# Intercepting a Stealthy Network

Mai Ben Adar Bessos
Bar-Ilan University, Israel
Email: mai.bessos@gmail.com

Amir Herzberg
Bar-Ilan University, Israel
Email: amir.herzberg@gmail.com

***Abstract*—We investigate a new threat: *networks of stealthy routers (S-Routers)*, communicating across a restricted area. The 'classical' approach of transmission-detection by triangulation fails, since S-Routers use short-range, low-energy communication, detectable only by nearby devices.**

**We investigate algorithms to intercept S-Routers, using one or more mobile devices, called Interceptors. Given a source of communication, Interceptors find the destination, by intercepting packet-relaying by S-Routers along the path. We evaluate the algorithms analytically and experimentally (simulations), including against a parametric, optimized S-Routers algorithm.**

**Our main result is a (centralized) Interceptors algorithm bounding the outcome to $O\left(N\log^2(N)\right)$, where $N$ is the number of S-Routers. We later improve the bound to $O\left(N\log(N)\log(\log(N))\right)$, for the case where the transmission schedule of the S-Routers is continuous.**

## I. Introduction

Interception of unauthorized wireless transmitters has been studied and deployed since the World War I. There is extensive research on interception-avoidance, mostly focusing on low-energy, stealthy transmissions [1]–[6]. However, low-energy implies also limited range and low bandwidth, limiting the use of these techniques. This motivated other directions; one alternative is covert channels over overt networks [1], [4]; however, this only applies where transmitters can use such overt network. Another alternative are advanced hardware designs such as spread-spectrum transmission [7] and directed antennas [8], however, these involve high costs - and are still detectable, using advanced intercepting hardware.

However, we are on the verge of a revolution in this area, due to the proliferation of miniature, low-cost networked devices, including IoT devices, micro-robots, insect-cyborgs and drones. Examples for the research on tiny mobile autonomous vehicles include [9]–[11], and examples for research on computer-controlled tiny insect-cyborgs include [12], [13].

These tiny, low-energy devices [14] can be used as S-Routers, combined into *stealthy communication networks*, which facilitate interception-avoiding communication across restricted areas. The reduced costs, mobility and small footprint of these devices, would facilitate extensive use including commercial and private privacy-intrusive applications. Example applications may include outdoor or indoor eavesdropping/surveillance [15] [16], industrial espionage, and a command-and-control channel for physical attacks, including for terrorism. Stealthy networks may also be used to facilitate communication between sensors across insecure domain, e.g., environment-protecting sensors used to detect and localize illegal rainforest logging [17]. These trends are likely to also result in further increase in the already extensive use of stealthy communication mechanisms, as well as of interception mechanisms used as countermeasure, in military applications, including intelligence, terrorism and counter-terrorism.

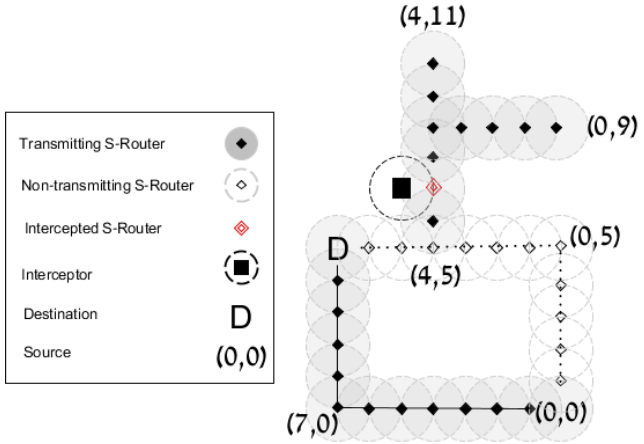


Fig. 1. Illustration of one time-step, with a single Interceptor searching for destination $D$, located at $(7,5)$ (in 2D, i.e., $\mathbb{R}^2$). Two routes of S-Routers connect the source $(0,0)$ to $D$ (at $(7,5)$); the first (via $(7,0)$) transmits at this time-step (black diamonds), the other not (white diamonds). Several other S-Routers (from $(4,6)$ to $(4,11)$ and to $(0,9)$ ) transmit 'dummy' messages, to divert the Interceptor from the real routes, and lead it to two 'dead ends': $(4,11)$ and $(0,9)$.

In spite of their low-energy, stealthy transmission, S-Routers may still be intercepted and located - by an Interceptor which is located in their close proximity. A mobile Interceptor may be able to intercept S-Routers distributed in a large area, by visiting multiple locations; such search could be done more effectively, by using multiple Interceptors in parallel. However, such search could consume excessive time and resources, when the area is large.

This work presents the first research of this problem. We present, analyze and experimentally evaluate algorithms for ensuring efficient search by Interceptors, to intercept and locate S-Routers. To study the problem analytically, we introduce a model for evaluating algorithms for stealthy routing, as well as algorithms for their interception. The model is a turn-based process between two parties: S-Routers and Interceptors. Interceptors provide an algorithm for exposing a communication network hidden in two-dimensional space by operating an Interceptor, while S-Routers dictate which parts of the route transmit at each turn.

We focus on the scenario illustrated in Fig. 1, where the S-Routers route information to a special S-Router called the *destination (D)*; the goal of the Interceptors is to find $D$. We assume a single, known source of communication, which, WLOG, we fix at the center of coordinates, i.e., $(0,0)$. There are $N$ S-Routers, one of which is the destination $D$; the other S-Routers relay information from the source $(0,0)$ to $D$. S-Routers may transmit up to one data unit at each turn. An Interceptor exposes an S-Router if the S-Router transmits within the interception range of the Interceptor.

Algorithms are measured by their impact on the *outcome*, which is the number of total data units that are transmitted before the destination is exposed by Interceptors; S-Routers attempt to maximize the outcome, while Interceptors attempt to minimize it. Note that in the particular scenario we study, Interceptors do not 'disable' S-Routers, even if exposed. Other scenarios are possible, and supported by our model; however, this particular scenario is interesting, since disabling S-Routers before finding $D$ may trigger defensive and evasive reactions by S-Routers.

Contributions.

- We introduce the threat of stealthy networks and the algorithmic problem of efficient interception, with a model facilitating analysis of algorithms.
- We study two approaches for Interceptors: network graph search and spatial search, and explore advantages and disadvantages of each approach.
- We present and analyze Divide And Conquer Algorithm: a (centralized) spatial search algorithm for Interceptors which ensures that the expected outcome is in $O(N\log^2(N))$.
- We present and analyze D&Q$_{\text{CTA}}$, a simple modification to Divide And Conquer Algorithm, which reduces the expected outcome to $\mathcal{O}(N\log(N)\log(\log(N)))$, when the communication from the source $(0,0)$ to $D$ is continuous.
- We introduce a parametric algorithm for S-Routers, and use a genetic algorithm to optimize these parameters. We then use the resulting (optimized) S-Routers algorithm, to experimentally evaluate the Interceptors algorithms presented in this work, via simulations.

### A. Related work

We are not aware of previous works studying stealthy networks or their interception. However, there are several highly-related areas, and we now briefly discuss some of the most relevant areas, and few specific works.

The performance of an Interceptor to sense transmissions may be limited if the transmitter utilizes Low-Probability-of-Interception (LPI) techniques [18], [8]. However, in the scenario we consider, the Interceptors are mobile; when an Interceptor reaches a point between the communicating S-Routers, detection is significantly harder to prevent. Additionally, for smaller devices, the benefits of these techniques are limited, since the frequency spectrum and reception area depend primarily on the antenna's size. For example, a study [7] on the 802.15.4 standard shows that obfuscation is possible with the available spectrum through frequency hopping. However, the obfuscation gain may not prevent a nearby Interceptor, with the appropriate equipment, from detecting the transmission.

In the last few years several studies examined the use of ad-hoc covert networks (ACNs) [4] [1] as a method for extending the distance between endpoints of wireless covert channels. ACNs allow multi-hop communication through an overt network, thus reducing the power output required for transmitting data. However, such an overt network is often not available.

Few other works utilize transmission detection mechanisms in adversarial environments, assuming detection of an adversarial agent (e.g., Interceptor or S-Router) and taking defensive actions such as avoiding or approaching and eliminating. For example, an intrusion-detection system may alert the user upon detection of drone-controlling activity [19]. Another example are strategies for coping with jamming attacks against mobile ad-hoc networks, e.g., in [20], guiding mobile agents away from a detected jammer. Some work examines the effectiveness of transmission detection when searching for eavesdropping devices e.g. [21] [22] using mobile agents.

The model we study shares some aspects with Security games [23] and several Search games [24], in particular *Network-interdiction* [25]. Network-interdiction models represent scenarios where one player selects a route between a start and destination points, and its adversary attempts to predict it by affecting the network and increase the probability of detection in certain points. However, unlike the model we study in this paper, the network is known to the searcher, and it only allocates resources in advance.

Few studies examine network structures that are resilient to cascading damage, such as the discovery of a hidden terrorist cell [26] [27] [28]; this is similar to the impact of intercepting an S-Router, or some S-Routers, in our model. However, the existing results focuses on minimization of the number of hops between nodes, which is not directly applicable to our problem.

## II. The 2D Stealthy Network Model

In this section, we present a general model for studying stealthy network problems in the two-dimensional plane $\mathbb{R}^2$, and then, in subsection II-B, we make specific choices relevant to the destination-search problem which we focus on. We tried to keep the model general, as we believe that this may allow it to be used to study different security problems involving stealthy networks. On the other hand, we avoided generalizations where we felt it would introduce significant complexity which is not required by the specific problem and algorithms investigated in this work.

### A. General 2D Stealthy Network Model

The model studies executions of centralized algorithms for two sets of agents: the *Interceptors* set, $\Pi_I$, and the *S-Routers* set, $\Pi_S$. The model operates in consecutive, discrete

time-steps $t \in \mathbb{N}$; at each step, it invokes each of the two algorithms, on output from previous step, and then invokes a third algorithm, $\Pi_E$, which models the *environment*. We believe that the modeling of the environment by $\Pi_E$ gives significant flexibility to the model.

The initial input to all three algorithms is the size of both sets, i.e., the number of Interceptors, denoted M, and the number of S-Routers, denoted $N$.

The environment algorithm $\Pi_E$ determines the results of the actions of the Interceptors and S-Routers, including the inputs for next step, as well as the outcome of the process upon its termination. The modeled process is, intuitively, reminiscent of a zero-sum game, with outcome being the 'cost' for the Interceptors- and the 'reward' for the S-Routers.

*a) Interceptors algorithm $\Pi_I$:* The Interceptors algorithm $\Pi_I$ receives as input the results of the interceptions/observations by all of the Interceptors in the previous step, given by an input for each of the Interceptors. The algorithm $\Pi_I$ outputs $M$ outputs, one for each Interceptor. The algorithm also maintains a (centralized) state. Namely, in general: $\Pi_I : [\{0,1\}^*]^M \times \{0,1\}^* \rightarrow [\{0,1\}^*]^M \times \{0,1\}^*$. Note that in the specific scenario we study, the output for each Interceptor represnts the location for that Interceptor in the next time step, i.e., it is a binary encoding of a point in the plane (in $\mathbb{R}^2$); see below.

*b) S-Routers algorithm $\Pi_S$:* In general, the S-Routers' algorithm has similar inputs and outputs to the Interceptors' algorithm $\Pi_I$, namely:
$\Pi_S : [\{0,1\}^*]^N \times \{0,1\}^* \rightarrow [\{0,1\}^*]^N \times \{0,1\}^*$. However, in the specific scenario we focus on, $\Pi_S$ is much simplified; see below.

*c) Environment algorithm $\Pi_E$:* $\Pi_E$ models the behavior of the environment, as a (probabilistic) algorithm, allowing analysis of different stealthy-network scenarios and and goals. The inputs to $\Pi_E$ are the outputs of both $\Pi_I$ and $\Pi_S$. The outputs are the observations/interceptions to be provided, in the next step, to the S-Routers and Interceptors algorithms. The environment also has an arbitrary state as input and output; the initial value of the state can be used as a source of randomness to the execution, if desired. Finally, the environment has one more output field, which is used to signal the termination of the execution as well as the outcome of the process, upon termination. The outcome is given as an integer, and the special symbol $\perp$ marks that the execution is not completed yet. Namely, $\Pi_E : [\{0,1\}^*]^M \times [\{0,1\}^*]^N \times \{0,1\}^* \rightarrow [\{0,1\}^*]^M \times [\{0,1\}^*]^N \times \{0,1\}^* \times \{\mathbb{Z}^2 \cup \{\perp\}\}$.

### B. Focusing on destination-search scenario

In this subsection, we present specific choices for the general model, which allow us to focus on the specific problem and scenario on which we focus in this work, and in particular, on the task of finding the destination of the S-Routers. We begin with by presenting the somewhat-simplified model for the Interceptors and the S-Routers.

*Interceptors algorithm $\Pi_I$.* The inputs to each Interceptor is a list of points from which transmissions were detected at the previous step (up to one point per Interceptor). Also, the output for each Interceptor is an encoding of the location, in $\mathbb{R}^2$, for the corresponding Interceptor. We abuse notation and use $\mathbb{R}^2$, rather than the binary encoding, as the inputs and outputs, i.e. we define $\Pi_I$ as:

$$\Pi_I : \{\mathbb{R}^2\}^* \times \{0,1\}^* \rightarrow (\mathbb{R}^2)^M \times \{0,1\}^*$$

*S-Routers algorithm $\Pi_S$, and its $S_L(i)$, $S_T(t)$ representation.* In this work S-Routers do not move, although allowing them to move would be a natural extension. Therefore, we find it convenient to specify the algorithm of the S-Routers by a pair of functions, the *location* $S_L(i)$ of each S-Router $i \in \{0, ...N-1\}$, and the *transmission schedule* $S_T(t) \subset \{0, ..., N-1\}$, identifying the S-Routers that transmit at time $t \in \mathbb{N}$. We omit the simple mapping from $S_L$, $S_T$ to the algorithm $\Pi_S$ implementing them.

