Optimizing Agent Placement for Flow Reconstruction of DDoS Attacks

Ömer DEMİR* and Bilal KHAN

Computer Science Program, Graduate Center, City University of New York,
New York, NY, USA
odemir@gc.cuny.edu,bkhan@jjay.cuny.edu

Abstract. Distributed denial of service (DDoS) attacks are a serious problem in the present-day Internet. We consider the design of a scalable agent-based system for collecting information about the structure and dynamics of DDoS attacks. Our system requires placement of agents on inter-autonomous system (AS) links in the Internet. The agents implement a self-organizing and totally decentralized mechanism capable of reconstructing topological information about the spatial and temporal structure of attacks. The system is effective at recovering DDoS attack structure, even at moderate levels of deployment.
In this paper, we demonstrate how careful placement of agents within the system can improve the system's effectiveness and provide better tradeoffs between system parameters and the quality of structural information the system generates. We introduced two agent placement algorithms for our agent-based DDoS system. The first attempts to maximize the percentage of attack flows detected, while the second tries to maximize the extent to which we are able to trace back detected flows to their sources. We show, somewhat surprisingly, these two objectives are concomitant. Placement of agents in a manner which optimizes in the first criterion tends also to optimize with respect to the second criterion, and vice versa. Both placement schemes show a marked improvement over a system in which agents are placed randomly, and thus provide a concrete design process by which to instrument a DDoS flow reconstruction system that is effective at recovering attack structure in large networks at moderate levels of deployment.

Key words: DDoS, flow reconstruction, network security

1 Motivation

Denial of service (DoS) occurs when legitimate users are prevented from getting service from shared resources or services [10]. When a DoS attack is mounted from large numbers of distributed sources, it is termed a Distributed Denial-of-service (DDoS) attack. Although DDoS attacks are just one kind of Internet-based attack, they have been identified as the most critical concern by the Internet Service Providers in Arbor Network's 2008 survey [1].

* The author thanks the Turkish National Police for funding this research.

There are many obstacles to resolving the DoS/DDoS problem that arise from the design of Internet architecture itself; here we give only a brief summary – a more detailed treatment is given in [6]. First, in order to gain the most of the Internet, its network link resources are shared among all users. Unfortunately, there is no intrinsic enforcement that the sharing is fair. Second, the network core must deal with very high volumes of traffic which must be processed fast, so core network components do as little as possible per packet, requiring all complex computations to be at the edge computers and thereby leaving a core that is unable to provide much security. Finally, the Internet is composed of interconnected networks each managed by different authorities, and their heterogeneous nature makes widespread deployment of defense mechanisms difficult.

2 Related Work

Given the aforementioned inherent obstacles, the notion of "Solving the DDoS problem" has many interpretations. Here we focus on the problem of determining the true origins and mechanics of attacks. Identifying the source of attack packets is difficult because the source address field in the IP header of packets is easy to forge or "spoof". The problem is further rendered intractable by the fact that current network standards do not require network devices to maintain information about the paths which packets take. We define *Flow Reconstruction* as "Actions taken to find the true sources and/or routes of packets to a given destination". There have been different approaches to this problem, including actively interacting with network traffic [5, 9], probabilistic and packet marking techniques [2, 11], and hash based logging [12]. We give a brief synopsis of prominent examples of each of these approaches:

Active Interaction is a strategy of interfering with attack traffic to deduce information about attack sources based on the systemic reaction to the interference. Backscatter is the prototypical example of this technique [9], operating at the level of BGP level routers. Backscatter finds the point of entry of the attack packets into BGP-level Internet backbone by having a backscatter server announce itself as the destination for spoofed IPs being used as sources addresses in the attack. Then, the backscatter server sends a BGP route announcement to make the destination network being attacked unreachable. Since attackers continue to send packets to the victim but the target is no longer reachable, the ingress routers reply with a "Destination unreachable" message to the source IP of the attack packets. This ICMP message gets delivered to the backscatter server, thus revealing the entry point of the attack packets into the backbone. While innovative, Backscatter has many drawbacks, most notably, a huge collateral effect in which legitimate traffic to the victim is blocked at the BGP level. Also, Backscatter requires some small modifications to the BGP protocol.

Packet Marking relies on routers adding identification information to the packets that they forward, so as to

reveal the path the packets have taken [2]. Marking every packet is not feasible because of packet processing overhead introduced by checksum recalculation.

