An Attacker-Defender Game for Honeynets

Jin-Yi Cai[1,2], Vinod Yegneswaran[3], Chris Alfeld[1], and Paul Barford[1]

[1] Computer Sciences Department, University of Wisconsin
Madison, WI, USA
[2] Radcliffe Institute, Harvard University
Cambridge, MA, USA
[3] Computer Sciences Lab, SRI International
333 Ravenswood Ave, Menlo Park, CA, USA

Abstract. A honeynet is a portion of routed but otherwise unused address space that is instrumented for network traffic monitoring. It is an invaluable tool for understanding unwanted Internet traffic and malicious attacks. We analyze the problem of defending honeynets from systematic mapping (a serious threat to their viability) as a simple two-person game. Our theoretical ideas provide the first formalism of the honeynet monitoring problem, illustrate the viability of network address shuffling, and inform the design of next generation honeynet defense systems.

1 Introduction

Malicious activity in the Internet is a significant and growing problem. The spectrum of threats includes denial-of-service (DoS) attacks, self-propagating worms, spam, spyware, and botnets. Malicious parties (Black Hats) are constantly looking for new potential victim systems, which are typically identified by scanning across large portions of the Internet address space. Many tools have been developed to facilitate this task, and researchers describe and evaluate the magnitude of the threat posed by highly efficient scanning methods (*e.g.,* [7]).

The network security community has recognized that the process of scanning for victims actually offers a unique opportunity to track malicious activity in the Internet. Standard configurations for destination networks typically drop packets destined for addresses that are not associated with a live system. Changing this configuration to route packets to a monitoring system enables passive collection of scanning packets. Even more compelling is the notion of routing these packets to systems with the ability to engage in conversations with the malicious hosts, thereby enabling a detailed attack profile to be gathered. This configuration is commonly referred to as a *honeynet* (*i.e.,* a network of honeypots).

Over the past several years, many honeynet installations have emerged across the globe, which have been used to monitor and identify many important details of malicious activity, such as the characterization studies of Internet "background radiation" [4], large worm outbreaks [2], and botnet activity [3]. Furthermore, research on honeynet systems is active and focused on enabling scalable and secure measurement at greater levels of detail.

The successes and utility of honeynets are not likely to be lost on the Black Hats. Several recent studies have shown that well-known monitoring entities such as Dshield.org [8] can be identified effectively by using different probing methods [1, 6, 5]. Black Hats can adopt these techniques to create blacklists of address space for honeynets, which would effectively render them useless.

In this paper, we address the problem of how to thwart attempts to map honeynets. To the best of our knowledge, we are the first to address this problem formally (it is mentioned briefly at the end of [9]). We will model the situation as a two-player game between an Attacker and a Defender. The objective of the Attacker is to identify the segment of monitored addresses (*i.e.,* the honeynet) within a larger address space. The Attacker does this by sending probes to the address space. The responses typically reveal distinguishing characteristics of the type of addresses being probed, *i.e.,* corresponding with either a live system or an address within the honeynet. An important capability in some honeynets is that they can respond to probes in a protocol compliant way (*e.g.,* [11]). When a probe is received on a honeynet address, the Defender can obfuscate its response by mimicking what would have been sent by a live system on a regular IP address. However, this obfuscation takes resources, and for a given resource bound, the Attacker will eventually be able to distinguish those responses coming from a honeynet.

The objective of a honeynet is to collect data as long as possible without being mapped by an Attacker. However, once identified, a honeynet will have to be assigned to a new set of addresses. This remapping or *shuffling* is a costly process. An additional objective of the Attacker is to identify the honeynet with a *minimum number of probes*. The Defender has two objectives. The first is to prevent the honeynet from being identified, which is done by periodically shuffling its location within the address space. The second objective is to extend the duration of shuffling epochs in order to minimize demands on the system responsible for shuffling. The Defender can use its protocol mimicking capability as a means for delaying the reshuffling. The game itself will focus on what transpires between shuffling epochs.

Mathematically, we model the response by the Defender for any probe as either 0 or 1. If the probe is sent to an IP address that is not part of the honeynet, then the response is 0. If the probe is sent to an IP address that is part of the honeynet, then the Defender may either respond truthfully with 1, or obfuscate with a "lie" 0. We assume that the Attacker cannot distinguish between a lie and the response from a live host. We model resource limitations as a *global lie budget* for the Defender. Within this context, we are able to prove optimal or near-optimal strategies for both the Attacker and Defender.

In Section 2.1, we first formally define the game. Then we develop general bounds, and the analysis leads to two particular strategies. The main theorem is Theorem 2, which establishes a *unique optimality* result. The proof is based on a key combinatorial lemma (Lemma 1) about certain optimal packing strategies. This lemma is easy to understand and quite plausible intuitively, but, its proof is more difficult. We then extend our basic formulation of the game to consider

the situation of having multiple segments of honeynets within the address space. Here we introduce a strategy with an amortized complexity analysis that is optimal up to a constant factor.

Our basic game theoretic formulation plus the extensions enable us to analyze the processes of attacking and defending honeynets experimentally. We conducted a series of evaluations over a range of possible configurations to assess the trade-offs between address space size, honeynet size, probe rates, and shuffling frequency. These results are omitted because of space considerations. In [10], we discuss practical considerations involved in network address shuffling and evaluate our implementation of a network address shuffling middlebox called Kaleidoscope.

The remainder of this paper is organized as follows. In Section 2 we formally define our game theoretic formulation of the problem along with some preliminary results. In Section 3 we prove our main theorem, assuming a key combinatorial lemma. In Section 4 we prove the combinatorial lemma. Because of the space limit, the extension to the basic game that considers multiple blocks of monitor segments is presented in an appendix. We also further discuss our basic model in the appendix. Summary and conclusions are in Section 5.

2 The Attacker-Defender Game

2.1 Basic Formulation

We model the adversary/honeynet interaction as a 2-person game between an Attacker and a Defender. A contiguous segment of monitored addresses is placed randomly within an address space. We call monitored addresses (*i.e.,* the honeynet) "black" and all other addresses "white". During the game the Attacker queries addresses and receives a reply based on the color of the address queried. White addresses must reply "white" (represented by a bit 0) but black addresses may answer black (represented by a bit 1) or "lie" and answer "white" (a bit 0). We impose a global limit on the number of lies allowed but they can be used flexibly. This global limit is imposed to reflect the cost on total system memory and other resources. (See appendix for more discussions on the model and its variations.)