We next define the notions of connectivity and connected points.

**Definition 1** (Connectivity)**.** *Two points $p_1, p_2 \in \mathbb{R}^2$ are* connected *iff their Euclidean distance is at most one, i.e., $||p_1 - p_2|| \leq 1$. A list of points is connected if every pair of two consecutive points is connected. Two points $p_1, p_2 \in \mathbb{R}^2$ are connected via a set of points $P$ if there is a list of points in $P$, say $(l_1, ...l_k)|(\forall i)(l_i \in P)$, s.t. the list $(p_1, l_1, \ldots, l_k, p_2)$ is connected.*

Finally, we define the specific environment $\Pi_E$ used in this work.

*a) Initialization:* Upon initialization, $\Pi_S$ determines the (fixed) locations of all S-Routers, as points in $\mathbb{R}^2$, including the destination , denoted as $\boldsymbol{D} \in \mathbb{R}^2$. One exception is the source, which is always at $(0,0)$. The set of S-Routers must ensure connectivity from source $(0,0)$ to destination .

*b) Termination and outcome:* As in the general model, $\Pi_E$ determines the termination of the execution, as well as its outcome. Upon termination, $\Pi_E$ also determines its outcome. The value of outcome represents the success/reward of the S-Routers, i.e. the value of the information they were able to deliver to $D$ before it was intercepted. The value of outcome is also the cost/penalty for the Interceptors, whose goal is to minimize the value of information delivered to $D$ before interception.

The outcome is the number of time-steps $t \in \mathbb{N}$ during which there is a set $P$ of S-Routers, s.t. the source $(0,0)$ and $D$ are connected via $P$, and all S-Routers in the list transmit at time $t$, i.e., $\{s_i\}_{i=1}^k \subseteq S_T(t)$. Let $outcome(t) = 1$ if such set $P$ exists for time $t$, and 0 otherwise; outcome is then simply the sum over $outcome(t)$ for all time steps in the execution.

The normal case is for $\Pi_E$ to terminate the execution when one of the Interceptors is (directly) connected to the destination $D$; intuitively, this means that the destination was exposed. Alternatively, $\Pi_E$ terminates the execution if 'requested' by $\Pi_S$, i.e., the S-Routers. In particular, the S-Routers may terminate upon collecting sufficient data, i.e., sufficiently high outcome; this is required to let the S-Routers terminate if the Interceptors never reach $D$. The environment ($\Pi_E$) will also

terminate the execution if either party acts in a way forbidden for that execution.

Note that rewarding the S-Routers only for steps in which there is a path of transmitting S-Routers from $(0,0)$ to $D$, implies that S-Routers can not buffer messages and deliver them later. Future work may remove this restriction, to allow for delay-tolerant networking by S-Routers. One reason for this (simplifying) restriction, is that each of our 'time-steps' represent a (potential) physical movement by the Interceptors, and movements are normally much slower than communication.

As presented so far, the model allows time-steps $t$ in which there is no transmission-path from source $(0,0)$ to $D$ (i.e., $outcome(t) = 0$). However, it seems that in many scenarios, Interceptors will transmit continuously to $D$. We refer to this as the *continuous transmission constraint/assumption*. In Section V we show that this allows more efficient Interceptors algorithm.

**Definition 2** (Continuous Transmission Constraint)**.** *We say that the Continuous transmission constraint holds for an execution, if for every time step $t$ in the execution, hold $outcome(t) = 1$, i.e., $(0,0)$ and $D$ are connected via a set of S-Routers who transmit at time $t$. We say that $\Pi_E$* Enforces Continuous Transmission *if $\Pi_E$ terminates the execution, with $outcome = 0$, upon a time step in which the constraint does not hold.*

## III. Introducing Interceptors algorithms

We found that the design of efficient Interceptors algorithm is more challenging than appears initially, with resulting algorithm being somewhat counter-intuitive. Obviously, we cannot repeat here all the variations we experimented with; however, we present few basic algorithms, which we believe will help the reader understand the problem better, preparing the ground for the more efficient - but less intuitive - algorithms presented in the following sections.

The section contains four subsections. In the first subsection, we observe that the Interceptors may limit their search to a bounded area, specifically, a *disc*. We then present an algorithm which essentially searches this disc. In the second and third subsections we present two naive attempts to find $D$ by 'following the path' from the source $(0,0)$ to $D$. Finally, we consider the use of more advanced graph-search algorithms, and discuss some of the challenges of this (very natural) direction. To our disappointment, we did not yet find an efficient graph-search algorithm, that works in the general case. Therefore, our efficient algorithms, presented in the following sections, use a different approach. However, it is possible that future work would find better ways to use the graph-search approach, at least for 'typical case'; some graph-search variants we evaluated had good experimental results, see Section VI.

### A. Naive Disc Search Algorithm

We begin with a very simple algorithm that we call *naive disc search*, presented in Fig. 2. Basically, the naive disc search algorithm exhaustively searches for $D$ in a bounded disc. We first show that it suffices to search for $D$ within a disc, specifically, the disc of radius $N$ whose center is the source $(0,0)$; this simple bound is also used by the more advanced algorithms. First notations, then the observation bounding the disc.

**Notation:** $Disc$**.** Given a point $c \in \mathbb{R}^2$ and a distance $r \in \mathbb{R}$, let $\boldsymbol{Disc}(r,c) = \{p \in \mathbb{R}^2 | \ ||p-c|| \leq r\}$ denote the region of a disc whose center is $c$ and whose radius is $r$.

**Lemma 1.** *Assume $D$ is connected to the source $(0,0)$ via the locations of the S-Routers, namely $\{S_L(i)\}_{i\in\{0,\ldots,N-1\}}$. Then $D \in Disc(N,(0,0))$.*

*Proof.* In Appendix A □

Since it suffices to search for $D$ within $Disc(N,(0,0))$, a simple, naive approach is to exhaustively search this disc. More precisely, such algorithm will visit different points within the disc, where each point results in covering a disc of radius 1 centered in that point, until the entire disc was covered - or $D$ found.

The order of visitations may affect the performance of the algorithm. For example, if all S-Routers are located 'densely' around $(0,0)$, as illustrated in Fig. 4(a), Interceptors may sort all points by (increasing) distance from $(0,0)$ then visit them in that order in order to find $D$ efficiently in $O(N)$ time steps. However, if the search is deterministic and known in advance, $D$ may be placed so it is found only by the very last searched points. For example, if Interceptors keep visiting points with increasing distance from $(0,0)$ but S-Routers are located as illustrated in Fig. 4(b), roughly the entire disc $Disc(\frac{N}{2},(0,0))$ will be covered before $D$ is found. Hence, a random order is slightly preferable for Interceptors.

**Input from $\Pi_E$:** $A$: result of latest search attempts,
$A_S$ algorithm state - previously visited points,
$N, M$: global constants /*The environment provides $A_S$ as it was returned from the previous call to the algorithm (or uninitialized, if first call) */

1: **if** $A_S$ is uninitialized **then**
2: $A_S \leftarrow \{\}$
3: **end if**
4: $S \overset{R}{\leftarrow} (DiscCoverage(N) \setminus A_S)^M$ s.t. elements are distinct
5: $A_S \leftarrow A_S \cup S$
6: **return** $(S, A_S)$

Fig. 2. Naive disc search algorithm. The algorithm searches for $D$ within $Disc(N,(0,0))$ exhaustively. At each call, additional $M$ new points are selected for the next visitation. The search disregards the previously intercepted transmissions that are specified in $A$. The point list $A_S$ is the algorithm state, and accumulates in each call all visited points.

The main challenge of such algorithms is the choice of the set of points to search, i.e., the covering of a disc of radius $N$ by discs of radius 1. Let $\boldsymbol{DiscCoverage(N)}$ denote the set of points that cover $Disc(N,(0,0))$. Minimizing the size of $DiscCoverage(N)$ is difficult, see Theorem 1 in [29], but its

complexity is necessarily $O(N^2)$, and simple implementations for $DiscCoverage(N)$ can achieve it. Efficient algorithms to find $\boldsymbol{DiscCoverage(N)}$ are discussed in [30]. Note that in these implementations, each point is connected to $O(1)$ other points in the coverage. The algorithm is defined in Fig. 2; essentially, the Interceptors search for $D$ by visiting every point in $DiscCoverage(N)$ in random order. At each invocation, the algorithm expects an input of all previously visited points, and outputs a list of $M$ unvisited points to visit next.

**Proposition 1.** *The expected outcome of the Naive Disc Search algorithm is in $O(N^2/M)$.*

*Proof.* In Appendix A □

### B. Naive Graph Search Algorithm

The naive disc search algorithm (Fig. 2) does not use the interceptions (detections), which seems wasteful. Surely, we can use interceptions to find $D$ more efficiently. One natural idea is to exploit the fact that $D$ must receive transmissions for the source $(0,0)$; we can try to 'follow' these transmissions, by always searching in the vicinity of one of the points where we intercepted a transmission, plus the source $(0,0)$. This algorithm is defined in Fig. 3; it keeps a set $A_S$ of locations from which a transmission was intercepted (initialized to $\{\}$), then visits at each time step a point that is connected to one of the points in $A_S$, chosen at random with uniform probability.

**Input from $\Pi_E$:** $A$: result of latest search attempts,
$A_S$ alg. state-accumulated successful search locations,
$N, M$: global constants
1: **if** $A_S$ is uninitialized **then**
2: $A_S \leftarrow \{\}$
3: **end if**
4: $A_S \leftarrow A_S \cup A$
5: $S \overset{R}{\leftarrow} (DiscCoverage(N) \cap \{(x,y) | \exists c \in A_S \cup \{(0,0)\} : Connected((x,y),c)\})^M$ s.t. elements are distinct
6: **return** $(S, A_S)$

Fig. 3. Naive Graph Search Algorithm. $A_S$ is initialized with the source $(0,0)$), and the algorithm visits points that are connected to $A_S$ in an attempt to extend the exposed part of network

**Proposition 2.** *The expected outcome of the Naive Graph Search algorithm is in $O(N^2/M)$.*

In Appendix A.

### C. Uniform Graph Search Algorithm

Since the Naive Graph Search algorithm selects points from $A_S$ with uniform probability at each step, points that were added earlier to $A_S$ have more opportunities for being selected. Intuitively, if newly discovered interceptions will be visited more frequently, the performance of the algorithm may be significantly improved. In order to examine this approach, we have designed the *Uniform Graph Search* algorithm. The algorithm assumes that only a single Interceptor is available, i.e., $M = 1$. The algorithm is defined similarly to the Naive Graph Search Algorithm, with the following modifications:

1) For each point $p$ in $DiscCoverage(N)$, initialize a counter: $\rho(p) = 0$
2) Each time a point $p$ is visited by the algorithm, increase $\rho(p)$ by 1
3) When selecting a point to visit (from points that are connected to $A_S$), select points with minimal $\rho$ value.

Namely, all points that the algorithm may visit will be (roughly) visited an equal number of times, and for each point added to $A_S$, the new point and its neighbors will be visited repeatedly, until their associated $\rho$ value is no longer minimal. For example, if S-Routers use the network illustrated in Fig. 4(b), and the continuous transmission constraint, as defined in Definition 2, holds, then the Uniform Graph Search will frequently intercept new S-Routers (and eventually $D$) at the 'front' of the few routes, since data is transmitted through at least one of the routes at each step, and the algorithm will repeatedly visit points near the 'front' after each interception. Note that this scenario is handled inefficiently by the Naive Disc Search and the Naive Graph Search algorithms.

Unfortunately, in the worst case, the performance of this algorithm is not significantly better (compared to the naive algorithm). Even if the continuous transmission constraint holds, if the network graph includes many separate alternate routes that connect $(0,0)$ and $D$, the transmission rate in each route may be reduced proportionally (as illustrated in Fig. 4(c)), and interception of new S-Routers will be infrequent.

**Proposition 3.** *The expected outcome of the Uniform Graph Search is in $\Omega(N^2/log(N))$.*

*Proof.* In Appendix A. □

In the following sections we present Interceptors algorithms and prove their expected performance is significantly better (assymptocially) compared to the Uniform Graph Search algorithm. However, for many cases, where S-Routers is small enough, the Uniform Graph Search algorithm outperforms all other algorithms, as illustrated in Fig. 10. A more detailed performance comparison is given in section VI.