Probabilistic packet marking (PPM) circumvents this by having routers select which packets are to be marked randomly as they transit. Router information is written into the IP packet's identification field (typically reserved for rebuilding fragmented packets). Whenever a router marks a packet, information written there by previous routers is overwritten. In order to find the full reverse path, there has to be at least one packet marked by the router closest to the attacker. The probability of the victim's seeing a packet marked by the furthest router is $p \cdot (1-p)^{(d-1)}$, where p is the probability of marking a packet and d is the hop distance of the marking router. The main limitation of packet marking is that path convergence is slow. Park and Lee [3] showed that PPM is effective at localizing the attack origin in single-source attacks but that as the number of mounted attack sources increases, the traceback is rendered more difficult. Also, Packet Marking requires overloading the semantics IP packet fields, and entails an increase in router packet processing load.

Hash-Based Traceback was introduced by Snoeren and Alex [12], whose source path identification engine (SPIE) consists of data generation agents (DGA), SPIE collection and reduction agents (SCAR), and SPIE traceback managers (STM). A DGA is the agent which calculates and stores the packet digest in digest tables. To do this, it relies on k fixed hash functions of the infrequently changing fields of the payload of each IP packet, with each function giving rise to an n bit number called a packet digest. The packets are stored in a space-efficient data structure called a Bloom filter [4] as follows: Initially (and periodically thereafter) a 2^n-sized bit-array is initialized to all zeroes. Whenever a packet arrives the array is marked at indices corresponding to the k distinct n-bit packet digests. When an IP Traceback request arrives, it specifies the time, and copy of the packet to be traced. The SCARs then ask the routers, starting from the last router, whether they they have any record of the given packet. These queries are answered by consultation with the router's local Bloom filter databases. If the indices in the array corresponding to the packet's digests are not all set, then the packet has not been seen previously. If all the indices are set, then it is likely (though not certain) that the packet has been seen previously. The main drawback of hash-based IP traceback is that in order to compute perform traceback, a copy of an attack packet must be presented to the system as soon as the attack starts–since otherwise Bloom filter tables might be discarded and the records lost. Also, Hash Based Traceback requires coordination and management.

Although there have been several approaches to flow reconstruction, as described above, each presents drawbacks in terms of requiring modification of existing Internet standards, increases in router processing load, or centralized coordination or management.

3 Objectives

We sought to develop scalable system for collecting information about the structure and dynamics of DDoS attacks. The system must (1) self-organize and require no centralized coordination, (2) effectively produce structural and tem-

poral data concerning DDoS attack flows, (3) be effective even when system deployment levels are modest, (4) not significantly increase router processing load, (5) not require modification of router internals, and (6) be built using existing Internet protocols. An example of the kind of question we seek to be able to answer using the system is: *What did the flow tree structure look like during the attack?* The agent-based system we devised is able answer such a question.

4 System Design

We begin by giving an informal description of the agent architecture and its operation. A more detailed description at the level of finite state machines can be found in earlier work by the authors [8]. Each agent may be be viewed as devices which reside on inter-AS links. Each agent can (i) passively listen to ingress/egress traffic on a switch port, and (ii) optionally inject control traffic into the stream. The agent operates by passively listening to traffic, by sampling packet headers and aggregating statistics based on destination IP address. In addition, the agent listens to all ICMP reply packets (header and payload) regardless of their destination. Whenever traffic volume to a destination IP triggers an alarm function (e.g. exceeds a system threshold) the agent creates a Alert to Downstrean (AD) message and sends it towards the victim. The purpose of this message is two-fold: (i) it informs downstream agents about a hypothesized attack on the victim, and (ii) it initiates the formation of a logical link in an agent overlay network specifically instantiated in response to the attack. The AD message is implemented as the payload of ICMP reply packet, whose destination address is the victim's IP address; it is sent by starting with a TTL of 1, and incrementing the TTL gradually until an acknowledgment is received from the next downstream agent in the direction of the victim. Whenever an agent sees an AD message in an ICMP reply packet, it replies with an acknowledgment, revealing itself to be the next downstream agent in the direction of the victim, and causing the upstream agent to terminate its TTL-increasing search process. The two agents can then share information concerning the attack on the victim over this newly formed logical link. If we view each logical link as a directed edge from upstream agents to downstream agents, the resulting logical network yields a distributed representation of a tree that is a maximal solution of the flow reconstruction problem corresponding to the scenario at hand. Information about the structure of this overlay network can be queried in real time by sending a broadcast message in the overlay network. Agents store their incident logical adjacencies in a distributed database that can be queried to determine the structure and dynamics of DDoS attacks.

Example. We illustrate a DDoS attack where v is the victim, clouds numbered 1, 2, 3, 4, 5, 6, 7, 8 represent ASs, A1, A2, A3, A4 represent attackers, circles represent routers, happy faces represent agents, and straightlines represent inter-AS link between autonomous systems. Figure 1 shows the reconstructed attack flow. The path to the attacker in AS-8 is fully reconstructed in full detail.