Formally, let m, k and ℓ be positive integers. Let $n = mk$. Our address space is a circular array of n cells, identified with the additive group $\mathbf{Z}_n$, *i.e.,* each cell is indexed by some $i \in \mathbf{Z}_n$. We use the following notation:

Definition 1. *A* block *is a contiguous subset of $\boldsymbol{Z}_n$ of size m. Denote by $B_j = \{j, j+1, \ldots, j+m-1\}$ (indices are mod n), the block that starts at the j-th cell. As an alternate notation, $B^j = \{j-m+1, \ldots, j-1, j\}$ denotes the block that ends at j. The honeynet monitors are identified with a single block of black addresses B^c that ends at some c.*

The game starts with a random spin of B^c, that is, a uniformly distributed $c \in \mathbf{Z}_n$. We assume that both players know the parameters m, k, and ℓ. The

position c is known to the Defender D but unknown to the Attacker A whose aim is to find it. In a round of the game (*i.e.,* between shuffling epochs), A moves first and the two players alternate their moves until A knows c. A move of A consists of a *query* at a cell j. If $j \notin B^c$ then D replies with the bit $b = 0$. If $j \in B^c$ then D has a choice: D can either answer $b = 1$ or, *commit a lie* by answering $b = 0$. However, D can only lie $< \ell$ times in total. In short, n is the size of the circular address space, m is the length of a monitor block, $k = n/m$, and ℓ is the *quota* of *lies*. (Note that $\ell = 1$ is the smallest possible and represents no lies allowed.)

Denote by A (respectively D) a strategy by the Attacker (respectively the Defender). We will primarily analyze *pure* (nonrandomized) strategies; but with a slight abuse of notation we will use the same notation for randomized strategies. Let $V = V(A, D)$ denote the number of queries A makes against D until A learns the value c. Thus, V is a variable randomized over the uniform choices of $c \in \mathbf{Z}_n$ (and possibly over the randomization of the randomized strategies of the two players). Let v denote the expectation $v = v(A, D) = \mathbf{E}[V(A, D)]$. The objective of the Attacker is to minimize this quantity v. The objective of the Defender is to maximize v.

2.2 The Strategies: Round-Robin (RR) and Delay-Delay (DD)

Before giving a more detailed analysis of this game, we make some general observations. Suppose, during a round of the game, A learns a cell $j \in B^c$. If $m = 1$, then $c = j$ and the game is over. If $m > 1$, then, because B^c occupies a contiguous segment of cells, A can be sure that B^c is located in one of the following m possible ways (addition is done in $\mathbf{Z}_n$): $c = j, j + 1, \ldots, j + m - 1$. Then A can "zero-in" on the boundaries of B^c by performing the following binary search. First, A queries the cells $j^- = j - \lfloor m/2 \rfloor$ and $j^+ = j + \lfloor m/2 \rfloor$. At least one of j^- and j^+ must belong to B^c because the number of cells located strictly in between these two cells is $2\lfloor m/2 \rfloor - 1 < m$. Thus, D must either choose to answer truthfully $b = 1$ at least once, or (if permissible by the quota) commit an additional lie. If A receives both answers with $b = 0$, then he knows that a lie has been committed. So A continues to query j^+ and j^- until D answers $b = 1$. Assume D answers $b = 1$ to the query j^+; the case for j^- is symmetric. At this point, A knows that c is among the following $\lceil m/2 \rceil$ locations: $c = j^+, j^+ + 1, \ldots, j + m - 1$. So A can effectively consider a "shrunken" version of the problem as follows: Identify the cells between j and j^+ as a single cell. This identification effectively reduces the length of the block by $\lfloor m/2 \rfloor$ to $\lceil m/2 \rceil$. Now we can consider the problem of placing the shortened block on the condition that it contains a certain cell (the identified j and j^+). This describes a "binary search" process.

It is clear that every step of this "binary search" reduces the length from m' to $\lceil m'/2 \rceil$. When the length has shrunk to 1, the game is over. If $2^{r-1} < m \leq 2^r$, then it takes exactly $r = \lceil \log_2 m \rceil$ steps to reduce the length to 1. If A queries j^- and j^+ in random order, then on average each step takes 1.5 queries. Let ℓ' be the number of lies D is still allowed to commit ($\ell' < \ell$ and may be much less

because of lies committed prior to the "binary search"). Then we have an upper bound of $2\ell' + 1.5\lceil \log_2 m \rceil$ queries under this Attacker strategy, valid for any D.

We see that once the Attacker gains the knowledge of one $j \in B^c$, A can zero-in fairly rapidly. Therefore, it is reasonable to assume that a rational Defender will not reveal any $j \in B^c$ until being forced to (this statement will be qualified, see below). This reasoning leads us to the following Delay-Delay (DD) strategy for the Defender.

Definition 2. *The strategy* Delay-Delay *(DD) is as follows. Lie as long as the quota of lies allows.*

Definition 3. *The strategy* Round-Robin *(RR) is as follows. Pick any cell to start, e.g., $j = 0$. Query j consecutively ℓ times. Then set $j := j+m$, and repeat. As soon as a reply of $b = 1$ is received switch to a "binary search" strategy to zero-in.*

The following can be proved (the proofs are omitted):

$$V(\mathrm{RR}, \mathrm{DD}) \leq k\ell + 2\lceil \log_2 m \rceil \tag{1}$$

in the worst case, and

$$(k+1)\ell/2 + \lceil \log_2 m \rceil \leq v(\mathrm{RR}, \mathrm{DD}) \leq (k+1)\ell/2 + 1.5\lceil \log_2 m \rceil \tag{2}$$

on average. And against any other D,

$$v(\mathrm{RR}, D) \leq k\ell/2 + 1 + 1.5\lceil \log_2 m \rceil + 2(\ell - 1). \tag{3}$$

2.3 Delay-Delay against Any Attacker

We provide a simple lower bound for $v(A, \mathrm{DD})$. These considerations justify our adoption of DD for the Defender in the following. (Curiously, if the Defender knew that the Attacker is RR, then he actually should do the opposite of DD, and save his entire quota of lies for the "binary search" stage, saving a constant factor.) We note that (1) DD performs well against any A, and (2) for one particular A, namely $A = \mathrm{RR}$, DD performs almost as well as any D (Section 2.2).

