Optimizing Agent Placement for Flow Reconstruction of DDoS Attacks

Ömer Demir
Dept. of Information Tech.
Turkish National Police
Ankara, Turkiye
omerdemirkos@gmail.com

Bilal Khan
Dept. of Math & Comp. Science
John Jay College, CUNY
New York, USA
bkhan@jjay.cuny.edu

Ghassen Ben Brahim
Computer Science Dept.
Prince Mohamed Univ.
Al-Khobar, Saudi Arabia
gbrahim@pmu.edu.sa

Ala Al-Fuqaha
Computer Science Dept.
Western Michigan Univ.
Kalamazoo, USA
ala.al-fuqaha@wmich.edu

***Abstract*—The Internet today continues to be vulnerable to distributed denial of service (DDoS) attacks. 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.

***Keywords*—DDoS, network traffic, flow reconstruction.**

I. Introduction

A denial of service (DoS) is the act of preventing service or shared resources (or services) from reaching legitimate users [10]. When a DoS attack is mounted from large numbers of distributed sources, it is termed a Distributed Denial-of-service (DDoS) attack. Arbor Network's identifies DDoS as the most critical type of attack faced by Internet Service Providers [1].

The Internet architecture itself presents obstacles to the resolution of the DoS/DDoS problem. First, network link resources are shared among all users, but there is no explicit enforcement of fair sharing. Second, core network components need to be simple so they can quickly deal with very high volumes of traffic. This in turn means they must do as little as possible per packet, so the core is unable to provide much security, implying that it is typically a service enforced at the edges. Lastly, interconnected autonomous systems are each managed by different authorities, and their heterogeneity makes widespread deployment of defenses difficult. A more detailed treatment of architectural features of the Internet and their implications for DDoS is given in [6]

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. The source of attack packets is not easy to identify because the IP header's source address may be spoofed and network devices are not required to keep information about the path traveled by packets. 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], [12], and hash based logging [13]. Next, 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 destination network under attack is made unreachable by having the backscatter server originate a BGP route announcement message. 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. Despite its originality, Backscatter approach requires a modification to the BGP protocol and suffers from a collateral effect where good traffic destined to the victim is being dropped at the BGP level.

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

978-1-4673-2480-9/13/$31.00 ©2013 IEEE

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. If we can identify a packet that is marked by a router in a close neighborhood of the attacker, then the full reverse path can be reconstructed. 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. This technique suffers from both: packet processing load increase caused by overloading the semantics of the IP packet fields and slow path convergence. 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.

Hash-Based Traceback was introduced by Snoeren and Alex [13], 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 zeros. As packets are received, k distinct packet digests are computed and the corresponding bits in the array are set. When an IP Traceback request arrives along with the time and a copy of the packet that need to be traced, the SCAR asks the routers, starting from the last router, whether 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. Otherwise (all indices are set), then it is highly likely that the packet was seen earlier. The main drawback of hash-based IP traceback is that in order to 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. Therefore, some level of management and coordination between different routers and networks must exist.

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.

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 temporal 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 to answer such a question.

II. System Design

In this section we, informally describe the agent architecture and it main 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 viewed as devices which reside on inter-AS links. Each agent can (i) optionally inject control traffic into the stream, (ii) passively listen to ingress/egress traffic on a switch port and aggregate statistics based on the destination IP address of sampled packet headers set, and (iii) listen to all ICMP reply packets 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 Downstream (AD) message and sends it toward the victim. The AD message serves for 2 purposes. It notifies the downstream agents about a potential attack on a victim node and build an agent overlay network in response to the attack by adding one more Logical Link (LL) to it. The AD message is sent as part of the payload of the ICMP reply packets destined to the victim IP address. The TTL field in the AD message is initially set to 1 and gradually incremented until an ACK from the next downstream node (toward the victim) is received. The TTL update process is terminated by the upstream agent node following the receipt of an ACK message originated by a node identifying itself as the next downstream agent. Following this process, both agents are ready to start sharing information about the attack on the victim over the newly added LL. The set of all LLs, together represents the structure of a distributed representation of a tree which is a maximal solution to the flow reconstruction problem. Agents store their incident logical adjacencies in a distributed database that can be queried by sending broadcast messages over the overlay network to determine the structure and dynamics of DDoS attacks. A detailed exposition of the system can be found in [7], [8].

