positions. This is commonly known in the literature as the *wormhole attack* [11]. In general, all the localization method based on the strength of received messages like [3], [17] are poorly secure, since an adversary has an easy play on falsifying this information.

Dang et al. [5] uses a set of drones to localize a set of ground nodes, by measuring the relative distances between them. Our system measures the relative distances too, but it assures the security of the computed positions by using verifiable multilateration. Verifiable multilateration poses special requirements on the path that the drone has to follow. In particular, we must range each node from three waypoints, and the triangle formed by them must contain the node. This requirement was not present in classic trilateration.

III. Preliminaries

A *distance bounding protocol* [2] is a cryptographic protocol able to measure a distance between two devices, in such a way that an adversary cannot fake the measurement to be shorter than real (*reduction attack*). A distance bounding protocol determines a distance by precisely measuring the round-trip time between a challenge and a response message. The messages convey numeric quantities which are unpredictable by the adversary. The adversary cannot reduce the round-trip

time measurement, because she should guess and transmit in advance the messages.

A simple example of distance bounding protocol is the following:

M1: $A \longrightarrow B : a$
M2: $B \longrightarrow A : b$
M3: $B \longrightarrow A : \mathrm{sign}_k(A, B, a, b)$,

where a and b are random numbers unpredictable by the adversary, and k is a shared secret, by which B authenticates the protocol execution. M1 and M2 are the challenge and the response messages. To precisely measure the round-trip time, the challenge and the response are usually transmitted by means of impulse-radio ultra-wideband (IR-UWB) PHY protocols [12], resulting in a precision of centimeters. From now on, we will say "A ranges B" as a shorthand for "A executes a distance bounding protocol with B."

Verifiable multilateration [19] is a method for the secure measurement of positions which leverages distance bounding. In verifiable multilateration, the position of a *node* is determined by measuring the distances between the node and at least three *anchors* whose positions are known (Figure 2).

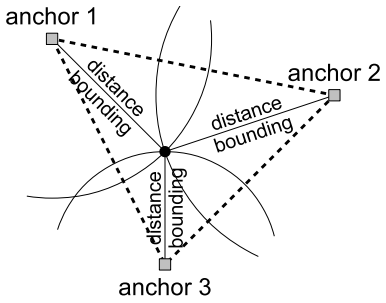

Fig. 2. Verifiable multilateration. The computed position is accepted only if it lies inside the verifiable triangle (the thick dashed one).

The distance measurements are performed by means of distance bounding protocols. The node's position is computed by trilateration. Such a position is accepted only if it lies inside the triangle formed by the anchors (*verifiable triangle*). Otherwise, it is discarded as untrusted. In fact, if an adversary wants to fake a position inside the verifiable triangle, then she must perform a reduction attack against at least one distance bounding protocol, which is infeasible. Note that the coverage of verifiable multilateration is only the verifiable triangle, because the outside positions are discarded.

IV. Drone-Based Verifiable Multilateration

Verifiable multilateration is able to securely measure the position of a node, but it needs a large number of anchors. In case of a set of nodes sparsely deployed on a large area and with a short communication range, it is necessary to deploy many anchors to reach them all. In addition, the coverage is restricted to the verifiable triangle, so it is not enough that three anchors are within the communication range: they have to "surround" the node in order to locate it securely. The scalability challenges of verifiable multilateration have been studied in [19]. In particular, the work in [19] proposed to place the anchors following a regular triangles' grid, in order to minimize the anchors necessary to cover a given area. Although a regular anchor placement improves the scalability, the number of anchors cannot scale better than linearly with the size of the area to cover.

With the development of the drone technology [14] and their increased availability on the markets, it became affordable to replace many fixed anchors with a single drone. The drone follows a *path*, touching a sequence of *waypoints*. At each waypoint it acts as an anchor (Figure 3), performing one or more distance bounding protocols with the nodes on the ground.

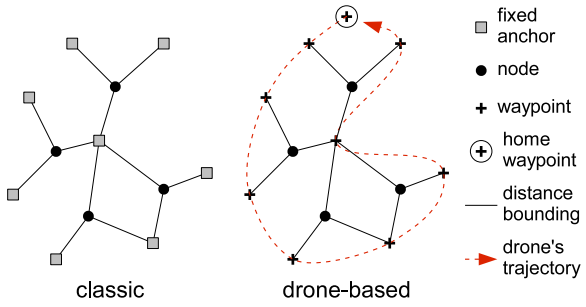

Fig. 3. Classic vs. drone-based verifiable multilateration.

The drone shares a different secret k with each node, by which the distance bounding protocols are executed. The mechanism just described solves the scalability issue. The problem becomes now how to find a convenient path for the drone in order to securely measure a set of positions in an efficient way. We assume to have an a-priori knowledge of the nodes' positions, from which we compute the drone's path. We call them *supposed positions*. The supposed positions are not trusted, because an adversary could have displaced the nodes. Therefore we want to securely verify them by means of verifiable multilateration. If the positions determined by verifiable multilateration are not consistent with the supposed ones, a node displacement attack is detected.