Definition 4. *For an Attacker A and Defender D, the* Capitulation-Time*, $L = L(A, D)$, is the first time (the number of queries made) when A has made ℓ queries in B^c against D.*

For an arbitrary pair (A, D) it is possible that the game ends (when A learns c) before A ever hits ℓ times in B^c. In this case we define $L = \infty$.

We can prove the following

Theorem 1.

$$\mathbf{E}[L(A, \mathrm{DD})] \geq (\lfloor k\ell/2 \rfloor + 1)/2 > k\ell/4.$$

Note that our bound on $\mathbf{E}[L(A, \mathrm{DD})]$ is within a factor 2 of the case when the Attacker uses RR against any Defender strategy D. Once the Attacker is at Capitulation-Time, all that remains is to pinpoint c. Doing so takes $\Omega(\log m)$ time. Thus we arrive at the following corollary.

Corollary 1. *$v(A, DD) > kl/4 + \Omega(\log m)$.*

3 Round-Robin is optimal against Delay-Delay

In this section we establish our main result that RR is optimal against DD. In fact, we will prove a stronger theorem of *unique optimality* (Theorem 2).

Definition 5. *An Attacker strategy A is* essentially Round-Robin *(eRR) if it is of the following form. The strategy A first makes ℓ queries at some cell j. It then updates j to j' where $j' \equiv j \pmod m$ and j' has not been queried and makes ℓ queries at j'. The strategy repeats this behavior until it finds some $j \in B^c$,* i.e.*, receives $b = 1$, at which point it switches to a "binary-search".*

Clearly,

$$\mathbf{E}[L(\mathrm{eRR}, \mathrm{DD})] = \mathbf{E}[L(\mathrm{RR}, \mathrm{DD})]. \tag{4}$$

Theorem 2. *For any Attacker A that is not eRR,*

$$\mathbf{E}[L(\mathrm{RR}, \mathrm{DD})] < \mathbf{E}[L(A, \mathrm{DD})]. \tag{5}$$

Lemma 1. *Let m, k, p and ℓ be positive integers, and $n = mk$. Let $\mathcal{S}$ be a multi-set $\{S^1, S^2, \ldots, S^p\}$, where each $S^i \subset \mathbf{Z}_n$ is a contiguous segment with $|S^i| = m$. We say $c \in \mathbf{Z}_n$ is* **heavy** *with respect to $\mathcal{S}$ if c is covered by at least ℓ blocks $S^i \in \mathcal{S}$. Then,*

$$|\{c \in \mathbf{Z}_n \mid c \text{ is heavy}\}| \leq m \cdot \lfloor \frac{p}{\ell} \rfloor.$$

Let $C(\mathcal{S}) = |\{c \in \mathbf{Z}_n \mid c \text{ is heavy}\}|$ denote the number of heavy points w.r.t. $\mathcal{S}$. We note that a weaker bound $C(\mathcal{S}) \leq \lfloor m \cdot \frac{p}{\ell} \rfloor$ follows easily from a volume argument. But we need the stronger version to prove Theorem 2. The proof of the lemma is presented in the next section. Here we prove the Theorem assuming the Lemma.

Proof (of Theorem 2). Clearly $\Pr[L(\mathrm{RR}, \mathrm{DD}) = \infty] = 0$. If $\Pr[L(A, \mathrm{DD}) = \infty] \neq 0$, then (5) is obvious. Suppose $\Pr[L(A, \mathrm{DD}) = \infty] = 0$. We are concerned with

$$\mathbf{E}[L] = \sum_{i=1}^{\infty} \Pr[L \geq i] = 1 + \sum_{i=1}^{\infty} \Pr[L > i], \tag{6}$$

for $L = L(A, \mathrm{DD})$ and $L(\mathrm{RR}, \mathrm{DD})$. Our goal is to show that RR minimizes $\mathbf{E}[L]$. Let $\mathbf{E}_A[L] = \mathbf{E}[L(A, \mathrm{DD})]$ denote the expectation of the Capitulation-Time for Attacker A against Defender DD. Similarly, $\Pr_A$ indicates the probability

for Attacker A against Defender DD. Our first goal is to prove the (nonstrict) dominance of RR:

$$\mathbf{E}_A[L] \geq \mathbf{E}_{\mathrm{RR}}[L]. \tag{7}$$

We establish $\mathbf{E}_A[L] \geq \mathbf{E}_{\mathrm{RR}}[L]$ by proving term by term $\Pr_A[L > i] \geq \Pr_{\mathrm{RR}}[L > i]$ in the sum (6). Observe that if $i \geq k\ell$, then $\Pr_{\mathrm{RR}}[L > i] = 0$. Thus, we only need to consider $i < k\ell$. The inequality is equivalent to $\Pr_A[L \leq i] \leq \Pr_{\mathrm{RR}}[L \leq i]$. Define H_i to be the number of hits B^c received among the first i queries. Then the event $[L \leq i]$ is equivalent to $[H_i \geq \ell]$. Thus, we seek to show for all i,

$$\Pr_A[H_i \geq \ell] \leq \Pr_{RR}[H_i \geq \ell]. \tag{8}$$

Imagine that we are given the first i (not necessarily distinct) query points. Each query point j defines a block B_j. We observe that j hits B^d iff d belongs to the block B_j. Let $\mathcal{S}_i$ be the configuration consisting of i blocks corresponding to the queries the Attacker *would* make *assuming the Defender answers* $b = 0$ to the first $i - 1$ queries. We claim

$$\Pr_A[H_i \geq \ell] = \frac{C(\mathcal{S}_i)}{n}. \tag{9}$$

The equality is clear if all first i answers are indeed $b = 0$. We prove that (9) is valid for the actual interaction that defines H_i. We prove this by induction. For $i = 1$ the result holds. Assume the result holds up to $< i$. The probability $\Pr_A[H_i \geq \ell]$ is $1/n$ times the number of d such that B^d was hit at least ℓ times during the first i queries.