Example. Figure 1 shows the reconstructed attack flow following a DDoS attack. In this scenario, v is the victim, clouds numbered from 1 through 8 represent AS, A1 through A4 are the attackers, circles are the routers, inter-AS links are represented with solid lines. In this scenario, the attack path is properly reconstructed in AS-8 despite the fact that attack traffic from AS-3 came across non-participating AS-2. Attacks from AS-7 are traced only up to A-5. Traceback are enabled to the possible extent through agent deployment. The reconstructed flows provide the victim with actionable information about both attack structure and dynamics.

III. Mathematical Model

In this section, we formally define the problem that agents are trying to solve. This step is necessary to quantify the performance of the solution quality of the proposed system.

A. Formal Statement of Problem and Solution

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.

A routing table is represented by the function $R : V \times V \to V$, where $R(u,d) = v$ is the next hop v on the path from nodes u to d. set $D \subseteq V$ is a set of attacking nodes

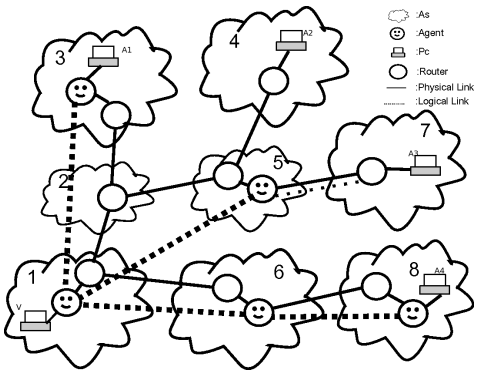

Fig. 1. Sample Flow Reconstruction

simultaneously attacking the victim node $v \in V$. 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, by taking $f(d,v,0) = d$, and $f(d,v,n+1) = R(f(d,v,n),v)$.

The **flow** from d toward v represents the sequence of flowsteps $F(d,v) = (f(d,v,i); i = 0,1,...)$. A **valid solution** is represented by a logical overlay network $L = (S,E_S)$ on a subset of agents $S \subset A$ where $E_S \subset S \times S$. Informally, (i) all agent nodes in the set S lies on the path from an attacker to the victim. (ii) there is a LL in L_S between 2 agents if both agents appear successfully in the path from an attacker to the victim. Formally:

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

Considering the above conditions

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. That is:

$$e \in A \wedge \exists d \in D \wedge i \in \mathbb{N} \wedge (f(d,v,i) = e) \ \Rightarrow e \in S. \quad (1)$$

In [7], the authors proved that there is a unique maximal valid solution for the flow reconstruction problem. 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 $s(G,R,A,D,v)$.

B. 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

$$d_{G,R,v}(x,Y) = \min_{y \in Y}\{d_{G,R,v}(x,y)\}. \quad (2)$$

The set of **undiscovered** attackers $U \subset D$ is defined as the set of vertices for which $d_{G,R,v}(u,S) = \infty$. The discovered attackers are then the complement set $D \setminus U$. 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**

$$M1((G,R,A,D,v),(S,E_S)) \stackrel{def}{=} \frac{|U|}{|D|}. \quad (3)$$

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**

$$M3((G,R,A,D,v),(S,E_S)) \stackrel{def}{=} \frac{1}{|D \setminus U|} \sum_{d \in D \setminus U} \frac{d_{G,R,v}(d,S)}{d_{G,R,v}(d,v)} \quad (4)$$

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. Considering the scenario in Figure 1, $A1$ is unfolded by the agent in 3, $A2$ is unfolded by the agent in 1, $A3$ is unfolded by the agent in 5, and $A4$ is unfolded by the agent in 8. This makes $|U| = 0$. Since $|D| = 4$, in this example $M1 = \frac{0}{4} = 0$, resulting in all attackers being 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 $M3 = \frac{1}{3}(\frac{0}{3} + \frac{4}{4} + \frac{1}{4} + \frac{0}{3}) = \frac{5}{12}$ is the normalized distance to the discovered attackers.

C. 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 victim. 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.

Considering our definition of undiscovered attack rate, we attempt deriving a performance metric $E[M1]$ that depends only on G, R, and A. This metric captures the expected fraction of attackers which will be discovered following the engagement of a set of nodes in attacking a victim node.