The drone's path is centrally computed in an off-line fashion, and it must respect the following requirements.

- The path has to start and terminate at a special fixed waypoint, called *home waypoint* (this is where the path is computed and loaded in the drone). The drone takes off from the home waypoint, performs the mission, lands at the home waypoint again, and communicates the outcome of the position verification to some base station.
- Each node has to be ranged from three distinct waypoints, and the verifiable triangle formed by them must contain the node. This is required for the localization to be secure.
- The drone has a limited communication range. Far away nodes could be impossible to reach with a distance bounding protocol.
- The position determined by verifiable multilateration must be sufficiently precise in order to be useful for the verification. Such a precision depends (also) on the

relative positions of the drone and the node. For example, if the drone ranges the node from exactly above, the precision will be extremely low. We have to avoid this.
- The path has to be tolerant to sources of imprecision. First, the movements of the drone are not perfectly controllable, because the wind strongly affects them. It is a good practice to provide for some tolerance, since the "true" waypoints actually visited by the drone could be different from the planned ones. The altitude could not be perfectly controllable too. Secondly, the supposed positions of the nodes could be imprecise.
- The mission time should be as short as possible. This is preferable for saving time and drone's battery life.

For the sake of simplicity, we assume that the drone moves at a constant speed. Minimizing the mission time is thus equivalent to minimizing the path length. We assume there are no obstacles to the drone's movements (e.g., walls, buildings, trees). The problem of path planning for secure location verification in presence of obstacles is interesting as well, but it falls outside the scope of this paper.

Many drones are limited in the movements they can do. For example, they cannot perform sharp curves or sudden direction changes (the problem is more stringent for fixed-wing drones, less for quadcopters). In order to respect the dynamic constraints of the specific drone, a path must be successively translated into a *trajectory*. The trajectory generation is a well-studied problem [8], and it falls outside the scope of this paper.

We suppose that all the nodes are on the ground, while the drone flies at a non-negligible *altitude* (h). We imagine the waypoints to be on the ground. The drone "visits" a waypoint when its position is above the waypoint (apart from drone control errors). The verifiable triangle is considered to be on the ground too. We assume that the ground is flat enough to allow the drone to be always in the line-of-sight with the ranged node.

We also suppose that, even if the position and the altitude of the drone are not perfectly controllable, they are actually *measurable*. The drone employs a technology that allows it to always know its own position and altitude with *negligible error*. An example of such a precise technology is differential GPS, which is sometimes installed on drones [9]. When the drone performs a distance bounding protocol with a node, what is measured is the *slant distance* (s), which is the line-of-sight one. However, for the aim of localization, we are interested in the *ground distance* (d), which is the one projected on the ground. The system computes the ground distance by:

$$d = \sqrt{s^2 - h^2}. \tag{1}$$

Once the drone has completed its path, three ground distances have been collected for each node. So the system can determine the positions of the nodes by trilateration, and verify if they are consistent with the supposed ones.

V. TLVP Formalization

In this section, we formalize the *Traveller Location Verifier Problem* (TLVP). TLVP can be considered as a generalization of the classic Traveller Salesman Problem (TSP), in which the nodes must not be visited, but rather verified for their positions. TLVP regards finding the shortest path to securely verify a set of ground positions by means of drone-based verifiable multilateration.

The *supposed positions* are a set of points on the Cartesian plane $\{N_1, \ldots, N_n\}$ that must be securely verified. A *path* (P) is a couple of sequences: a sequence of *waypoints* $\{W_1, \ldots, W_m\}$, and a sequence of *ranged nodes sets* $\{Rng_1, \ldots, Rng_m\}$. Each waypoint is a point on the Cartesian plane. The drone visits the waypoints in the order specified by the sequence. At each waypoint, the drone performs a distance bounding protocol with every node in the corresponding ranged nodes set. The path is closed, in the sense that the drone goes again to W_1 at the end. The first waypoint coincides with the *home waypoint* (W_{home}). Note that a node could be ranged from three waypoints that are non-consecutive in the path. For example, the drone could range twice a node in two successive waypoints, then move to a completely different zone to range other nodes, and finally return in the neighborhood to perform the third distance bounding. At the end of the path, each node must have been ranged from three distinct waypoints, and the verifiable triangle must contain the node.

We assume that the drone control error is bounded. We call such a bound the *drone control precision* (γ_W):

$$\|W_j - W'_j\| \leq \gamma_W, \tag{2}$$

where W'_j is the actual position from which the drone performs the distance bounding. In addition, the altitude is not perfectly controllable, but it is supposed to be bounded above by a *maximum altitude* (h_{max}):

$$h \leq h_{max}. \tag{3}$$

Finally, we assume that the error on the supposed positions is bounded. We call such a bound the *supposed positions precision* (ε_N):

$$\|N_i - N'_i\| \leq \varepsilon_N, \tag{4}$$

where N'_i is the actual position of the node.