$$\Pr_A[H_i \geq \ell] = \Pr_A[H_{i-1} \geq \ell] + \Pr_A[(H_i = \ell) \wedge (H_{i-1} < \ell)]. \tag{10}$$

By induction $\Pr_A[H_{i-1} \geq \ell] = \frac{C(\mathcal{S}_{i-1})}{n}$. Now, for the second term, the conjunction $(H_{i-1} < \ell)$ implies that DD answers the first $i - 1$ queries with $b = 0$. Thus, the probability $\Pr_A[(H_i = \ell) \wedge (H_{i-1} < \ell)]$ counts the number of heavy points $d \in \mathbf{Z}_n$ that are heavy in $\mathcal{S}_i$ but not heavy in $\mathcal{S}_{i-1}$. It follows that the sum of these two probabilities is exactly $\frac{C(\mathcal{S}_i)}{n}$, completing the induction.

By Lemma 1, the probability $\Pr_A[H_i \geq \ell]$ is maximized by RR. Therefore, by the dominance of RR, term by term in (6), we obtain $\mathbf{E}_{\mathrm{RR}}[L] \leq \mathbf{E}_{\mathrm{A}}[L]$.

To prove the strict dominance of RR (and eRR) over any other A against DD, we reason as follows. If the first ℓ queries are not at the same cell, then at the ℓ-th query, RR produces exactly m heavy points while A has strictly less. Thus, at the ℓ-th term in the sum for $\mathbf{E}_A[L]$, the inequality $\Pr_A[H_\ell \geq \ell] < \Pr_{\mathrm{RR}}[H_\ell \geq \ell]$ is strict. As we have the (nonstrict) dominance of RR for every term we arrive at $\mathbf{E}_{\mathrm{RR}}[L] < \mathbf{E}_A[L]$.

We now assume that the first ℓ queries are at a single cell. Let j_1 be that location. If the next ℓ queries are not at some cell j_2 then consider the time step 2ℓ. By the same argument, at 2ℓ we have a strict inequality and a (nonstrict) dominance of RR elsewhere, that again gives $\mathbf{E}_{\mathrm{RR}}[L] < \mathbf{E}_{\mathrm{A}}[L]$. This argument

proves that to be optimal, the locations of the queries $j_1, j_2, \ldots$ must be repeated ℓ times each in succession.

Finally, consider the possibility of two query locations j and j' with $j \not\equiv j' \pmod{m}$. Then at time step $i = k\ell$, RR produces a *perfect* cover of all $\mathbf{Z}_n$ with multiplicity ℓ. Meanwhile, A has an *imperfect* cover of all $\mathbf{Z}_n$ that produces a strict inequality in favor of RR. So again, we find that $\mathbf{E}_{\mathrm{RR}}[L] < \mathbf{E}_A[L]$.

This proves the strict optimality of RR (and eRR).

4 The Packing Lemma

In this section we prove Lemma 1 which is a lemma about a packing problem.

We first remark that the lemma assumes $m|n$. This integral divisibility is essential. Here is a counter example when this is not true. Let $n = 3$, $m = 2$, $\ell = 2$ and $p = 3$. Define $\mathcal{S} = \{\{1, 2\}, \{2, 3\}, \{3, 1\}\}$. Then, the number of heavy points $C(\mathcal{S}) = 3$, but the bound is $m \cdot \lfloor \frac{p}{\ell} \rfloor = 2$. This is one indication of the subtlety of this lemma.

The bound in the Lemma is trivial if $p \geq k \cdot \ell$. So we assume $p < k \cdot \ell$. Write $p = q \cdot \ell + r$, where $q = \lfloor \frac{p}{\ell} \rfloor < k$ and $0 \leq r < \ell$.

Consider RR (or eRR). Each query at j defines a block $B_j = \{j, j+1, \ldots, j+m-1\}$ that is a contiguous segment of m cells, starting at j. It is obvious that $j \in B^c$ iff $c \in B_j$. Now it is clear that the upper bound in the Lemma is achieved by RR (and eRR), since each batch of ℓ repeated queries defines a block, each of which is repeated ℓ times, and these blocks are spaced exactly m cells apart from one batch to the next. Thus, the first $q \cdot \ell$ queries define q distinct and disjoint blocks each repeated ℓ times, and together they cover $m \cdot q$ *heavy* points. The point of the lemma is that one cannot possibly use the leftover "capacity" of rm to cover more *heavy* points. (The example above shows that this "no use of leftover capacity" is a subtle point.)

We note that if $p \equiv 0 \pmod{m}$, then the lemma is trivial. However, this is not sufficient for the proof of the Main Theorem. We assume $p \not\equiv 0 \pmod{m}$.

Proof (of Lemma 1). Fix p. Let $\mathcal{S}$ be a configuration of p blocks that maximizes the number of heavy points, denoted by $C(\mathcal{S})$. The proof of the lemma consists of two major steps. In the first step we will show that we may always modify the configuration $\mathcal{S}$ to another $\mathcal{S}'$ with the same number of blocks, such that $C(\mathcal{S}) \leq C(\mathcal{S}')$, and in $\mathcal{S}'$ no point c is covered by more than ℓ blocks. The second step of the proof shows that for such configurations $\mathcal{S}'$, $C(\mathcal{S}') \leq m \cdot \lfloor \frac{p}{\ell} \rfloor$.

Step 1: Given $\mathcal{S}$. Define a $k \times m$ matrix N as follows: The entries of N will be indexed by (i, j), where $0 \leq i < k$ and $0 \leq j < m$, and $N_{i,j}$ is the number of blocks in $\mathcal{S}$ covering the cell $c = im + j$. For later purposes we will also denote $N_c = N_{im+j} = N_{i,j}$. Note that each cell $c \in \mathbf{Z}_n$ has a unique expression $c = im + j$, where $0 \leq i < k$ and $0 \leq j < m$. On each column j of N, the blocks counted in $N_{i,j}$, for $0 \leq i < k$, all come from distinct blocks in $\mathcal{S}$, because each block has length m, and therefore cannot appear in more than one count in a column.