Although the attack traffic from AS-3 comes through non participating AS-2, the path to the attacker is successfully reconstructed. The attacker from AS-7 can be traced only up to AS-5. The system enables traceback to the extent possible with agent deployment. The reconstructed flows provide the victim with actionable information about both attack structure and dynamics.

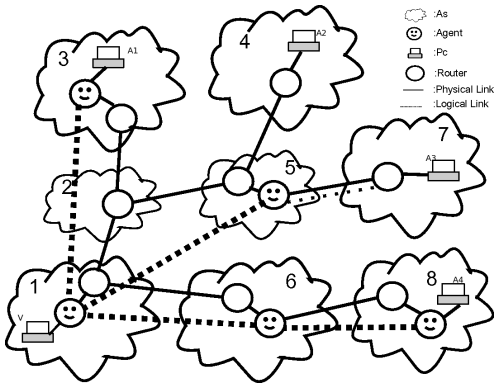


Fig. 1. Sample Flow Reconstruction

5 Mathematical Formulation

Before we can hope to quantify the performance of the proposed system, it is necessary to formally describe the problem that the agents are attempting to solve. Only then can we define the structure of a solution, and performance measures that quantify the solution quality with respect to the problem instance.

5.1 Formal Problem

An instance of the **flow reconstruction problem** is a tuple (G, R, A, D, v) where $G = (V, E)$ is a network on nodes V and $E \subset V \times V$ is a set of undirected edges between nodes. The function $R : V \times V \to V$ represents a routing table, where $R(u, d) = v$ is the next hop v on the path from u to d. The set $D \subseteq V$ is a set of attacking nodes which simultaneously attack the victim $v \in V$. The agents have been deployed on a set of links $A \subseteq E$.

To begin, we define **flowstep** $f(d, v, n)$ for every non-negative integer n, inductively, as follows:

- $f(d, v, 0) = d$,
- $f(d, v, n + 1) = R(f(d, v, n), v)$.

The sequence of flowsteps $F(d, v) = (f(d, v, i); i = 0, 1, ...)$ is called the **flow** from d towards v.

5.2 Formal Solution

A **valid solution** is a logical overlay network $L = (S, E_S)$ on a subset of agents $S \subset A$ where $E_S \subset S \times S$ and:

- $\forall e \in S \Rightarrow \exists d \in D \wedge \exists i \in \mathbb{N}(f(d, v, i) = e)$
- $\forall (e, x) \in E_S \Rightarrow \exists d \in D \wedge \exists i, k \in \mathbb{N}$ such that
  - $f(d, v, i) = e$,
  - $f(d, v, k) = x$,
  - $\forall j \in (i, k)\ f(d, v, j) \notin S$.

Stating the above conditions informally: (i) Every agent in the solution set S lies on the flow from some attacker to the victim; (ii) If two agents appear successively in the flow from some attacker to the victim, then these two agents are connected by a logical link in L_S. A solution $L = (S, E_S)$ is said to be **maximum valid** if every agent through which an attacker-originated flow transits, appears in S. Formally:

$$e \in A \wedge \exists d \in D \wedge i \in \mathbb{N} \wedge (f(d, v, i) = e) \ \Rightarrow e \in S.$$

The authors showed previously [7] that for any instance (G, R, A, D, v) of the flow reconstruction problem, there is a unique maximal valid solution. Accordingly, we define the **solution function** s which assigns to each instance (G, R, A, D, v) of the flow reconstruction problem this unique maximal valid solution. Hereafter, we denote the unique maximum valid solution of (G, R, A, D, v) by $s(G, R, A, D, v)$.

5.3 Performance Measures

A performance measure is a function that evaluates the quality of a solution (S, E_S) with respect to a problem instance (G, R, A, D, v). We need to establish some preliminary notations using which we can define our performance measures. We begin by defining a function $d_{G,R,v} : V \times V \rightarrow \mathbb{N} \cup \{\infty\}$. Intuitively, $d_{G,R,v}(x, y)$ equals the number of hops that a packet takes to reach y when it is sent by x to v in graph G according to routing table R. Note that $d_{G,R,v}$ is not generally symmetric or transitive, and hence does not define a metric on V. Now given any $Y \subset V$ and $x \in V$, we define the distance from x to Y

$$d_{G,R,v}(x, Y) = \min_{y \in Y} \{d_{G,R,v}(x, y)\}$$

The set of **undiscovered** attackers $U \subset D$ is defined as

$$\{u \in U \quad | \quad d_{G,R,v}(u, S) = \infty\},$$

and the discovered attackers are then simply taken as the complement $D \setminus U$. With all this notation in hand, we are now ready to define two performance measures by which to assess the quality of a solution with respect to a specific problem instance.