### D. Graph Search Solution Attempts

In order to avoid exhaustive search for $D$, Interceptors have to consider previously successful search attempts. Intuitively, a graph-searching algorithm may be used for traveling along the path of connected S-Routers, similarly to the naive algorithms presented in the previous sections. However, since the network structure is unknown and S-Routers that do not lead to $D$ may deliberately expose themselves, improving the performance the Interceptors is difficult. Fig. 4 illustrates several problematic cases. In Fig. 4(d), the route may have several splitting points, and in most cases only one leads to $D$. Since paths that lead to $D$ will not necessarily have a different transmission schedule from other paths, they are hard to identify. Consequentially,

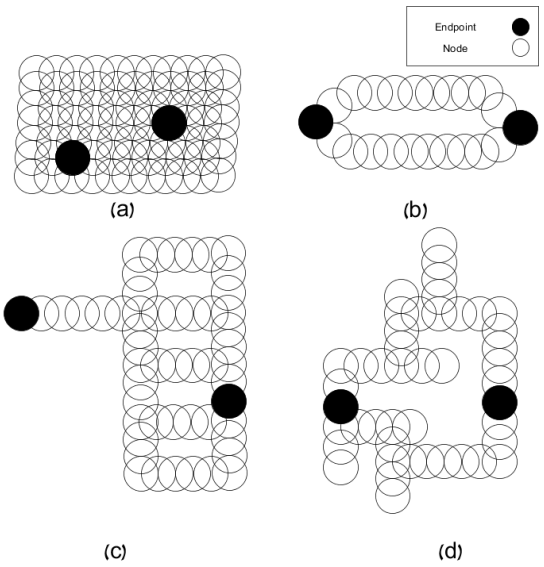


Fig. 4. Examples of different S-Router networks: (a) All S-Routers are located 'densely' around the source $(0,0)$. An exhaustive search around $(0,0)$ may reach $D$ effiifently. (b) Only few separate 'long' alternate routes connect $(0,0)$ and $D$. If Continuous Transmission Assumption holds, the rate of transmission in at least one of these routes is relatively high, which allows the Uniform Graph Search algorithm to expose S-Routers efficiently. (c) A network which uses numerous separate alternate routes. The transmission rates in different points may vary significantly; in particular, this prevents Uniform Graph Search algorithm from exposing new S-Routers efficiently even if the Continuous Transmission Assumption holds. (d) A network with few paths but numerous 'dead ends'. Even if a graph-search algorithm can efficiently cope with routes which transmit slowly, it is difficult to discern such routes from actual 'dead ends'. Since most 'walks' in the network lead to a 'dead end' and S-Routers may make 'dead end' routes appear exactly like other routes, it is also difficult to avoid them

each time the Interceptor reaches a 'dead end', it must advance through another point in the path without knowing which one.

In Fig. 4(c), each S-Router may transmit data at different rate, according to the number of alternate routes to $D$. If Interceptors attempt to travel through a path and successfully advance at some rate, they have no way to discern whether S-Routers transmit data at a lower rate in that path specifically, at all paths that connect to $D$ or whether no data was transmitted through network at all. Moreover, a combination of the problems illustrated in (c) and (d) is possible as well, where the desired path has a significantly different rate from the path that led to it, and the path also includes numerous 'dead ends' with similar transmission schedule. In this case, the Interceptor may not select a route randomly and 'persist', since with a high probability the path is a 'dead end'. However, attempting to travel through another route after a period of mis-interceptions may also be detrimental if the route was not a 'dead end'.

## IV. Divide and Conquer Interceptors Algorithm

In this section we present and analyze an algorithm for the Interceptors, the *Divide And Conquer Algorithm*, which bounds the outcome to $O(N\log^2(N))$ for a single Interceptor, or $O(N\log^2(N)/M)$ for $M$ Interceptors. Counter-intuitively, and in contrast to the less efficient Graph Search algorithms of the previous section, this algorithm does *not* try to 'search' the graph of S-Routers from $(0,0)$ to $D$. Instead, this algorithm takes a 'divide and conquer' method, to find the destination $D$ 'directly' - without exposing the entire path to it.

We begin, in the following subsection, with few definitions, notations and observations/lemmas, which are important to understand the design and the analysis presented in the subsequent subsections. In most of the section, we focus on the single Interceptor case; the extension to $M$ Interceptors is straightforward, as we present in the last subsection.

### A. Preliminaries: Ranges and Walls

We begin this section with few additional topological concepts which are used throughout this paper.

First, given a location $l \in \mathbb{R}^2$, let $Range(l)$ denote its *range*, i.e., the set of points whose communication would be intercepted by an Interceptor located at location $l$. Formally, $\boldsymbol{Range}(l) = \{x \in \mathbb{R}^2 | Connected(l,x)\}$. The range notation extends to a set of points $L$, namely we denote $Range(L) = \bigcup_{l\in L} Range(l)$.

As previously mentioned, the Divide And Conquer Algorithm searches for $D$ directly. To do so, we use the fact that $D$ must be within $Disc(N,(0,0))$, as shown in Lemma 1. The algorithm partitions $Disc(N,(0,0))$ into smaller regions, and then examines these regions by visiting points on their boundaries. If the boundaries 'separate' between $(0,0)$ and $D$, and considering the S-Routers transmit from $(0,0)$ to $D$, it follows that these transmissions must 'cross' one or more of the boundaries which are visited by the algorithm. We show that if the points are sufficiently-close, then the algorithm may intercept transmissions from at least one of these points.

We now provide more details. We first define two topological notions which are important in this algorithm: a $\sqrt{3}$-spaced *wall* and *closed wall*.

**Definition 3** (Wall, closed wall, and In/Out regions)**.** *An $\sqrt{3}$-spaced wall is a list of points $L = \{l_1, l_2, \ldots, l_k\} \in \left(\mathbb{R}^2\right)^k$ such that the distance between every two consecutive points $l_i, l_{i+1}$ is at most $\sqrt{3}$. A $\sqrt{3}$-spaced closed wall $L$ (abbreviated to* closed wall*), is a wall where the distance between $l_1$ and $l_k$ is at most $\sqrt{3}$. We denote the outer region by $\boldsymbol{Out}(L)$, and the internal region, excluding $Range(L)$ itself, by $\boldsymbol{In}(L)$.*

In the definition above, to define the inner and outer region, we use basic topological notions such as boundary and region, which are quite intuitive and standard; precise definitions can be found, e.g., in [31].

We focus on $\sqrt{3}$-spaced walls and closed walls, since a $\sqrt{3}$-spaced closed wall separates between (points in) its *internal region*, $In(L)$, and (points in) its *outer region*, $Out(L)$. This is proved in the next Lemma, and illustrated in Fig. 5.

Due to this focus, in the rest of this of the paper - including the following Lemma - we simply write *walls* and *closed walls*, always referring to $\sqrt{3}$-spaced walls and closed walls. It is easy to confirm that an $x$-spaced closed wall with $x > \sqrt{3}$

may fail to provide the separation property referred to in the Lemma.

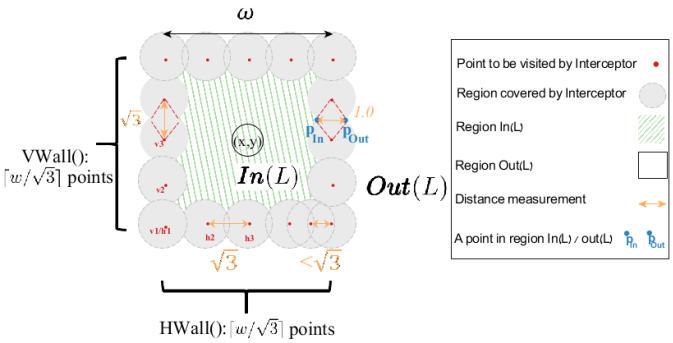


Fig. 5. The closed wall $L = \{v_1, v_2, \ldots\} \cup \{h_1, h_2, \ldots\} \cup \ldots$ separates the plane into the the inner and outer regions $In(L)$, $Out(L)$ and its range $Range(L)$. Note that all points in $L$ rest on the boundary of the same square. Any point from region $In(L)$ is not connected to any point in region $Out(L)$. This property is used by our algorithms: Interceptors inspect, randomly, the points in closed walls separating $D$ (inside) from the source $(0,0)$.

**Lemma 2.** *Given a ($\sqrt{3}$-spaced) closed wall $L$, no pair of points $p_{In} \in In(L)$ and $p_{Out} \in Out(L)$ are connected. Namely, for all $p_{In} \in In(L)$ and $p_{Out} \in Out(L)$ holds $||p_{In} - p_{Out}|| > 1$.*

*Proof.* In Appendix B □

The divide and conquer algorithm generates closed walls, then instructs the Interceptor to visit them in a random order. In order to calculate the probability of interception when visiting a point in a closed wall that 'separates' $(0,0)$ and $D$ (such as the closed wall illustrated in Fig. 5), we formally define 'separating' closed walls, then in Proposition 4 we prove that at least one S-Router may be sensed from one of the points of such closed walls.

**Definition 4** (separating closed walls)**.** *We use separating closed wall to refer to a closed wall that contains $D$ but excludes $(0,0)$, namely a closed wall $L$ for which $D \in In(L) \cup Range(L)$ and $(0,0) \in Out(L) \cup Range(L)$ hold.*

**Proposition 4.** *Let $L$ be a separating closed wall, and let $t \in \mathbb{N}$ be a time-step s.t. $outcome(t) = 1$. There exist $v \in L$ and $x \in S_T(t)$ s.t. $Connected(S_L(x), v)$.*

*Proof.* In Appendix B □

### B. Divide And Conquer Algorithm

We now present the Divide And Conquer Algorithm; the pseudo-code appears in Fig. 6. Note that the code uses several calls to functions which appear in section IV-C, for specific tasks such as determining specific 'watched' points that Interceptors visit. These 'watched' points are placed as a closed wall around *squares* containing $D$; we begin with very large squares and repeatedly divide them into smaller squares, until we find $D$. Let us first present our notation for a square.

*Notation.* (Square). Given a point $(x,y) \in \mathbb{R}^2$, and a length $w \in \mathbb{R}$, let $\textbf{\textit{Square}}(w,(x,y)) = [-\frac{w}{2}+x, \frac{w}{2}+x] \times [-\frac{w}{2}+y, \frac{w}{2}+y]$ denote the region of a square whose center is $(x,y)$ and each of its edges are of length $w$. For example, Fig. 5 gives a visualization of the closed wall $L$, where all points in $L$ rest on the boundary of the square $Square(w,(x,y))$.

The algorithm searches for $D$ in $Square(2N,(0,0))$; this contains $Disc(N,(0,0))$, and hence, from Lemma 1, $D$ must also be in this square. The algorithm partitions $Square(2N,(0,0))$ into smaller squares and places an Interceptor at one of several random points along $\sqrt{3}$-spaced walls on their boundaries. That is, a closed wall is kept per square, s.t. one of these 'watched' closed walls contains the destination $D$; we refer to these square-shared closed walls as *square wall*. If a square wall is also separating (according to Definition 4), we refer to it as *separating square wall*.

When a square wall shows signs of possibly containing $D$, namely when a transmission was intercepted from one of it's points, the algorithm further divides the corresponding square into four quarters, and repeats the process for these smaller squares, until finding $D$. For efficiency, the total size of walls of 'watched' squares should be small; to find $D$, the regions must contain it. The algorithm carefully ensures both properties.

It is crucial to randomize the location of the squares, to foil S-Router placements that exploit predictable locations of square walls. S-Routers may lead the algorithm into 'watching' many additional regions that do not contain $D$, due to S-Routers that deliberately expose themselves at specific locations. Hence, we first select a random point $o$ from $Square(2N,(0,0))$ - see line 1 of Fig. 7. From the beginning, we 'watch' the four $2N \times 2N$ squares shown in Fig. 8(a), s.t. $o$ is a shared corner. From Lemma 1, we can assume that $D$ is within one of these four squares.

At each step, we put the Interceptor at a random point in the walls of 'watched' squares. (Even if we have more Interceptors, we cannot populate all walls at each step.) We try to detect the transmissions by S-Routers crossing these walls, from source $(0,0)$ to $D$; this identifies 'suspect' squares, worthy of further decomposing into four sub-squares. From Lemma 1, it suffices, however, to put Interceptors at points which are within $Disc(N,(0,0))$; see our 'focus' on $Disc(N,(0,0))$ in Fig. 8(b) (this 'filtration' is omitted from Figs. 6, 7 and from analysis).

If we use only large squares, e.g., the four large, $2N \times 2N$ squares shown in Fig. 8(a), then it is quite possible that no path from $(0,0)$ to $D$ will cross their walls at all - since $D$ will be within the same large square. Indeed, in Fig. 8(a), we see that $D$ and $(0,0)$ are both in the lower-right large square (shown more closely in Fig. 8(b)). To ensure that each path of S-Routers from $(0,0)$ to $D$ will cross one of the 'watched walls', we divide, from the very beginning, each of the 'watched squares' containing $(0,0)$ into its four sub-squares, until reaching squares small enough to ensure localization of $D$; see the 'repeat loop' in Fig. 7. This is not *that* wasteful: the additional length of all these initial sub-

squares is less than the length of the initial $2N \times 2N$ squares.

Assuming $D$ is in $Disc(N,(0,0))$, at least one of the initial square walls includes $D$ and excludes $(0,0)$; this ensures Proposition 4 holds, and at least one point in one of the initial square walls will allow the Interceptor to detect a transmission. Points to search are selected at random with uniform probability from all square walls, such that the probability of selecting a point from any certain square wall is proportional to its size (line 17). For each successful search, on top of the square walls associated with that search point, four smaller square walls that encircle the same region are added (from line 4), in an attempt to decrease the size of smallest square wall that includes $D$ and excludes $(0,0)$. Sufficiently small squares are covered by a single visit.

### C. Building Square Walls

In this section we present 'utility functions' that Divide And Conquer Algorithm uses, to build *square walls* and *watch points* in them.

The closed walls are implemented as a list of points on the boundary of a square region, as illustrated in Fig. 5. The following utility functions are used in order to generate such closed walls and for dividing closed walls into smaller ones (example of this process may be found in Fig. 8(a)-(d)).