Each block in $\mathcal{S}$ contributes a total of m to the sum $\sum_{i,j} N_{i,j}$. In fact on each column, each block contributes exactly 1; therefore for every $0 \le j < m$, $\sum_{i=0}^{k-1} N_{i,j} = p$. In particular, since $p < k\ell$, there must be at least one $N_{i,j} < \ell$ for every column $0 \le j < m$.

If all entries $N_{i,j} \le \ell$, then **Step 1** is done. Assume otherwise. We now perform what we call a sequence of "**wipe-the-cream**" operations: Start at any $c_0 = i_0 m + j_0$ with $N_{i_0,j_0} \le \ell$. Cyclically increase c_0 by 1, until we find the first $c = im + j$ such that $N_{i,j} > \ell$. If $c' = c - 1 = i'm + j'$ is the previous cell, then it is clear that $N_{i,j} > \ell \ge N_{i',j'}$, and that the increase must be due to some blocks starting at the location c. Let f be the number of blocks starting at the location c, then $f \ge N_{i,j} - \ell$. Let $f' = N_{i,j} - \ell$, then $f \ge f' > 0$. Now *move* f' of the f starting blocks one cell to the next, to start at $c + 1$ instead. Note that for the new configuration the new value at c is $\tilde{N}_{i,j} = \ell$.

We then start at c and repeat. This step of "wipe-the-cream" operation is continued until there is no more $N_{i,j} > \ell$. We claim this happens eventually.

Consider the effect of one step of this "wipe-the-cream" operation on the matrix N. Exactly two entries of this matrix N are changed by this one step: the entry $N_{i,j}$, which is changed from $> \ell$ to $= \ell$, and the entry $N_{i+1,j}$, which is increased by f'. Hence the number of heavy points C is *not* decreased. Moreover, if we perform the above sequence of "wipe-the-cream" operations in row major order, but we focus on its effect on each column of the matrix exclusively, the entire cyclic sequence of "wipe-the-cream" operations can be viewed *as if it were to be performed* on each column individually, in a parallel fashion. (At this point, the reader probably can see why we call it a "wipe-the-cream" operation.) One can think of the effect on any column j as a virtual "wipe-the-cream" operation that merely shifts the numbers $N_{i,j}$ in excess of ℓ to the next entry on the same column, as described above: If $N_{i,j} > \ell$, then the new $\tilde{N}_{i,j} = \ell$ and the new $\tilde{N}_{i+1,j} = N_{i+1,j} + f' = N_{i+1,j} + (N_{i,j} - \ell)$.

We note that when we perform one step of this "wipe-the-cream" operation at c, the sum of two consecutive terms on a column at c and $c + m$ is unchanged. If we actually shifted some blocks for r consecutive terms in *one* column, this creates a plateau of r consecutive terms with the respective $N_{i,j} = \ell$. These terms effectively no longer participate in any future "wipe-the-cream" operations; in any future rounds they simply pass any leftover "cream" (if there is any) from the previous cell to the next cell on the same column. If we focus on an individual column, and consider the virtual "wipe-the-cream" operation, this effectively reduces the length of the column on which we perform this virtual "wipe-the-cream" operation. If we disregard the cells already having $N_{i,j} = \ell$ (and thus removed for the virtual "wipe-the-cream" operation), every actual step that shifts a positive amount from one cell to the next strictly reduces the length of the column. However, each column sum is $p < k\ell$ and is preserved throughout, some entry $< \ell$ on each column must remain all the time. Therefore, this virtual length of the column cannot be reduced to 0. This proves that the sequence of the "wipe-the-cream" operation must terminate on each column, and therefore must terminate overall. **Step 1** is proved.

Step 2: Assume $\mathcal{S}$ is a configuration with all $N_c \leq \ell$, and achieves maximum count $C(\mathcal{S})$ of heavy points.

We have assumed that $p < k\ell$ and $p \not\equiv 0 \pmod{\ell}$, for otherwise the Lemma is easy to prove. Note that these two conditions imply $q < k$ and $1 \leq r < \ell$ in $p = q\ell + r$. In particular, we may assume $\ell > 1$. We may consider the Lemma already proved inductively for any smaller values of ℓ. The base case $\ell = 1$ is trivially true. We may also consider the Lemma already proved inductively for any smaller values of p; again the base case is trivial.

Now we define two notions: a **gap** and a **train**. A gap G is a contiguous set of points $\{a, a+1, \ldots, b-1\} \subseteq \mathbf{Z}_n$, where $a \neq b$ (thought of as $a < b < a + n$), such that $N_{a-1} = N_b = \ell$ and $N_a, N_{a+1}, \ldots, N_{b-1}$ are all $< \ell$. We say the length of the gap is $b - a$. (Strictly speaking we take the positive modulus of $b - a \pmod{n}$.)

A (w, t)-train T is a collection of $w \cdot t$ many blocks, such that there exists a $c \in \mathbf{Z}_n$, and for all $0 \leq i \leq t - 1$, there are exactly w blocks in T that start at the cell $c + im$. Thus, a (w, t)-train T covers tm contiguous cells starting at c, each with multiplicity w. We say T has width w and length tm.

For any $w > 0$, a (w, k)-train T is called a complete train. A complete train wraps around the whole circular array $\mathbf{Z}_n$. If we have a complete train, we can remove all the blocks in a $(1, k)$-complete train, and obtain a configuration $\mathcal{S}^*$ which covers the same number of heavy points for the parameters $\ell' = \ell - 1$ and $p' = p - k$. *i.e.*, $C(\mathcal{S}) = C(\mathcal{S}^*)$. By inductive hypothesis, this $C(\mathcal{S}^*) \leq \lfloor p'/\ell' \rfloor \cdot m$. But by $k > q$,

$$p' = q\ell + r - k \leq q\ell + r - (q + 1) = q(\ell - 1) + (r - 1),$$

where $0 \leq r - 1 < \ell - 1 = \ell'$. Thus, $\lfloor p'/\ell' \rfloor \leq q$. It follows that $C(\mathcal{S}) \leq q \cdot m$. Thus, any complete train finishes the proof.