A. Formalization of the constraints

If the supposed positions are imprecise, then the node could lie outside the verifiable triangle, and the drone will fail in verifying its position. To avoid this, the verifiable triangle must contain the whole circle centered in N_i and with radius ε_N. However, this is not enough, since we have to be tolerant also to the drone control error. If the drone control is imprecise, then the verifiable triangle actually drawn by the drone (*real verifiable triangle*) could be different to the planned one. As a consequence, the node could lie outside the verifiable triangle again. To avoid this, it is sufficient that the verifiable triangle contains the whole circle centered in N_i and with radius $\varepsilon_N + \gamma_W$. We can prove this by considering the worst case, shown in Figure 4. The real waypoints are shifted (with respect to the planned ones) of γ_W on the direction orthogonal to the edge of the verifiable triangle. The real node's position is shifted (with

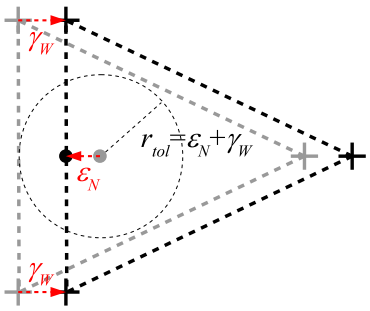

Fig. 4. Error tolerance in the worst case. The visited waypoints (black crosses) are shifted wrt the planned ones (gray crosses). The real node's position (black dot) is shifted wrt the supposed one (gray dot).

respect to the supposed one) of ε_N on the opposite direction. The real verifiable triangle must contain the node even in this case. By geometrical evidence, we obtain this if and only if the planned verifiable triangle contains the circle with center N_i and radius $\varepsilon_N + \gamma_W$. We call such a radius the *tolerance radius* (r_{tol}):

$$r_{tol} \triangleq \varepsilon_N + \gamma_W. \tag{5}$$

The error on the slant distance principally depends on the employed IR-UWB technology and the quality of the receivers. We suppose that the error on the slant distance is bounded and we call such a bound the *slant precision* (ε_s). In the IEEE 802.15.4a IR-UWB standard [12], the slant precision is usually of the order of centimeters. For example, the IR-UWB transceivers commercialized by DecaWave have a precision of 10cm [6]. The *ground precision* (ε_d) is the bound on the ground distance error. It is always worse than the slant precision. Especially if the drone is in plumb-line above the ranged node, a small error on the slant distance will translate into a huge error on the ground one. Figure 5 shows an evidence of this.

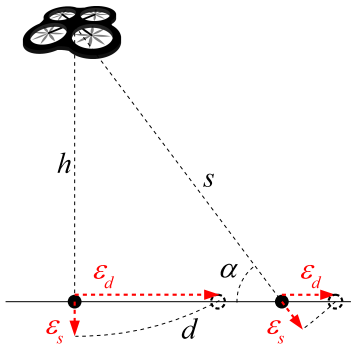

Fig. 5. Ground precision. If the drone is above the ranged node, a small imprecision on the slant distance (ε_s) will translate into a huge imprecision on the ground one (ε_d).

If the altitude and the ground distance are sufficiently large compared to the slant precision, then the following approximate relationship will hold:

$$\varepsilon_d \approx \varepsilon_s \cdot \frac{1}{\cos(\alpha)} = \varepsilon_s \cdot \sqrt{1 + (h/d)^2}, \tag{6}$$

where α is the angle of incidence of the slant distance to the ground (cfr. Figure 5). To guarantee a sufficiently precise localization, we impose an *objective ground precision* ($\bar{\varepsilon}_d$):

$$\varepsilon_d \leq \bar{\varepsilon}_d \Rightarrow \varepsilon_s \cdot \sqrt{1 + (h/d)^2} \leq \bar{\varepsilon}_d. \tag{7}$$

By expliciting d from (7) we get:

$$d \geq h \cdot \sqrt{((\bar{\varepsilon}_d/\varepsilon_s)^2 - 1)^{-1}}. \tag{8}$$

In other words, the ground distance must be large enough if we want to meet the objective precision. This depends on the altitude too: the more it is, the larger the ground distance must be. Also here, we have to take into account the worst case. We define a *minimal distance* (d_{min}) in this way:

$$d_{min} \triangleq h_{max} \cdot \sqrt{((\bar{\varepsilon}_d/\varepsilon_s)^2 - 1)^{-1}} + \varepsilon_N + \gamma_W. \tag{9}$$

If the ground distance between the planned waypoint and the supposed position is greater than or equal to the minimal distance, then we achieve the objective ground precision. In (9) we added ε_N and γ_W and we supposed the maximal altitude h_{max} to make sure that the requirement is respected even in the worst case.