The **undiscovered attacker rate**

$$M1((G,R,A,D,v),(S,E_S)) = \frac{|U|}{|D|}.$$

When M1 is zero, every flow from every attacker is intercepted and hence detected by some agent. When M1 is one, every flow from every attacker is goes undetected by the agent system. Clearly, lower
values of M1 are preferred.

The **mean normalized distance to discovered attackers**

$$M3(G,R,A,D,v),(S,E_S) = \frac{1}{|D \diagdown U|} \sum_{d \in D \diagdown U} \frac{d_{G,R,v}(d,S)}{d_{G,R,v}(d,v)}$$

When M3 is close to zero, every attacking flow that has been intercepted by an agent has been intercepted close to the attacker. In this case, traceback succeeds in getting close to the attack sources. When M3 is close to one, every attacking flow that is intercepted by an agent has been intercepted close to the *victim*. In this case, traceback fails to reach the attack source. Clearly, lower values of M3 are preferred.

Example. In Figure 1, nodes $A1$, $A2$, $A3$, and $A4$ are attacking to victim V. In this case $A1$ is discovered by the agent in cloud 3, $A2$ is discovered by the agent in cloud 1, $A3$ is discovered by the agent in cloud 5, and $A4$ is discovered by the agent in cloud 8. This makes $|U| = 0$. Since $|D| = 4$, in this example $M1 = \frac{0}{4} = 0$, which means all the attackers are discovered.

Since $d_{G,R,v}(A1,S) = 0$, $d_{G,R,v}(A1,v) = 3$, $d_{G,R,v}(A2,S) = 4$, $d_{G,R,v}(A2,v) = 4$, $d_{G,R,v}(A3,S) = 1$, $d_{G,R,v}(A3,v) = 4$, $d_{G,R,v}(A4,S) = 0$ and $d_{G,R,v}(A4,v) = 3$, it follows that $M3$ for this example is $M3 = \frac{1}{3}(\frac{0}{3} + \frac{4}{4} + \frac{1}{4} + \frac{0}{3}) = \frac{5}{12}$.

5.4 Expected Performance Measures

Unfortunately, in practice, we do not know where the attackers $D \subset V$ lie in G, nor do we know which victim they will choose to target. DDoS attacks are frequently orchestrated by botnets, and thus involve arbitrary sets of attacking nodes located all over the Internet which collude to attack the chosen vicitm. Because we do not know the locations of the attackers or victims, the M1 and M3 performance measures defined in the previous section cannot be directly computed.

Starting from our definition of M1, we seek a derived performance measure that depends only on G, R, and A. This measure, denoted $E[M1]$, is the expected fraction of attackers which will be discovered when a random set of nodes collude to attack a random victim. Note that the **expected fraction of undiscovered attackers** $E[M1]$ on the triple (G,R,A), can be computed as:

$$\sum_{D\subseteq V,|D|=1,v\in V} \frac{M1((G,R,A,D,v),\ s(G,R,A,D,v))}{|V|^2}$$

Similarly, instead of M3, we use the expected mean normalized distance to attacking nodes. This measure quantifies the answer to the following question: if a random collection of attacking nodes were to attack a random victim, then on the flows which were intercepted by our agents, what is the expected value of the normalized distance from our agents to the attackers? This quantity, **expected normalized distance to discovered attackers**, denoted $E[M3]$, may be computed for a triple (G, R, A) as follows:

$$\sum_{D\subseteq V,|D|=1,v\in V} \frac{M3((G,R,A,D,v), s(G,R,A,D,v))}{|V|^2 \cdot (1 - E[M1])}$$

6 Agent Placement Algorithms

We note that the two performance measures $E[M1]$ and $E[M3]$ defined in the previous section were functions of only the network $G = (V, E)$, the routing table R, and the agent set $A \subset E$. Thus, for a fixed network and routing table, these two measures $E[M1]$ and $E[M3]$ can serve to differentiate between different agent sets.

More concretely, given two equinumerous agent sets $A_1, A_2 \subset E$ for which $|A_1| = |A_2| = n$, an assertion like

$$E[M1](G, R, A_2) > E[M1](G, R, A_1)$$

can be interpreted as expressing the fact that placing n agents according to the specification A_1 yields a lower fraction of undetected attack flows than placing the agents according to specification A_2. The placement A_1 is thus better.