In order to generate a closed wall on the boundary of a square, we first generate a separate $\sqrt{3}$-spaced lists of points for each of the four square edges, then concatenate them. The following two notations $(VWall, HWall)$ will not be used in the pseudocode, and are only used for defining the points of $SquareWall()$ (defined below). Given a length $w \in \mathbb{R}$ and a starting point $(p_x, p_y) \in \mathbb{R}^2$ we denote:

- $VWall(w,(p_x,p_y)) = ((p_x,p_y),(p_x,p_y+\sqrt{3}),(p_x,p_y+2\cdot\sqrt{3}),\ldots,(p_x,p_y+\lfloor\frac{w}{\sqrt{3}}\rfloor\cdot\sqrt{3})),(p_x,p_y+w))$.
- $HWall(w,(p_x,p_y)) = ((p_x,p_y),(p_x+\sqrt{3},p_y),(p_x+2\cdot\sqrt{3},p_y),\ldots,(p_x+\lfloor\frac{w}{\sqrt{3}}\rfloor\cdot\sqrt{3},p_y),(p_x+w,p_y))$.

In Fig. 5, points generated by $VWall(w,(x,y))$ are listed as $v_1, v_2, v_3 \ldots$ and points generated by $HWall(w,(x,y))$ are listed as $h_1, h_2, h_3 \ldots$ . In all calls by Divide And Conquer Algorithm, $w$ is of the form $\frac{k}{2^i}$ where $i,k \in \mathbb{N}$. Therefore, $w$ is not divisible by $\sqrt{3}$ and the distance between the final two points in each list $((p_x,p_y+\lfloor\frac{w}{\sqrt{3}}\rfloor\cdot\sqrt{3}),(p_x,p_y+w)$ and $(p_x+\lfloor\frac{w}{\sqrt{3}}\rfloor\cdot\sqrt{3},p_y),(p_x+w,p_y))$ is less than $\sqrt{3}$. The sizes $|VWall(w,(p_x,p_y))|$,$|HWall(w,(p_x,p_y))|$ are simply $\lceil w/\sqrt{3}\rceil$.

We are now ready to generate a closed wall over the boundary of a square region. Given the length $w \in \mathbb{R}$ and point $(p_x,p_y) \in \mathbb{R}$, we use $\boldsymbol{SquareWall}(w,p_x,p_y)$ to denote the following concatenation of two $VWall()$ lists and two $HWall()$ lists: $HWall(w,(x-\frac{w}{2},y-\frac{w}{2}))$ || $VWall(w,(x+\frac{w}{2},y-\frac{w}{2}))$ || $HWall(w,(x-\frac{w}{2},y+\frac{w}{2}))$ || $VWall(w,(x-\frac{w}{2},y-\frac{w}{2}))$.

Note that $L = SquareWall(w,p_x,p_y)$ separates the square into $In(L)$ and $Out(L)$, as defined in Def. 3; see Fig. 5.

The closed wall $SquareWall(w,(p_x,p_y))$ lies on the boundary of its corresponding square $Square(w,(p_x,p_y))$. In order to divide the closed wall into four smaller closed walls, we first define their four corresponding smaller squares. The edge length of all smaller squares is $\frac{w}{2}$, but each square has a distinct center point. Let $d_x, d_y \in \{-1,1\}$ indicate which of the four centers we wish to find. The center is given by: $(x+d_x\frac{w}{4}, y+d_y\frac{w}{4})$. The list of all centers is denoted with $\boldsymbol{NextCenters}(w,(x,y)) = \bigcup_{d_x,d_y\in\{-1,1\}}(x+d_x\frac{w}{4}, y+d_y\frac{w}{4})$.

The Divide And Conquer Algorithm uses the variable $C$ to map points to search to the square walls they compose (see Figs. 6, 7). At each time step, the algorithm places Interceptor at one of the points from the domain of $C$, i.e. at one of the points that are being 'watched'. We use a standard notation for function domain $\boldsymbol{dom(C)}$ for referring to these points.

**Input:** $A$: result of latest search attempt,
$A_S$ algorithm state,
$N$: a global constant
/*For definitions of $NextCenters()$,$SquareWall()$ and $dom(C)$, see section IV-C*/
1: **if** $A_S$ is uninitialized **then**
2: $C \leftarrow DivideAndConquerInit(N)$ /*$C$ maps 'watched' points to their associated regions (a set of regions per point)*/
3: **end if**/*Test if interception occurred at $s$:*/
4: **for all** $s_l \in A$ **do**
5: /*Divide a square wall into quarters, unless the region is small enough to be covered entirely*/
6: **for all** point $\rightarrow$ region pairs $s_l \rightarrow (s_c, s_w)$ in $C[s_l]$ **do**
7: **if** $s_w > 2$ **then**
8: $G \leftarrow NextCenters(s_w, s_c)$
9: **for all** $g \in G$ , $p \in SquareWall(s_w/2, g)$ **do**
10: $C[s_l] \leftarrow C[s_l] \cup (p, s_w/2)$
11: **end for**
12: **else**
13: $C[s_l] \leftarrow C[s_l] \cup (s_l, 0)$ /*No additional divisions are needed. After an additional visit to the center of the square wall, its entire area will be covered.*/
14: **end if**
15: **end for**
16: **end for**
17: $S \xleftarrow{R} dom(C)$ /*select a point to visit from the list of 'watched' points, with uniform probability*/
18: $A_S \leftarrow C$ /*Update the algorithm's state*/
19: **return** $(S, A_S)$

Fig. 6. Pseudocode of DivideAndConquerAlgorithm. $C$ and $s$ are components of the algorithm's state. $s$ indicates the latest point visited by Interceptors, and $C$ maps points for the algorithm to visit, to their associated square regions (a region is represented by the length and center). Any of the points in $C$ ($dom(C)$), may be visited by Interceptors. These points compose square walls, and from initialization of $C$ (line 2) and later, at least one square wall separates $D$ and $(0,0)$. If visitiation results in interception, the square regions associated with the visited point are divided into smaller squares, and corresponding points are added to $C$. If the divided square wall contained $D$, the algorithm gets 'closer' to exposing $D$

**Input:** $N$: Number of S-Routers (including $D$)

1: $o \overset{R}{\leftarrow} Square(2N,(0,0))|(o[x] \neq 0, o[y] \neq 0)$ /*begin with random offset*/
2: $w \leftarrow 4N$
3: $C \leftarrow$ empty mapping
4: /*Populate $C$ with a list of square walls s.t. at least one necessarily separates $D$ and $(0,0)$:*/
5: **repeat**
6: $G = NextCenters(w,o)$ /*G lists centers for four sub-regions of $Square(w,o)$*/
7: **for all** $g \in G$, $p \in SquareWall(w/2,g)$ **do**
8: $C[g] \leftarrow C[g] \cup (p, w/2)$ /*Some square walls share points. Therefore, points in $C$ may be mapped into several regions*/
9: **end for**
10: $o \leftarrow g \in G : (0,0) \in Square(w/2,g)$ /*Finds center of the sub-region that includes $(0,0)$*/
11: $w \leftarrow w/2$
12: **until** $w \leq 1$
13: $C[o] \leftarrow C[o] \cup (o,0)$ /*If even the smallest square wall contains both $D$ and $(0,0)$, this ensures that at least one point in $C$ is within range of $D$ */
14: **return** $C$

Fig. 7. Pseudocode of DivideAndConquerInit(). Used to initialize DivideAndConquerAlgorithm() (see in Fig.6). The algorithm generates and returns $C$ - a mapping from points to search, to their corresponding regions. The points of $C$ compose square walls. After initialization is done at least of the square walls is a leading square wall i.e. there exists at least one square walls $L$ s.t. $D \in In(L) \cup Range(L))$ and $(0,0) \in Out(L) \cup Range(L)$

### D. Algorithm analysis

In this section we prove the following Theorem:

**Theorem 1.** *The expected outcome of Divide And Conquer Algorithm is $O(Nlog^2(N))$.*

The proof of the theorem, which we present at the end of this section, relies on two key observations. The first observation is an upper bound for how many times that the algorithm 'closes in', i.e., subdivides a leading square wall, until $D$ is found; this observation is stated and proven in Lemma 3. The second observation is an upper bound on the expected number of transmitted data units until the algorithm generates a new square wall which further 'closes in' on $D$; this observation appears within the proof of Theorem 1. The proofs of the lemmas in this section are in Appendix B.

Recall that at each time step, the algorithm places Interceptor at one of the points of $dom(C)$, where each point is part of at least one square wall. Most of these square walls do not contain $D$, and even if a transmission is intercepted while visiting one of their points, dividing their regions will not help in finding $D$. We next define *leading square walls*, which are the 'smallest' separating square wall; by dividing these walls, the algorithm 'zooms in' on $D$.

**Definition 5** (leading square walls)**.** *Let $\mathcal{L} = \{L_1, L_2, \ldots\}$, where $L_i \subseteq dom(C)$, be the set of all separating square walls in $dom(C)$. We refer to a separating square wall $L \in \mathcal{L}$ as a leading square wall if no other separating square wall is contained in $L$ i.e. $\forall L' \in \mathcal{L} : L' \not\subseteq (In(L) \cup Range(L))$.*

If the algorithm divides the region that corresponds to a leading square wall into quarters, and, as a result, additional square walls with their respective points are incorporated into $C$, then a subset of the newly added square walls will be the new leading square wall. As we state in Lemma 3, after this repeats for several times, $D$ will be found.

In Fig. 8 and in some of the proofs for the lemmas in this section, we use the notion of *depth* to separate square walls of different sizes. Formally, given a square wall $L = SquareWall(w,(x,y))$ for some $x, y$, we say that $L$ is *of depth* $\boldsymbol{d_w} = log_2(\frac{N}{w}) + 1$. $d_w$ indicates the number of repeated divisions needed for dividing the initial square wall of size $2N \times 2N$ into squares of size $w \times w$. For example, Fig. 8(a) includes square walls of depths $0, 1, 2, 3$.

**Lemma 3.** *The algorithm finds $D$ after the algorithm divides a leading square wall (DivideAndConquerAlgorithm(), line 4) at most $\lceil log_2(N) - 1 \rceil$ times.*

*Proof.* In Appendix B. □

In the proof of Proposition 4, we also show that if a data unit is transmitted, then visiting (at least) one specific point in $dom(C)$ will make the algorithm divide a leading square wall. Since the algorithm selects a point to visit with uniform probability, the performance of the algorithm is closely related to the size of $dom(C)$ at each time step. In Lemma 4 we give an upper bound for expected value of $|dom(C)|$ at any time step until the algorithm finds $D$.

**Lemma 4.** *The expected number of points in $E(|dom(C)|)$ when $D$ is found, is bounded from above by $O(Nlog(N))$.*

*Proof.* In Appendix B. □

Finally, we prove Theorem 1.

*Proof of Theorem 1.* By Lemma 4, the maximal number of 'watched' points is $O(Nlog(N)))$. By definition of DivideAndConquerInit(), there is at least one leading square wall in $dom(C)$. Let $L$ be a leading square wall, and let $t$ be a time step. If $outcome(t) = 1$, then by Proposition 4, there exists $l \in L$ from which transmission may be sensed. If $l$ is visited at time $t$, $L$ will be divided. The probability of selecting $l$ is $\frac{1}{|dom(C)|} = O(\frac{1}{Nlog(N)})$, and therefore the expected number of attempts before a successful selection occurs is in $O(Nlog(N))$. By Lemma 3, after $O(\log(N))$ successful selections are done, $D$ is found. □

### E. Multiple Interceptors

In the previous subsections we presented and analyzed the algorithm, only for a single Interceptor. In the analysis of the algorithm (Theorem 1), it is assumed that visiting a point from $dom(C)$ will lead to the division of the leading square wall with probability of: $1/|dom(C)|$ (given that S-Routers transmit data at that time step), since only one of the $|dom(C)|$ points