Finally, the drone has a limited communication range. Given the maximum communication range (s_{max}), we define a *maximal distance* (d_{max}):

$$d_{max} \triangleq \sqrt{s_{max}^2 - h_{max}^2} - \varepsilon_N - \gamma_W. \tag{10}$$

If the ground distance between the planned waypoint and the supposed position is less than or equal to the maximal distance, then the node is within the communication range. In (10) we subtracted ε_N and γ_W to make sure that the requirement is respected even in the worst case.

To sum up, we identified three constraints: the tolerance radius (r_{tol}), which guarantees that the localization is secure; the minimal distance (d_{min}), which guarantees that the node is ranged with a satisfactory precision; the maximal distance (d_{max}), which guarantees that the node is within the communication range. These constraints refer to the single node, and they can be represented by three circles centered on the node's supposed position (Figure 6). The r_{tol}-circle must be contained inside the verifiable triangle, and the waypoints must lie at a distance between d_{min} and d_{max} from the node.

B. Final problem formulation

The Traveller Location Verifier Problem (TLVP) relates to the finding of the shortest path that allows a drone to securely verify the positions of a set of nodes under the r_{tol}, d_{min}, d_{max} constraints. Formally stated:

$$\begin{aligned} \underset{P}{\text{minimize}} \quad & \text{length}(P); \\ \text{subject to} \quad & \forall N_i : \\ & \text{triangle}(\text{wp}_P(N_i)) \supseteq \text{circle}(N_i, r_{tol}) \\ & \forall W_j \in \text{wp}_P(N_i)\ d_{min} \leq \|N_i - W_j\| \leq d_{max}, \end{aligned}$$

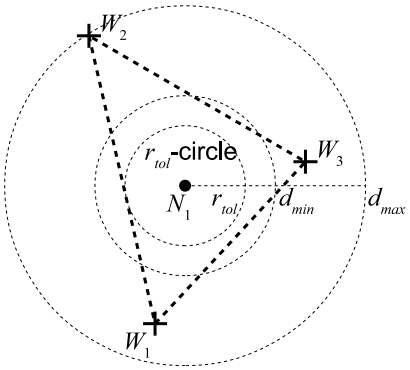

Fig. 6. TLVP constraints representation. The r_{tol}-circle must be contained inside the verifiable triangle, and the waypoints must lie at a distance between d_{min} and d_{max} from N_i.

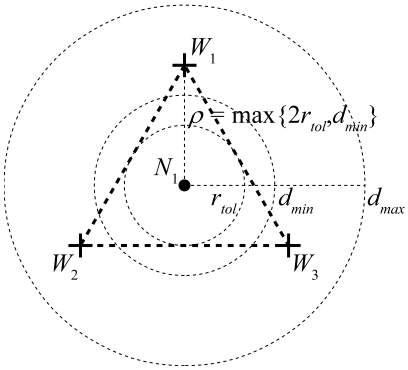

Fig. 7. Minimal verifiable triangle.

where $\mathrm{length}(P)$ indicates the length of the path P; $\mathrm{wp}_P(N_i)$ indicates the triplet of waypoints which range node N_i according to the path P; $\mathrm{triangle}(\cdot)$ indicates the triangle formed by a triplet of waypoints; $\mathrm{circle}(N_i, r_{tol})$ indicates the circle with center N_i and radius r_{tol}.

VI. VerifierBee Path Planner

TLVP is a generalization of the classic Traveller Salesman Problem (TSP), which is NP-hard. We present VerifierBee, an algorithm that finds an approximate solution to TLVP. VerifierBee uses a TSP solver algorithm as a building block to find a first valid solution. Then, such a solution is iteratively improved, following a greedy strategy. The TSP solver is used as a black box. It is required to find an approximate shortest path that visits all the points in a list, and then returns to the first point (closed path). It is not required to be optimal. Approximate TSP algorithms are acceptable as well. Of course the performances of the TSP solver will affect those of VerifierBee both in terms of optimality and processing time. VerifierBee operates in three phases: (i) basic path computation; (ii) greedy improvement; (iii) waypoint reordering.

A. Basic path computation

VerifierBee computes an ordered list of waypoints: the home waypoint plus three waypoints for each node, placed at fixed positions to form a *minimal verifiable triangle* (Figure 7). The minimal verifiable triangle is a regular triangle centered on N_i and inscribed to a circumference of radius $\rho = \max\{2r_{tol}, d_{min}\}$. This radius is the smallest one which respects both r_{tol} and d_{min} constraints. VerifierBee orients all the minimal verifiable triangles with a vertex toward north. The angular orientation is indifferent, because the successive greedy improvement phase will rotate and distort the triangles in order to find shorter paths. After having built the list of waypoints, we run the TSP solver on them to find an approximate optimal path that touches them all. The basic path is thus complete, and it is formed by $3n+1$ waypoints (where n is the number of nodes) and $3n+1$ ranged nodes sets. The first ranged nodes set is empty (it is the home waypoint), and the other ones contain a single node each. Figure 8 shows an example of basic path for 30 nodes.