The strategy in **Step 2** will be the following. We will consider a *gap with minimum length*, and define an appropriate *train* and a *movement* of the train. Assume there are at least two gaps in $\mathcal{S}$. The movement of the train will not change $C(\mathcal{S})$, and will either (a) reduce the number of gaps in $\mathcal{S}$, or (b) reduce the length of the minimum-length gap without increasing the total number of gaps in $\mathcal{S}$, or (c) move the minimum-length gap one cell closer to another gap while not affecting any gap other than the minimum-length gap. In case (c), if repeated, a sequence of such train movements will eventually merge the minimum-length gap with a nearby gap and thus reducing the total number of gaps. It is clear by $p < k\ell$ that there must be at least one gap. The above sequence of train movements will eventually reduce the situation to a configuration with only one gap, which will be directly handled.

Now we carry out this plan. Let $G = \{a, a + 1, \ldots, b - 1\}$ be a gap with minimum length. If s is the number of blocks starting at b and e is the number of blocks ending at $b - 1$, then $s - e = N_b - N_{b-1} > 0$. Let $w = s - e$. We will define a (w, t)-train for some t. As $w \leq s$ there are at least w blocks starting at b. Choose any such w starting blocks. Consider cell $b + m - 1$. If $N_{b+m-1} < \ell$, then we may move these w blocks one cell to the left, thereby producing a new

configuration with $\tilde{N}_{b-1} = \ell$, $\tilde{N}_{b+m-1} = N_{b+m-1} - w$, and with no other changes to N. As $b + m - 1$ was already not a heavy point, this movement would have increased $C(\mathcal{S})$ by 1. Since $\mathcal{S}$ was assumed to be optimal, this is impossible. Therefore we may assume $N_{b+m-1} = \ell$.

Next, consider cell $b + m$. Either $N_{b+m} < \ell$, or $N_{b+m} = \ell$. If $N_{b+m} = \ell$, then since there are at least w blocks ending at $b + m - 1$, there must be at least that many blocks starting at $b + m$. We pick any w of these starting blocks and together with the w blocks starting at b to form a $(w, 2)$-train. On the other hand, if $N_{b+m} < \ell$, then $b + m$ must be the starting cell of a gap G'.

In the case with $N_{b+m} = \ell$, we will, by the same argument, continue at $b + 2m$, $b + 3m, \ldots$, and keep "hooking up" the train until we have defined a (w, t)-train that borders on a gap G' that starts at the next cell $b + tm$. Note that if the (w, t)-train is extended all the way by wrapping around to $b - 1$, we would have a complete (w, k)-train. As noted earlier, this completes the proof of the Lemma. Thus, we assume $t < k$.

Now there are two cases: Either $G' = G$ or $G' \neq G$. If $G' \neq G$, then by moving the (w, t)-train one cell to its left, we change N_{b-1} from $< \ell$ to $\tilde{N}_{b-1} = \ell$ and N_{b+tm-1} from $= \ell$ to $\tilde{N}_{b+tm-1} < \ell$. No other changes occur to N. This keeps $C(\mathcal{S})$ unchanged, thus still optimum, while removing one cell out of the gap G and adding one cell to the gap G', which was to start at $b + tm$. No changes occur to any other gap. Of course it is possible that the gap G disappears (if its length was 1), or its length is reduced by 1. The gap G' may stay as a single gap, with size increased by 1, or it may be merged with another gap to its left. In any case, either the number of gaps is reduced or the number of gaps stays the same with the length of the minimum gap reduced (or both).

Let's consider the case $G' = G$. (Note, this does not imply that $\mathcal{S}$ has only one gap.) This means the (w, t)-train wraps around all the way to end in $a - 1$; in particular $a \equiv b \pmod{m}$. If we also move the (w, t)-train one cell to its left, we would have effectively moved the cell locations of the minimum-length gap G one cell to its left. No changes occur to any other gap. This may of course merge G with a gap to its left, or it simply moves the location of G one cell to its left without changing the status of any other gap. In the latter case, both the total number of gaps and the minimum length of a gap are unchanged. We can then proceed to define another (w', t')-train starting at $b - 1$ and repeat the above process. (Note that it is possible that $w' \neq w$ and $t' \neq t$.)

We have proved that the goals (a), (b) and (c) can be accomplished as stated earlier. It follows that eventually we can arrive at a configuration that has the same p, still having maximum $C(\mathcal{S})$, and at most one gap. Since $p < k \cdot \ell$, a gap must exist, therefore it has *exactly one* gap.

If we start at this unique gap $G = \{a, a + 1, \ldots, b - 1\}$ and repeat the above process of finding a (w, t)-train, it must end in the same gap $G' = G$. Thus $a \equiv b \pmod{m}$, and therefore the gap G has a length a positive multiple of m. Let $b - a = g \cdot m$, then $0 < g < k$. This is the length of the train defined.

Since G is unique, all $b - a$ points are heavy points covered by $\mathcal{S}$ with multiplicity exactly ℓ. Thus by a volume argument, $(b - a)\ell \leq pm$, and thus

$g\ell \leq p$. However g is an integer, so $g \leq \lfloor \frac{p}{\ell} \rfloor$. It follows that the total number of heavy points $C(\mathcal{S}) = b - a = g \cdot m \leq m \cdot \lfloor \frac{p}{\ell} \rfloor$. This completes the proof.

5 Summary and Conclusion

In the perennial struggle against network intruders and malicious attacks, safeguarding honeynet monitors is becoming an urgent problem. This paper abstracts the problem in a game theoretic framework, and analyzes optimal strategies for both the Attacker and Defender. To achieve provable results and mathematical elegance, it is necessary to abstract away many systems details. But these abstractions aim to capture the most salient features of the network reality, and to achieve a reasonable balance of system relevance and theoretical tractability. As far as we know, our paper is the first to provide a theoretical foundation for honeynet defense. It has also proven useful in guiding the development of Kaleidoscope, an experimental middlebox for safeguarding honeynet monitors. Our experience with Kaleidoscope also reveals a number of system issues and variants that can be further analyzed in a game theoretic setting.