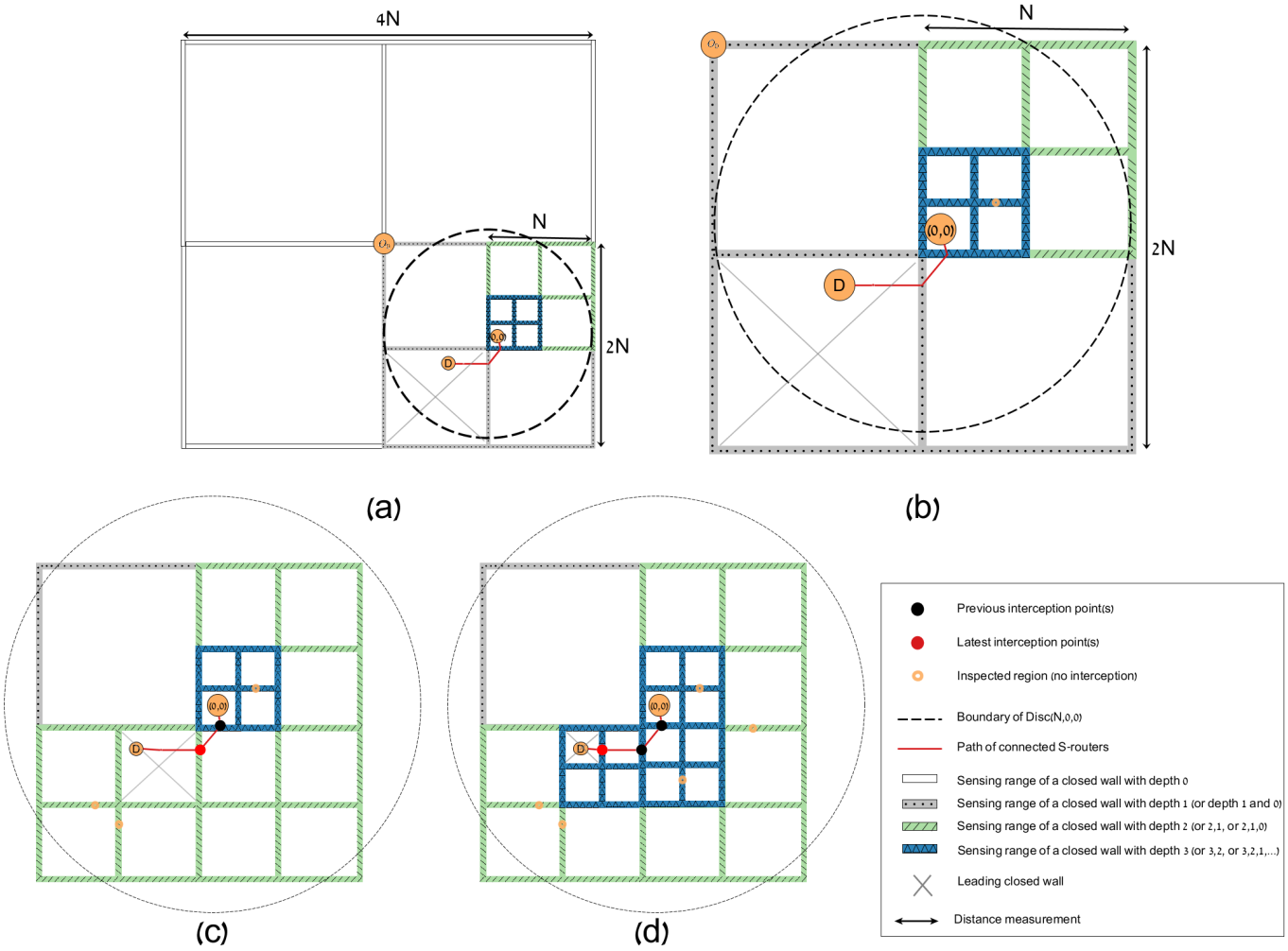


Fig. 8. Illustration of the operation Divide And Conquer Algorithm. The algorithm begins by visiting points on square walls generated in initialization, and with each interception it generates additional square walls which potentially include $D$. (a) Illustrates the initialization of the Divide And Conquer Algorithm, where four square walls of size $2N \times 2N$ are generated with random offset such that the entire searched region $Disc(N,(0,0))$ is contained by them. (b) Illustrates the next initialization step. The square wall that includes $(0,0)$ is repeatedly divided for $log(N)$ times. Note that one of the square walls includes the destination, but excludes $(0,0)$) (c) Illustrates the first detection of a transmission while visiting a point in one of the square walls (the red circle). The two square walls adjacent to the point of detection(s) are divided into four. The smallest square wall which includes the destination and excludes $(0,0)$ is smaller. (d) Illustrates the second and third transmission detection. The second detection occurs at the same point, which leads to additional division. The third transmission detection (the red circle) will lead to another division (excluded from this illustration), and then to the detection of $D$.

is ensured to intersect leading square wall. The Divide and Conquer Algorithm For Multiple Interceptors (abbreviated to D&Q$_{\text{MI}}$) increases this probability by visiting several points from $dom(C)$ simultaneously.

The D&Q$_{\text{MI}}$ is a simple generalization of Divide And Conquer Algorithm; instead of selecting a single point to search (DivideAndConquerAlgorithm(), line 17), the algorithm selects $M \leq |dom(C)|$ distinct points.

**Proposition 5.** *The expected outcome of D&Q$_{MI}$ is* $O(\lceil Nlog(N)/M \rceil log(N))$.

*Proof.* In Appendix B. □

*F. Improving performance*

The implementation presented in Figs.6,7 achieves the desired complexity (the expected number of transmitted data units is $O(N \cdot log^2(N))$, as in Theorem 1). However, its performance in practical implementations may be further improved. In particular, the following simple improvements were omitted in order to simplify the code and its analysis.

**Limiting search to $Disc(N,(0,0))$:** According to Lemma 1, it may be assumed that $D$ and all interceptions on routes that connect $D$ to $(0,0)$ will occur within $Disc(N,(0,0))$. Therefore, all leading square walls are contained in the disc (at least partially), and the algorithm may safely avoid visiting points outside the disc without decreasing the probability of interception when visiting a point in a leading square wall. This roughly decreases the number of steps by $8N$, since most of the points that are filtered out are part of the initial square walls the algorithm generates.Details omitted.

**Eliminating fully overlapping 'watched' points:** For each

square wall the algorithm divides, a significant portion of the new points to search that are added to $C$ are redundant, and may be filtered out. Consider a square wall which lies on the boundary of the square $S$, and is divided into smaller square walls that lie on the boundaries of the squares $S' = \{s_1, s_2, s_3, s_4\}$. Each of the new 'watched' points either lies on the boundaries of $S$ and one of the squares in $S'$, or on two of the squares in $S'$. Therefore, by filtering out new points $p$ for which $Range(p) \subset Range(C)$, i.e., points that do not increase the searched region, the algorithm can reduce the total size of $dom(C)$ while ensuring that at least one of the points in $dom(C)$ still leads to the division of a leading square wall.

This method roughly divides the size of $dom(C)$ by 3, since each time a square wall of size $w \times w$ is divided, about $4 \times \lceil w/\sqrt{3} \rceil$ points may be filtered, and only $2 \times \lceil w/\sqrt{3} \rceil$ new distinct point are needed.

## V. Intercepting Continuously-Transmitting S-Routers

In many scenarios S-Routers transmit at known, fixed rate, or, to further simplify, continuously, as per the continuous-transmission constraint (Def. 2). Namely, for every time-step $t$, holds $outcome(t) = 1$, i.e., $(0, 0)$ and $D$ are connected via a set of S-Routers who transmit at time $t$. The continuous transmission constraint seems, intuitively, to make it easier for the Interceptors to find the S-Routers and ultimately the destination $D$, and hence, to reduce the outcome.

In this section, we confirm this intuition, by presenting D&Q$_{\text{CTA}}$, an Interceptors algorithm which is a simple modification to the Divide And Conquer Algorithm, yet that achieves better results, when the constraint holds. Specifically, D&Q$_{\text{CTA}}$ reduces the expected outcome to only $\mathcal{O}(N \log(N) \log(\log(N)))$, improving significantly compared to the $\mathcal{O}(N \log^2(N))$ outcome of Divide And Conquer Algorithm.

Similarly to Divide And Conquer Algorithm, D&Q$_{\text{CTA}}$also generates square walls and visits their points in a randomized order. D&Q$_{\text{CTA}}$is based on the observation that, when the continuous-transmission constraint holds, and hence that the communication on the paths that 'cuts through' leading square walls is also continuous, D&Q$_{\text{CTA}}$bounds from below the probability of intercepting at least one transmission after visiting a leading square wall. Wee show this in Lemma 5 below.

**Lemma 5.** *Let $L = \{l_1, \ldots, l_{|L|}\}$ be a leading square wall to be visited by an Interceptor. Let $\Pi$ be a random permutation (order) on $\{1, \ldots, |L|\}$, and let $\mathcal{T} = \{t_1, \ldots, t_{|L|}\}$ be a sequence of $|L|$ different time steps. Then, for any $(S_L, S_T)$ satisfying continuous transmission, if an Interceptor visits the points of $L$ in order $\Pi$ at time steps $\mathcal{T}$, the probability that the Interceptor will sense at least one transmission $P(\exists i \leq |L|, j \in S_T(t_i)\ s.t.\ S_L(j) \in Range(\Pi(l_i))$ is $\geq 1 - \frac{1}{e}$, where the probability is over uniform choice of $\Pi$.*

*Proof:* In Appendix C. □

Specifically, D&Q$_{\text{CTA}}$prioritizes newly added 'watched' points, which include the points of the leading square wall in particular, instead of revisiting other points. If the algorithm fails to divide the leading square wall, it will waste a number of steps before visiting these points again.

D&Q$_{\text{CTA}}$is defined similarly to Divide And Conquer Algorithm (Figs. 7, 6), with the following additions:

(i) For each distinct point $p$ added to $C$, initialize a counter $\rho(p)$ to 0.
(ii) Each time a point $p$ is visited by the algorithm, increase $\rho(p)$ by 1.
(iii) When selecting a point to visit (Fig. 6, line 17), exclude points with non-minimal $\rho(p)$.
(iv) When adding a new square wall to $C$, for each point $p$ that was already included in $C$, reset the counter $\rho(p)$ to 0. This resets all counters associated with a newly added leading square walls.

### A. Analysis

By Lemma 5, after visiting once all the points of a leading square wall, in random order, an interception occurs with high probability. Therefore, only few repeated visits are required, until dividing the leading square wall. However, since the algorithm may not discern between the leading square wall and other square walls, the algorithm will visit many irrelevant points in $C$ for each mis-interception.

The following Theorem bounds from above the expected outcome of the algorithm. For proving it, we calculate the expected maximal number of mis-interceptions in the same leading square wall, derive an upper bound for the number of repeated visitations in each of the points of $C$, then find an upper bound for the expected number of time steps before $D$ is found.

**Theorem 2.** *If the continuous transmission constraint holds, then the expected outcome of D&Q$_{CTA}$ is $O(N \log(N) \log(\log(N)))$.*

*Proof:* In Appendix C. □

A natural question is the performance of D&Q$_{\text{CTA}}$ when the continuous transmission constraint does *not hold*. In Section VI-C we show S-Router algorithms which can cause arbitrary outcome for D&Q$_{\text{CTA}}$, when the continuous transmission constraint does not hold. Design of algorithm that will achieve the same performance when the assumption holds, and yet achieve good results also when it doesn't hold, remains a challenge.

## VI. Evaluation and Results

There is always a challenge in evaluating the practical performance of a defensive mechanism, whose results depend completely on the behaviour of the adversary - in our case, the S-Routers. Ideally, we would compare the performance of each algorithm against the best S-Router algorithm against it - however, how do we find that 'best S-Router algorithm' (for each Interceptors algorithm)?

Our approach was to use a set of S-Routers algorithms, each one 'optimized' for per each Interceptors algorithm. Of course, we do not really know how to produce the 'best' S-Router algorithm (in general, or for a particular Interceptors algorithm). Instead, we tried our best to develop good S-Routers algorithms, in two steps.

In the first step, we developed heuristic S-Routers algorithms, based, on our analysis of different Interceptors algorithms, and on 'trial and error', using a simulation and visualization environment we developed for this purpose. The visualization was helpful, in understanding unexpected advantages/disadvantages of different algorithmic choices and parameter sets; we will make this tool freely available, to allow further research and reproducibility. We found it possible to combine all these heuristic algorithms into a single, *parametrized* S-Router algorithm, s.t., with specific choices of parameters, we obtain the best results for each Interceptors algorithm. This step is described in Section VI-A.

In the second step, we used genetic programming to optimize the parameters of the parametrized S-Routers algorithm for each of the Interceptors algorithms, and then compared the results of the different Interceptors algorithms - each running against the 'best' parameters (of the parametrized S-Routers algorithm). We present the results in Section VI-B.

### A. Parametric Segmented Network S-Routers algorithm

In this section we present the *Parametric Segmented Network* S-Routers algorithm. The algorithm is heuristic and parametrized; it was designed for evaluating different Interceptor algorithms. In its design, we combined different ideas, including from the S-Routers algorithms that were used to bound from below the outcome of the Naive Disc Search, Naive Graph Search and Uniform Graph Search algorithms in propositions 1,2,3. In fact, all of these algorithms may be represented by specific parameter sets of the Parametric Segmented Network algorithm.

Predicting which S-Routers strategies will minimize the performance of the $D\&Q_{CTA}$ algorithm is quite challenging. For example, increasing the number of (alternative) transmission routes decreases the probability for dividing a leading square wall after it is visited, which is favorable for S-Routers. However, this also leads to more 'dense' topologies, which decreases the total number of divided square walls (and decreases the size of $dom(C)$), which is favorable for Interceptors.

Note that the parametrized algorithm only represents a fair, 'best effort'; in fact, we already have some ideas about possible further improvements to it, which may generate somewhat better results against both Divide And Conquer Algorithm and $D\&Q_{CTA}$, and was not yet implemented as part of Parametric Segmented Network.

The Parametric Segmented Network algorithm receives the following parameters as input:

- Number of separate network segments $c$.
- For each segment:
  - Portion $s_i \in [0,1]$ of all S-Routers that compose this segment s.t. $\sum_{1 \le i \le c} s_i \le 1$.
  - Number of parallel paths $k$ in this segment.
  - Portion $0 \le s'_i < s_i$ of S-Routers that is used for composing 'dead end' routes.
  - Number of parallel paths $k' < k$ used for 'dead end' routes.
  - Distance between parallel paths.
- Disconnected S-Routers distance $d \in [0,1]$. The location of each S-Router that is not used as part of a segment is chosen by selecting one of the S-Routers that are connected to $D$, with uniform probability, and a distance, distributed uniformly over $[0, dN]$.

The length of 'dead end' routes is determined by spreading the $s'_i \cdot N$ S-Routers as evenly as possible between the $k'$ routes. At each time step, one of these routes will transmit in each segment.

Fig. 9 illustrates a network composed of $N = 128$ S-Routers which are separated into three segments. The segments are composed of $0.35N, 0.3N, 0.3N$ S-Routers (from left to right), where $N = 128$. $D$ is the leftmost S-Router, while $(0,0)$ is the rightmost S-Router. Parallel paths of the same segment begin and end with a joint path (perpendicular to the parallel paths), and adjacent segments share one S-Router that connects the joint paths. The remaining $0.05N$ S-Routers that are disconnected transmit continuously, in order to mislead non-graph search algorithms (such as the Divide And Conquer Algorithm). The leftmost segment transmits into one additional 'dead end' parallel route.

The Parametric Segmented Network algorithm satisfies the continuous-transmission constraint. At each time step, every segment transmits data through one of its paths, and S-Routers of joint (perpendicular) paths transmit continuously. In Fig. 9, the top path in each segment transmits. It is easy to confirm that if at least one of the parallel path in each segment transmits then exists a path of transmitting S-Routers that connects $(0,0)$ and $D$.