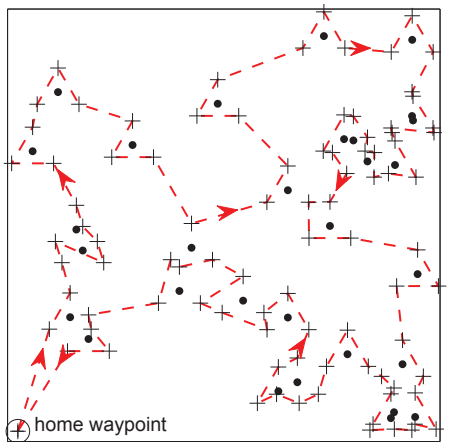

Fig. 8. Example of basic path with 30 nodes. The black dots are the nodes, the crosses are the waypoints, the red dashed line is the path.

Note that the drone passes very close to each node. This makes the path sub-optimal, since the drone does not use its full communication range. The basic path is a simple solution to TLVP. We will use it as a term of comparison to evaluate the performance of VerifierBee.

B. Greedy improvement

After having computed the basic path, VerifierBee changes it iteratively, following a greedy strategy. At each step, VerifierBee analyzes the possible changes (e.g., moving a waypoint in another position) and applies the most convenient one, that is the one that decreases more the total path length. The greedy improvement phase terminates when no change is possible or convenient anymore, meaning that we found a local minimum.

The changes are of two kinds: waypoint moving and waypoint pruning. *Waypoint moving* changes the position of a waypoint, while *waypoint pruning* removes a waypoint and "substitutes" it with another existing one. Both moving and pruning make use of the concept of *freedom space*. The freedom space of a waypoint is the area where the waypoint can be moved without violating any constraint of the problem

(all the other waypoints remaining fixed). It can be computed geometrically, as illustrated in Figure 9. The curved borders of the freedom space are the limits of the d_{min} and d_{max} constraints. The straight borders are the limits of the r_{tol} constraint.

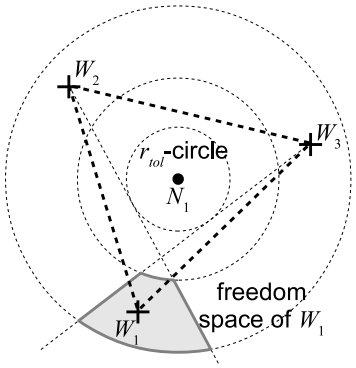

Fig. 9. Freedom space of W_1 (the gray area). The two straight borders lie on the two rays originating from the other waypoints of the verifiable triangle (W_2 and W_3) and tangent to the r_{tol}-circle.

Waypoint moving changes the position of a waypoint, in such a way to shorten the global path. Figure 10 shows an example.

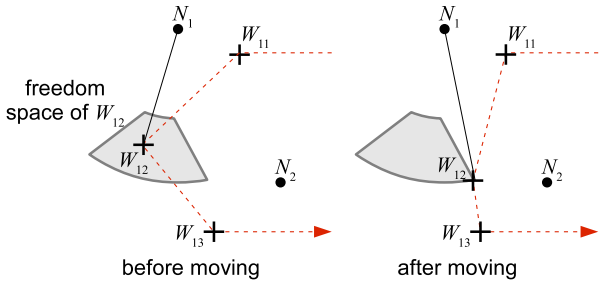

Fig. 10. Waypoint moving. W_{12} is moved so that the drone shortens the path going from W_{11} to W_{13}.

The best position where to move a waypoint is always: (i) somewhere on the border of the freedom space (like in Figure 10), or (ii) coincident with another waypoint in the interior of the freedom space. In the latter case, we do not apply waypoint moving but rather waypoint pruning (see below), that is we eliminate the waypoint and substitute it with the other one. Therefore, waypoint moving always moves a waypoint along the border of the freedom space.

Waypoint pruning removes a waypoint (*pruned waypoint*) and substitutes it with another existing one (*substitute waypoint*). The drone will not visit anymore the pruned waypoint. As a consequence, it will miss to run a distance bounding protocol. The missing distance bounding is run when the drone passes through the substitute waypoint. Waypoint pruning reduces the total number of waypoints. Figure 11 shows an example of waypoint pruning. The pruned waypoint W_i and the correspondent ranged nodes set Rng_i are eliminated from the path, while the nodes that were in Rng_i are added to the