Note that the **expected fraction of undiscovered attackers** $E[M1]$ on the triple (G,R,A), can be computed as:

$$\sum_{D \subseteq V, |D|=1, v \in V} \frac{M1((G,R,A,D,v),\ s(G,R,A,D,v))}{|V|^2} \quad (5)$$

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:

$$\sum_{D\subseteq V,|D|=1,v\in V} \frac{M3((G,R,A,D,v),s(G,R,A,D,v))}{|V|^2\cdot(1-E[M1])} \tag{6}$$

IV. 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

$$E[M1](G,R,A_2) > E[M1](G,R,A_1) \tag{7}$$

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

$$E[M3](G,R,A_1) > E[M3](G,R,A_3). \tag{8}$$

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

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:

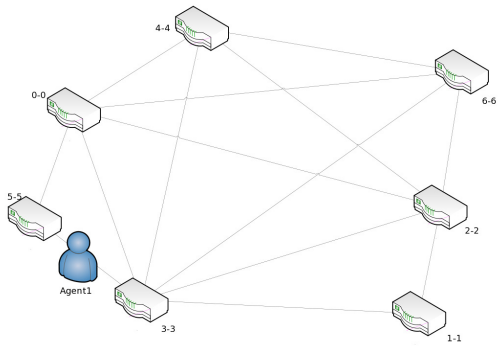

Fig. 2. Hypothetical initial agent placement.

- 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. For 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.

Figure 2 and 3 illustrates the behavior of two agent placement algorithms in a graph G with a routing table R. Figure 2 shows the location of the already existing agent. Figure 3 shows that the M1 and M3 places agents on different edges. M3 algorithm tries to minimize the expected difference to an attacker and M1 algorithm tries to maximize the expected number of attackers.

V. Results

A. Experiment 1

The purpose of the first experiment is to quantify how the two agent placement algorithms perform at comparable

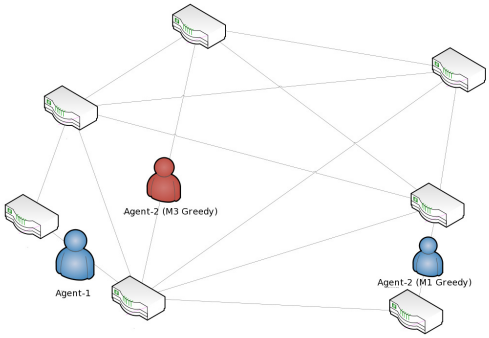

Fig. 3. Behavior of M1 and M3 Greedy placement of the 2nd agent.

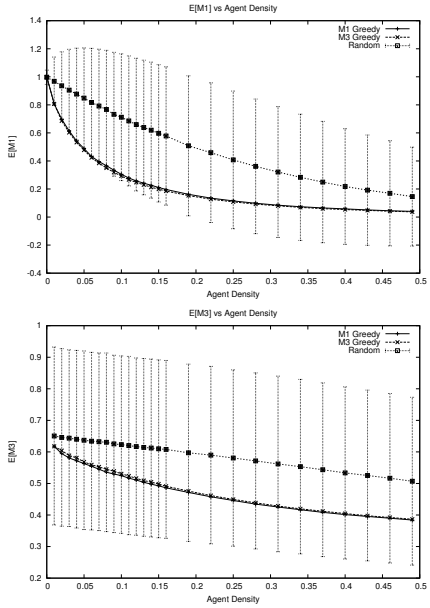

Fig. 4. M1 (top) and M3 (bottom) versus Agent Density

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.

Figure 4 (top) 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 *M3-Greedy* algorithms perform significantly better than *Random* placement of agents when we consider the E[M1] values.

Figure 4 (bottom) shows M3 versus agent density curves for *Random*, *M1-Greedy*, and *M3-Greedy* algorithms. Just as in Figure 4 (left) the *M1-Greedy* and *M3-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 $E[M3] \approx 0.5$).

B. 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 two graphs consider the performance of the M1-Greedy algorithm on different Waxman networks, while the bottom two graphs consider the performance of the M3-Greedy. The first and third graphs consider the E[M1] measure, while the second and fourth graphs 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