### B. Results

In this section, we compare the theoretical bounds of the previous section to results obtained by simulations. In the following figures, S-Routers use the Parametric Segmented Network algorithm. In order to limit the dimension of optimization, the number of segments was limited to four. The parameters for the algorithm were optimized separately for each scenario i.e. for each Interceptor algorithm and each $N$ pairing. After the selection of the best parameters for each scenario, we ran the process enough times to ensure the evaluation of the parameter set is accurate. The results are displayed in Fig. 10. A confidence interval of 99% is used. All Interceptors algorithms utilize a single Interceptor. Due to the high computational costs, we used the genetic algorithm only for $N \le 128$. For larger $N$ values, the parameter sets were selected according to the conclusions found from smaller values.

The genetic algorithm consistently keeps a population size of 500. At each epoch, the algorithm applies standard roulette selection. Crossovers are done until for 75% of the population

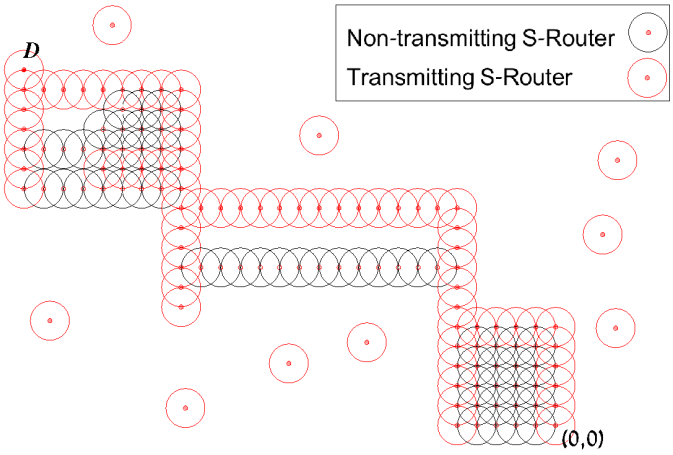


Fig. 9. 'Screen-shot' of the Parametric Segmented Network algorithm visualized by the simulation, for 128 S-Routers that are separated to three segments, at a specific time-step. The segments are composed of $0.35N, 0.3N, 0.3N$ S-Routers (from left to right), where $N = 128$. S-Routers marked in red are currently transmitting. At each step, one of the parallel paths in each segment transmits; additionally, the S-Routers connecting adjacent segments transmit continuously. Hence, data may flow from the rightmost S-Router, $(0, 0)$ to the leftmost S-Router, $D$.

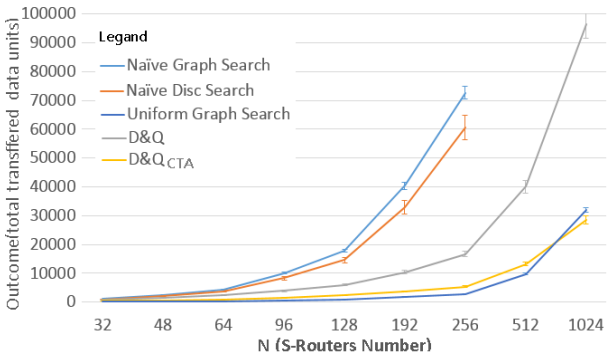


Fig. 10. Performance comparison of different Interceptors algorithms. Performance is estimated against Parametric Segmented Network S-Routers algorithm, where the algorithm's parameters were optimized for each Interceptor algorithm (using a single Interceptor) and each $N$ value separately.

is replaced, and then mutation is applied to 20% of the population where on average two elements of each mutated parameter set change. Changes are done by adding a Gaussian distributed random value. Since the S-Routers algorithms are making extensive use of randomization, the value of a given parameter set was the average outcome when running the process repeatedly for 15 times. We allocated 24 hours of CPU time for each scenario. Running a larger scenario takes significantly more time; therefore, the number of generations varied from several hundreds (for $N \leq 64$) to 48 (for $N = 128$).

*Discussion.* After examining the parameter sets that resulted from optimization, we discovered that certain properties consistently maximize the outcome for S-Routers. S-Routers that were disconnected from $(0, 0)$ and 'dead end' paths were not useful, and optimized solutions did not express them. Specifically, when S-Routers face the Uniform Graph Search algorithm, only two segments are needed, where the segment adjacent to $(0, 0)$ is composed of $0.1N$ to $0.4N$ S-Routers that will necessarily be intercepted by the algorithm. Additionally, parallel routes should be spread with distance three from each other, in order to maximize the number of distinct 'watched' points. The overhead for creating multiple routes appears to be significant, particularly for $N \leq 64$.

A result we did not expect was that when S-Routers face the Divide and Conquer and D&Q$_{\text{CTA}}$ algorithms, the outcome is maximized when only a single, long route was used, i.e., all segments express a single transmission route. In hindsight, we understood this; if using multiple routes than the Interceptors algorithms will eventually generate square walls that are too small to intersect all parallel paths of the same segment. As a consequence, these square walls have a lower probability for being divided, the total number of points that are 'watched' by the Interceptors algorithms decreases, and their performance improves (lower outcome). This may motivate an S-Routers algorithm that combines the graph-search approach with the divide-and-conquer approaches.

### C. False Assumption S-Routers algorithm

Two of the presented algorithms, D&Q$_{\text{CTA}}$ and Uniform Graph Search, assume that the continuous transmission constraint holds. As illustrated in Fig. 10, these algorithms perform significantly better than algorithms that do not rely on it. However, if this constraint does not hold, S-Routers may abuse the counters ($\rho$) used in these algorithms by deliberately exposing S-Routers that do not lead to $D$. We now describe the False Assumption algorithm, which allows S-Routers to guarantee any desired outcome.

The algorithm $(S_L, S_T)$ receives two constants $a, b \in \mathbb{R}$. $S_L = \{(-2, 0), (-1, 0), (0, 0), \ldots, (N - 3, 0)\}$ where $D = (-2, 0)$. That is, all S-Routers lie on a single straight line, where $D$ and $(0, 0)$ are in distance 2. For $0 \leq i \leq a$, $S_T(i) = \{(0, 0), (1, 0), \ldots (\lceil N/2 \rceil - 1, 0)\}$. For $a < i \leq a+b$, $S_T(i) = \{(0, 0), (1, 0), \ldots (N - 3, 0)\}$. Otherwise, $S_T(i) = S_L$.

**Proposition 6.** *If the continuous transmission constraint does not hold and Interceptors use D&Q$_{CTA}$ or Uniform Graph Search, the expected outcome of the False Assumption algorithm is unbounded.*

*Proof.* In Appendix D. □

If Interceptors are not certain whether the continuous transmission constraint holds or not, a simple solution would be using two Interceptors algorithms simultaneously (e.g. by interleaving the sequence of visitations). This allows Interceptors to increase performance if the assumption holds, and avoid an unbounded penalty otherwise.

## VII. Conclusions and Extensions

Stealthy networks, comprised of tiny, hard-to-locate devices, will soon become a part of reality; we use the term S-Routers for such devices, who can relay information, to form large networks. Stealthy networks will be used for different applications; many of the applications may represent threats to

privacy of individuals and organizations. Hence, it is important to develop efficient countermeasures. Due to the small size of the devices and their use of short-range communication, we envision the use of mobile devices, Interceptors, to localize the S-Routers. Ultimately, this becomes a 'game' between two sets of devices: the S-Routers and the Interceptors.

In this work, we investigated algorithmic issues related to stealthy networks and their interception. Our focus was on developing efficient algorithms for Interceptors, to expose the destination of stealthy network; we believe that such algorithms may be deployed as part of the design of countermeasures to stealthy networks.

There are many directions for improvements, extensions/variations, and further research. For example, in some scenarios the destination may be hard to detect except when it further relays information (e.g. to it's operator), similarly to S-Routers, and in contrast to the assumption specified in section II-B; however, the performance of presented algorithms seem unaffected by such change, except for the Naive Disc Search algorithm. As another example, if S-Routers' transmission schedule is not continuous, but Interceptors may predict it, then the D&Q$_{\text{CTA}}$ may still be used, if it is invoked only when S-Routers transmit.

Improvements may also be possible for the analysis. The current analysis provides upper bounds for the expected outcome of the problems, but lower bounds are yet to be found. Developing algorithms for S-Routers may be required for narrowing the gap between $O(N)$ and $O(Nlog^2(N))$ for the case of arbitrary transmission schedule and between $O(N)$ and $O(Nlog(N)log(log(N)))$ for the case of continuous transmission.

The presented model is general enough, to allow investigation of several related problems, including (1) multiple sources and/or destinations, (2) allowing S-Routers to buffer data, (3) introducing mobility, and much more.

Finally, note that the current model does not support decentralized algorithms. It is reasonable to expect that, in some practical scenarios, S-Routers may have to risk exposure in order to coordinate. An extension to the model is necessary for studying such scenarios.

## REFERENCES


[1] G. Shabsigh, "Covert communications in the rf band of primary wireless networks," Ph.D. dissertation, University of Kansas, 2017.

[2] O. Chen, C. A. Meadows, and G. Trivedi, "Stealthy protocols: Metrics and open problems." in *Concurrency, Security, and Puzzles*, 2017, pp. 1–17.

[3] L. Wang, G. W. Wornell, and L. Zheng, "Fundamental limits of communication with low probability of detection," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3493–3503, 2016.

[4] J. Hu, S. Yan, X. Zhou, F. Shu, and J. Wang, "Covert communication in wireless relay networks," *CoRR*, vol. abs/1704.04946, 2017.

[5] P. H. Che, M. Bakshi, and S. Jaggi, "Reliable deniable communication: Hiding messages in noise," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 2945–2949.

[6] B. A. Bash, D. Goeckel, D. Towsley, and S. Guha, "Hiding information in noise: Fundamental limits of covert wireless communication," *IEEE Communications Magazine*, vol. 53, no. 12, pp. 26–31, 2015.

[7] B. Muntwyler, V. Lenders, F. Legendre, and B. Plattner, "Obfuscating IEEE 802.15. 4 communication using secret spreading codes," in *Wireless On-demand Network Systems and Services (WONS), 2012 9th Annual Conference on*. IEEE, 2012, pp. 1–8.

[8] C. Santivanez and J. Redi, "On the use of directional antennas for sensor networks," in *Military Communications Conference, MILCOM 2003 IEEE*, vol. 1. ieee, 2003, pp. 670–675.

[9] B. G. D. I. R. J. W. Andrew T. Baisch, Onur Ozcan, "High Speed Locomotion for a Quadrupedal Microrobot," *The International Journal of Robotics Research*, 2014.

[10] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A Low Cost Scalable Robot System for Collective Behaviors," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3293–3298.

[11] "Russian Scientists Create Cockroach Spy Robot," accessed: 2015-10-07. [Online]. Available: https://thestack.com/iot/2015/09/25/russian-scientists-create-cockroach-spy-robot

[12] E. Ackerman, "Dragonfleye project wants to turn insects into cyborg drones," 2017, [Online; accessed 17-June-2017]. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/

[13] M. M. Maharbiz, "A cyborg beetle: Wireless neural flight control of a free-flying insect," in *Proceedings of the 22Nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes*, ser. SBCCI '09. New York, NY, USA: ACM, 2009, pp. 3:1–3:4. [Online]. Available: http://doi.acm.org/10.1145/1601896.1601900

[14] D. Thomas, R. McPherson, and J. Irvine, "Power analysis of local transmission technologies," in *Ph. D. Research in Microelectronics and Electronics (PRIME), 2016 12th Conference on*. IEEE, 2016, pp. 1–4.

[15] I. Bobic, "Ted cruz wants police to 'patrol and secure' u.s. muslim communities after brussels," Mar 2016, [Online; accessed 21-Nov-2017]. [Online]. Available: http://www.huffingtonpost.com

[16] M. Sidahmed, "Nypd's muslim surveillance violated regulations as recently as 2015: report," Aug 2016, [Online; accessed 21-Nov-2017]. [Online]. Available: https://www.theguardian.com

[17] C. Nunez, "Your old cell phone can help save the rain forest," Jun 2017, [Online; accessed 22-Nov-2017]. [Online]. Available: https://news.nationalgeographic.com

[18] M. K. Simon, J. K. Omura, S. R. A., and B. K. Levitt, *Spread Spectrum Communications Handbook*. New York, NY, USA: McGraw-Hill, 1994.

[19] S. Birnbach, R. Baker, and I. Martinovic, "Wi-fly?: Detecting privacy invasion attacks by consumer drones," *NDSS 2017: Network and Distributed System Security Symposium*, 2017.

[20] S. Bhattacharya and T. Başar, "Differential game-theoretic approach to a spatial jamming problem," in *Advances in Dynamic Games*. Springer, 2013, pp. 245–268.

[21] M. F. Miskon and R. A. Russell, "A repetitive observation strategy for recognizing a true anomaly and estimating its position," in *Austalasia Conference on Robotics and Automation, Canberra*, 2008.

[22] M. Ben-Adar Bessos, S. Birnbach, A. Herzberg, and I. Martinovic, "Exposing transmitters in mobile multi-agent games," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*. ACM, 2016, pp. 125–136.

[23] T. H. Nguyen, D. Kar, M. Brown, A. Sinha, M. Tambe, and A. X. Jiang, "Towards a Science of Security Games," in *New Frontiers of Multi-Disciplinary Research in STEAM-H*, 2016.

[24] R. Hohzaki, "Search games: Literature and survey," *Journal of the Operations Research Society of Japan*, vol. 59, no. 1, pp. 1–34, 2016.