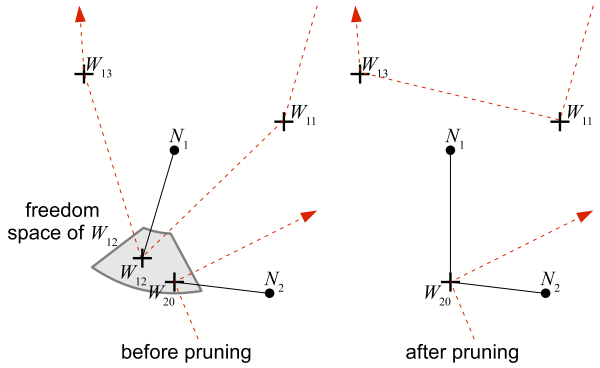

Fig. 11. Waypoint pruning. W_{12} is pruned and substituted by W_{20}. When the drone visits W_{20}, it runs two distance bounding protocols: one with N_1 and one with N_2.

ranged nodes set Rng_j of the substitute waypoint W_j. It is possible to prune a waypoint when its freedom space contains the substitute waypoint. The home waypoint cannot be pruned. After pruning, the substitute waypoint has to range two nodes instead of one. Consequently its freedom space will narrow, because it has to take into account the constraints relative to both nodes. The resulting freedom space is the intersection of the freedom spaces relative to the single nodes. In some cases, the freedom space of the waypoint to prune contains many waypoints. All these waypoints are suitable candidates to be the substitute waypoint. Which one to choose is indifferent in terms of path length. VerifierBee chooses the one which narrows less its freedom space, in such a way to leave more "freedom" to the next steps of the greedy improvement.

To sum up, the greedy improvement phase computes all the possible waypoint movings and prunings. If no change is further possible or convenient, the phase terminates. Otherwise we apply the most convenient change (either moving or pruning) and then we recompute all the possible changes again.

C. Waypoint reordering and complete algorithm

The greedy improvement may change the position and the number of the waypoints, but it does not change their order, which remains the same of the basic path. As a consequence, sometimes it is convenient to reorder the waypoints by running the TSP solver again. This is the third phase of the VerifierBee algorithm. The greedy improvement and the waypoint reordering phases are repeated, until the path length stops decreasing.

Algorithm 1 shows a pseudo-code description of VerifierBee. The function $\mathrm{SolveTSP}(\cdot)$ is our black-box TSP solver. The function takes a path, reorders the waypoints to form an (approximate) optimal path, and then returns such a new path. The function $\mathrm{GreedyImprove}(\cdot)$ takes a path, performs a greedy improvement, and returns the resulting path.

Figure 12 shows an example of VerifierBee path for 30 nodes (the same ones of Figure 8). This path is much shorter than the basic path of Figure 8. Many waypoints have been pruned and the other ones have been moved in more convenient positions. The path passes very close to the external nodes (N_1). On the contrary, interior nodes are ranged from

Algorithm 1: VerifierBee

Require: $\{N_i\}$, W_{home}, γ_W, ε_N, ε_s, ε_d, h_{max}, s_{max}

Determine r_{tol}, d_{min}, d_{max} by means of Eqs. 5, 9, 10
$Path \leftarrow$ a list of waypoints, one on W_{home},
  and the others on the minimal verifiable triangles.
$Path \leftarrow \text{SolveTSP}(Path)$ {basic mission}
loop
  $Path \leftarrow \text{GreedyImprove}(Path)$
  if $Path$ has not been improved **then**
    exit loop
  end if
  $Path \leftarrow \text{SolveTSP}(Path)$
  if $Path$ has not been improved **then**
    exit loop
  end if
end loop
return $Path$

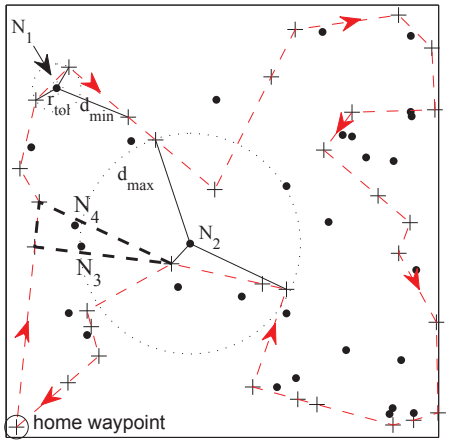

Fig. 12. Example of VerifierBee path.

far away (N_2). Note also that nodes close to each other (N_3 and N_4) are ranged by the same waypoints, and enclosed by the same verifiable triangle.

VII. Experimental Evaluation

We implemented VerifierBee with the Matlab programming language and tested its performance under different conditions. For the TSP solver, we employed an off-the-shelf algorithm available on Mathworks [15].