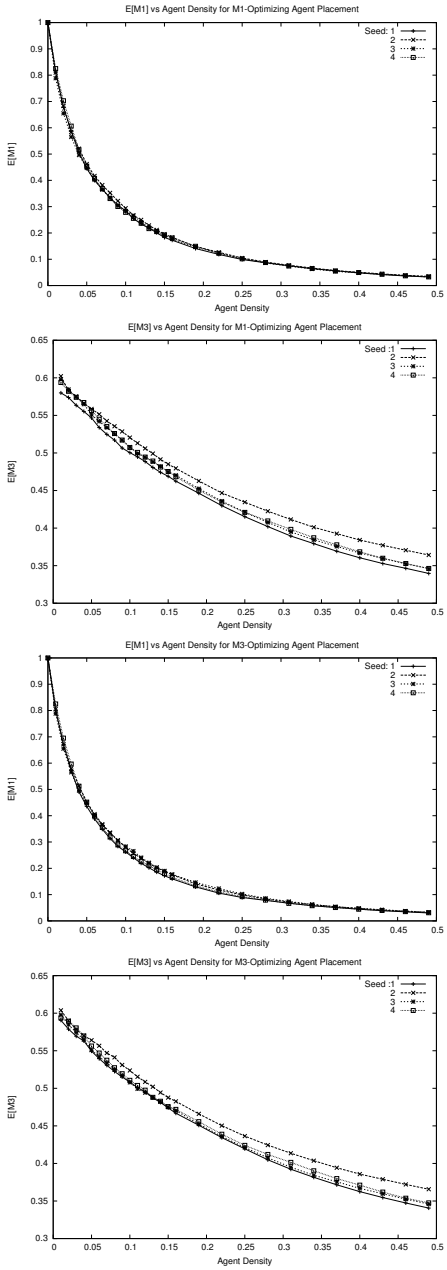

Fig. 5. Simulation results of Experiment 2: Robustness

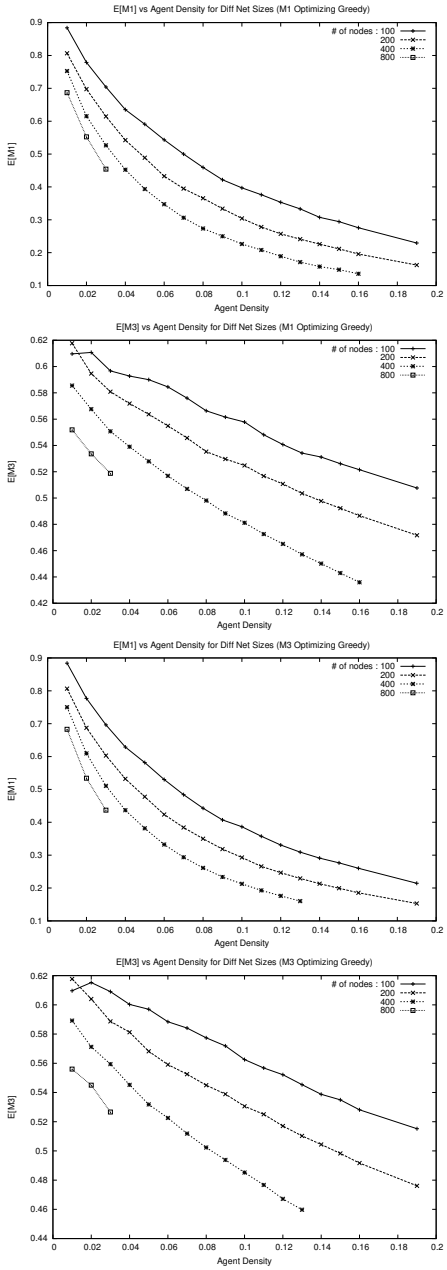

Fig. 6. Simulation results of Experiment 3: Scalability

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.

C. Experiment 3

The purpose of the third set of experiments is to quantify the scalability of our conclusions with respect to network size. 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 two graphs consider the performance of the M1-Greedy algorithm on different Waxman networks, while the bottom two graphs consider the performance of the M3-Greedy. The 1st and 3rd graphs consider the E[M1] measure, while the 2nd and 4th graphs consider the E[M3] measure.

The first 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 second graph 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 third graph is very similar to the first one: 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. Likewise, the fourth graph is similar to the second one: 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 proving the scalability of the proposed algorithm.

VI. Conclusion and Future Work

We consider the design of a scalable agent-based system for collecting information about the structure and dynamics of DDoS attacks. The agents implement a self-organizing and totally decentralized mechanism capable of reconstructing topological information about the temporal and spatial 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.

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 [11] 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-be llovin-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. Proceedings of Int. Communications Conference (ICC) 2010, Cape Town, South Africa, 23-27 (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, Modeling 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] Hyun, Y., Huffaker, B., Andersen, D., Aben, E., Luckie, M., Claffy K. C., and Shannon, C. The IPv4 Routed /24 AS Links Dataset - 11/15/2009 , http://www.caida.org/data/active/ipv 4_routed_topology_ aslinks_dataset.xml

[12] 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)

[13] 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)

[14] Waxman, B. M.: Routing of Multipoint Connections.: Broadband Switching: Architectures, Protocols, Design, and Analysis. IEEE Computer Society Press, Los Alamitos, CA, USA (1991)