References

1. J. Bethencourt, J. Franklin, and M. Vernon. Mapping Internet Sensors with Probe Response Packets. In *Proceedings of USENIX Security Symposium*, 2005.
2. E. Cooke, M. Bailey, M. Mao, D. Watson, F. Jahanian, and D. McPherson. Toward Understanding Distributed Blackhole Placement. In *Proceedings of CCS Workshop on Rapid Malcode (WORM '04)*, October 2004.
3. German Honeynet Project. Tracking Botnets. `http://www.honeynet.org/papers/bots`, 2005.
4. R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet Background Radiation. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, 2004.
5. M. A. Rajab, F. Monrose, and A. Terzis. Fast and Evasive Attacks: Highlighting the Challenges Ahead. In *RAID*, 2006.
6. Y. Shinoda, K. Ikai, and M. Itoh. Vulnerabilities of Passive Internet Threat Monitors. In *Proceedings of USENIX Security Symposium*, 2005.
7. S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, 2002.
8. J. Ullrich. Dshield. http://www.dshield.org, 2005.
9. M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. Snoeren, G. Voelker, and S. Savage. Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm. In *Proceedings of ACM SOSP '05*, Brighton, UK, October 2005.
10. V. Yegneswaran, C. Alfeld, P. Barford, and J.-Y. Cai. Camouflaging Honeynets. In *Proceedings of IEEE Global Internet Symposium*, 2007.
11. V. Yegneswaran, P. Barford, and D. Plonka. On the Design and Use of Internet Sinks for Network Abuse Monitoring. In *Proc. RAID*, 2004.

6 Appendix

6.1 Extension to Multiple Monitoring Blocks

We briefly discuss the situation when multiple monitoring blocks are present in the address space. We assume for simplicity that the monitor blocks are pairwise disjoint and each has length m. Assume there are b monitor blocks, $b < k$, and $n = km$ is the total size of the circular address space identified with $\mathbf{Z}_n$ as before.

The Attacker can still start with a Round Robin. Randomly pick a cell $j_0 \in \mathbf{Z}_n$ to start. If the current query cell is j, then query it until receiving an answer $b = 1$, or have queried it ℓ times. Then replace j by $j + m$. We repeat this until $j = j_0$ again. At this point, we will have queried $k = n/m$ locations. At each such location j we record the final bit b_j. Notice that these bits are all correct answers. We will show that, in the presence of multiple monitoring blocks, this Round Robin strategy followed by a certain 'one-sided binary search' (we denote it by OSBS) can achieve essentially as good an upper bound as when there is only one block of length m, regardless of Defender's strategy. This justifies our consideration of *only* this Round Robin strategy for A. At the same time, a suitable Defender's strategy can achieve essentially the same quantity as a lower bound for the Defender, and therefore we have characterized the game.

Consider those bits $b_j = 1$. These indicate the presence of the monitoring blocks. We observe that no two bits can refer to the same block and each block must make its presence known through one of these bits. The bits $b_j = 1$ partition the k queried locations into contiguous runs of 1's separated by 0's in a circular array of k bits. We will concentrate on one such run of 1's. There is no interference between different runs of blocks indicated by the corresponding runs of 1's. Now consider one particular run of 1's. Without loss of generality suppose the run is $b_{rm} \dots b_m b_0 = 1 \dots 11$ and 0 is the right-most location where we have the bit $b_0 = 1$. This bit indicates the presence of a block, B^c, where $c \in \{0, 1, \dots, m-1\}$. OSBS will try to determine the location of c. After that, it can remove this (right-most) block B^c (in the run). If the run has more blocks left, we repeat OSBS on the new right-most block $B^{c'}$, with the slight modification that the right-most location for c' is $c - m$, *i.e.,* $c' \in \{-m, \dots, c - m\}$. As $c \leq m - 1$, the range is $c + 1 \leq m$. Therefore the bound we prove for OSBS on B^c applies to every block in the run. Now consider OSBS on B^c. If D does not commit any lies during this search, then one can perform an ordinary binary search, i.e., if $m \geq 2$, our first query is at $i = \lfloor m/2 \rfloor$. The answer may be 0 or 1, indicating $c < i$ or $c \geq i$, and we continue until finding c. It is well known that this takes $\lceil \log_2 m \rceil$ steps.

Now we have to consider the complication that D can lie a total of $< \ell$ times. However, it can lie only in one direction, namely, if the query i is such that $c < i$, the answer must be $b = 0$. Only when $c \geq i$, does the Defender D have the option of lying with $b = 0$, while the true answer is $b = 1$.

OSBS algorithm: In view of this, we carry out our OSBS as follows. We select our next new query point just as in ordinary binary search, as if there were no lies. Record the answer bits as $\beta_1 \beta_2 \dots$, where for each $\beta_i = 0$ or 1 we branch left or right, respectively. If any $\beta_i = 0$ at a cell x, however, we modify

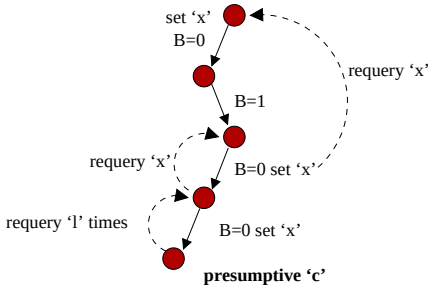

Fig. 1. Example OSBS traversal

ordinary binary search as follows. We will come back to do a 'repeated querying' at this location x, one query per each step, as we descend in the binary search to its left, until either:

- (a) we reach bottom (finding a presumptive c). If so, confirm $\beta_i = 0$ by querying ℓ times, else go to (b).
- (b) we discover $\beta_i = 0$ at x was a lie. If so, abandon work to the left of x, resume at x branching to the right.
- (c) have made ℓ queries at x, confirming $\beta_i = 0$ is true. If so, resume OSBS with no more queries at x.
- (d) got a new bit $\beta_j = 0$ at a cell y, further down in the binary search (*i.e.,* $j > i$). If so, replace y with x and proceed recursively.

Note that in case (b), it is possible that, due to recursion, there is a 'higher' x' in the binary search tree where the corresponding bit $\beta_{i'} = 0$, and the 'repeated querying' at x' was suspended because of x by case (d) for x'. At this point we resume the 'repeated querying' at x'. The search ends by finding c and the confirmation that the smallest cell to its right for which the bit $\beta_i = 0$ is not a lie (by having made a total of ℓ queries there). If all $\beta_i = 1$, then we must end in $c = m - 1$. This must be correct and there were no lies committed. In Figure 1, we illustrate OSBS traversal with an example.