[25] R. A. Collado and D. Papp, "Network interdiction–models, applications, unexplored directions," Citeseer, Tech. Rep., 2012.

[26] R. Lindelauf, P. Borm, and H. Hamers, "The influence of secrecy on the communication structure of covert networks," *Social Networks*, vol. 31, no. 2, pp. 126–137, 2009.

[27] A. Gutfraind, "Optimizing topological cascade resilience based on the structure of terrorist networks," *PloS one*, vol. 5, no. 11, p. e13448, 2010.

[28] N. Memon, J. D. Farley, D. L. Hicks, and T. Rosenorn, *Mathematical methods in counterterrorism*. Springer Science & Business Media, 2009.

[29] R. Kershner, "The number of circles covering a set," *American Journal of mathematics*, vol. 61, no. 3, pp. 665–671, 1939.

[30] G. K. Das, S. Das, S. C. Nandy, and B. P. Sinha, "Efficient algorithm for placing a given number of base stations to cover a convex region," *Journal of Parallel and Distributed Computing*, vol. 66, no. 11, pp. 1353–1358, 2006.

[31] J. R. Munkres, *Topology*. Prentice Hall, 2000.
[32] B. Eisenberg, "On the expectation of the maximum of iid geometric random variables," *Statistics and Probability Letters*, vol. 78, no. 2, pp. 135–143, 2008.
[33] R. L. Graham, *Concrete mathematics: a foundation for computer science*. Pearson Education India, 1994.

## APPENDIX A
## PERFORMANCE BOUNDS FOR NAIVE ALGORITHMS

*Proof for Lemma 1.* Since $D$ is connected to $(0,0)$ via a set of at most $N$ points, then there is a list of points $(l_1,\ldots,l_k)$, for $k \leq n$, s.t. $(0,0)$ is connected to $l_1$, $D$ is connected to $l_k$, and every two consecutive points $l_i, l_{i+1}$ are connected. From triangle inequality,

$$||D-(0,0)|| \leq ||D-l_k + \sum_{i=1}^{k-1} ||l_i - l_{i-1}|| + ||l_1-(0,0)|| \leq N$$

Hence, $D \in Disc(N,(0,0))$. □

*Proof for Proposition 1.* $DiscCoverage(N)$ covers $Disc(N,(0,0))$ entirely. Therefore, $D$ may be sensed from at least one point in $DiscCoverage(N)$, and the expected number of joint search attempts are needed before $D$ is found is $O(N^2/M)$. If S-Routers transmit continuously, it is also the expected number of transmitted data units until Naive Disc Search finds $D$. □

*Proof for Proposition 2.* We assume that for the coverage given by $DiscCoverage(N)$, each point is connected to $O(1)$ other points in the coverage. For any two connected S-Routers $p_1, p_2$ and a location of a successful search $l$ such that $Connected(p_1,l)$, the expected number of search attempts in $V = \{v_i | Connected(v_i,l)\}$ until $\exists v_i \in V$ : $Connected(v_i,p_2)$ holds is constant. The probability that any of the Interceptors finds a point connected to a new S-Router is in $O(\frac{M}{|A_S|})$, if S-Routers transmit through a single route at that time step. The expected number of transmissions done until a successful search occurs is $T(|A_S|,M) = O(|A_S|/M)$. Considering the worst case, where $N$ successful searches are needed for advancing from $(0,0)$ to $D$, the expected number of visitations is $T(1,M)+T(2,M)+\ldots+T(N-1,M) = O(\frac{N^2}{M})$. □

The proofs for Proposition 3 below and for Theorem 2 (in appendix C) rely on the following known results.

*Expected value of the maximum of $n$ geometric random variables*. For estimating the expected value of the maximum of $n$ independent, identically distributed geometric random variables, denoted with $E(M_n^*)$. From [32], if the mean of the variables is $1/p$, then:

$$\frac{1}{\lambda}\sum_{k=1}^{n}\frac{1}{k} \leq E(M_n^*) < 1 + \frac{1}{\lambda}\sum_{k=1}^{n}\frac{1}{k} \qquad \text{(EM*)}$$

where $1-p = e^{-\lambda}$ (further discussion on this bound and a more accurate approximation for $E(M_n^*)$ are also presented in [32]). Harmonic numbers of the form $\sum_{k=1}^{n}\frac{1}{k}$ are approximated by $\Theta(ln(n))$ (A proof is presented in [33], chapter 9).

*Proof of Proposition 3.* We consider a specific S-Routers algorithm $(S_L, S_T)$. The algorithm is a special case of the Parametric Segmented Network algorithm, as defined in section VI-A, where the network consists of two segments. The first segment is adjacent to $(0,0)$ and has a single route and consists of $\frac{N}{2}$ S-Routers, and the second segment is adjacent to $D$ and is composed of $\Theta(N/log(N)))$ parallel routes, each of length $\Theta(log(N))$, with distance two between every two parallel routes.

We use 'front' for referring to the S-Routers that are the furthest from $(0,0)$, and their transmission was previously intercepted by an Interceptor. The Uniform Graph Search will intercept the points of the first segment in $O(N)$ time steps, since these S-Routers transmit continuously. However, for each of the $\Theta(N/log(N))$ parallel routes, the probability that the algorithm will intercept a new S-Router at that route depends on the number of repeated visitations at the front of that route. Namely, the probability may be defined as a function of the maximal counter value $M_\rho$ (of all points) : $1-(1-log(N)/N)^{M_\rho}$.

After the algorithm detects $\Theta(log(N))$ new points in the same route, $D$ will be exposed. The expected number of time steps this takes may be calculated by regarding the number of detected points in each route as an independent geometric variable with probability $1-(1-log(N)/N)^{M_\rho}$ for success, where its mean is $1/(1-(1-log(N)/N)^{M_\rho})$. Then, we find the minimal $M_\rho$ for which the expected maximum value of the $\Theta(N/log(N))$ variables is in $\Theta(log(N))$.

Using the result mentioned in this appendix, by assigning in $(EM*)$ $n = \Theta(N/log(N))$ and $1/p = 1/(1-(1-log(N)/N)^{M_\rho})$, the maximum value may be calculated with: $\Theta(\frac{1}{1-(1-log(N)/N)^{M_\rho}} \cdot H_{|N/log(N)|})$ which is in $\Theta(log(N))$ only if $M_\rho$ is in $\Theta(N/Log(N))$. If $M_\rho$ reaches $\Theta(N/Log(N))$, and $\Theta(N)$ points of the graph were visited $M_\rho$ times (this holds at least for the points of the $N/2$-sized first segment), then the total number of visitation is bounded from below by $\Theta(N)\cdot\Theta(N/Log(N)) = \Theta(N^2/Log(N))$. □

## APPENDIX B
## PROOFS SUPPORTING THEOREM 1 AND PROPOSITION 5

*Proof for Lemma 2.* Recall that throughout the paper, walls and closed walls are always $\sqrt{3}$-spaced. Let $l_i, l_j$ be any two connected points in $L$ with distance $d_1 \leq \sqrt{3}$. We denote the distance between the two intersection points of $Range(l_i)$ and $Range(l_j)$ with $d_2$. As illustrated in Fig. 5, $d_2$ bounds from above the minimal distance between any $p_{In} \in In(L), p_{Out} \in Out(L)$. Adding auxiliary lines between $l_1, l_2$ to the intersection points creates a rhombus with sides of length 1 and a diagonal of length $d_1 \leq \sqrt{3}$, which allows calculating the length of the other diagonal $d_2 = 2\sqrt{1-(0.5d_1)^2} \geq 1$. □

For proving Lemma 3, we first prove that if an interception occurs in a leading square wall of the final depth $\lceil Log(N) - 1 \rceil$, then after adding the point to $C$, (as the algorithm does in DivideAndConquerAlgorithm(), line 12), $D$ necessarily may

be sensed by at least one the points in $C$. Afterwards, we bound from above the number of times the algorithm divides leading square wall before $D$ is found.

*Proof for Proposition 4.* Follows directly from Lemma 2: Since $outcome(t) = 1$, then $\{x|S_L(x) \in Range(L)\} \neq \{\}$, and for some S-Router $x$ there is at least one point in $L$ from which it may be sensed. □

Note that the dual result, where $(0,0) \in In(L)$ and $D \in Out(L)$, also holds and can be proven similarly, however we do not use this.

*Proof of Lemma 3.* After initialization (DivideAndConquerInit()), the edge length of all square walls is $\leq 2N$, including the edge length of all leading square walls. From the definition of leading square wall, each time a leading square wall is divided, the edge length of the new leading square wall is half the length of its predecessor. Therefore, after a leading square wall is divided $\lceil log_2(N) - 1 \rceil$ times, the edge length of all leading square walls is $\leq 2$. If interception occurs in leading square wall with edge length $\leq 2$, it will not divided (DivideAndConquerAlgorithm(), line 7). Instead, the algorithm will add to $C$ an additional point at the center of the leading square wall. Therefore, we only need to show that the points in $dom(C)$ suffice for intercepting $D$.

Let $SquareWall(s_w, (s_x, s_y))$ be a leading square wall s.t. $s_w \leq 2$. We find all points $(x, y)$ in $Square(s_w, (s_x, s_y))$ in which $D$ may be, but may not be sensed by the Interceptor from the points: $Range(SquareWall(s_w, (s_x, s_y)) \cup (s_x, s_y))$. Wlog, we assume $(s_x, s_y) = (0, 0)$, and we get the following constraints:

$$
\begin{aligned}
& w \leq 2 \\
& -w \leq x \leq w \\
& -w \leq y \leq w \\
& ||(x,y) - (0,0)|| \geq 1 \\
& ||(x,y) - (-w,0)|| \geq 1 \\
& ||(x,y) - (w,0)|| \geq 1 \\
& ||(x,y) - (0,-w)|| \geq 1 \\
& ||(x,y) - (0,w)|| \geq 1
\end{aligned}
$$

For $w > \sqrt{3}$ additional points are included in $SquareWall(s_w, (s_x, s_y))$ (due to the definition of $VWall()$ and $HWall()$ that compose square walls). However, it is easily confirmed that no $(x, y)$ satisfy these constraints; thus, introducing additional constraints will be redundant. □

We bound from above the expected number of square walls at each depth by bounding from above the expected number of all interceptions in all possible square walls of the previous depth (regardless of the number of square walls that were generated by the algorithm), using the result proved in Lemma 6 [1]. As we later see in that lemma, the expected number of interceptions depends directly on the total area covered by Interceptors. The region covered by visiting the points of a single square wall is composed of partially overlapping circles (of radius 1). Namely, for a square wall $L$, the region is: $\bigcup_{(p_x,p_y)\in L} Range(p_x, p_y)$. We bound this region from above using a single square shape with a boundary which is 'wide' enough to contain the entire region that is actually covered, as illustrated in Fig. 11. In order to define this shape, we approximate the region of each circle using a $2 \times 2$ square which includes that circle, then join all squares. For a square wall $L$, we use $\boldsymbol{Range^{\square}}(L) = \bigcup_{(p_x,p_y)\in L} Square(2, (p_x, p_y))$ to denote this region.

We use $(\boldsymbol{o_0})$ to denote the initial offset $o$ chosen by the algorithm (DivideAndConquerAlgorithm() , line 1) with uniform distribution from $Square(N, (0, 0))$.

All square walls at all depths are translated by $o_0$. Therefore, for depth $d_w$, the covered region is composed of all possible square walls of the form: $Disc(N, (0, 0)) \cap Range(SquareWall(w, (x_0 + (i + 0.5) \cdot w, y_0 + (j + 0.5) \cdot w)))$ where: $i, j \in \mathbb{Z}, -2^{d_w} \leq i, j \leq 2^{d_w}$. We separate that region (of all possible square walls together) into vertical and horizontal components and bound it from above using $Range^{\square}()$. Namely, we use $\boldsymbol{G_V}(d_w, x)$ to denote the vertical component: $\bigcup_{-2^{d_w} \leq i \leq 2^{d_w}} Range^{\square}(\ VWall(\ N, (x + iw, -N)))$ (illustrated in Figure 11(a)), and we use $\boldsymbol{G_H}(d_w, y)$ to denote the horizontal component: $\bigcup_{-2^{d_w} \leq i \leq 2^{d_w}} Range^{\square}(\ HWall(\ N, (-N, y + iw)))$ (illustrated in Figure 11(b)).

We denote the set of S-Router locations with $\boldsymbol{\mathcal{S}_L} = \{S_L(0), S_L(1), \ldots, S_L(N-1)\}$.

**Lemma 6.** *it holds that:* $E(|G_V(d_w, x_0) \cap \mathcal{S}_L|) \leq \frac{2N}{w}$ *and* $E(|G_H(d_w, y_0) \cap \mathcal{S}_L|) \leq \frac{2N}{w}$.

*Proof.* By linearity of expectation:

$$E(|G_V(d_w, x_0) \cap \mathcal{S}_L|) = \sum_{r \in \mathcal{S}_L} E(|G_V(d_w, x_0) \cap \{r\}|) = \sum_{r \in \mathcal{S}_L} P(r \in G_V(d_w, x_0))$$

Due to the distribution of $x_0$, for any $r \in \mathcal{S}_L$: $P(r \in G_V(d_w, x_0)) \leq \frac{2}{w}$. $E(Y)$ may be calculated similarly. □

In order to bound $|dom(C)|$ from above in Lemma 4, we consider all square walls which may be generated by the algorithm throughout its run, and prove (in Lemma 6) that the expected number of all square walls of a certain size is inversely proportional to their size.