We assumed the following parameters for our experiments: a slant precision of $\varepsilon_s = 10cm$ (claimed by DecaWave for their IR-UWB transceivers [6]); an objective ground precision of $\bar{\varepsilon}_d = 25cm$; a communication range of $s_{max} = 300m$ (claimed by DecaWave for their IR-UWB transceivers [6]); a maximum altitude of $h_{max} = 160m$; a drone control precision of $\gamma_W = 10m$; a precision on the supposed positions of $\varepsilon_N = 5m$. The TLVP constraints stemming from these parameters are: $r_{tol} = 15.00m$, $d_{min} = 84.83m$, $d_{max} = 238.77m$. We simulated a random deployment of the nodes on a $1000m \times 1000m$ map, and we executed VerifierBee to find a path that securely verifies their positions. We put the home waypoint on the south-west angle of the map. Figure 13 shows the average path length for different numbers of nodes.

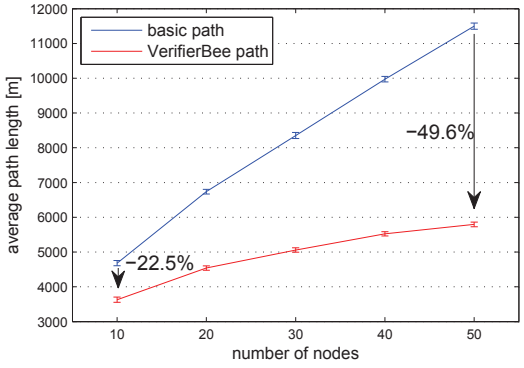

Fig. 13. Average path length wrt the number of nodes. Each point stems from 100 experiments. 95%-confidence intervals are displayed in error bars.

We can see that the basic path increases linearly, whereas the VerifierBee path shows a sub-linear trend. The length saving of VerifierBee with respect to the basic path is quite significant, especially in case of many nodes (-49.6% for 50 nodes). This is because the drone ranges more nodes from a single waypoint if the nodes are denser. Figure 14 shows the average processing time for the basic path computation and for the complete VerifierBee algorithm, running on a 2.4GHz Intel Core i5 processor.

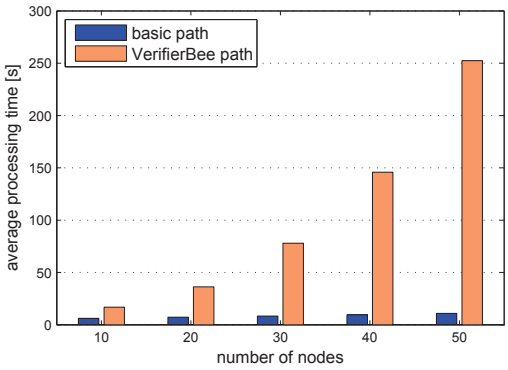

Fig. 14. Average processing time wrt the number of nodes for 100 experiments.

A path for 50 nodes takes roughly 4 minutes to be computed. This should be fully acceptable for an off-line computation. We remark that VerifierBee must be executed only once. The computed path remains valid as far as the positions of the nodes do not change. However, implementing the algorithm in C language (instead of Matlab language) should improve the performances even more.

We analyze now the influence of the parameters on the path length. Figure 15 shows the average path length for different values of the communication range with 20 nodes. As expected, the basic path is unable to leverage the full communication range. On the contrary, an improved communication range has a positive effect on the VerifierBee path

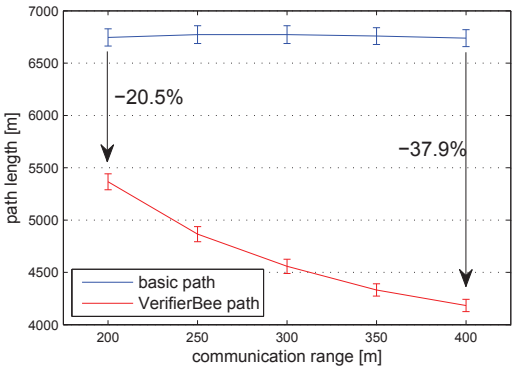

Fig. 15. Average path length wrt the communication range with 20 nodes. Each point stems from 100 experiments. 95%-confidence intervals are displayed in error bars.

(up to -37.9% of path length with a communication range of $400m$). Figure 16 shows the average path length for different drone control precisions with 20 nodes.

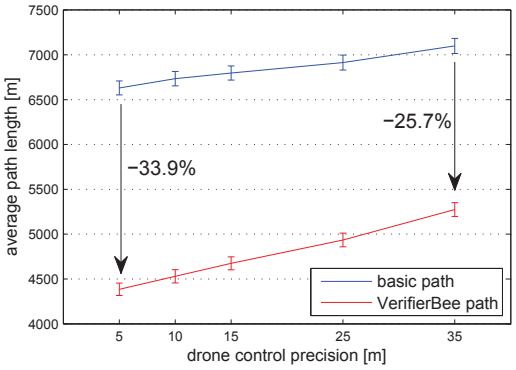

Fig. 16. Average path length wrt the drone control precision with 20 nodes. Each point stems from 100 experiments. 95%-confidence intervals are displayed in error bars.