We claim that we make at most $O(\ell + \log m)$ queries in total. It is clear that this bound holds if no lies are committed. The $O(\log m)$ pays for the 'binary search' and $O(\ell)$ pays for the final confirmation. Suppose at some step during OSBS a lie was committed at x. We make the following observation: in the descent in the binary search to the left of x, *all* subsequent answers with a $\beta = 0$ are lies. In this case, every step of the 'repeated querying' at x, as well as recursively all 'repeated querying' at locations to the left of x with a $\beta = 0$ in OSBS is charged to a total quota of $< \ell$ lies. Each of up to ℓ lies pays for $O(1)$ queries. This pays for the ordinary binary search queries at new locations descending from x as well. We also note that in case (c) above, a confirmation at x for a $\beta_i = 0$ also confirms all 'higher' x' in the binary search tree where the bit $\beta_{i'} = 0$.

In short, OSBS makes $O(\ell + \log m)$ queries in total. Corresponding to an ordinary binary search, it does the 'repeated querying', which costs at most two

queries per each ordinary binary search step. In addition, when OSBS discovers a lie at x, it abandons the portion of the work done after x. But that amount of work is proportional to the number of queries made at a location with a lie, and therefore costs $O(\ell)$. Every lie $\beta = 0$ is eventually discovered. The total amount of work consists of (1) at most double the work in ordinary binary search, and (2) the abandoned work (due to the discovery of lies) which is at most $O(\ell)$. Finally it costs $O(\ell)$ to confirm the answer. This proves the upper bound $O(\ell + \log m)$.

It is also clear that $O(\ell + \log m)$ is optimal up to a constant factor. The Defender can use $O(\ell)$ to delay, and information theoretically it takes $O(\log m)$ to find the $c \in \{0, \ldots, m-1\}$. Overall, for b blocks, as $b < k$, we get a simultaneous upper and lower bound $\Theta(k\ell + b(\ell + \log m)) = \Theta(k\ell + b \log m)$. The lower bound means that the Defender can achieve this on the average. If $k\ell \geq b \log m$, then clearly it requires this much with DD (even with only one block). If $k\ell < b \log m$, we can imagine that all b blocks are separated, then information theoretically we have the bound $\Omega(b \log m) = \Omega(k\ell + b \log m)$.

6.2 Model Discussion

In this Subsection we discuss various choices we made for the game theoretic model, along with a number of alternatives.

It is natural to model the landscape of network threats and security safeguards as an Attacker-Defender game. As is often the case, to model real-world systems as complex as operational computer networks, one has to focus on certain aspects of the problem and be willing to abstract away many others. This is particularly true for a theoretical investigation if we hope to prove something elegant and exact. Otherwise it is often hard to prove things in real-world settings. However, our goal is an abstraction that captures some key relevant aspects of the real problem, and provides some insight to the practice in the real-world.

We chose to first consider a single contiguous segment of the address space as a honeynet space. This is perfectly reasonable since many existing honeynets are configured in such a way. Frequently the honeynet size is a power of 2, which does divide the entire address space. In some instances one uses several contiguous segments to be hidden honeynet space. It would be typical to keep them the same size as well. To view the entire address space to be a circular array *is* a mathematical abstraction, which makes the theorems and proofs elegant. However, to consider the address space simply as an array would not change the essential conclusions of the theorems. One can always think of a new segment to be defined as starting at the boundary. Similarly, in addition to considering several contiguous segments of equal size, one can vary these sizes. However, our results will still be generally valid, with some modifications.

One can argue that the Attacker should not be assumed to know the block size m. This is reasonable; however, an Attacker can always try to "learn" the size information, by probing first at a large m, and then at $m/2$, and so on. This strategy incurs at most twice as much cost to the Attacker. (But the exactitude of Theorem 2 will be replaced by something weaker.) The following concern also

contributes to the choice of our model. In cryptography and security work, one generally tries to err on the side of assuming an Attacker knows more rather than less information about the system. It can be argued that the information about the size m is generally not that secure (it may even be published), and it typically does not change from one reshuffling epoch to the next. Also, an Attacker can always learn m (approximately) as in the above strategy of doubling the number of inquiry cells (the effect of replacing m by $m/2$), without too much extra work. At any rate the approach of assuming m is public knowledge gives a stronger security conclusion.

Another issue is modeling the cost of obfuscation by a total lie budget ℓ. Maintaining this obfuscation in real-world systems requires a certain amount of resources [**?**]. We provide some example shuffling policies in our Kaleidoscope implementation [10]. One extreme model is a perfect honeypot, where each packet is allocated a permanent space in system memory, because the ideal honeypot's response depends on all questions it has observed and how it has answered them. Shuffling is resource intensive because of the cost required to safely migrate legitimate network connections. However, the more the adversary can probe within an epoch, the more likely it can tell the difference between a real system and a honeypot. A Defender can only effectively obfuscate a certain amount; after which it is neither feasible nor cost effective. We model this cost of maintaining complete honeynet interaction history within an epoch in terms of the global lie budget ℓ. The duration within a reshuffling epoch should not be considered as indefinite; the trade-off between the cost of obfuscation and reshuffling determines this lie budget ℓ.

Another issue is why we assume the Attacker knows the lie budget ℓ. The argument is the same as for the Attacker knowing m: that this information is not that secure, that an Attacker can use a doubling strategy to approximate this ℓ, etc. If an Attacker uses a guessed value for ℓ, he can still implement RR (with suitable modifications, namely in the i-th iteration he can query Round-Robin with a guessed value 2^i where he guessed $\ell \leq 2^i$), this incurs again a cost only a constant factor more.

One could also posit that in the network defense game, there could be multiple (and independent) Attackers against a single honeynet (or defender). This situation in our model is simply treated as if the Attackers are colluding. This makes a stronger Attacker, and thus our security results more valid.

There are possibly other issues one can raise about the model. While other reasonable choices are possible and can possibly serve for future research, we believe that our model serves as a good starting point for consideration of the problem.