Now suppose we have a third agent placement A_3 of n agents (i.e. $|A_3| = n$) for which $E[M1](G, R, A_1) = E[M1](G, R, A_3)$ but

$$E[M3](G, R, A_1) > E[M3](G, R, A_3.$$

This would imply that the two agent placement schemes A_1 and A_3 discover the same fraction of attack flows, but in placement A_3 the normalized distance from intercepting agents to attack flow sources is smaller. Placement A_3 is better than placement A_1.

In this paper we will describe two deterministic algorithms for placing agents in a network G (with routing table R). These algorithms are namely *M1-Greedy* and *M3-Greedy*. We will compare their performance relative to the expected performance of an adversary named *Random*, which places agents randomly on network edges. In the next sections we will describe each algorithm in detail.

Random. The Random agent placement algorithm serves as a baseline against which to compare our agent placement schemes. The Random placement algorithm takes as input the network $G = (V, E)$, the routing table R, and

the number of agents n which are to be placed. It operates by randomly selecting an edge $e \in E$ which does not already have any agent on it, and places an agent on that link e. This is repeated until the desired number of agents have been placed in G.

M1-Greedy Algorithm. The M1-Greedy algorithm sequentially places the specified number of agents, one by one, in a manner that greedily minimizes $E[M1]$ at each step. Suppose agents $1, ..., i-1$ have been placed already. The algorithm places agent i as follows:

- Create a map from E to natural numbers; initialize all entries to 0.
- For all possible attacker-victim pairs in $V \times V$, do the following:
  - Start from attacker, proceed hop by hop according to R. At each hop, if there is no agent on the edge increment the number associated with the edge by 1. If there is an agent on the edge, continue on to the next attacker-victim pair.
- Return the edge associated with the highest value as the placement for the next agent and add an agent on that link.

M3-Greedy Algorithm. The M3-Greedy algorithm sequentially places the specified number of agents, one by one, in a manner that greedily minimizes $E[M3]$ at each step. Suppose agents $1, ..., i-1$ have been placed already. The algorithm places agent i as follows:

- Create a map from E to real numbers; initialize all entries to 0.
- For all possible attacker-victim pairs in $V \times V$, do the following:
  - Start from the attacker, and proceed hop by hop according to R. Find the first agent f on the flow from the attacker towards the victim. If no agent is present, continue on to the next attacker-victim pair. Otherwise on each edge e that lies on the flow from the attacker to f, compute the reduction in M3 that would be obtained by placing agent i on e. Increment the number associated with e by the magnitude of the reduction obtained.
- Return the edge associated with the highest value as the placement for the next agent and add an agent on that link.

7 Results

Experiment 1 The purpose of the first experiment is to quantify how the two agent placement algorithms perform at comparable deployment levels, both relative to each other, and relative to the expected performance of the random placement scheme.

For this part of the research we carried out three experiments. Each experiment runs on the same Waxman [14] network of 200 nodes. The network is created by randomly placing nodes in a 2D plane and randomly selecting links between nodes with respect to their Waxman probabilities (where the likelihood

of having a link between two nodes is inversely proportional to their Euclidean distance) and build the routing table for the network using the Bellman-Ford algorithm. For succinctness, we will hereafter refer to this type of network as a Waxman network.

Within this Waxman network $G = (V, E)$, we placed $\delta \cdot |E|$ agents according to the M1-Greedy, M3-Greedy and Random placement schemes, where the agent density δ was varied from 0.0 to 0.50 fraction of the links. For each value of δ, measured both $E[M1]$ and $E[M3]$ of the system.