The path length clearly increases as the imprecision grows. This is because the drone has to pass farther away from the nodes, to be sure to enclose them in the verifiable triangle even in the worst-case control error. VerifierBee saves a roughly constant length for each value of the drone control precision.

VIII. Conclusions

In this paper, we explored the approach of using drones to securely verify a set of positions. We formally stated the Traveller Location Verifier Problem (TLVP), that regards finding the shortest path for a drone to securely verify a set of nodes by means of verifiable multilateration. We proposed VerifierBee, a path planning algorithm that finds an approximate solution to TLVP. The results of our experiments showed that VerifierBee improves the path length of some 50% with respect to a simple solution.

Acknowledgment

This work has been supported by EU FP7 Integrated Project PLANET (grant no. FP7-257649), by Italian MIUR PRIN Project TENACE (no. 20103P34XC), and by the University of Padua PRAT 2014 Project on Mobile Malware. Mauro Conti is supported by a European Marie Curie Fellowship (no. PCIG11-GA-2012-321980).

References

[1] D. Bhadauria, O. Tekdas, and V. Isler. Robotic data mules for collecting data over sparse sensor fields. *J. of Field Robotics*, 28(3):388–404, 2011.

[2] S. Brands and D. Chaum. Distance bounding protocols. In *Advances in Cryptology–EUROCRYPT'93*, pages 344–359. Springer, 1993.

[3] F. Caballero, L. Merino, P. Gil, I. Maza, and A. Ollero. A probabilistic framework for entire WSN localization using a mobile robot. *Robotics and Autonomous Systems*, 56(10):798–806, 2008.

[4] P. Corke, R. Peterson, and D. Rus. Coordinating aerial robots and sensor networks for localization and navigation. In *Distributed Autonomous Robotic Systems 6*, pages 295–304. Springer, 2007.

[5] P. Dang, F. L. Lewis, and D. O. Popa. Dynamic localization of air-ground wireless sensor networks. In *MED'06*, pages 431–453. Springer, 2007.

[6] DecaWave. ScenSor SWM1000 Module. http://www.decawave.com/products/dwm1000-module.

[7] H. Ergezer and K. Leblebicioglu. Path planning for UAVs for maximum information collection. *IEEE Trans. on Aerospace and Electronic Systems*, 49(1):502–520, 2013.

[8] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *J. of Intelligent and Robotic Systems*, 57(1–4):65–100, 2010.

[9] G. Heredia, F. Caballero, I. Maza, L. Merino, A. Viguria, and A. Ollero. Multi-unmanned aerial vehicle (UAV) cooperative fault detection employing differential global positioning (DGPS), inertial and vision sensors. *Sensors*, 9(9):7566–7579, 2009.

[10] G. Hollinger, S. Choudhary, P. Qarabaqi, C. Murphy, U. Mitra, G. Sukhatme, M. Stojanovic, H. Singh, and F. Hover. Underwater data collection using robotic sensor networks. *IEEE J. on Selected Areas in Communications*, 30(5):899–911, 2012.

[11] Y.-C. Hu, A. Perrig, and D. B. Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In *INFOCOM'03*, volume 3, pages 1976–1986. IEEE, 2003.

[12] IEEE Computer Society. IEEE Std 802.15.4a-2007 (Amendment 1: Add Alternate PHYs), 2007.

[13] J. Jiang, G. Han, C. Zhu, Y. Dong, and N. Zhang. Secure localization in wireless sensor networks: a survey. *J. of Communications*, 6(6):460–470, 2011.

[14] F. Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *J. of Field Robotics*, 29(2):315–378, 2012.

[15] J. Kirk. Traveling Salesman Problem - Genetic Algorithm. http://www.mathworks.com/matlabcentral/fileexchange/13680-traveling-salesman-problem-genetic-algorithm.

[16] M. Ma, Y. Yang, and M. Zhao. Tour planning for mobile data-gathering mechanisms in wireless sensor networks. *IEEE Trans. on Vehicular Technology*, 62(4):1472–1483, 2013.

[17] P. N. Pathirana, N. Bulusu, A. V. Savkin, and S. Jha. Node localization using mobile robots in delay-tolerant sensor networks. *IEEE Trans. on Mobile Computing*, 4(3):285–296, 2005.

[18] S. Čapkun, K. Bonne Rasmussen, M. Cagalj, and M. Srivastava. Secure location verification with hidden and mobile base stations. *IEEE Trans. on Mobile Computing*, 7(4):470–483, 2008.

[19] S. Čapkun and J.-P. Hubaux. Secure positioning in wireless networks. *IEEE J. on Selected Areas in Communications*, 24(2):221–232, 2006.

[20] Y. Zhang, W. Liu, Y. Fang, and D. Wu. Secure localization and authentication in ultra-wideband sensor networks. *IEEE J. on Selected Areas in Communications*, 24(4):829–835, 2006.