*Proof for Lemma 4.* The initial number of points in $C$ is determined at the initialization of the algorithm (done in DivideAndConquerInit()), and this number increases with each

[1] The probability that different square walls will be divided depends on the specific (yet unknown) placement of routers, and therefore not independent from each other. As a consequence, formulating a simple recurrence relation for representing their number is difficult

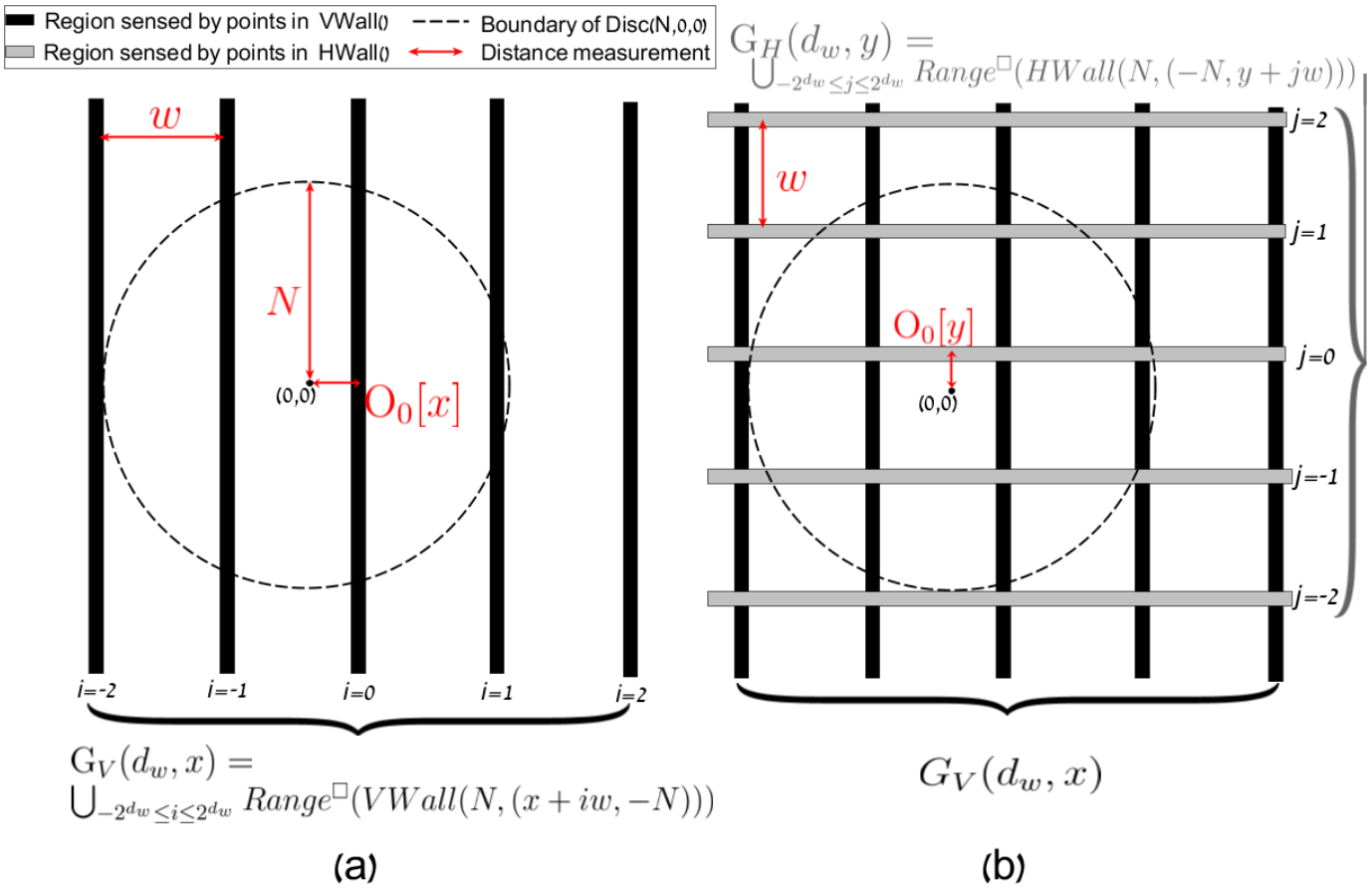


Fig. 11. Illustration of the walls that compose square walls of depth $d_w$, with the initial offset $o_0$. (a) Illustrates only the vertical component $G_V(d_w, o_0[x])$ and (b) also illustrates the horizontal component $G_H(d_w, o_0[y])$. Note that the walls are 'wide' i.e. bound from above the actual sensed region, since $G_V()$ and $G_H()$ are composed of regions of the form $\cup Range^{\square}()$

square walls that the algorithm divides, DivideAndConquerAlgorithm(), line 4.

Upon initialization (done in DivideAndConquerInit()), for each depth $1 \ldots \lceil log_2(N) - 1 \rceil$ the square wall which includes $(0,0)$ is added and further divided. Each time a square wall of depth $d_w$ is divided into four smaller square walls, $6 \times \lceil w/\sqrt{3} \rceil$ additional points are added. The total number of points in these square walls is:

$$\sum_{d_w=1 \ldots \lceil log_2(N)-1 \rceil} 6 \times \lceil w/\sqrt{3} \rceil = O(N)$$

The expected number of all points in square walls of all depths that were generated due to interceptions may be bounded from above as follows. While visiting the points of square walls with depth $d_w$, a transmission $r \in \mathcal{S}_L$ can only be intercepted if $r \in G_V(d_w, x_0) \cup G_H(d_w, y_0)$. Two square walls will be divided into smaller square walls (DivideAndConquerAlgorithm(), line 4), for each interception in one of the points of $G_V(d_w, x_0)$, and independently for each interception in one of the points of $G_H(d_w, x_0)$. By Lemma 6, the expected number of square walls of depth $d_w + 1$ the algorithm generates is $2 \times (2 \times \frac{2 \times N}{w})$. Each time a square wall is divided, $6 \times \lceil w/\sqrt{3} \rceil$ points are added.

For all square walls, the maximal edge length is $4N$ and the minimal depth is $-2$. Since the algorithm will not divide a square wall with edge length $\leq 1$ (DivideAndConquerAlgorithm(), line 7), all depths are between $-2$ and $\lceil log_2(N) - 1 \rceil$.

The expected number of points added to $C$ may be calculated by summing the number of points added in each depth separately. Formally:

$$\sum_{d_w=-1 \ldots \lceil log_2(N)-1 \rceil} (4 \times \frac{2 \times N}{w})(6 \times \lceil w/\sqrt{3} \rceil) = O(Nlog(N))$$

Finally, if interception occurs at a square wall with edge length $\leq 1$, the algorithm will add to $C$ an additional point (the center of the square wall). This may occur at most once for each square wall of depth $\lceil log_2(N) - 1 \rceil$, and the expected number of such square walls is bounded from above by $O(N)$.

The algorithm does not increase the number of points in any other occasion. Therefore, the expected number of points at any time step is bounded from above by $O(N) + O(Nlog(N)) + O(N) = O(Nlog(N))$. □

*Proof for Proposition 5.* The proof is similar to the proof of Theorem 1. However, for each transmitted data unit, the probability of dividing a leading square wall is in

$O(\frac{\min\{Nlog_2(N),M\}}{Nlog_2(N)}) = O(\min\{1, \frac{M}{Nlog_2(N)}\})$. Therefore, the expected number of transmitted data units until a leading square wall is divided is $O(\lceil Nlog(N)/M \rceil)$. Since $log_2(N)$ leading square wall divisions are still necessary, the result follows. □

## APPENDIX C
## PROOFS SUPPORTING THEOREM 2

*For Lemma 5.* We denote the set of S-Router locations which may be sensed by visiting $L$ with $U = \{u | S_L(u) \in Range(L)\}$, define the number of opportunities the Interceptor has for sense each S-Router with $f_s : U \to \mathbb{N}$ where $f_s(u) = |\{t \in \mathcal{T} | u \in S_T(t)\}|$, and denote $\overline{f_s} = \sum_{u \in U}(f_s(u)/|U|)$. Since S-Routers transmit continuously, by Lemma 2 there is at least one opportunity for sensing a transmission at each time step. Therefore: $\sum_{u \in U} f_s(u) \geq |L|$ and $\overline{f_s} \geq |L|/|U|$ hold. Since $\Pi$ is a random order, the probability that an S-Router $u \in U$ will be sensed is $\geq f_s(u)/|L|$, and the probability that at least one S-Router will be sensed is: $1 - \prod_{u \in U}(1 - f_s(u)/|L|)$. Notice that for any series $A = \{a_1, a_2, \ldots, a_n\}, a_i \in \mathbb{R}$ and its average $\overline{A}$, it holds: $\prod_{a_i \in A} a_i \leq \overline{A}^n$. By using this on $f_s$ we get:

$$P(\exists i \leq |L|, j \in S_T(t_i)\ s.t.\ S_L(j) \in Range(\Pi(l_i)) \geq 1 - (1 - \overline{f_s}/|L|)^{|U|} \geq 1 - (1 - 1/|U|)^{|U|} > 1 - 1/e.$$

□

*Notation.* Let $l \in dom(C)$ be a point to search, and $t \in \mathbb{N}$ be a time step. We use $\boldsymbol{\rho_t(l)}$ to denote the value of $\rho(l)$ at time $t$ *before* the algorithm selects a point to visit. For points $l \notin dom(C)$ we define $\rho_t(l) = \infty$.

*Proof of Theorem 2.* After a leading square wall will be divided several times, $D$ will be found. Wlog we assume there is a single leading square wall for each time step until $D$ is found (if more leading square walls exist at once, the probability for dividing one of them at each time step may only increase). Let $\mathcal{L} = \{L_0, L_1, \ldots\}$ be the different leading square walls that are generated by the algorithm before $D$ is found. By Lemma 3, after a leading square wall is divided for at most $log_2(N)$ times, $D$ will be found, and therefore $|\mathcal{L}| \leq log_2(N)$.

By Lemma 5, after all points of a leading square wall are visited once (at a random order), it will be divided with probability $> 1 - 1/e$. Let $P \subseteq dom(C)$ be a set of points to search, $t, t' \in \mathbb{N}$ be time steps and let $r \in \mathbb{N}$ be some value s.t. $t' > t$, $\forall p \in P : \rho_t(p) = r$ and $\forall p \in P : \rho_{t'}(p) \geq r + 1$. Since the algorithm selects a point to visit with uniform probability from points with similar $\rho$ value, the relative order in which the points were visited was (indirectly) selected uniformly from all possible permutations on $P$. If the counters of some points in $P$ were reset between $t$ and $t'$ then since the algorithm exclusively visits points with minimal $\rho$, these points will be prioritized over other points in $P$ until the counter reaches $r$ again (and then the selection continues in a similar fashion). Therefore, for a leading square wall for which all associated $\rho$ values are $\geq r$, Lemma 5 may be applied at least $r$ times.

For each leading square wall $L_i \in \mathcal{L}$, let $t_G(i) \in \mathbb{N}$ be the time-step it was generated at, and let $t_D(i)) \in \mathbb{N}, t_D(i) > t_G(i))$ be the time-step in which the it was divided at i.e. the first time-step in which interception occured at $l_i \in L_i$. We use the random variable $X_i = \rho_{t_D(i)}(l_i)$ to count the number of repeated mis-interceptions (in the entire square wall) before the first successful interception. The expected maximal value of all $X_i$ affects directly on the performance of the algorithm, since for each mis-interception in a leading square wall in which the maximal $\rho$ in $dom(C)$ increases, the algorithm may visit up to $O(Nlog_2(N))$ additional points (in $dom(C)$) before another opportunity for dividing leading square wall arises.

Each $X_i$ may be bounded from above by a geometric distribution with probability $(1 - 1/e)$ for success.

For bounding the expected value of the maximum of several independent, identically distributed geometric random variables, we use the result mentioned in Appendix A.

For $X_i$ the mean is $1/p = 1/(1 - 1/e)$, and the number of variables is $|\mathcal{L}|$. Therefore, by assigning in $(EM*)$ $n = |\mathcal{L}|$ and $1 - p = e^{-\lambda} \leftrightarrow \lambda = 1$, the expected maximum for the $|\mathcal{L}|$ variables is bounded from above by $O(ln(|\mathcal{L}|))$. In Lemma 4 we show that the expected number of points in $C$ is bounded from above by $O(Nlog(N))$. The expected number of the maximal $\rho$ for any point in $C$ never exceeds $O(log(|\mathcal{L}|))$, since at any time step the algorithm increases the lowest $\rho$, and (at least) for the points of the leading square wall, the expected value of $\rho$ is in $O(log(|\mathcal{L}|))$. Therefore, after an expected number of $O(Nlog(N)log(|\mathcal{L}|))$ time steps, where $|\mathcal{L}| \leq log_2(N)$, $D$ will be found. □

## APPENDIX D
## PROOF OF PROPOSITION 6

*Proof.* For the first $a$ steps, D&Q$_{\text{CTA}}$ is able to expose only the points $\{(0,0), (1,0), \ldots (\lceil N/2 \rceil - 1, 0)\}$. For $a > 4 \times (N/2)Log(N/2)$, each of these points will be visited at least once ($dom(C)$will have up to $\Theta(N)$ points per depth), and for $a = k\Theta(NLog(N)), k \in \mathbb{N}$, each point will be visited $k$ times. For the next $b$ steps, Interceptors will continue visiting points randomly, until one of the points in $\{\ldots(\lceil N/2 \rceil - 1, 0), \ldots (N - 3, 0))\}$ will be discovered. The counter $\rho$ associated with that point will be minimal for at least $k$ steps (in practice, more than a single point will be discovered, and it will be minimal for significantly more steps). Therefore, S-Routers will transmit to $D$ for at least $k - b$ steps. □