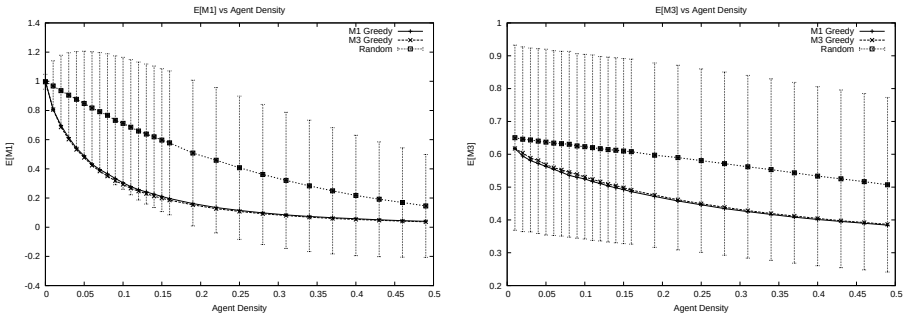


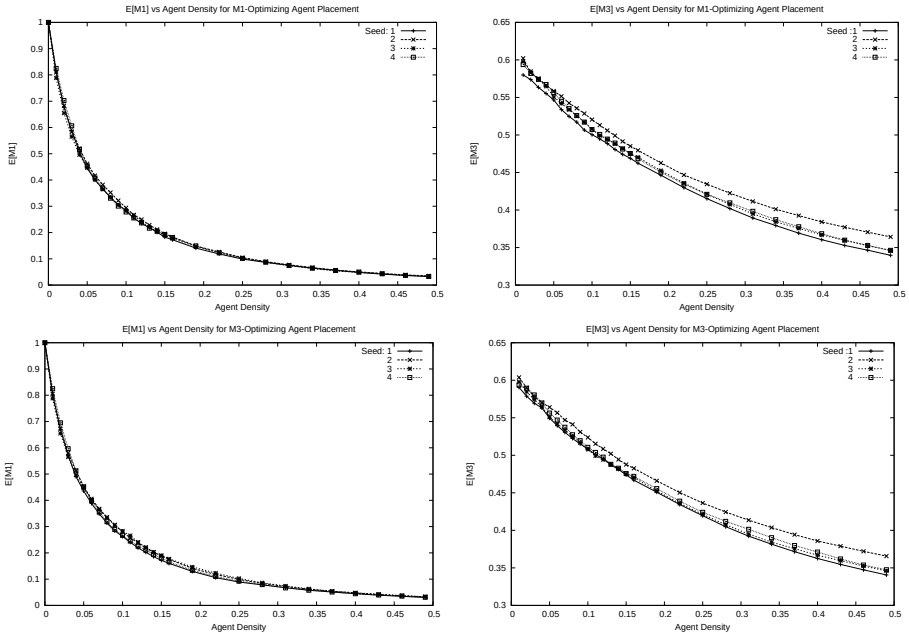
Fig. 2. M1 (left) and M3 (left) versus Agent Density

Figure 2 (left) shows E[M1] versus agent density curves for *Random*, *M1-Greedy*, and *M3-Greedy* algorithms. It shows that M1-Greedy and M3-Greedy algorithms have similar performance with respect to E[M1] under identical agent deployment levels. It also shows that for smaller agent densities, the resulting values for M1 and M3-Greedy algorithms are one full standard deviation below the performance of the Random agent placement algorithm. At 10% percent agent deployment, the expected value of E[M1] for a random placement is 0.71, while M1-Greedy and M3-Greedy achieve E[M1] values of approximately 0.30. This agent density is the point where the difference between random and greedy algorithms is maximal. As agent density increases, the performance of the three algorithms begin to coincide. This graph clearly shows that *M1* and *M2-Greedy* algorithms perform significantly better than *Random* placement of agents when we consider the E[M1] values.

Figure 2 (right) shows M3 versus agent density curves for *Random*, *M1-Greedy*, and *M3-Greedy* algorithms. Just as in Figure 2 (left) the *M1-Greedy* and *M2-Greedy* outperform *Random* on the E[M3] measure. Curves for M1-Greedy and M3-Greedy decreases faster than the curve for *Random*. At 19% deployment the E[M3] value of Random is 0.59 while it is 0.471 and 0.475 for M1-greedy and M3-Greedy respectively. At this deployment level, the greedy algorithms performs approximately 20% better than Random. This performance advantage is maintained as deployment levels increase.

The results of Experiment 1 show that the M1 and M3-Greedy algorithms always perform better than Random. More surprisingly perhaps, they show that optimizing greedily with respect to M1 is in concordance with optimizing with respect to M3. More precisely, a greedy placement of agents which sought to optimize E[M1] tends to be a placement which is quite good with respect to E[M3] as well, and vice versa. The two new algorithms we have developed yield very effective agent deployments, even at low agent densities: A 10% deployment according to either of the proposed greedy algorithms can detect 70% of the attack flows (since $E[M1] \approx 0.3$) and trace back halfway to the attacking nodes (since $E[M3] \approx 0.5$).

Experiment 2 The purpose of the second set of experiments is to quantify the extent to which our conclusions in experiment 1 might have been particular to the specific network in question. To judge this, we repeated the same experiment with different networks by using different random number seeds when generating the Waxman network. Below we show the curves for just four of the networks, which were generated using random seeds 1, 2, 3, and 4 respectively.

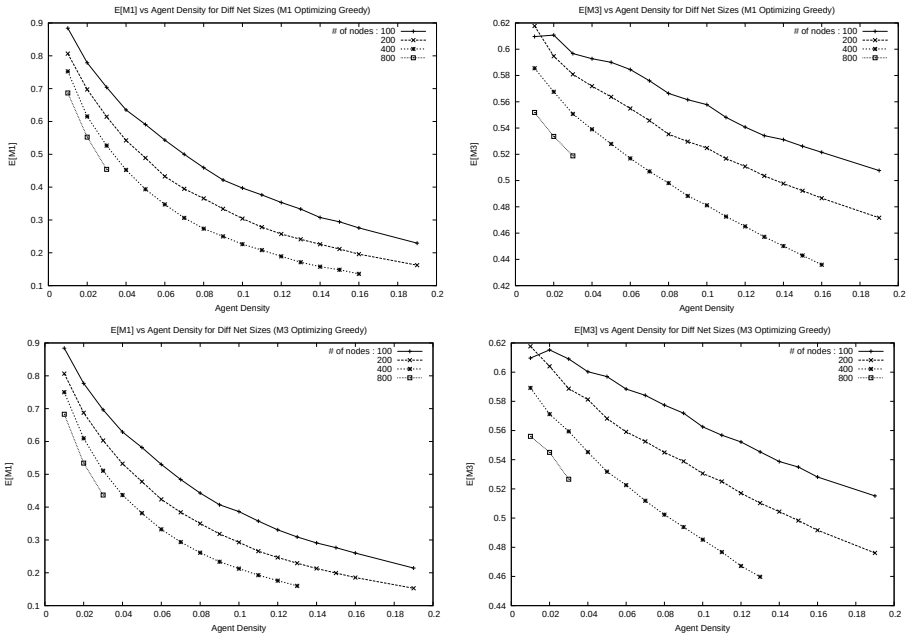


The top row's two graphs consider the performance of the M1-Greedy algorithm on different Waxman networks, while the bottom row's two graphs consider the performance of the M3-Greedy. The two graphs in the left column consider the E[M1] measure, while the two graphs in the right column consider the E[M3] measure. As can be seen, the E[M3] measure is more sensitive to the choice

of network (i.e. random seed) than the E[M1] measure. This is to be expected since E[M3] takes geometry and distance into consideration, where E[M1] is only concerned with geodesics (without reference to metrics). Also one can see that the graphs in the top row exhibit the same sensitivity to the random seed, as the corresponding graphs in the bottom row. This reflects the fact that the M1-Greedy and M3-greedy algorithms produce solutions that are comparable with respect to both the E[M1] and E[M3] measures (as was noted in the conclusion of Experiment 1).

The results of Experiment 2 indicate that conclusions drawn concerning the performance of the two schemes relative to each other are robust against the specific choice of network (of fixed size). Knowing this, we can now proceed to consider the impact of network size on the performance of the schemes.

Experiment 3 The purpose of the third set of experiments is to quantify the scalability of our conclusions with respect to ever larger networks. To determine this we carried out 8 experiments, which were largely identical except for the Waxman network size. Here, we report on the results of experiments on Waxman networks of sizes 100, 200, 400, and 800 nodes, from which the reader can ascertain nature of the relationship of the Greedy algorithm's performance advantage as network size increases.



The top row's two graphs consider the performance of the M1-Greedy algorithm on different Waxman networks, while the bottom row's two graphs consider

the performance of the M3-Greedy. The two graphs in the left column consider the E[M1] measure, while the two graphs in the right column consider the E[M3] measure.

The top-left graph shows how the E[M1] curve changes when the M1-Greedy algorithm is used to place agents in different networks of different sizes. There are four curves in the graph, with each curve representing the results of the experiment for networks of a different size (100, 200, 400, and 800 respectively). Each of the curves individually shows similar characteristics. However, as the network size increases, we note that the entire curve shifts downward. At an agent density of 0.03 the E[M1] values for 100, 200, 400, and 800 node networks are 0.704, 0.614, 0.526, 0.454 respectively. As the number of nodes in the network doubles, the E[M1] value decreases approximately 13%.

The top-right shows how the E[M3] curve changes when the M1-Greedy algorithm is used to place agents in different networks of different sizes. once Again four curves in the graph represent the results of the experiment for networks of sizes 100, 200, 400, and 800 respectively. Each of the curves individually shows similar characteristics. Once again, as the network size increases, we note that the entire curve shifts downward.

The bottom-left figure is very similar to the top-left figure. It shows how the E[M1] curve changes when M3-Greedy algorithm is used to place agents in different networks of different sizes. For the agent density of 0.03 the E[M1] values for 100, 200, 400, and 800 node networks are 0.696, 0.603, 0.510, 0.436 respectively. As the number of nodes in the network doubles, the E[M1] value decreases approximately 14%. The graph shows us that the M3-Greedy agent placement performs better as the net size gets bigger.

The bottom-right is similar to the top-right figure. It shows how does the E[M3] curve behaves when M3-Greedy algorithm is used to place agents in different networks of different sizes. Once again the graph shows that as network size increases, the curve shifts downwards.

Taken together, the results of Experiment 3 show that the performance of the proposed greedy algorithms *improves* for larger networks!

8 Conclusion and Future Work

We presented a scalable agent-based system for collecting information about the structure and dynamics of DDoS attacks. The agents in our system are placed on inter-autonomous system (AS) links and implement a self-organizing decentralized mechanism capable of reconstructing topological information about the spatial and temporal structure of attacks. We showed that our system is effective at recovering DDoS attack flow structure, even at moderate levels of agent deployment. We described two effective schemes for selecting the precise locations at which agents should be placed: M1-Greedy and M3-Greedy. We quantified the performance of these schemes and assessed their scalability. In experiment 1, we saw that the greedy algorithms always perform significantly better than random placement, and provide good flow reconstruction capabilities even at

modest agent deployment densities. We also showed that the two optimization criteria (M1 and M3) are concomitant: optimizing one tends to optimize the other. In experiment 2 we saw that these conclusions were consistent across different Waxman networks of the same size. Finally, in experiment 3 we saw that the effectiveness of the schemes actually *improves* as network size increases. Taken together, these results point to the viability of the proposed system as a solution to DDoS flow reconstruction.

Future work. Having established that the greedy algorithms proposed here are effective at determining the placement of agents for optimal DDoS attack flow reconstruction, we plan to use the proposed schemes to determine optimal placement of agents within the real Internet topology, under at various deployment level assumptions. For this purpose we intend to use the inter-AS connectivity database maintained by the CAIDA [13] project. Using the CAIDA topology we will assess the extent to which the proposed system can deliver effective DDoS flow reconstruction services to the Internet community. This will enable us to determine the necessary deployment level required in a real global system, and moreover, give us the precise inter-AS links on which agents should be placed in order to maximize attack flow interception and optimize traceback to attacking nodes.

References

1. Arbor Networks, `http://www.arbornetworks.com`
2. Bellovin, ICMP traceback messages,RFC draft, September ‘`http://tools.ietf.org/draft/draft-bellovin-itrace/draft-bellovin-itrace-00.txt` (2000)
3. Bellovin, Cert advisory ca-1996-26, Cert Advisory, ‘`http://www.cert.org/advisories/CA-1996-26.html` (1996)
4. Bloom, B. H.: Space time trade-offs in hash coding with allowable errors, Commun. ACM, vol. 13, no. 7, pp. 422–426, (1970)
5. Burch and Hal :Tracing anonymous packets to their approximate source, Proceedings of the 14th USENIX conference on System administration. Berkeley, CA, USA: USENIX Association, 319–328 (2000)
6. Demir O.: A Survey of Network Denial of Service Attacks and Countermeasures. City University of New York, Computer Science Department. (2009)
7. Demir, O., Khan, B. : An Agent-based Architecture for Flow Reconstruction of DDoS Attacks. Submitted to International Communications Conference (ICC) 2010, Cape Town, South Africa, 23-27 May 2010
8. Demir, O., Khan, B.: Quantifying Distributed System Stability through Simulation A Case Study of an Agent-based System for Flow Reconstruction of DDoS Attacks. In: Proceedings of the 1st Intelligent Systems, Modelling and Simulation Conference, Liverpool, England, 27-29 January 2010
9. Gemberling B., Morrow, C., and Greene, B.:ISP security-real world techniques. presentation, nanog. NANOG, `www.nanog.org` (2001)
10. Gligor V.D.: A Note on Denial-of-Service in Operating Systems. IEEE Trans. Softw. Eng. 10, 320–324 (1984)
11. Savage, S, Wetherall, D. Karlin, A. and Anderson, T.: Practical network support for IP traceback, SIGCOMM Comput. Commun. Rev., vol. 30, no. 4, pp. 295–306, (2000)

12. Snoeren, A. C.:Hash-based IP traceback, in SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications. New York, NY, USA: ACM, pp. 3–14, (2001)
13. The IPv4 Routed /24 AS Links Dataset - 11/15/2009 , Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, Matthew Luckie, kc claffy, and Colleen Shannon, `http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml`
14. Waxman, B. M.: Routing of Multipoint Connections.: Broadband Switching: Architectures, Protocols, Design, and Analysis. IEEE Computer Society Press, Los Alamitos, CA, USA (1991)